Contributions to computer-aided analysis of cuneiform tablet fragments

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r I n g e n i e u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Denis Bernd Fisseler

Dortmund

2019

Tag der mündlichen Prüfung:	07. Februar 2019
Dekan:	Prof. Dr. -Ing. Gernot Fink
Gutachter:	Prof. Dr. Heinrich Müller
	Prof. Dr. Mario Botsch

Acknowledgments

At this point I would like to thank everyone who supported me during the writing of my dissertation and contributed to the success of the work. In particular, I would like to thank my supervisor Prof. Dr. Heinrich Müller, who gave me a lot of freedom and supported me with good advice, patience, and motivating deadlines for the completion of the work. Furthermore, I would like to thank Prof. Dr. Mario Botsch for taking over the second assessment of the work at short notice. Special thanks go to my project partners Prof. Dr. Gerfrid G.W. Müller and Dr. Michele Cammarosano from Hethitology, without whose interesting research topics and interdisciplinary cooperation this dissertation would not have been possible. In this context, I would also like to thank Dr. Frank Weichert, who was significantly involved in the application process of the projects linked to the research of this dissertation and made them possible by taking over the project management. Special thanks go to my colleagues Thomas Wiederkehr, Adrian Böckenkamp, Pascal Libuschewski, Daniel Bachmann and Thomas Harweg for the countless technical discussions and the productive working atmosphere.

My greatest thanks go to my parents. They always supported me during my studies and doctorate and were always there for me.

Abstract

This thesis presents methods for computer-aided three-dimensional analysis of digitized cuneiform tablets, an ancient type of writing documents. Since cuneiform script is predominantly conserved in the form of fractured clay tablet fragments, identifying matching fragments is a central task of manuscript reconstruction. This goal can benefit from the increasing 3D digitization of cuneiform fragments, which offers access to highly accurate cuneiform representations.

The main contribution of the thesis is a novel model-based method for the extraction of individual cuneiform wedges and associated wedge geometries from 3D scans, which can serve as a base for a statistical analysis of script features. This new automated approach enables access to large amounts of accurate quantitative cuneiform script features, which were not accessible by previously available 2D methods and can be employed for script similarity-based identification of candidates for fragment joining. A central aspect and challenging task is the robustness of the presented extraction method against scanning issues and mesh errors. This is achieved by employing a watershed-based wedge area extraction operating on a surface distance field with a subsequent constructive multi-stage model fitting. The extracted wedge models are refined by a wedge type classification followed by an effective wedge validation to handle false detections on fracture faces and damaged surfaces. An evaluation with respect to extraction rates, robustness, and performance shows the suitability of the developed methods that goes beyond an application purely for cuneiform fragment joining.

To address some compromises made during the wedge extraction regarding the representation of complex features, a fast supplementary approach for extracting skeletal surface features is presented. These features provide an alternative readable cuneiform representation and are created using a thinning approach on an approximated distance field. The quality of the resulting skeletons is optimized by employing a complex junction resolution, branch pruning and branch simplification methods, where both pruning and simplification can be used to adjust the resulting representation to different use cases. Aside from manual feature analysis, possible application scenarios also include providing a representation that can be handled by GraphCNNs for retrieval related tasks on cuneiform structures.

The cuneiform segmentation methods are complemented by a set of visualization concepts for a cuneiform segmentation framework. This includes a hierarchical concept for data handling and persistent storage of the generated segmentation data. Beyond, methods for fast rendering of large meshes, visualization methods to achieve good depth perception, detail enhancement, and semi-realistic surface shading are integrated. In order to not only address application scenarios like fragment joining and collation related tasks, the framework provides a sophisticated, highly interactive, and flexible segmentation data visualization that additionally offers fast geometry selection methods. A good accessibility of the generated data is guaranteed though an XML-based file format for storing segmentation data and through providing flexible data export methods. Although the framework is primarily intended for real-time segmentation, most segmentation methods can also be scheduled to process large numbers of fragments without user interaction.

All presented methods are evaluated with respect to performance aspects and their suitability for a set of philological use cases. The developed methods can be used flexibly in the scope of many aspects of the investigated application cases. This does not only apply to the automated feature extraction, but also to manual analysis aspects, which were discovered only by the new availability of the methods. The usability of the framework is underlined by the fact that it is actively being used by philologists from the Hethitologie-Portal Mainz, an established online resource in Hittitology.

Contents

1 Introduction **1**

1.1 Computer-aided analysis of cuneiform tablet fragments 1

1.2 Contributions of the thesis 4

1.2.1 Cuneiform wedge extraction and analysis 4

1.2.2 Fast wedge skeletonization 5

1.2.3 Visualization concepts for a cuneiform segmentation framework . . . 6

1.3 Organization of the thesis 6

1.4 Contributions by the author 7

1.5 Acknowledgment 8

2 Principles of philological research on cuneiform tablets **9**

2.1 Cuneiform script 9

2.2 Terms and definitions 12

2.2.1 Manuscripts 13

2.2.2 Wedges 14

2.2.3 Wedge model 16

2.3 Cuneiform script analysis 20

3 Digitization of cuneiform tablets **27**

3.1 Traditional reproduction 27

3.2 2D Digitization 30

3.2.1 Photographic documentation 30

3.2.2 Flatbed scanning 31

3.2.3 Reflectance transformation imaging 31

3.3 3D Digitization 32

3.3.1 Laser scanning 34

3.3.2 Photogrammetric scanning 36

3.3.3 Structured-light scanning 37

3.4 Mesh data structures 41

3.5 2-manifoldness 46

3.6 Mesh repair 47

3.7 Curvature estimation 50

3.7.1 Discrete differential curvatures . . . 50
3.7.2 Scale-space curvature estimators . . . 53
3.8 Conclusion . . . 56

4 Cuneiform wedge extraction **57**
4.1 Introduction . . . 57
4.2 Related work . . . 60
4.3 Outline of the method . . . 64
4.4 Mesh error detection . . . 65
4.5 Surface distance reconstruction . . . 70
4.5.1 Ambient occlusion computation . . . 70
4.5.2 Surface reconstruction . . . 80
4.5.3 Distance computation . . . 87
4.6 Region of interest extraction . . . 91
4.7 Wedge (area) extraction . . . 95
4.8 Wedge model extraction . . . 105
4.8.1 Wedge face extraction . . . 107
4.8.2 Wedge edge and feature point extraction . . . 123
4.8.3 Wedge classification . . . 133
4.9 Wedge validation . . . 141
4.9.1 Wedge segmentation faults . . . 143
4.9.2 Similarity analysis . . . 144
4.9.3 Corrective heuristics . . . 150
4.10 Extraction results . . . 151
4.10.1 Extraction rates . . . 152
4.10.2 Extraction robustness . . . 155
4.10.3 Extraction performance . . . 157
4.11 Component extraction quality . . . 161
4.11.1 Milling machine conversion and stamping process . . . 165
4.11.2 Clay drying process optimization . . . 168
4.11.3 Measurements . . . 169
4.12 Conclusion . . . 173

5 Cuneiform wedge skeletonization **175**
5.1 Introduction . . . 175
5.2 Application scenarios . . . 177
5.2.1 Wedge micro-structure analysis . . . 177
5.2.2 Wedge feature analysis . . . 178
5.3 Related work . . . 179
5.4 Skeleton construction . . . 182
5.4.1 Initial area construction . . . 183

5.4.2 Distance field approximation . . . 186
5.4.3 Thinning and vertex classification . . . 188
5.4.4 Edge tree construction and junction handling . . . 190
5.4.5 Skeleton partitioning . . . 193
5.4.6 Pruning and simplification . . . 195
5.5 Extraction results . . . 202
5.5.1 Skeletonization quality . . . 202
5.5.2 Extraction performance . . . 205
5.6 Conclusion . . . 211

6 Visualization concepts for cuneiform segmentation **213**
6.1 Introduction . . . 213
6.2 Framework concept . . . 214
6.3 Mesh rendering . . . 218
6.4 Visualization techniques . . . 222
6.4.1 Ambient occlusion . . . 222
6.4.2 Lit-sphere rendering . . . 223
6.4.3 Radiance scaling . . . 225
6.4.4 Logarithmic depth buffer . . . 229
6.4.5 Fragment joining . . . 234
6.4.6 Meta-data visualization . . . 235
6.5 Geometry selection . . . 242
6.5.1 Primitive selection . . . 243
6.5.2 Wedge and meta-data selection . . . 246
6.6 Performance evaluation . . . 248
6.6.1 Mesh rendering and geometry selection . . . 249
6.6.2 Memory consumption . . . 253
6.6.3 GPU data update . . . 254
6.7 Conclusion . . . 255

7 Discussion and future work **257**

References **263**

CHAPTER 1

Introduction

This thesis presents methods for computer-aided three-dimensional analysis of digitized cuneiform tablet fragments. The main contribution is a novel model-based method for the extraction of individual cuneiform wedges and wedge features which can serve as a base for a statistical analysis of script features, with the goal to identify join candidates for manuscript reconstruction. The wedge extraction is complemented by a fast method for extracting skeletal features as an alternative readable representation with an application dependent adjustable level of detail. To address both script analysis and fragment joining, a visualization concept optimized for handling large 3D-scanned cuneiform fragments and the generated segmentation data is presented. All aspects are integrated into the CuneiformAnalyser, a new segmentation framework for cuneiform data. This introduction gives a short overview of the targeted issues of cuneiform script analysis and associated scientific challenges in section 1.1 and the worked out contributions in section 1.2 which also includes an overview of the structure of the thesis.

1.1 Computer-aided analysis of cuneiform tablet fragments

Cuneiform script is one of the oldest known writing systems and was used over several millennia in the Ancient Near East. It consists of small wedge shaped imprints, which were usually imprinted in documents consisting of clay, and is therefore a three-dimensional script. Analyzing cuneiform manuscripts is a time-consuming task, since most documents are preserved in the form of poorly documented, unassembled clay fragments, with over 500.000 fragments discovered so far. Traditional cuneiform analysis is predominantly conducted by collating the original fragments or working with hand-drawn copies and photographic reproductions. This includes many analytical steps like transcribing, translating, and dating being pursued with the aim of reconstructing complete manuscripts from the fragments. However, particularly the large number of fragments and their restricted availability prevent a

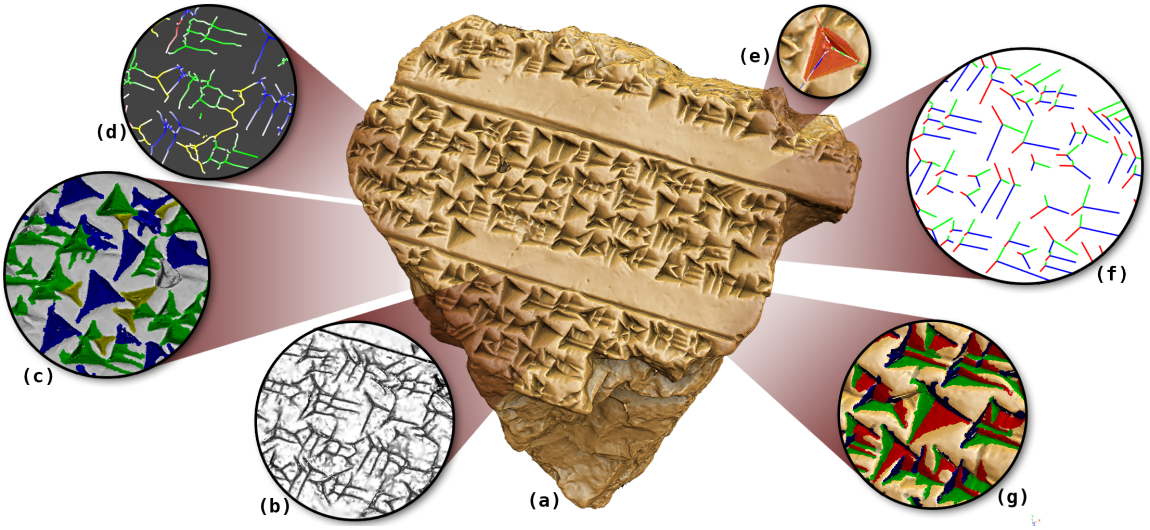


Figure 1.1: A selection of aspects of computer-aided cuneiform analysis. A high quality visualization of cuneiform fragment *Bo2743* is depicted in **(a)**, with magnified sections showing a stylized autography-style rendering **(b)**, extracted wedges colored by type **(c)**, extracted wedge skeletons **(d)**, a single **(e)** and multiple **(f)** fitted wedge models and extracted wedge faces with model specific colorization **(g)**.

rapid reconstruction. An additional challenge arises from the fact that cuneiform script on original clay fragments is usually a comparatively small script, where single wedges exhibit a size of only a few millimeters. This renders manual script feature measurements a difficult task with the risk of damaging the artifacts. Small-scale measurements are therefore usually carried out on photographic reproductions, which are not well suited for large scale high precision measurements and cannot represent the three-dimensionality of cuneiform script. Beyond that, many manuscripts are affected by a steady deterioration of their conservation status. For these reasons, the digitization of cuneiform fragments has been promoted in recent years, leading to several digital 3D cuneiform collections, which may, for example, be optimized for conservational or script analysis purposes. The existence of these data inventories does, however, not result in an equally increased availability of the data for associated research purposes or an automated exploration of the contained cuneiform writings. This is due, among other things, to the considerable effort involved in creating high-quality scans and the fact that automated cuneiform analysis is a challenging task.

These challenges are, as will be detailed in chapters 4 and 5, not only founded in data quality issues, like scanning errors or damaged writing surfaces but also in the complex geometric nature of cuneiform writing itself. In addition to that, the increased available precision of 3D datasets when compared to traditional representations does not necessarily lead to a speed-up of traditional cuneiform research methods. Instead, new formerly unavailable layers of data become accessible and their exploration requires additional time. One particularly important example are wedge-scale script features that are not only nearly impossible to measure accurately and precisely using manual measurement methods but are also nearly

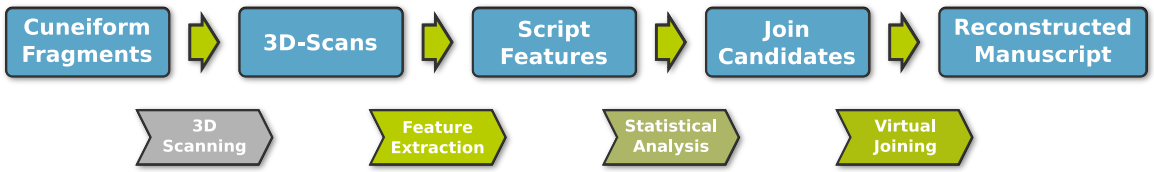


Figure 1.2: Pipeline concept for computer-aided cuneiform manuscript reconstruction. The data entities are shown in blue in the top row and the corresponding methodical steps in below the arrows in the bottom row. The saturation on the methodical steps reflects their coverage in this thesis.

impossible to handle manually due to the large number of wedges. At this point, computer-aided feature extraction and analysis methods are essential for making wedge-scale script features accessible and manageable, which is targeted by the main contributions of this thesis.

Aside from feature extraction and analysis, a not negligible aspect of computer-aided cuneiform analysis consists of providing methods for intuitive and flexible exploration of the analyzed and generated data, and ensuring compatible and persistent data storage. This aspect is covered by the visualization concept presented in chapter 6 which includes an appealing and high-quality visualization while at the same time providing the required performance to handle large amounts of data. To give a first visual impression of the achieved results, figure 1.1 shows a selection of aspects targeted by the main contributions of this thesis, including segmentation results of the wedge extraction and wedge skeletonization and high quality visualization.

All contributions of this thesis aim to address the overall concept of joining fragments by using the extracted wedge features for identifying related fragments on the basis of statistical script features. However, the application scenario is not limited to fragment joining. An exemplary high-level manuscript reconstruction pipeline is shown in figure 1.2. The most important methodical contribution is made with respect to the model-based extraction of wedge impressions and thus to the indispensable extraction of statistically evaluable cuneiform features. Closely related to this is a visualization concept, supporting not only fragment joining but also collation of 3D scans and working with generated segmentation data. Of the methodical steps shown, 3D scanning was targeted with respect to defining quality criteria for the scan suppliers and adapting the segmentation methods to be robust against inevitable scanning issues. Statistical script analysis was targeted partially by providing the script features, some basic statistical methods and a set of highly flexible data export methods. In contrast, the statistical script analysis with respect to philological criteria was conducted by the colleagues from Ancient Near Eastern Studies. It is worth mentioning that the linearity of the pipeline in figure 1.2 contrasts with the usually more iterative traditional methods of philological manuscript reconstruction. These traditional methods, which can be categorized into collation, script analysis, and manuscript reconstruction, have a strong tendency to cycle between sub-tasks like taking measurements, philological and paleographical analysis, creating hand copies and join layout sketches, and identifying and verifying joins. Since the exact

application and sequence of methods varies between philologists, a universal methodology is difficult to identify. The feature extraction methods presented in this thesis are made accessible via the developed visualization concept which is closely coupled with methods for the persistent storage of the data. As a result, the presented methods are now actively used for the analysis of hittitological questions by researchers of the Hethitologie-Portal Mainz [4] and thus make an important contribution to the establishment of digital script analysis methods in the Ancient Near Eastern Studies.

1.2 Contributions of the thesis

The subsequent subsections outline the three main contributions of the thesis, including short previews of the targeted issues and the developed methods. The contributions consist of a novel model-based extraction method for cuneiform wedges on three-dimensional cuneiform scans, an extension in form of a fast wedge skeletonization, and an associated multi-component visualization concept for 3D cuneiform segmentation framework.

1.2.1 Cuneiform wedge extraction and analysis

Analysis of cuneiform script features can be conducted on different non-overlapping levels and with varying analysis goals. This includes, but is not limited to, sub-wedge-scale features for analyzing the shape of the stylus, wedge-scale features which contain characteristics for the handwriting of specific scribes, and features on cuneiform-sign-scale which include the composition of wedges and are directly connected to the semantic meaning of cuneiform signs. To determine script characteristics for manuscript reconstruction, extracting wedge-level features in the form of wedge models is a promising approach, because wedge-models contain wedge-scale features and can also be hierarchically combined to represent larger features, like cuneiform signs and words. In addition to that, a cuneiform wedge model and its components are philologically intuitive features as opposed to feature representations like Bag-Of-Features [154] or neural network parameterizations [22]. As a result, they are highly compatible with manual analysis methods which are strongly rooted in philology and are also directly linked to the semantic content of cuneiform script.

Consequently, chapter 4 presents an original model-based method for the robust extraction of cuneiform wedges. The multi-stage approach draws its robustness from the strategy to incorporate as much geometric information as possible into a constructive model fitting process. The result includes not only the fitted wedge models but also a mapping of the underlying scanned geometry to the extracted wedges and a post-processing step for wedge validation. Specifically, the method first determines the geometric wedge areas using a watershed approach adapted to work on 3D-scanned cuneiform meshes. For this, the original free writing surface is approximated based on ambient occlusion [190] information and a Poisson reconstruction [106] to derive a distance field for the watershed-method to operate on. The performance of the distance field computation is optimized by exploiting the geometric relation of the scanned

geometry and the reconstructed surface. Starting from the extracted wedge areas as a basis, the wedge models are fitted constructively in several subsequent steps. This process can handle wedges with significant mesh damage and wedges with significant erosion damage on the original artifact. Since the model fitting consists of multiple steps, the approach can also provide partial extraction results of wedge components for a later correction. Applying the method uniformly to all potential wedge areas allows differentiating between damaged wedges and wedge-like structures on fracture faces in a similarity-based post-processing step. This step validates the extracted wedge models and automatically adapts to the script characteristics of each individual cuneiform fragment. A notable advantage of the automatically extracted models is the objectiveness of the results, as opposed to manual measurements or the subjective contents of hand-copies which therefore have to be re-collated frequently.

The extracted wedge models have various uses in cuneiform script analysis. This includes building a 3D cuneiform sign inventory and performing wedge shape and sign comparisons. The suitability for the intended use case of determining join candidates for manuscript reconstruction has been verified by Michele Cammarosano, who conducted a structural analysis of text groups [38] that showed good agreement with a set of previously known affiliations of the cuneiform texts.

1.2.2 Fast wedge skeletonization

The wedge models extracted in this thesis offer an abstract wedge representation that is well suited for statistical script analysis and classification. The abstraction is, however, connected with the trade-off that some aspects like partially existent or extensively damaged wedges and geometric details on the wedge edges cannot be captured. Nonetheless, these details may contain valuable information for scribe identification, cuneiform sign completion, or fragment joining. In addition to that, a non-model-based approach can capture features which do not necessarily need to be composed of predefined structures. Therefore, chapter 5 presents a fast supplementary method to extract wedge skeletons from the geometric wedge areas used during wedge extraction. Since cuneiform manuscripts contain script related features besides wedges and similar structures on damaged surfaces or fracture faces with varying detail, a suitable feature extraction method must be robust and provide mechanisms for determining the amount of detail to be captured. The presented skeletonization method accomplishes this by using a robust thinning approach on an approximated distance field combined with sophisticated branch junction resolution and is computationally inexpensive. Depending on the application scenario, the resulting graph representation can be optimized by changing the parameterization of the post-processing stages for pruning and simplification. The adjustable level of detail varies between an abstract representation similar to the previously extracted wedge models and a highly detailed representation whose resolution is limited by the underlying triangle mesh. One important property is the possible compactness of the skeletons that allows representing features at full available resolution while being significantly less memory consuming than the underlying triangle mesh. This and the readability of the skeleton representation allow a

variety of applications in the context of graph-based analysis and retrieval methods such as GraphCNNs [153] and SplineCNNs [67].

1.2.3 Visualization concepts for a cuneiform segmentation framework

A visualization concept suitable for real-time analysis of digitized cuneiform fragments has to consider several independent philological objectives, such as digital collation, fragment joining, and extraction and analysis of cuneiform script features. On a more technical level this includes performant handling of large meshes, high quality visualization for assessing the scanned datasets and the generated segmentation data, and a seamless and intuitive integration of meta-data management and measurement tools.

These aspects are addressed by the visualization concept presented in chapter 6 that uses a hierarchical structure of projects, fragments, meta-selections and selections to manage multiple cuneiform fragments simultaneously. For fragment joining, each project manages multiple fragments that can be positioned and oriented independently to create virtual joins. For feature management, each fragment may contain extracted wedges in the form of so-called "selections" and cuneiform signs and wedge constellations in the form of so-called "meta-selections".

To provide a flexible and high quality 3D visualization, the concept includes visual enhancements like lit-sphere-rendering [204], ambient occlusion [190], radiance-scaling [224], a high precision depth buffer [107], and a sophisticated meta-data visualization. With respect to the 3D-scanned geometry this ensures a realistic, high quality and real-time surface rendering while at the same time improving the perceptibility of subtle surface features, which is important for digital collation. With respect to the meta-data visualization, the concept aims to provide a fast, flexible, and intuitive solution for displaying and interacting with the extracted segmentation data. This includes fast multi-modal selection mechanisms and a flexible interface that can be adapted to the users requirements. To improve visualization performance without compromising on the geometry processing performance, the concept uses parallel data storage for geometry processing on the CPU and visualization on the GPU interfaced by fast synchronization mechanisms. The visualization methods are complemented by a set of flexible measurement mechanisms to support manual collation and a configurable visual interface for segmentation data export. This way, the concept covers all aspects from fragment joining to collation including in depth script feature extraction and analysis. It has been successfully used by the Hethitologie-Portal Mainz [4] for research and teaching.

1.3 Organization of the thesis

Following this introduction, chapter 2 presents some background knowledge regarding cuneiform script in general and provides necessary definitions of required terms with respect to cuneiform manuscripts and cuneiform script features. Furthermore, several use cases in traditional cuneiform script analysis for evaluation with the newly developed computer-aided methods are identified. The subsequent chapter 3 then introduces traditional and modern digital cuneiform

representations and discusses their respective strengths and weaknesses to emphasize the importance of digital representations. To improve the understanding of specific technical aspects of the cuneiform datasets used in this thesis and the associated geometry processing, chapter 3 also covers mesh data structures, topological mesh properties, mesh repair, and curvature-based feature computation on triangular meshes. The above mentioned main contributions wedge extraction, wedge skeletonization, and the visualization concept are presented in the subsequent chapters 4, 5 and 6. Aside from the presentation of the methods, each chapter includes related work and component evaluations where necessary. Chapter 7 concludes with an evaluation of the achieved benefits with respect to the philological use cases introduced in chapter 2.

1.4 Contributions by the author

This section details the author's contributions with respect to associated publications and projects that cover partial aspects of the methods addressed in this thesis. The presented novel methods for three-dimensional cuneiform wedge extraction, cuneiform wedge skeletonization, and the visualization concept for cuneiform segmentation including their evaluation are original work by the author of the thesis. In the course of the interdisciplinary research project *3D-Joins und Schriftmetrologie*, in which most of the research was carried out, parts of the covered topics have been published at peer-reviewed conferences and periodicals. A short version of the wedge extraction, including some visualization aspects has been published at the *Scientific Computing and Cultural Heritage 2013* conference [71]. The surrounding concepts for improving digital collation and analyzing cuneiform script features to identify join candidates among cuneiform fragments, has been published with a philological focus in the periodical *Welt des Orients* [40] and with a focus on informatics at the *EUROGRAPHICS Workshop on Graphics and Cultural Heritage 2014* [72]. Throughout all three publications, the development of the segmentation and visualization methods was carried out by the author, based on the philological requirements identified by the project colleagues Gerfrid G.W. Müller and Michele Cammarosano from the Ancient Near Eastern Studies. While the author was responsible for the methodological and performance evaluation, the evaluation of the results according to philological quality criteria and the statistical evaluation of the extracted script features was carried out by the project partners. The development work was characterized by a frequent exchange of ideas and feedback between the author and the colleagues from Ancient Near Eastern Studies in order to translate the philological objectives into formal problems that could be solved by means of computer science.

Corresponding research results with further philological evaluation of the data obtained using the methods developed in this thesis have been published separately by Michele Cammarosano [38, 39]. A partial presentation of the cuneiform visualization concept from chapter 6 was published by the author in the peer-reviewed *MDPI Informatics* journal [70], as part of the work on the research project *Web-basierte Analyse von 3D Keilschriftdaten*, which involved

the development of a web-based high-quality 3D cuneiform viewer for collaborative work, integrated into the Hethitologie-Portal Mainz [4].

The wedge model, introduced in chapter 2 was first presented in the periodical *Welt des Orients* [40] and developed in close cooperation with Michele Cammarosano and Gerfrid G.W. Müller, who provided the base for the underlying terminology by integrating established philological components for working with wedge edges and faces. The components were then extended by the author to include additional feature vertices and angles, an associated visual appearance to improve the structural consistency from an informatics point of view, and to meet the requirements for geometry segmentation and visualization. A further extension was introduced by Michele Cammarosano by adding several derived properties for cuneiform stylus analysis [39].

The 3D scans used in this thesis were predominantly taken and post-processed by Michele Cammarosano and Gerfrid G.W. Müller, as part of their work on the research project *3D-Joins und Schriftmetrologie* that also included the development of a suitable scanning work-flow.

Based on the segmentation and visualization results obtained with the methods presented in this thesis, the author contributed to multiple publications with the target of performing similarity-based retrieval tasks on 2D and 3D cuneiform representations. A 2D word spotting approach, based on BoF-HMMs (Bag-of-Features-Hidden-Markov-Models) was evaluated in cooperation with Rothacker et al. [180] and a 3D Graph-CNN in cooperation with Kriege et al. [117]. For both publications, the author generated and annotated the required ground-truth datasets.

1.5 Acknowledgment

The work on this thesis has been supported by the *Bundesministerium für Bildung und Forschung* (BMBF) within the interdisciplinary research project *3D-Joins und Schriftmetrologie* as a cooperation between *TU Dortmund University*, the *University of Würzburg* and the *Akademie der Wissenschaften und der Literatur, Mainz*. The Cuneiform-WebGLViewer, mentioned in chapter 6 was realized as part of the research project *Web-basierte Analyse von 3D Keilschriftdaten*, supervised by Frank Weichert with financial participation of the *TU Dortmund University*, the *University of Würzburg* and the *Gisela and Reinhold Häcker Foundation*.

Unless indicated otherwise, the copyright of all 3D scans of cuneiform artifacts used in this work is held by the *Hetitologie-Portal Mainz* [4].

CHAPTER 2

Principles of philological research on cuneiform tablets

Since the application specific parts of this thesis will assume basic working knowledge of cuneiform related research in the humanities, this chapter will focus on imparting relevant aspects of Ancient Near Eastern Studies regarding philology and paleography. To begin with, section 2.1 will give a short overview of cuneiform script, outlining its history, usage and properties as a writing system. Section 2.2 will then introduce a set of basic terms and definitions, including an abstract geometrical model for describing cuneiform wedges that will be used within this thesis. Subsequently, section 2.3 will describe basic approaches for analyzing cuneiform script by pointing out the respective research goals and current limitations of the methods. For a deeper insight into cuneiform studies, please refer to standard works on cuneiform writing [221], Assyriology [47] and Hittitology [171].

2.1 Cuneiform script

Cuneiform script is a writing system that was used in the geographical area labeled "Ancient Near East". With its time period of active use ranging from the 31st century BCE to the 1st century CE [47], cuneiform script is not only one of the oldest human writing systems but apart from Egyptian hieroglyphs also the longest-lasting one. Cuneiform script was, in most cases, written by imprinting a sharp-angled reed stylus into moist clay tablets or incised in stone, resulting in characteristic wedge-shaped impressions and is therefore a truly three-dimensional script.

The general appearance of cuneiform script has however changed significantly over the centuries. Starting in late fourth and early third millennium BCE with the proto-literate period, which was characterized by pictograph rich inscriptions configured in line separated vertical columns,

cuneiform symbols took increasingly more abstract shapes. Along with this tendency to shape simplification, the sign inventory was reduced substantially from approximately 1000 signs for Sumerian [66] to 600 signs for Akkadian [31] and further to 375 signs for Hittite cuneiform [183]. However, despite the reduced sign inventory, many signs still had a large number of variations, with 3000 sign variations listed for Hittite cuneiform in [183]. A major step in cuneiform script development was the transition from the initial vertical top-to-bottom writing direction to a horizontal left-to-right system and the introduction of a new wedge shaped stylus around 2500 BCE. In the resulting so-called archaic cuneiform, the symbols were no longer created by drawing the stylus over the writing surface, but by pushing the tip of the stylus into the clay, using different positions and directions. The thereby created wedge shaped impressions were used to approximate the previously used and now rotated pictograms and resulted in a simplification of the typeface. By 2000 BCE the archaic cuneiform, adopted by the Akkadians had been developed to such a high level of abstraction that all pictograms had been removed and it only consisted of a set of five basic shape types. The remaining semantically distinctive feature of these shape types was their direction, which could be horizontal, vertical, and diagonal in three different forms.

At this stage, cuneiform script had developed into a highly flexible but also highly complex writing system. The pictograms (image-based representation) and syllabograms (phonetic representation) used in archaic cuneiform were developed into syllabic and logographic signs and even exhibited a significant degree of polyvalence [44]. This required sophisticated methods to clarify the meaning of individual signs but on the other hand made the writing system very versatile. As a result, cuneiform script could be used to represent a variety of ancient languages, such as Akkadian, Elamite, Hittite, Hurrian, Luwian, Sumerian, and Old Persian [13]. Concurrent use of multiple cuneiform variants was not only common in official correspondence between countries but also as language for special purposes [95]. Sumer cuneiform, for example, was used as a language for science and cultic texts long after its use as the main language[1]. Eventually, cuneiform script was replaced by alphabet-based writing systems and became extinct around 75 CE [78].

Figure 2.1 shows examples of cuneiform in different languages and from various periods of time. It can be seen that many cuneiform variants use lines as structuring elements for text lines and columns. The visually most distinctive cuneiform type shown is old Sumerian cuneiform in figure 2.1a. Although it is visually very similar to archaic cuneiform, it is already composed of stylus imprints instead of being a stylus drawing. While the visual appearance of the cuneiform samples in figures 2.1b to 2.1f is very distinct from one another, many cuneiform types are hard to distinguish on a purely visual level. For example, Hittite cuneiform is very similar to the Akkadian cuneiform shown in figure 2.1c. As will be detailed later, a distinction of similar cuneiform variants can be made based on the textual content of the manuscripts.

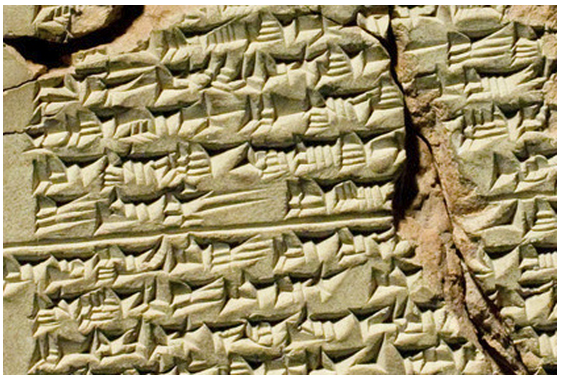
[1] http://www.uni-heidelberg.de/presse/ruca/2009-3/02texte.html



(a) Sumerian clay brick, 2500-2340 BCE
P222427



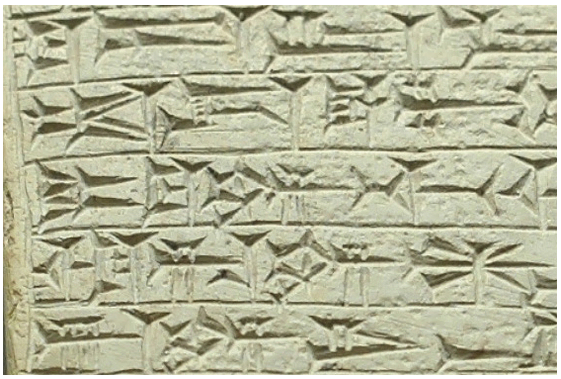
(b) Akkadian stone tablet, 1400-1000 BCE
P472670



(c) Akkadian clay prism, 1400-1000 BCE
P393923



(d) Old Assyrian clay tablet, 1950-1850 BC
P358454



(e) Babylonian stone tablet, 2000-1900 BCE
P386352



(f) Akkadian stone tablet, 911-612 BCE
P240211

Figure 2.1: Examples of sections with visually different types of cuneiform writing. The P-numbers in the second line denote the CDLI inventory numbers.
(Sources: CDLI [45] collection of the British Museum [213] and the Musée du Louvre [150])

The choice of clay tablets as the main writing medium had several implications regarding the usability and durability of cuneiform texts. In daily use, clay tablets could serve as a contemporary, rewritable medium by re-smoothening the tablet surface, or they could be dried for longer-term use [47]. However, even fully dried clay tablets are fragile and can be destroyed by moisture. As a countermeasure, the clay could be fired in a kiln. When protected from erosive forces, tablets treated in such a way can survive several thousand years. Although this method was not common for conserving cuneiform manuscripts, many artifacts have been conserved like this because they were affected by building fires. Important cuneiform inscriptions intended for non-portable and outdoor use were typically incised in rocks or big blocks of stone and exhibited a larger script size, like shown in figure 2.1. While these two factors could lead to the presumption of a higher durability, this was only true for cases, when the manuscripts were buried below the ground surface and therefore protected from erosive forces. Less commonly used writing media for cuneiform script include metal and wax-filled wooden writing boards, which offered fracture resistance or rewritability.

The currently conserved inventory of cuneiform manuscripts consists of more than 500,000 fragments of clay tablets, most of which are part of collections stored in museums. Among the institutions hosting the most extensive collections are the British Museum in London (currently 72,158 texts cataloged), the Penn Museum in Philadelphia, Pennsylvania (currently 26,610 texts cataloged), the Yale Babylonian Collection, New Haven, Connecticut (currently 26,670 texts cataloged), the National Museum of Iraq, Baghdad (currently 18,450 texts cataloged) and the Vorderasiatisches Museum, Berlin (currently 12,889 texts cataloged)[2]. Regarding their state of conservation, the large majority of stored artifacts consists of fired and unfired fractured clay tablets. Especially in the case of unfired tablets special conservational measures have to be employed to maintain their current state [167]. Otherwise, as can be seen in case of the collection of the National Museum of Iraq, the cuneiform tablets degenerate quickly into a state, where reading the inscriptions becomes impossible.

2.2 Terms and definitions

Analytical research on cuneiform script makes intense use of technical terms that are sometimes connected to a meaning unique to the field of Ancient Near Eastern Studies. This section will therefore clarify a subset of these terms that will be used frequently throughout this thesis. The subset will be limited to terms referring to geometrical properties, object names and structural features but will not cover profound semantic or translational aspects. Furthermore, this section will define a novel geometric model for describing cuneiform wedges that was developed in close cooperation with the Chair of Ancient Near Eastern Studies of Würzburg University.

[2] http://cdli.ox.ac.uk/wiki/doku.php?id=collections_of_more_than_10_000_tablets

2.2.1 Manuscripts

When talking about cuneiform texts, the most essential unit for content-based organization of texts is the *manuscript*. A manuscript usually denotes a compilation of text, whose content is intended to be used as a whole. In most cases, this is equivalent with the text being written on a single medium by a single scribe. However, it is possible for a manuscript to consist of multiple media and/or be written by multiple scribes. The most common portable medium for cuneiform manuscripts is the *clay tablet*, often abbreviated to *tablet*, which denotes a piece of clay formed to a compact shape with a smooth surface for writing. As they were usually formed by hand, clay tablets in most cases exhibit a cushion like shape without sharp corners, with their size ranging from a diameter of few centimeters to several decimeters. Although a tendency may be observed for smaller tablets to exhibit a pronounced rounded shape and larger tablets to be shaped like a square brick, there is no strict correlation between size and shape. The specific shape of a tablet may however be an indication of its intended type of use and its time period of origin. For example, large cylindrical or prism shaped manuscripts, like the Cyrus Cylinder [166], were mostly used for royal inscriptions. Hittite tablets, intended for storage in archives or libraries, were usually of larger size, containing multiple text columns and had a characteristic plano-convex shape. This shape, with a flat obverse and a convex reverse, was functionally induced to protect the script on the obverse while writing on the reverse.

The amount of text contained on a single tablet is limited by the size of the *writing surface* which is the smoothed portion of the tablet surface intended to contain cuneiform text. Since the total area of the writing surface is permanently defined upon creation of a tablet and there is a visual and technical lower limit to cuneiform script size, it was not uncommon for the writing surface to encompass the whole tablet surface. The limited space then enforced writing on and all around the borders of the clay tablets [171] and also local changes of the text's line direction to squeeze more text on the available writing surface. Larger Hittite tablets were usually divided into two or more columns, ordered from left to right on the front face and from right to left on the reverse. As tablets were rotated around a horizontal axis when changing sides, the leftmost column on the reverse was a direct continuation of the rightmost column on the front side [171].

Depending on its state of conservation, a clay tablet may consist of a complete or incomplete set of multiple *tablet fragments* or *fragments* for short, with mutually shared *fracture faces*. Initially matching fracture faces may exhibit a non-matching surface structure since a separation of fragments originating from the same tablet may have led to a different degree of time imposed damage. The real or theoretical process of reassembling two tablet fragments and the resulting specific combination of fragments to a new object, is called a *join*. However, a join does not necessarily involve a set of matching fracture faces but can instead be constructed by extrapolating the textual content, color texture or other material properties of the fragments. For that reason, a distinction is made between *direct joins* which denote joins with matching fracture faces and *indirect joins* with a significant gap between the fragments. Building

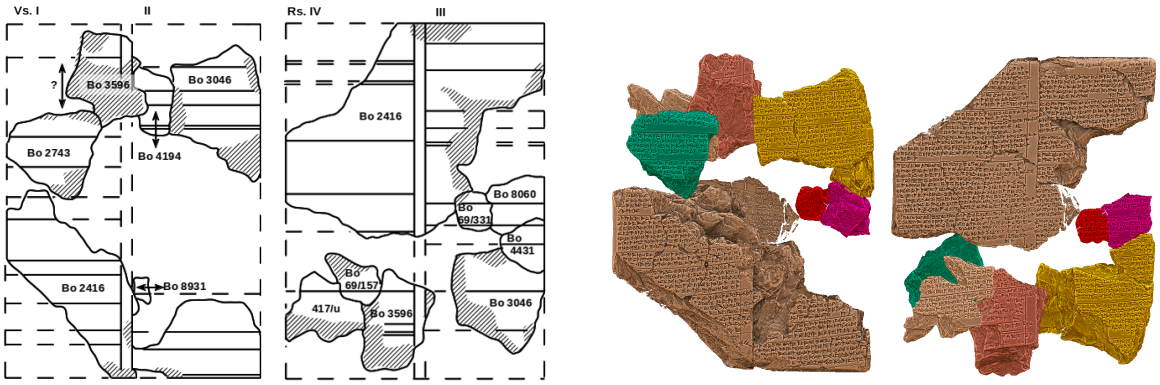


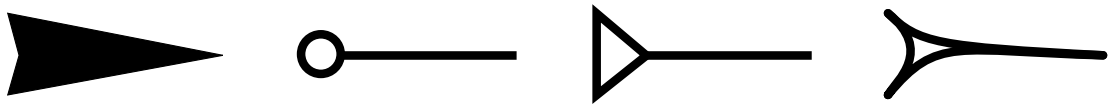
Figure 2.2: Traditional join layout sketches **(a)** and virtual join **(b)** for the front and back side of cuneiform tablet *417/u+*.

upon this terminology, a join of three-dimensional scans of multiple fragments will be called *virtual join*. Proposals for specific joins are usually documented in the form of *join layout sketches* that contain outlines of two or more involved fragments, including a basic structural representation of the textual content of the fragments. Although join layout sketches are intended to be a true to scale representation, they may contain severe distortions and missing fragment parts. This originates, in most cases, from their hand-drawn construction using photographs of fragments with non-matching camera perspectives or from being freehand drawings of real fragments. Figure 2.2 shows a comparison of join layout sketches and a virtual join for the front face and the reverse side of cuneiform tablet *417/u+*. Despite the existing distortion and some non-drawn fracture faces, the layout sketches enable an easy assignment of fragment positions and their mutual relationship. The labeling of layout sketches usually includes names of the involved fragments, designations of specific sides of a manuscript and possible positional uncertainties being visualized as two-sided arrows. As the color coding on the virtual join in figure 2.2 indicates units of fragments scanned as a whole, it can be deduced that these specific layout sketches represent an artifact state prior to the real world assembly of some of its parts.

The scientific names of tablets and tablet fragments are in most cases determined by catalog or archive numbers like “Bo532” or “VAT7679”, assigned by the museums and collections that host the artifacts. It is worth mentioning that fragments belonging to the same tablet do not necessarily share the same naming scheme and are in many cases stored at different locations or even different countries. This can render the physical verification of a join very complicated since it may depend on political decisions or complex claims of ownership.

2.2.2 Wedges

The most basic elements of cuneiform writing are the wedge shaped impressions called *cuneiform wedges* or *wedges* for short. Their shape closely resembles an inverted non-equilateral tetrahedron, usually exhibiting one direction of pronounced spatial extent. Since the width



(a) (b) (c) (d)

Figure 2.3: Examples for possible 2D representations for a horizontal wedge with the wedge tail oriented towards the right.

of a wedge varies along this direction, it can be assigned a thick *head* and a thin *tail* which together define the *position* and *direction* of a wedge. The head of a wedge is produced by the tip of the stylus, while the tail originates from a lateral stylus edge [38]. This reduction of a wedge to two basic elements leads to various possible 2D wedge representations suitable for transcribing cuneiform manuscripts, some of which are shown in figure 2.3. However, not all possible 2D representations contain the same amount of data and are suitable for the same types of cuneiform. Instead, the use of specific ways of drawing wedges depends on the type of cuneiform to be transcribed and on the intended purpose of the transcription. For example, the symmetric arrow-like representation in figure 2.3a has no separate elements for head and tail and is comparatively bulky. This limits its usage to uniform bulky wedges with a comparatively large spacing, which do frequently occur in some older cuneiform typefaces. The likewise symmetric representations in figures 2.3b and 2.3c are better suitable for densely packed wedges and have separate elements for head and tail. One disadvantage of the uncommon circular representation in figure 2.3b is its inability to capture all three corners of a wedge. However, it would be still sufficient as a readable representation. While all representations can account for a variable tail length, which was used for sign composition, only the representation in figure 2.3d can store additional information about the general shape of the wedge, such as asymmetric or curved features. Therefore, this representation is particularly useful for capturing the stylistic appearance of individual cuneiform manuscripts and is frequently used for transcriptions of Hittite artifacts. It is also popular, because it represents a compromise between border focused representations, like in figure 2.3a, and edge focused ones, like in figure 2.3c.

Depending on their direction, wedges of the more abstract cuneiform writing systems, like the Akkadian and Hittite, are categorized into five basic types [171]. These wedge types are, as shown in figure 2.4, *horizontal*, *vertical*, *downward diagonal*, *upward diagonal* and *Winkelhaken*. In this respect, the so-called *Winkelhaken* is basically a diagonal wedge produced by a vertical impression of the stylus tip and as a result, is the only wedge type without a tail.

A real life example of some of these wedge types can be seen in figure 2.5. It is worth mentioning that the usual tetrahedron shaped appearance of the wedges is altered significantly by wedge superposition effects. This can be seen when comparing the 3D scan in figure 2.5a with the wedge shapes in figure 2.5b, where the outlines of the wedges are drawn in their unaltered, tetrahedron shaped appearance. Aside from the basic wedge types, cuneiform

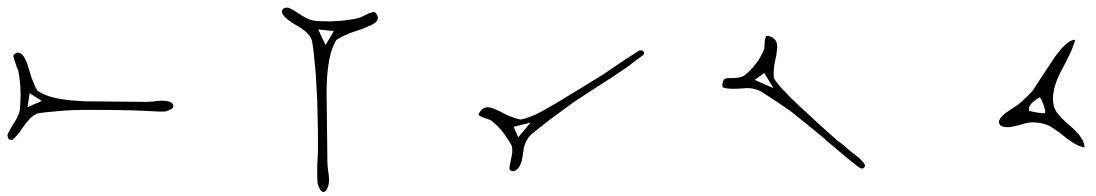


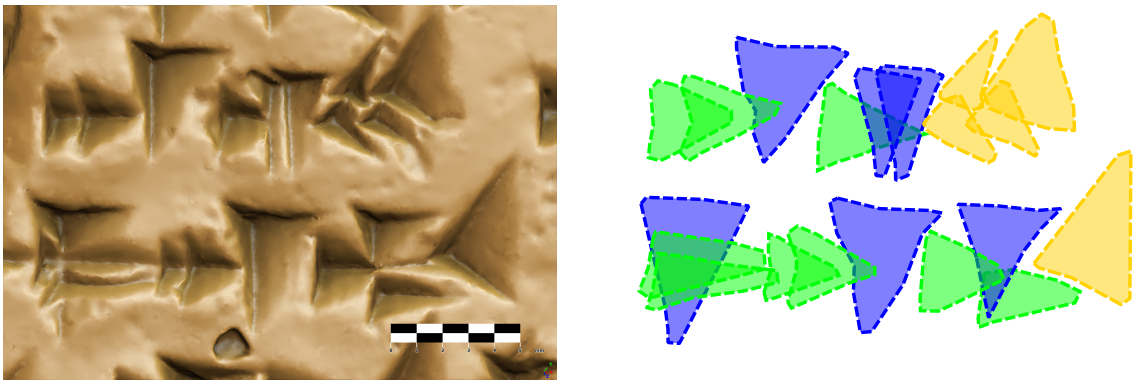
Figure 2.4: The five basic wedge types of more abstract cuneiform variants horizontal **(a)**, vertical **(b)**, upward diagonal **(c)**, downward diagonal **(d)**, and Winkelhaken **(e)**.

tablets may contain text structure elements like *column separators* and *paragraph separators*, which are created by dragging an edge of the stylus over the writing surface or by imprinting small rods or strings. For the left-to-right cuneiform systems used within this thesis, column separators are defined perpendicular and paragraph separators are defined parallel to the *writing direction* [171]. Another type of impressions are *cuneiform seals* that are produced by imprinting complex seal objects into the clay, either by pushing the object vertically into the clay or by performing a rolling motion over the writing surface. Seals may contain wedges, lines of various shapes, pictographs and, in most cases, exhibit an inverted visual appearance compared to conventional impressions. As a result, wedges produced by seals are usually elevated instead of being indented. Some examples for the above mentioned text structure elements on an exemplary cuneiform manuscript are visualized in figure 2.6. The manuscript includes several text paragraphs, delimited by paragraph and column separators and a round cuneiform seal in the center. The seal itself contains two circular lines of embossed cuneiform text and a flower pictogram in the center. One last type of impressions are *firing holes*. These approximately wedge sized, circular shaped holes of various depth were named after their occurrence on clay tablets that were fired to extend their durability. However, their exact purpose is still being controversially discussed [164].

The cuneiform wedge characteristics introduced so far, such as wedge type, position and direction, are predominantly semantic features with regard to the content of the manuscripts. Combining multiple wedges to local *wedge constellations* with a particular mutual relation of wedge type, position and direction can result in *cuneiform signs*. Whereas a wedge constellation is only a visually related group of wedges and does not need to be associated with a semantic meaning, a cuneiform sign represents a wedge constellation associated with a specific semantic meaning and assumes the function of a character. In many cases, determining this meaning not only requires considering the textual context of the sign but also taking into account the historical context of the respective manuscript.

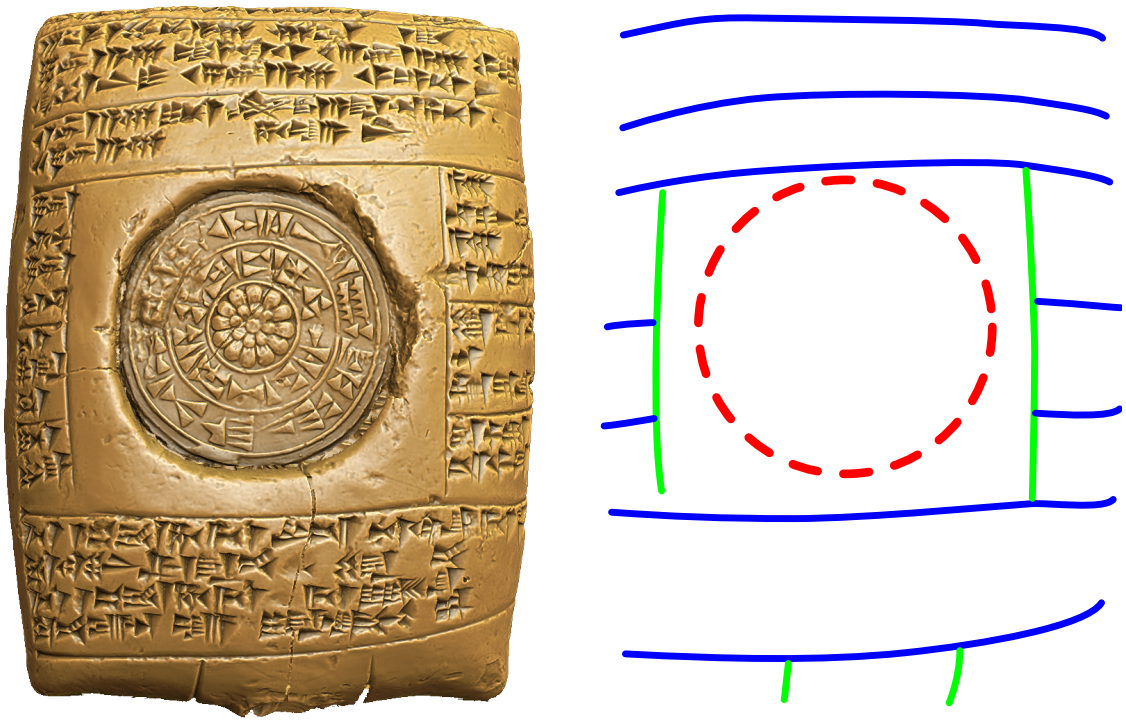
2.2.3 Wedge model

Although the semantic features described above are in many cases suitable for describing the content of a manuscript, they are only of limited use for describing cuneiform on a level suitable for an in depth metrological script analysis. Related research tasks, like recognition of



(a) (b)

Figure 2.5: Two exemplary line segments with different types of cuneiform wedges on cuneiform fragment *Bo532*. A 3D scan of the manuscript is shown on the left and an overlay of the participating wedges is shown on the right, with markings for horizontal (green), vertical (blue) and Winkelhaken-type (yellow) wedges. The black and white scale object on the 3D scan marks a millimeter scale.



(a) (b)

Figure 2.6: Text structure elements on cuneiform manuscript *Bo90/671*. A 3D scan of the manuscript is shown on the left and an overlay for the text structure elements is shown on the right, with markings for paragraph separators (blue), column separators (green) and a cuneiform seal (dashed red).

individual hand writing or reconstruction of the shape of cuneiform styli were, until recently, limited by an insufficient resolution of available measurement tools. Especially high resolution photography was affected by a nearly complete loss of the characteristic depth-component of cuneiform writing. As a result, for instance, the identification of scribes from their hand writing was conducted based on manual qualitative assessment of original artifacts. In contrast, the utilization of modern 3D scanning technology, as will be described in chapter 3, provides quantitative access to layers of information on wedge- and sub-wedge-scale. The resulting point- or triangle-based geometric representations may contain several thousand primitives for a single wedge. In view of the resulting geometric complexity of 3D-scanned cuneiform text (see chapter 4), the introduction of an abstraction layer for quantitative description of wedge-scale cuneiform features is a logical step. Therefore, the following will detail geometrical features of cuneiform wedges and their representation in a newly developed wedge model that was created in close cooperation with Michele Cammarosano and Gerfrid G.W. Müller from the Chair of Ancient Near Eastern Studies of Würzburg University. The model was first presented on SCCH 2013 [71] and extended later [40] to contain more definitions of derived script features. The main benefits of this wedge model are a unification of the wedge features which facilitates computer-aided feature comparisons, and an enhancement of scientific communication about cuneiform script features by providing a common terminology.

Starting from the tetrahedron shaped appearance of cuneiform wedges, the wedge model will use a non-equilateral tetrahedron as its basic shape. When viewed perpendicular to the writing surface, a so idealized cuneiform wedge consists of three linear edges in the center which form a star-like shape, and three linear edges at the borders which form a triangle, surrounding the wedge. This reflects the basic assumption of the wedge model that all contained edges are approximately linear, which is mostly true for the more recent types of cuneiform. Building on top of this, the primary components of the wedge model, like shown in figure 2.7, are structured into four interlinked layers, including vertices, faces, edges and angles. Since Hittitology was strongly influenced by German-speaking researchers, it was chosen as the primary language for the associated terminology. It was first presented in [40], with available translations in English, French and Italian[3], providing intuitive access for philologists by taking established terms into account. This thesis will use the English set of terms for consistency reasons. Nonetheless, all translations use the same set of element symbols taken from the German terms in order to avoid ambiguities.

The most important components of a wedge, as indicated by the frequent utilization in existing 2D wedge representations, are its edges, which provide easily accessible visual information about a wedges position, direction, type, and shape characteristics. The three edges e_s, e_r, and e_l, intersecting in the center of a wedge, will be called *inner edges* and contain information about the position and the direction of a wedge. Thus, the inner edge e_s specifying the direction of the wedge will be called *directional edge* or *spine*, whereas the remaining two

[3] http://www.cuneiform.de/uploads/media/terminology.pdf

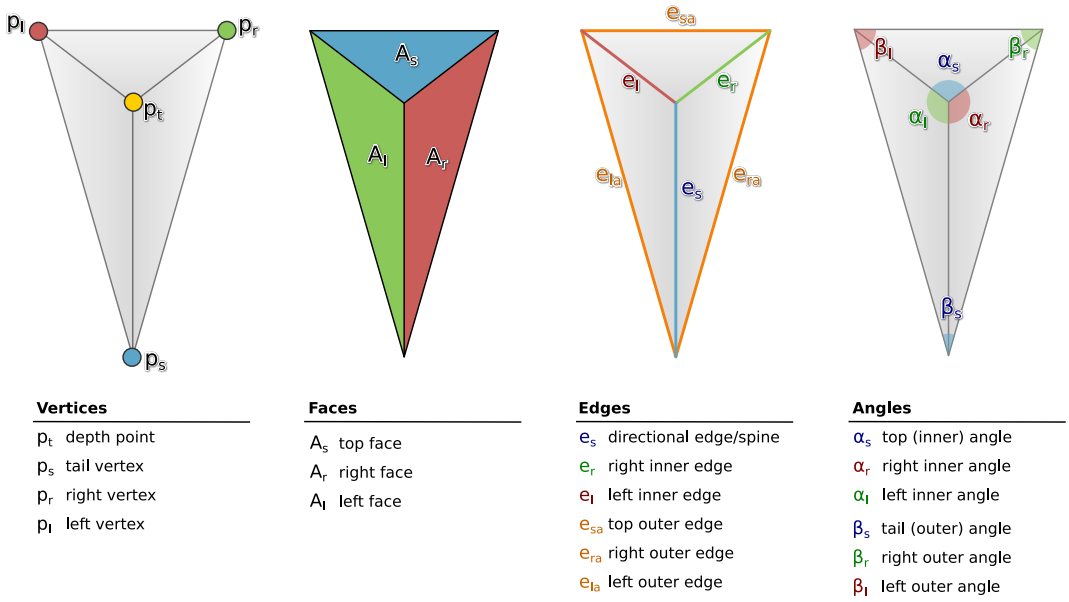


Figure 2.7: The terminology for the cuneiform wedge model, based on [40].

inner edges are named after their relation to the directional edge, *right inner edge* e_r and, *left inner edge* e_l. On the other hand, the three edges e_{sa}, e_{ra} and, e_{la} at the border will be called *outer edges*. The one opposing the directional edge will be called *top outer edge* e_{sa} and the remaining two are again labeled according to their relation to e_{sa}, *right outer edge* e_{ra} and *left outer edge* e_{la}. It is worth mentioning that the top outer edge is not always, as its name suggests, the top most edge of a wedge with respect to the writing direction but originates from the German term “Stirnkante”, referring to the previously defined head of the wedge.

Derived from the edges, the wedge model contains a set of four vertices p_s, p_t, p_r, p_l denoting, in this order, the *tail vertex*, *depth point*, *right vertex* and *left vertex* of the wedge. While the *depth point* p_t is the intersection point of the inner edges and thus usually exhibits the largest distance to the *tablet surface* O, the other three vertices are located at the intersection points of the inner edges with the tablet surface. In this context, the tablet surface O refers to the surface of the tablet before the impression of wedges, not the geometric surface generated by the impression of wedges. The *depth* of a wedge can therefore be determined by computing the distance between the depth point and the tablet surface. For single, non-damaged wedges, O can therefore be locally represented by the plane defined through p_s, p_r and p_l. In practice, the vertices are mainly used for expressing positional relations of wedge shape characteristics, multiple wedges and signs.

Similar to the nomenclature of the outer edges, the wedge model includes the three triangular *wedge faces* A_s, A_r, A_l, where A_s is the *top face* defined counterclockwise by the vertices p_t, p_r and p_l. The *right face* A_r is defined counterclockwise by the vertices p_t, p_s and p_r and the *left face* A_l is defined counterclockwise by the vertices p_t, p_l and p_s. Since the remaining face of the tetrahedron has no corresponding surface geometry in the wedge, it is not included in

the model. In contrast to vertices, the size of the wedge faces is primarily used to easily refer to specific relations of edge sizes and orientations. Wedge faces may also contain features that cannot be represented by this wedge model, like grooves from the utilized styluses.

The last primary component of the wedge model consists of two sets of angles that are used to facilitate describing specific complex wedge features. As shown in figure 2.7, these so-called *inner angles* and *outer angles* are defined between pairs of adjacent inner edges or outer edges and provide a size-invariant access to wedge shape characteristics. Their naming scheme is again aligned to the directional edge, resulting in *top (inner) angle* α_s, *right inner angle* α_r, *left inner angle* α_l, *tail (outer) angle* β_s, *right outer angle* β_r and *left outer angle* β_l.

Along with the definition of terms, the terminology includes a color scheme for the basic wedge components. Vertices, inner edges, faces and angles are colored with respect to their relation to the directional edge. Therefore, tail vertex, directional edge, top face, top angle and tail angle are colored blue, elements associated with the left inner edge are colored red, those associated with the right inner edge are colored green. The only exceptions are the outer edges, colored in orange, to emphasize their combined function as the wedge border and the depth point, colored in yellow, to emphasize its function as the wedge center, with no association to any side. The main benefit of introducing a color scheme is to provide an intuitive, easy to read visual representation, suitable for visualizing big amounts of wedge segmentation results. Therefore, the color scheme will be primarily used in chapters 4 and 6.

2.3 Cuneiform script analysis

Cuneiform script analysis is a central aspect of several branches of Ancient Near Eastern Studies like Assyriology and Hittitology, which specialize in different cuneiform using cultures and the respective languages that were already mentioned in section 2.1. In the case of cuneiform, for a comprehensive understanding of one language, the knowledge of related neighboring languages is highly beneficial. This dissertation will deal specifically with script analysis aspects of Hittitology, due to the technical orientation of the colleagues involved in the associated project *3D-Joins und Schriftmetrologie*. Nevertheless, large parts of the basic methodological framework can also be applied to other cuneiform disciplines.

In terms of scientific disciplines, cuneiform script analysis consists of components from philology and paleography. For cuneiform, philology comprises the general study of written historical cuneiform sources, with a focus on determining their meaning and interpreting the historic context. Paleography on the other hand acts as a historical auxiliary science which focuses not on the textual contents but on the writing itself, with respect to deciphering, and reconstructing the historical development of the script. This also includes the writing process, its development, the resulting forms of writing and the dating of ancient documents. With this in mind, it can be seen that the application areas of the contributions of this thesis have a strong connection to cuneiform paleography.

General sub-tasks of cuneiform script analysis are transliteration, transcription, and translation, which are used as components during the processing on many specific research questions. In this context, transliteration refers to reproducing the text sign-by-sign phonetically, separating the syllables by hyphens. Transcription on the other hand targets reproducing the pronunciation of the words [172]. Both steps are intended to facilitate reading the text through the use of Latin script. For a translation, a distinction between literal translation and analogous translation can be made, where the former targets a word-by-word translation and the latter aims to produce a well readable text.

In contrast to the above defined three philologically oriented sub-tasks, which focus on semantic aspects of cuneiform script, Hittitology also includes several working methods and research tasks that are more closely related to the core contributions of this thesis. The following will describe a selection of these methods and point out the respective research goals and current limitations of the methods. These methods include collation, the dating of manuscripts, searching for instances, scribe identification, manuscript reconstruction, stylus shape reconstruction, identification of incomplete signs and the creation of hand copies.

In philology, *collation* is the process of assessing an original manuscript for a comparison of texts with regard to completeness and accuracy. One particularly frequent process in Hittitology is the collation of a manuscript to check the correctness of a hand copy. Since the original manuscripts are not always available, this task is in many cases performed using photographic reproductions. However, unless the original artifact was affected by significant damage after producing the photographic documentation, the collation of the original artifacts is still regarded to be the most accurate option due to the usual inadequacies of the reproductions. Specific tasks during original fragment collation include viewing the artifact surface through a magnifying glass to compensate the small script size and taking measurements with calipers and rulers. This work often requires the employment of multiple light sources that are positioned to emphasize the tiny wedge structures by contrast enhancing self-shadowing. The fact that cuneiform wedge structures are highly unrecognizable with inappropriate lighting or a conflicting surface texture, is not only an issue for manual collation but also for the creation of photographic reproductions. In addition to that, manual measurement activities must be conducted with extreme precaution because they expose the artifacts to the risk of being damaged. For this reason, manual measurements are in many cases forbidden by the owners of the artifacts or may only be carried out by specially trained personnel. Together with the limited manual measurement precision, comparative script feature measurements may often only be taken on photographic reproductions by utilizing the included scale objects. In most cases, these scale objects consist of rulers or strips of alternating black and white boxes and can be used to reconstruct the original scale of features. One widely used technique for taking measurements includes enlarging the 2D reproductions using optical magnification or digitization methods and utilizing 2D measurement tools on the resulting representation. The latter is a first approach for computer-assisted collation.

The *dating of manuscripts* is a high priority task in Hittitology since Hittite manuscripts are predominantly undated. This is partially a result of immature archaeological methods during excavation, but it is also connected to the fact that the finding spots of Hittite cuneiform fragments do only poorly correlate with the fragment's place of origin due to the long period of usage. Dating a text can be conducted using paleographic, orthographic and linguistic criteria. This comprises dividing the texts into language levels (younger or older) according to their appearance for determining original texts and transcripts, a comparison of sign forms, determining usual and occasionally used signs but also the identification of scribes and their writing habits. In particular, the last three points are usually carried out by manual statistical evaluations that are very time-consuming because of the high number of required script samples. As for content-based manuscript dating, the text can be searched for occurrences of the names of well-known historical personalities or for already dated historical events. For this, philologists traditionally use manually created annotation archives based on indexed cards that cannot be searched automatically with computer-aided methods.

Searching for instances of cuneiform wedges, signs and words is an essential component of content-based work on cuneiform manuscripts and is also important for analyzing low-level script features, like wedge geometries. Aside from the function of proving individual scientific results, instances are also intensively used for extending the established and widely used working media of Hittitology, like dictionaries with references [76], sign inventories [183] and to a lesser extend grammar books [75]. Especially sign inventories are useful for classifying specific signs without first having to locate additional references within other cuneiform manuscripts for comparison and verification. Searching for graphical instances on real artifacts, photographic reproductions and hand copies is time-consuming since computer-aided methods for accessing the available data are not well-developed. This is true, even when using partially digitized sign inventories and dictionaries. However, the work can be accelerated by searching digital transliteration and transcription data, which is based on Latin script. A standardized digital 3D representation of script features, wedge constellations or words, including annotations could greatly enhance the established retrieval methods.

The *identification of scribes* can be as simple as finding a signature but may also include identifying scribe specific habits regarding the usage of words and signs or analyzing the specific shape of signs over multiple manuscripts and extracting clues from the textual content. It is not always easy to distinguish between writer-specific characteristics and characteristics of writing schools since individual characteristics of cuneiform writing are also passed on during the writing learning process. For this reason, a reliable identification of scribe characteristics requires particularly skilled experts.

The *reconstruction of a cuneiform manuscript* can be as simple as recognizing a set of matching fracture faces by chance but is in most cases connected to analyzing the pieces of available information about multiple fragments to match, amongst other features, the origin, the used language, the date of origin, the textual content and the geometric properties of the fragments.

Since many of these features are difficult to retrieve, manuscript reconstruction is a time-consuming task. Once a set of join candidates is identified, the involved fragments are checked for features that support a direct join. Without simultaneous access to all involved parts, the fragments are collated for matching fracture faces, combinations of matching column or line separators, matching line patterns, and writing surface curvatures. Through extrapolated text separators and line patterns, it is also possible to identify indirect joins with no direct contact faces. Another possibility to match fragments is based on the textual content and can be used in cases where a fragment contains a portion of an already known text. This is not an uncommon case, because many historical documents like treaties, legislative texts or instructions for religious rituals were duplicated and distributed in multiple copies. Verified joins are then recorded in form of the join layout sketches already mentioned in section 2.2.1. In the case that two matching join candidates can be brought together in one place and the respective owners agree to performing a real join, the fragments can be glued together using reversible glue. Since this process results in only one remaining artifact, joining is a delicate process with respect to ownership and sometimes political issues. An alternative to real joins is the verification of joins based on photographic reproductions. However, the individual photographies have first to be matched in size, either by using optical or digital aids, and usually do not contain a detailed depiction of the fracture faces aside from the fragment borders visible from above. This complicates an exact verification. Virtual joins of 3D-scanned cuneiform fragments benefit from the largely increased availability of the digital datasets regarding simultaneous access to the join candidates and the possibility of conducting a join without risking inflicting damage to the original artifacts. In contrast to matching 2D reproductions, the fracture faces can be fully incorporated into the matching process. In addition to that, indirect joins can be evaluated more flexibly and the result can immediately be used for script analysis and shared for collaborative join verification.

Stylus shape reconstruction is a highly specific research topic that serves the purpose of understanding the writing process and, in some cases, to identify writing cultures. One simple example for this are wedges that show reed markings on their faces, which is a characteristic of Mesopotamian cuneiform that was written using styluses made of reed [39]. However, a good reconstruction of a stylus shape requires highly precise measurement tools. This precision can usually not be achieved using manual measurements. As a result, the stylus shape is often researched by constructing potential styli and comparing the achievable writing results with the ancient originals. Researching stylus shapes is complicated by the fact that the shape of a cuneiform wedge does not necessarily represent the inverse shape of a stylus, since the stylus may have been rotated or shifted during impression [39].

Cuneiform fragments often contain signs with partial damage or signs in extensively damaged areas. An *identification of incomplete signs* is usually done by comparing the available part of a sign against possible completions from sign inventories or other locations on the same manuscript. A more metrologic approach builds upon the fact that, in many cases, the damaged surface areas still contain potential traces of wedge impressions. To verify these traces as

potential wedges, the location and depth of the indentations can be measured and correlated with common wedge depth and distances on the respective manuscript. However, measuring the depth of an indentation requires a reference surface which can most easily be approximated on digital 3D representations. Furthermore, the accuracy of manual measurement tools is often insufficient for measuring the subtle features that occur in this application case.

Hand copies are often created upon the initial analysis of a cuneiform text and can be an important part of its publication. A hand copy is a manual line-drawing that tries to reproduce the complete text contained on a cuneiform fragment in the form of a readable representation. For this, the text is planarized, which may be difficult on manuscripts with a curved writing surface where the text is written on and around the edges. Each wedge is represented by a standardized line-drawing, making the three edges and the wedge center recognizable. This leads to a drawing with horizontal text lines, additionally annotated with line numbers and museum inventory or catalog numbers. In contrast to photographic reproductions, hand copies show a well readable representation of the text including the text layout, the wedge shapes, marked damaged sections and the border of the writing medium including fracture faces. This leads to a good reproduction of the script characteristics. Hand copies are often created in larger quantities by the same Author, which facilitates the comparison of script features. For Hittite manuscripts, the large collection of hand copies *Keilschrifttexte aus Boghazköi* [68] represents a standard work in Hittitology. Aside from the large collections, hand copies are an important tool for personal work on a cuneiform manuscript that is enriched with annotations. However, a hand copy is always a subjective interpretation of a cuneiform manuscript, an opinion of what is written on the original artifact. As a result, hand copies made by other philologists are often re-collated using the original artifacts or photographies before using them in own research. This issue could be mitigated by using automatically extracted data form 3D scans to create hand-copy-like representations.

Currently established digital working methods in cuneiform script analysis include archives of digitized 2D representations like photographic reproductions and hand copies. Notable examples in this respect are the Cuneiform Digital Library Initiative (CDLI) [45], which hosts a large collection of digitized photographic material and high quality color images of cuneiform fragments taken using a standardized flatbed scanning procedure for cuneiform content[4]. This archive also includes parts of museum collections such as the Musée du Louvre Collection [150], the British Museum Collection [213] and the Iraq Museum Collection [214]. Specifically for Hittite cuneiform the Hethitologie-Portal Mainz [4] hosts a large collection of photographic reproductions of Hittite cuneiform documents, a Hittite bibliography, a text catalog and the concordance of Hittite cuneiform tablets. While there are no automated methods for searching the contents of the photographic reproductions, the bibliography, the catalog, and the concordance contain searchable information about the publications, the dating, the association of fragments, inventory numbers, hand copies, transliterations, transcriptions,

[4] http://cdli.ox.ac.uk/wiki/doku.php?id=submission_guidelines

and translations for many manuscripts. However, none of the available data inventories includes readily accessible information about low level script features or a similarity-based retrieval method for cuneiform signs or words.

Specifically with respect to supporting the previously detailed philological working methods, the low-level aspects targeted by the methods presented in this thesis are to improve digital collation and, for the first time, allow a quantitative comparison of low-level geometric cuneiform script features. In contrast to previous manual methods this will explicitly include the three-dimensional nature of cuneiform script. For joining cuneiform fragments the aim is to speed up the identification of join candidates via automated script analysis and provide methods for conducting and verifying virtual joins.

CHAPTER 3

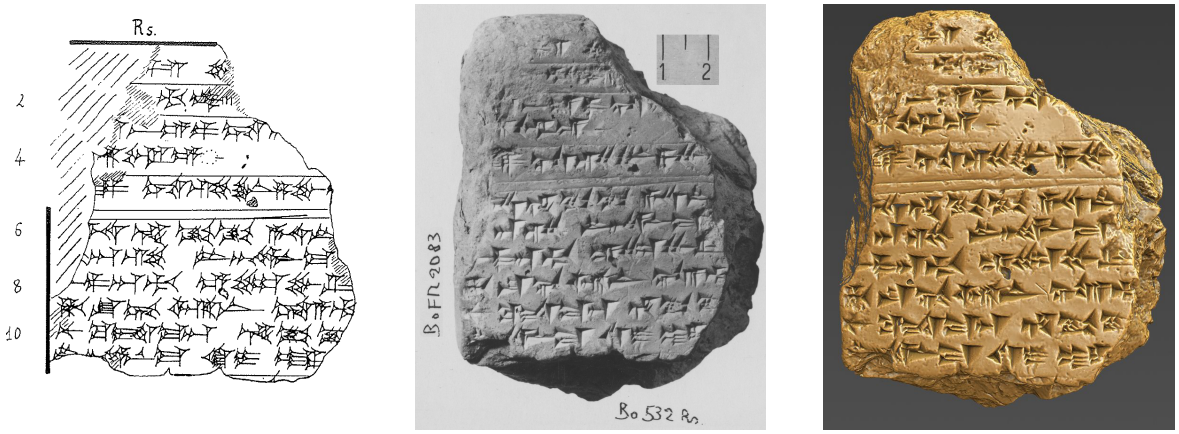
Digitization of cuneiform tablets

Metrological measurements on cuneiform manuscripts require suitable methods for systematic capture of the associated artifacts. This includes the selection and evaluation of capturing methods and hardware and the identification of appropriate methods and data structures for storage and data processing. Beyond the cuneiform fundamentals in chapter 2, section 3.1 will introduce traditonal reproduction methods for cuneiform artifacts. Sections 3.2 and 3.3 will then provide an overview of 2D and 3D digitization methods and outline their respective strengths and weaknesses. Section 3.3 will also include a detailed description of the 3D scanning hardware and the scanning method employed for capturing the cuneiform tablet data used in this thesis. Since the digitization also requires suitable methods for data processing and basic feature extraction, the subsequent sections 3.4 to 3.7 will cover aspects of mesh data structures, 2-manifoldness, mesh repair and curvature estimation.

3.1 Traditional reproduction

Prior to the availability of digital reproduction methods, archaeological artifacts containing cuneiform script were documented using variety of techniques including mostly two-dimensional but also three-dimensional methods. Among the most popular methods are reproductions in the form of hand copies and photographs. Less common methods are pencil and chalk rubbings, squeezes, and moldings.

As already mentioned in section 2.3, a *hand copy* is a hand drawn reproduction of a cuneiform manuscript which, besides other elements, also contains planarized text lines, fracture faces, text structure elements and line numbers. The main purpose of a hand copy is to serve as a readable representation for analyzing the contained cuneiform text. Other uses include the reconstruction of manuscripts and the documentation of the conservation state of the



(a) **(b)** **(c)**

Figure 3.1: Hand-copy **(a)** (Source: [188]), photographic documentation **(b)** (Source: [4]) and 3D scan **(c)** for the reverse side of cuneiform fragment *Bo532*.

artifact. Since cuneiform hand copies are usually published during the analysis of the contained fragments or as larger compilations [68], they are generally accessible to the scientific community. A potential lack of printed originals is increasingly being compensated by the digitization of printed inventories. Disadvantages of hand copies are that they do not always include a scale object for taking measurements and although the creators try to reproduce the script characteristics, hand copies always remain a subjective interpretation of the respective cuneiform artifacts. This is especially true for hand copies created as freehand drawings. However, hand copies can also be created by overdrawing photographic reproductions which significantly improves the accuracy with respect to taking measurements. This approach can be transferred to the digital domain by creating vector drawings on top of scanned photographs[1]. An example for a hand copy is shown in figure 3.1a in direct comparison with a photographic reproduction in figure 3.1b. It can be clearly seen that the hand copy is significantly distorted but offers a much better readability. One additional disadvantage of hand copies, even in digital form, is the lack of content-based retrieval methods for this kind of representation. However, their visual similarity to written documents in Latin script in terms of being line drawings and containing well aligned text lines suggest a general suitability of methods which have already been successfully applied to medieval documents.

Photographic reproductions, like shown in figure 3.1b, are a comparatively old and common technique for documenting cuneiform artifacts. As such, large inventories of historic black-and-white photographs taken at original excavation sites and in museum collections exist. These are increasingly digitized to provide access to this mostly analog form of representation. Under optimal conditions, black-and-white photographs can still offer very competitive detail reproduction and resolution, even in comparison to modern 3D techniques. With ongoing development of technical photographic capabilities, black-and-white photography was replaced

[1] http://www.digitalcuneiform.net/2015/07/how-to-draw-copies-of-cuneiform-tablets.html

with color and digital photography to extend the documentation and accessibility of museum collections. In contrast to black-and-white photographs, which usually only contain a non-standardized scale object and hand written inventory numbers, color and digital photographs offer extended capabilities for color reproduction. However, this does not necessarily help to overcome the inherent issues of photographic cuneiform documentation, the most prominent of which is that cuneiform wedges usually become only visible on photographs by enforcing appropriate self-shadowing. Therefore, a small light source has to be positioned in a way that the outer edges of the wedges cast a shadow onto the inner area of the wedges without introducing deceptive shadow lines near the inner edges of the wedges. As can be seen in figure 2.1, this can be achieved for most cuneiform types by using a light source positioned in an approximately 45° angle from the top left or top right side with respect to the line direction. The depiction of the shadowed areas is then improved by using appropriate fill lights. Since the optimal lighting angle varies between cuneiform types and is connected to the writing direction, finding a globally working light setup can be impossible, especially in the presence of curved writing surfaces or curved text lines. As a result, it may be inevitable to take multiple differently lighted photographs to capture a whole manuscript sufficiently. Aside from the shadow casting, controlling the depth of field may be an issue. Taking high resolution photographs of small objects is usually performed with camera lens setups that exhibit only a small distance interval in which the resulting images are sharp. This effect can be counteracted by using a smaller aperture or a smaller sensor which negatively affects the image resolution, the overall sharpness and the required exposure times. Instead, this issue is frequently addressed in older photographs by taking additional differently focused images of the affected surface areas. In digital photography, the issue can be targeted by a technique called focus-stacking, in which multiple images with different focus distances are fused to extend the depth range in a post-processing step. Another problematic aspect is related to the texture of the cuneiform fragments. In particular, surfaces with black burn marks or structure rich textures are difficult to capture in a way that reproduces the contained cuneiform structures properly. A complete solution is only achieved by the use of three-dimensional capture techniques which can handle geometry and textures separately. Despite its shortcomings and its inability to account for the three-dimensional nature of cuneiform script, photographic documentation is still a widely used technique.

Taking *rubbings* is a technique that was predominantly used for larger cuneiform inscriptions in stone. In order to create a rubbing, an appropriately sized sheet of paper is positioned over the artifact surface. Then, a piece of dark chalk of graphite is rubbed over the paper, coloring the paper depending on the resistance of the underlying surface. On the resulting rubbing, imprinted cuneiform wedges appear lighter than the surrounding writing surface, creating a 1:1 scale negative of the surface. A rubbing is well suited for taking large scale measurements of cuneiform stone reliefs but has very limited detail reproduction.

A *squeeze* is a three-dimensional reproduction that can be obtained by firmly applying one or multiple layers of wet blotting paper to a surface, pressing the paper onto the surface using

a brush and then letting the result dry[2]. After drying, the paper layer can be removed as a whole and retains the geometric shape of the surface to which it has been applied. Although this technique is very old, it can be very durable and, depending on additional conservation measures, even be used to cast a copy of the original object using plaster. In contrast to rubbings, a squeeze can also capture finer details and has a detail reproduction close to the original for larger structures [152].

Moldings can be produced of an object by submerging the object into a fluid substance that hardens over time and can be removed completely from the object surface as a whole. The resulting mold can then be used to cast a real world sized copy of the original object using plaster, plastic or metal.

In contrast to photos and hand copies, taking, rubbings, squeezes, or moldings are not contact-free methods and can therefore damage the original artifact. This issue can be mitigated to a certain extent by using professional casting materials, but using these methods on highly sensible artifacts is in many cases not an option.

3.2 2D Digitization

This section will introduce and discuss common 2D cuneiform digitization methods with respect to the suitability of the results for computer-aided script analysis. For this, the following will detail direct capturing methods like photographic documentation, flatbed scanning and Reflectance Transformation Imaging.

3.2.1 Photographic documentation

Creating digital photographic documentations of cuneiform artifacts generally follows the same procedures as the traditional analog photographic documentation. Working with a digital camera, however, removes some cost, time, and space restrictions connected with the development and storage of analog photographs and films. On the other hand, a digital infrastructure for storing the reproductions is required. Regarding the documentation process, digital photography introduces much more direct possibilities of quality control, since taken photographs can be checked immediately for quality issues. In contrast to flatbed scanning, taking photographs of cuneiform manuscripts still requires creating a suitable lighting setup to improve the contrast of the wedge features. As for analog photography, the inclusion of a scale object is still recommended, since the digital reproductions include no automatic measure for size reconstruction. In contrast, omitting color calibration objects is possible if the camera setup is calibrated with respect to the color of all included light sources. Aside from the direct creation of digital cuneiform images, a large amount of digital photographic cuneiform documentation is created by scanning existing analog photographs. Regardless of the creation method, further processing of digital cuneiform images is complicated by their large variation in lighting setups, resolution, sharpness, color rendition, and contrast.

[2] http://www.inschriften.net/themen/der-epigraphische-tipp/folge-15-was-ist-ein-papierabklatsch.html

This makes processing these representations for automated cuneiform retrieval a challenging task. Since many photographic cuneiform documentations originate from large documentation projects, e.g., in museum collections like the British Museum [213] or the collection of the Hethitologie-Portal Mainz [4], it may, however, be possible to exploit a certain degree of uniformity in the capture conditions of the images. Taking manual measurements on digital cuneiform images requires an extraction of the contained scale objects to obtain comparable results. Given a sufficient planarity of the captured writing surfaces and a low distortion camera setup, script feature measurements can be conducted, as long as no depth information is required and the size of the features is much larger than the distortion errors introduced during taking the respective photographs. However, this photographic error can in many cases only be estimated because the exact parameters of the camera and lens setup are unknown.

3.2.2 Flatbed scanning

Flatbed scanning is a cuneiform digitization technique suitable for cuneiform artifacts with a comparatively flat writing surface. This results from the fact that flatbed scanners can only illuminate and capture surfaces within a distance of a few centimeters above the scanning bed. In addition to that, the glass surface of flatbed scanners has a limited size and supports only a small amount of weight which imposes a size restriction on the artifacts to be scanned. The advantages of flatbed scanners for digitalizing small cuneiform fragments include the low cost of the hardware and the small amount of computational power required to process the scans. Besides an overall good color rendition and a low projective distortion, flatbed scanners also offer a competitive resolution. At 1200 dpi, the pixel size of a scan is approximately 21 micrometers, which is near the recommended resolution range for scanning cuneiform artifacts with a 3D scanner. For a successful complete capture of a cuneiform artifact, all sides have to be scanned. This usually requires the artifact to be held in place by a support structure on the scanner in several orientations. A comprehensive set of guidelines[3] for conducting cuneiform scans with a flatbed scanner has been specified by the CDLI [45] to enable scientists to contribute to the CDLI cuneiform collection. For this, a minimum resolution of 600 dpi is recommended to meet current archival image standards for small archaeological artifacts. Cuneiform flatbed scans are suitable for conducting script analysis measurements. As long as the scanning resolution is known, the features of interest can be measured in the 2D plane and the writing surface is sufficiently flat. However, 3D features, such as the wedge depth, can not be measured at all. In addition to that, the uniform illumination of flatbed scanners tends to produce images with little contrast on subtle cuneiform features, which makes many features on sub-wedge-scale hard to discover.

3.2.3 Reflectance transformation imaging

Reflectance transformation imaging (RTI) is a photogrammetric reproduction technique developed by Malzbender et al. [59, 133] which is able to create a so-called 2D+ representation

[3] http://cdli.ox.ac.uk/wiki/doku.php?id=submission_guidelines

of geometric objects. In this context, 2D+ refers to the fact that the resulting information is still image-based but enriched with specularity, normal, or depth related components. An RTI image is produced by creating a fixed camera setup with respect to the object to be captured and then taking multiple images while varying the position of a single point light source used to illuminate the scene from known positions. The dependence of surface luminance and light position is then modeled as a polynomial, whose parameters are approximated using least squares regression. With a solution for the parameters, images for arbitrary light directions can be interpolated. As an extension, the surface normals of diffuse surfaces can be approximated for each image point by exploiting that the luminance is maximal in the direction of the surface normal. The resulting normal field can then be used in conjunction with the recorded surface color to evaluate illumination functions like Bidirectional Reflectance Distribution Functions (BRDF) [80] and illuminate the image like a 3D object. An additional application uses the normal field to reconstruct a 3D mesh from the directional information encoded in the normals. Since this reconstruction is again performed by solving a minimization problem and the normal information contains inaccuracies, the resulting surface is affected by globally propagated distortions of the z-component perpendicular to the camera xy-plane. This distortion is especially prominent in surface areas with normals oriented perpendicular to the viewing direction of the capturing camera. Beyond the distortion issues, one important additional issue of RTI reconstructions is that an RTI image can only capture surface elements that are visible from the single camera position used. Overlapping geometry with respect to the associated viewing ray directions can not be captured at all. Although the produced 3D meshes can be superior to structured-light scans in terms of the amount of captured detail, the distortions prevent a usage of RTI-reconstructed surfaces as reliable sources for measuring cuneiform script features. An example of an RTI reconstruction of a cuneiform tablet from the Cornell University cuneiform collection, which was kindly provided by Hendrik Hameeuw from the KU Leuven[4], is shown in figure 3.2. While the top view in figure 3.2a suggests a moderately curved approximately brick shaped tablet, the side view in figure 3.2b shows the occurring global distortions in form of a significant curvature of the writing surface. Figure 3.2c shows a section of the tablet surface to demonstrate the good detail reproduction that even reproduces reed stylus markings in the wedges.

3.3 3D Digitization

Relevant properties of 3D scanning techniques with respect to scanning cuneiform geometry for script analysis not only include the accuracy of the measuring system in terms of resolution and color reproduction but also the resolution-dependent acquisition speed, the size of the scanning field, the object sizes that can be processed, the robustness, the mobility, the ease of use, and the price tag of the associated software and hardware.

[4] https://www.arts.kuleuven.be/oudegeschiedenis/english



(a) **(b)** **(c)**

Figure 3.2: An RTI reconstruction of cuneiform fragment *CUNES49-15-687* (Source: KU Leuven & Cornell University) from the top **(a)**, from the side **(b)** and a magnified section **(c)**.

For taking scans of small- to medium-sized cuneiform artifacts which are usable for collation purposes, a minimum required resolution of 25 μm which equals 1000 dpi is suggested[5]. However, some applications, like visualization for museum presentation, may require less resolution but instead a more accurate color reproduction. The acquisition speed in terms of individual scans or the whole process to digitalize an artifact is another important factor. Restrictions in this respect can emerge from the total number of artifacts in a collection and the resulting time required to scan all artifacts but also from the size of available time slots in a museum for taking scans of artifacts. Since the acquisition time of a scanner is often a trade-off against scanning resolution and the size of the scanning field, identifying a sufficient resolution and scanning field is a prerequisite to calculating the throughput of a scanning process. The required time in terms of man-hours can often be optimized by separating the scan post-processing from the acquisition process or automatizing the acquisition process by the use of devices to automatically reorient the scanned object or to position the scanner to capture the object from all directions. However, positioning scanners using robotic arms can be problematic if there is no guarantee that the movement will not collide with the artifact and cause damage. Other important aspects to consider are connected to the handling of the hardware. A scanner for acquiring cuneiform artifacts in foreign museums should be portable, require a manageable amount of setup time, and be robust against usual transportation conditions. Beyond that, the operation of the scanner should be manageable by one person.

In addition to the data acquisition, the processing and storage of the data has to be considered. Although, in most cases, 3D scans are stored in a proprietary data format used by the supplied scanner software, the simultaneous use of compatible and widely used data formats should not be neglected to ensure sustained data availability. The cuneiform tablets used in this thesis are

[5] http://www.museumsandtheweb.com/mw2003/papers/watkins/watkins.html

mostly stored in the flexible Polygon File Format (PLY)[6], developed at the Stanford University in the 1990s for the use with 3D scanners. Moreover, the raw data of the scanning software is archived in order to allow a later reconstruction of 3D meshes with improved algorithms or changed parameters. Since the raw data also contains the single scans before merging, this also allows later improvements to the scan alignment.

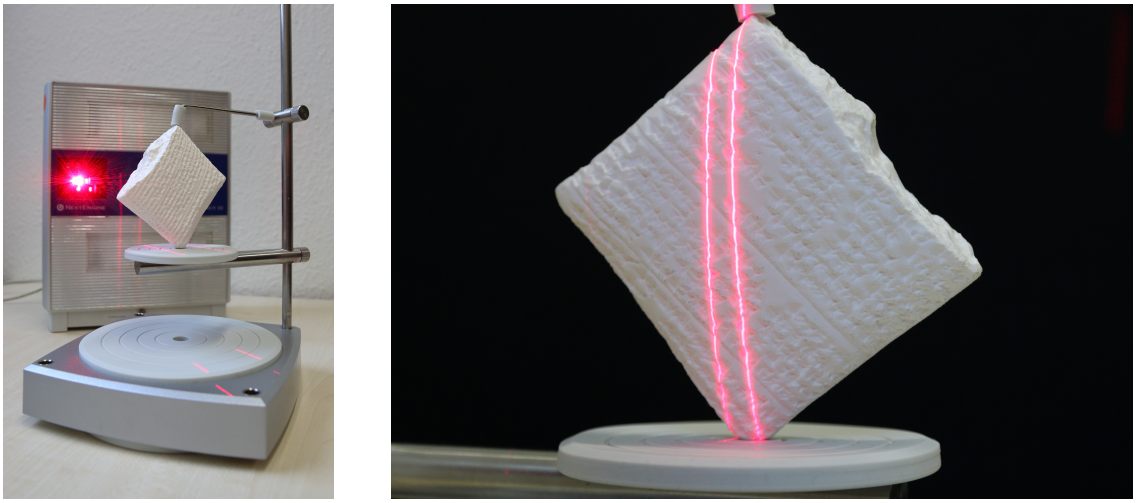
Aside from these general considerations, the respective weaknesses and strengths of the scanning technology have to be taken into account. Since cuneiform artifacts are usually sensible to body contact by hard objects, the use of tactile scanners that scan a surface by touching it with a hard needle or a spherical probe is not an option. With respect to popular contact free scanning techniques, the following subsections will address the usage of laser scanners, photogrammetric scanning and structured-light scanning more in detail. Holographic imaging, like described by Demoli et al. [56] will not be considered, since the resulting representations are not well suited for taking script feature measurements.

3.3.1 Laser scanning

Laser scanning 3D objects works by projecting a laser onto an object and capturing the reflected light to reconstruct the associated positions on the object's surface. This technique uses the ability of the laser light to be precisely focused on an object, even at large distances. The three main classes of laser scanners are mirror-based Time-of-Flight (ToF), phase-shift, and triangulation-based scanners.

Mirror-based ToF scanners control the direction of a laser beam via a set of adjustable mirrors and send out a laser pulse for each direction which is recorded with a light sensor. In most cases, the laser is guided over the object by using a scan-line or grid strategy to produce a two-dimensional coverage field over the scanning area. 3D positions of the laser are then reconstructed by measuring the time of flight for the laser pulse to be reflected at the object and return to be detected by the light sensor in conjunction with the known light direction produced by the mirrors. The grid accuracy of these systems is predominantly determined by the laser point size and the accuracy of the mirror system. The depth accuracy, on the other hand, is coupled to the accuracy of the time measurement accuracy of the laser pulse and the measuring distance. A smaller measuring distance requires a more accurate time measurement. Depending on the use case, this leads to a typical resolution between a few millimeters and several hundred meters for landscape scanning, with corresponding scanning distances of half a meter to several kilometers [100]. As a result, mirror-based laser scanners can only be used for larger cuneiform artifacts, such as stone reliefs. An additional disadvantage of ToF scanners is their acquisition speed. Since the scanner must wait for the laser pulse to return to take a measurement, the acquisition speed is limited to several hundred up to a thousand measurement points per second.

[6] http://paulbourke.net/dataformats/ply



(a) (b)

Figure 3.3: Cuneiform scanning with the NextEngine 3D laser scanner. The scanning unit with the auto-rotation unit is shown in **(a)** and the laser strips on the object to be scanned in **(b)**.

Phase-shift scanners represent an alternative to ToF scanners which continuously send out light. This results in a much higher data rate of several hundred thousand points per second as a trade-off against the usable dynamic range which also negatively affects the maximum scan distance. In contrast to ToF scanners, the distance information is retrieved by evaluating the phase of the reflected light. The application area of phase-shift laser scanners is similar to the one of ToF scanners, which results in similar usability for cuneiform scanning. As will be detailed in section 3.3.3, the phase-shift principle can also be used with white light instead of laser light, which is exploited by structured-light scanners and also affects working safety. White light has, however, a significantly smaller intensity than laser light, which lowers the available scanning distance.

Triangulation-based laser scanners have an accuracy of up to 0.05 mm, with typical working distances of ten centimeters to several tenth of meters [100]. This type of laser scanners usually works by projecting a laser stripe on the objects to be scanned and triangulate the respective positions by detecting the laser position using a single or stereo camera setup. For one individual scan, one or multiple laser stripes are moved across a predefined scanning field which the object to be scanned has to be positioned in. Since the triangulation-based position reconstruction neither requires high precision time measurement nor phase reconstruction hardware, this type of laser scanners can be produced at comparatively low costs. Scanners for small sized objects like cuneiform artifacts are often table-top or tripod mounted devices that offer a comparatively good portability. Using laser scanners for light sensitive objects can be an issue due to the high intensity of the utilized laser light.

Figure 3.3 shows an example of scanning a cuneiform fragment replica with the triangulation-based low-cost NextEngine 3D laser scanner. For taking a complete scan, the fragment is

positioned on an auto-rotation device in front of the scanning unit and scanned from multiple directions by alternately rotating the object and taking a single direction scan. During each scan, several laser strips are moved across the object and the laser illuminated positions are triangulated using two cameras in the scanning unit. In addition to that, the surface color is recorded at different brightness levels by taking multiple pictures while illuminating the object using a set of CCFL tubes in the main unit. The individual scans exhibit a resolution of 120 μm to 360 μm [203] and are aligned and fused in a post-processing step. Since taking one individual scan at full resolution can take up to 204 seconds, a complete scan of an object with two perpendicular object revolutions at 10 individual scans each can take up to 68 minutes, not considering the time for scan post-processing. Both the resolution and the acquisition time of the NextEngine can not compete with state-of-the-art structured-light scanners. However, considering the NextEngine costs only 1/10th the price of a medium priced structured-light scanner, its performance can be attractive for some applications.

More detailed information about using laser scanning systems for cultural heritage scanning, including several case studies, has been published by David Jones [100]. An evaluation of low- and high-cost scanners specifically for scanning archaeological artifacts, including Breuckmann OptoTOP structured-light scanners and the NextEngine 3D scanner, is available from Slizewski et al. [203].

3.3.2 Photogrammetric scanning

In general, photogrammetry denotes the science of taking measurements from photographic representations with the target to reconstruct positions on the depicted geometry. A very basic cuneiform application scenario is given by the traditional photographic manuscript reproductions that contain scale objects to enable metric measurements. With respect to 3D scanning, photogrammetry can be used to reconstruct complete surfaces from multiple images of the same real geometry. To go beyond the RTI reconstructions described in section 3.2.3, which also use photogrammetry techniques, a 3D surface reconstruction requires multiple images of the surface, taken from multiple viewing angles. If the intrinsic and extrinsic camera parameters are known, the position of points on the surface can be triangulated using multiple images. For this, the intrinsic camera parameters, such as the lens distortion, are usually obtained by a camera calibration procedure that is conducted before capturing the images of the geometry to be reconstructed. The extrinsic camera parameters, like position and orientation, are reconstructed by identifying a set of surface features that is included in multiple images. Consequently, the camera extrinsics are obtained as the solution of a system of equations constructed from the feature positions in multiple images. This enables all camera poses to be represented in the same coordinate system. Such a reconstruction can be carried out on the basis of any set of images as long as all areas of the surface to be reconstructed are sufficiently often covered and are represented from as diverse directions as possible. The object coverage can be improved by using precise manual camera pose planning or an automated system to position the camera for the individual shots. If an automated

device such as a robotic arm or a turntable is used, the extrinsic camera parameters can also be determined from the control parameters of this device. Thereby, it is theoretically irrelevant whether the object or the camera is moved. However, since moving the object may increase the chances of damaging the object it is generally preferred to move the camera. To enable true to scale measurements, a scale object of known size should be included in the images. Otherwise, photogrammetric reconstructions are representations without scale. In contrast to laser scanning, the distance range of photogrammetric scanning is only limited by the lens system, which can be adapted to capture near objects or far away objects. Another advantage of photogrammetric reconstructions is the possible low price of the involved hardware and the ability to acquire images remotely without a PC. Therefore, the recording procedure can be conducted by only using a device powered by a small rechargeable battery.

Aside from the previously described reconstruction approach with calibrated cameras, newer techniques also allow reconstructions from sets of un-calibrated images with unknown camera extrinsics. These dense stereo matching techniques [148, 209] focus on defining pixel correspondences in stereo image pairs [191] and benefit from recent advances in structure-from-motion [192, 217] techniques. Despite the available speed optimizations, most of these techniques are computationally expensive, especially for the reconstruction of very dense point sets, as required for cuneiform scanning. In addition to that, especially incremental structure-from-motion approaches suffer from error propagation issues, which may lead to global distortions.

Specifically regarding the digitization of small cuneiform artifacts, a promising photogrammetric approach is developed at the Pergamon Museum in Berlin. It includes a table-top device that can swivel a calibrated camera around an artifact lying on a small turntable in the center and achieves detail resolution comparable to and above the resolution of current structured-light scanners. The captured images are processed via a structure-from-motion-based software[7]. Photogrammetric methods were also intensively used in the creation of a digital 3D version of the Pergamon Altar[8]. A more complete and in depth review of photogrammetric techniques is available from Kraus et al. [116] and, with a focus on applications for museum purposes, from Mudge et al. [148].

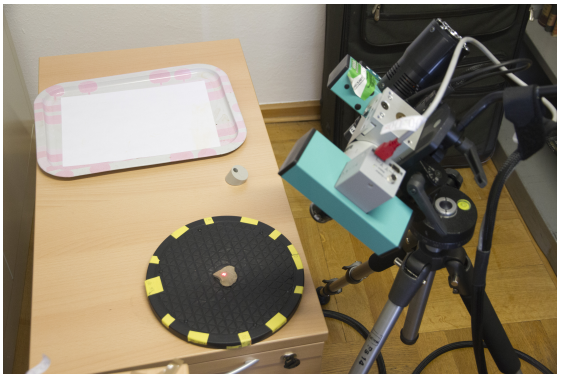
3.3.3 Structured-light scanning

Modern structured-light scanning uses a combination of techniques to produce contact-free high resolution scans. It works by projecting one or multiple sharply focused alternating stripe patterns onto a surface and recording the resulting light pattern with one or two cameras. These cameras are positioned to observe the surface from perspectives deviating significantly from the one of the light projecting unit. The resulting images are then processed utilizing aspects from the light section method, triangulation-based scanning, coded light projection, and phase-shift measurements.

[7] https://www.capturingreality.com
[8] https://www.igd.fraunhofer.de/projekte/pergamon-altar-3d

The light section method exploits that a thin projected line of light is deformed on a surface when viewed from a perspective shifted to the side with respect to the line projector. The surface geometry on this line can now be reconstructed by triangulation if the distance and viewing angle differences between the projector and the camera are known. This technique can also be used on the contrasting dark/light transitions of a sharp linear pattern of alternating equal width stripes, which enables measuring the surface geometry at many parallel and densely packed intersection lines. In order to distinguish the individual lines on the captured surface, the coded light approach can be used. For this, several alternating light and dark stripes are projected onto the surface in a short sequence, starting with a pattern consisting only of one light and one dark stripe. Each stripe is then successively replaced with one dark and one light stripe of half width until the finest stripe resolution is reached. The individual stripes in the final pattern can now be identified by a unique light-dark sequence in the recorded images which corresponds to a binary Gray code sequence. A more advanced description of structured-light pattern design is given by Gupta et al. [83]. In order to further increase the accuracy of a structured-light scanner, the device can use two cameras for triangulation and additionally evaluate the phase information of the individual camera pixels, like in phase-shift laser scanning. The increase in scanning accuracy when using two cameras is achieved by increasing the range of covered surface orientations. Therefore, the additional camera should be located at the opposite side of the projector with respect to the first one. A completely different structured-light scanning approach uses a projected pattern only to equip the surface to be scanned with a set of unique features and reconstructs the geometry purely photogrammetric. For these approaches, it is generally very beneficial to have more than one camera in the setup.

Typical structured-light setups include a tripod mounted scanning unit with two cameras and one projector using white light. Although white light offers less contrast than laser light, and thus a smaller possible scanning distance, white light is better suitable for light-sensitive surfaces and is generally safer to handle. The scanning distance, which, in combination with the size of the scanning field, is one factor for the resolution of a structured-light scanner, can be adjusted by using different lens setups for the projector and the acquisition cameras. Stationary devices can achieve spatial resolutions from 5 μm down to 100 μm and depth resolutions of up to 2 μm [85]. For small scale artifacts like cuneiform fragments that require a high resolution, typical scanning field sizes can range from approximately 60 mm to 90 mm with corresponding resolutions of 15 μm to 50 μm [64]. Although these scanning field sizes are in principle sufficient for capturing smaller cuneiform artifacts in one acquisition process, cuneiform artifacts often require taking multiple scans from different directions to capture all inner sides of the wedges sufficiently. This is caused by the fact that structured-light scanners perform best on surfaces oriented towards the optical axis of the scanner. Typical wedge geometries, however, tend to exhibit at least one wedge side oriented almost perpendicular to the optical axis of the scanner, resulting in a hole in the reconstructed mesh. These holes can be closed by superimposing the incomplete scan with a second one that has a better coverage of the affected surface orientation. Another issue requiring additional scans is connected to



(a)



(b)

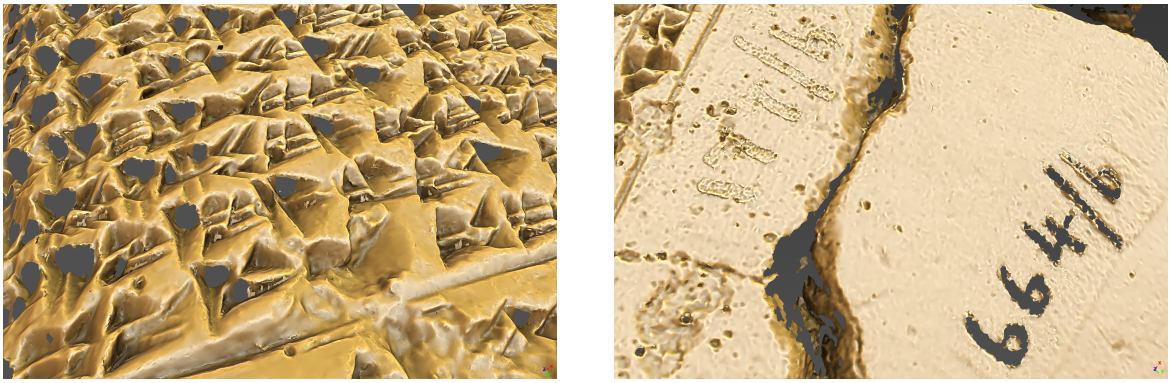
Figure 3.4: Cuneiform scanning with the Breuckmann OptoTOP HE structured-light scanner. The scanning unit on a tripod over a turntable is shown in **(a)** and one resolution of the stripe light pattern on the object to be scanned in **(b)**.

the possibility that cuneiform fragments can show strong color variations on their surface, for example, due to burn marks. As a counter measure, scans with different exposures can be taken. Sufficiently capturing the geometry in the scanning field may therefore require two to three scans from different directions and for each direction two or three different exposures. This multiplies the scanning times but is mitigated by the fact that one capturing process of a structured-light scanner can take less than a second. In terms of hardware costs, high precision structured-light scanners can be classified as expensive with prices starting at 40,000 €. Some smaller hand-held solutions with less precision are available starting at around 10,000 €.

In order to make optimal use of the available scanning times in museums, it is often preferable to perform additional scans on suspicion rather than a much more time-consuming direct examination of the scanning results. The latter is problematic because small coverage issues of a scan can only be seen after the scanner software has reconstructed and fused the scanning data from all previously acquired scans of a specific surface area. This requires the post-processing steps of aligning the scans, selecting the correctly exposed scans, and reconstructing the surface by fusing all aligned scans to one triangular mesh. Just like for laser scanning, this post-processing is usually carried out outside the available scanning times in order not to shorten the effective scanning time.

Most of the cuneiform scans used in this thesis were taken by Gerfrid G.W. Müller and Michele Cammarosano with a Breuckmann (now AICON 3D Systems[9]) OptoTOP HE structured-light scanner. This scanner can be operated by one person using a high performance mobile workstation. However, the mobility is restricted by the fact that the scanner requires mains power for operating the projector lamps and the cameras. Typical setup times of 15 minutes including unpacking and assembling the device enable an efficient use of scanning times. The specific lens configuration used results in a maximum resolution of 30 μm. For one scanned

[9] http://www.aicon3d.de



(a) **(b)**

Figure 3.5: Structured-light scanning issues on cuneiform fragments *120/m* and *177/b + 664/b*. An incomplete wedge face coverage due to a curved writing surface is shown in **(a)** and capturing issues on areas with written inventory numbers in **(b)**.

cuneiform fragment, this resolution produces file sizes ranging from a few megabytes to over one gigabyte, depending on the physical size of the fragments. Larger fragments require to sample the surface with a grid of overlapping scanning windows to cover the whole surface and have enough overlapping area between adjacent scans for performing a good mutual alignment. As described above, each scanning window requires multiple scans to be covered sufficiently. This resulted in scanning times of approximately 10 minutes for small cuneiform manuscripts and up to 2 hours for large fragments. The final scans were stored together with the raw scanning data to be able to reconstruct the scans with improved methods or parameters at a later point of time. Since the scans were acquired with the target of performing cuneiform script analysis, the primary goal was not to produce a complete error free and perfectly colored scan but to capture the script covered regions of the artifacts sufficiently well for the task. In contrast to scans taken for conservation purposes, the resulting meshes can therefore contain holes and inaccuracies produced during the scan merging process. This may have been prevented by investing significantly more time into the scanning and post-processing process. The resulting consequences for geometry processing will be detailed in chapter 4.

Figure 3.4 shows an example of scanning a cuneiform fragment with a Breuckmann Opto-TOP HE structured-light scanner. The scanner is mounted on a tripod and points at a turntable, with the optical center of the scanner being highlighted by a laser point for aiming. The yellow markings on the turntable shown in figure 3.4a are intended to help with the manual rotation of the artifact to evenly spaced orientations to cover one full circle. The medium fine stripe pattern on the cuneiform fragment in figure 3.4b shows the deformation of the contrast lines by the surface structure.

Two examples for typical structured-light scanning issues are shown in figure 3.5. The wedges on cuneiform fragment *120/m* in figure 3.5a have incomplete faces due to an insufficient scanner coverage caused by the curvature of the writing surface. Figure 3.5b on the other

hand shows artifacts on areas with written inventory numbers on the surface. These areas could not be captured well as a result of unsuitable exposure settings that were optimized to capture the surface without number writings. This can lead to noisy surface patches like for the inventory number “177/b” on the top left or holes in the mesh like for the inventory number “664/b” in the bottom right. With respect to script analysis, this is usually not an issue in the case of inventory numbers, since they are intentionally placed on script free areas of the surface. However, cuneiform fragments can also be affected by large-scale black burn marks that interfere with the scanning of the script and require exposure compensation.

More in depth information on specific structured-light scanning setups is available from Slizewski et al. [203] and FactumArte [64].

3.4 Mesh data structures

The choice of an appropriate data structure for managing three-dimensional meshes depends primarily on the intended type of usage and the application scenario. However, the requirements in different application scenarios are not always compatible with each other so that several data structures may have to be used simultaneously. In the context of processing 3D-scanned meshes, three primary types of usage can be defined, namely, geometric processing, visualization, and storage of meshes. Focusing on geometric processing, this section will detail the requirements for each usage type and identify suitable mesh data structures for the application scenario of cuneiform segmentation. Since the performance of a mesh data structure does not only depend on its concept but also on its implementation, the suitability of several mesh libraries will also be discussed.

As the most essential capability, a data structure for the processing of triangle meshes, like usually produced when taking 3D scans, needs to be able to represent all required topological peculiarities of the underlying meshes. This includes aspects like the presence or absence of mesh borders and holes, a required orientability [2] of the surface or applicable assumptions about the manifoldness [2] of the meshes. A second aspect is related to the types of algorithms the meshes will be processed with since this defines the required mechanisms for efficiently accessing elements of a mesh in the form of triangles, vertices or edges. Aspects to be considered include whether the mesh needs to be modified and which components will be affected since structural and topological modifications impose a different set of requirements to a data structure than a simple deletion of elements or a repositioning of vertices. Aside from that, algorithms may require efficient direct or iterative access to the mesh elements, constant-time access to adjacency information defining element neighborhoods, or the flexibility to add complex per-element properties to the mesh. However, most such additional capabilities include a trade-off against storage size, which implies that there is no data structure that is optimal for all applications.

The most basic data structure for triangular meshes is the triangle soup, where all triangles are stored individually in a list, each as a set of three coordinates for the participating vertices.

Though, the ability of this data structure to handle any topological and spatial arrangement of arbitrary triangles comes at the expense of very low memory efficiency for storing the geometry and almost no available connectivity information. For an improvement of memory efficiency, the vertex coordinates can be stored as a duplicate-free list with the triangles being defined as a separate list which references elements from the vertex list via their indices. Although this index-based shared-vertex data structure eliminates the redundancies of the triangle soup and adds connectivity information for vertices and faces, extracting local neighborhoods of vertices or faces still requires considerable computational effort. This lack of efficient access to connectivity information is addressed by edge-based data structures, such as the winged-edge [15] and the half-edge data structures [41].

The winged-edge data structure assumes bidirectional edges and stores the connectivity information in the form of references to other elements. The references per edge include four references to the next and previous edges with respect to the two adjacent faces, two references to the adjacent faces, and two references to the adjacent vertices, which results in a total of six references per edge. References from the edges to faces and vertices need only to be stored if those elements carry additional information, such as colors, normals or coordinates. Both faces and vertices store a pointer to one adjacent edge. Assuming the number of faces is approximately twice the number of vertices and the number of edges is approximately three times the number of vertices, a winged-edge data structure needs to store $27n$ references for a mesh containing n vertices. The mesh geometry can now be traversed by following the element references. A visual example for a winged-edge representation is shown in figure 3.6. The winged-edge $\mathbf{E}$ between the vertices $\mathbf{V}_1$ and $\mathbf{V}_2$ has adjacent faces $\mathbf{F}_1$ and $\mathbf{F}_2$ with the respective previous and next edges $\mathbf{E}_{1P}$, $\mathbf{E}_{1N}$, $\mathbf{E}_{2P}$ and $\mathbf{E}_{2N}$. Issues of the winged-edge data structure are the inconsistent orientation of the edges with respect to the faces and the consequence that traversing the edges around one face or one vertex requires code with many conditional branches.

These issues are addressed by the half-edge data structure which splits each edge into two oppositely oriented half-edges. In this formerly called directed-edge data structure [41], each half-edge includes references to one target vertex, to the adjacent face, to the next edge with respect to the adjacent face, and to the opposite half-edge. Optionally, a reference to the previous half-edge with respect to the adjacent face can be stored, resulting in a total of five references per half-edge. Additionally, for each vertex, one reference to an incoming half-edge is stored and for each face a reference to one adjacent half-edge is stored. Using the same assumptions as for the winged-edge data structure, the half-edge data structure needs to store $30n$ references for a mesh with n vertices. If no previous half-edges are stored, the memory consumption equals the one of the winged-edge data structure. A visual example for a half-edge representation is shown in figure 3.6b. The half-edge $\mathbf{HE}$ between the vertices $\mathbf{V}_{prev}$ and $\mathbf{V}_{next}$ has the adjacent face $\mathbf{F}_1$ with the previous and next edges $\mathbf{HE}_{prev}$, $\mathbf{HE}_{next}$ and the opposite half-edge $\mathbf{HE}_{inv}$. In contrast to the winged-edge data structure, the half-edges are oriented consistently clock-wise around the faces and traversing the adjacent half-edges

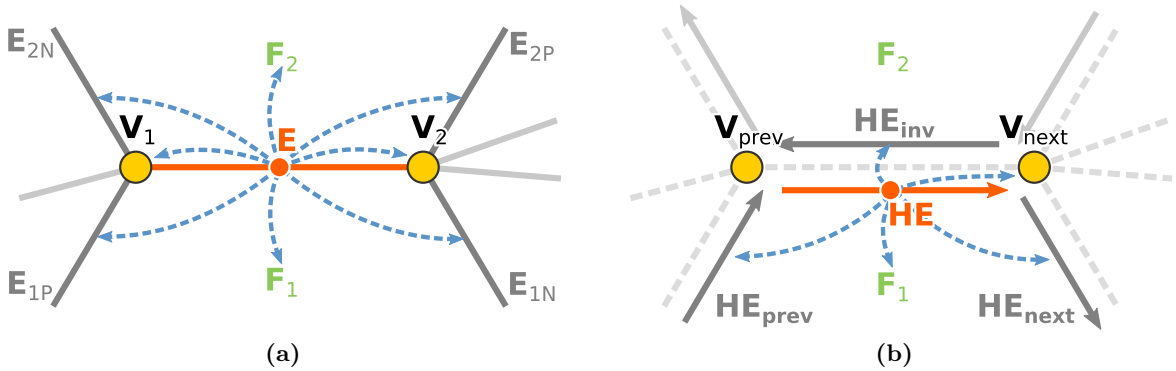


Figure 3.6: Winged-edge data structure **(a)** and half-edge data structure **(b)**. The **V** denote vertices, **E** Edges, **HE** half-edges and **F** faces. The blue arrows represent references.

of one face or one vertex requires no conditional branches. Instead, the half-edges can be cycled by following the next half-edge references around a face or by alternating between next half-edge and opposite half-edge references around a vertex. Consequently, as required for curvature computations and morphological operators, local neighborhoods of vertices and faces can be efficiently determined which is why the half-edge data structure is used as the mesh representation for geometry processing in this thesis. As will be detailed later, this includes using some implementation specific modifications of the mesh library OpenMesh.

The most important features of mesh data structures for visualization of large meshes include memory efficiency and compatibility to the data structures supported by modern graphics hardware. A generally advantageous approach for static meshes includes uploading the geometry and the associated element attributes and textures to GPU memory once and triggering the rendering process as required using low overhead rendering methods. Since most GPU data formats are array-based, it is advisable to also use array-based data formats on the CPU side. For OpenGL- or DirectX-based hardware the most suitable format for rendering large numbers of triangles includes indexed rendering. For this, the vertex coordinates are uploaded to the GPU as an array, complemented by a second array containing the triangles as triplets of indices to the vertex list, which is identical to the index-based mesh representation described earlier. This process of reducing the required data per rendered primitive can be optimized further by using triangle strips instead of triangles. However, using triangle strips for rendering requires a computationally expensive restructuring of the mesh [62] and changes the primitive order with respect to the half-edge mesh data structures used on the CPU side. Furthermore, triangle-strips are much less effective on indexed geometry because they only reduce the size of the index list to one third but not the size of the vertex list. This thesis uses index-based rendering methods to avoid expensive mesh optimization and index-remapping operations associated with triangle-strip-based rendering. An additional way to optimize mesh data structures for visualization is to reorder the vertices for optimal usage of the vertex caches of current graphics hardware. This is connected to an optimization of the vertex locality.

Techniques like the vertex cache optimization proposed by Hoppe et al. [93] are able to achieve rendering speed-up factors of 5 and higher depending on the original network structure and the hardware used. On the other hand, the vertex reordering requires considerable computational time which is an argument against repeating this process every time a mesh is loaded. The optimization would be best performed once after exporting the mesh in the scanner software and is therefore regarded as an optional external processing step. For performant primitive attribute updates, like required for updating vertex or triangle colors, the primitive attributes should be stored as compact arrays instead of interleaving all attributes for each primitive. This facilitates selective attribute updates by avoiding re-uploading unchanged data. Since the index-based mesh data structure for rendering does not support fast geometry processing, the data concept used in this thesis manages separate mesh data structures on the CPU side for geometry processing and on the GPU side for visualization. Storing primitive attributes as arrays on both sides enables fast data synchronization between the data structures.

A suitable data structure for the sustainable storage of 3D-scanned cuneiform meshes should be well-supported by existing software, memory-efficient, well documented, easily parsable, and include all necessary element attributes. Well-documented and broadly supported mesh storage formats for static meshes include the Stereolithography STL format [1], the Wavefront OBJ[10] format, and the Polygon File Format[11] (PLY). The STL-Format is the most common mesh storage format for rapid prototyping and 3D printing. It features an ASCII and a binary version and stores all mesh information per triangle as a polygon soup. The per triangle data usually includes the triangle's normal vector and the three vertices, which the binary format version stores with 32 bit precision. Disadvantages of the STL format include its memory inefficiency, the inability to store extended primitive attributes, such as colors, and missing connectivity information, which has to be reconstructed in a time-consuming vertex matching process after parsing an STL file. Especially the possibly non-deterministic reconstruction of the triangle connectivity disqualifies this format for storing cuneiform meshes for geometry processing. The Wavefront OBJ format is a pure ASCII format that supports storing meshes in an indexed way. The supported core feature set includes normals, colors, and texture coordinates for vertices and polygonal faces with an arbitrary number of sides. More advanced implementations also support free-form curve and surface definitions and extended connectivity information. While the feature set matches the requirements of cuneiform scans, parsing the text-based format can take considerable time on large meshes. Also, considering the many control sequences, the OBJ format is not particularly memory efficient. Much like the OBJ format, the Polygon File Format, developed at the Stanford University, features an index-based storage of vertices, polygonal faces, and edges. The attributes of these primitive types can be freely defined and include commonly supported properties such as positions, colors and normals but also user defined properties. Especially the possibility to define data types for the attributes makes the PLY format very flexible. In addition to that, its binary

[10] http://www.martinreddy.net/gfx/3d/OBJ.spec
[11] http://paulbourke.net/dataformats/ply

version is comparatively memory efficient. Together with its good support by widely used mesh processing software, this makes the PLY format a good choice for storing 3D-scanned cuneiform meshes.

In view of the existence of readily available and well tested mesh processing libraries, re-implementing an established mesh data structure is only rarely advisable. Furthermore, existing mesh libraries often include a non-negligible set of already implemented mesh processing algorithms. Well-known open source mesh libraries are the Computational Geometry Algorithms Library (CGAL) [63], the VCG Library [220] and OpenMesh [33].

The CGAL library is an international project by several universities, started in 1996, with a strong focus on supplying computational geometry algorithms. As such it includes a large variety of algorithms for triangulation, mesh generation, mesh simplification, shape reconstruction, bounding volumes, spatial searching/sorting, and arithmetic and algebra components. The library supports different 2D, 3D and nD geometry kernels and various data structures including the half-edge data structure. However, despite its good flexibility and its large set of functionalities, CGAL has several disadvantages for the here targeted use case. While dynamic sizes are supported for list-based mesh kernels, its array-based kernels can only handle static mesh sizes. This represents a performance problem with respect to updating the array-based GPU data structures. Moreover, GCAL does not support the temporary extension of primitives with attributes at run time. All used attributes must be permanently allocated, which leads to a bad memory efficiency. As for programming efficiency, CGAL uses elegant but heavily template-based data structures which can cause severe debugging overhead.

The VCG Library is a library for processing triangle and tetrahedral meshes and is developed at the Visual Computing Lab of the Italian National Research Council Institute ISTI[12]. It is mostly known as being the core component of MeshLab[13], an open source mesh processing tool. Much like CGAL, it contains a large selection of geometry processing methods but with a focus on triangular meshes. It supports multiple mesh representations, like the half-edge data structure, but relies mostly on the cell-tuple [35] data structure. To ensure a working basis for the implemented algorithms, the mesh representation used in the VCG Library must support capabilities, such as vertex-to-face adjacency and face-to-face adjacency, the latter of which is required for most of the implemented algorithms. Compared to CGAL and OpenMesh, the VCG Library features faster loading of many mesh file formats, a very compact memory footprint and can represent non-2-manifold geometry. On the other hand, many geometric computations like the calculation of discrete curvatures and remeshing are comparatively slow and can not cope with non-2-manifold geometry. This is particularly serious since the mesh repair methods for resolving the issues are also comparatively slow on large meshes.

OpenMesh [33] is a mesh library developed at the Visual Computing Institute of the RWTH Aachen University. It acts as a reference implementation of the half-edge data structure and

[12] http://vcg.isti.cnr.it
[13] http://www.meshlab.net

is intended as a generic, flexible and performant solution for processing polygonal meshes. An additional focus lies on ease-of-use and facilitates its usage by less experienced programmers. Like CGAL and the VCG Library, OpenMesh supports custom mesh kernels that are not limited to array-based representations. On the other hand, OpenMesh includes only a very small amount of algorithms. Its main advantage is the high flexibility with respect to the ability to extend all elements of a mesh with custom attributes at run time. Navigation on the represented meshes is mainly performed via iterators and circulators, which provide an efficient way to access local connectivity information. Concerning non-2-manifold geometry, OpenMesh can handle singular vertices but can not handle singular edges, both of which will be defined in section 3.5. When using an array-based data kernel, OpenMesh interfaces well with array-based GPU data structures and has a low memory footprint. Moreover, the explicit arrays for storing vertices, edges, half-edges and faces can also be accessed non-iterator-based, which enables the use of OpenMP[14] to accelerate geometric computations by using parallel processing. OpenMesh was used as the base for the development of the cuneiform segmentation and visualization methods presented in this thesis.

A more detailed comparison of mesh processing libraries regarding design, memory efficiency, and processing performance is given by Sieger et al. [202] including OpenMesh, CGAL, Mesquite and the newly developed Surface_mesh data structure.

3.5 2-manifoldness

Many algorithms in computational geometry require making assumptions about given properties of the underlying geometry in order to work correctly. For triangular meshes in three-dimensional space, one such property is the 2-manifoldness of the surface.

An n-manifold is a topological-space, where each point has a local neighborhood homeomorphic to the Euclidean space of dimension n [2]. A homeomorphism is a continuous function between topological spaces that has a continuous inverse function [121]. For two-dimensional manifolds, which are also called surfaces, this means that the local neighborhood of any point must be homeomorphic to a disc. A point that does not fulfill this criterion is called a singular point. Transferred to three-dimensional triangle meshes, which form internal connections via edges and points, this implies the existence of singular vertices and singular edges on non-2-manifold meshes. A set of possible mesh patches with respective topological properties is shown in figure 3.7. The 2-manifold mesh patch in figure 3.7a exhibits no singular components. The example in figure 3.7b on the other hand has a singular vertex highlighted in yellow. Taking a local spherical cutout around this vertex will always result in a geometric patch that is not homeomorphic to a disc, since the two patch components can be rotated around the point in a way so that the face orientations are no longer consistent. This is not true for the configuration in figure 3.7d. However, the respective cutout patch in figure 3.7d can not be

[14] https://www.openmp.org/

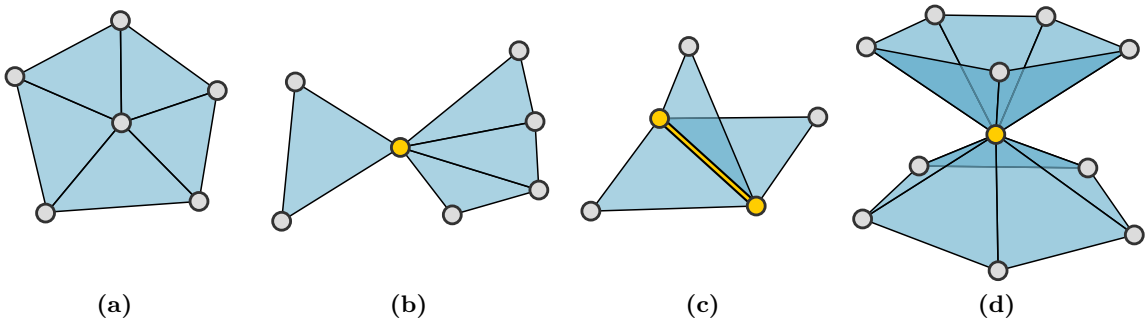


Figure 3.7: Examples for non-2-manifold geometric configurations on three-dimensional triangle meshes with the non-2-manifold components marked in yellow. A 2-manifold mesh patch is shown **(a)** and a singular vertex in **(b)**, a singular edge in **(c)** and a second type of singular vertex in **(d)**.

planarized, which consequently means it can not be mapped to a disc. This is also the case for the singular edge in figure 3.7c that has three connected faces.

Singular vertices and edges are not only an issue during algorithmic processing but directly affect the ability to be represented in certain data structures. For example, the half-edge data structure can not represent edges with more than two connected faces or two mutually adjacent faces with oppositely oriented normals. The latter would result in two connected half-edges where the respective opposite half-edge faces in the same direction as its counterpart. This is not allowed, since it would result in unpredictable circulator behavior because of unambiguous primitive referencing. The same is true for singular vertices, which is why many mesh libraries split singular vertices by duplication on loading non-2-manifold meshes. Singular edges are handled by splitting the edges and creating duplicates as required. For this error correction, the absence of a globally valid criterion for which edges to separate and which edges to keep fused, often prevents the production of optically pleasing solutions. Instead, in many cases, all involved edges and vertices are split. In this process, the number of faces stays constant. However, all corrected vertices, edges and half-edges should be marked to indicate the previously non-2-manifold configuration. Further aspects of mesh repair will be covered in the following section.

3.6 Mesh repair

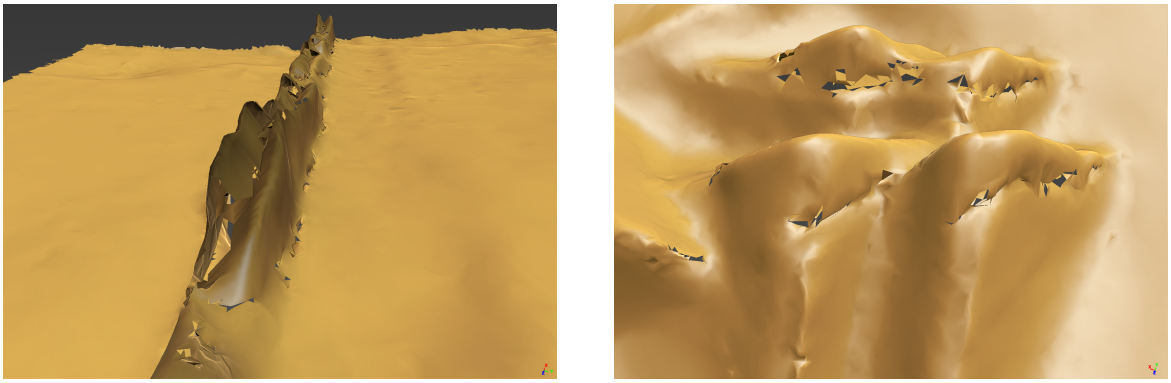
In the case of 3D scans of cuneiform artifacts, there is no computable quality measure that naturally meets all philological requirements in practice. Instead, available quality measures often focus on technical aspects like smoothness, closing holes or uniform triangle sizes. However, enforcing these objective quality criteria is not necessarily a good choice with respect to the deviation from the scanned real geometry and may therefore counteract philological objectives. Although cuneiform fragments can be generally assumed to exhibit a closed and mostly smooth surface, they may also contain sharp fractures, small holes or self-occluding micro structures in the form of adhering dirt particles. Especially small, deep features may

not have been completely captured by the scanner, leading to holes in the resulting mesh. In this regard, geometric patches that have been created during the fusion of neighboring or overlapping scans are particularly prone to geometric defects in the form of non-2-manifold or unrealistically overlapping geometry. Having these defects on a mesh is generally undesirable but it can also indicate issues with the data quality. Repaired mesh errors, on the other hand, may no longer be visually detectable and may at the same time produce unrealistic geometry which no longer resembles the original surface of the scanned object. This can only be detected by collating the original artifact.

Occurring mesh errors include local and global error types. Among the local connectivity errors are the 2-manifoldness issues already mentioned in section 3.5, like isolated or singular vertices and edges and inconsistent normals at touching faces. In addition to that, vertex noise, degenerated triangles, holes, gaps, and unwanted changes in mesh tessellation density may occur. Global issues, on the other hand, include topology issues, like a non-realistic surface genus, isolated surface patches, or mesh deformations due to scanning artifacts, which may lead to overlapping geometry. The decision whether a particular type of error needs to be addressed by mesh repair operations depends not only on the severity of the error and the intended use case but also on the mesh representation used. While a non-2-manifold mesh can be visualized without any issues, storing such a mesh in a half-edge data structure or using certain types of algorithms may be problematic.

In general, mesh repair can be performed surface- or volume-based. Surface-based methods operate on the existing mesh surface and apply modifications to fix the geometry. This may not only include single operations, like removing singular elements, but can require multiple pipelined steps to achieve topology reconstruction or remove degeneracies [12]. Volume-based approaches first transform the existing mesh into a volumetric representation, like an octree, voxels, or a distance field that is then used as a base for the repair operations. Regardless of surface- and volume-based operation, a distinction can be made between local geometry correction and global reconstruction. However, the transition between the two approaches is fluid. In most cases, mesh repair through global reconstruction is equivalent to surface reconstruction from point clouds, which is covered in depth by Campen et al. [42] and Berger et al. [17, 18]. Notable examples include the Power Crust Method [6] that uses a medial axis transform for reconstruction, the advancing front surface reconstruction [48], and the Poisson surface reconstruction, which will be described more precisely in section 4.5.2. Approaches for global reconstruction can also be used for a class of local methods which in a first step remove existing mesh errors and then fill the resulting holes in a subsequent step. The hole filling can be realized by considering the surrounding surface via volumetric diffusion [54], copying other surrounding surface patches for context aware filling [199], ensuring surface smoothness by using simulated annealing [227], or using a Poisson approximation [239].

For 3D-scanned cuneiform meshes used in this thesis, some of the above mentioned methods are already applied during the assembly process of the scans in the software provided by the



(a) **(b)**

Figure 3.8: Examples of scanning errors with ambiguous mesh repair quality targets on cuneiform fragments *485/b Rs* and *Bo2234*. An ambiguous column separator scan is shown in **(a)** and an angle dependent merging error in **(b)**.

manufacturer. Since the utilization of this bundled software is often preferred by the users or even mandatory for many scanners, the use of repair methods can not be fully controlled and may remain undetected. Moreover, scan merging may contain a significant amount of manual work, especially if the scans are not taken with an automatic mechanism for scanner or object positioning. Consequently, most 3D scans have gone through a scan post-processing in which the user tries to maximize the quality of the scanned data based on visual assessment of the result or at least maintain a subjectively sufficient data quality. Since the quality criteria of this process are not necessarily known by the actual users of the scan, further automatic repairs may worsen the data quality with respect to these criteria.

Two examples for these kinds of quality criteria conflicts are shown in figure 3.8. Figure 3.8a shows a scan of a paragraph separator viewed from below the surface that in reality consists of a single tapered elongated indentation. However, the scan of this geometry consists of two intersecting indentations riddled with holes, resulting from merging two misaligned or distorted single scans of the respective area on the cuneiform fragment. The combined geometry appears as two parallel grooves, or ridges, when viewed from the reverse side. Automatically repairing the geometry could result in a single groove which does not match either of the two ridges in the scan data or only one of the ridges. The result could also consist of two grooves or be significantly smoothed and no longer reflect the original depth of the groove. Since the most reality conform result can not be determined based on the 3D scan, leaving the geometry untouched, so that the error can still be seen, becomes a viable option. The wedge group shown in figure 3.8b shows consistent mesh damage to the right faces of the wedges. Repairing the mesh by removing the damaged regions and filling the holes would result in geometry with a very small deviation from the damaged version. However, doing so would render the defects undetectable and some small remaining scratches could be mistaken for structures on

the cuneiform stylus. The decision whether this would be a disruptive effect depends on the application case.

In some cases with extensive mesh damage or obvious scan alignment issues, repeating the scan merging and post-processing process in the scanner software may be an option if the 3D-scanned raw data is still available. However, significantly changing the mesh processing parameters for specific scans may introduce systematic errors into statistical script feature evaluations. It is therefore strongly encouraged to develop a standardized scanning and scan post-processing method that is used consistently for all scans.

In this thesis, mesh repair is only performed to break up singular vertices and edges to ensure processability by the OpenMesh half-edge data structure. This prevents unwanted false data resulting from mesh repair and leaves mesh errors detectable. Instead, the segmentation methods presented in the chapters 4 and 5 are constructed to be robust against mesh errors but also include a mesh error detection, which will be detailed in section 4.4.

3.7 Curvature estimation

Much like in 2D image processing, surface analysis and segmentation tasks on three-dimensional meshes and point clouds often require the presence of features in the form of first and second order derivatives. As a consequence, the computation of normals and curvatures plays an important role in the analysis of 3D-scanned meshes. However, triangular meshes are piecewise linear surfaces, which is why the usually continuous definition of normal vectors and curvatures can not be trivially transferred to the computation and interpolation of these properties on triangular meshes. An exact computation may not even be possible, since the vertices of a mesh may be a normal and curvature independent sampling of a continuous surface which was acquired by taking a 3D scan. Therefore, normals and the depending curvatures on meshes are usually approximated using discrete differential methods or via estimators in scale-space.

3.7.1 Discrete differential curvatures

In the continuous case, the curvature of a point $\boldsymbol{p}$ with normal vector $\boldsymbol{n}$ on a 2-manifold surface S can be computed by considering the curvature of all curves resulting from intersecting S with normal planes through $\boldsymbol{p}$. A normal plane is defined as a plane that includes $\boldsymbol{n}$ and is perpendicular to the tangent plane in $\boldsymbol{p}$. Assuming the direction of each normal plane is given by a unit vector $\boldsymbol{e}_\theta$ at $\boldsymbol{p}$ that lies in the respective plane, the normal curvature $\kappa^N(\theta)$ is defined as the curvature of the intersection curve of S with the normal plane. All normal curvatures can be used to derive the mean curvature κ_H which is defined as

$$\kappa_H = \frac{1}{2\pi} \int_0^{2\pi} \kappa^N(\theta) d\theta. \tag{3.1}$$

This includes two extremal normal curvatures, the maximum curvature κ_1 and the minimum curvature κ_2 on perpendicular normal planes, which are called principal curvatures. With the dependency $\kappa^N(\theta) = \kappa_1 \cos^2(\theta) + \kappa_2 \sin^2(\theta)$ [207], the mean curvature can also be expressed as

$$\kappa_H = \frac{\kappa_1 + \kappa_2}{2}. \tag{3.2}$$

Aside from the mean, minimum and maximum curvatures, the Gaussian curvature can be used to characterize the surface and is defined as

$$\kappa_G = \kappa_1 \kappa_2. \tag{3.3}$$

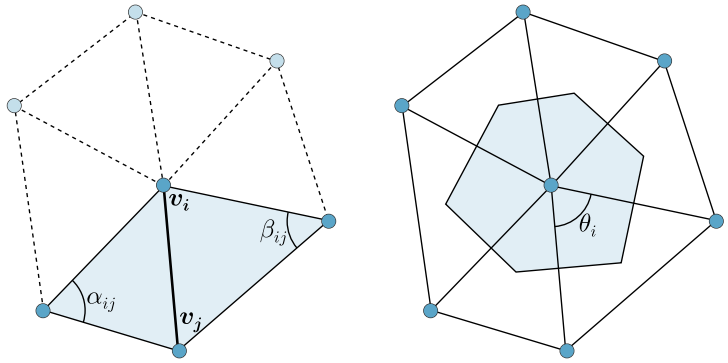
Positive Gaussian curvatures indicate that the surface is shaped locally like a dome with all normal curvatures sharing the same sign. In contrast, negative Gaussian curvatures indicate a surface shaped like a saddle point. A Gaussian curvature of zero occurs when one of the principal curvatures is zero and it is associated with a parabolic surface shape. All four types of curvatures are well suited to serve as surface features in 3D surface segmentation tasks.

The continuous curvature formulations can be applied to polygonal meshes. A very basic but common technique for approximating vertex normals on triangular meshes consists of calculating all triangle normals using a cross product and then deriving the vertex normals as a weighted average of the normals of the adjacent triangles. The weighting factors can be uniform, consist of the triangle areas or of the triangles' angles towards the vertex. However, since these options do not fully incorporate the neighboring vertices into the approximation, the result may deviate significantly from the respective continuous differential operators and higher order derivatives, like curvatures, tend to become unstable. This is especially an issue for tasks like mesh simplification and mesh subdivision.

An improvement in this respect is the method proposed by Meyer et al. [145] that uses averaging Voronoi cells to provide a good approximation to the continuous differential operators. The central idea of the method is to extend the continuous normal and curvature definitions to discrete meshes using spatial averages. For the computation of per vertex curvatures, this requires a consistent assignment of surface patches for averaging to the mesh vertices. As a starting point, these surface patches are computed as the Voronoi area A_{Voronoi} around each vertex $\boldsymbol{v}_i$ using its 1-ring neighborhood $N_1(i)$ as

$$A_{\text{Voronoi}} = \frac{1}{8} \sum_{j \in N_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij}) \|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2. \tag{3.4}$$

As shown in figure 3.9, the angles α_{ij} and β_{ij} are the opposing angles in adjacent triangles of the edge between the vertices $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$. This definition of the Voronoi area works well on non-obtuse triangles but for obtuse triangles the circumcenter point lies outside the triangle area. This is an issue, since an important condition for the accuracy of the area assigned



(a) **(b)**

Figure 3.9: Opposing edge angles for the computation of the Vononoi area in a 1-ring neighborhood around vertex $\boldsymbol{v}_i$ **(a)** and the resulting Voronoi area with inner angle θ_i **(b)**.

to each vertex is that every surface point does only contribute to one vertex. As a solution, a modified Voronoi area A_{mixed} has been introduced [145] that uses the mid-points of the triangle edges opposing the obtuse angle. Since the obtuse angle of a triangle does not change, regardless in which 1-ring it is considered, this leads to a consistent area assignment for all vertices. Consequently, the computation of A_{mixed} is derived from the computation of A_{Voronoi} by replacing individual vertex contributions in the sum with the respective new values if the considered triangle is obtuse. This new values are equal to half of the triangle area if the obtuse angle is located at the center vertex of the 1-ring and equal to one quarter of the triangle area if the obtuse angle is located at another corner.

The resulting modified Voronoi area A_{mixed} can now be used in the mean curvature normal operator

$$K(\boldsymbol{v}_i) = \frac{1}{2A_{\text{Mixed}}} \sum_{j \in N_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\boldsymbol{v}_i - \boldsymbol{v}_j) \tag{3.5}$$

to compute the mean curvature κ_H at vertex $\boldsymbol{v}_i$ as $\kappa_H(\boldsymbol{v}_i) = \frac{1}{2}\|K(\boldsymbol{v}_i)\|$. The Gaussian curvature κ_G can be computed by iterating the faces of the 1-ring using

$$\kappa_G(\boldsymbol{v}_i) = (2\pi - \sum_{j=1}^{f} \theta_j)/A_{\text{Mixed}}, \tag{3.6}$$

where f denotes the number of faces in the 1-ring. The remaining principal curvatures κ_1 and κ_2 can be obtained as

$$\kappa_1(\boldsymbol{v}_i) = \kappa_H(\boldsymbol{v}_i) + \sqrt{\Delta(\boldsymbol{v}_i)} \tag{3.7}$$

$$\kappa_2(\boldsymbol{v}_i) = \kappa_H(\boldsymbol{v}_i) - \sqrt{\Delta(\boldsymbol{v}_i)}, \tag{3.8}$$

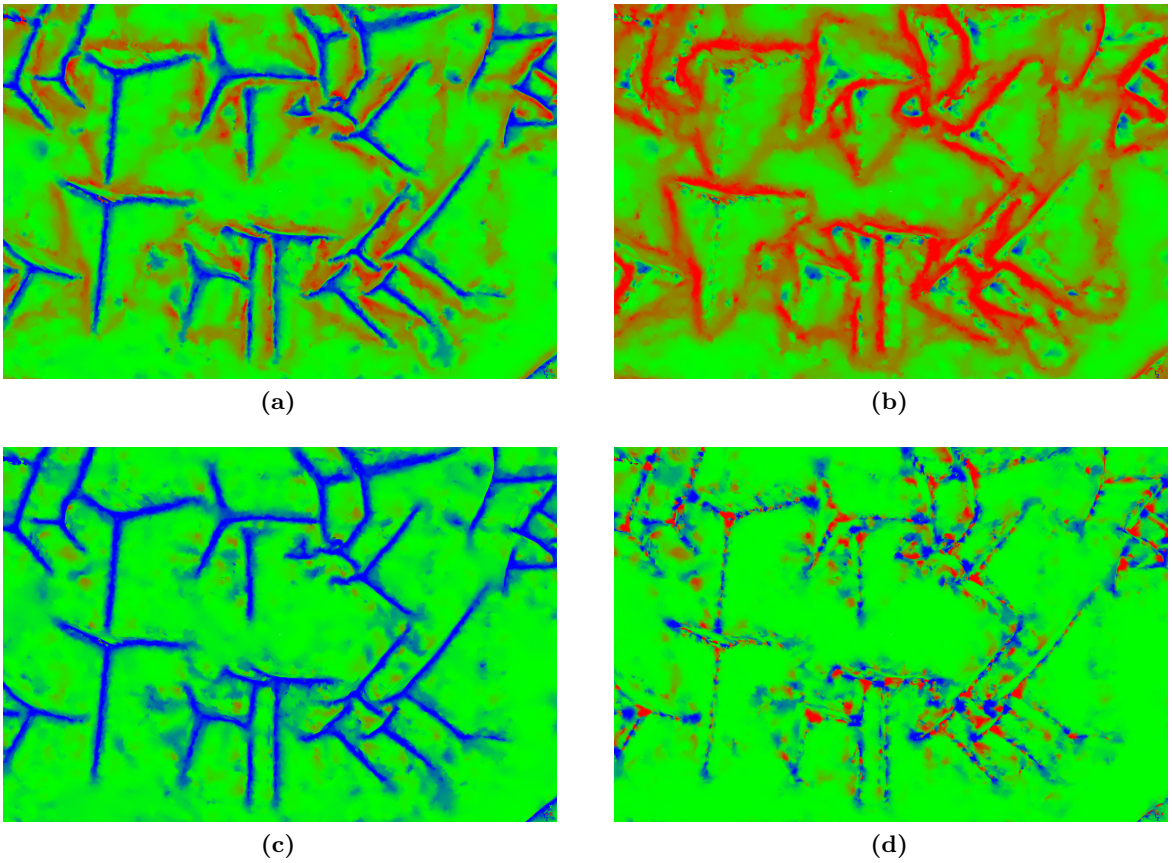
with $\Delta(\boldsymbol{v}_i) = \kappa_H^2(\boldsymbol{v}_i) - \kappa_G(\boldsymbol{v}_i)$. It is worth mentioning that the method of Meyer et al. is susceptible to numerical issues in the case of degenerated triangles and when $\Delta(\boldsymbol{v}_i)$ is smaller than zero. These issues can be counteracted by introducing appropriate thresholding of numerically critical components into the computation and has been found to be sufficiently stable for the applications covered in this thesis despite the problematic characteristics of 3D-scanned meshes. This is partly due to the fact that many uses of curvatures in chapters 4 and 5 use remapped or thresholded curvature values. Particularly for visualization, extreme curvature values can also be clipped by normalizing the curvature values to an interval depending on the standard deviation of the vertex curvatures on a mesh. However, this method can not only be used for visualization purposes but also to facilitate the choice of threshold values for further processing, like detailed in chapters 4 and 5. A histogram equalization can be used in a similar way although this has the tendency to heavily modify the relative strength of curvature features. Since all computations are performed on the 1-ring of the vertices, the curvature computation can be parallelized and efficiently computed using the half-edge data structure from section 3.4. Consequently, computing the curvatures is comparatively fast. For a medium-sized mesh with 3.4 million vertices this can be performed in 1.26 seconds on an Intel Core i7 4790K. As will be shown in section 4.10.3, this computation time scales approximately linear with the vertex count. For use cases with higher demands to the curvature quality and robustness, Rusinkiewicz described a method based on averaging per face curvatures [182] that can be extended to higher order surface derivatives.

To demonstrate the suitability of discrete curvatures as features for cuneiform segmentation and visualization, an example of the four vertex curvature types computed for a cuneiform mesh is shown in figure 3.10. The employed color mapping uses blue for negative curvatures, green for zero curvature and red for positive curvatures. In addition to the color mapping, the visual clarity of the curvature values has been increased by normalizing the curvature values to two times the standard deviation of the curvature values.

3.7.2 Scale-space curvature estimators

Curvatures calculated using discrete differential geometry are coupled to the local topology of the underlying mesh and to its sampling density. This may impose restrictions to the scale of the features that are represented by the respective curvatures, especially when comparing uniform and adaptive sampled meshes. In contrast, methods for estimating second-order properties in scale-space are not necessarily bound to the mesh resolution or the local topology of the mesh. As a result, these estimators can be used to compute curvature related measures with respect to a user defined scale, on multiple scales, and for geometry that can not be handled by the method of Meyer et al. like point clouds.

Most scale-space methods for analyzing manifolds in 3D space can be grouped in one of three categories being based on differential geometry, numerical approximation, or local regression. Differential geometry-based methods relying on the connectivity information include several established 2D methods adapted to the domain on three-dimensional meshes,



(a) **(b)** **(c)** **(d)**

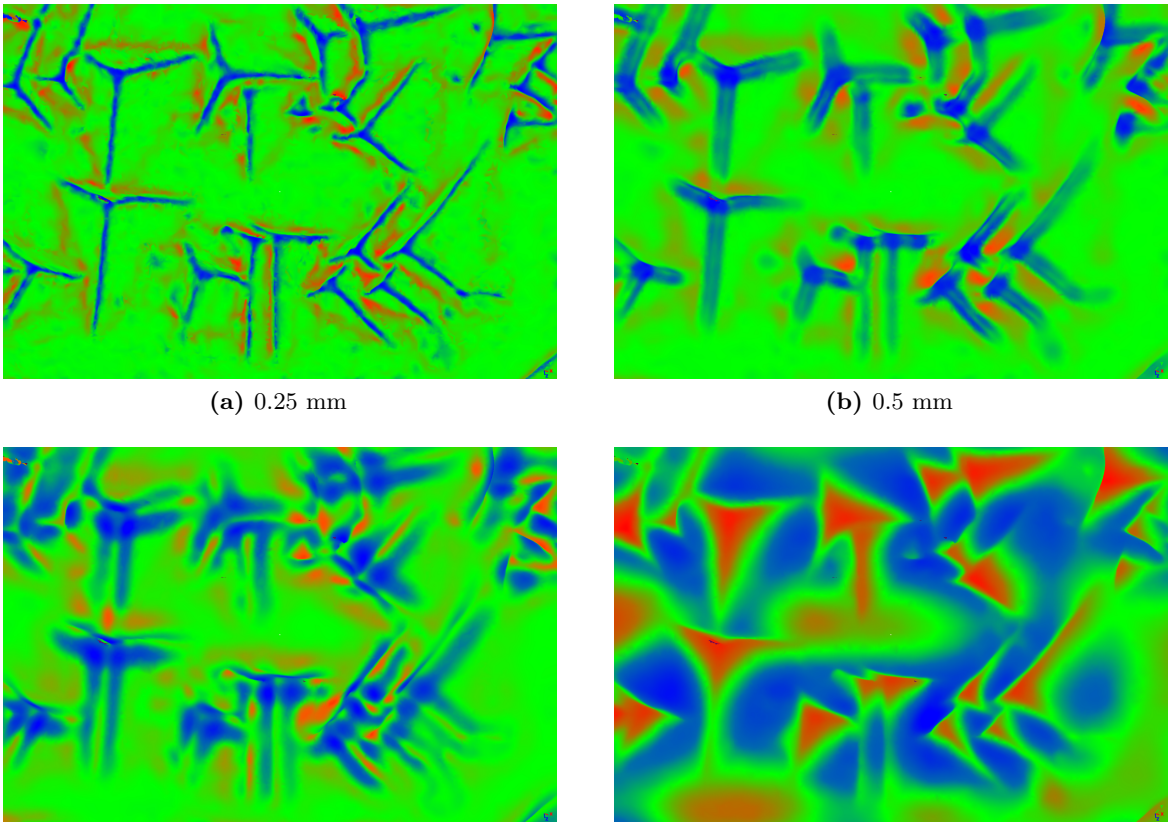
Figure 3.10: Discrete curvatures on cuneiform fragment *Bo532*. Shown are the mean curvature **(a)**, the minimum curvature **(b)**, the maximum curvature **(c)** and the Gaussian curvature **(d)**.

like Difference of Gaussians (DoG), Histogram of Gaussians (HoG) and Scale-invariant Feature Transform (SIFT) [130]. The adapted versions like MeshHoG [238], MeshDoG [238] and MeshSIFT [205] are particularly useful for geometry matching since the generated points of interest can be used to characterize similar objects. However, since the points of interest are generated automatically, searching for specific small geometric features can be an issue. As a second application, methods like MeshHOG can also generate multi-scale descriptors for local geometric surface features. Numerical approximation methods like integral invariants [162] usually compute a local manifold surface measure that is related to the curvature, like the volume of a sphere intersection with the surface or an integral over an available distance function. The approximation target can be one of the four curvature measures defined in section 3.7 but also derived measures, like the difference of the principal curvatures. One strength of integral invariants is their ability to act as local geometry descriptors by considering multiple scales. This can, for example, be used to detect low level surface features like edges, smoothness or planarity [124] but also to identify more complex features. A method for applying integral invariants to the task of cuneiform wedge segmentation will be covered in section 4.2.

A very similar approach is taken by methods that use local regression. There, a simple geometric kernel, like a plane, a sphere, or a paraboloid, is fitted to a set of local input points via regression. The fitted kernel is then used to derive properties related to differential invariants. A well-known method of this group is Moving Least Squares (MLS) [125], which can be used for surface interpolation, smoothing, and derivatives' approximations when applied to 3D data. A notable example for the regression-based construction of scale-space descriptors in 3D is the Growing Least Squares (GLS) method of Mellado et al. [142, 143]. As an extension of MLS, the method works by fitting algebraic hyper-spheres to a local neighborhood of normal annotated points using a fitting method of Guennebaud et al. [82]. Missing normals can be approximated in advance using covariance analysis. The resulting hyper-spheres, whose predefined radius corresponds to the feature scale, are then reparameterized, yielding the algebraic offset distance between the evaluation point and the 0-isosurface of the sphere, a unit normal of the scalar field at the evaluation point, and the signed mean-curvature of the hyper-sphere as the new set of parameters. Since this set of three parameters already contains a curvature, an evaluation of the sphere fitting on a single scale for every vertex of a mesh results in a curvature approximation that can be used for detecting features on this scale.

Figure 3.11 shows four examples for the mean curvature term of Growing Least Squares for scales between 0.25 millimeters and 4 millimeters. While the result in figure 3.11a resembles the discrete mean curvature in figure 3.10a, the remaining scales show significantly different characteristics, which demonstrates that certain features are only available on specific scales. This is addressed in Growing Least Squares by considering multiple scales. For this, the geometric descriptors in form of the sphere parameters are evaluated on multiple scales and scale-space analysis is used to identify scales where the geometric descriptors exhibit minimal variation according to a variation function that involves all three parameters. The idea behind this is that descriptors which are only insignificantly influenced by a varying scale are likely to represent characteristic features. Thus, GLS targets to identify multiple pertinent scales to derive geometric feature descriptors for points. These features can then be used to identify similar geometric regions on a mesh. However, this requires the features of interest to exhibit specific features on specific scales. While the method works well for detecting edges or bumps on consistent scales, complex features like cuneiform wedges, which can exhibit significantly differing sizes and contain edges of significantly differing curvature, can be an issue.

The computation times of GLS increase quadratically with the scale which determines the size of the neighborhoods for sphere fitting. For adaptively resampled meshes, like commonly produced by scanner software, this is usually not an issue since most small features can be computed using neighborhoods with manageable amounts of points. In addition to that, GLS can be efficiently accelerated by performing the neighborhood search around the evaluation points with spatial hashing [212] and by implementing the method on the GPU [143]. As a result, the computation times of GLS on typically sized cuneiform fragment with several hundred thousand to million vertices range from several seconds to a few minutes, depending on the choice and the number of scales.



(a) 0.25 mm **(b)** 0.5 mm

(c) 1 mm **(d)** 4 mm

Figure 3.11: Growing Least Squares curvature on cuneiform fragment *Bo532* for a set of four different scales.

3.8 Conclusion

From the presented traditional and digital 2D and 3D cuneiform reproduction methods, 3D-scanned cuneiform meshes exhibit, by far, the best suitability for metrological script analysis. They not only surpass the measurement capabilities of the other methods but also make new levels of script features accessible. Amongst the available scanning techniques, structured-light scanning can be identified as the most appropriate method for cuneiform scanning, considering both resolution and scanning field sizes. The set of requirements to mesh data structures varies for different applications like geometry processing, visualization and storage of triangle meshes. However, different concepts like the half-edge data structure, the PLY format and indexed data structures can be combined to cover the demands of the methods, developed in chapters 4 to 6. This includes the fulfillment of topological prerequisites like 2-manifoldness and the fast computation of differential geometric properties like curvatures. The creation of error free scans is an important aspect of cuneiform scanning but being able to handle meshes with errors can be an equally vital task to not introduce additional errors.

CHAPTER 4

Cuneiform wedge extraction

Reliable wedge and wedge component extraction are prerequisites for a model-based quantitative analysis of cuneiform script. This chapter introduces a novel fault-tolerant pipeline for automatic wedge and wedge feature extraction on 3D-scanned cuneiform manuscripts, with the goal of providing quantitative access to formerly unavailable layers of information. Starting with a description of available data sources and philological requirements in section 4.1, section 4.2 will give an overview of methods for 3D geometry analysis and previous work regarding 3D analysis of cuneiform data. Section 4.3 will then outline the wedge extraction pipeline, with a detailed description of pipeline stages in sections 4.4 to 4.7. Subsequent sections 4.8, 4.9, 4.10 and 4.11 will then present methods for wedge model extraction and wedge validation, and evaluate the segmentation quality of the presented pipeline. The chapter will close with a conclusion.

This chapter is based on and extends the author's publication at SCCH 2013 [71].

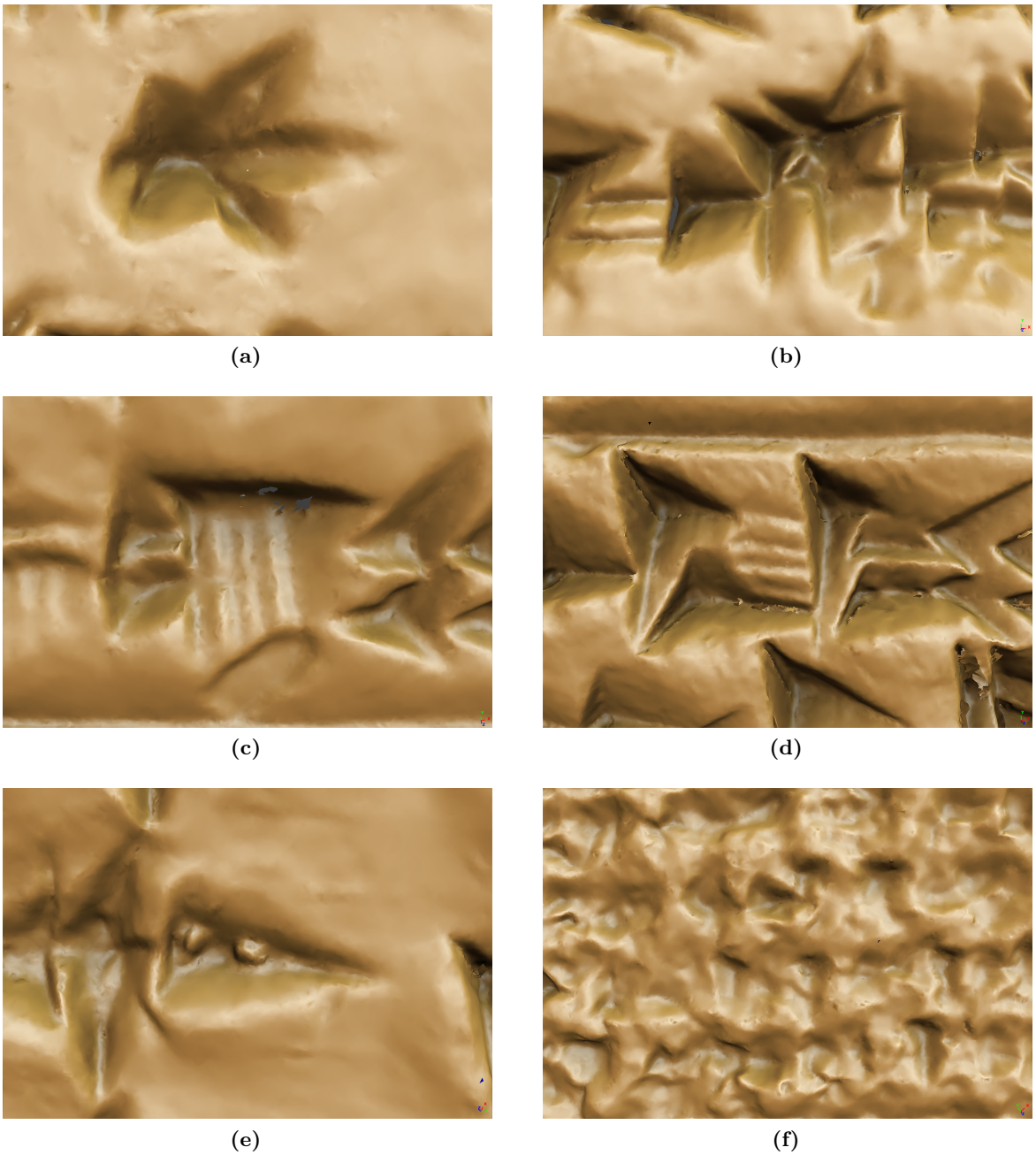
4.1 Introduction

Extraction and analysis of cuneiform script characteristics from 3D-scanned datasets, as described in section 3.3, to target the use cases covered in section 2.3, is a complex task. Aside from the geometrical issues resulting from damaged artifacts and the process of 3D scanning, as discussed in section 3.5, the geometric structure of cuneiform script itself introduces multiple demanding segmentation challenges. These script-induced issues include cuneiform wedges varying significantly in size and shape, mutually overlapping wedges, damaged wedges, and very subtle imprints that are practically indistinguishable from geometrical details contained in fracture faces. Figure 4.1 shows several examples of problematic wedges. The two wedges in the center of figure 4.1a forming the cuneiform sign *nu* mutually protrude each other, leading

to wedges that can not be represented by one connected patch of triangles in the underlying 3D mesh. Instead, wedges like this may consist of multiple unconnected geometry patches with additional interfering features, caused by other overlapping wedges. In addition to that, large parts of the individual wedge geometry may no longer be represented by actual clay surface geometry. A more complex example with at least four overlapping wedges is shown in figure 4.1b. The overlapping vertical wedges shown in figure 4.1c share a common top face with virtually no geometric features separating the contributions of the participating wedges. In addition to that, the protrusion partially eliminated the geometric representation of multiple inner edges and outer points according to the wedge model presented in section 2.2.3. Although the visual separation of overlapping wedges usually does not pose a problem to a cuneiform expert, many cases exist where considering the surrounding wedges regarding possible wedge combinations to form a known cuneiform sign is inevitable. A common philological strategy for the interpretation of largely overlapping wedges includes deriving an impression order for the participating wedges. This way, the assignment of surface patches to wedges can be simplified using a principle of exclusion. The four subtle horizontal wedges shown in the center of figure 4.1d exhibit a very low amount of internal geometric features and a comparatively small surface area. This renders a successful wedge extraction, let alone a model-based feature extraction very difficult. Additional issues are caused by wedges like the one shown in figure 4.1e that are partially filled with soil or small particles. As a result, a geometric representation of the original depth point is not available and the scanned surface may even assume the form of multiple smaller wedges. A last class of problematic wedges includes damaged wedges like shown in figure 4.1f. These wedges are located on an abraded surface area and are partially difficult to distinguish from wedge-free fracture faces.

In view of these data inherent peculiarities, a usable wedge segmentation method should provide a stable wedge extraction and target as many of the above mentioned issues as possible. However, since even philologists experience difficulties in resolving some of the issues by only considering local geometric features, a wedge segmentation method operating locally on 3D-scanned cuneiform data is not expected to yield perfect segmentation results. Instead, it should be complemented with techniques for improved scan data reviewing which will be described in section 6 and techniques for flexible segmentation result reviewing and editing which will be detailed in section 6.4.6. This way, philologists can judge the quality of the segmentation and extend the segmentation meta-data with subjective interpretations.

The overall goal of the wedge extraction presented in this chapter is to provide a fault tolerant method for extracting individual model-based quantitative wedge data. Since the extracted data is supposed to be used for statistical script analysis, cuneiform sign metrology, and creating a 3D cuneiform sign database, the model fitting is required to yield comparable, reliable, and reproducible results for a wide range of geometric wedge representations and with a sufficient extraction rate. In this context, the fault tolerance should target mesh errors, like described in sections 3.3 and 3.5, and geometric peculiarities of cuneiform script on the



(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 4.1: A collection of examples for complex wedges on scanned cuneiform tablets *1587/u*, *26/e*, *27/a*, *431/s* and *E1544*. Two occurrences of prodruding wedges are shown in **(a)** and **(b)**, four vertical wedges sharing a common face in **(c)**, significantly damaged lines of wedges in **(d)**, four subtle horizontal wedges in **(e)** and a filled horizontal wedge in **(f)**.

scanned artifacts. The extraction results can then be used for statistical cuneiform script analysis.

4.2 Related work

The geometric analysis of 3D-scanned datasets, representing surfaces in the form of point clouds or triangular meshes, consists predominantly of methods for object recognition, model segmentation and 3D feature detection. While object recognition focuses mostly on the classification and localization of whole 3D objects, model segmentation targets the decomposition of 3D point clouds and surfaces into semantically meaningful parts. In contrast, the main goal of 3D feature detection is the identification of distinctive features which can be used for object recognition or model segmentation but also for feature tracking and feature matching tasks. The specific approach for extracting wedge shaped surface features from 3D-scanned triangular meshes presented here, predominantly comprises aspects of model segmentation and feature detection, although wedge imprints could be regarded as objects.

Many traditional methods for object recognition, like described in [197], are based on 2D image analysis. These methods are basically applicable to 3D point clouds and triangle meshes by generating 2D images from the 3D datasets using projective transformations. However, limitations apply for colorless, non-textured 3D datasets and sparse points clouds with a comparatively small surface coverage. Many 2D object detection/recognition approaches are focused on the task to separate an object from the background and to classify subsets of images as being part of specific objects or the background. Contrary to that, object recognition in 3D datasets often involves an analysis of the objects structure to differentiate multiple object types, which often requires incorporating methods for 3D segmentation or 3D feature detection. One popular group of methods to realize this are skeletonization methods [131], which construct a 3D skeleton from the dataset to simplify the structure of the analyzed objects. This typically results in a graph-based representation which can be employed for object recognition by using methods for matching graphs [123]. Skeletonization methods for feature detection will be covered more in depth in chapter 5.

Another popular group of methods for 3D object recognition uses features and templates. There, the object is described by a reference set of features or a so-called template that is matched with features on the dataset to be analyzed. Methods for general feature detection on 3D objects include Normal Aligned Radial Features (NARF) [208], the Scale Invariant Feature Transform (SIFT) [130, 185], Fast Point Feature Histograms (FPFH) [184] and Signature of Histograms of Orientations (SHOT) [218]. An in depth comparison of these feature detectors can be found in [86]. Two additional groups of methods for object recognition are based on classifiers and object motion. Classifier-based methods try to generate a mapping for the objects to be detected into a set of known object classes. In most cases, this is realized through the use of machine learning approaches, like Support Vector Machines (SVMs) [52, 160] or Convolutional Neural Networks (CNNs) [22, 206, 236] using a set of training data.

This also works well on voxel data [163]. As 3D cuneiform datasets are of static nature, methods based on object motion are not expected to be applicable on this type of data. One disadvantage of many 3D object recognition approaches is that the methods detect the objects and provide their location but do not include additional information about specific properties of the detected objects. Therefore methods for object recognition are often combined with methods for 3D segmentation to extract additional properties of the recognized objects. This general approach will be adapted here for the extraction of wedge areas in section 4.7 and a subsequent model fitting in section 4.8.

Three-dimensional segmentation methods usually operate either on 3D point clouds, polygonal meshes or voxel data. As the cuneiform scans considered here are only available as point clouds and triangle meshes, methods for handling voxel data will not be covered in this work. Some common approaches for the segmentation of 3D point clouds include clustering and model fitting. Clustering methods try to group the points based on a similarity measure to maximize similarity within the groups, but at the same time minimize the similarity between groups. For cases when the number of clusters is not known in advance, hierarchical clustering methods can be used. A more in depth overview on clustering approaches will be given in the course of section 4.8.1, as the wedge face extraction will be realized clustering-based. The basic idea of model fitting is to find a set of parameters for a model to approximate the given data. This includes methods like the Hough transformation [94] and the Random Sample Consensus (RANSAC) [69] method. While the Hough transformation can be used to detect, for example, large amounts of lines or circles in the existing data, RANSAC is more suitable for fitting a single model into a subset of point data. Point cloud or triangle meshes can then be segmented by approximating them with multiple fitted primitives or, for example, by using the detected lines to partition the data. A more in depth description of RANSAC will be given in subsection 4.8.2, as it will be used for the extraction of the wedge edges.

Since the cuneiform datasets processed in this thesis consist of triangle meshes, the additional connectivity information can be used to incorporate topological or surface-based features into the segmentation. Therefore, methods specifically designed for point clouds will not be covered in depth here. However, as 3D polygonal meshes are defined on a set of points, methods for point cloud segmentation can naturally be applied to 3D meshes. For more details on point cloud segmentation please refer to the review of Grilli el al. [81] and the documentation of the Point Cloud Library [185].

The additional surface-based information, available on triangle meshes can be especially useful for the detection of local surface-based features, like cuneiform wedges. More global features, related to the topology of a mesh can in contrast be used to guarantee specific mesh properties, like the 2-manifoldness, already covered in section 3.5, that may simplify the processing of a mesh. Enforcing these properties is covered by methods for mesh repair [12, 42] which are particularly useful on non-synthetic, 3D-scanned meshes. However, repairing a mesh can introduce false data into a 3D scan and result in false measurements. While

some additional surface features on triangle meshes can be accessed by simply combining the connectivity information with the surface normals, more advanced features rely on curvature-based information. Aside from the discrete curvatures described in section 3.7, curvature-based features can be extended well to represent features on multiple scales. Thus, methods like Curvature Scale-Space 3D [65] or Growing Least Squares [143] can be used to detect complex multi-scale features and features with a specific scale. Especially line like features on the mesh surface can then be used to divide the model at locations with creases or ridges and decompose it into semantically distinct patches [178]. This works particularly well if the scale of the dividing curvature features is known in advance, but may not work at all in the presence of mesh holes or non-2-manifold geometry. When applying a curvature-based decomposition to cuneiform wedges, it must be considered that discrete curvatures may also contain scale dependent information, when computed on a curvature adaptive remeshed triangle mesh. As these kinds of meshes are frequently produced by many scanning software during scan fusion, this could be an issue. However, as discrete curvatures are, in most cases, less computationally expensive to compute, they should be preferred where appropriate.

Like on point clouds, SIFT-based methods can also be used on triangle meshes to detect multi-scale features like faces [132, 205]. Being able to detect multi-scale features on cuneiform data could be advantageous for distinguishing fracture face geometry from writing surface and wedges. Aside from their usability for object recognition, skeletonization techniques can further be employed for 3D segmentation purposes. For this, the geometry can be assigned to the branches of the skeleton and separated accordingly [168, 169]. Segmenting a 3D object into multiple perceived patches can additionally be realized using watershed-based methods [134, 155]. Watershed methods submerge the processed domain based on suitable property that acts as a virtual height field, and use the resulting basins as segments. However, the tendency of watershed methods for over segmentation and their susceptibility to mesh errors can be an issue. As the wedge extraction will be based on a watershed approach, this will be covered in more detail in section 4.7.

Like object recognition, object segmentation can as well be approached via methods from the field of machine learning. However, this usually requires the availability of thoroughly labeled training meshes [102]. With respect to extracting wedges from cuneiform data, this could be an issue, as the creation of labeled datasets is a very time-consuming task. This factor becomes even more important, when trying to optimize a machine learning approach for different cuneiform variants as it would require training data for each script variant. In addition to that, some classification candidates, like fracture faces and abraded writing surfaces or different types of wedges, are extremely similar, which would probably require even bigger amounts of training data for a proper differentiation. Then again, the lack of labeled datasets could be addressed by using unsupervised learning methods like auto-encoders [127] on randomly chosen patches of cuneiform meshes. Using auto-encoder approaches could be particularly interesting for learning cross-domain representations on available multi-domain data, like photographic reproductions, hand copies and 3D models of the same cuneiform manuscripts.

For more feature-based methods related to 3D shape analysis and mesh segmentation, please refer to [36, 165, 195].

The large majority of related work, specifically dedicated to automated processing of digital cuneiform representations, focuses only on the digitization of artifacts. This includes 2D and 3D representations. Since there are only few works that deal with a further processing of the data for the purpose of an extraction of script characteristics or classification, the following will mention works related to the processing of 2D and 3D cuneiform data. This also makes sense because some 2D methods can be extended to work on 3D data as well.

An early experiment on processing cuneiform hand copies to classify cuneiform signs was conducted 1997 by Roshop et al. [177] to show that neural networks are appropriate for 2D cuneiform handwriting recognition. Their similarity to modern handwritten documents, however, makes hand copies a comparatively easy target for feature detection. In one of the first works on processing 3D cuneiform data, Anderson et al. [7] presented an approach for automated unwrapping of 3D cuneiform scans and evaluated the application of accessibility-based coloring for detail enhancement in 2002. The latter is similar to the ambient occlusion used in section 4.5.1 but contains a scale related component. Works with a focus on digitization, like [120, 234], cover topics such as the development of scanning strategies and the registration of individual scans. Starting in the early 2000s, this lead to several digitization related projects like the Hethitologie-Portal Mainz [4], Digital Hammurabi [229], the Cuneiform Digital Forensics Project (CDFP) [233], and the Cuneiform Digital Library Initiative (CDLI) [45], some of which are also dedicated to the preservation of cuneiform manuscripts and related philological research. A new impulse for processing 3D cuneiform data was given in 2010 to 2013 by Mara el al. [136–138] which presented an automated method for extracting a 2D vectorized representation from 3D-scanned cuneiform documents. This method was based on using multi-scale integral invariants to extract the scale dependent details of cuneiform wedges for the construction of an intermediate skeleton representation and, eventually, a usable 2D vector representation. This vector representation mimics the usual appearance of hand copies and is therefore an important step in cross-domain processing of cuneiform data. The integral invariants were further used to optically enhance the cuneiform script data during visualization of the 3D-scanned meshes, which allowed to enhance specific features like wedges or fingerprints. A compact version of the robust automated wedge and wedge feature extraction technique presented in this thesis was first published in 2013 [71]. This technique was subsequently integrated into a processing pipeline for automated detection of candidates for finding matching cuneiform fragments [72]. Later, some alternative approaches for fitting a wedge model into manually extracted wedge geometry were evaluated by Richardson et al. [170]. These, however, did not include an automatic method for extracting the required wedge geometries or an approach for handling damaged meshes.

Regarding the computer assisted reconstruction of cuneiform manuscripts Collins et al. presented a GPU-accelerated method for matching fracture faces based on depth maps [50]. This

approach can solve the fragment reconstruction problem for selected pairs of fragments if their bounding boxes can be aligned, the fracture faces are not occluded, and the fragments share a pair of directly matching fracture faces. The Growing Least Squares approach presented by Mellado et al. [50, 143] can extract scale dependent surface features similar to the integral invariants used by Mara et al. [138]. These features could be used for cuneiform script detection and for matching fracture faces. The work of Mellado further included a method for semi-automatic matching of fracture faces [142]. This method is interesting in the context of cuneiform fragments, as it does not require directly contacting fracture faces. In 2015, Rothacker et al. used a 3D curvature-derived 2D visualization from [72] for segmentation-free word spotting of cuneiform structures [180] to address the general issue of automated content-based search in cuneiform representations. Targeting a very similar subject, Bogacz and Mara used 2D vector representations of cuneiform signs to derive a graph representation for character recognition [29]. The cuneiform vector representations were based on splines and either drawn by hand or generated using the method from [138]. In a subsequent work, Bogacz et al. [30] included the translation of 2D raster data into a similar skeletal representation like the intermediate skeletal representation in [138]. In contrast to the manual vectorizations used in [29], the automatically generated graph representations require further work to separate graph components belonging to individual wedges. This issue was targeted in [30] with methods for wedge-head detection and wedge-arm tracing. Further work on matching cuneiform sign components was conducted in [28] to match cuneiform signs in the vectorized representation. With a similar intent as Rothacker et al., Bocagz et al. evaluated a method for searching large scale 2D cuneiform representation databases [27] using spatial n-grams. The work on vectorizing 2D raster images of cuneiform hand copies was continued in [140] by introducing an extraction method based on bitmap tracing and skeletons. This included additions to the methods for identifying wedges in the produced skeleton data. In order to continue the efforts to search in databases, Bogacz et al. presented a segmentation free method for the spotting of cuneiform characters [25] in hand copies. Subsequently, using the same cuneiform data, Bogacz et al. evaluated the current possibilities for automated transcription of cuneiform manuscripts [26].

4.3 Outline of the method

The wedge segmentation methods proposed in this chapter will target two primary objectives, a watershed-based wedge extraction and a feature-based wedge model extraction. Both objectives are part of the wedge extraction pipeline depicted in figure 4.2. This pipeline operates on the input of a three-dimensional triangular mesh, obtained by scanning cuneiform artifacts with a structured-light scanner, as described in chapter 3. The first stage described in section 4.4 includes a mesh error detection as part of the mesh preprocessing, computing an estimation for the quality of the input mesh, which can be used by subsequent pipeline stages. A second preprocessing stage, described in section 4.5, performs a reconstruction of the hypothetically wedge free writing surface to determine a reference system for measuring

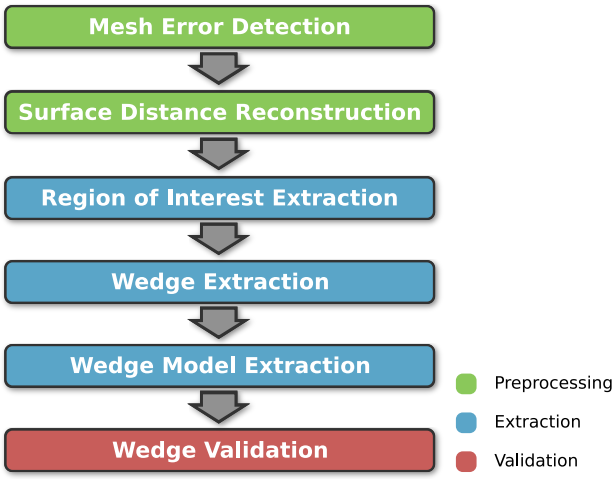


Figure 4.2: The wedge extraction pipeline, with the pipeline stages colored depending on their operational domain.

the surface distance of the vertices of the wedge geometry. The region of interest extraction, as detailed in section 4.6, then narrows down the data to be processed in subsequent pipeline stages by identifying geometric regions that potentially contain wedges. By adapting the extraction parameters, these regions can already be used for determining wedge groups. The regions are processed further in section 4.7 to extract candidate regions for individual wedges. The wedge candidate regions are then subjected to a detailed analysis in section 4.8 to attempt extracting the tetrahedron wedge model described in section 2.2.3 from the geometry. In a last pipeline stage in section 4.9, the successfully extracted wedge model data is validated against a set of necessary preconditions that can be safely assumed to be true for valid wedges. This includes a similarity analysis that is used to sort out false detections. A first result component of the wedge extraction consists of a triangle mesh with vertex and triangle-based annotations regarding the extraction process, such as assignments of geometric regions to specific wedge faces. The second result component consists of the wedge model extraction data. As the resulting components are linked together, the vertex and triangle data can extend the wedge model with properties for statistical analysis and manual visual result validation.

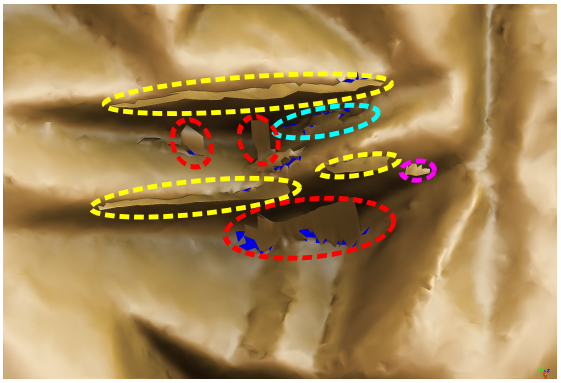
4.4 Mesh error detection

As detailed in sections 3.3 and 3.5 the 3D-scanned meshes may contain holes, non-2-manifold geometry, and overlaying surface patches resulting from issues during scan creation and scan fusion. The target is therefore to improve the handling of affected mesh areas in subsequent pipeline steps, without modifying or repairing the underlying geometry. Keeping the integrity of the original geometry is especially important, as repairing the geometry would imply to incorporate interpretation-based data and require the existence of an objective mesh repair quality function with respect to the original geometry. The former is an issue, since the trust of a philologist in the geometric accuracy of sections of a 3D-scanned dataset may play an

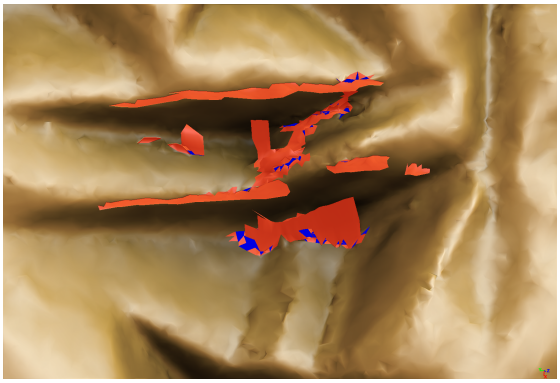
important role in the interpretation of the data. The latter is an issue, because in most cases the 3D scans in question are already the most accurate available measure of the original geometry. It has to be noted that some mesh libraries, like the here used OpenMesh, perform basic mesh repair operations by splitting up singular edges and vertices and trying to merge the involved primitives without producing non-2-manifold geometry again. Although this does modify the connectivity of the geometry and may create additional vertices, the number of faces and their position stays the same. Vertices affected by such repairs are usually marked and can be handled just as unfixed non-2-manifold geometry in the following considerations.

Mesh error detection is a preprocessing step that can help to improve the reliability of subsequent segmentation steps or circumvent geometric processing issues. In this context, a mesh error is defined as a piece of mesh geometry representing geometric circumstances that were not present in the underlying 3D-scanned real object. This includes holes, overlapping geometry, non-2-manifold geometry, and small isolated floating surface patches. While holes may actually have existed in the real geometry, the remaining mentioned geometry types are very unlikely to have occurred on the surface of a cuneiform fragment consisting of clay. In addition to that, the characteristics of a structured-light scanner, as used for the large majority of scans in this thesis, may cause surface regions located near borders of the resulting mesh to be affected by accuracy issues. This results from the fact that surface regions at the outer rim of the scanning area and those with a disadvantageous alignment to the scanner are scanned at the physical limits of the detection system. Excluding mesh borders caused by sharp features of the scanned object, this indicates a correlation between mesh borders and possible accuracy issues. It is worth mentioning that non-2-manifold geometry is usually not created by reconstructing single 3D scans, but by merging multiple scans of neighboring surface regions using the scanner software. Since most cuneiform geometry is scanned from multiple directions to guarantee complete scan surface coverage, this is an ongoing issue.

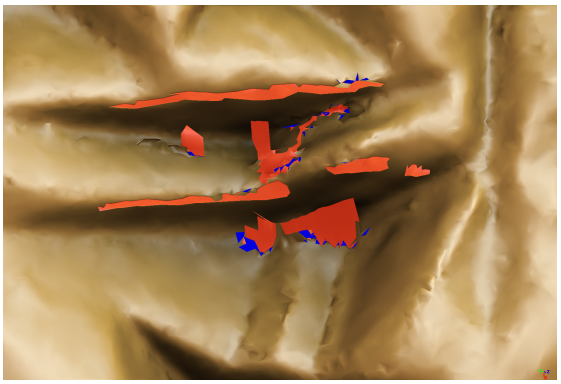
Detecting mesh regions affected by the errors described above can be used to exclude the affected geometry from further calculations or, as a more flexible alternative, to derive a reliability measure. This reliability measure can be used to weight the influence of vertices in subsequent processing steps, thus favoring presumably error-free geometry. Figure 4.3a shows a representative example of several mesh error types highlighted by colored elliptical markings. The red markings indicate overlapping geometry which creates multiple stacked surface layers on the mesh, while the geometry in the pink marked section contains a small floating surface patch. The long yellow markings contain floating stripes of geometry with sparse, often non-2-manifold connections to the mesh and the cyan marked area contains a geometry section with seemingly random non-2-manifold vertex connections. Holes in the mesh can be seen at places where the blue colored background is visible. As none of these geometric constructs may have existed on the original clay surface, they are very likely to exert a negative influence on the accuracy of a subsequent wedge segmentation and should therefore be detected in advance.



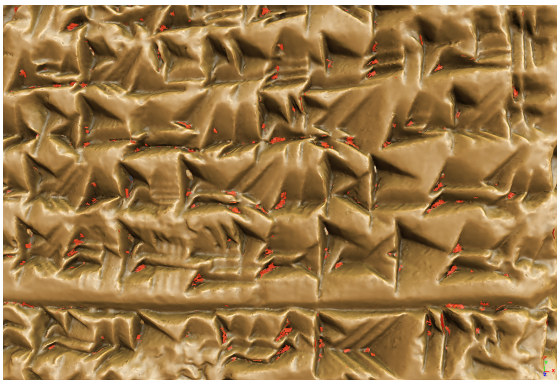
(a) Mesh error types



(b) Error detection (overfitting)



(c) Error detection (optimized)



(d) Error detection (distibution)

Figure 4.3: Mesh error detection on fragment *857/c*. Some representative error types are shown **(a)**, with elliptical markings for overlapping geometry (red), isolated surface patches (pink), non-2-manifold geometry (cyan) and floating surface patches with non-2-manifold mesh connections (yellow). Results of mesh error detection as red colored geometry are shown in **(b)**, with slight over-fitting and optimized **(c)**. A typical distribution of mesh errors on a larger text paragraph is shown in **(d)**.

The method proposed for detection of such geometric defects consists of a simple detector for non-2-manifold vertices and border faces, and a set of morphological dilation and erosion operations on the vertices of the mesh. As a starting point, all vertices on the mesh that touch the boundary or are non-2-manifold are selected. Boundary faces and non-2-manifold vertices can be efficiently detected using the employed half-edge data structure by scanning the faces for an open half-edge and scanning the vertices for more than one outgoing boundary half-edge. Faces containing open half-edges or non-2-manifold vertices form an initial face selection which is then expanded n times using a morphological dilation operator. In each step, the selection is extended to include all faces touching a face selected in a previous step. After that, the selection is retracted $n+1$ times by a morphological erosion operator. In each step, the selection is now reduced to exclude all faces that touch a non-selected face of the previous step. Performing one retraction step more than the number of expansion steps compensates the initial selection of faces at the border and near non-2-manifold vertices.

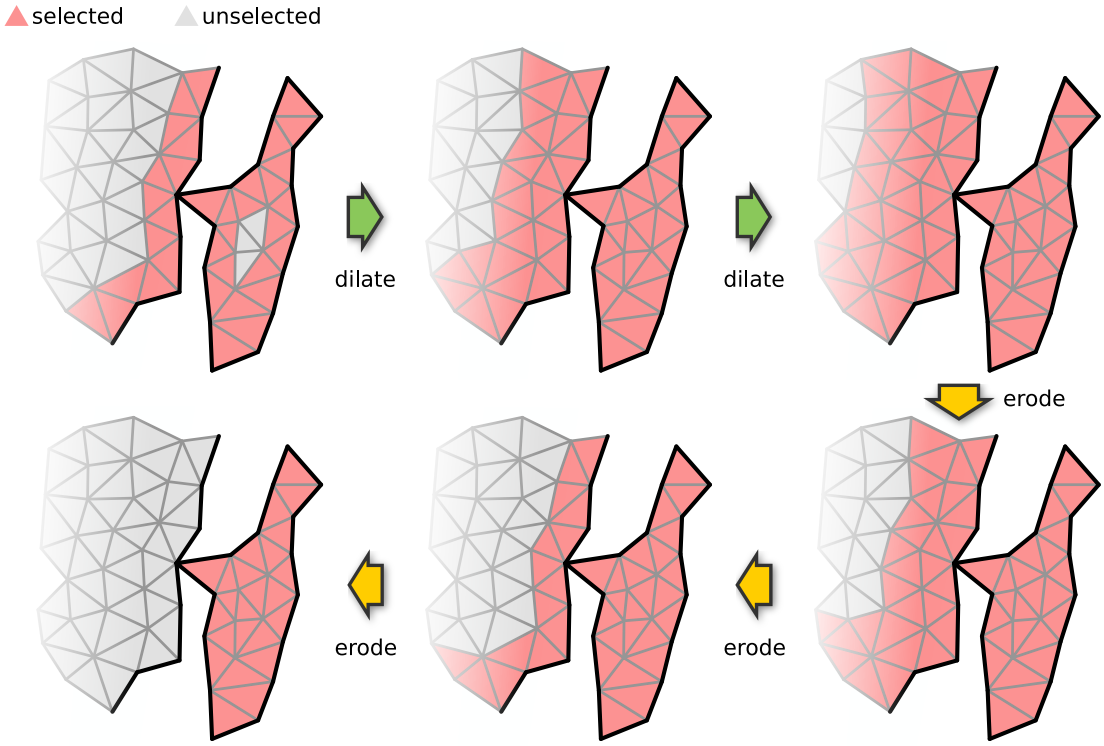


Figure 4.4: Application of the face-based mesh error detection on an exemplary mesh region for $n = 2$. Selected faces are colored in red, while non-selected faces are colored in gray. Bold edges denote boundary edges.

An application of this process for $n = 2$ to an exemplary section of a mesh is shown in figure 4.4. Starting with the initial selection of boundary faces and faces touching non-2-manifold vertices, the selection is dilated two times and subsequently eroded three times. This results in a final face selection, corresponding to the potentially error affected mesh regions.

The mechanism works by exploiting that the expanded face selection can not retract in the same manner if passing through a narrow section. As the retraction is not propagated from borders but only from non-selected faces, narrow isolated or nearly isolated triangle patches remain selected. This is also true for many constellations directly adjacent to non-2-manifold vertices, as the initial face selection may already contain faces not touching any non-selected faces. The remaining selected faces can then be labeled to be part of mesh errors. To adjust the sensitivity of the approach the number n of iterations can be adapted manually to the typical size of the error regions. To account for special mesh characteristics, the number of additional retraction iterations can be increased. Some exemplary results of the mesh error detection for $n = 2$ are shown Figure 4.3. The detection result with one additional retraction iteration in figure 4.3b shows slight over-fitting, while the result with two additional retraction iterations in figure 4.3c is more appropriate in this case. Figure 4.3d shows an exemplary distribution of detected errors on a mesh that can still be successfully processed by the subsequent wedge model extraction.

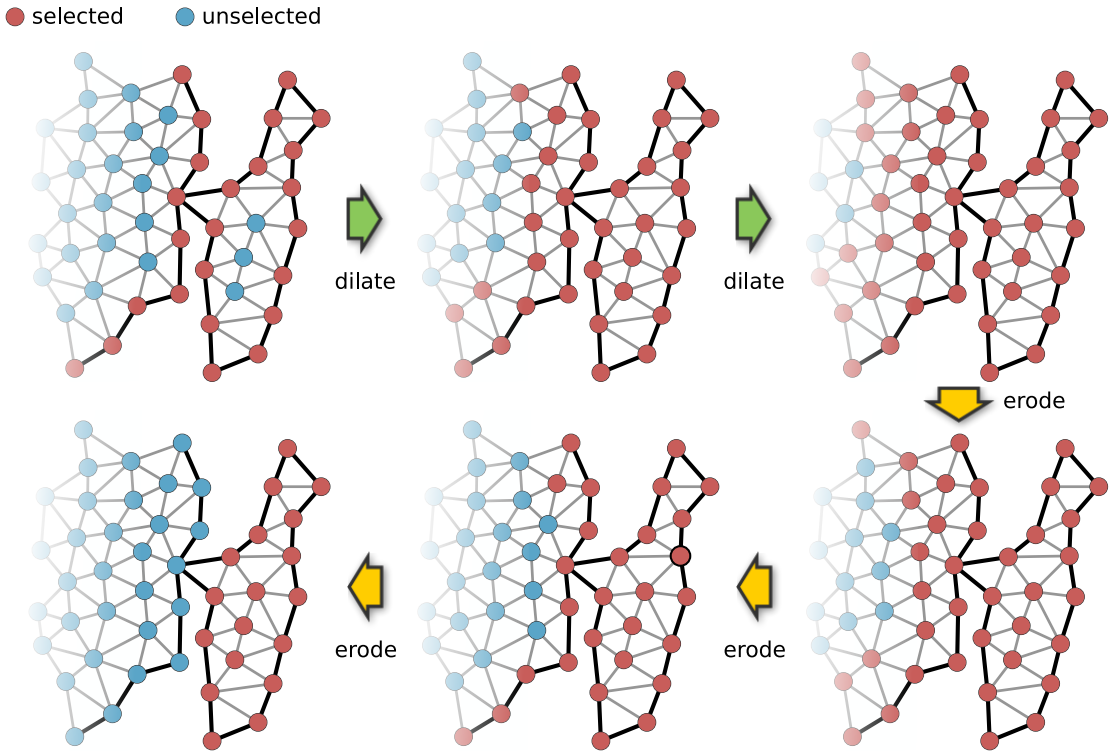


Figure 4.5: Application of the vertex-based mesh error detection on an exemplary mesh region for $n = 2$. Selected vertices are colored in red, while non-selected vertices are colored in blue. Bold edges denote boundary edges.

For a more sensitive mesh error detection, the binary labeling can be replaced by a distance-based labeling. For this, all faces are labeled with a floating-point weight $r \in [0,1]$ during erosion, which is initialized to 1. In each erosion iteration i, a value of

$$r(i) = (n+1-i)/(n+1) \tag{4.1}$$

is assigned to the faces that become deselected, where i assumes values between 1 and the number of iterations n. The remaining faces are assigned a value of $r(i) = 0$. This way, faces affected by mesh errors are assigned a value $r < 1.0$, while error free faces are assigned a value of 1. The assigned weights can then be used in subsequent processing steps as a reliability value.

For cases in which a vertex-based error detection is required, the process can be modified to operate on vertices. Starting with an initial selection of non-2-manifold and border vertices, the dilation and erosion operations are applied to the associated vertex selection. As can be seen in figure 4.5, the basic functionality is very similar to the one in figure 4.4, although the results may differ slightly. Regardless of the constructional domain, a simple conversion based on touching faces and vertices can be used to obtain an error region selection for both domains.

It is worth mentioning that the mesh error detection, described above can only handle mesh errors with a limited spatial extent. Large overlapping areas on the mesh must be handled by optimizing the individual scan fusion in the scanner software or using a more capable scanning and scan fusion method. This becomes particularly clear when considering the fact that, in the case of large overlapping geometry areas, all involved layers can be correct in the context of their respective individual scans.

4.5 Surface distance reconstruction

A useful prerequisite for surface-based extraction of individual wedges is a reconstruction of the original free writing surface based on the free writing surface on a cuneiform fragment. Here, the *free writing surface* refers to smoothed sections on the surface of a cuneiform fragment that are not covered with wedge imprints or part of fracture faces. In contrast, the *original free writing surface* of a fragment refers to the whole smoothed surface of a cuneiform fragment before wedges were placed on it. It is identical to the *writing surface* of a blank cuneiform manuscript like defined in chapter 2.2.1. The explicit differentiation is made because the cuneiform fragments considered here already contain text. Their writing surface is therefore geometrically not the same as that of a blank cuneiform manuscript. Consequently, the original free writing surface would be an excellent reference system for determining which regions on the mesh surface correspond to wedges and which do not. It can efficiently be used to reduce the amount of data to be processed in subsequent steps of the wedge extraction pipeline. To achieve this, this section will detail an efficient method for reconstructing the original free writing surface using ambient occlusion and a Poisson reconstruction in order to derive a distance field for surface extraction. The high quality distance field will also be a central prerequisite for the watershed-based wedge extraction in section 4.7.

The original free writing surface of a fragment, is considered to be mostly convex and exhibit a low and uniform curvature. Depending on the writing density and the degree of damage, its geometric representation in the 3D scan is interspersed with wedge imprints, superficial damage and fracture faces. As the latter do not include wedges by definition, the task of extracting those parts of the geometry that contain wedges is closely linked the task of reconstructing the original free writing surface. The wedge geometry can then be derived based on a distance measure between the reconstructed writing surface and the scan geometry. In addition to that, the free writing surface provides an important reference system to derive further depth-based wedge properties.

4.5.1 Ambient occlusion computation

Ambient occlusion (AO) is used in offline rendering and real-time graphics as a shading technique to visualize realistic self shadowing of 3D objects and complete scenes under diffuse lighting conditions. Ambient occlusion values on the surface of a 3D object can therefore be seen as an accessibility measure of the surface to outside light, which is usually modeled as a

uniform spherical surface of light, surrounding the affected geometry. As this lighting model closely resembles real world lighting conditions involving daylight, its use greatly enhances the human depth perception of 3D rendered scenes and objects.

A common mathematical formulation for the ambient occlusion can be derived from a directional version of the radiance equation

$$L(\boldsymbol{p},\boldsymbol{\omega}_{out}) = \int_{\omega\in\Omega} f(\boldsymbol{p},\boldsymbol{\omega}_{out},\boldsymbol{\omega})V(\boldsymbol{p},\boldsymbol{\omega})(\boldsymbol{n_p}\cdot\boldsymbol{\omega})L(\boldsymbol{p},\boldsymbol{\omega})d\omega, \tag{4.2}$$

which describes the outgoing radiance $L(\boldsymbol{p},\boldsymbol{\omega}_{out})$ at point $\boldsymbol{p}$ with surface normal $\boldsymbol{n_p}$ in direction $\boldsymbol{\omega}_{out}$ [49, 80]. It is composed of the incoming radiance $L(\boldsymbol{p},\boldsymbol{\omega})$ at point $\boldsymbol{p}$, integrated over a hemisphere Ω at $\boldsymbol{p}$ and weighted with a reflectance function f, a visibility function V and Lambert's cosine law. Here, $d\omega$ denotes the infinitesimal solid angle of the integration variable $\boldsymbol{\omega}$. The visibility function V is defined as

$$V(\boldsymbol{p},\boldsymbol{\omega}) := \begin{cases} 0 & \text{if } \boldsymbol{p} \text{ is occluded in direction } \boldsymbol{\omega} \\ 1 & \text{if } \boldsymbol{p} \text{ is visible in direction } \boldsymbol{\omega}. \end{cases} \tag{4.3}$$

Considering only the incoming radiance of a uniform environmental light source and normalizing the result to a fraction of the hemisphere, the ambient occlusion A at a surface point $\boldsymbol{p}$ can then be written as

$$A(\boldsymbol{p}) = \frac{1}{\pi}\int_{\omega\in\Omega} V(\boldsymbol{p},\boldsymbol{\omega})(\boldsymbol{n_p}\cdot\boldsymbol{\omega})d\omega. \tag{4.4}$$

It has to be noted that in this formulation a high value of A actually corresponds to a low degree of occlusion, while a low value of A corresponds to a high degree of occlusion. An advantage of this formulation is that it can be directly used as a dampening factor for incoming environmental light.

The computation of physically correct ambient occlusion on a 3D surface would generally include the global computation of an infinite number of light bounces on the objects surface. Therefore, ambient occlusion can only be approximated. To speed up computation times, two commonly employed simplifications are to omit light contributions caused by surface bounces to a certain degree and to introduce a maximum distance for geometry to be included in the occlusion computation. The second simplification is based on the assumption that the relative contribution of geometry elements to the occlusion is inversely proportional to their distance. Offline rendering techniques using Monte-Carlo or finite element methods usually compute occlusion values by sampling a hemisphere of directions, surrounding the affected surface element to determine the occluded fraction of outside lighting. The computed ambient occlusion values are then stored on the scene geometry to be multiplied with the diffuse color values of the surface before shading. As a result, geometric regions exhibiting high amounts

of ambient occlusion get darkened in the final image, producing the desired effect. Being still computationally expensive, this global approach is only suitable for offline rendering and real-time rendering of static scenes, since dynamically moving or deforming geometry would require a global recomputation of ambient occlusion. For static scenes, ambient occlusion may be computed in a preprocessing pass, stored with the geometry and rendered as needed.

Important preliminary works for an accelerated computation of ambient occlusion by deriving indirect illumination from a distributed pseudo-light-source using obscurance maps were presented 1998 by Zhukov et al. [240]. In 2004 Pharr et al. [156] introduced the term of ambient occlusion maps and a GPU-accelerated approach for a fast approximation of diffuse lighting based on precomputing view-independent occlusion information using GPU-accelerated ray-tracing. A multi-pass approach based on hierarchically approximating geometry through occlusion disks was presented by Bunnell et al. [37], which also considered multiple light bounces and scenes with dynamic geometry. A similar approach of Kontkanen et al. computed real time ambient occlusion fields for dynamic objects by approximating the occlusion contribution of objects through sphere caps [115]. A solution for ambient occlusion on volumetric medical data was given by Ruiz et al. [181] to improve depth perception on voxel datasets. While these techniques are suitable for real time rendering of dynamic geometry, the degree of complexity in modern gaming environments and high resolution 3D scan data constricts their practical usability. In addition to that, interactive dynamic environments intended for real-time rendering of complex scenes have a bias towards minimizing computation times and maximizing visual appeal, rather than maximizing physical correctness of lighting. To accommodate these needs, image-based post-processing techniques to approximate ambient occlusion have been introduced. A first attempt to approximate ambient occlusion in screen space was made by Shanmugam et al. [196] by accumulating occlusion contributions of eye-space pixels represented as micro-spheres. A more sustainable step are the so-called screen-space ambient occlusion (SSAO) methods, initially developed by Vladimir Kajalin at Crytec [146]. The basic approach includes sampling depth values on the rendered image to estimate the mutual occlusion of geometry, associated with randomly chosen local pixels. SSAO methods exhibit various advantages. They are independent of geometric scene complexity, can naturally be employed to dynamic scenes, require no special preparation steps, have a low memory footprint, and can be computed entirely on the GPU. More sophisticated variants like screen space directional occlusion (SSDO) [174] even include directional colored light bounces to approximate light bleeding. However, even when incorporating geometry normals, the resulting ambient occlusion values are view-dependent and capture only local occlusions. They are also prone to high frequency sampling noise and depth discontinuities and are not available on the rendered geometry elements, such as points and triangles. This makes SSAO primarily suitable for visualization purposes.

The use case of surface detection requires a view independent ambient occlusion on a per vertex basis. This is achieved by adapting and extending the hardware-accelerated approximation method described by Sattler et al. in 2004 [190]. The basic idea of this “outside-in” approach

is to replace the local integration of directions around a surface point to determine its ambient occlusion, with a discretization that depends on a global set of predefined light directions. To compute the ambient occlusion at a vertex $\boldsymbol{v}_i$ with normal $\boldsymbol{n}_i$ for k directional light sources $\boldsymbol{l}_j$, equation 4.4 can be modified to

$$A(\boldsymbol{v}_i, \boldsymbol{n}_i) = \frac{1}{k} \sum_{j=1}^{k} V(\boldsymbol{v}_i, \boldsymbol{l}_j)(\boldsymbol{n}_i \cdot \boldsymbol{l}_j). \tag{4.5}$$

While the required visibility tests can be efficiently hardware-accelerated using shadow-mapping techniques, the method additionally benefits from the fact that the number of required lights k is usually much smaller than the vertex count n of a 3D-scanned geometry. As a result, the naive gathering approach for the vertices can be efficiently replaced by a scattering approach for the corresponding directional lights. Evaluating the visibility per light and not per vertex significantly reduces the number of necessary rendering passes. For all combinations of vertices $\boldsymbol{v}_i$ and lights $\boldsymbol{l}_j$, Sattler et al. store the results of the occlusion tests in a visibility matrix M_{ij}. Figure 4.6 shows a simplified version of the original algorithm which extensively uses the OpenGL `ARB_OCCLUSION_QUERY` [109] extension to accelerate the computation.

```
1   create k light directions l_j
2
3   for all light directions l_j
4   {
5     camera direction = l_j
6     enable polygon offset
7     render mesh to depth texture
8     disable polygon offset
9
10    for all vertices v_i
11    {
12      issue occlusion query i to GPU
13    }
14    for all vertices v_i
15    {
16      retrieve occlusion result i form GPU
17      M_ij = visible ? n_i·l_j : 0
18    }
19  }
20
21  for all light directions l_j
22  {
23    for all vertices v_i
24    {
25      A_i += M_ij
26    }
27    A_i /= k
28  }
29
```

Figure 4.6: Pseudo-code of the original ambient occlusion computation.

At first, the algorithm generates a number of light directions using a hierarchical subdivision scheme. Then, for all light directions, the scene is orthogonally rendered into a depth texture. To avoid rounding issues during depth comparisons, this is done using a polygon offset that defines a variable depth offset for each polygon. For each depth texture, the occlusion of all vertices is determined using occlusion queries which perform a simple depth comparison against the projected vertices by rendering the vertices as a point set. The vertices are cycled two times in lines 9 and 12, due to the asynchronous nature of the occlusion queries, where the first cycle renders the vertices and the second cycle downloads the data to the CPU. After all light directions have been processed, the occlusion results for each vertex are accumulated from the visibility matrix M_{ij} and normalized. The computation is optimized for dynamic scenes by performing occlusion tests only for newly visible vertices and adaptively reducing the number of lights when in motion. As stated in the original paper, a drawback of this method is the storage of the visibility matrix on the CPU.

To adapt the method of Sattler et al. to the static surface detection use case, the optimizations for dynamic scenes are removed. As a second modification, the original hierarchical octahedron-based method for the generation of the light directions is replaced since it allows only $2+4^{s+1}$ light positions, with s being the subdivision count. As no dynamic refinement capabilities are required, a spiral distribution technique is used to generate evenly distributed points on a sphere. This allows generating uniform direction sets with an arbitrary number of directions. The chosen method described by Saff and Kuijlaars [187] works by slicing a sphere with horizontal planes into N circles of latitude, where each circle hosts one point of the final distribution. Starting with the degenerated first or last circle, the distribution points are incrementally generated by advancing to the next cut along a sphere meridian and the advancing a fixed distance on the circle. As described in [46], this longitude increment can be chosen to depend on the golden section, which further improves the longitudinal distance neighboring points. A non-optimized pseudo-code version of this method is shown in figure 4.7, where `numDirections` denotes the number of directions to generate.

```
1     float N = numDirections;
2     float inc = PI * (3.0 - sqrt(5.0));
3     float off = 2 / N;
4     for (int k = 0; k < N; k++)
5     {
6       float y = k * off - 1.0 + (off / 2.0);
7       float r = sqrt(1.0 - y*y);
8       float phi = k * inc;
9       direction[k] = Vector(cos(phi) * r, y, sin(phi) * r);
10    }
11
```

Figure 4.7: Pseudo-code of an unoptimized version of the spiral sphere distibution algorithm.

The third modification aims to store the occlusion results on the GPU, in order to diminish a major disadvantage of the original method and to further accelerate the computation. This is realized by replacing the occlusion queries with a common method used in shader-based GPGPU programming that uses multiple simultaneous render targets to increase the amount of computations per shader execution. The basic concept includes storing all necessary data in textures accessible by a fragment shader and rendering the results to a set of textures, which are bound as multiple draw buffers. As a static use case does not benefit from storing the occlusion attributions per light separately, the M_{i*} can be stored and accumulated on the GPU in form of the unnormalized A_i values. The accumulation is efficiently realized using floating-point blending, where new values can be additively blended with existing ones in the render targets. Thus, the method requires at least the OpenGL extensions `ARB_draw_buffers`, `EXT_gpu_shader4` and `ARB_texture_float` [109]. A simplified pseudo-code version of the modified method is shown in figure 4.8.

```
1   prepare textures for vertices, normals and occlusion accumulation
2   upload data to vertex and normal textures
3   create k light directions l_j with the spiral method
4
5   for all light directions l_j
6   {
7     camera direction = l_j
8     enable polygon offset
9     render mesh to depth texture
10    disable polygon offset
11
12    enable additive blending
13    enable AO computation shaders
14    disable depth test
15    render full screen quad to accumulator texture
16    enable depth test
17    disable AO computation shaders
18    disable blending
19  }
20
21  for all vertices v_i
22  {
23    download accumulated occlusion result A_i form accumulator texture on GPU
24    A_i /= k
25  }
26
```

Figure 4.8: Pseudo-code of the modified ambient occlusion computation.

For the texture preparation step, two important targets are to maximize the amount of parallelism and optimize the data access. This is realized by maximizing the number of textures that are accessed simultaneously, and by using fast access patterns for the textures. To begin with, the texture size t is determined depending on the maximum number d of supported simultaneous draw buffers and the number of vertices n. For old hardware, a GPU-convenient texture size would imply power-of-two texture sizes. Since this is usually not an issue with post 2005 hardware, arbitrary texture sizes may be used within the supported

limits. A favorable restriction is the usage of square textures to keep the texture sizes from reaching the hardware-induced limits. For the sake of improving memory alignment, an additional size restriction to multiples of 2^l can be employed. Thus, a texture size t of

$$t = \left\lceil \frac{\left\lceil \sqrt{\frac{n}{d}} \right\rceil}{2^l} \right\rceil \cdot 2^l \tag{4.6}$$

can be used to create d $t \times t$-sized texture layers for each, vertices, normals, and occlusion results, which can be accessed using identical access patterns. Computing the texture size as described above, usually creates an amount of unused texture space of size $d \cdot t^2 - n$. For low memory environments, this space could be minimized by employing rectangular instead of square textures. A set of two suitable texture side lengths t_a and t_b can be obtained solving

$$\begin{array}{rrcl} \min & f_{g_1,g_2}(t_a,t_b) & = & g_1(t_a \cdot t_b - n) + g_2(t_a + t_b) \\ \text{s.t.} & t_a \cdot t_b & \geq & n. \end{array} \tag{4.7}$$

The weights g_1 and g_2 control a trade-off between unused space minimization and equally sized side lengths of the texture. A practical solution could be computed with the help of Fermat's factorization method [173], as it is capable of determining a factorization of n with factors close to $\sqrt{n}$. As this method only exhibits good running times in cases, where factors close to $\sqrt{n}$ exist and low amounts of memory were not an issue for the test environments used in this thesis, square texture sizes were used. The most common maximum texture size for currently used graphics hardware is 16384 [232], while the number of supported draw buffers is almost always 8 [232]. This results in a total number of 2^{31} addressable elements. As each element requires the storage of 7 floating-point values for vertex position, normal and occlusion accumulation this would result, in a total of over 481 GB of data assuming 32Bit floating-point accuracy or over 240GB for 16Bit accuracy. This is much larger than the amount of graphics memory on current hardware and therefore considered as a sufficient number of elements.

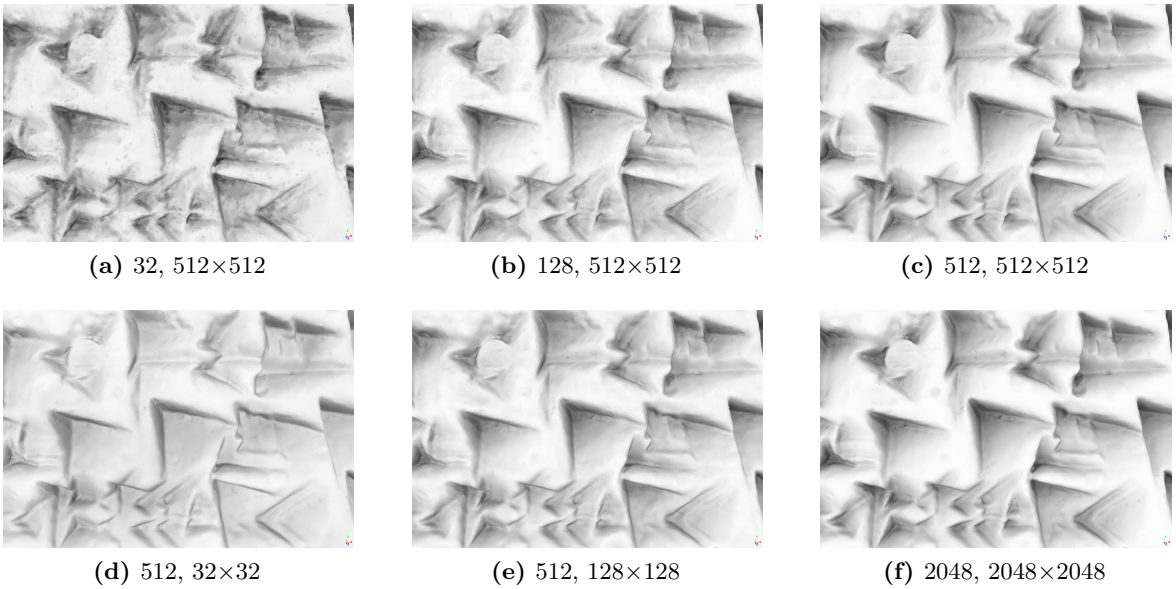
After the textures have been created, the vertex and normal data are uploaded to the GPU using Pixel Buffer Objects (PBOs) [111], which are an OpenGL mechanism for asynchronous pixel transfer operations. For this method a PBO is created on the GPU and mapped into client memory. The data is then asynchronously transferred to the buffer object. From there, the data is copied to the target texture and the PBO is deleted afterwards. Compared to a conventional texture upload using `glTexSubImage` this resulted in speedups of around 68% on a test system with a Core i7-4790K and a Geforce GTX 780 through overlapping the data upload and the ambient occlusion computation. As PBOs do not provide increased transfer speed, but only improve asynchronous behavior, the download of the accumulated occlusion results does not benefit from using PBOs, as the downloaded data is accessed immediately. Instead, a performance decrease of up to 35% was observed as opposed to transferring the

data via `glReadPixels` or `glGetTexImage`, which results from the creation and copy overhead caused by the PBO.

The occlusion computation is executed for each light direction and includes a depth rendering pass, like in the original method, and an occlusion pass. As the occlusion in the modified method is intended to be computed in parallel via a fragment shader, the shader must be called in a way to process all vertices. This is realized by binding the vertex and normal data textures, attaching the d accumulator textures to an Frame Buffer Object (FBO) [110] as individual color attachments, setting the viewport size to match the texture size t, and rendering an arbitrary geometric object occupying the full screen. The rasterization of the object then results in the fragment shader to be executed once for every pixel of the $t \times t$-sized FBO textures. By processing the same pixel of all d color attachments in the same shader call, every input vertex is processed.

The actual occlusion test in the shader is realized by fetching one vertex position and normal from the data textures, transforming the vertex into clip space and using it to call the function `textureProj`, with the prerendered depth texture as a shadow map. The binary result represents an approximation if the respective vertex is visible from that direction. As the depth texture is usually much smaller than the data textures, the depth texture resolution is the most prominent factor limiting the approximation accuracy of directional visibility. If a vertex is identified as being visible, the occlusion contribution is computed according to Lambert's cosine law and written to the respective layer of the output texture. This leads to a result accumulation, due to activated additive blending. As can be seen in figure 4.8, the normalization of the occlusion results using the number of view directions is performed after downloading the data from the GPU.

The visual quality of the resulting ambient occlusion values is primarily determined by the number of view directions and the depth texture resolution. Figure 4.9 shows a selection of occlusion results with varying parameterization. While figures 4.9a to 4.9c indicate a dependency of occlusion noise and the number of views, figures 4.9d, 4.9e and 4.9c show that the depth texture resolution exerts a strong influence on the adaptiveness of detail representation. Low numbers of view directions tend to produce noisy occlusion values because they retain characteristics of the individual shadow casting processes. Low depth texture resolutions on the other hand cause large regions of vertices to be compared against the same depth value, which impairs the representation of details. As increasing either parameters negatively affects the running time, a trade-off against approximation quality has to be determined. Seeing that a further increase did not result in a significantly better approximation, cf. figures 4.9c and 4.9f, values of 512 for both the number of view directions and the depth texture resolution have been used. With these values, the average running time for ambient occlusion computation was approximately 23% of the total time required for loading a mesh file in ply format. For an average sized mesh with 3.4 million vertices, this



(a) 32, 512×512 **(b)** 128, 512×512 **(c)** 512, 512×512

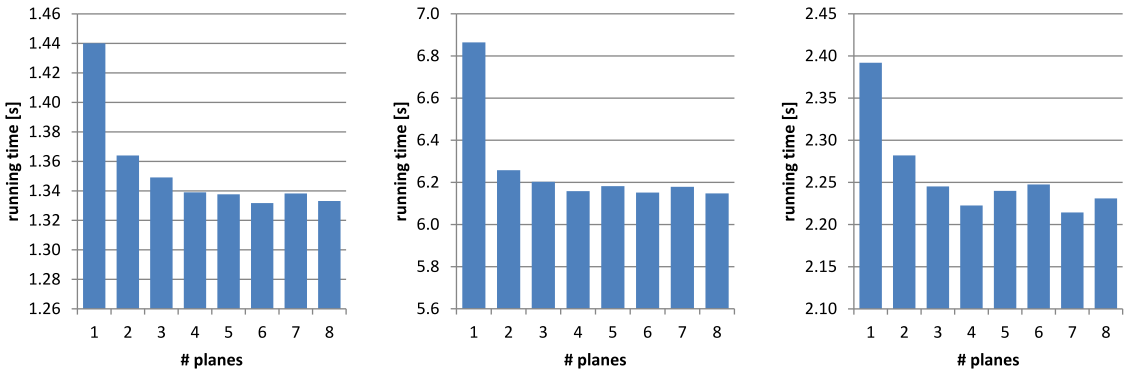
(d) 512, 32×32 **(e)** 512, 128×128 **(f)** 2048, 2048×2048

Figure 4.9: Visual quality of ambient occlusion results on fragment *Bo2743* for six different combinations of numbers of view directions and depth texture resolutions.

was equivalent to a computation time of 3.2 seconds on the test system mentioned above, including data upload and download.

For estimating the order of time savings through occlusion accumulation on the GPU, an exemplary occlusion computation on the same 3.4 million vertices mesh is considered. Assuming 512 light directions, a measured average upload time of 0.122 seconds and a download time of 0.045 seconds, naively downloading the computed occlusion data in each occlusion pass alone would cost in total over 23 seconds. This is a multiple of the achieved total ambient occlusion computation time of 3.2 seconds for result accumulation on the GPU. Although these timings could be improved by interleaving computation and download activities, but this would also require multiple result buffers on the GPU and therefore cause an increased memory footprint.

Regarding the performance increase achieved by computing multiple occlusion results per shader call to increase the shader workload, occlusion computation performance was measured on multiple meshes while varying the number of used texture planes between 1 and 8. Some exemplary measurement series on a Geforce GTX 780, a Quadro K1100M and a Geforce GTX 960 are shown in figure 4.10, with the individual measurement values per texture plane configuration being averaged over 10 samples each. Like before, the measurements were taken for computing 512 light directions and a depth texture size of 512×512. The vertical scale of the subfigures is non-uniform to ensure the plane usage dependent speedup is visible for all tested GPUs.



(a) Geforce GTX 780 **(b)** Quadro K1100M **(c)** Geforce GTX 960

Figure 4.10: Exemplary ambient occlusion computation times measured in seconds on a dataset with approximately 1.6 million vertices for 1 to 8 texture planes, excluding data upload and download times.



(a) **(b)**

Figure 4.11: Ambient occlusion discontimuities in cuneiform wedges with a significant deviation from a depth propotional field caused by variations in surface curvature **(a)** and overlapping geometry **(b)**.

The results show a performance increase when increasing the number of texture planes, quickly converging to a maximum performance at around 4 planes, with the biggest performance increase between 1 and 2 texture planes. This behavior was consistently observed on all measured GPUs. A further consistent performance increase when using more texture planes could not be observed. Instead, the performance variations from 4 to 8 planes seemed to depend on the specific combination of GPU and the processed mesh. This suggests that the shader workload gets saturated at about 4 texture planes for each of the tested GPUs.

The occlusion values obtained at this point can not be used directly as a distance field approximation. As figure 4.11a shows, curvature discontinuities in wedge grooves cause the occlusion values to deviate from a surface distance proportional term. In this example, parts of

the side faces near the inner edges are steeper than the remaining side face area and therefore have a higher degree of occlusion. In addition to that, surface areas with overlapping geometry, as shown in figure 4.11b, exhibit strong distortion of the occlusion values. All areas affected by these surface peculiarities usually show darker occlusion values that are equivalent to a higher degree of occlusion. This does, however, not influence the ability of the occlusion values to indicate free surface areas that are characterized by a low degree of occlusion. The ambient occlusion is therefore used as a detector to identify a set of mesh vertices, corresponding to the regions of the free writing surface, which is not covered by cuneiform signs. This set of vertices will be used in the next section as a base to approximate the original free writing surface as defined at the beginning of section 4.5. An initially assumable downside of this method is that it does not handle fracture faces separately. Fracture faces may also contain areas of low occlusion, but are not part of the free writing surface per definition. This does not constitute a disadvantage here, because a differentiation between wedges on the free writing surface and grooves on fracture faces can be made after wedge extraction. As will be shown later, postponing this decision provides a better basis for decision-making.

Obtaining a sufficiently accurate classification of the free writing surface can be realized by thresholding the computed ambient occlusion values. As shown in figure 4.12b, thresholds around 0.96 provide a good separation between the free writing surface and the red marked wedge areas. At the same time, large parts of the fracture faces and damaged writing surface are classified as non-writing surface as well. This is not an issue because it does not compromise the free writing surface classification, as these areas are located below the original free writing surface. The determined threshold is applicable on the vast majority of examined cuneiform fragments, although a minority of tablets exists where a bumpy writing surface or very subtle wedges required minor tuning.

4.5.2 Surface reconstruction

Since a reconstruction of the surface distance requires the free writing surface as a reference system, this section will describe how to reconstruct the free writing surface from the 3D-scanned, ambient occlusion annotated point set.

In view of the 3D scan inherent data quality, which may feature non-uniform sampling, noisy data, and outliers as a result of the scan fusion and reconstruction process, or even misaligned scan borders, a method is required that can robustly handle those types of issues. An additional important property would be the handling of holes, either in the form of missing scan areas, or, like in this case, in the form of the removed non-surface areas. A suitable method should generate a close approximation of the available surface geometry, while at the same time smoothly extrapolating the holes in a depth level consistent way. The resulting surface should also be 2-manifold and resemble the topology of the original free writing surface with an expected inherent degree of global smoothness. Depending on the 3D scan properties this does not necessarily exclude borders or holes.



(a) (b)

Figure 4.12: Surface filtering using thresholded ambient occlusion for a low threshold of 0.85 **(a)** and a near optimal value of 0.96 **(b)** with the non-surface areas marked in red.

Considering these requirements and the fact that the original scanned artifacts are almost exclusively closed surfaces with a smooth surface motivates the use of an approximation method locally optimizing for smoothness. Most surface reconstruction methods based on an optimization of smoothness can be assigned to a few high level classes [17], two of them containing methods optimizing the local smoothness and the global smoothness. Locally optimizing methods generate a surface approximation from evaluating a local neighborhood of samples. This makes local methods like moving least squares (MLS) [125] highly susceptible to strong deviations in sampling density of the underlying point set. As these kinds of deviations are expected on the surface filtered cuneiform point sets, a global smoothness optimizing method is used.

One possible approach for global optimization is to describe the object to be approximated by means of an indicator function. Applied to 3D objects, an indicator function $I_O : \mathbb{R}^3 \to \{0,1\}$ defines a subset of $\mathbb{R}^3$ as belonging to an object O with

$$I_O(\boldsymbol{x}) := \begin{cases} 1 & \text{if } \boldsymbol{x} \in O \\ 0 & \text{if } \boldsymbol{x} \notin O \end{cases}, \boldsymbol{x} \in \mathbb{R}^3. \tag{4.8}$$

The goal is then to reconstruct the surface of the boundary layer, which can be accomplished by using gradient-domain techniques. As described by Kazhdan et al. [106], one possibility to realize this is determining an implicit function $\chi : \mathbb{R}^3 \to \mathbb{R}$ to represent the indicator function and extracting the corresponding iso-surface. In contrast to I_O, χ can be defined as $\frac{1}{2}$ inside O and $-\frac{1}{2}$ outside of O to simplify the derivations. Considering the surface normals are given

in the form of a vector field $\vec{\boldsymbol{N}}$, χ can be determined by exploiting that the direction of its gradient must be aligned with the direction of $\vec{\boldsymbol{N}}$. This yields the minimization problem

$$\underset{\chi}{\operatorname{argmin}} \int \|\nabla\chi(\boldsymbol{x}) - \vec{\boldsymbol{N}}(\boldsymbol{x})\|_2^2 d\boldsymbol{x}, \tag{4.9}$$

where $\vec{\boldsymbol{N}}(\boldsymbol{x})$ denotes the normal at position $\boldsymbol{x}$. Its solution can be expressed as a Poisson problem and can be efficiently computed using the Poisson surface reconstruction as described by Kazhdan et al. [105, 106]. This method solves the minimization problem by transforming the Poisson equation

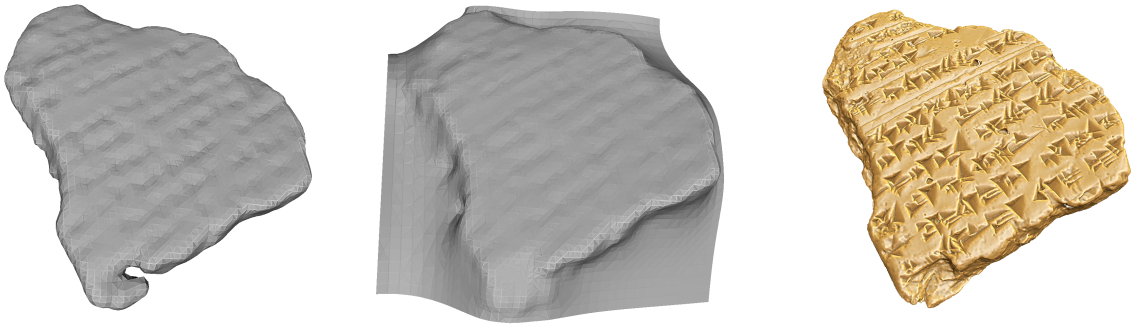
$$\Delta\chi = \nabla \cdot \vec{\boldsymbol{N}} \tag{4.10}$$

into a finite-dimensional representation using the Galerkin formulation with B-Spline basis functions $B_i : \mathbb{R} \to \mathbb{R}^3$. The solution can then be expressed as a weighted sum of k basis functions B_i with compact support radius as

$$\chi(\boldsymbol{x}) = \sum_{i=1}^{k} w_i B_i(\boldsymbol{x}). \tag{4.11}$$

The required discretization of the resulting linear system is realized by adapting an octree to the input samples of the underlying point set. Then, the supporting basis functions are associated with the resulting octree nodes, adopting the respective scale and position of the nodes. This sampling strategy adaptively couples the resolution of the solution representation to the local detail of the point set to be reconstructed. The solution for the linear system is then computed using a multigrid approach.

Solving the linear system requires a fully defined system and thus, a suitable boundary condition that defines the function values at the borders. Two possible watertight boundary conditions proposed by Kazhdan et al. are the Dirichlet and the Neumann boundary conditions [79]. While using the Dirichlet condition sets the boundaries to a value of $-\frac{1}{2}$ and forces the reconstructed surface to close off before reaching the boundary, with the Neumann condition, the boundary values are forced to zero. This effectively draws the surface to extend to and close off with the boundary. For the use case of 3D-scanned cuneiform fragments both boundary conditions may yield advantageous results depending on the scan characteristics. Predominantly complete scans of compact objects, with comparatively small holes, benefit from Dirichlet boundary conditions, as the reconstruction will create a closed, watertight object. On the other hand, objects which were scanned only from one direction and exhibit a good bounding-box coverage, may be better reconstructed using Neumann boundary conditions, extrapolating to the borders of the bounding box. The effect of different boundary conditions on the reconstruction result is shown in figure 4.13.



(a) (b) (c)

Figure 4.13: Comparison of the Dirichlet **(a)** and Neumann **(b)** boundary conditions for a scan of cuneiform fragment *Bo532* **(c)** with a non-scanned reverse side.

Once an approximation $\boldsymbol{\chi}$ for the indicator function is found, the desired surface can be extracted from its zero-level-set. To account for approximation errors, the average function value is subtracted from the individual input samples [105], which also decouples the extracted iso-surface from scaling $\boldsymbol{\chi}$. A polygonal representation is then extracted using an octree adapted version of the marching cubes algorithm [129, 230] including some modifications to correctly handle zero-crossings with respect to the finest octree subdivision levels.

This approach, however, exhibits the disadvantage that it tends to produce an over-smoothing of the data. Although, for free writing surface reconstruction, the required approximation is intended to create a very smooth representation, issues might occur at sharp fracture faces. To address this, Kazhdan et al. suggested the screened Poisson surface reconstruction [105] which changes the formulation of the minimization problem to

$$\underset{\chi}{\operatorname{argmin}} \int \|\nabla \boldsymbol{\chi}(\boldsymbol{x}) - \mathbf{N}(\boldsymbol{x})\|_2^2 d\boldsymbol{x} + \lambda \sum_{\boldsymbol{p_i} \in \boldsymbol{P}} \boldsymbol{\chi}^2(\boldsymbol{p_i}), \tag{4.12}$$

by additionally incorporating the original point cloud as positional constraints. The points $\boldsymbol{p_i}$ are weighted by a factor λ that controls the strength of the constraint and forces the extracted surface to follow the samples of original point set more closely. At this point, it is also possible to incorporate vertex weights from the mesh error detection described in section 4.4 to reduce the influence of potentially error affected regions on the approximation.

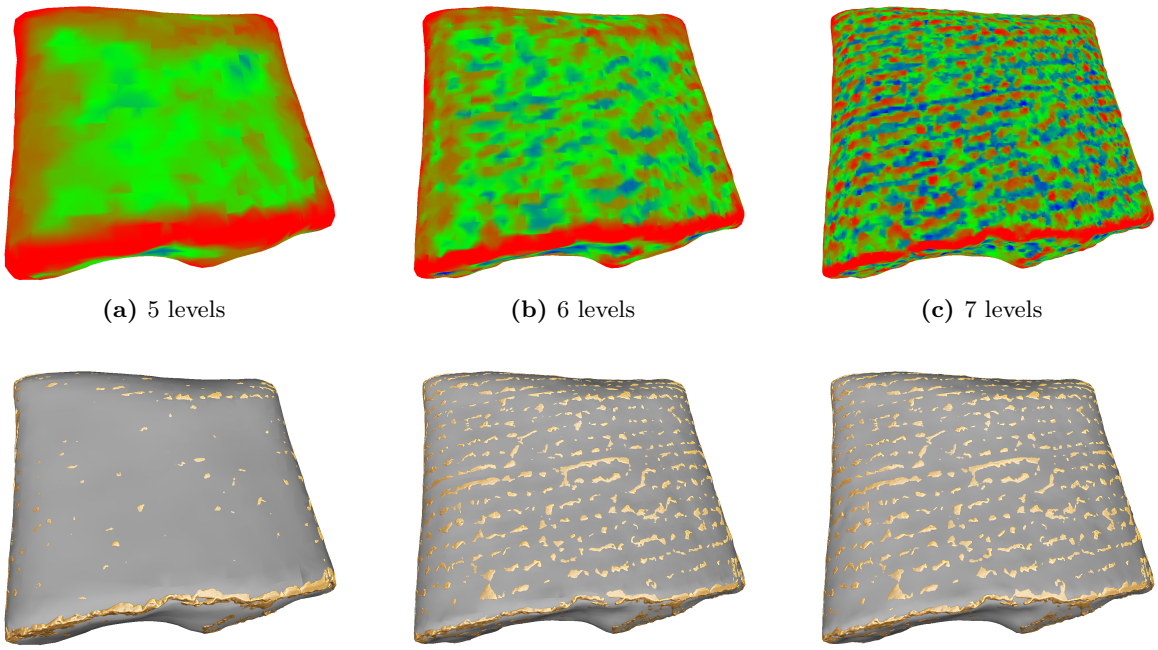
The parameter set for the screened Poisson surface reconstruction includes a minimum and maximum level l_{min} and l_{max} for the sampling octree, the above mentioned point weight λ, a sample count per node, and a scale factor for the bounding-box that controls the size of the octree root. Additional two parameters control the accuracy of the solver, specifically the amount of solver divisions and the amount of iso-value divisions. Taking into account that the Poisson reconstruction processes the ambient occlusion filtered point set, the most important parameters are the ambient occlusion threshold and the maximum octree level l_{max}. While the

first parameter determines the influence of script geometry on the reconstruction, the second parameter controls the maximum possible level of detail and thus the level of smoothness.

The actual use case requires to employ a low resolution reconstruction to create a smooth approximation of the free writing surface. Too high octree subdivision levels would cause the reconstruction to unintentionally follow the wedge imprints, while too low subdivision levels would create an over-smoothed reconstruction. The effects of different reconstruction resolutions are shown in figure 4.14. A low resolution for 5 octree levels, depicted in figures 4.14a and 4.14d, shows a smooth surface, but at the same time fracture faces become very smooth and most of the reconstructed surface is located above the intended original writing surface. On the other hand, a slightly too high resolution, as can be seen in figures 4.14c and 4.14f, shows a good surface approximation but at the same time the line structures of the cuneiform script already become visible on the reconstructed surface. The resolution chosen for figures 4.14b and 4.14e is an optimal choice in this case as it combines good surface approximation with a reasonably smooth surface, where nearly all wedge related structures have been smoothed out.

In practice, maximum octree levels of 6 to 7 have been found to generate the most usable results for scans using a millimeter scale. The results do, however, depend on a suitable relation of reconstruction resolution and the actual size of the wedge features in the underlying scan. At this point, it is important to also consider the overall size of the scans as it determines the size of the bounding box that is acting as a root node for the octree. Scans of larger objects would therefore theoretically require more octree levels to adapt the resolution to the size of the wedges. This would be an issue since the size of wedges is not known a priori. Negative effects of using a constant value for the maximum number of octree levels are, however, only observable to a very limited extent. This results from the intrinsic data characteristics of many cuneiform tablets. Small tablets or tablet fragments may exhibit a pronounced round or uneven shape, whereas bigger cuneiform documents often exhibit a rather even, flat or cylindrical shape. Therefore, the latter are comparatively insusceptible to low reconstruction resolutions.

When looking at the representative low resolution cuneiform tablet reconstructions depicted in figure 4.14, it becomes apparent that the surface of the reconstructed mesh is mostly located at or slightly above the surface of the original mesh. This is the intended behavior, caused by the ambient occlusion surface pre-filtering. To illustrate the necessity of the pre-filtering step, figure 4.15 shows a direct comparison of a Poisson approximation, here depicted as a gray surface, constructed without and with ambient occlusion filtering. In the unfiltered case in figure 4.15a, the gray-colored approximation is shifted significantly below the writing surface because the indicator function is influenced by the whole wedge geometry. This also causes the surface to exhibit visible bumps at areas with wedge imprints, as can be seen in the curvature visualizations in figure 4.15c when compared to figure 4.15d. Thus, the pre-filtering step improves both the surface position and smoothness.



(a) 5 levels **(b)** 6 levels **(c)** 7 levels

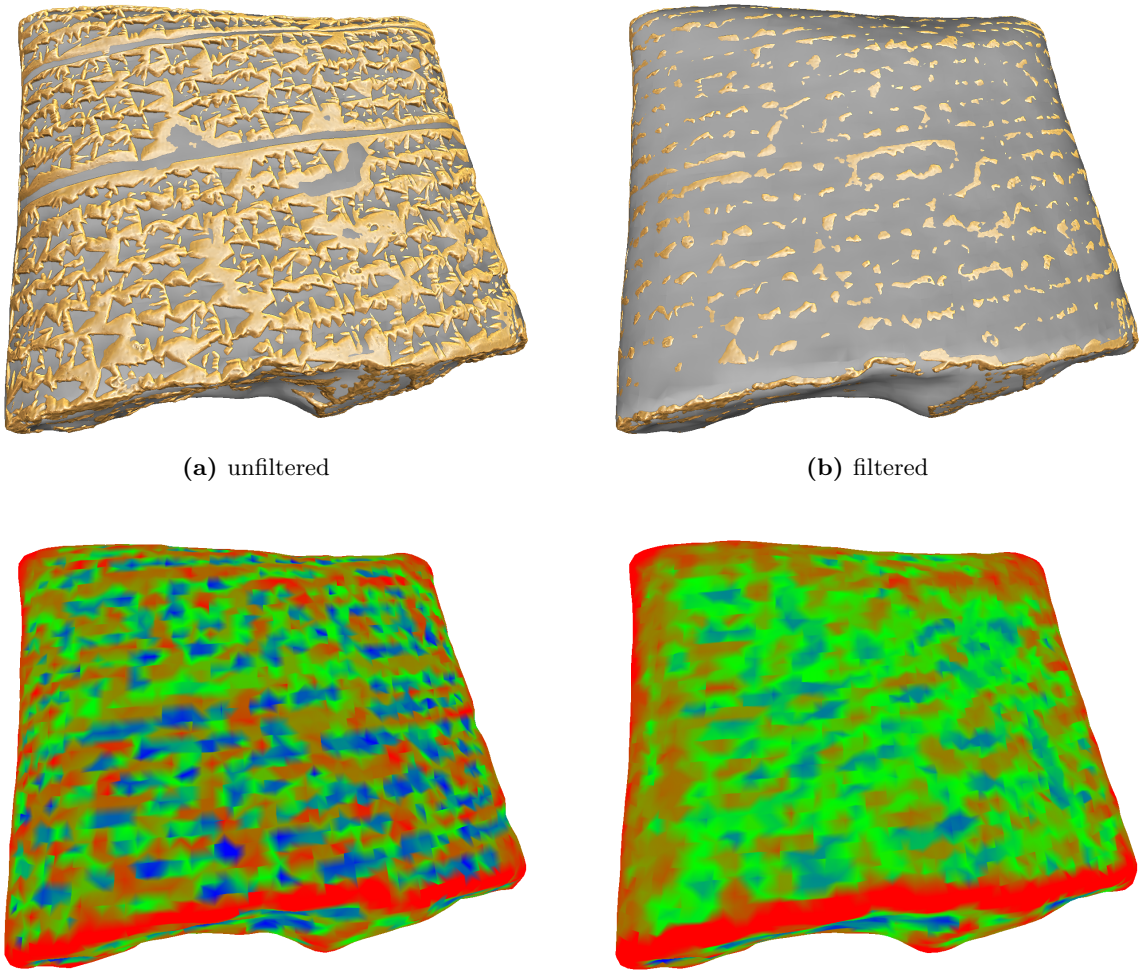
(d) 5 levels & mesh **(e)** 6 levels & mesh **(f)** 7 levels & mesh

Figure 4.14: Comparison of the Poisson reconstruction quality for three different maximum octree levels. The top row shows a visualization with color-coded mean curvature, while the bottom row shows the reconstructed mesh (gray) overlayed with the original mesh.

An issue with the Poisson reconstructed meshes are discontinuities in the density of the resulting mesh, resulting from its construction by the modified marching cubes algorithm. As can be seen in figure 4.16a, these discontinuities also lead to characteristic block artifacts in the discrete curvatures, originating from large differences in triangle sizes. This is an unwanted effect that may be an issue during distance field computation, as will be shown later. To counteract these artifacts, an isotropic remeshing is applied to the result of the Poisson reconstruction. Since water-tightness is a basic property of Poisson reconstructed meshes [105], this can be expected to not compromise topological mesh quality. The chosen method, as described by Botsch et al. [32], works based on a target edge length l by iteratively applying the following series of simple local mesh operations:

1. Split all edges longer then $\frac{4}{3}l$.
2. Collapse all edges shorter than $\frac{4}{5}l$.
3. Optimize vertex valences via edges flips.
4. Optimize vertex distribution on surface by tangential smoothing.

Both the edge split and collapse operations are executed with respect to the edge midpoints. While the optimization target of step 3 is to minimize the deviation from valence 6 or from

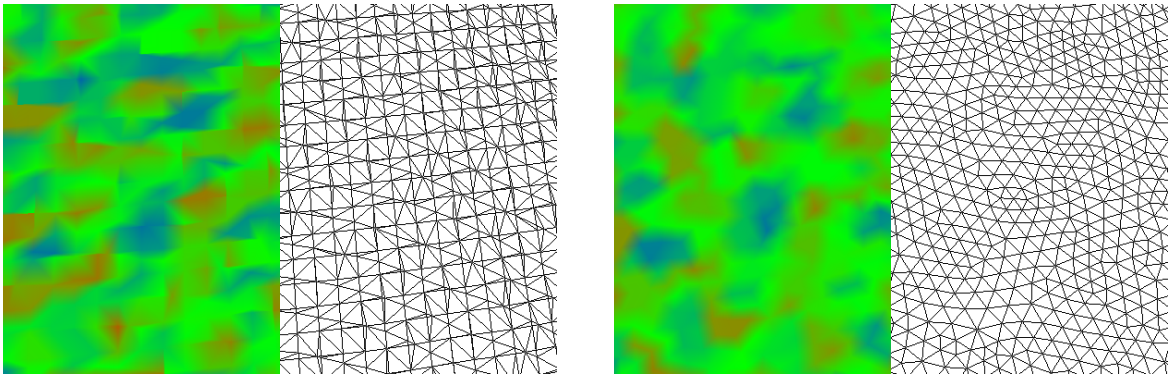


(a) unfiltered **(b)** filtered

(c) unfiltered **(d)** filtered

Figure 4.15: Influence of ambient-occlusion-based surface filtering on the Poisson reconstruction of a cuneiform fragment regarding surface approximation (top row) and mean curvature (bottom row). A reconstruction of an unfiltered point cloud is shown in **(a)** and **(c)** as opposed to the filtered version in **(b)** and **(d)**, with the reconstructed mesh being colored gray (top row) and with color-coded mean curvature (bottom row).

valence 4 on boundaries, step 4 aims to prevent unintentional nesting of vertices under special circumstances. A repeated application of these steps to a mesh creates triangulations with edge lengths close to l and a valence close to 6. As the parameterization of this algorithm is based only on the target edge length, this parameter has to be carefully chosen. In view of the target to optimize the Poisson mesh structure without unnecessarily raising the vertex count or smoothing out details, the average edge length of the reconstructed mesh is used as the target edge length. Exemplary results in figure 4.16 clearly show the improved curvature representation and mesh structure. Aside from the positive effect of the remeshing to the smoothness of the curvature values, the resulting regular structure of the mesh additionally



(a) non-remeshed **(b)** remeshed

Figure 4.16: Influence of isotropic remeshing on a Poisson reconstructed mesh. Both curvature and mesh are shown for a non-remeshed **(a)** and a isotropically remeshed **(b)** section of a poisson reconstructed cuneiform fragment.

yields the advantage of uniform vertex distances, which will be exploited during distance field computation in the following section.

4.5.3 Distance computation

In this section, the reconstruction of the original free writing surface will be used to derive a robust, high quality 3D distance field as a prerequisite to the watershed method described in section 4.7. This distance field will be defined with respect to the Poisson reconstructed mesh and needs to be evaluated solely at the vertex positions of the original mesh. Therefore, a local distance measure has to be computed between the two meshes, as opposed a global mesh distance measure, like the Hausdorff distance [84]. The latter could, however, be used to measure the approximation quality of the Poisson surface reconstruction. A second important property of the desired distance field emerges from the need to distinguish elements below and above the reconstructed writing surface. As a consequence a signed distance field is required.

Most methods for computing accurate signed 3D distance fields for triangle meshes require the underlying meshes to fulfill several preconditions, such as being closed orientable 2-manifold objects [101]. However, the necessity of most of these properties can be bypassed by compromising on the accuracy of the resulting distance field. Applicable methods can create and operate on an approximation of the underlying mesh data that exhibits the required properties. The distance field is then only accurate with respect to the approximation, like in the distance field method for arbitrary polygon soups proposed by Xu et al. [235].

In the given situation, the original scanned mesh can not be assumed to be closed or 2-manifold. The Poisson reconstruction of the free writing surface, however, inherits the properties of being 2-manifold and watertight by construction [106]. For the latter to be true with Neumann boundary conditions, the bounding-box as the construction domain has to be taken into account. This also allows to define a closed geometry based on the property of water tightness,

and consequently to compute a sign of the distance field. A Poisson reconstructed mesh is therefore a good candidate for a distance field computation.

Based on the 3D-scanned mesh and the Poisson reconstruction of the free writing surface, the goal of computing a writing surface distance field can be simplified to computing the distance of all vertices $\boldsymbol{v}_i$ in the 3D-scanned mesh to the reconstructed mesh. This distance is defined as the distance between a vertex $\boldsymbol{v}_i$ and its nearest point on the reconstructed mesh, which can be positioned anywhere on its surface. Thus, the problem can be partitioned into solving a nearest-point-on-triangle component and determining the nearest triangle.

The nearest point on a triangle with respect to an arbitrary point $\boldsymbol{p}$ can be determined by projecting $\boldsymbol{p}$ perpendicularly onto the plane defined by the triangle, and continuing with a simple distinction of cases. If the projected point lies inside the triangle, the desired distance equals the distance of $\boldsymbol{p}$ from the triangle plane. If the projected point lies outside the triangle, the distance of $\boldsymbol{p}$ and the triangle equals the shortest distance to of its three boundary line segments. A computationally more efficient way to compute this includes determining which of the triangle's so-called "Voronoi feature regions" the projection of $\boldsymbol{p}$ lies in [60]. The projective space of $\boldsymbol{p}$ in the triangle plane is divided into 7 such regions that can be classified into 3 types. The projection of $\boldsymbol{p}$ may be anywhere inside the triangle or outside the triangle and either closest to one edge or closest to a corner of the triangle. In the first case, the desired distance equals the distance of $\boldsymbol{p}$ from the triangle plane while the remaining two cases can be handled by computing a point-line or a point-point distance.

With this, only the issue of retrieving the closest triangle remains. Assuming n is the vertex count on the original mesh, which equals the amount of closest distances to be computed and m is the amount of triangles on the reconstructed mesh, then the trivial solution would require $O(n \cdot m)$ triangle distance computations. Using hierarchical organization structures for the triangles, like k-d trees [16] or orthogonal space partitioning trees, this can be reduced to not depend directly on m but on the resulting approximately logarithmic tree depth $\log(m)$. A k-d tree can be constructed for the triangle set of the reconstructed mesh by starting with the meshes' axis aligned bounding-box and iteratively splitting the bounding-boxes at alternating axis producing equally sized triangle subsets. This aims at creating a balanced tree. An alternative subdivision strategy includes splitting the tree node associated bounding-boxes at the mid point of the longest axis to most quickly cut down on the spatial extent of the tree nodes. Special attention has to be paid to the recomputation of the bounding boxes after performing the split. Each bounding box must be extended to contain all triangles that are touching its volume. This leads to an overlap of the two sub-boxes, which can negatively affect the running time. Since this overlap depends on the actual triangle distribution, determining a precise theoretical running time is a complex task. However, for cuneiform meshes with the usual dense and compact triangle distributions, this overlap is comparatively small. A more detailed analysis of running times associated with kd-trees is given by Wald et al. [228]. The constructed tree can be iteratively searched for a nearest triangle to an input point of the

original mesh. A comparison against all triangles is avoided by excluding whole tree branches based on the respective previously computed shortest distance.

Since the distance computation is intended to be used in a real time environment for segmentation, long computation times would be an issue. Performance measurements for an average sized mesh of 3.3 millon vertices and a corresponding Poisson reconstruction consisting of 86790 faces, yields 41.63 seconds on a Intel Core-i7 4790K, only considering the tree-accelerated distance computation. This results from a large number of point to triangle distance calculations for each nearest-point-on-mesh computation.

A good starting point for improving the performance would therefore be to significantly reduce the amount of required triangle distance calculations. This is realized by exploiting that the Poisson reconstructed mesh follows the original mesh closely and exhibits a uniform vertex distribution after being processed with the isotropic remeshing algorithm. The k-d tree for organizing the triangles is therefore combined with a second spatial partitioning approach to inexpensively compute a good early drop out distance for speeding up the tree-based search. The method chosen at this point is called spatial hashing [212] and usually employed for collision detection or neighborhood computations in particle systems in order to search for neighbor elements within a specific radius.

Spatial hashing processes an input point set by assigning the points to a sparse, grid-based partition (voxelization) of the input domain with freely selectable grid cell size. This is realized by computing a cell index with a spatial hash function, using the 3D coordinates of the cell as an input. Depending on the grid size, multiple points may be assigned to one cell and only non-empty cells need to be stored. Optimized storage of the cell assignments can be achieved with perfect spatial hashing, as described by Lefebvre et al. [122]. The cell assignments are stored by inserting the input point indices into a cell-sorted, compact array and using an additional offset table which contains the cell starting points in that array. Combined with a perfect, collision free hash function this enables $O(1)$ retrieval of the points, contained in a specific cell. A typical query includes retrieving all elements from the cell assigned to the query coordinate and the 26 neighboring cells. This retrieves at least all elements that are closer to the query coordinate than one side length of a grid cell. For further acceleration, this technique can be implemented on the GPU resulting in very low preprocessing times and storage sizes [122].

To accelerate the distance field computation, spatial hashing is employed on the triangles of the reconstructed mesh. Considering the spatial hashing requires an input point set and isotropic remeshing produced approximately equilateral triangles, the triangle mid points can be used to represent the triangles. To avoid a precomputation of triangle midpoints it would also be sufficient to use the vertices of the reconstructed mesh as the input point set, as they provide a similar dense sampling of the reconstructed surface. Subsequently, a hash and offset table are constructed for the chosen point set, taking two times the target edge length l of the isotropic remeshing as the grid cell size. Then, for each vertex $\boldsymbol{v}_i$ of the original mesh, the

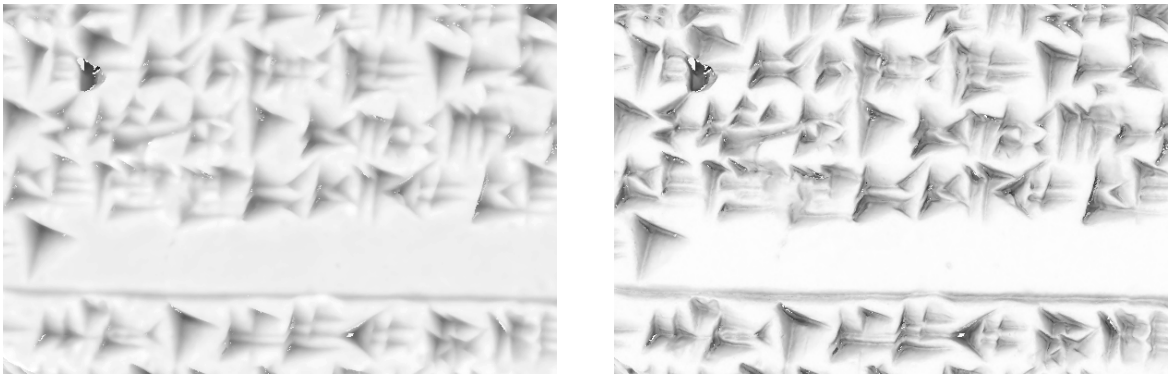
Fragment	Vertices	Time	Triangle tests	Vertex avg	Hit rate	Box tests
Bo2743	1,610,047	16.31s	750,792,310	466.31		3,286,224,029
+hashing		1.45s	39,333,264	24.42	96.44%	227,243,409
Bo71/222	3,373,634	41.63s	2,018,681,788	598.37		8,884,404,400
+hashing		2.42s	47,843,792	14.18	99.40%	321,697,574
Bo68/255	22,434,299	185.53s	8,730,159,727	389.14		38,639,787,749
+hashing		11.63s	259,892,888	11.58	98.49%	1,834,563,857

Table 4.1: Performance of the distance field computation regarding computation time, total triangle test count, average triangle tests per input vertex, optimization hit rate and total bounding box test count. For each fragment, the performance without hashing optimization is shown in the top row and the one with optimization in the bottom row. The test was run on an Intel Core-i7 4790K with a Geforce GTX 780.

spatial hashing is used to compute one upper bound for the minimum distance by retrieving one element from the hash cell of $\boldsymbol{v}_i$ and computing its distance. Hereby the approximation property of the Poisson mesh and the choice of the grid cell size lead to a high probability, this cell contains at least one element. The resulting distance is then used as an early drop out distance for the subsequent tree-based nearest neighbor search. As a result, large parts of the tree do not need to be traversed since many of the bounding-boxes associated to the tree nodes only include elements that are further away than the drop out distance. For the unlikely case that the hash cell of $\boldsymbol{v}_i$ is empty, the tree search is performed without an initial drop out distance. An additional advantage of this technique is that the hash and offset tables only need to be created once and can be reused for all $\boldsymbol{v}_i$.

As table 4.1 shows, this considerably reduces the overall computation times for the creation of the distance field by far over one order of magnitude. On the testing platform, the factor of performance increase for large meshes settled at around 17. The high optimization hit rates, which denote for which percentage of vertices an early drop out distance was found, proves the suitability of the chosen hash grid size of $2l$. Setting the grid size to l decreased the hit rates by about 15 to 20% including an associated negative impact on computation times. In general, hit rates decrease moderately, when the respective fragments contain large areas with strongly concave fracture faces, as these fragments exhibit a lower Poisson approximation accuracy.

It is notable that the number of triangle distance and box distance computations decrease at a much higher rate than the computation times. This difference results from the overhead of spatial hashing and memory management costs. Additional CPU profiling on the testing platform with big meshes showed that the overall impact of saved bounding box tests on the performance is over two times bigger than the impact of saved triangle tests. This is predominantly an effect of the large number of tests since the computation of bounding box distances is computationally less expensive than the computation of triangle distances.



(a) **(b)**

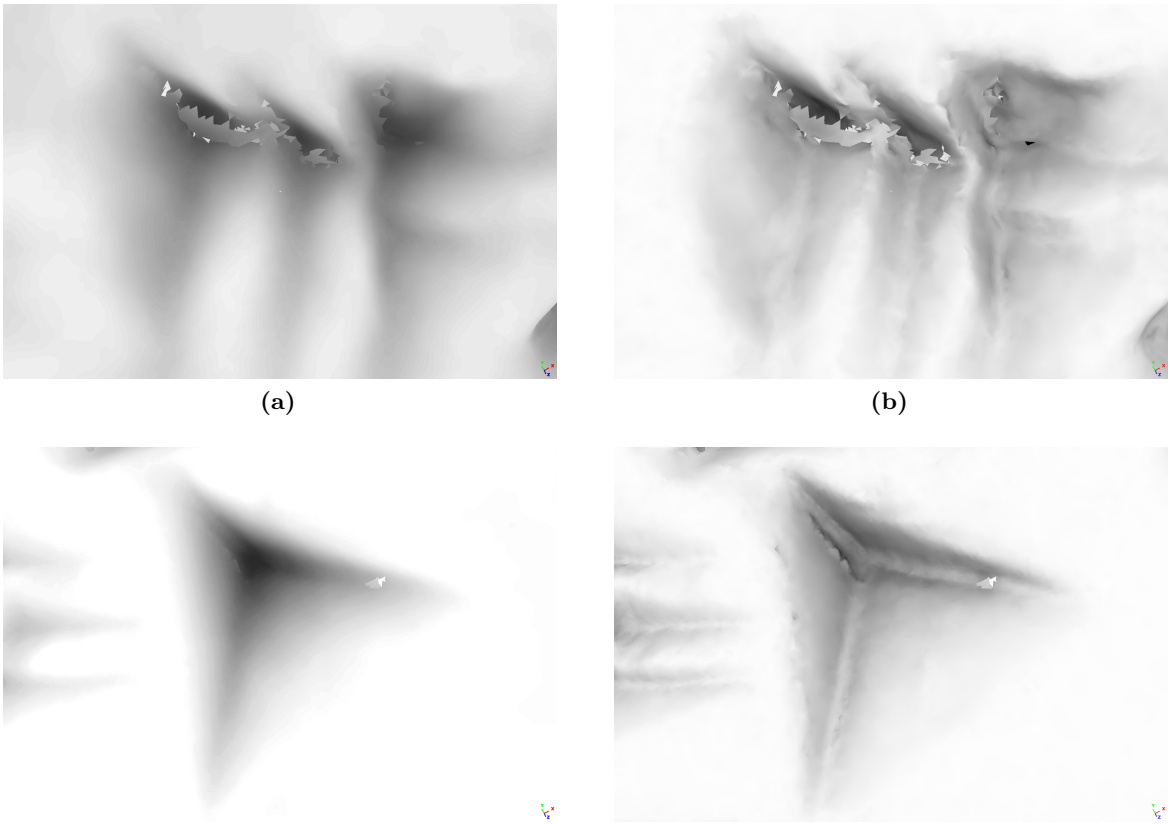
Figure 4.17: Comparison of a grayscale mapped version of the generated distance field **(a)** to the ambient occlusion **(b)** on cuneiform fragment *Bo2743*. For the distance field, larger distances are represented by darker color values.

The performance measurements in table 4.1 were produced with a CPU-only version of the spatial hashing algorithm. The surface reconstruction and distance computations were additionally accelerated using OpenMP [53]. This resulted in a small variation of performance measurements regarding the count of performed triangle and bounding box distance tests, as the OpenMP parallelization causes the processing order of vertices in the isotropic remeshing process to change. The produced, slightly different meshes in turn influence the processing order of the tree-based nearest triangle search and thus the update of the current minimal distances which have a direct influence on the amount of necessary distance computations.

A visual comparison of the generated distance field to the previously generated ambient occlusion data shows significant improvements regarding the intended usage in a watershed algorithm. As can be seen in figure 4.17, the distance field is much smoother than the ambient occlusion data and exhibits the expected approximately linear gradients in the wedges. Also, in contrast to the ambient occlusion data, the distance field is practically artifact free, with no curvature-related disturbances. This becomes especially apparent in magnified wedge regions, as depicted in figure 4.18. The problematic mesh region with overlapping geometry in figures 4.18a and 4.18b shows a clear improvement of depth consistency, especially regarding the contact points of the overlapping geometry patches with the rest of the geometry. In addition to that, curvature-induced discontinuities in wedges, as can be seen in figure 4.18d, are no longer present in the distance field. Therefore, the constructed distance field is an ideal basis for further use in the watershed method that will be presented in section 4.7.

4.6 Region of interest extraction

Restricting the surface area to be considered to sections potentially containing wedges, is a useful prerequisite for the wedge extraction in section 4.7. This becomes especially clear when considering the fact that the wedge extraction effectively only performs a partition of an initial



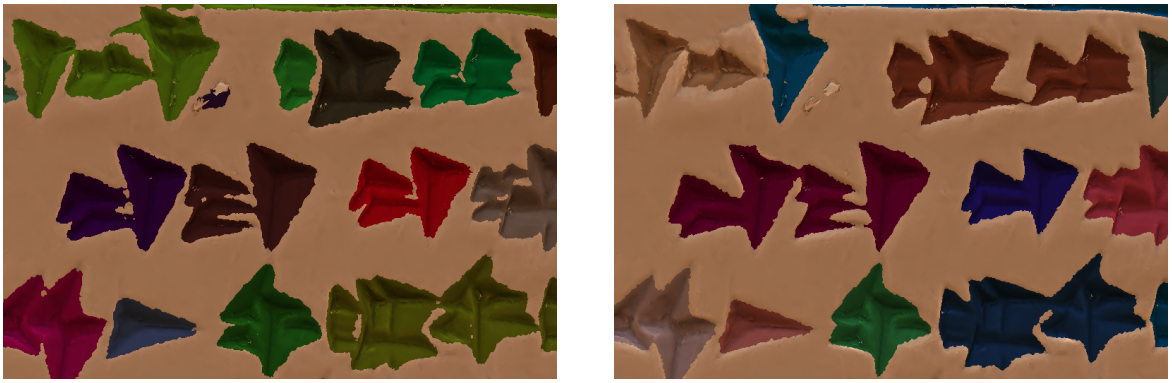
(a) **(b)**

(c) **(d)**

Figure 4.18: Improved handling of surface defects and wedge details of the generated distance field **(a, c)** as opposed to the ambient occlusion **(b, d)**.

surface area. It is therefore necessary to define a suitable construction strategy for this initial region of interest (ROI).

The cuneiform fragments at this point are expected to contain sections with free surface area, fracture faces, and areas with predominantly overlapping wedges. However, a clear separation of fracture faces and areas containing wedges is a challenging task, as damaged wedge areas and fracture faces are difficult to distinguish. The initial area should therefore be designed to include all wedge areas while not necessarily excluding the surface sections corresponding to fracture faces. A distinction between fracture faces and wedge areas can then be made at a later point in time, using more sophisticated segmentation techniques that can take the shape of wedges into account. Regarding the shape of wedges, a second design goal of the initial area involves the wedge borders touching the free writing surface. These borders are not influenced by overlapping wedges and should consequently exhibit a clean separation between wedges and free surface areas. This does not apply to the same extent to a separation between free surface area and fracture faces. While damaged wedge areas may be difficult to distinguish from fracture faces, some types of flat fracture faces can be safely assumed to not contain



(a) **(b)**

Figure 4.19: Comparison of the region of interest extraction based on ambient occlusion with a threshold of 0.95 **(a)** and based on surface distance with a threshold of 0.1 **(b)**. Connected components are colored using random colors.

any wedges. Since wedges, with the rare exception of cuneiform seals, are concave features, approximately wedge-sized convex features can also be ruled out as not containing wedges.

The set of geometry satisfying these conditions is very similar to the complement of the geometric area identified for surface distance reconstruction in section 4.5.2. This ambient-occlusion-based extraction has, however, issues regarding a precise separation between wedge areas and free writing surface. Utilizing the same threshold as for free writing surface extraction, the resulting area shows significant overfitting at wedge borders and rounded wedge tips. This partially results from the opposite objective in section 4.5.2 to retain only the free writing surface that benefits from a moderate amount of underfitting. Lowering the threshold reduces overfitting but at the same time results in unsteady borders and double pointed wedge tips, as can be seen in figure 4.19. Additionally, as mentioned earlier, ambient occlusion is very susceptible to artifacts near non-2-manifold and overlaying geometry.

In contrast, using a thresholded version of the approximated surface distance as the basis for initial area computation produces much more consistent results. This also includes robust handling of non-2-manifold and overlaying geometry. Figure 4.19 shows a comparison of two regions of interest extractions based on ambient occlusion and surface distance. The surface-distance-based version in figure 4.19b produces better defined wedge borders, pointier wedge tips and no artifacts caused by non-2-manifold geometry. In cases where the incorporation of shallow features, like scratches or elongated wedge tips, is especially important, the thresholded surface distance can be complemented by a thresholded curvature component. With this, a vertex is included in the region of interest if it satisfies the surface distance condition or if its curvature indicates a specific degree of surface concavity. Typical indicators for not incorporating the curvature are noisy or significantly damaged meshes as the resulting region of interest would then include a large quantity of non-wedge geometry.

A disadvantage of using the surface distance, as opposed to the ambient occlusion, is the scale dependence of the threshold value. While the ambient occlusion threshold does not depend on the metric scale of the wedges, the surface distance threshold must be chosen in relation to the scale of the wedges. Wedge scale dependent factors with an influence on the optimal threshold value include the depth of the wedges and the size of typical depth variations on the free writing surface.

An automatic determination of this threshold is complicated by the fact that in general, no simplifying assumptions can be made about the distributions of surface types on a cuneiform fragment. The surface of a cuneiform fragment can consist exclusively of free writing surface and wedges in an unknown writing density, but can also contain a very high amount of arbitrarily structured fracture faces. While in the first case, an automatic determination of a threshold may be possible based on an analysis of the distribution of ambient occlusion values, this approach would fail in the presence of many fracture faces with unknown surface structure.

In practice, determination of a suitable threshold is not an issue because typical wedge depths and the structure of the undamaged writing surface are comparatively consistent for a certain type of manuscripts, like the Hittite clay tablet fragments found in Boğazkale, Turkey. Therefore, suitable values can be determined based on wedge size measurements and successfully used on a large amount of cuneiform fragments. In this thesis, surface distance threshold values of 0.05 mm to 0.1 mm have been used in combination with 1 mm to 4 mm deep wedges. For a different type of manuscripts these values might have to be adjusted.

Aside from determining the initial area for wedge extraction, the region of interest extraction can be used to extract a preliminary form of an additional cuneiform script feature related to the text and wedge structure. As mentioned in section 2.1, cuneiform manuscripts may consist of text paragraphs, lines, and cuneiform signs. Some types of cuneiform, like the one depicted in figure 4.20a, additionally show a pronounced spatial separation between individual cuneiform signs using white spaces, as opposed to other types of cuneiform, like shown in figure 4.20b, with a sparse use of white spaces. These text elements can be identified on some manuscripts by evaluating the connectedness of the extracted region of interest. An exemplary result of such a connected component extraction is shown in figure 4.19. In both sub-figures, the connected components are colored using random colors. However, being able to differentiate cuneiform signs or even lines based on the connectedness may be only possible with spaciously written cuneiform.

Geometry wise, the connectedness of components is determined on vertex level by iterating the vertices in a depth-first-search manner. Two vertices that have passed the thresholding are connected if they share a common half-edge. Traversal is performed efficiently using the methods described in section 3.4. To avoid complex data structures, the resulting vertex sets are stored via vertex labeling. These labeled vertex sets can be quickly converted to labeled triangle sets. More precisely, a triangle is assigned a vertex label if all its vertices share the



(a) **(b)**

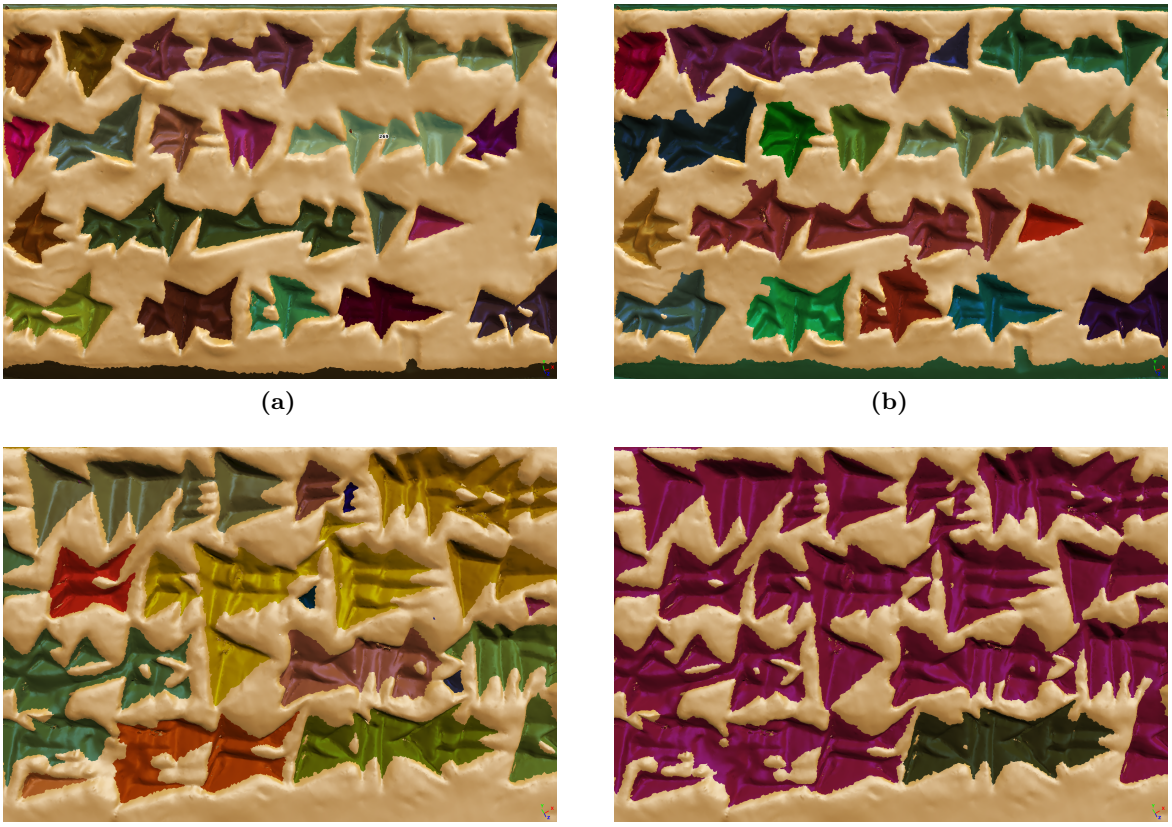
Figure 4.20: Visual comparison of manuscripts with **(a)** lower and **(b)** higher writing density.

same label, which results in an unambiguous labeling. As will be detailed later, triangle-based sets facilitate conducting area dependent computations on the mesh and are more suitable for some visualization tasks.

It is notable that the distribution of connected components is highly susceptible to small variations in the extraction threshold resulting in thin connections between larger geometry patches. A good example of this effect can be observed in the upper text line of figure 4.19 where the four green wedges at the left-hand side are connected in figure 4.19a and split in figure 4.19b. Conversely, the eleven brown wedges at the right-hand side in figure 4.19b are connected as opposed to figure 4.19b. The thin connection areas responsible for this can be removed by modifying the threshold or using a geometric opening operator on the labeled triangles or vertices. Lowering the threshold yields the advantage to be able to consider multiple levels of connectedness but moves the threshold away from the value range suitable for region of interest extraction. This can be addressed by using one or more separate thresholds for connectedness analysis and extrapolating the resulting indices to individual wedge areas, extracted in section 4.7, by choosing the index with the highest coverage for each wedge area. Apart from differentiating word boundaries, this kind of connected component analysis can also distinguish superimposed and touching wedges from isolated wedges. Figure 4.21 shows some real data examples of the effects described above. While the low density writing on manuscript *133/r* allows a good separation of signs or words, the densely written text on manuscript *Bo2743* yields considerably less useful connected components.

4.7 Wedge (area) extraction

The state of wedge extraction at this point consists of the region of interest, which represents geometric areas containing one or more wedges, column or paragraph separators, and parts of fracture faces in any possible composition. Some of these areas may be pre-labeled based on the connected component analysis in the previous section. As the wedge model extraction in



(a) **(b)** **(c)** **(d)**

Figure 4.21: Visual comparison of connected component analysis on manuscripts *133/r* (top) and *Bo2743* (bottom) with higher (left) and lower (right) thresholds.

section 4.8 requires wedge candidate regions for individual wedges, this section presents an approach to partition the region of interest into subregions based on potentially contained wedges. Aside from exceptions regarding superimposed wedges, the goal is to extract one region for each wedge.

The wedges contained in the region of interest are, according to the process of their creation, expected to exhibit the geometric form of small indentations with three more or less pronounced edges. However, deviations from this ideal shape may be caused by, but not limited to, overlapping wedges, corrosion damage, impurities on the surface, fracture faces, non-2-manifold geometry, or a damaged stylus. These geometric deviations may affect all components of a wedge. In addition to that, overlapping wedges may obliterate large parts of neighboring wedges. This produces new features, which often take the form of small ridges at the mutual borders of the indentations. To remain distinguishable, it is necessary for the involved wedges to exhibit significant differences in either position, size or orientation.

A suitable approach for extracting geometric areas that represent candidates for single wedges should be robust against the geometric issues mentioned above. Since a classification of the

wedge components, including possible geometric issues, can be performed later during model fitting, a good correspondence of extracted wedge candidate areas and real wedges is the main goal of this processing step.

The extraction is therefore based on the observation that the most general and common feature of a wedge is its characteristic as an indentation with respect to the surrounding geometry. This feature is observable for the large majority of wedges, to a greater or lesser extent, including heavily damaged and overlapping wedges. Since the absence of this feature has been observed to mostly occur in cases of overlapping wedges with an extremely high mutual coverage, the presence of an indentation will be assumed to be mandatory for the presence of a wedge. On the other hand, not every indentation is associated with a wedge but may also originate from air bubbles in the clay surface or structures on fracture faces.

On a more formal level, an indentation can be defined, with respect to the surface distance field. An indentation is a compact surface area on a mesh where all border points exhibit an equal surface distance and all inner points have a higher surface distance than the border. This definition also accounts for the fact that even single wedges can contain multiple minima of the surface distance field. These are caused by the presence of noise or surface impurities, like grains of sand. An application to triangular meshes needs to consider the discrete sampling of the surface field at the vertices and the possible presence of holes in the mesh. The discrete sampling does not pose an issue, as the vertex properties can be interpolated on the triangles. For a connected surface area consisting of whole triangles, the criteria for an indentation are then fulfilled by a subset of geometry delimited by a polygonal intersection curve. Holes in the mesh do not pose an issue either since the surface distance can be computed regardless of the mesh connectivity and triangle mesh borders are valid borders of a compact surface area. However, holes below the surface distance level of the indentation border may require lowering the level to the lowest level affected by the hole to comply with the indentation definition.

The required existence of an indentation with an equal surface distance at the border motivates the use of a watershed-approach for segmentation. The watershed transformation is an established 2D image processing technique that is predominantly used for segmentation tasks. Its basic concept, introduced by Beucher and Lantuéjoul [20], includes interpreting a scalar input as a height field and assigning image elements to so-called catchment basins that are separated by ridges in the height field. Values for the height field are usually computed by deriving a gradient magnitude on the underlying image data which forces the watershed method to position the resulting ridges at the occurrences of significant gradients in the image. At this point, the assignment strategy for the image elements to the catchment basins is the most important factor in the design of a watershed transformation. The two most common strategy classes are the recursive immersion-based strategies, like introduced by Vincent and Soille [225], and distance-function-based strategies, like introduced by Meyer [144]. Immersion-based strategies rely on successively flooding the height field, in most cases generating an ad hoc segmentation, while finding a trade-off between running time and segmentation quality.

This is predominantly realized by employing a priority queue of candidate elements that determines the order of processing. Variants of this class of watershed strategies include the complex inverse filling method presented by Planchon et al. [157], the Terraflow method of Arge et al. [10] with improved running time and a small memory footprint, and the fast Priority-Flood method of Barnes et al. [14].

The majority of distance-based strategies rely on computing an initial set of minima in the height field and providing a distance-based definition for directly assigning elements to catchment basins. In contrast to flooding methods, this can provide an explicit mathematical definition of the generated segmentation. Popular strategies of this type include the topographical distance-based approach by Meyer et al. [144] and several variants based on shortest paths algorithms, gradient descent, and hill climbing variants. Additionally, some methods for direct computation of the watershed transform exist, like the Union-Find-based method of Meijster et al. [141].

Both immersion- and distance-based watershed techniques usually suffer from a significant amount of over-segmentation. This effect can be counteracted in pre-/postprocessing stages by filtering the height field, fusing multiple basins or complementing the watershed approach by other segmentation methods [21]. A more complete overview of two-dimensional watershed techniques can be found in [175].

A transfer of traditional 2D watershed techniques to the domain of three-dimensional triangle meshes was presented by Mangan et al. [134]. For this approach, the required height field is derived from the magnitude of a curvature measure, computed on the mesh surface. The assignment of vertices to the catchment basins is realized by computing and labeling local minima and plateau areas first and then assigning labels to plateaus and vertices through ascending in the height field until a labeled area is reached. The issue of over-segmentation is addressed in a post-processing step by rule-based merging of neighboring basins. With Fast Marching Watersheds, Page et al. [155] presented a hill-climbing technique, using principal curvatures and the perception-based minima rule [92].

The watershed method presented here adopts the basic principles described by Mangan et al. [134] to transfer the watershed approach to the domain of three-dimensional triangle meshes. This includes, as a first aspect, the mesh vertices as processing elements, with the neighbor relation being defined by the edge connections. A second aspect is the precomputation of a height function on the vertices that serves as the base for the watershed transformation. The height function used here is, however, not a curvature, but the surface distance, computed in section 4.5. A third aspect, the optimization of element access by presorting the elements, is borrowed from the immersion-based watershed method of Vincent et al. [225]. Despite the vertices being chosen as the operational domain of the watershed method, the resulting segments for wedge model extraction are expected to consist of triangle patches. Aside from the required conversion of the segmentation result, this needs to be considered when applying the watershed method.

An outline of a simplified version of the proposed watershed method in pseudo-code is shown in algorithm 4.22. The starting point is the derivation of a processing order for the vertices $\boldsymbol{v}_i$ contained in the extracted region of interest. This is realized by sorting the vertices in ascending order according to their surface distance. Unlike the 8-bit grayscale images commonly processed by 2D watershed methods, the surface distance is a continuous function represented by floating-point numbers and sampled with a much higher sampling density. This, and the uneven nature of real world surfaces makes the occurrence of planar areas in the distance field highly unlikely. Thus, no separate treatment for plateaus, like in [134] is required. Also, the method does not rely on a precalculation of local minima, as in other watershed approaches. Instead, the minima are calculated on-the-fly during the immersion process, as will be detailed later.

The current state of the immersion process and the segmentation result in form of catchment basins are stored in form of integer vertex labels l_i. Vertices are labeled based on the assignment to individual catchment basins with unique positive numbers. Unvisited vertices and border vertices are labeled with the special labels 0 and -1. Before the start of the immersion process, all labels are reset and labeled as unvisited, including the vertices outside the region of interest.

During the immersion process, all vertices are processed in ascending order according to their surface distance values. This causes the deepest vertices to be processed first, as the signed values d_i of the surface distance field are negative for vertices below the free writing surface. However, the distance ordered processing may result in a seemingly random processing, when viewed in relation to the position on the mesh surface. When processing a vertex $\boldsymbol{v}_i$ with surface distance d_i, it can therefore be assumed that all unprocessed vertices exhibit a higher value of d_i. To process an unvisited vertex $\boldsymbol{v}_i$, its 1-ring $N_1(\boldsymbol{v}_i)$ is scanned for labels of neighboring vertices that have already been assigned to a catchment basin. In this process, the number of touching basins and the biggest and smallest basin are determined.

In a first case, where no basin is found in the 1-ring, $\boldsymbol{v}_i$ is part of a local minimum in the distance field. This results from the fact that the 1-ring exclusively contains unvisited vertices, which must have surface distance values higher than or equal to d_i, according to the processing order. Consequently, an unused label value is assigned to l_i to start a new basin. At this point, the version of the immersion algorithm shown in figure 4.22, includes an additional 2-ring check. If an existing basin is found in the 2-ring of $\boldsymbol{v}_i$, the vertex is added to that basin, instead of creating a new one. This is a countermeasure against over-segmentation based on the observation that the minima of individual wedge areas are usually not located near the border of the wedge areas. Instead, this constellation mostly occurs in the presence of noisy or non-2-manifold geometry and creates unintended additional basins. The resulting basins with the same label, naturally merge upon further expansion during the immersion process. Since this countermeasure is based on a 2-ring check instead of distance calculation, it adapts to the mesh resolution and there is no need to define an additional scale dependent threshold. It will be shown later that this countermeasure has only minimal impact on the

```
1   sort mesh vertices in ascending order by surface distance
2   for all v_i in mesh
3   {
4     l_i = 0 // reset label
5   }
6
7   for all v_i in ROI
8   {
9     if (l_i == 0) // vertex is unvisited
10    {
11      basin_count = 0
12      l_sb = 0
13      l_bb = 0
14      for all v_j in N_1(v_i)
15      {
16        count basins
17        update l_sb // track smallest basin
18        update l_bb // track biggest basin
19      }
20      if (basin_count == 0)
21      {
22        l_tmp = 0
23        for all v_j in N_2(v_i)
24        {
25          if (l_j != 0)
26          {
27            l_tmp = l_j
28            break;
29          }
30        }
31        if (l_tmp == 0)
32        {
33          l_i = new label // create new basin
34        }
35        else
36        {
37          l_i = l_tmp // add vertex to existing basin
38        }
39      }
40      if (basin_count == 1)
41      {
42        l_i = l_sb // add vertex to existing basin
43      }
44      if (basin_count > 1)
45      {
46        if (size_of_smallest_basin < threshold)
47        {
48          l_i = l_bb
49          for all v_j in smallest basin
50          {
51            l_j = l_bb // add to biggest basin
52          }
53        }
54        else
55        {
56          l_i = -1 // label as border vertex
57        }
58      }
59    }
60  }
61
```

Figure 4.22: Pseudo-code of the watershed labeling algorithm for wedge extraction. Here, the $v_?$ denote vertices and $l_?$ denote labels.

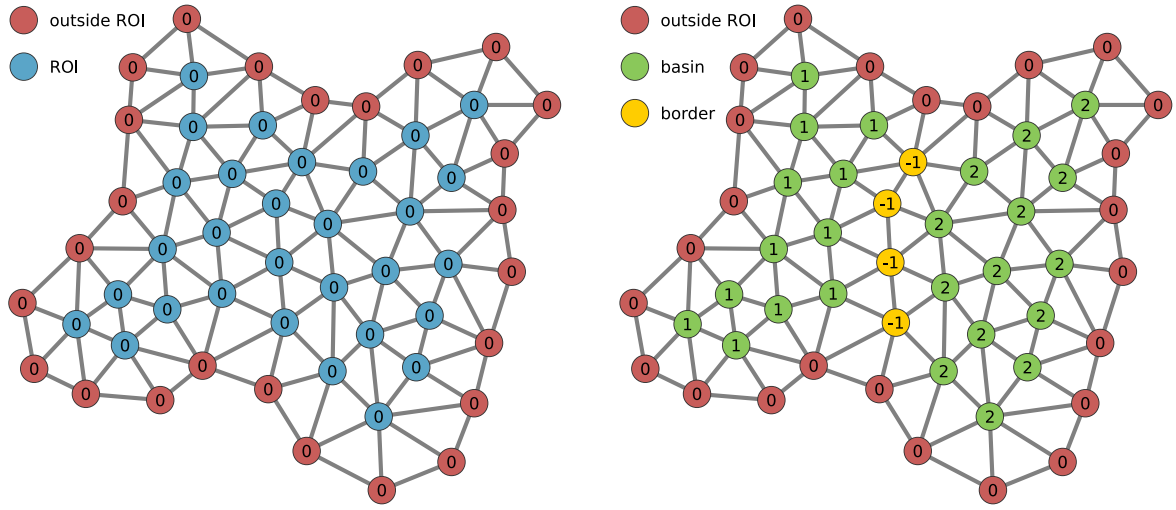
final segmentation result when applied to comparatively smooth meshes with a balanced vertex density in combination with the basin merging described below.

The second case, where one basin is detected in the 1-ring of $\boldsymbol{v}_i$, covers the usual case of basin expansion. The presence of exactly one touching basin implies that the surface distance value of all vertices in the basin is smaller than or equal to d_i. On the other hand, all remaining vertices in $N_1(\boldsymbol{v}_i)$ must exhibit a surface distance greater or equal to d_i. Therefore, $\boldsymbol{v}_i$ can be safely added to the detected basin.

The third case, where more than one basin is detected in the 1-ring, covers the collision of multiple basins. The current vertex $\boldsymbol{v}_i$ can then either connect the neighboring basins or become part of a border, separating the basins. This decision is made depending on a merging-rule that evaluates the significance of the involved basins. Insignificant basins are then added to more significant basins.

The design of a suitable merging-rule has to consider the geometrical properties of the 3D-scanned cuneiform surfaces. During the watershed submersion process, basins are expected to form near the depth points of the wedges as the surface distance controls the processing order. Common merging criteria for watershed basins are the depth of the basins or a depth difference of neighboring basins. In this context, the depth of a basin is defined as the height difference of the vertices with the smallest and the biggest height value in that region. As the height is given by the surface distance, this would imply that the importance of a wedge depends on its depth. However, regardless of their importance, the watershed basins forming in the wedges may be either shallow or deep and exhibit varying degrees of curvature. The difference in depth of neighboring wedges may also vary significantly, which disqualifies depth differences. Using an area, associated with the basins does not work either, as in this case regions with few large triangles could be rated more important than regions with many small triangles. Instead, a merging criterion based on the vertex count of the basins is used. This builds upon the assumption that the importance of a basin is endorsed each time a vertex is added to that basin. The resulting merging criterion combines a size-related aspect with a basin-age-related aspect but at the same time does not depend as much on the absolute scale of the wedges. The merging-rule is then realized by introducing a minimum basin size with respect to the basin vertex count. For the cuneiform datasets processed in this thesis, a minimum basin size of 25 worked well on the large majority of datasets. Datasets from other scanning sources like the Leuven light dome [231], however, required to adapt this value manually to the changed mesh characteristics.

With regard to the design of the merging rule as a thresholding of the basin vertex count, the merging-rule only needs to be evaluated on the smallest basin in the neighborhood of $\boldsymbol{v}_i$. This basin has already been determined while scanning the neighborhood of $\boldsymbol{v}_i$. If the size of the smallest basin is below the threshold, the basin's vertex labels are replaced with the labels of the largest basin which has also been previously determined. This minimizes the amount of vertices that need to be relabeled. In the case of more than two neighboring



(a) (b)

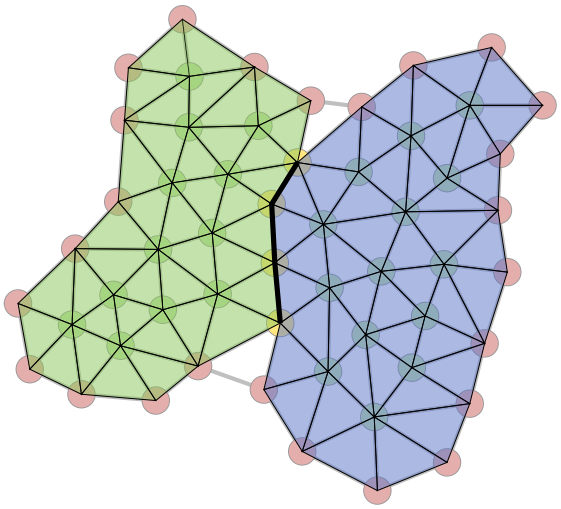
Figure 4.23: Comparison of the vertex labeling **(a)** before and **(b)** after the application of the watershed algorithm. Vertices in the region of interest are colored in blue and vertices outside the region of interest are colored in red. The relabeled green vertices have been assigned to basins and the yellow vertices have been relabeled as borders.

basins the label of the biggest basin could be propagated to all smaller basins, however, the occurrence of this case has been very rarely observed. In addition to that, the execution of multiple merging operations in one step caused only minor changes in the segmentation result because non-merged basins are likely to be merged during the subsequent processing of neighbor vertices of $\boldsymbol{v}_i$. Therefore, the number of merged basins per step is limited to two.

At this point it is important to consider the intended result of the wedge extraction to consist of triangle patches. Vertices with multiple neighboring catchment basins that do not pass the merging rule will therefore be labeled as border vertices. This creates a layer of boundary vertices between the basins which provides a consistent base for converting vertex patches to triangle patches.

Figure 4.23 shows the starting point and the final state of the watershed vertex labeling process for a small exemplary region of interest on a mesh, containing two wedges. The red vertices outside the ROI have also been labeled, because they are scanned as part of the 1-ring of the vertices within the ROI. However, only labels within the region of interest are changed during the watershed vertex labeling process.

For further processing, the labeled vertex sets are converted to an equal number of triangle patches. Because of the employed border vertices, this can be trivially archived by traversing all mesh triangles and assigning a triangle to a patch if one of its vertices has a non-zero positive vertex label. Performing this conversion on the exemplary segmentation result of figure 4.23, leads to the two triangle patches shown in figure 4.24. By construction, the resulting triangle patches are delimited exclusively by non-ROI vertices and border vertices.

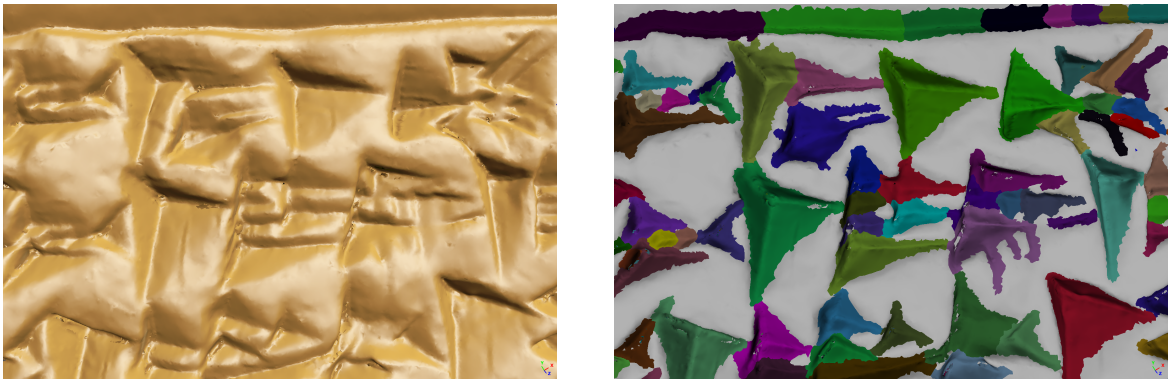


(a)

Figure 4.24: Conversion of the exemplary vertex-based watershed segmentation from figure 4.23 to two triangle patches (green and blue). The thick line marks the boundary between the patches.

Neighboring triangle patches exhibit a gap free contact boundary consisting of border vertices. As opposed to a segmentation without border vertices, all geometry between the patches is included and no unsteady borders are produced. Furthermore, the border vertices enforce the boundary of neighboring triangle patches to be positioned exactly on the ridge between the involved basins.

A result of the watershed segmentation on real data is shown in figure 4.25 for a densely written section of cuneiform fragment *Bo2743*. This section features many overlapping wedges and a not negligible quantity of geometric issues like small holes and non-2-manifold geometry. As can be seen in figure 4.25b, the resulting triangle patches show a very small amount of inconsistencies. A closer inspection reveals that for the 56 wedges contained in the depicted section, only two wedges were segmented into two separate triangle patches and two triangle patches cover more than one wedge center. False detections of this type usually cause the wedge model extraction, described in section 4.8, to fail or to miss some wedges. In contrast, the paragraph separator at the top of the section, represented by multiple triangle patches, will not result in false detections as the individual patches do not exhibit the required geometrical properties for successful model fitting. As will be detailed later, occasional narrow wedge tip regions fused with the triangle patches of other wedges do not affect the wedge model extraction either. It has to be noted that the number of false detections concerning multiple strongly overlapping wedges, represented by only one triangle patch may rise significantly on cuneiform manuscripts containing many occurrences of such wedges. However, as will be shown in section 4.8, the wedge model extraction will mostly manage to extract the most prominent of the contained wedges.



(a) **(b)**

Figure 4.25: Watershed segmentation results for a section of fragment *Bo2743*. A shaded version of is shown in **(a)** and the final segmentation result in form of randomly colored triangle patches in **(b)**.

Determining general statistically significant values for the success rate of the wedge extraction is difficult as it can only be evaluated manually and varies significantly between individual manuscripts. Even the evaluation of one large manuscript with several thousands of wedges would be very time-consuming but the results would only be transferable to a very limited extent to other manuscripts. In addition to that, the success rates also depend heavily on the presence of fracture faces in the scan data. As fracture faces and damaged wedge areas may geometrically be extremely similar, they are processed in the same manner during wedge area extraction with wedge verification being postponed until after wedge model extraction. These areas may, at this point, contain large numbers of false wedge candidates, which will be sorted out later. Therefore, this thesis does not present average wedge extraction success rates. However, for reasonably damage free manuscripts, like the one shown in figure 4.25, and not taking fracture faces into account, wedge extraction success rates of over 90% have been observed. In contrast, the extraction success rates on very noisy or significantly damaged manuscripts can become arbitrarily bad, which is also true for a manual wedge extraction performed by a human expert.

The effectiveness of the design components against over-segmentation of the presented watershed approach can best be demonstrated by examining the vertex labeling. Figure 4.26 shows a color-coded comparison of vertex labelings obtained by selectively deactivating the basin merging and the 2-ring check before basin creation for the same fragment section as in figure 4.25. As expected for a watershed approach, a result without any over-segmentation countermeasures, as shown in figure 4.26a, exhibits an extensive amount of over-segmentation. Applying the 2-ring check, as shown in figure 4.26b, yields a large improvement, but still produces a significant amount of over-segmentation. Applying the basin merging, as shown in figure 4.26c, produces a very clean result that is nearly identical to the result using both countermeasures, as shown in figure 4.26d. The main difference of figures 4.26c and 4.26d is

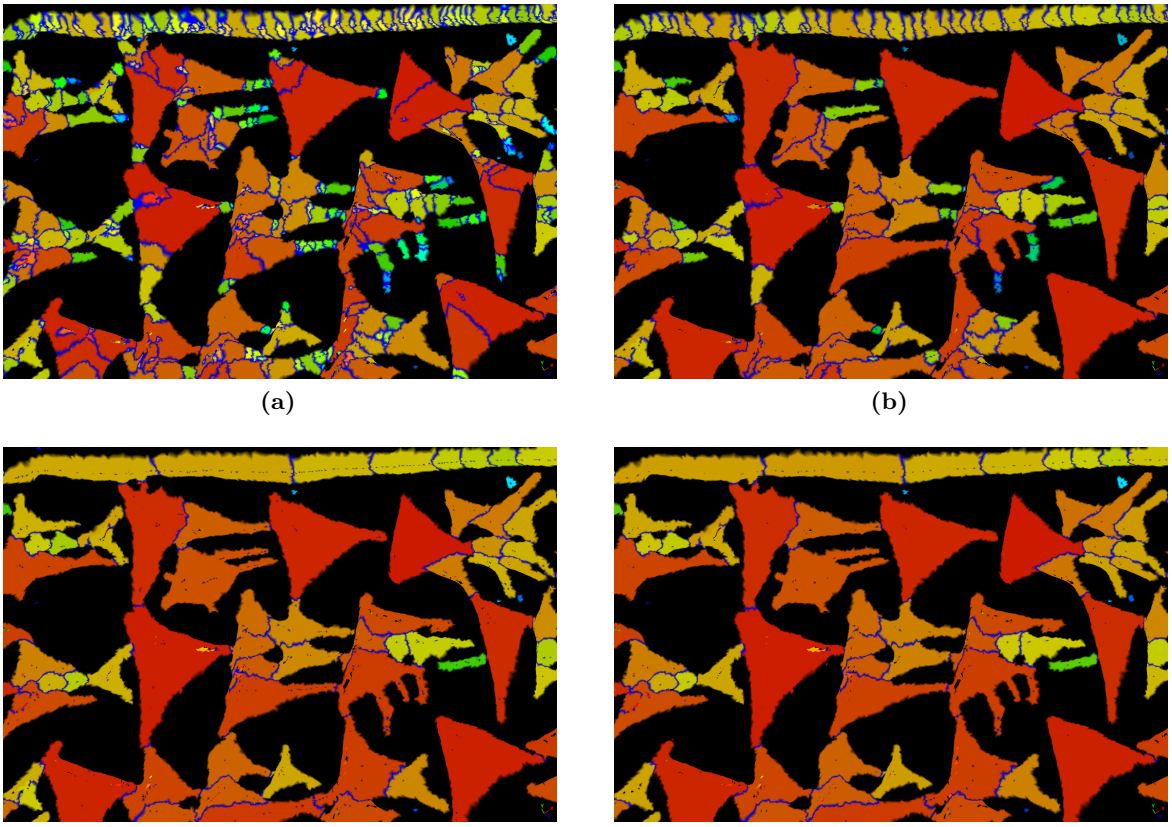
a strong reduction of basin start vertices which appear as tiny dark dots which is especially visible in the paragraph separator at the top. This does not affect the final segmentation to a large extent, as the basin merging is much more effective than the 2-ring check on non-noisy meshes. In contrast, on noisy meshes and meshes with steep features, the coverage of the 2-ring check can increase faster, than the coverage of the usual basin growing and exert a positive effect on the segmentation result. However, in the case shown in figure 4.26, the 2-ring check causes the position of border vertices between basins to improve slightly.

It is worth mentioning that the incorrectly extracted wedges mentioned earlier, are correctly separated by border vertices in figure 4.26b, and have been fused in figures 4.26c and 4.26d. This could indicate that the threshold value 25 for basin merging is set slightly too high. Lowering the merging threshold in this case, however, causes other basins to split incorrectly before causing the incorrectly fused basins to split. This underlines the fact that the choice of the merging threshold is part of a trade-off between over- and under-segmentation.

The result of the watershed segmentation is a set of triangle patches that create an unambiguous, non-overlapping partition of the region of interest on the mesh. The obtained triangle patches, the so-called wedge areas, represent candidate regions for geometric areas on the surface that contain wedges. As intended, the large majority of these regions shows a good correspondence with real wedges. A model-based extraction approach for the contained wedges, will be described in the following section.

4.8 Wedge model extraction

The wedge model extraction in this section takes the extracted wedge areas as an input and targets to fit a wedge model, as defined in section 2.2.3, into each one of them. As already shown in the examples in figure 4.1, cuneiform wedges can occur in a variety of configurations of varying complexity. For undamaged wedges, this mainly includes differently sized single wedges, multiple superimposed wedges of the same orientation, and multiple superimposed wedges with differing orientations. In reality, each of these configurations may additionally be affected by varying amounts of surface damage and mesh errors. This combined leads to a large amount of possible configurations. To simplify matters, it is therefore exploited that the wedge extraction from the previous section already produces wedge areas with a good separation of individual wedges, even for the majority of superimposed wedges. According to this assumption, a suitable wedge model extraction then only has to consider the remaining possible variations of one wedge that can be contained in one wedge area. Aside from the trivial case of an undamaged single wedge, the identified cases of complex and therefore potentially problematic cases include filled wedges, wedges with incomplete edges, wedges with incomplete faces, and wedges with a non-existing or an ambiguous depth point. The missing wedge parts can be either caused by superimposition effects or by holes and errors in the underlying triangle mesh. However, individual components of the wedge model can only be extracted if the wedge area still contains a minimum amount of corresponding geometric representation.



(a) **(b)** **(c)** **(d)**

Figure 4.26: Comparison of different levels of watershed over-segmentation countermeasures. A segmentation result without basin merging and 2-ring check is shown in **(a)**, a result with 2-ring check in **(b)**, with basin merging in **(c)** and with basin merging and 2-ring check in **(d)**. The vertex color scheme uses black for vertices outside the ROI, dark blue for basin start vertices and border vertices. Basin vertices are colored ascending using a red-yellow-green gradient depending on the surface distance of their start vertices.

One additional case to consider is that a wedge area may have been extracted at a location that does not contain any actual wedges and may therefore contain non-wedge-shaped geometry. This case will not be handled separately by the here presented wedge model extraction but in a post-processing step for wedge validation in section 4.9.

A usable wedge model extraction should be able to handle wedges of significantly differing sizes and be robust against deformed, incomplete, and damaged wedges and mesh errors. Here, the wedge size does not primarily refer to the spatial extent of the wedge areas but to the number of triangles in the respective section of the 3D-scanned mesh. The model extraction should work well on small wedge areas with under one hundred triangles and on big ones with several thousands of triangles at the same time, as both variants may occur on the same cuneiform fragment. In view of the expected geometric issues, the model extraction should take into

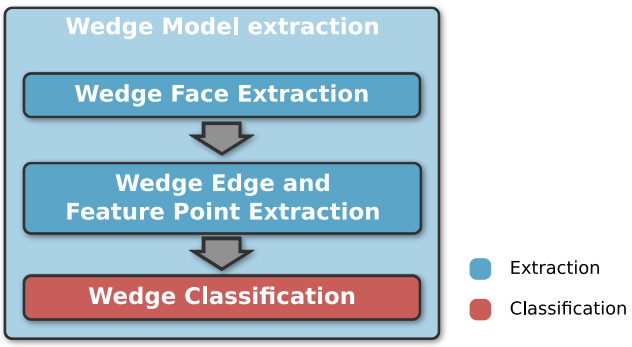


Figure 4.27: The wedge model extraction pipeline, with the pipeline stages colored depending on their operational domain.

account as much of the geometric information contained in the wedge areas as possible and use robust extraction methods that are not susceptible to data noise.

To improve extraction results, the design of the wedge model extraction should also consider prior knowledge about the geometric quality of the 3D scans and the relative importance of the wedge components. As the faces of a wedge are more accessible to a structured-light scanner than its edges and the wedge faces usually cover a much larger area than the wedge faces, they can be expected to be scanned more accurately. In addition to that, the wedge edges are more likely to be affected from scan merging artifacts because they are located in regions with a comparatively high curvature. These aspects suggest that the geometry of the wedge faces should be preferably used for fitting the wedge model. On the other hand, the wedge edges are usually the most important philological features of a wedge, which is reflected by the fact that most cuneiform hand copies use an edge centered or edge only representation. Therefore, the wedge model extraction presented here will prioritize a good wedge edge approximation over a good wedge face approximation, despite the expected lower quality of the edge geometry. These conflicting goals are realized through improving the wedge edge extraction by using the wedge face geometry. Consequently, the wedge model extraction is realized by performing a normal-based wedge face extraction to locate the required data points for a subsequent edge and depth point approximation.

A pipeline graphic of the proposed wedge model extraction is shown in figure 4.27. The initial extraction of the wedge faces will be described in subsection 4.8.1, followed by the wedge edge and feature point extraction in subsection 4.8.2. The model extraction is then completed by the wedge classification, described in subsection 4.9.1.

4.8.1 Wedge face extraction

According to their construction by the planar side face of a writing stylus, wedge faces on the surface of an extracted wedge area are expected to consist of compact geometric areas with comparatively low curvature. These areas should in most cases be delimited by three special

types of features. The first type of features are wedge edges that should usually be represented by thin, line like surface areas with a concave curvature. The second and third types of features are wedge/wedge and wedge/surface transitions, which are expected to consist of line like surface areas with a convex curvature. However, in a real world scenario on a 3D-scanned mesh, wedge faces may contain significantly curved features and holes or be delimited by mesh borders, very subtly curved features, or sparsely connected, non-2-manifold geometry. In addition to that, geometry patches belonging to one wedge face may not necessarily be connected, when affected by cracks in the surface or overlapping wedges.

Extracting wedge faces from a comparatively small patch of a triangular mesh can be regarded as a partitioning problem. As such, it shares some common features with the wedge area extraction problem, discussed in section 4.7. This includes an expected result in form of a set of compact triangle patches and the borders of the resulting triangle patches being linked to some kind of curvature related surface features. An obvious approach would therefore be to apply a similar extraction method. However, the watershed approach requires a sufficiently smooth height field to operate on, which in this case would have to be derived from a curvature related term. This is an issue, since the discrete curvatures as described by Meyer et al. [145], are far to noisy for this application on the available 3D-scanned cuneiform meshes. Smoothing out these curvatures or using a smooth, scale dependent curvature, like the Growing Least Squares method [143], is not an option as it would require prior knowledge about the scale of the delimiting curvature features. The unavailability of this information becomes clear when considering that wedges of the same physical size can have differently sharp edges and that the wedge size can vary significantly. In this case further disadvantageous aspects of the watershed approach include its inability to extract unconnected surface patches as one region and the fact that it relies on a locally operating greedy strategy. In view of the expected geometric variability, using a more global approach for wedge face extraction seems to be more promising, because it takes the whole wedge geometry into account. Therefore, the chosen approach for wedge face extraction will rely on a cluster analysis.

A suitable attribute for clustering should be independent of the position, the orientation and the scale of the extracted wedge areas and the contained wedges. This set of conditions is fulfilled by the triangle normals of the wedge areas. In addition to that, the triangle normals, as a first order derivative, are much less sensitive to local noise than the discrete mesh curvatures, which are a second order derivative. Figure 4.28 shows a perspective projected visualization of the distribution of triangle normals for three exemplary wedges. For this purpose, the normalized normal vectors are viewed in normal space, separated from their position on the triangle mesh. As all vectors start at the coordinate origin, the normals can be visualized as a set of points on a sphere of radius 1. The dense areas of the distributions in figures 4.28a and 4.28b correspond to the triangle normals on the three wedge faces of the wedges shown in figures 4.28d and 4.28e. The less dense elongated areas in between correspond to the triangle normals in the wedge edges. As can be seen in figures 4.28b and 4.28c, these areas in the triangle normal distributions tend to be less pronounced in filled, damaged and small wedges.

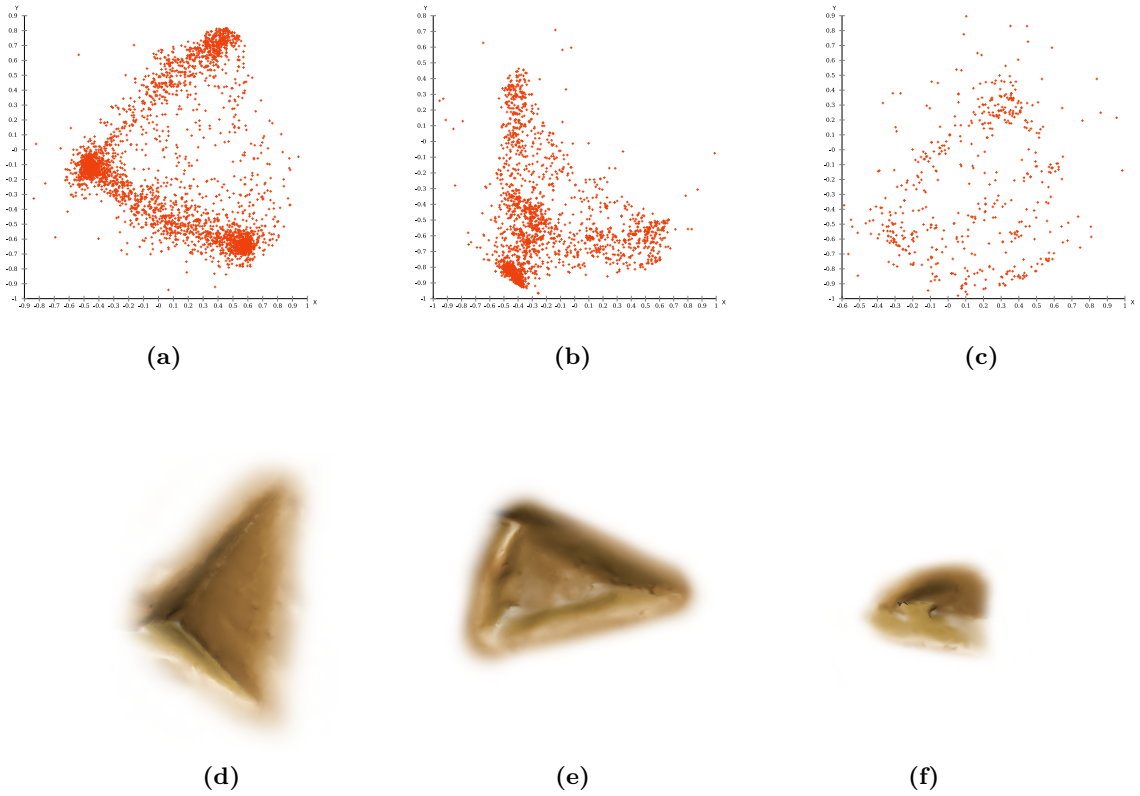


Figure 4.28: Projective visualization of the distribution of triangle normals within a wedge, for a big, well-defined wedge **(a/d)** a filled wedge **(b/e)** and a small subtle wedge **(c/f)**. Normals are shown as red points on a unit sphere projected onto the writing surface as seen in the respective wedge screenshots in the bottom row.

Methods for cluster analysis can be classified based on the underlying cluster model. The most common methods used for image processing and, in particular, point cloud segmentation purposes include connectivity-based clustering, density-based clustering, distribution-based clustering, and centroid-based clustering. A short overview of clustering methods for point cloud data will be given in the following.

Connectivity-based clustering identifies point clusters by building on the assumption that nearby data points are more likely to be related than far away points. Connectedness, or linkage of points, is therefore defined as a distance-based measure, with clusters being defined as sets of connected points. Therefore, the points in one cluster share a common value for their linkage criterion. Clustering methods of this type usually use a divisive, top down or an agglomerative, bottom up approach to determine the final clustering [176]. In a divisive approach, the clustering starts with a single cluster containing all points and the linkage criterion is successively restricted to split clusters. Agglomerative approaches start with individual clusters for each point and fuse clusters by relaxing the linkage criterion. For both approaches, the final clusters need to be selected according to a user specified criterion

from the resulting cluster hierarchy. Consequently, connectivity-based clustering is also called hierarchical clustering. The main disadvantages of this clustering model include a bad robustness against outliers, the necessity to extract the final clusters from the hierarchy and a comparatively slow running time. Especially the running time of $O(n^3)$ in the general case makes this model unsuitable for larger datasets [149]. Optimizations exist for special cases, like single-linkage (SLINK) [201] and complete-linkage clustering (CLINK)[55], to reduce the running time to $O(n^2)$. However, this comes at the cost of only computing an approximative solution. Regarding the wedge face segmentation problem, especially the slow running time and the tendency of hierarchical clustering to yield thin unbalanced clusters are problematic.

Density-based clustering methods [118] are not only able to determine clusters as a partition of the original data, but to also handle outliers. This results from the connectedness criterion of the clusters being based on a density related measure, and makes these methods comparatively robust. Consequently, like for connectivity-based clustering, the shape of clusters can assume arbitrary forms, including concave and nested shapes. However, for cluster borders to be well-defined, density-based approaches require a significant density gradient at the cluster borders. The most common members of density-based clustering methods include the DBSCAN algorithm [61] and the OPTICS algorithm [9]. The DBSCAN algorithm determines the density based on a distance ϵ for neighbor search and identifies so-called core points with a minimum number of neighbors. Clusters are then formed by connected core points and non-core points within the ϵ distance of the core point clusters. Remaining points are classified as outliers. This works well for clusters of similar density, but can usually not handle clusters with significantly differing densities. In addition to that, the choice of a suitable ϵ can be difficult for unknown data characteristics. These issues are addressed by the OPTICS algorithm by introducing a core distance and a reachability distance. The latter acts as a density dependent $\epsilon' < \epsilon$ to determine the connectivity of points. Since the OPTICS algorithm operates on a priority queue, with all points being annotated with the smallest reachability distance and the core distance, the output can be visualized as a reachability graph to view the cluster induced hierarchy. For reasonable choices of ϵ the running time of both, DBSCAN and OPTICS is about $O(n \cdot \log n)$ with a worst case of $O(n^2)$, thus outperforming hierarchical clustering methods. In view of the varying densities within each of the directional distributions shown in figure 4.28, DBSCAN is not expected to yield satisfying results on the wedge face normals. OPTICS on the other hand would be able to adapt to the density variations. This, however, would lead to unevenly sized clusters and consequently to an imbalanced assignment of triangles located at the transition between wedge faces. Additionally, the outliers detection of density-based clustering methods may classify the triangles located at the transition between wedge faces as not belonging to any face, which would be especially problematic on less clearly defined wedges. This is due to the requirement of the subsequent wedge edge extraction to rely on the extracted wedge face triangle patches to share some common mesh edges.

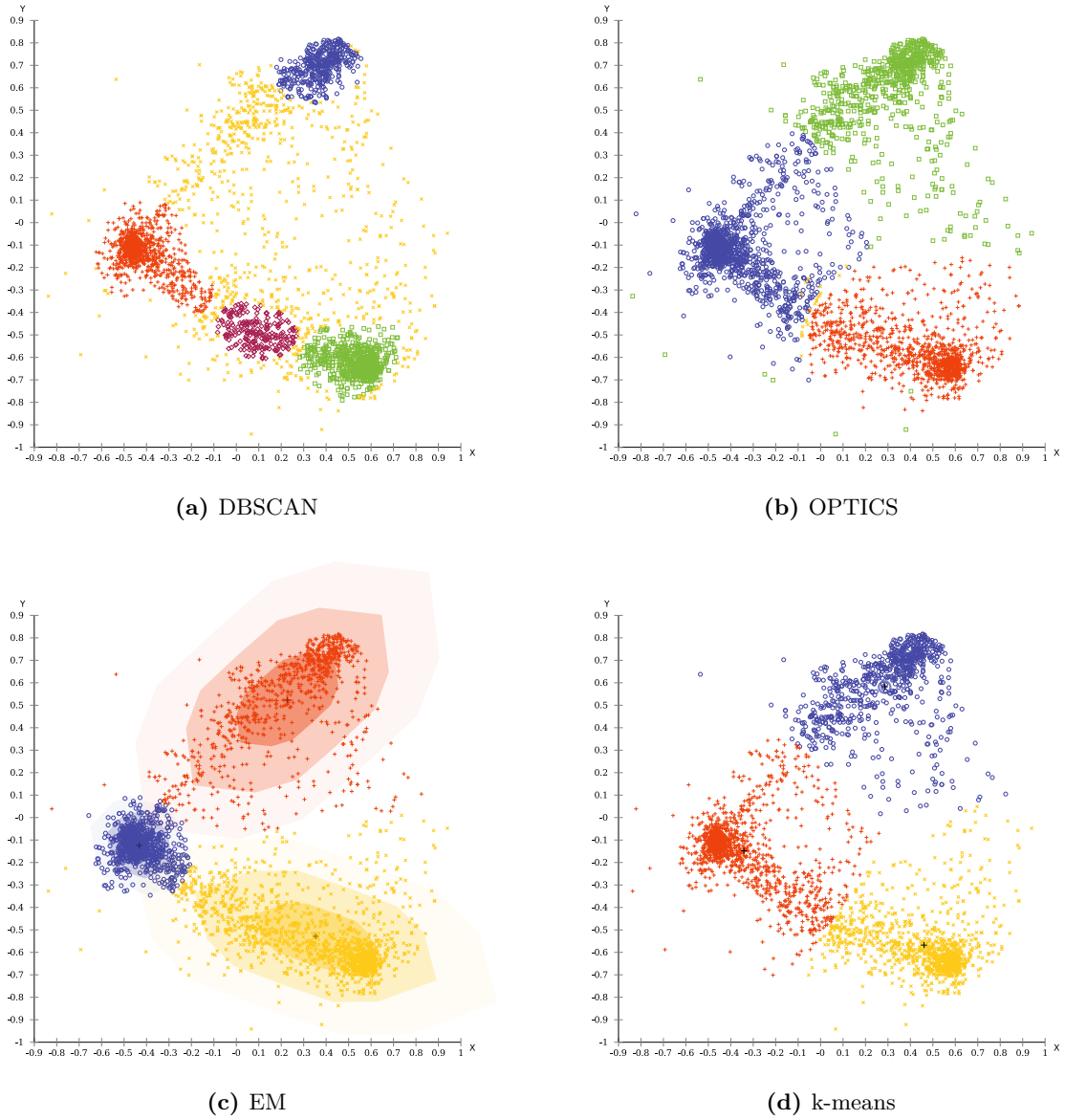
Distribution-based clustering methods, like expectation-maximization (EM) clustering [57], rely on finding a statistical model, including its parameterization, to derive a predefined number

of clusters k. One of the most commonly used models is the Gaussian mixture model [88] that models the data as a mixture of Gaussian distributions. The EM clustering method is able to yield a local numerical solution for the maximum-likelihood parameters of a chosen statistical model, despite the issue that an analytical solution for the parameters is not available if the parameter set includes latent, non-observable variables. Therefore, the parameters are approximated in an iterative process starting with a set of random parameters. The iterative part of the EM clustering then consists of two basic steps. The first step is the expectation step that estimates the hidden parameters based on the currently computed values of the model parameters and the observable input data. The resulting estimation function allows to improve the assignment of the data points to individual distributions. The second step is the maximization step, in which a new set of model parameters is computed using the maximum-likelihood method for the assigned data points. Both steps are repeated until convergence is archived. The main advantages of the EM clustering method are its capability to handle incomplete distributions and to detect clusters of varying sizes. Furthermore, complex models can be applied to account for complex data distributions. However, the running time not only depends on the data correlated time to compute the E and M steps but also on the number of required iterations until convergence is archived. Therefore, a general runtime complexity can not be specified. The usual limitations [74] of EM clustering, namely, a slow convergence, impracticability for very high cluster numbers, and an algorithmic break down for cases of near singular covariance matrices of clusters should not apply to the use case considered here. However, one issue of EM clustering regarding the wedge faces extraction problem is that it is expected to yield varying results for constant parameters if the input distribution does not match well with a multivariate Gaussian mixture, containing $k = 3$ Gaussians, like the ones shown in figure 4.28. Nevertheless, EM clustering can be used to optimize the clustering results, obtained using other approaches [74] like agglomerative hierarchical clustering.

Centroid-based clustering methods create a partition of the dataset into k clusters, in which each cluster is represented by a center vector. The assignment of the data points to specific clusters is then induced by the respective center with the smallest distance, which results in the necessity to specify the number of clusters k in advance. To introduce a data dependency, centroid-based clustering can be formulated as an optimization problem with respect to the minimization of an error function. However, the known NP-hard complexity [77] of this optimization problem limits practical methods to only being able to approximate an optimal solution. For the most popular clustering method of this group, the k-means algorithm, the minimization is performed with respect to the sum of squared distances of the data points to the cluster centers. A first version of the k-means algorithm, known as the Lloyd's algorithm [128], was introduced for quantization purposes in pulse code modulation (PCM). Starting with a set of k randomly chosen cluster centers, the algorithm works by assigning the data to the centers and then redefining the centers based on the data assignment. Although most k-means variants use the average cluster coordinate for center refinement, versions with other methods, like the k-medians [34] and k-medoids [88] algorithm, exist. The steps of data assignment and

refining the centers are the repeated until convergence is archived. Consequently, the running time of the algorithm is $O(nkdi)$, with n being the number of points, d being the number of dimensions, k being the cluster count and i being the iteration count until convergence [87, 99]. An important factor to influence the speed of convergence and the quality of the result is the initialization of the cluster centers. This seeding problem is targeted by the k-means++ algorithm [11] by trying to enforce a good data coverage of the initial centers. For this purpose, the first center is chosen randomly from the data points. The remaining centers are then selected according to a probability function proportional to the squared distance to the closest center already created. Despite the additional overhead for selecting the centers, this positively affects both the speed of convergence and the quality of the solution that has been shown to be $O(\log k)$ approximative in expectation. As the k-means algorithm produces a Voronoi-cell partition of the data, the resulting clusters tend to exhibit a comparable size. Although the general shape of extracted clusters may be arbitrary, the convex nature of Voronoi cells prevents intertwined convex and concave features of neighboring clusters. Regarding the wedge face extraction problem, the compactness of the produced clusters, the comparatively low running time and the production of a partitioning cluster set without outliers is advantageous. In the general case, the unknown number k of clusters may be an issue. A popular approach for trying to estimate an appropriate k is the so-called elbow approach [215], where the clustering is repeated for different choices of k. The value of k is then derived from an elbow shaped angle in the function graph of variance values for the clusters.

Initial experiments to chose an appropriate clustering algorithm for the wedge face extraction problem were performed using the ELKI framework [193]. This framework integrates a large variety of cluster algorithms, including DBSCAN, OPTICS, SLINK, CLINK, k-means, and EM clustering. Samples were produced by taking the set of triangle normals for several wedge areas, which were extracted using the method described in section 4.7, and reapplying the clustering result in form of a color coding onto the original triangle mesh data. The quality of the clustering was then determined via a manual visual inspection of the clustering results. Evaluation criteria included the concordance of the extracted clusters with the real wedge faces, the straightness of the polygonal lines formed by the contacting triangle edges between the clusters, the compactness of the clusters on the mesh surface and the number of cluster components produced on the mesh. Regarding the cluster concordance with the real wedge faces, the clusters should be shaped in a way that a cluster is formed by all triangles belonging to a patch of geometry formed by the same side of the writing stylus upon one impression. The straightness of cluster contact lines is an important criterion, as it reduces the data variance during the subsequent wedge edge extraction. In addition to that, compact clusters and a low number of clusters are desirable as these criteria minimize over-segmentation, and thus, the need to fuse surplus clusters in a post-processing step. The actual number of required clusters depends on the amount and type of wedges in the considered surface patch. While a triangle patch containing two wedges with largely differing orientations may require six clusters to correctly differentiate the wedge faces, a patch with only one or multiple parallel wedges only



(a) DBSCAN

(b) OPTICS

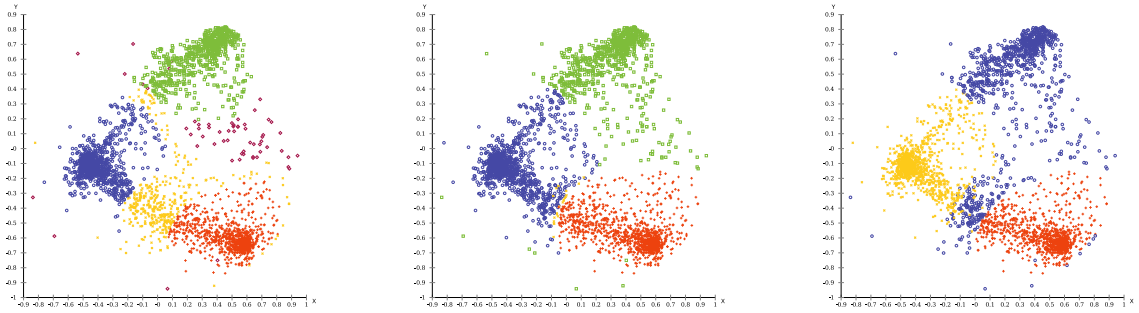
(c) EM

(d) k-means

Figure 4.29: Projective visualization of triangle normal clustering results within a wedge, with DBSCAN **(a)**, OPTICS **(b)**, EM-clustering **(c)** and k-means clustering **(d)**.

requires three clusters. In the latter case, parallel wedge faces are not separated clustering-wise as the triangle-normal-based clustering can only account for the triangle orientation but not for the spatial position on the mesh.

Some exemplary clustering results for using different clustering methods on the wedge geometry depicted in figure 4.28d are shown in figure 4.29. The DBSCAN result in figure 4.29a clearly shows the expected difficulties of DBSCAN to adapt to varying densities without sharp density gradients. Although the hand optimized parameters of $\epsilon = 0.1$ and $\text{minpts} = 100$ were able



(a) minpts = 400 **(b)** minpts = 500 **(c)** minpts = 600

Figure 4.30: Projective visualization of triangle normal clustering results within a wedge, with OPTICS and different values for minpts of 400 **(a)**, 500 **(b)** and 600 **(c)**.

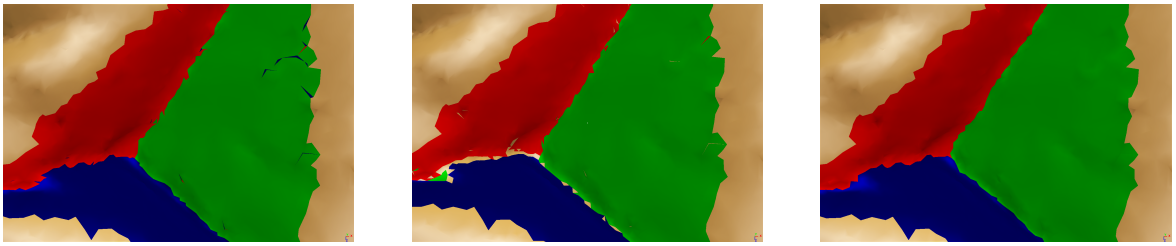
to produce separate clusters (red, blue and green) for the central areas of the wedge faces, the triangle normals belonging mostly to the wedge edges are classified into one separate cluster (purple) and noise (yellow). This clustering, although stable for a set of parameters ϵ and minpts, is highly sensitive to parameter changes. Deviations from the above mentioned parameter values quickly lead to multiple small and dense clusters, embedded in a big cluster of low density, representing noise. Alternatively a variety of long and narrow shaped clusters is produced that include triangle normals from multiple wedge faces. In addition to that, the general characteristics of the clustering resulting from a set of parameters are not consistent across multiple wedge areas even when considering only those that contained only one wedge. Especially the classification of large parts of the dataset as noise contradicts the intended partition of the dataset into clusters, representing the faces. These issues make DBSCAN an unsuitable candidate for wedge face extraction.

As expected, a parameter optimized clustering using OPTICS is more capable of adapting to the density variations of the considered triangle normal distribution. The OPTICSxi result shown in figure 4.29b is obtained using $\epsilon = 0.1$, minpts = 500, xi = 0.005 and a squared Euklidean distance function. Except from the small yellow noise cluster, the clustering results show a good correspondence with the actual wedge faces. However, the parameterization exhibited similar issues as the parameterization of DBSCAN, namely a delicate choice of parameters to obtain a usable result and an inconsistent behavior of parameters across multiple wedge areas. As even small parameter changes had significant effects on the shape and the number of clusters, this could not be counteracted with discarding small clusters. Especially the choice of minpts is critical, as too big values can easily lead to large portions of the dataset being classified as noise. Raising minpts to 550 leads to the complete green cluster to be classified as noise. An example for the effects of parameter variation on cluster count and shape is shown in figure 4.30. The respective noise clusters are colored yellow in figures 4.30a and 4.30b and blue in figure 4.30c.

A clustering result obtained using EM-clustering on a Gaussian mixture model with parameters of $k = 3$ and delta $= 0.001$ is shown in figure 4.29c. Unlike the DBSCAN and OPTICS results, the number of clusters k is a predetermined input parameter. As expected, the utilized multivariate Gaussian mixture model has a tendency to adapt to the elongated more dense sections of the distribution. This results in an unbalanced clustering of the wedge faces, with a biased assignment of wedge edge areas to only one wedge face. This undesired effect is predominantly a result of limiting the number of distributions to 3. However, raising the number of distributions to 6 is not able to overcome this issue. Even though the central areas of the wedge faces are then represented as small clusters, like with the DBSCAN method, the result includes three additional clusters of varying size that are largely overlapping with the distributions of other clusters. This is an issue, as it leads to strong cluster fragmentation on the triangle mesh. Using a spherical Gaussian model does not solve the issue either, as spherical models produce extremely overlapping distributions with cluster assignments that do not match the wedge areas at all. In addition to that, the EM-clustering results show slight stability issues regarding the assignment of data points to the clusters for repeated clustering using the same parameters. This instability originates from the random choice of model parameters as the starting point for EM-clustering.

An exemplary result for k-means clustering using Lloyd's algorithm and $k = 3$ is shown in figure 4.29d. The depicted clusters are of a similar size and exhibit a balanced normal assignment in areas, corresponding to the wedge edges. This is due to the fact that the wedge faces are usually represented by a comparatively large number of triangles with a similar orientation, which, in turn, draws the cluster centers to the average positions of these regions in the distribution. The cluster centers are therefore very likely located close to the average normal direction of the wedge faces. This leads to the observed clustering result, since the characteristic Voronoi cell like cluster structure of the k-means clustering is directly induced by the cluster centers. As a consequence, the k-means approach naturally produces highly suitable clustering results for the wedge face extraction problem. This behavior adapts well to most wedge areas containing a single wedge and to many wedge areas containing multiple wedges, as will be shown later. However, the applicability of the assumption of $k = 3$ necessarily depends on good results in wedge area extraction. An additional issue can arise in cases, where the input mesh was processed with a strong amount of curvature-dependent adaptive remeshing. This can lead to a significant reduction of the numbers of triangles on the wedge faces and thus to a shift of the cluster centers towards the wedge edges. As a countermeasure, the re-computation of cluster centers can be weighted with the triangle area associated with each normal.

Despite the general issue of determining a suitable number of clusters, the most promising results are obtained using the k-means algorithm and assuming $k = 3$. This is possible due to the good extraction results of the preceding wedge area extraction which, with a high probability, produces triangle patches corresponding to single wedge imprints. The data partitioning property of the Voronoi cell induced k-means clusters results in a clean separation



(a) EM **(b)** OPTICS **(c)** k-means

Figure 4.31: Detail visualization of clustering results on the mesh for EM-clustering **(a)**, OPTICS **(b)** and k-means clustering **(c)**.

of clusters in normal space. Transferred to the mesh surface, the clusters show a good coverage of wedge faces with a mostly clean separation at the wedge edges. However, it is important to consider that one cluster in normal space can correspond to multiple mutually unconnected triangle patches on the mesh. Especially wedges with complex features and high resolution triangle patches are affected, from this so-called cluster fragmentation. This is an issue, since the wedge face extraction aims to extract one connected triangle patch for each of the expected three wedge faces. As a countermeasure, the clustering results on the mesh are processed with a connected component analysis to identify connected triangle patches of the same cluster id. A typical clustering result then contains one large triangle patch for each wedge face that covers the large majority of the wedge face area, and several smaller triangle patches scattered throughout the wedge area. The small cluster fragments are mostly caused by damaged wedge areas, impurities or multiple wedges located in the same extracted wedge area. As they exhibit usually a much smaller size than the main cluster patch representing the wedge face, the clusters are filtered depending on their surface area and all cluster fragments except of the three biggest ones are discarded. Although this approach automatically discards many of the small geometry patches detected during mesh error detection, not all of these patches show a sufficient normal direction deviation from their surroundings to be classified into separate cluster fragments. Therefore, the mesh error detection remains important for sorting out such kinds of geometry.

Figure 4.31 shows an example for the advantageous clustering quality of k-means clustering in contrast to EM-clustering and OPTICS for the considered application scenario. Even on this large, well-defined and mesh error free wedge, the clustering result of k-means has significantly cleaner cluster borders and nearly no cluster fragmentation. EM-clustering shows a high susceptibility to micro-fragmentation, the occurrence of small cluster fragments with only one or two triangles, surrounded by other clusters, like at the upper-right border of the green cluster in figure 4.31a. Especially the OPTICS result suffers from gaps between the clusters in the regions of the wedge edges, which complicates further processing. On smaller, less clearly defined wedges and wedges including holes and non-2-manifold geometry, the advantages of k-means clustering become even more apparent.

Employing k-means clustering in the form of Lloyd's algorithm to the wedge face extraction problem yields fluctuating triangle assignments at the cluster boundaries, when applying the algorithm multiple times to the same data. This non-deterministic behavior is connected to an inherent instability of Lloyd's algorithm which results from the random initialization of the centers and the algorithm getting stuck in locally minimal solutions which can be arbitrarily bad [103]. Moreover, the aforementioned NP-hardness of the underlying centroid clustering problem restricts practical methods to finding an approximation of the globally optimal solution. A better approximation can therefore not only be expected to improve the quality of the solution but also lead to fewer fluctuations in cluster assignments and consequently increase the reproducibility of the results. An improvement of the approximation quality is, however, expected to come at the expense of a compromise in running time. As the considered cuneiform tablet fragments may contain thousands of wedges, a trade-off between clustering quality and running time has to be found. Therefore, two approaches to compute a more stable k-means clustering solution were evaluated.

The first method was introduced by Kanungo et al. as a local search approximation algorithm for k-means clustering that yields a $(9+\epsilon)$-approximation [103]. The algorithm relies on performing a local search by temporarily swapping several cluster centers with a selection of candidate centers to move out of local minima. The quality of the solution is judged by computing a so-called distortion term for the current centers and the data points. For this, the distortion is defined as the sum of squared distances of the data points to their respective closest centers. Since this swapping heuristic can efficiently move out of a local minimum but only converges slowly into such a minimum, the approach is combined with an iterative version of Lloyd's algorithm that can efficiently converge into local minima. The two methods are then used in an alternating way to generate a practically usable algorithm from the swapping heuristic. In addition to that, several simplifications are applied to improve usability. This includes the data points being used as swapping candidates and performing the swaps one by one. The utilized variant of the Lloyd's algorithm is applied in several stages to be able to pick the best solution. It starts with randomly sampled centers and is repeated until the improvement in distortion falls below 10% within three stages. Then, the next swapping stage is performed until a predefined number of stages is exceeded. An implementation of this hybrid k-means approach is available via the kmlocal c++ library [147] which also includes simulated annealing [186]. In kmlocal, simulated annealing is used to control the acceptance of solutions and to decide when to switch from Lloyd's algorithm to the swapping heuristic. This includes that solutions can also be accepted without being an improvement over previous solutions. A more detailed description of the algorithm can be found in the kmlocal documentation [147].

In contrast to Lloyd's algorithm, the application of the hybrid local search k-means algorithm to the wedge face extraction problem resulted in a significant increase of the solution consistency when repeatedly executing the clustering on the same dataset. Especially on small wedges represented by only a small number of triangles and on wedges containing extensive mesh damage, the results also exhibited a higher quality. Quality, in this case, refers to the

concordance of a wedge face extraction result with the expectation of a cuneiform expert. This subjective evaluation was necessary for optimizing the kmlocal parameterization as no objective quality evaluation method was available. The parameters for kmlocal consist predominantly of termination criteria for the three algorithm stages, swapping, Lloyd's algorithm and solution acceptance with simulated annealing, which are serially executed in every run. As the most central parameters, for the Lloyd's stage, a maximum number of 200 random runs and a minimum accumulated relative distortion loss of 0.0125 for detecting a successful stage was used. The swapping stage was configured to perform a maximum number of 5 swaps per run. For the simulated annealing part, an initial probability of acceptance of 0.5 was used and the temperature was changed after 10 stages with a reduction factor of 0.975.

While kmlocal with the above-mentioned set of parameters resulted in good segmentation result ratings by the involved cuneiform experts, and a very good stability of the cluster assignments, the running time for wedge face extraction could be in the order of several minutes on larger cuneiform tablets. To increase the interactivity, a second more simple variant of k-means clustering was compared against these results. Considering that the normal cluster centers of a tetrahedron shaped indentation exhibit most likely a big mutual distance in normal space, the k-means++ method [11] seemed to be a good candidate for cluster center initialization. To counteract the instability issue of the random initialization of the first cluster center, the method was executed multiple times in succession while tracking the best rated clustering result. Using 5 repetitions with a maximum of 200 iterations for the Lloyd's algorithm, the quality and stability was perceived to be at a similar level as the kmlocal implementation. Besides, by using KMeansRex [97] as a vectorized k-means++ implementation, a performance increase could be achieved without observable compromises on the clustering quality. Depending on the dataset, the k-means++ approach performed 50% to 100% faster than the hybrid local search k-means.

When considering the performance on a single wedge, the k-means++ results are far more consistent than the hybrid local search results. This predominantly results from the random swapping and the simulated annealing controlled termination of the hybrid local search approach. Figure 4.32 shows an example of the observable performance variations. Although the kmlocal clustering performance on a single wedge can be easily cut in half, the variations even out when considering a large amount of wedges.

An evaluation of both approaches in terms of practical application does not yield a clearly superior approach. Both contestants provide an approximation guarantee that prevents the generation of arbitrarily worse results, as opposed to the original Lloyd's algorithm. The performance advantage of the k-means++ approach using the chosen set of parameters is significant for real time processing of large cuneiform fragments with many wedges. However, for small to medium-sized fragments, this performance increase is negligibly small and large fragments may also be batch-processed without being monitored by a user. In addition to that, the k-means++ performance heavily depends on the utilized repetition count, which has

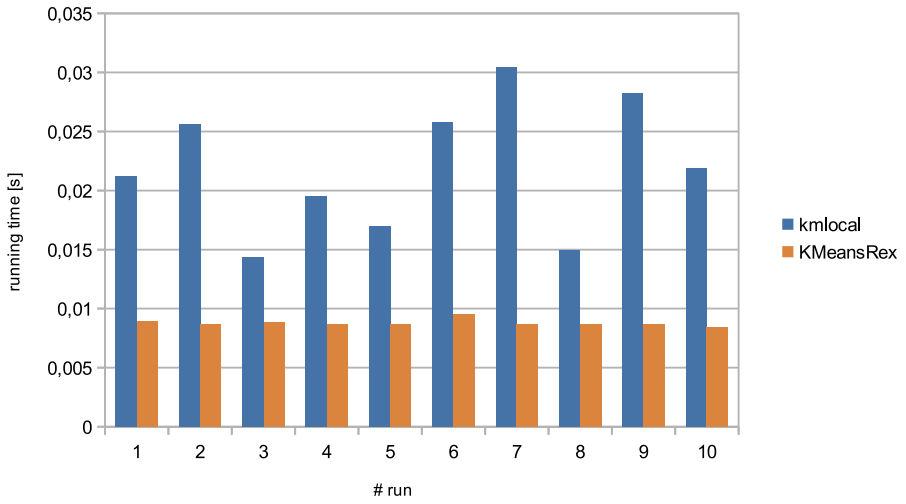
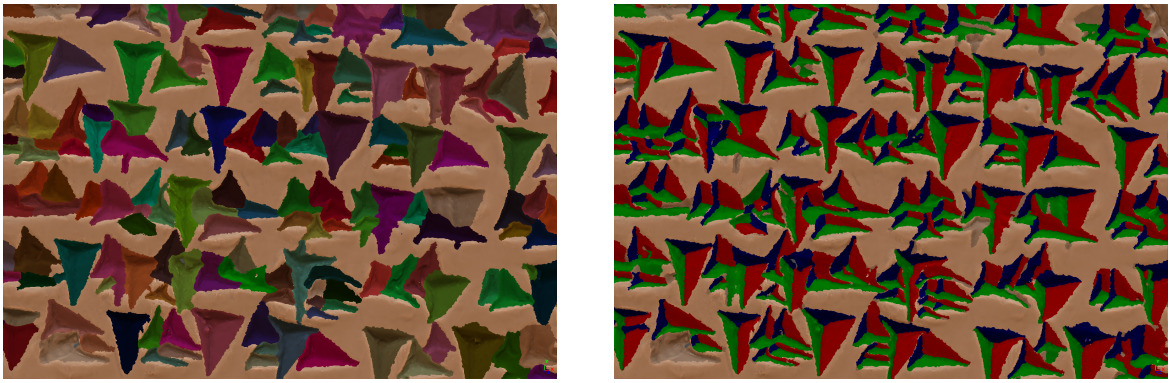


Figure 4.32: Performance comparison of k-means clustering approaches, executed repeatedly on a single wedge.

to be adjusted manually to result in similar clustering quality and stability as the hybrid local search approach. The fact that no automatic method is available to measure the clustering quality with respect to philological criteria, makes a precise quality comparison of the two approaches an issue. An advantage of the hybrid approach is that it explicitly contains a mechanism for leaving local minima. Therefore, this approach performs better on small cuneiform wedges with a low scan resolution or significant input noise and on damaged wedges. On the datasets used in this thesis, a quality advantage could only be observed on single wedges on a case-by-case basis but not when considering whole fragments with large amounts of wedges. To favor extraction quality over running time, the hybrid local search approach is used for wedge face extraction in this thesis.

A wedge face extraction result on a densely written section of a cuneiform fragment is shown in figure 4.33 with the wedge faces being colored red, green and blue according to the wedge model defined in section 2.2.3. At this point, the wedge orientation is usually still unknown, which results in a random order of the red, green and blue marked wedge faces. The orientation will be determined later, based on the properties of the extracted wedge model, once the model extraction is complete. In this case, the orientation of the wedges in figure 4.33b has already been corrected based on the information of the subsequent model extraction to present a more consistent and visually less confusing extraction result. This also includes the faces being sorted to match the clockwise order of the wedge model, which is red, green, blue. The slightly darker regions on the tablet surface, which are colored neither red, green or blue, as can be seen in the lower left corner of figure 4.33b, have been discarded upon filtering all but the three biggest clusters per wedge area.



(a) **(b)**

Figure 4.33: Result of the wedge face extraction on a section of cuneiform fragment *Bo4891*. For comparison, the starting point of the extracted wedge areas is shown in **(a)** with random color coding for the wedge areas. The corresponding wedge face extraction result in **(b)** is colored in red, green and blue, according to the wedge model.

While the extracted triangle patches for the wedge faces tend to exhibit comparatively smooth borders, they can be affected by spiky artifacts where one acute-angled triangle protrudes into a triangle patch of another wedge face. These artifacts can not only be caused by noisy meshes but also by a non-optimal mesh triangulation with respect to the mesh curvature. As these spikes, when consisting out of a single triangle, are most likely not representing a significant feature, they are removed to improve the smoothness of the cluster borders for the subsequent wedge edge extraction. For this, a triangle can be defined as a spike if it exhibits an acute angle of less than 90 degrees at one mesh vertex and the edges that form this angle each touch a triangle that has been assigned to a different cluster than the triangle itself. While this definition works on spikes located along a polygonal-line-shaped border of two triangle patches, it may affect the connectedness at locations where more than two differently clustered triangle patches are involved. It is therefore advantageous to ensure the mesh vertex at the tip of the acute angle also only touches triangles of one different cluster. This can efficiently be checked by circulating around each mesh vertex, looking for a single touching triangle with an acute angle that is assigned to a different cluster than all other touching triangles. Obviously, this test also includes checking the neighboring triangles of the spike triangle candidates and is therefor sufficient for removing spikes. Iterating the mesh vertices and filtering triangles depending on the outcome of this test may introduce an order dependency to the results in some cases. The effects of this order dependency have, however, been found to be negligible and the spike filter is able to remove nearly all spikes.

Figure 4.34 shows several more detailed extraction results for some of the complex wedge examples from figures 4.1a, 4.1b and 4.1d. Each example consists of a manually drawn wedge overlay with wedge numbers that visualizes the real locations of the involved wedges and a visualization of the wedge face extraction result. For the two largely overlapping wedges in

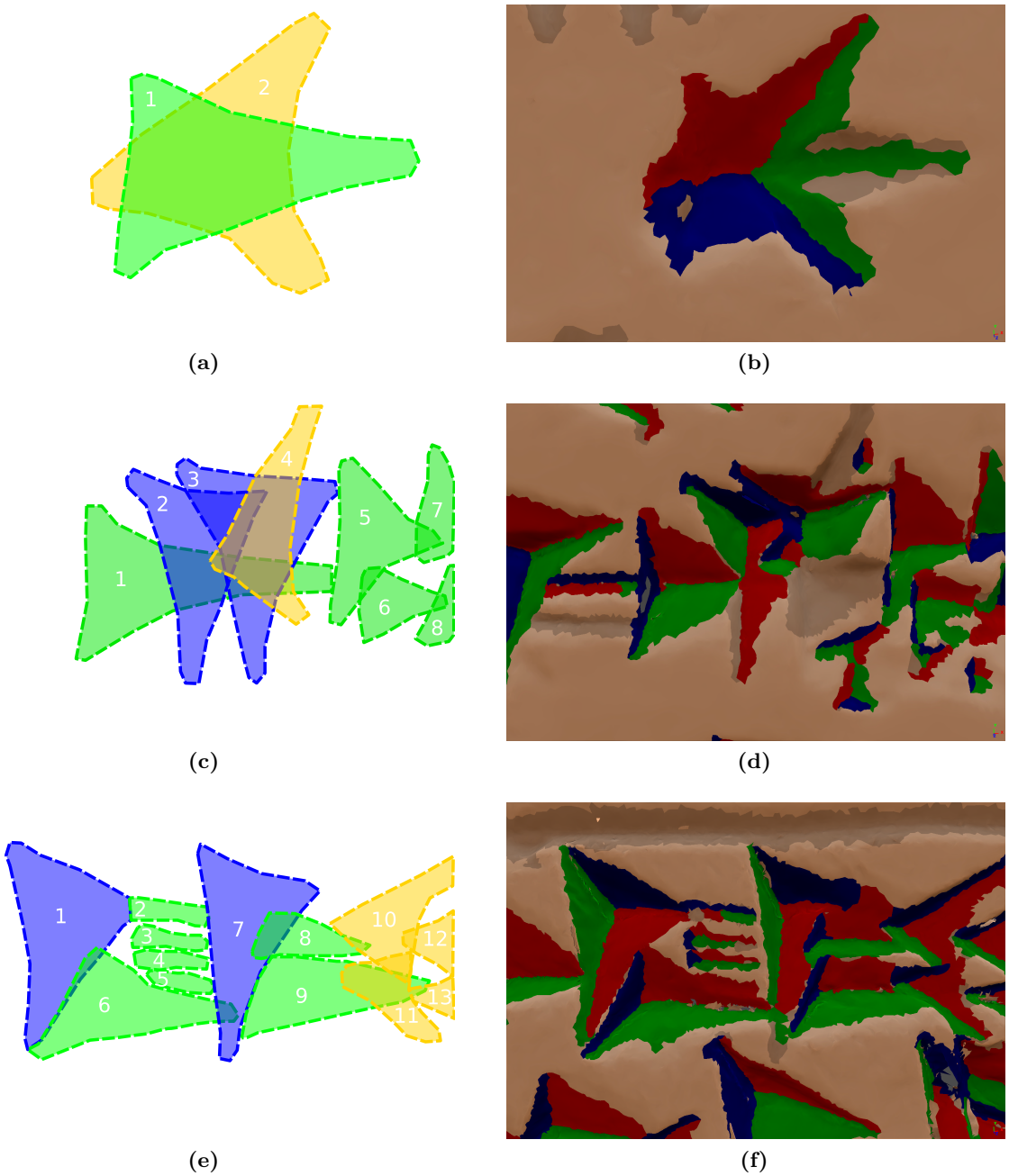


Figure 4.34: Result of the wedge face extraction on complex wedges. A wedge overlay for horizontal (green), vertical (blue) and Winkelhaken-type (yellow) wedges is shown on the left-hand side and the wedge face extraction result on the right-hand side.

figures 4.34a and 4.34b, the wedge area extraction could extract only one area for both wedges as they share the same depth point. However, the wedge face extraction still produces a usable result and at least extracts the faces of Winkelhaken-type wedge 2. Due to the cluster filtering, some triangle patches at the right side are discarded.

The constellation of superimposed wedges in the center of figures 4.34c and 4.34d consists of the horizontal wedge 1, the vertical wedges 2 and 3 and the Winkelhaken-type wedge 4. Due to the superimposed wedges 2, 3 and 4, only the right half of wedge 1 still has a geometric representation in the mesh. For this area, a correct wedge area and the wedge faces have been extracted. The original shape of wedge 2 is perforated by wedge 1 at the left side and by wedges 3 and 4 at the right side. While the perforations at the right side only decrease the area of the blue and the red wedge faces, the perforation on the left side divides the green wedge face into two patches. Although the smaller, lower patch is discarded by filtering, over 80% of the wedge faces have been successfully extracted. Despite the large missing areas caused by overlapping of wedges 1, 2 and 4 the upper part of wedge 3, including its faces, has also been successfully extracted. In this case the upper part of wedge 4 was assigned to the wedge area of wedge 3, but discarded by the cluster filtering, thus, exerting only a small negative influence on the extraction result of wedge 3. However, the missing lower parts of wedge 3, being partially discarded by the cluster filtering, are expected to cause significant inaccuracies during further wedge model extraction. Aside from its incorrectly assigned upper tip, the only geometric remains of wedge 4 is a small area between wedges 2 and 3. This wedge area has been successfully extracted, including the wedge faces. Although of small size, this extraction result will still allow determining the important depth point and a sufficient approximation of the direction of the wedge. The small patches on the right half of figure 4.34d, which are not present in figure 4.34c, result from surface damage artifacts, which are of similar shape like subtle wedges. These artifacts are extracted and treated as wedges until enough wedge model information is extracted to identify the false detection. Due to the similarity of some of these patches with real wedges, this is not always possible without incorporating philological knowledge about the usual shape of cuneiform signs. In such cases, an automatic detection of false positives is an issue.

Figures 4.34e and 4.34f show a wedge constellation with well extracted big wedges, including an array of very subtle horizontal wedges. The wedges 2, 3, 4 and 5 are barely resolved by the scanner and represent a limiting case of the wedge model extraction. While wedges 3 and 4 have been successfully extracted, wedge 2 is missing one wedge face that has been obstructed by wedge 1. As a result, the wedge face extraction for wedge 2 works but only yields a non-usable result. In addition to that, an extraction of wedge 5 did not succeed as it is missing a depth point with respect to the writing surface and its wedge area was included in the wedge area of wedge 6. Although subtle wedges, like in this example, are at the limits of scanner resolution and can often only be seen with the help of optical enhancement techniques, the wedge face extraction can still handle many of these wedges. Determining meaningful extraction rates for these limiting cases is difficult as they vary largely depending on the

cuneiform fragment. Even for one fragment, the required manual evaluation would be very time-consuming and not be transferable to other datasets. The extraction rates are, however, sufficient for specific tasks of statistical cuneiform script analysis, as will be shown later.

4.8.2 Wedge edge and feature point extraction

The amount of information extracted up to this point, namely the wedge areas and an unordered assignment of the contained triangles to different wedge faces, provides the base for extracting the components of the wedge model, introduced in section 2.2.3. The wedge edges and the feature points are extracted simultaneously because they are mutually dependent.

Although the extraction of the wedge model can be seen as a general shape registration problem, the extraction method introduced here will use a more component oriented approach. This approach aims to construct the model in multiple consecutive stages rather than iteratively fitting the whole model, like when using an Iterative Closest Point (ICP) [19] related method. The reasons for this decision are partially connected to the fact that ICP-based methods were originally used for the registration of rigid, similarly dense point sets. While a suitable point set naturally exists for the extracted wedge areas, an appropriate point set for the wedge model would have to be constructed first. This model would have to match the sampling density of the wedge areas that may vary largely on a per cuneiform fragment basis and also, depending on the scanner pipeline used for data acquisition. As the exact shape of the examined wedges is unknown in advance, the point set derived from the wedge model would also have to exhibit a non-rigid construction or update mechanism. Existing non-rigid ICP methods may need sparse markers, landmarks or complex shapes with distinctive, rotationally non-symmetric features to yield usable results [211, 226], all of which are not always available on incomplete and damaged cuneiform wedges. ICP approaches with more sophisticated and flexible methods for correspondence search are often comparatively slow [126]. In addition to that, ICP methods tend to yield only locally optimal results, which emphasizes the requirement of a good starting point for the iterative alignment process. As the quality of an ICP solution can be evaluated by means of global distance or energy function [5, 8] on, for example, single or bidirectional point correspondences, there is usually no possibility to compute a partial extraction result. Instead, the final result is determined by employing a threshold condition to ensure the termination of the iterative fitting process. This condition may be coupled to a fixed number of iterations or a convergence of the result quality. The quality of the model fitting can then be evaluated as a whole via the distance function. Consequently, the output is a fitted model with an associated quality value or, if combined with a quality threshold, an accepted or discarded model. However, in the context of extracting cuneiform wedge models, even a partial extraction of only the depth point or the directional wedge can be a valuable extraction result. Aside from that, the definition of a suitable distance measure to describe the global quality of model alignment can be an issue, especially on geometrically incomplete wedges. An approach for component-wise model extraction is therefore an advantage.

The most important components of the employed wedge model are the depth point and the direction of the wedge edges. While the depth point can be used to pinpoint the location of wedges, the direction of the edges is essential for determining the type and the characteristics of a wedge. As the edges in the wedge model are defined as lines that connect the depth point with the tail, left, and right vertex, the extraction of these components is linked to each other. It will, however, be shown that the extraction accuracy of the direction of the edges is much more important than the length of the edges and thus, the exact position of the outer wedge vertices. The model extraction will therefore initially approximate the wedge edges and successively introduce constraints to determine the depth point and the three wedge vertices.

As a first step, an approximation base for the wedge edges is derived from the wedge areas extracted in section 4.7, which are annotated with the wedge face assignments from section 4.8.1. For this purpose, three sets of points are determined that are located at the mutually contacting interfaces of the wedge faces. As a specific assignment of the wedge faces to the top, left and right face is not yet available, in the following, the faces will be referred to as the red, green, and blue wedge face based on their visualization in figures 4.31 and 4.33b. To determine the three edge data point sets P_{rg}, P_{gb}, and P_{br}, the set of mesh vertices of an extracted wedge area is cycled and a mesh vertex is added to one of the sets if it is adjacent to the respective combination of two wedge faces. Mesh vertices which are adjacent to red, green and blue triangle patches are added to all three sets. In an optimal case, the resulting point sets are expected to be located on a comparatively straight polygonal line between the triangle patches of two wedge faces and to extend from the depth point to the top, left and right vertex of the wedge. In a non-optimal case, each point set may be located on multiple fragments of non-smooth and curved polygonal lines. This fragmentation of lines may be caused by holes in the mesh and by cluster fragments filtered during wedge face extraction. Several examples for point sets generated in this way are shown in figure 4.35. According to the wedge model, where opposing pairs of faces and edges share the same color, the points are colored in blue for P_{rg}, in red for P_{gb} and in green for P_{br}. As figure 4.35a includes wedge areas of five wedges and edge data points are only generated on the wedge face transitions within each wedge area, some wedge face transitions do not generate point sets.

Based on the three point sets per wedge, a first approximation of the wedge edges is computed to determine the depth point. Considering the possible non-straight, branched and fragmented characteristic of the polygonal lines induced by the edge candidate point sets on the mesh, an approximation of the wedge edges by using a polygonal simplification method like the Douglas–Peucker algorithm [58] or a least squares approximation will only work to a limited extent. This results from the extraction objective to reconstruct a linear approximation of the wedge edges without disturbances caused by impurities, surface damage or the slightly curved transition of the edges near the depth point and the writing surface. Therefore, the Random Sample Consensus (RANSAC) method [69], a robust model fitting method that can handle outliers, is employed to fit a line into the point sets. Within the RANSAC method, the model is fitted to the dataset using a random set of model parameters, in this case, two points. The

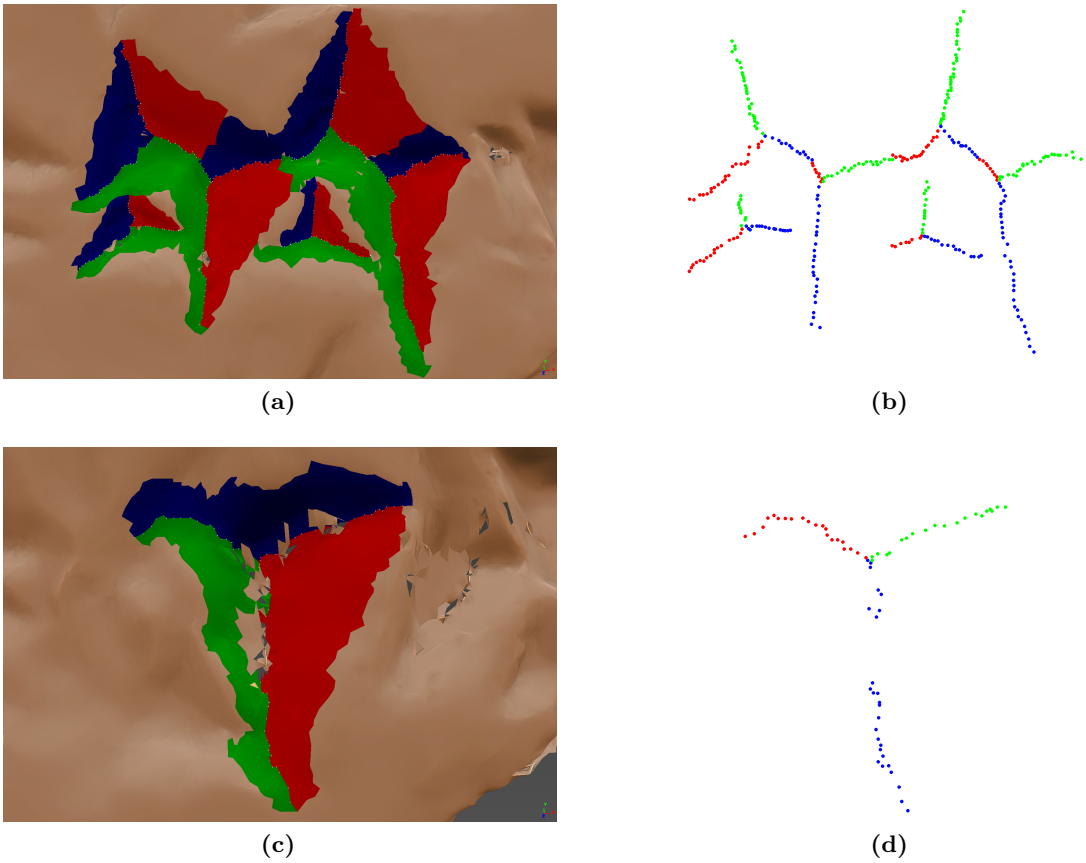


Figure 4.35: Several examples of edge data point sets for five connected wedges (top row) and a wedge with large mesh errors (bottom row). The results of the wedge face extraction are shown on the left-hand side and the resulting data point sets on the right-hand side.

dataset is then evaluated to what extent it supports the chosen model according to a specified loss-function. The result are two subsets of points, one subset that supports the model, the so-called consensus set, and one subset that does not support the model, the so-called outliers. This process of choosing and evaluating a model is repeated until a sufficiently large consensus set is found. The model parameters can then be improved by fitting the model into the largest identified consensus set using least squares, which yields the final approximation. If no consensus set of the required size is found, the model can not be fitted successfully. Improved versions of the RANSAC method include M-estimator Sample and Consensus (MSAC) [219] and Maximum Likelihood Estimation Sample and Consensus (MLESAC) [219] which can be used to counteract instabilities occurring with an inappropriately chosen outliers threshold.

For the wedge extraction problem considered here, the RANSAC method will output three lines on success. Due to inaccuracies in the scanned data and non-ideal wedge shapes, these lines are very unlikely to all intersect in the depth point, as required for the employed tetrahedron shaped wedge model. Therefore, the depth point is derived from the approximated edges in multiple steps. A preliminary estimation of the depth point is produced by finding a point

with a short distance to all three wedge approximation lines computed as described above. This is done by computing the mean value of the midpoints of the shortest connection lines between all pairs of edge approximation lines. As each such midpoint is the closest point to one pair of edge approximation lines, the resulting point is reasonably close to all three lines and should also be located near the real depth point.

In this state, it is possible to take into account that cuneiform wedges are in many cases partially filled with small amounts foreign materials like soil, resulting in a rounded off shape near the depth point. To improve the approximation of the depth point in these cases, the RANSAC approximation of the edge lines is repeated, with the important modification to ignore some mesh vertices of the point sets, which are located closer to the preliminary depth point than a specified threshold. This way, the rounded off geometry near the depth point gets excluded from the approximation and the resulting depth point moves further into the tablet surface and closer to its original position. As this correction does not exert a significant negative effect on the approximation of the lines for non-filled wedges, it can be safely applied to all wedges. To enforce the required intersection at the depth point, the three edge approximation lines are then recomputed by using the depth point as a fixed model parameter during RANSAC. This also includes to enforce the included least squares fitting to run through the depth point.

The resulting difference regarding the position of the depth point when applying the proposed filled wedge correction is shown in figure 4.36. As an abstraction for better visualization, the figure only considers two point sets P_{rg} and P_{gb} of mesh vertices, visualized on an intersection plane perpendicular to the writing surface. The dashed black lines represent the result of the first RANSAC edge approximation, which yields the green dot as a preliminary depth point estimation. This depth point and a predetermined distance threshold, induce an exclusion zone near the wedge center. Vertices in this zone are not included in the subsequent second RANSAC approximation of the wedge edges, which are shown as solid black lines. The yellow dot representing the final depth point is then derived via the midpoint averaging method described above. As intended, its position is deeper with respect to the writing surface than the preliminary depth point and captures the real position of the hypothetically non-filled wedge much better. The more pale colored red and blue vertices near the writing surface represent points that have been classified as outliers in the second RANSAC line fitting, which contributes to the steeper course of the second set edge approximation lines. In figure 4.36, both depth points are drawn at the 2D intersection points of the respective sets of edge approximation lines, because the shortest connection lines between the edge approximation lines in 3D are projectively reduced to these points.

The value of the exclusion distance threshold should be chosen so that there is still a sufficient number of mesh vertices for fitting the edge approximation lines. As the mesh resolution may vary depending on the curvature and the size of the wedges, the threshold is chosen relative to the size of the wedge edges. To compute a preliminary approximation of the wedge edge size,

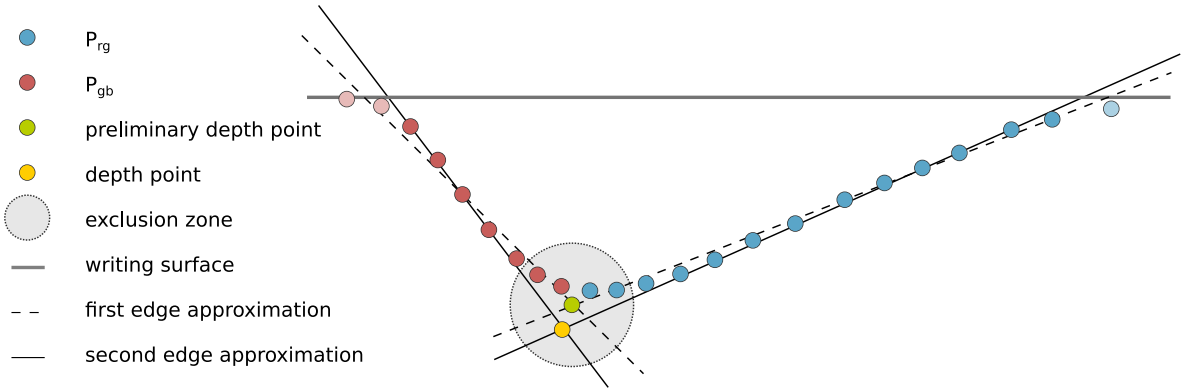
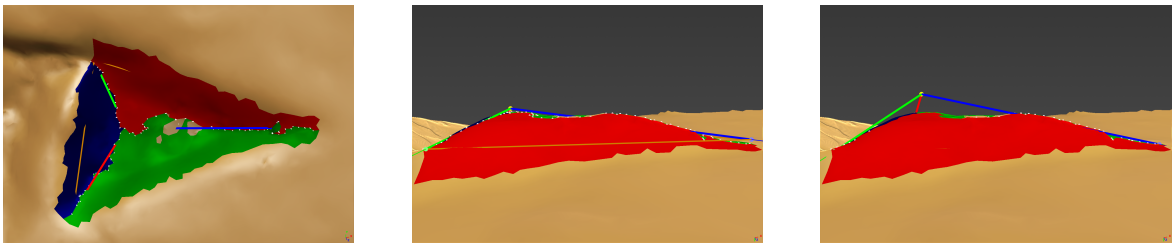


Figure 4.36: Correction of the depth point approximation on a partially filled wedge, illustrated on an intersection plane perpendicular to the writing surface for an isolated view the point sets P_{rg} and P_{gb}.



(a) **(b)** **(c)**

Figure 4.37: Example for the effect of the depth point correction on a filled wedge. A top view of the segmented wedge is shown in **(a)** complemented by a sideway view from below the writing surface, without the correction **(b)** and with the correction applied **(c)**.

the data points are projected on the fitted line and the distance of the projected points that are the furthest apart is used. This also accounts for possible big variations in the lengths of the three edges within one wedge. A value of 0.1 times the wedge edge length worked well in most cases. To support varying wedge characteristics, the threshold can be adapted manually on a per wedge level.

Figure 4.37 shows an example for the effect of the depth point correction on a filled wedge. The wedge edges without the correction in figure 4.37b are much to shallow with respect to the writing surface. In contrast, the edges in figure 4.37c follow the real edge directions in the non-filled part of the wedge much better and intersect at a more plausible location for the assumed non-filled wedge. It is worth mentioning that the wedge shown here exhibits a substantial amount of filling. The usual amount of filling is significantly more subtle. In addition to that, the wedge model depicted in figure 4.37 is already colored according to its orientation that will be determined later.

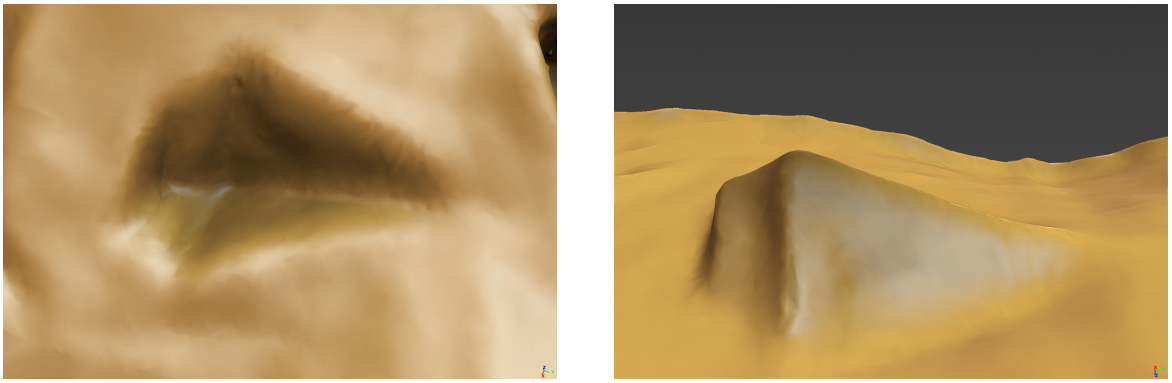
The extraction state of the wedge model at this point consists of the depth point and the directions of the three wedge edges. Therefore, only the computation of the tail, left, and

right vertex remains. A very straight forward approach for this would be to intersect the edge approximation lines with the Poisson mesh of the approximated writing surface as computed in section 4.5.2. While this produces good results on clean and undamaged wedges, it does not necessarily work on wedges, where the edges are partially superimposed by other wedges. In these cases, wedge edges may be represented by only a very small piece of mesh geometry, resulting in a significant precision loss, when intersecting the respective edge approximation lines with the writing surface to determine the exit point. Figure 4.38 shows an example of two superimposed horizontal wedges where the right wedge not only removes a large part of the geometry of the left wedge, but actually changes the direction of one edge so that this edge exhibits a negative slope with respect to the writing surface. An intersection point for the associated edge approximation line with the writing surface would not show any correlation with the generating edge of the writing stylus upon impression. Therefore, the wedge vertices are computed by projecting the points of the final RANSAC consensus sets onto the respective lines and using the projected points that are the furthest away from the depth point as the wedge vertices.

This solution has the advantage of the computed outer wedge vertices almost always lying within the boundaries of the wedge areas that represent the underlying wedge geometry. One side effect of the approach is that the extracted tail, left, and right vertices are most certainly positioned below the writing surface, even on non-superimposed wedges. This is a result of the edge data points in the smooth transition zone from the wedge edges to the tablet surface, like the pale colored points near the writing surface in figure 4.36, most likely not being included in the final RANSAC fitting. Although this initially appears to be a disadvantage, it is an advantage because analyzing the resulting distance of the outer mesh vertices allows detecting superimposed wedges. For example, an outer wedge vertex that is further away from the tablet surface than 0.25 times the wedge depth, is a comparatively certain indication for a superimposed wedge.

It has to be noted that the computation of the outer vertices differs from their definition in the wedge model from section 2.2.3. There, the vertices were defined as being positioned at the intersection points of the wedge edges with the surface. However, this definition was made, without explicitly considering non-ideal and superimposed wedges and must be adapted according to the above mentioned practical requirements.

The required parameterization of the RANSAC method in terms of the loss-function and a threshold to define the required size of the consensus set are determined data dependent, like the exclusion distance threshold. This is necessary since the absolute sizes of the cuneiform wedges and the relative sizes of the wedge edges may vary significantly. On cuneiform datasets with data characteristics like the ones used in this thesis, the sizes of the candidate point sets for edge fitting have been observed to mostly assume values in the range between 0 and 100. However, as the normalized histograms of candidate point set sizes in figure 4.39 show, the distribution of sizes is not uniform, but similar to a Gaussian mixture distribution. This is



(a) **(b)**

Figure 4.38: Example of two superimposed horizontal wedges shown from the top **(a)** and sideways from below the writing surface **(b)**, where the wedge on the left side has a negative edge slope on the shortened horizontal edge.

expected because the sizes of all wedge edges depend on a small set of similar target wedge sizes used by the cuneiform scribe upon writing. The resulting distribution of edge lengths is influenced by the sampling of the wedges from the 3D scanner when using the size of the point sets as an approximate measure for the edge lengths. For cases where the scanning resolution was chosen to be only slightly higher than necessary to capture the wedges, the distribution is expected to have its maximum value at a comparatively small point set size. In addition to that, the interval of the distribution on the left-hand side of this maximum is expected to be compressed, when compared to the right-hand side. The effects of both assumptions can be observed in figure 4.39 on two exemplary cuneiform datasets. For the cuneiform datasets acquired with the usual scanning procedure of the Hethitologie-Portal Mainz, a distribution could be observed with over 85% of point set sizes falling into the range between 5 and 40, with a peak around 10. The big amount of small-sized point sets indicate that there is only little headroom for outliers in many cases. Consequently, the minimum size of the consensus set should be kept as large as possible, while still allowing edge damage and mesh errors to be classified as outliers.

A minimum consensus set size of 75% with respect to the size of the edge data point set has been found to produce good quality edge fitting results on a large number of cuneiform fragments. The loss function to determine the consensus set is defined using a distance-based threshold relative to the average edge length of the underlying triangle mesh of the extracted wedge area. This takes into account that potential sampling related issues on the mesh are closely related to the sampling density, which, on a triangle mesh, is related to its average edge length. The quality of the line fittings was then evaluated on the resulting consensus set by using the sum of point distances to the model for choosing the best fitting set of model parameters.

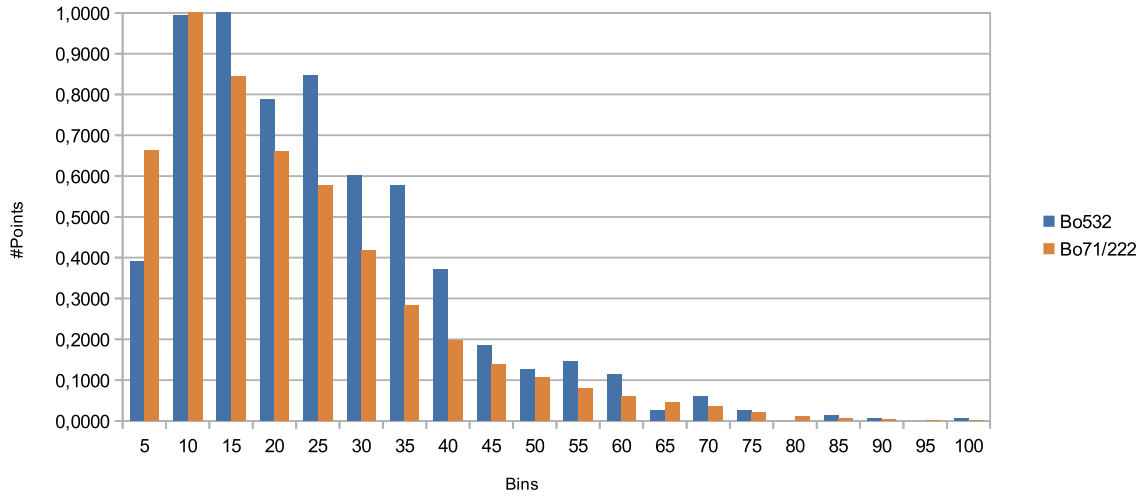


Figure 4.39: Normalized histogram of candidate point set sizes for edge fitting on two representative cuneiform tablets *Bo532*(blue) with 953 point sets and *Bo71/222*(red) with 9648 point sets and a bin size of 5.

Since no suitable automatic quantitative quality measurement method was available, the general suitability of the extraction results was assessed manually by checking whether the results were in line with a cuneiform expert's expectations. Aside from the subjectively perceived fitting quality of the model, this brought up the repeatability of the results as a conflicting quality criterion. The random choice of model parameters during the RANSAC method may lead to slightly differing fitting results, when repeatedly applying the method to the same data. This affects the reproducibility of the results and causes all dependent data to exhibit a small amount of extraction noise, which results in slight changes in the positions of the wedge vertices. In rare cases, the noise may also lead to differences in the acceptance of the evaluated RANSAC model parameters.

The amount of noise, in terms of the probability that two consecutive model fittings produce the same result, can be lowered by increasing number of RANSAC iterations or by increasing the threshold of the employed loss-function. More iterations increase the size of the set of model parameters to choose an optimum from, which consequently increases the chance to sample the same optimum. The threshold of the loss-function on the other hand is usually chosen to be two times the average edge length of the triangle mesh in the respective wedge area. This ensures that all points of the consensus set are comparatively close to the line to be fitted, while at the same time allowing a suitable degree of deviation, depending on the local triangle mesh resolution. Allowing some deviation is not only important to account for mesh errors and sampling effects but also for being able to fit curved wedges at all. Choosing an inappropriately high threshold will result in the RANSAC method to behave like a simple least squares fitting since the consensus set then includes all edge data points. While this

has a positive effect on the reproducibility of the results, the line fitting then is no longer robust against random impurities, located in the wedge edges. Also, the resulting edges tend to exhibit a lower slope, as the data points in the transition area from the wedge edges to the tablet surface are then always included in the line fitting.

The specific requirements to the behavior of the edge and feature point extraction do not only depend on data characteristics but also on individual preferences of the cuneiform expert conducting an analysis. Therefore, the parameters for the RANSAC line fitting and the depth point correction approach are normally used with the default values described above, but are also user configurable on a per wedge basis. This way, a user can input manual adjustments for unpredictable special features. However, non-uniform parameterizations should be avoided when conducting statistical evaluations on the resulting wedge data, as the differing extraction characteristics may, for example, introduce systematic errors into clustering results. To keep these kinds of errors detectable, the parameters of the wedge model extraction are stored on a per wedge basis within the generated meta-data upon each wedge extraction run. Therefore, the extraction parameters can still be accessed, after exporting the wedge model data for statistical analysis.

Several results of the wedge model extraction are shown in figures 4.40 and 4.41. Although large fractions of the wedges in figure 4.40a are affected by wedge superposition effects, the wedge model of all six considered wedges is properly extracted. This is possible, because all wedges still have a usable and separate geometric representation for their depth point. However, the overlapping wedges cause the affected edges to be shortened which also moves the respective feature points further away from the surface. This effect can also be seen in the pair of horizontal wedges in figures 4.40c and 4.40d. While the right wedge mostly retains its original shape in the extracted wedge model, the model of the left wedge is has a significantly shortened directional edge. In addition to that, a correct extraction of this pair of wedges is only possible, because the mesh of the approximated writing surface contains a small bump at the location of the wedges. This is essential, as the surface distance needs a local minimum for each wedge to separate the wedges. It can be easily seen that this can be an issue for wedges with a downward slope, when comparing the extraction result in figure 4.40d with the side view of the wedge pair in figure in figure 4.38b. Figures 4.40e and 4.40f show an extraction result for a larger portion of a densely written cuneiform manuscript. As this manuscript has an undamaged clean writing surface, a good wedge separation and only a small number of mesh errors, the extraction result can be regarded to be of high quality. However, it is not free from errors. While some contained errors are associated with wedge superimposition, the extraction result also includes a small number of missing wedges and false wedge detections in the geometry of the contained horizontal paragraph separator. While the former may cause issues during cuneiform sign retrieval tasks, it has only a small impact on the targeted statistical script analysis, as only a small number of wedges is affected. False detections are not an issue either, because they can mostly be detected using post-processing methods which will be described in section 4.9.

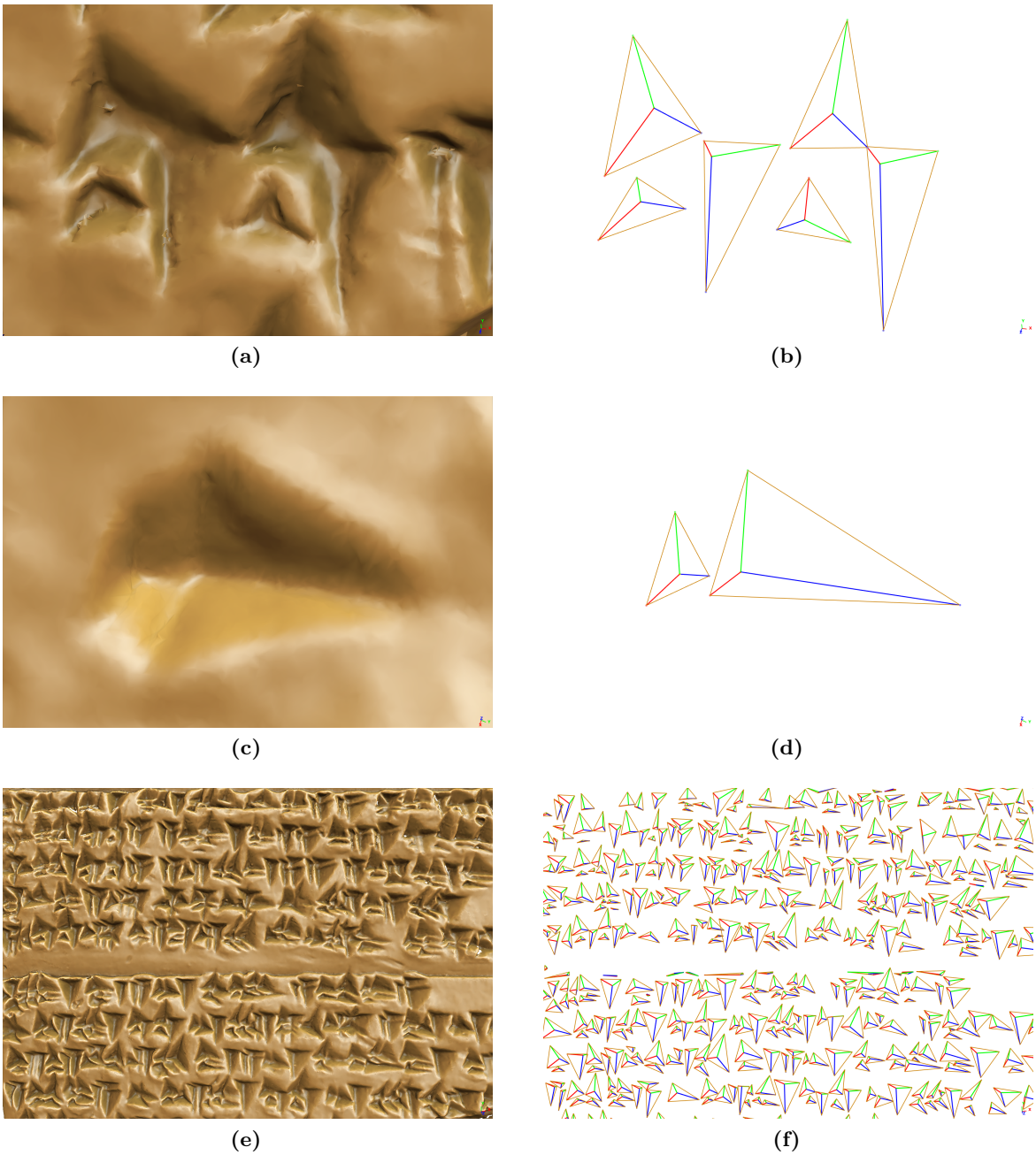


Figure 4.40: Examples of wedge model extraction results with the plain tablet surface shown at the left and the resulting wedge model shown on the right. While **(a)**-**(d)** show the results for the wedges from figures 4.35a and 4.38, **(e)** and **(f)** show an extraction result for a larger portion of a densely written manuscript.

To demonstrate the robustness of the approach against geometric issues of the underlying 3D-scanned mesh, figure 4.41 contains some detail images of an extraction result for a wedge affected by a very common type of mesh errors. While the wedge seems to be reasonably clean when looked at from a distance, as in figure 4.41a, a closer look at the extraction result in figure 4.41b reveals that large parts of the wedge edges and the depth point are interspersed with holes in the mesh. In addition to that, the ocher colored thin patch that can be seen in the center of figure 4.41c is a mostly detached piece of geometry. It hovers, like a second surface layer, in a short distance above the aforementioned holes in the mesh. As a result, the wedge is missing a good representation of a depth point and the edge data points, which can be seen as white dots in figures 4.41c and 4.41d are fragmented. Despite these issues, the wedge model is successfully fitted while also discarding the floating surface patch. The good extraction quality becomes particularly clear by looking at the resulting isolated wedge model in figures 4.41e and 4.41f. Being able to handle mesh issues like this and holes in the wedge faces is particularly important when considering that these are frequent issues produced by the software of 3D scanners during the assembly of multiple individual scans.

4.8.3 Wedge classification

The task of wedge classification is to determine the wedge type as defined in section 2.2.2. This is closely related to identifying the directional edge of the wedge. The wedge model extraction from the previous subsection 4.8.2 only fits the edges and the vertices of the wedge model to the geometry without making an explicit assignment of the edges and the outer vertices to the three edges and outer vertices defined in the model. The reason for this is that, in most cases, all three edges must be considered together to make this decision. As the left and right edges and the associated left and right vertices of a wedge are defined in section 2.2.3 clockwise with respect to the directional edge and the writing surface, the directional wedge and the wedge orientation with respect to the tablet surface have to be determined first.

Finding an unambiguous criterion for identifying the directional edge of a wedge from the three available candidates is a very challenging task. In order to make this decision, philologists usually not only consider the writing direction and the wedges in the immediate neighborhood of the wedge in question, but may also consider similar wedges located further away. This can lead to highly accurate decisions, even when the geometric quality of a wedge is very low. However, as this kind of contextual knowledge is not available at this point, the directional edge can only be determined based on the data of the extracted wedge model, the associated wedge area, and the local properties of the writing surface. With this in mind, the extraction rates for choosing the correct directional edge are expected to be at most as good as the extraction rates of a philologist, who can only base his decision on the same reduced amount of data. The issue is complicated by the fact that the specific form of cuneiform wedge types may vary depending on the cuneiform variant, the handwriting of the scribe and the shape of the writing surface. Some preliminary studies conducted with cuneiform experts from the

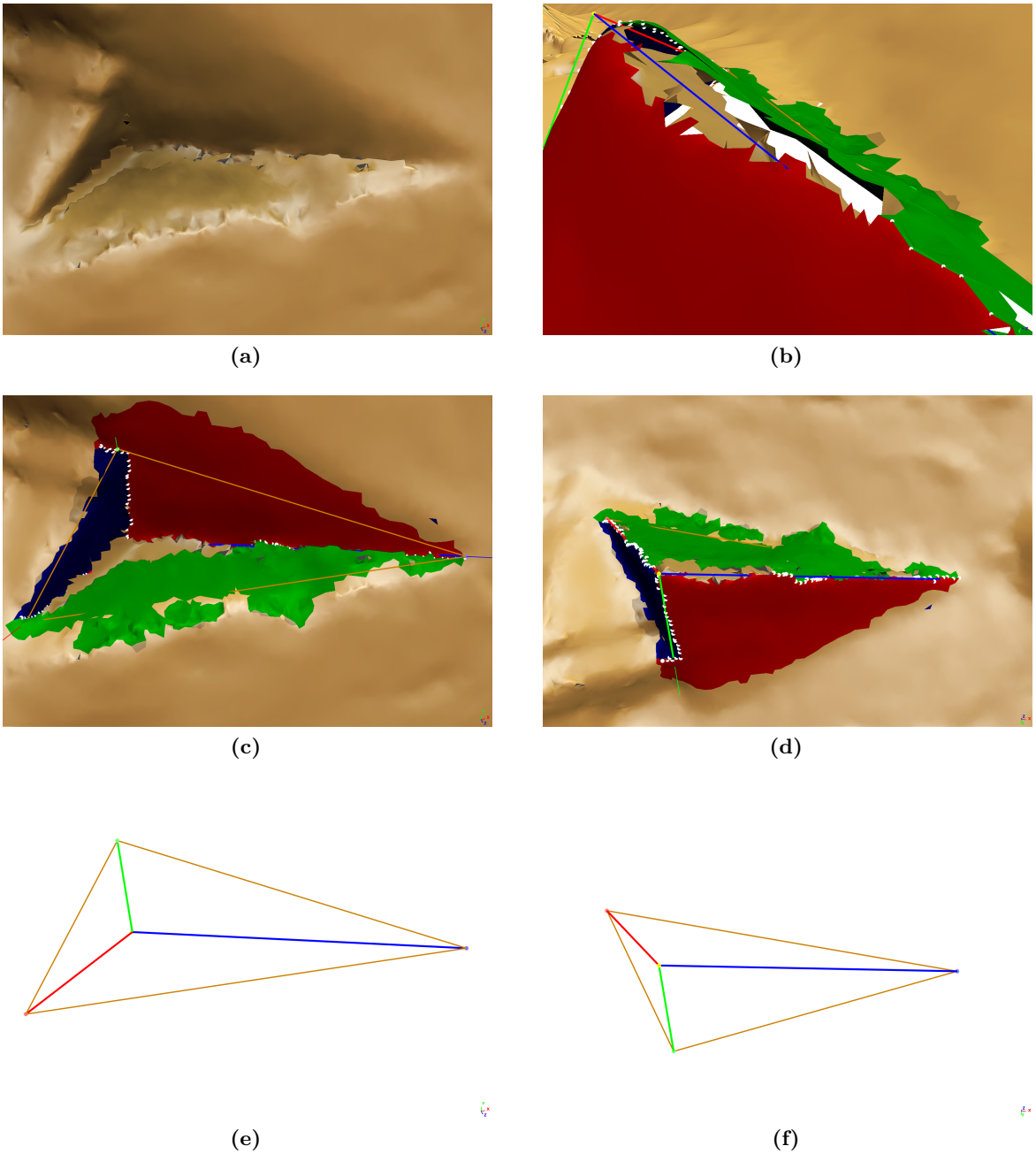


Figure 4.41: Example for the extraction of a wedge with significant mesh damage. The plain wedge is shown in **(a)** and a detail of the damaged wedge center with segmentation data in **(b)**. The extraction result is shown in **(c)** and **(d)**, supplemented by the resulting wedge model in **(f)** and **(f)**. The figures on the left are taken from the obverse of the mesh, while the figures on the right side are taken from the reverse.

University of Würzburg showed that even the manual extraction results of a philologist include a small, but non-negligible amount of false or unambiguous decisions under these conditions.

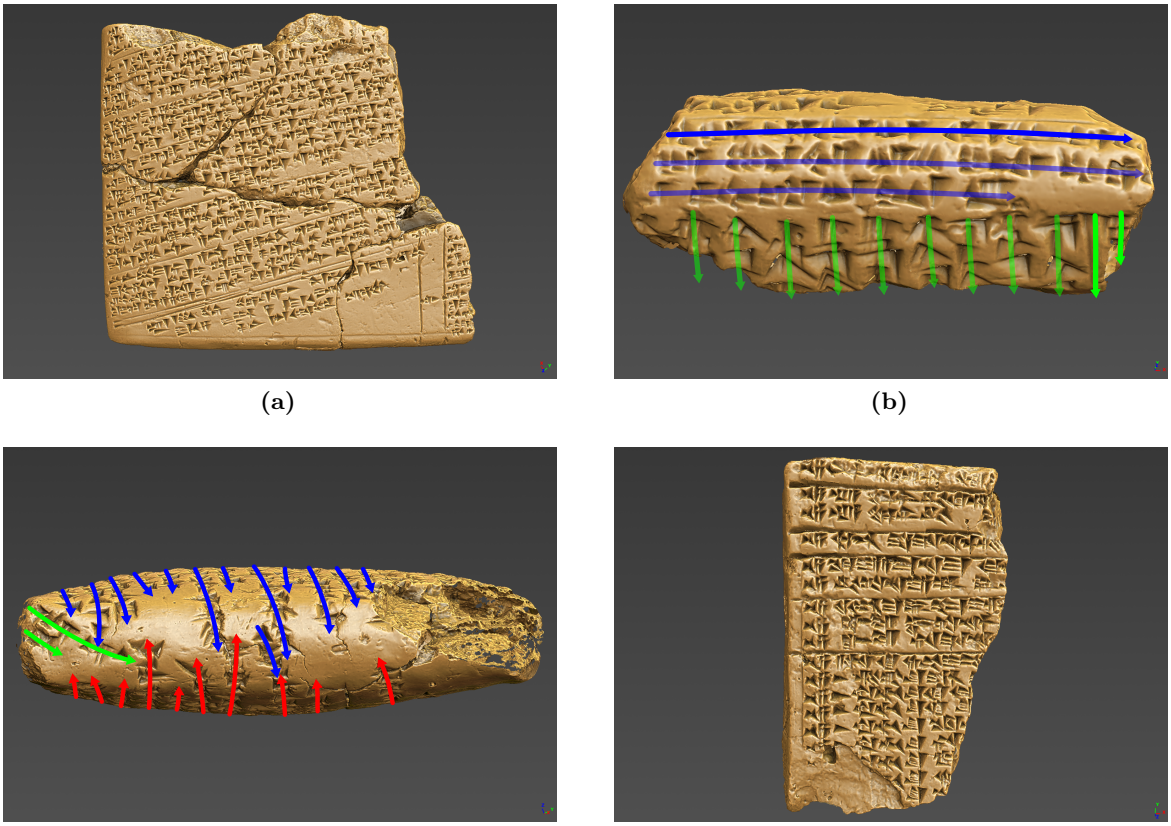
To account for the general complexity of the directional wedge decision problem, in the following, this thesis will discuss several heuristic approaches that have been identified to achieve reasonably good hit rates. These heuristics are based on observations made on existing cuneiform manuscripts and on assumptions about the writing process. A first very simple approach builds onto the observation that the directional edge in Hittite cuneiform has a tendency to be the longest edge within a wedge. This builds upon on the assumptions that an edge is elongated when inclining the stylus in the direction of that edge and that the directional edge is produced by a long lateral edge of the stylus [39]. One issue with the longest-edge heuristic is that it works reasonably well on non-superimposed horizontal and vertical wedges, but may fail, for example, on Winkelhaken-type wedges, vertical wedges with a significant sideways tilt, and many types of superimposed wedges. The problems with superimposed wedges can be easily seen when looking at the extraction results in figure 4.34. As an example, the horizontal wedge 1 in figures 4.34c and 4.34d has a significantly shortened directional edge that is superimposed by wedges 2, 3 and 4. This makes the downwards pointing edge the longest edge that is not the directional edge. In the same figures, only a small central area around the depth point of Winkelhaken-type wedge 4 and an unconnected fragment of the right edge remain intact. As a result, large parts of the wedge are not taken into account. For this wedge, it is worth mentioning that even under non-superimposed conditions, the longest edge would be the edge pointing diagonally upwards to the right and not the one pointing diagonally downwards, which is the correct directional edge. A similar effect can be observed on some of the Winkelhaken-type wedges in figures 4.34e and 4.34f.

A second simple approach is based on the observation that inclining the stylus in the direction of one edge should also make the opposing wedge face steeper with respect to the writing surface. Therefore, the steepest wedge face is determined by considering the surface normal and the normals of the three planes, defined by the depth point and pairs of two outer vertices each. Regardless of its length, the opposing wedge then automatically becomes the directional edge. This approach has the additional advantages that the inclination of the top face is not influenced by tilts of the stylus around its longitudinal axis and that it is comparatively robust against superposition effects on the wedge faces. The robustness against superposition effects is a direct result of the method only relying on the surface normal and the wedge vertices, which also includes the depth point correction of the model fitting. Determining the wedge face inclination based on averaging the normals of the involved triangle mesh faces instead would be much more susceptible to errors caused by filled wedges, impurities on the wedge faces, and mesh errors. Therefore, the mesh triangle normals are not used here. Looking at figure 4.34 again, it immediately becomes clear that the second approach resolves the issues encountered with the first heuristic.

To determine the left and right edge and the associated outer vertices from the identified directional edge, the orientation of the wedge with respect to the writing surface must be determined. A heuristic approach for computing the wedge orientation based only on the extracted wedge faces is to compute the average normals of the wedge faces. This is done by averaging the normals of the involved mesh triangles weighted by the triangle areas, thus accounting for potential effects of curvature adaptive remeshing. The resulting three normals per wedge are then averaged to form one wedge normal and are compared to the normal of the free writing surface approximated in section 4.5.2. The surface normal is determined at the approximated surface point closest to the depth point of the wedge. This approach exploits that the mesh triangles captured by a structured-light scanner must be necessarily visible from the direction of the surface. Checking if the dot product of both normals is greater than zero, provides an initial indication of the plausibility of the extraction result, because both normals are expected to roughly point into the same direction. In the case of a positive result, the plane defined by the surface normal can be used to determine the left and right edge clockwise, starting with the directional edge projected onto the plane. All basic properties of the cuneiform wedge model have then been extracted.

With the wedge model being extracted, it is desirable to determine the type of the wedges next. As the naming of the wedge types suggest, they are defined based on the relation of the wedge direction, which is given by the directional edge, to the writing direction of the cuneiform manuscript. Computing the writing direction on a 3D-scanned cuneiform manuscript is a challenging task. While some manuscripts have evenly spaced and parallel lines and a flat writing surface, thus providing a good starting point for a wedge-density-based line direction estimation, many cuneiform fragments exhibit much more difficult conditions. The writing density may be so high that the text lines are overlapping and there is no significant density drop in between the lines. In addition to that, lines may drift upwards and downwards or be very short on small fragments or tiny documents. Lines may even suddenly change directions up to 90° or go around the side of a clay tablet to interleave with lines on the reverse side of the tablet that have an opposing writing direction. These cases with complex writing directions do usually occur if the scribe had to squeeze an unexpectedly large amount of text onto the available free writing surface.

Some examples for non-trivial writing directions are shown in figure 4.42. The text lines in figure 4.42a are comparatively straight and evenly spaced, but rise diagonally over the text column and are affected by cracks in the writing surface. Figure 4.42b shows a tablet fragment, where the lines in the upper, blue marked text block are oriented from left to right and the one in the lower, green marked text block are rotated by 90° clockwise. The text lines at the side of the tablet in figure 4.42c are mutually interlocking from the left (green), top (blue) and bottom (red) side. As a last example, the table like structured lower part of the cuneiform document in figure 4.42d has characteristic vertical structures, similar to the horizontal left-to-right text lines, which complicate an automated line direction detection.



(a) **(b)** **(c)** **(d)**

Figure 4.42: Examples for non-trivial writing directions, with rising lines **(a)**, block-whise rotated lines **(b)**, interlocking lines **(c)** and lines with vertical structures **(d)**. The line directions in **(b)** and **(c)** have additionally been marked using colored arrows.

In view of these difficulties, the writing directions used here are obtained from a sparse vector field based on manual user input. For this, the user is required to define a sparse set of directional interpolating cubical splines on the writing surface. As an input guideline, the control points for the splines must be positioned so that the splines closely follow the writing direction and, at the same time, the curvature of the writing surface. Furthermore, the number of splines must be chosen to reflect local changes of the writing direction. The writing direction for one wedge is then determined by using the interpolated direction of the nearest point on a writing direction spline to the center point of the respective wedge. Therefore, a spline defined using only two control points may be sufficient for a cuneiform fragment with a flat writing surface and a uniform writing direction like the one shown in figure 4.42d. On the other hand, one spline per text line may be required in situations, where lines with opposing line directions are interlaced, like in figure 4.42c. In the case of two-sided cuneiform fragments, it is particularly important to ensure that splines on the reverse side do not interfere with splines on the front side due to a too low spline density. This issue can be counteracted by checking if the surface normal of the wedge and the surface normal near the taken point on

the writing direction spline face roughly into the same direction. It is worth mentioning that this system does not require each spline to cover a whole line of text, but can work with many small spline fragments. Typical cases include 3 to 4 splines per text column with 3 to 5 control points per spline, which can be created with negligible time effort.

Using the writing direction information and the extracted wedge model, heuristic approaches can be used to determine the type of the wedges. In the following, one simple and a more advanced approach will be presented. Both approaches are based on evaluating the direction of wedge edges on a plane defined by the local writing surface normal. For this, the edges are projected onto the surface plane using the depth point of the wedge as the support vector. To define an axis for a reference direction of 0°, the interpolated writing direction is then also placed at the depth point and projected onto the surface plane. Directions can then be expressed clockwise in this coordinate system using an angle between 0° and 360°. As the large majority of cuneiform documents processed for this thesis was written using Hittite cuneiform, the presented heuristics will only consider vertical, horizontal and Winkelhaken-type wedges and use observations explicitly made for this type of cuneiform.

A simple heuristic could just take the directional wedge, as computed with the method described above, and compare its direction to the writing direction using predefined reference angles and thresholds for the three wedge types. Ideally, the directional wedge should exhibit an angle of 0° for horizontal wedges, of 90° for vertical wedges and of 45° for Winkelhaken-type wedges. Using this approach with a 15° threshold on all three directions, extraction rates higher than 75% have been observed. However, this only applies under the condition of using clean cuneiform fragments with a uniform writing direction and comparatively perpendicular wedge orientations of horizontal and vertical wedges. In reality, the required reference directions and thresholds may deviate significantly from the values suggested above, resulting in much worse extraction rates. As a complicating factor, the orientation of wedges may even change within a text line due to the changing orientation of the human hand during writing. Also, this simple approach only works under the condition that the directional edge is chosen correctly in beforehand. This motivates the application of a more sophisticated heuristic, which will be presented in the following.

An advanced heuristic has to account for varying directional preferences of the wedge types to be applicable on a wider range of manuscripts. One specific aspect of Hittite cuneiform includes that the direction of the directional edge may be ambiguous, when considered individually. For example, directional edges of horizontal wedges may be oriented very similar to the ones of Winkelhaken-type wedges, which makes these wedges indistinguishable when using only this criterion. Similarly, the directional edge of vertical wedges may be very similar to the right edge of horizontal wedges. In cases like this, considering additional edges, besides the directional edge, can help with the differentiation the wedge types.

Figure 4.43 shows an exemplary pseudo-code for an advanced heuristic developed for the specific case of Hittite cuneiform. As a first step, the three wedge edges and the approximated

writing direction are projected onto the plane defined by the surface normal for the respective wedge, which approximates a tangential plane on the writing surface. Next, the angles of the edges to the writing direction are computed on this plane. The subsequent step of determining directional edge candidates builds onto a specific property of the targeted Hittite cuneiform variant, which has applicable restrictions on possible directions of the directional edges. This results in an interval of legal angles that is marked red in figure 4.44a. Depending on the number of candidates, the heuristic continues to differentiate the wedge type. If there is only one directional edge candidate, it must obviously be the directional edge, which, at this point, already allows to also assign the left and right edge accordingly. The wedge type can then be determined by comparing the directional edge against the blue, yellow and green intervals shown in figure 4.44a to determine the wedge type. As the yellow and green intervals overlap, the angle of the green right edge is additionally used to differentiate between the tow cases. Any wedge that can not be matched with the conditions for one of the three wedge types is labeled as undefined. However, this does not invalidate the directional edge if it has been identified in the process. In a case where two directional edge candidates have been identified, the heuristic discards the possibility that the wedge is of Winkelhaken-type, because this type of wedge is extremely unlikely to have two edges in the candidate interval. The differentiation between the remaining horizontal and vertical type is then made by considering the outer edge between the non-candidate edge and the CCW next edge, which is on the left side of the wedge. If this outer edge has a tendency to point towards the writing direction, when viewed as a vector from the right vertex to the left vertex, the wedge is classified as a vertical wedge and as a horizontal wedge otherwise. In the case three directional edge candidates have been identified, the wedge classification outputs a failure, because there is no known legal wedge type to feature this condition. In addition to that, this output is very unlikely, as the red interval of angles covers approximately one third of a full circle and the k-means wedge face clustering favors equal partitions of the angular space. All cases not covered by the heuristic up to this point result in the wedge being classified as undefined since the heuristic can not find a better classification. Thus, the advanced heuristic includes an intrinsic wedge level validation mechanism.

The specific values for the angles and the auxiliary conditions are based on a manual visual analysis of the cuneiform variant targeted for wedge type classification. Significantly differing cuneiform variants in terms of wedge types and wedge shapes may require a different set of values and conditions. However, the advanced classification heuristic presented here, has shown to produce good results on many variants of Hittite cuneiform writings. It is superior to the simple heuristic that can not distinguish wedge types on cuneiform types with overlapping occurrence intervals for the directional edges.

Figure 4.44b shows some examples for differently shaped wedges of horizontal, vertical and Winkelhaken-type. The wedges h_1, v_1 and w_1 in the first column have an idealized appearance, while the wedges in the remaining two columns represent exemplary variations that are also covered by the heuristic. The examples have been selected in such a way that they can be

```
function classifyWedge()
{
  project the three edges on the surface normal plane;
  compute edge angles to writing direction;

  // compute directional edge candidates
  directional_edge_candidates = 0;
  for all edges
  {
    if (edge_angle < 110 && edge_angle > 340) directional_edge_candidates++;
  }

  // evaluate edge angles
  if (directional_edge_candidates == 1)
  {
    make the candidate the directional edge;
    // candidate direction is in right sector for vertical wedges
    if (directional_edge_angle > 67.5 &&
      directional_edge_angle < 135)
      return "vertical";
    // candidate direction is in right sector for Winkelhaken and
    // the right edge has a tendency towards writing direction
    if (directional_edge_angle > 22.5 &&
      directional_edge_angle < 67.5 &&
      right_edge_angle > 270)
      return "winkelhaken";
    // candidate direction is in right sector for horizontal
    if (directional_edge_angle < 32.5 ||
      directional_edge_angle > 340)
      return "horizontal";

    return "undefined";
  }
  else if (directional_edge_candidates == 2) // can't be a Winkelhaken
  {
    determine outer edge between the non-candidate edge and the CCW next edge;
    // outer edge has a tendency towards the writing direction
    if (outer_edge_angle > 0 && outer_edge_angle < 90)
    {
      make the candidate with the bigger angle the directional edge;
      return "vertical";
    }
    // outer edge has a tendency away from the writing direction
    if (outer_edge_angle > 90 && outer_edge_angle < 180)
    {
      make the candidate with the smaller angle the directional edge;
      return "horizontal";
    }
  }
  else if (directional_edge_candidates == 3)
  {
    return "fail";
  }
  return "undefined";
}
```

Figure 4.43: Pseudo-code of a wedge classification heuristic for Hittite cuneiform. All angles are specified in the writing surface plane with respect to the writing direction.

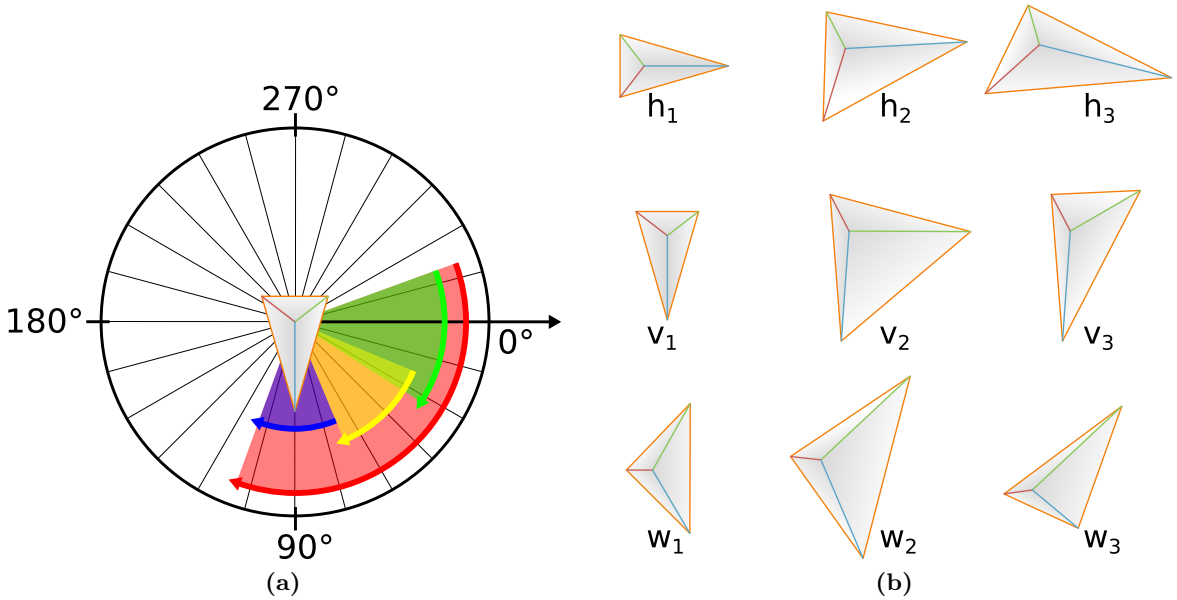


Figure 4.44: Visualization of the wedge classification heuristic. The coverages of the classification angle intervals are shown in **(a)** with the writing direction pointing to the right. Some differently shaped specimen for the three wedge types are shown in **(b)**.

used to reconstruct the decisions of the heuristic. For instance, the wedges h_2 and v_1 show the difference in the orientation of the left outer edge with respect to the writing direction, which is used to distinguish some horizontal and vertical wedges. With v_2, w_1 and w_3 the examples also include wedges, where the directional edge is not the longest edge. It is worth mentioning that the edge lengths of sample wedges can be varied by a significant amount without changing the outcome of the heuristic. This contributes to the good generalization of the approach and is the key to its applicability to superimposed wedges.

More details concerning the obtainable extraction rates of the wedge model extraction and the effectiveness of the wedge classification will be presented in section 4.10 and partially throughout section 4.9. The presentation of the results is postponed to be able to take the effects of the wedge validation into account that will be discussed in the following section 4.9.

4.9 Wedge validation

Up to this point, the extracted cuneiform wedge models are produced using no extended measures that go beyond the basic validation at wedge level from the previous section to check the validity of the results. The segmentation result therefore includes wedge areas extracted from actual wedge geometry and areas from fracture faces and surface impurities with either, a successful model fitting or no model fitting at all. However, the presence of a successfully fitted wedge model does not guarantee that the associated wedge area does indeed represent a cuneiform wedge. This section describes various approaches to validate the extraction results

and check their plausibility. Given the possible high similarity of geometric regions in fracture faces to actual wedges and the high variability of damaged wedges, it is expected that this will only work to a certain extent.

The plausibility of wedges can be considered on at least two different levels. The first level evaluates the plausibility of an extracted wedge by only looking at the extracted wedge model, if any, and the segmentation data on the corresponding wedge area on the mesh. Possible evaluation criteria at this level include the size of the wedge area, the relative size of the wedge faces, the inner angles of the extracted wedge model, the length of the wedge edges, the wedge depth and the direction of the wedge normals. This is based on the assumption that it is possible to identify value ranges for a subset of wedge properties that are very unlikely or impossible to occur on a real wedge on a given type of cuneiform manuscript. Examples for this include wedges with a depth of several centimeters or a wedge area of several square centimeters on cuneiform fragments with much smaller average values for these parameters.

When selecting a set of suitable wedge features with absolute or relative value ranges, the specific properties of the respective cuneiform variant must be taken into account. While some types of cuneiform contain, for example, wedges with directional edges pointing to the upper left (180°-270°), other types do not contain wedges like this. The size and direction tolerances on some Babylonian cuneiform variants are comparatively narrow, especially on big, stone incised documents. There, absolute angle deviations with respect to the scheme used for wedge classification in section 4.8.3 may well be below 5° for one wedge type. In addition to that, the wedge depths on stone incised manuscripts tend to be very uniform. Both features are therefore very well suited for wedge validation, in this case. On the other hand, wedge directions in Hittite cuneiform may vary in a +/- 20° range and wedge sizes and depths may vary by several 100% when considering all wedge types. In such cases, it is not only advisable to weigh the use of absolute and relative features against each other but also to specifically use position or rotation-invariant features. Examples for features that do not depend on the wedge position and orientation include the inner edge angles and the length of the wedge edges. However, the latter may, again, be affected by wedge super-positioning.

An exemplary set of conditions that has been found to produce a good indication for invalid wedges on many Hittite cuneiform documents analyzed in this thesis includes the following conditions:

- Wedges with a directional wedge between 135° and 315°.
- Wedges with one inner angle bigger than 180°.
- Wedges with a depth, bigger than three times the average wedge depth.

A wedge is considered invalid if any of the conditions is fulfilled and it is labeled accordingly. It is worth mentioning that this set of conditions is especially effective for identifying false

detections on damaged text passages and in random surface damages, like scratches and air bubbles. This is partially a result of the wedge direction being included in the first condition, which uses the writing direction as a reference. Since the writing direction is usually not well-defined for false wedge detections on fracture faces, this wedge filter is not very effective at eliminating false detections on fracture faces. However, eliminating any detection on a fracture face does not cause any harm either, as there are no valid wedges on fracture faces.

Two additional aspects of wedge validation to consider are invalid wedges as a result of wedge segmentation faults and wedges that do not fit well into the set of extracted wedges on the respective cuneiform fragment. While in the first case, wedges are already discarded at an earlier point in the extraction pipeline, in the second case, wedges can be discarded based on a comparison with other wedges. Both aspects will be discussed in the following.

4.9.1 Wedge segmentation faults

Wedge segmentation faults can be classified into two groups of issues, one of which is detectable by the here presented automated extraction approach and the other one is not. Detectable wedge segmentation faults are represented at this point by wedge areas classified as undefined or with an error type which has been assigned during wedge area or wedge model extraction. Possible causes for this type of faults are wedge areas with an insufficient size or a too low amount of features for wedge model extraction. Furthermore, the model extraction may fail because the wedge area represents a non-wedge shaped indentation or an indentation with large impurities or many mesh errors. Usually, in these cases, the RANSAC-fitting fails to find a suitable model for at least one wedge edge or the resulting wedge model fails the wedge level validation during the computation of the directional edge, as described in section 4.8.3. In contrast, only a small fraction of wedges is discarded during the actual wedge validation, described in section 4.9. This is because the issues leading to the invalidation of a wedge there, are far less likely to occur than the ones described above. An additional detectable reason for a segmentation failure occurs, when the clustering during wedge face extraction results in many small clusters of similar size. In this case, keeping only the three biggest clusters can lead to a situation where the clusters are no more mutually adjacent and no sufficient number of points for the three required edge data point sets can be determined. The proposed treatment of wedges with a detected faulty segmentation is to mostly exclude the extraction results of these wedges from the 3D visualization. This way, the user does not get distracted by non-successful segmentation results on the 3D mesh upon result inspection. However, the segmentation results are still available and can be unhidden any time if desired. This not only allows an in depth examination of possible causes for segmentation failures to optimize the wedge extraction parameters but also allows a manual correction of the extraction results.

The so-called undetectable wedge segmentation faults do predominantly occur in situations, where the automated extraction results do not match with the expectations of a cuneiform expert. Examples for this type of segmentation issues include fused wedges resulting from a too subtle depth point, separated wedges resulting from a second depth point in the same

wedge, extracted wedge models in scratches, air bubbles or fracture faces, and erroneously assigned wedge types and directional edges. The main reason for the undetectability of these kinds of issues results from the fact that all these situations may result in successfully fitted wedge models that could represent geometrically valid wedges. As the wedge model extraction is explicitly constructed to be robust against the usual inaccuracies of cuneiform wedges, a certain number of false detections is expected as part of a trade-off between higher extraction rates and a higher result quality.

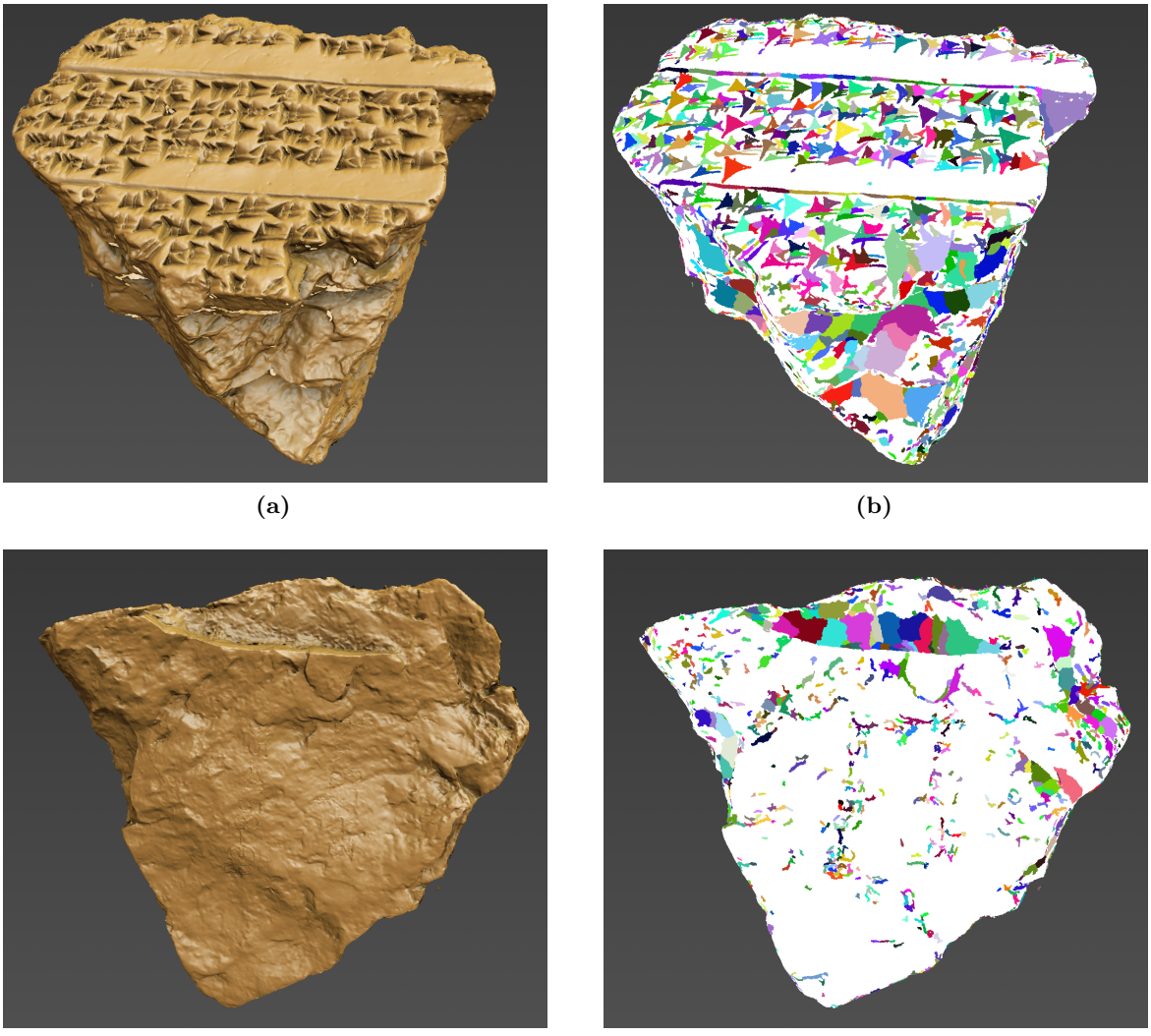
4.9.2 Similarity analysis

One approach to identify a large fraction of the undetectable segmentation faults mentioned in the previous section is to consider more than just one wedge to determine the validity of a wedge. Incorporating the context of a wedge-shaped indentation into its classification is a technique also used by cuneiform experts. This can be realized by using the knowledge about participating wedges of cuneiform signs or by considering similar geometric regions on the same cuneiform manuscript. This section will present a similarity-based approach to significantly lower the amount of false detections on fracture faces and damaged sections of the writing surface.

A typical extraction result on a medium-sized cuneiform fragment, including large fracture faces at the sides and the reverse side is shown in figure 4.45. The false detections on the fracture faces can be identified by comparing the shaded images on the left, with the visualization of the randomly colored extracted wedge areas on the right side. Some clusters of false detections on the reverse side assume the shape of numbers, because they were caused by scanning artifacts resulting from an inventory number, written with a black pen onto the fracture face. In view of the large amounts of false detections, the need for a countermeasure becomes apparent. Therefore, this thesis suggests to exploit the self similarity of cuneiform wedges for filtering out false detections.

The similarity of cuneiform wedges can be determined based on a comparison of their features. Here, only features like the type of the wedges and all direct or derived properties of the extracted wedge model will be considered. The usability of the features for similarity search depends on how well they can describe the wedges without losing the capability to match as many similar wedges as possible. Therefore, using position, rotation or scale invariant features may be advantageous. A set of suitable wedge features including their invariance properties is shown in table 4.2. As all contained features are position invariant, this property is not listed. Here, the edge directions are evaluated as an angle difference between matching edges of the compared wedges. This feature is scale invariant, but only partially rotation invariant, because it depends on the external definition of the writing direction.

To parameterize the similarity search, each feature is combined with a threshold that specifies the allowed deviation of the feature between two wedges to be classified as being similar. This threshold can be an absolute value or a ratio, relative to the value of the wedge in question.



(a) **(b)**

(c) **(d)**

Figure 4.45: Example for an unfiltered wedge extraction result on cuneiform fragment *Bo2743*. The plain fragment is shown in **(a)** and the random colored extracted wedge areas in **(b)**. A matching set of images of the reverse side is shown in **(c)** and **(d)**.

Feature	Rotation invariant	Scale invariant
edge directions	✗	✓
edge lengths	✓	✗
inner angles	✓	✓
outer angles	✓	✓
wedge type	✓	✓
dir. edge min. length	✓	✗

Table 4.2: A selection of suitable wedge similarity measures.

When using multiple features at the same time, two wedges are similar if they a classified as being similar for each feature used. As the first four features consist of three values each, the thresholded comparison has to be performed for each value individually. Only for the case when all three differences pass the thresholding, the compared wedges are classified as similar. The last feature, the minimum length of the directional edge, is less a comparison feature than a global filtering criterion. It is listed here, as it may also help with identifying invalid wedges. A combination of the edge directions and the edge length has been found to be especially effective for identifying similar wedges on Hittite cuneiform manuscripts. In the following, a difference threshold of 25° was used for the edge directions and a ratio of 3.0 was used for the edge length.

As a first step to identify the set of valid wedges, for each wedge, the number of similar wedges is computed. The wedges are then sorted according to their number of similar wedges. This way, common and less common wedges can be identified. Discarding wedges with less than two similar wedges, works already well to eliminate many false detections on cuneiform fragments with more than 25 wedges. An even better working approach is to select the most common wedge of each wedge type. Then, starting with one horizontal, vertical and Winkelhaken-type wedge, the selection is iteratively expanded. In each iteration the selection is extended by all wedges that are similar to one of the wedges that were in the selection produced by the previous iteration. If the set of similarity features is well selected, the set of selected wedges will be very similar to the set of actual wedges after a few iterations. It is worth mentioning that the selected set of wedges does not steadily converge against the set of real wedges with an increasing number of iterations. Too many iterations will, in many cases, lead to a situation, where an increasing amount of false detections becomes selected due to some similarities with less common wedges.

Figure 4.46 shows this process for the cuneiform fragment *Bo2743* from figure 4.45. The initial selection of one wedge per type is shown in figure 4.46a, the result of expanding the selection one time in figure 4.46b and the result for expanding the selection a second time is shown in figure 4.46c. As can be clearly seen, the final result in figure 4.46c includes almost every

Total	Most common	Valid 1st iter.	Valid 2nd iter.	Invalid	FP
2013	3	245	340	1673	5
2013	3	201	359	1653	49

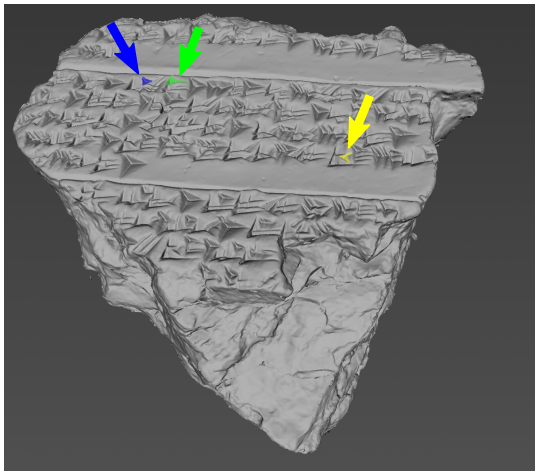
Table 4.3: Exemplary statistics for filtering the wedge extraction results on cuneiform fragment *Bo2743*. The first line shows the results obtained using edge directions and edge lengths similarity and the second line shows the results obtained using inner edge angles and edge lengths similarity. The last column contains the number of false positives in the final result.

wedge on the writing surface, but almost none of the false detections on the fracture faces and on the writing surface. Even the wedge areas detected in the paragraph separators have been sorted out.

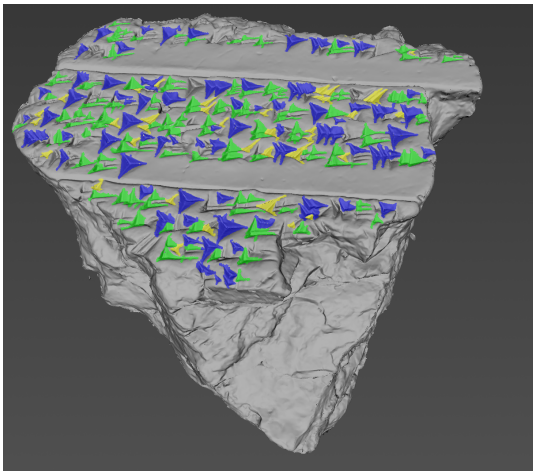
In numbers, as can be seen in table 4.3, that means that out of the 2013 total wedge areas extracted on fragment *Bo2743*, three most common wedges were automatically selected and expanded to 245 and 340 wedges in two similarity search iterations. This leads to 1673 of the 2013 wedges being filtered out and being classified as invalid. The result of 340 valid wedges contains only 5 wedge areas that have been erroneously classified as being wedges. None of these false positives is on the reverse side of the tablet and only one on the lateral fracture faces, as can be seen in the lower half of figure 4.46c.

It is worth mentioning that using the edge directions as a similarity measure and starting with the most common ones of each type, usually selects the wedges on the largest planar portion of text on the cuneiform fragment examined. This behavior can be used to find valid wedges on the front side and the reverse side separately. To achieve this, the process of selecting three common wedges and executing two similarity iterations can be repeated multiple times until the similarity count of the remaining wedges drops below a specified threshold. An exemplary extraction result of this method is shown in figure 4.47 for cuneiform fragment *133-r*. The good filtering performance becomes especially apparent, when looking at the extracted wedge models in the sideways view in figure 4.47c, where almost no wedge models show up outside of the text regions. Of the 7189 extracted wedge areas, 2031 have been classified as being valid wedges. These include only 12 (0.6%) false detections on fracture faces. However, as this cuneiform fragment exhibits many small bubble like surface defects in between the wedges, there are 61 (3.0%) false detections scattered throughout the text paragraphs.

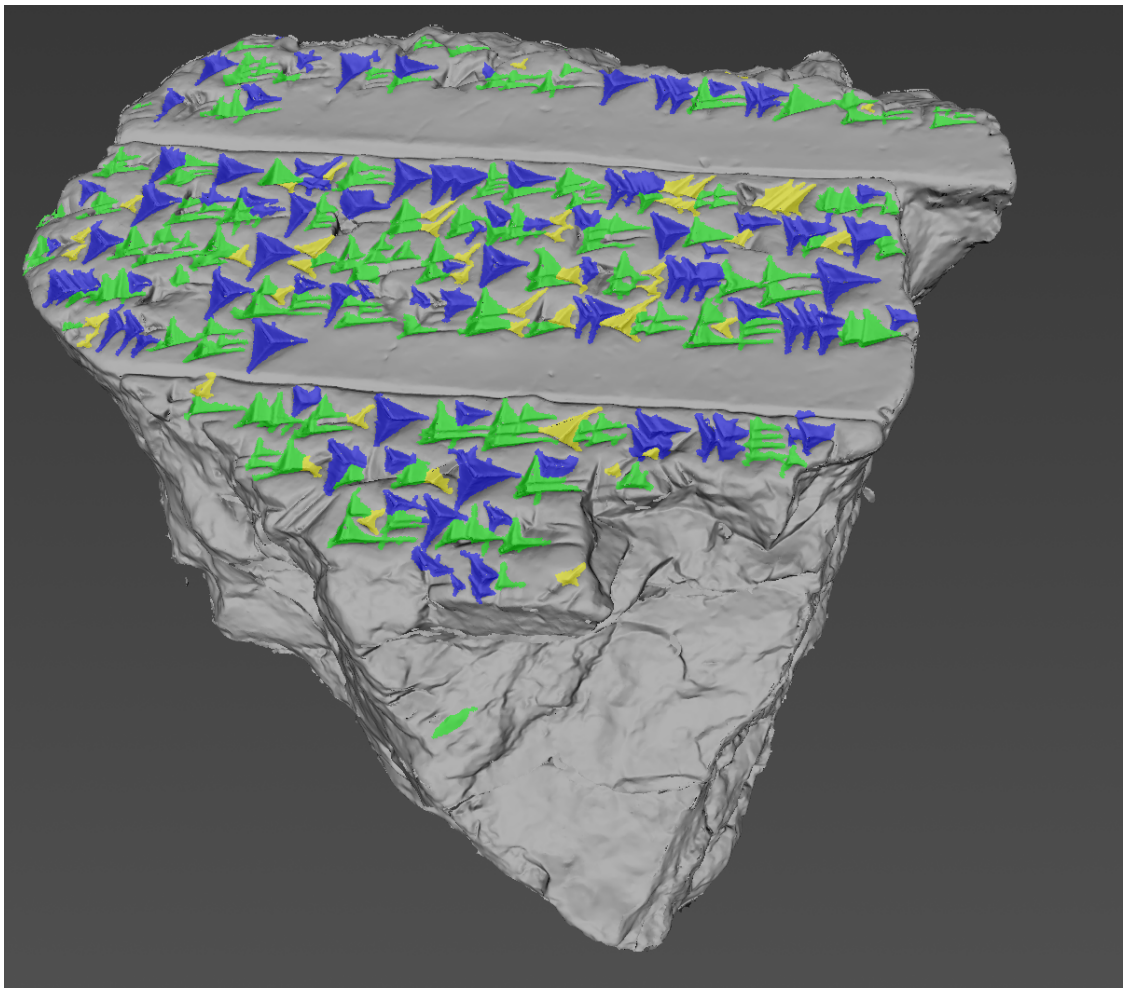
Obviously, using the edge directions as described above will be an issue on fragments with a significantly curved or round writing surface. On these kinds of datasets, the edge directions can be calculated in the coordinate system used for wedge classification as shown in figure 4.44. Considering the edge direction relative to the writing direction then allows matching similar wedges in all possible line directions. As this includes wedges on the reverse side of a fragment, wedges on different sides of a manuscript can not be distinguished anymore using this criterion. In addition to that, the writing direction must be defined accurately. In cases with less



(a)

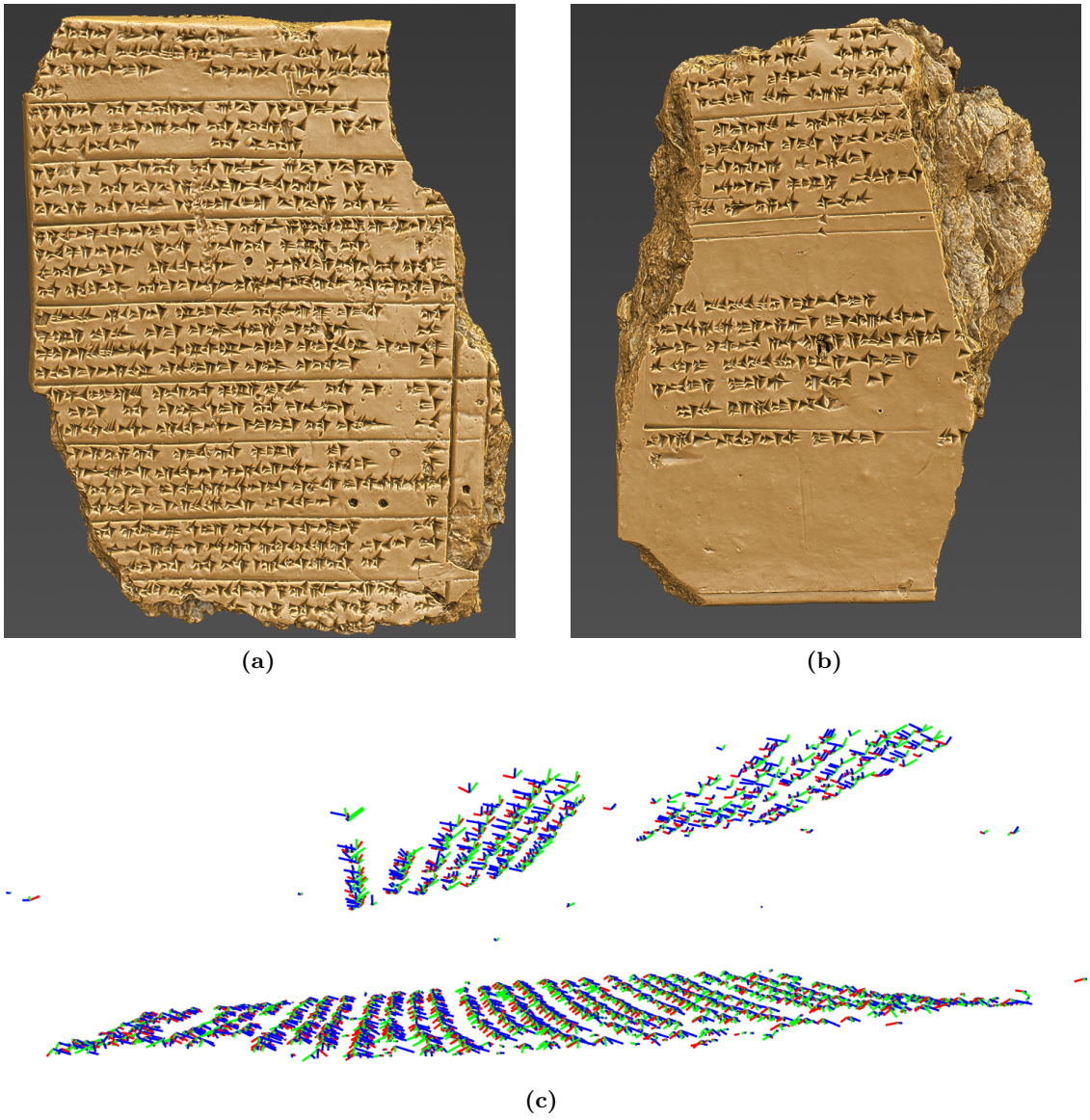


(b)



(c)

Figure 4.46: Example for filtering the wedge extraction result on cuneiform fragment *Bo2743*. The selected most common wedges of the three types are shown in **(a)**, emphasized with colored arrows. One similarity iteration to expand the selection leads to the results in **(b)** and two similarity iterations lead to the final result in **(c)**. All wedges are color-coded according to their computed wedge type.



(a) **(b)**

(c)

Figure 4.47: Example for filtering the wedge extraction result on cuneiform fragment *133-r*, which features text on the front side and the reverse. Both sides of the plain fragment are shown in **(a)** and **(b)** and a side view of the extracted and validated wedge models in **(c)**.

accurately defined writing direction, the edge directions can be replaced by the rotation invariant inner edge angles to still obtain usable results. In the example of fragment *Bo2743*, using the inner edge angles instead of the edge directions with a threshold of 15°, leads to the detection of 359 valid wedges after two iterations and 49 false positives. 30 of these false positives are located on the reverse side. This shows that the edge directions are a much better criterion for wedge comparison than the inner edge angles. However, the latter are still usable to extract results with sufficient significance for statistical script analysis.

The same method also works for wedges on damaged writing surfaces. Figure 4.48 shows a wedge extraction process on a cuneiform manuscript featuring extensive surface damage. The non-processed manuscript in figure 4.48a not only includes many damaged wedges but also areas with hardly any traces of wedges left. Instead, the surface is covered with many erosion-caused bumps and dents. As figure 4.48b shows, the unfiltered wedge extraction result captures many of these non-wedge related indentations and the extraction process is successfully able to fit wedge models into the corresponding regions. Consequently, the resulting set of wedge models shown in figure 4.48c, contains many elements that do not represent actual wedges. The result of filtering the wedges using the method described above is shown in figure 4.48d. Comparing this result with the non-processed manuscript in figure 4.48a shows, how well the extracted set of wedge models matches with the visually still perceivable wedges on the damaged manuscript surface. The large majority of false-positives has been successfully filtered.

As shown, the presented similarity-based approach works well to filter wedge extraction results on damaged surfaces and fracture faces. However, it requires a certain number of similar wedges to function at all. While this is usually not an issue on large and medium-sized manuscripts, small manuscripts with only a few wedges or cuneiform signs may become problematic. Possible workarounds in these cases include manually selecting the initial set of wedges for the similarity search or specifying a restrictive set of reference parameters and thresholds for allowed wedge geometries.

4.9.3 Corrective heuristics

An extraction result, computed as described in the previous sections produces a certain number of wedges classified as undefined or being filtered out by the similarity analysis. However, a subset of these wedges may possess a successfully fitted wedge model. For these wedges, it is possible to employ a basic correction heuristic and try to convert them into valid wedges retrospectively. This is done by swapping the assigned edges in a clockwise or counter-clockwise manner to replicate all possible variants of assigning the directional edge. The resulting three wedge models are then evaluated to determine whether one variant has a sufficient number of similar wedges in the set of already validated wedges. If this is the case, the wedge edges are swapped accordingly and the wedge can be added to the set of valid wedges.

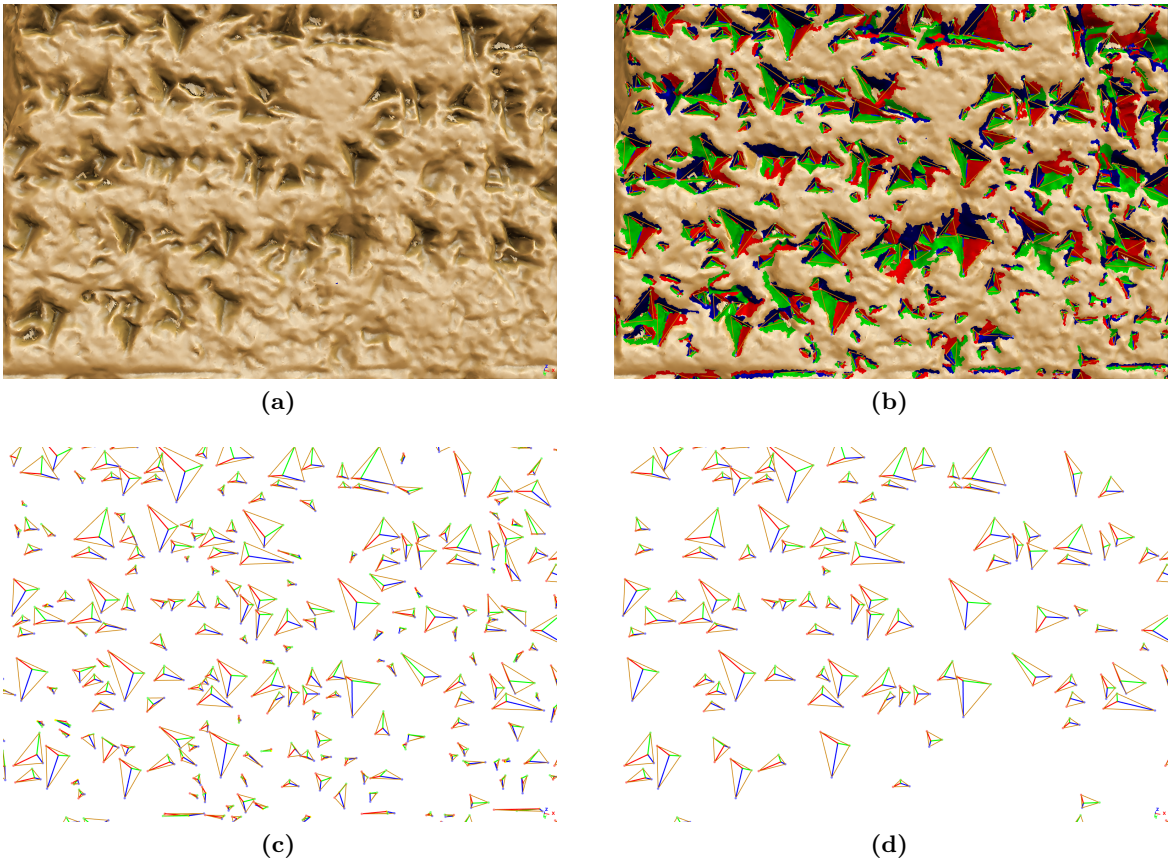


Figure 4.48: Example for a wedge extraction on a manuscript section with significant surface damage. The plain manuscript is shown in **(a)** and the initial extraction result in **(b)**. A wedge-model-only version of the extraction result is shown in **(c)** supplemented by an additionally similarity-filtered version in **(d)**.

4.10 Extraction results

Like for every extraction approach, it is desirable to determine quantitative and generalizable results to evaluate its performance. However, determining quantitative extraction rates for the presented cuneiform wedge extraction approach is only useful to a limited extent. The main reason for this is that the extraction rates show considerable variation on a per fragment basis. Thus, extractions rates for one cuneiform fragment have little significance with regard to other datasets. In general, the extraction rates can become arbitrarily poor with an increasing amount of surface damage, more scanning errors or more extensively overlapping wedges on the fragments. An additional issue with generating quantitative extraction results is that there is no automatic method to validate the results. Therefore, all extraction rates have to be determined manually by comparing the result wedge by wedge against a reference like a hand copy or by resorting to the expertise of a cuneiform specialist. Obtaining generalizable results is therefore only possible by evaluating a larger number of tablets with varying characteristics. However, in view of the fact that even medium-sized cuneiform fragments can contain several hundreds

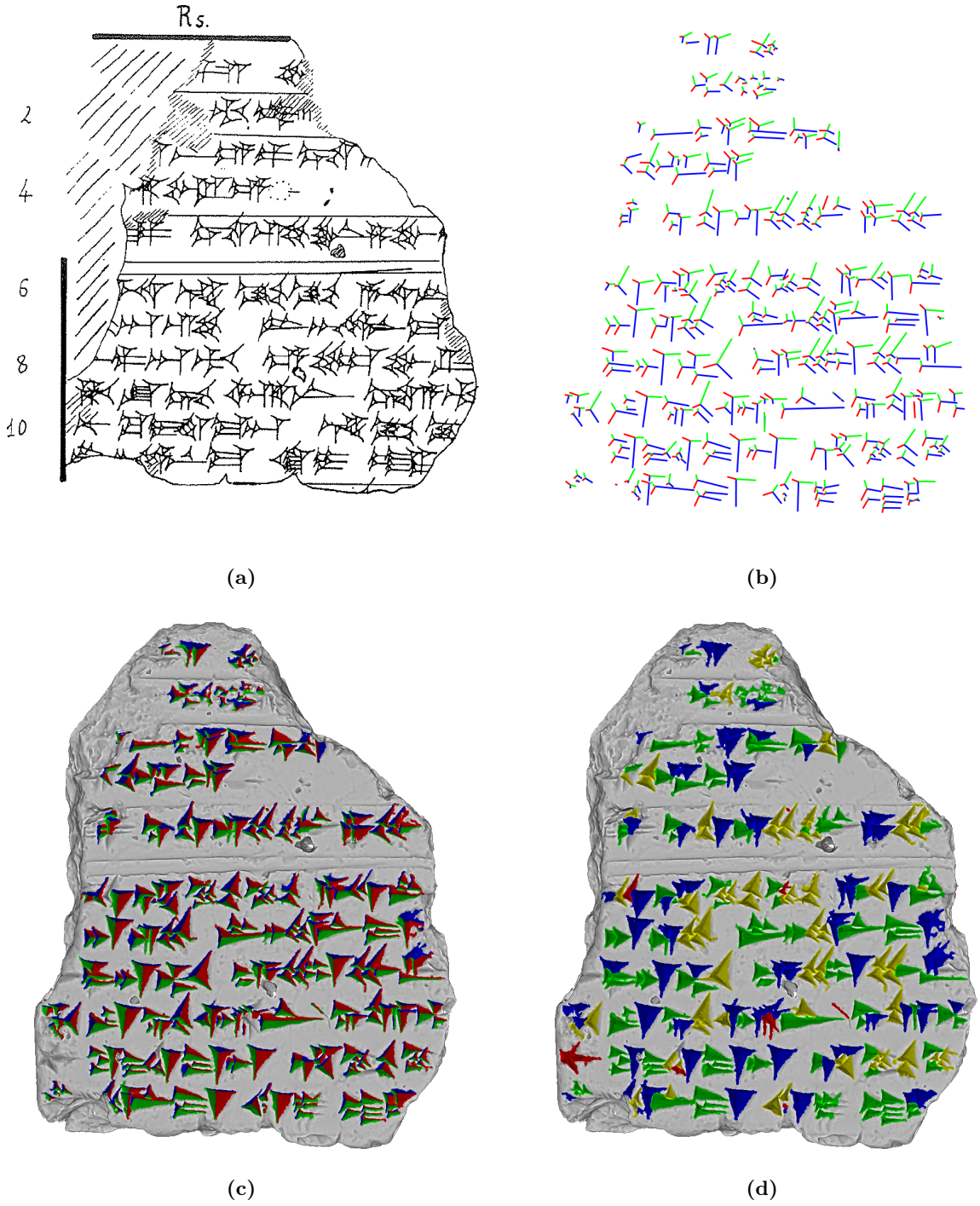
of wedges, the workload for manual evaluation quickly becomes unmanageable. Regardless of the reduced significance for other datasets, extraction results for an exemplary cuneiform fragment will be presented in subsection 4.10.1. Aside from the achieved extraction rates, the performance of the approach in terms of running time will be analyzed in subsection 4.10.3.

To prove the general usability of the extracted data for its intended primary application target, statistical script analysis, precise extraction rates are far less important than the ability to successfully classify the processed cuneiform manuscripts using statistical methods. The general suitability of the generated data for statistical script classification purposes has been proven by Michele Cammarosano [38]. For this, direct and derived features of the extracted wedge models were evaluated to check their significance with respect to a set of philologically motivated features, like the cursiveness of the cuneiform writing, the compactness and slope of wedges, or the distinction of individual scribes and cuneiform variants. In his study, Mr. Cammarosano was able to demonstrate the effectiveness of the approach for automated cuneiform fragment join recognition on a sample set of 297 3D-scanned cuneiform fragments. As a starting point, he used sets of common wedges determined via the feature-based similarity search described in section 4.9.2. The sets were then clustered and the results, in form of a hierarchical tree, were compared to the already known interdependencies of the fragments. Depending on the set of common wedges used, between 65% and 80% of the results were in accordance with the expectations. Considering that previously to this approach, no methods for automated analysis of three-dimensional cuneiform script features were available, this represents a major advance for analyzing large numbers of cuneiform manuscripts.

4.10.1 Extraction rates

In order to give an impression of the effort required for a precise analysis of the extraction rates on a real cuneiform fragment, the following will present and discuss an exemplary extraction result for a medium-sized cuneiform fragment with some damaged areas, a medium writing density, a uniform writing direction, average scanning quality, and a comparatively flat writing surface. The size of the fragment has been chosen to be small enough to keep the workload for manual evaluation still manageable, but at the same time big enough for being able to use the similarity-based filtering of the wedges.

A visual overview of the obtained extraction result on the chosen cuneiform fragment *Bo532* is shown in figure 4.49. For comparison and as a base for checking the validity of the results, figure 4.49a shows a hand copy of the fragment. This hand copy is an interpretation of a cuneiform expert about what is written on the tablet surface. The extraction result is visualized on three visual levels. While figure 4.49b shows the extracted and filtered wedge models as a collection of colored wedge edges, figures 4.49c and 4.49d contain shaded visualizations of the tablet surface with a color overlay for the wedge faces and the wedge types. In contrast to the results presented in section 4.9.2, the extracted wedge models for this fragment contain no false detections on the fracture faces. This is the result of a very high wedge filtering efficiency on this dataset.



(a) **(b)**

(c) **(d)**

Figure 4.49: Wedge extraction results on the medium-sized cuneiform fragment *Bo532*. A hand copy of the manuscript (Source [188]) is shown in **(a)** for comparison and the extracted wedge models are shown in **(b)**, complemented by the wedge face clustering result in **(c)** and color-coded wedge types in **(d)**. In **(d)**, vertical wedges are blue, horizontal wedges are green, Winkelhaken-type wedges are yellow and undefined wedges are red.

Line	Existing		Detected		Correct type		False detections	Missing
1	10		10		10		0	1
2	7	+(7)	7	+(2)	7	+(2)	2	4
3	14	+(3)	14	+(2)	14	+(1)	0	2
4	13	+(2)	13	+(1)	13		1	1
5	33	+(5)	32		30		4	5
6	38	+(2)	38	+(1)	33		6	1
7	29		29		28		1	0
8	32	+(2)	32		32		0	2
9	39	+(2)	38		36		3	2
10	43		37		36		3	7
11	34	+(7)	33		30		3	6
total	292	+(30)	283	+(6)	269		23	31

Table 4.4: Exemplary wedge extraction results on cuneiform fragment *Bo532*. The results are given in terms of the number of wedges that belong to the listed classes. Furthermore, the results are presented on a per text line basis to retain comparability with figure 4.49. The numbers in braces denote damaged wedges that are incompletely represented in the hand copy or the 3D scan.

The extraction results have been evaluated per text line, reusing the line numbers from the hand copy in figure 4.49a. For each line, the number of existing wedges as drawn by the author of the hand copy have been counted. This number is supplemented by the number of wedges that have been detected using the approach presented in this thesis. In addition to that, the number of wedges with a correct type assignment, the number of false detections, and the number of missing wedges is given. False detections are wedges with an undefined or false wedge type and wedges detected on non-wedge geometry or with geometry that does not match the existing wedges. Missing wedges are wedges which are represented in the hand copy and the 3D scan but have not been captured in the extraction result. Typical representatives for missing wedges are very subtle wedges which are represented by only a small amount of geometry in the scan. Some of the presented numbers are complemented by numbers in braces. These numbers represent wedges in the respective category that are only partially existent on 3D scan or in the hand copy. In the hand copy, these wedges are located in the crosshatched regions. Wedges of this type include, for example, damaged wedges with a missing head, which only consist of a small area around the directional edge, but were still detected by the automated wedge extraction.

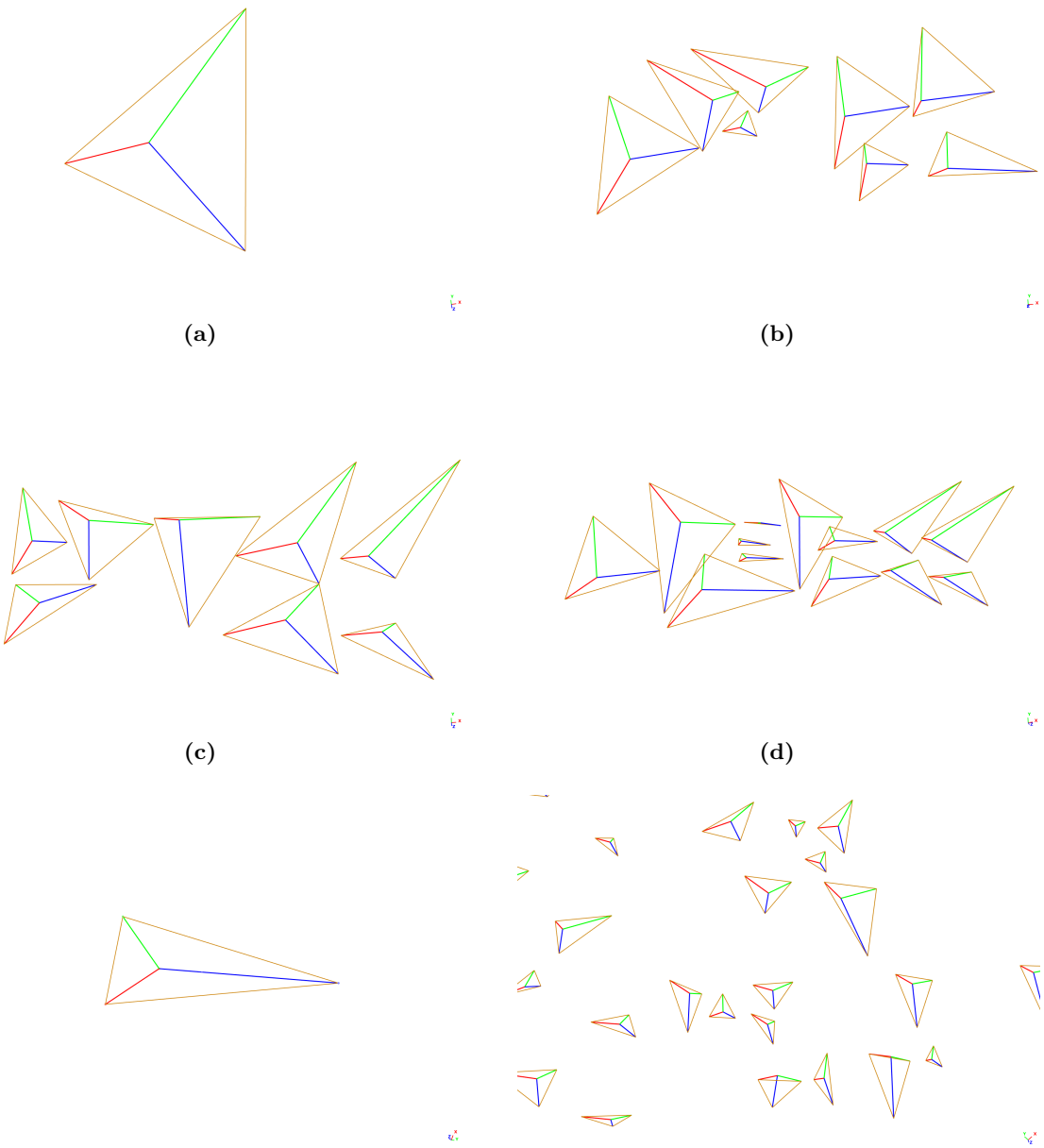
The total number of existing wedges on the chosen cuneiform fragment *Bo532* is 292. Not considering damaged and incompletely represented wedges, 96% of the wedges are successfully extracted and 92% of wedges are assigned to the correct wedge type. The main source for missing wedges on this fragment were overlapping wedges with no indentations left to separate the wedges. The good extraction rates lead to a high probability that the results can be

used effectively for statistical script analysis. They also provide a good base for creating a digital 3D cuneiform sign inventory by grouping the extracted wedges to signs and by adding corresponding annotations. This can be accomplished by using respective tools, integrated into the segmentation framework, which will be described in chapter 6.

4.10.2 Extraction robustness

The robustness of the presented wedge extraction method varies largely, depending on the cuneiform fragment and the parameterization. However, as shown in sections 4.9 and 4.10.1, good extraction rates have been obtained on the Hittite cuneiform scans of the Hethitologie-Portal Mainz. Changing the parameterization of the wedge extraction seems to be mostly necessary, when switching to a set of cuneiform scans, which has been acquired using a different scanning pipeline. This includes significantly changing the scanner resolution, the remeshing during scan fusion, or using a completely different 3D scanner. To show the limits of the method on the Hittite cuneiform fragments used in this work, some results for the complex wedge configurations shown in figure 4.1 are presented and discussed below. For a subset of these, intermediate results of the wedge face extraction have already been presented in figure 4.34.

Figure 4.50a shows the extraction result for the Hittite sign *nu* from figures 4.1a and 4.34a to 4.34b. As the two involved wedges share a common depth point, a wedge model has been only extracted for the Winkelhaken-type wedge. All wedges of the complex overlapping constellation from figures 4.1b and 4.34c to 4.34d have been extracted in figure 4.50b. However, some edges of the two vertical wedges and the Winkelhaken-type wedge have been significantly shortened due to superposition effects. The extraction result in figure 4.50c includes only two of the four vertical wedges visible in the center of figure 4.1c. This is the result of missing depth points for these wedges and their fused top face. It has to be noted that these wedges would not have been visible in figure 4.1c if no detail enhancing post-processing was used. In figure 4.50d three of the subtle horizontal wedges from the center of figure 4.1d and all remaining wedges have been successfully extracted. The lowest of these wedges was fused with the big horizontal wedge due to a missing depth point and therefore not extracted. The uppermost horizontal wedge has an extracted model, but in this case little more than the directional edge is usable. As an example for a wedge with large impurities, the filled wedge from figure 4.1e has been successfully extracted in figure 4.50e. For the significantly damaged writing surface from figure 4.1f, several wedges have been extracted in figure 4.50f. These include correct extractions for some of the less damaged wedges in this section of the tablet. However, the extraction rate and the quality of the model fitting is very low and nearly half of the extracted wedge models are false detections. Thus, the wedge extraction produces no usable result in this case.



(a) **(b)**

(c) **(d)**

(e) **(f)**

Figure 4.50: A collection of extraction results for the complex wedges from figure 4.1.

4.10.3 Extraction performance

The wedge extraction is designed to be usable in an interactive way where cuneiform fragments can be processed in a short amount of time and immediately be worked on. In this section, the running time of the wedge extraction will be evaluated to give an impression of achievable processing times. This will be done with regard to the overall processing times and also by considering the individual pipeline stages separately. The theoretical running time of the wedge extraction does not only depend on trivial properties of the cuneiform fragment to be processed, like its vertex count. Instead, complex properties, like the contained number of wedges, the number of mesh errors, the size of the region of interest, the number and size of the wedge areas, the relative number of fracture faces, and the overall shape of the fragment can all have a noticeable impact on the running time. Due to the unpredictability of some of these factors, the theoretical running time has only limited informative value with regard to the practical running time. Therefore, the performance will be evaluated on a set of 9 exemplary datasets covering a large range of mesh sizes. Nevertheless, the dominating factors that influence the running time will be indicated for each pipeline stage. All performance measurements were taken on a testing platform with an Intel Core i7 4790K with 32 GB RAM and a GeForce GTX 780.

The collection of cuneiform fragments selected for performance evaluation is shown in figure 4.51. All contained sub-figures have an identical image scale to be able to compare the relative sizes of the cuneiform fragments. Here, the largest fragment in figure 4.51i has a real world height of approximately 44 centimeters. As can be seen in table 4.5, the fragments sizes in terms of their vertex count range from just over half a million vertices to over 33 million vertices. This size range has been chosen to accommodate the fact that the running times on smaller fragments are increasingly influenced by uncontrollable factors such as multi-threading effects. On the other hand, larger fragments require more GPU-memory than available on average gaming class hardware, which is a limiting factor regarding practical usability. Aside from the vertex count of the mesh, the table includes the number of vertices in the ROI from section 4.6, the number of wedge areas and the number of remaining wedges after wedge validation, as these factors are helpful for understanding certain aspects of the measured running times.

The achieved overall running times of the wedge extraction in seconds are displayed in figure 4.52 as a function of the vertex count of the cuneiform fragments. Consequently, each fragment is represented by a vertical stack of data points, one for each data series. In addition to the overall running time, the running times for the wedge extraction and the wedge validation are shown, as these two components make up the large majority of the overall running time. On a side note, the overall running time is only approximately the sum of the individual times measured for extraction and filtering, because it includes a very small amount of GUI related processing that could technically not be excluded from the measurements.

When considering only cuneiform fragments with fewer than 10 million vertices, which represent the majority of available datasets, the overall processing time stays below 2 minutes. Especially



(a) Bo532 **(b)** Bo2743 **(c)** Bo51

(d) 126/p Rs **(e)** 101/b Vs **(f)** *485/b* Rs

(g) *482/b* Vs **(h)** *VAT7456 Vs* **(i)** *VAT422 Vs*

Figure 4.51: Fragments for performance evaluation with matched image scale.

Fragment	Vertices	Vertices(ROI)	Wedge areas	Validated wedges
Bo532	587,053	295,550	730	268
Bo2743	1,610,047	840,293	2,015	326
Bo51	3,938,454	2,290,298	4,670	1,626
126/p Rs	7,034,195	4,520,983	6,759	2,657
101/b Vs	8,742,711	5,181,850	11,498	6,112
485/b Rs	12,598,242	8,614,533	13,290	4,697
482/b Vs	14,328,507	8,510,300	20,399	10,245
VAT7456 Vs	16,141,130	10,188,086	16,470	8,441
VAT422 Vs	33,889,749	25,175,219	39,132	14,949

Table 4.5: Characteristics of the fragments for performance evaluation from figure 4.51.

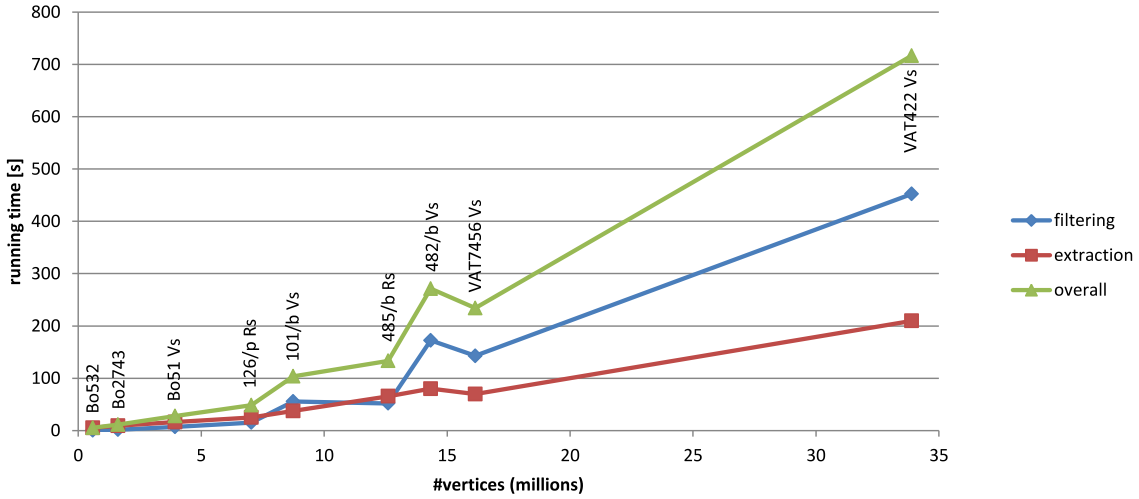


Figure 4.52: Overall wedge extraction performance for the fragments from figure 4.51.

when taking into account that the wedge area extraction, the wedge model extraction and the wedge validation are usually performed and parameterized as separate processing steps, this can be considered to be still in the range for interactive working. In particular, the processing times for the wedge extraction are approximately linear with respect to the vertex count n of the processed datasets. This results from the fact that most components of the wedge extraction have a running time that is approximately linear with respect to n or with respect to the vertex count of the region of interest n_{ROI} from section 4.6. The running time for the respective pipeline stages is visualized in the figures 4.53 and 4.54. Figure 4.53 contains the pipeline stages with a running time that is approximately linear with respect to n. While this can be easily seen for the curvature and ambient occlusion stages, the Poisson reconstruction stage has a running time of $O(n)$ only for a constant octree depth [106] like used here. In contrast, the distance computation represents a more complex case. By using the spatial

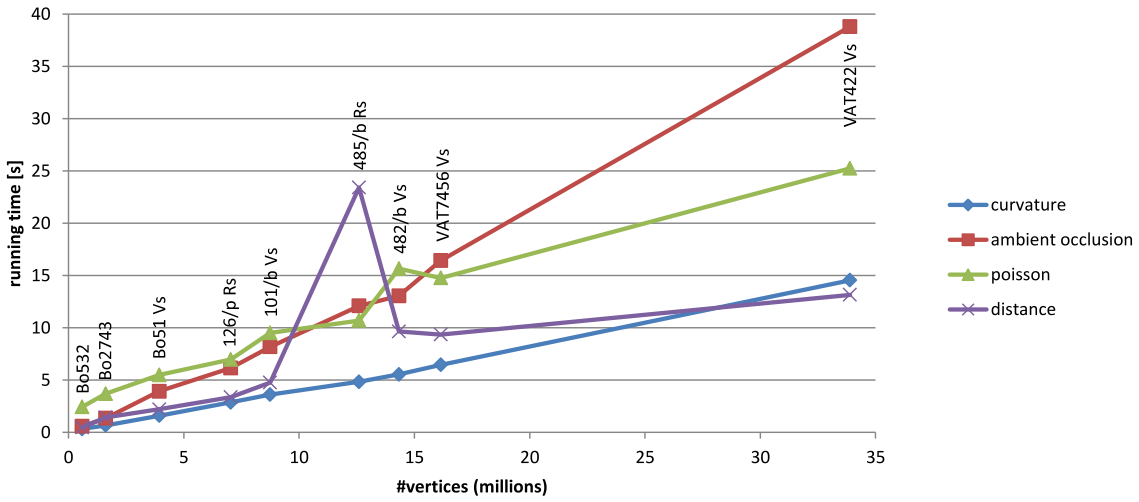


Figure 4.53: Relative running times of components that depend on the vertex count for the fragments from figure 4.51.

hashing optimization from sub-section 4.5.3, the running time is approximately linear with respect to n and the depth of the kd-tree, provided the optimization works. However, for fragments like *485/b Rs* form figure 4.51f, which contain large holes with large fracture faces, the Poisson approximation will tend to close these holes with a comparatively flat surface. This results in many mesh vertices having a large distance to the approximated surface, and therefore, the spatial hashing optimization will not be able to decrease the running time. As the spike in the running time graph for fragment *485/b Rs* shows, this can significantly increase the running time of the distance computation.

Since the watershed method for wedge area extraction and the following pipeline stages operate on the reduced vertex set of the ROI, their running time is linked to n_{ROI}. Figure 4.54 shows running times for the watershed method, the respective extraction of the connected triangle patches representing the wedge areas and the computation of the wedge types. While the running times of the watershed method and for computing the wedge are both linear with respect to n_{ROI}, collecting the triangle patches includes a DFS-scan of every wedge area. Therefore, the running time for collecting the wedge areas also depends on the amount of wedge areas, and their respective triangle count. This is the reason for the increased running time of fragment *482/b Vs*, which exhibits a much larger amount of wedge areas than its neighbors, according to table 4.5. Since the spikes in figures 4.53 and 4.54 occur on different fragments, the effects on the accumulated running time of the wedge extraction in figure 4.52 even out.

The most critical component of the overall performance in figure 4.52 is the wedge filtering, which occurs during wedge validation. The reason for this is the wedge similarity computation that has to compare all pairs of wedges to determine the amount of similar wedges for each

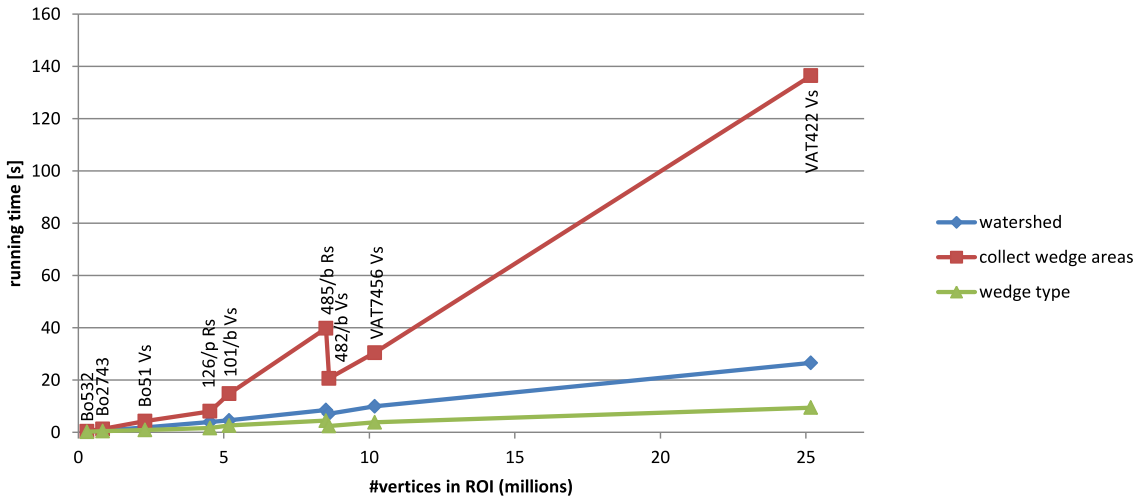


Figure 4.54: Running times of components that depend on the ROI vertex count for the fragments from figure 4.51.

wedge. This results in a running time of $O(w^2 \cdot c)$, where w is the amount of extracted wedges and c is the amount of wedge features to compare. While the wedge filtering for fragment *Bo532* takes up only 4.4% of the overall extraction time, this fraction increased to 63.1% for fragment *VAT422 Vs*. Since the contribution of the wedge filtering is quite dominant, it is excluded from the visualization of relative running time contributions in figure 4.55. In terms of using the extraction pipeline interactively, it has to be noted that the similarity-based wedge filtering is not necessarily required for some tasks. For example, if the dataset does not contain fracture faces and damaged surfaces, or if the non-valid wedges can be easily selected and hidden via a manual viewport operation, the extraction can be accelerated by skipping the wedge filtering. This can cut down the processing times even for large fragments with over 33 million vertices to very workable 200 seconds for a pure wedge model extraction. For cases that require processing larger amounts of cuneiform fragments for statistical analysis, the whole wedge extraction can be used via an automated batch processing mode that can be parameterized in beforehand.

4.11 Component extraction quality

In the following section, an application oriented evaluation of wedge extraction pipeline will be presented to determine the achievable extraction accuracy for some of the wedge model components. This is particularly interesting for assessing the meaningfulness of manual wedge parameter comparisons during the analysis of handwriting characteristics or taking measurements for reconstructing the shape of the cuneiform stylus used. The accuracy of the pipeline depends on multiple factors, with a varying degree of controllability. These include the quality of the scanning process and the extraction accuracy, which will be defined later on, after the specification of the evaluation scenario.

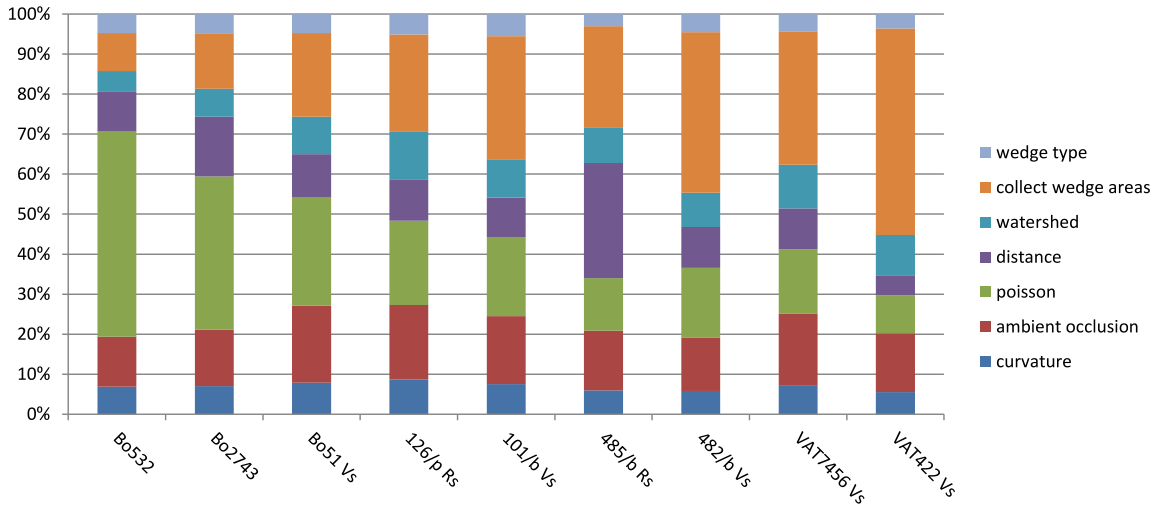


Figure 4.55: Relative running times for the pipeline stages of the wedge extraction pipeline for the fragments from figure 4.51.

The most important prerequisite for evaluating the extraction quality of the wedge model is an exact knowledge of the processed cuneiform fragment geometry. The precise geometry of the scanned artifacts and the precise parameters of the manual and therefore imprecise writing process, like wedge positions or the shape of the stylus, are, however, unknown before taking the actual 3D scan. The only way to obtain a reference geometry for determining the scan quality of ancient artifacts would therefore be to take reference scans using a highly precise scanner with a known and guaranteed accuracy. At the time of writing this thesis, the AIKON OptoTOP HE[1] structured-light scanner employed for digitizing the majority of the available cuneiform tablets was already the highest resolution device accessible for this task. It is affected by several shortcomings, preventing its use as a reliable reference measuring device. On one hand, its vendor specified maximum resolution of $30\mu m$ is negatively affected by the manual operation of the scanner with regard to scan acquisition and scan fusion. On the other hand, only little information about the errors introduced by the geometry reconstruction methods of the bundled AIKON OptoCat software is available. This results in an inability to separate error contributions from scanning and from the extraction pipeline, which is why real cuneiform artifacts were not suitable as a base for an assessment of wedge component extraction quality. The evaluation is therefore based on a method to produce suitable real world phantom data for quality assessment and evaluate the scanning and extraction pipeline as a whole. This way, the extraction results can be compared to a known reference object.

A geometric object, suitable for this task, has to meet several requirements. Its geometrical properties should be well-defined by an accurate and reproducible production process. In addition to that, it should contain a sufficient number of wedges with known position and orientation mimicking real wedges regarding shape and size. As the object should be scanned

[1] www.aicon3d.com

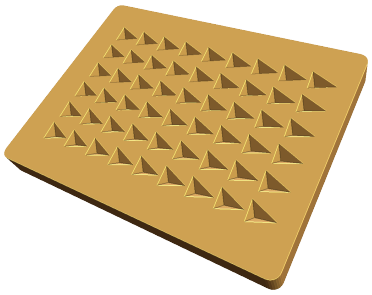


Figure 4.56: Prototype visualization of a suitable reference object for the evaluation of the wedge extraction quality.

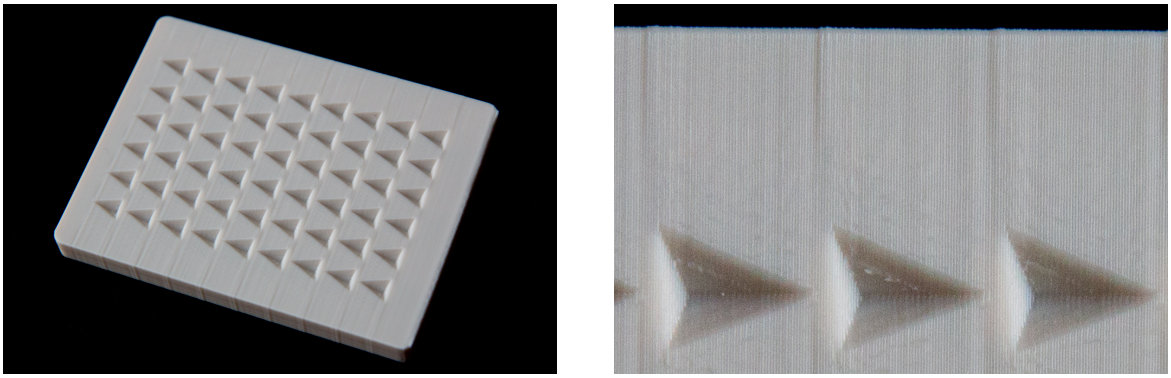
using the same structured-light scanning procedure and hardware used for real cuneiform tablets, the object surface must exhibit similar material properties as a clay tablet. This requirement is especially important, since structured-light scanners are susceptible to scanning artifacts caused by shiny, translucent or polished metal surfaces. Regarding the geometric design of the object, it should exhibit a flat writing surface and a sufficient number of wedge imprints with a shape similar to real cuneiform wedges. The shape of the wedges and their position on the writing surface should be known as precise as possible. In addition to that, the object should be constructed in a way that the number of individual scans, necessary to scan all important aspects of the object is as low as possible. This includes that the objects fits into the scanning area of the available OptoTOP HE scanner.

Therefore, the reference object was designed as a small clay tablet with an evenly spaced rectangular grid of wedges, all facing the same direction. This way, the relative positions of the wedges can be evaluated even if the production process for the reference object exhibits some inaccuracies regarding the object scale. A CAD template draft for such a reference object is shown in figure 4.56. Aside from its simple structure, this type of object exhibits the advantage that the configuration of wedges in the grid can be easily modified to meet different requirements. Some parameters like the wedge scale can be varied per line or per column to measure the size dependence of the extraction quality.

For the production of the reference object, several methods were evaluated to determine the most suitable production process. An initial construction attempt was made, using a prototype Roland DWX-30 4-axis milling machine with a mechanical resolution of 1250 μm on blank blocks of gray colored porous ceramics. Although this material was sufficiently rigid and its surface was suitable for structured-light scanning, the size of available spherical milling heads for this machine was too big ($r = 0.25$ mm) to produce the small concavities of the desired wedge sizes. As none of the accessible milling machines could handle smaller or cone shaped milling heads, producing the reference object using milling was discarded. Taking into account the inability of the subtractive manufacturing process to produce the

required sharp concave features, three additive manufacturing processes were evaluated. As a first attempt, the object was 3D-printed on a Makerbot Replicator 2 FFF (fused filament fabrication) printer using warm gray PLA (Polylactic acid). This printer deposits a heated filament to construct objects layer by layer. While the position accuracy of 11 μm on the X- and Y-axis, 2.5 μm on the Z-axis and a layer resolution of 100 μm were sufficient for the task, the fast oscillating print head movements, necessary to produce the wedges, introduced wave shaped surface artifacts and surface impurities in the wedges and layer alignment issues, some of which can be seen in figure 4.57b. These artifacts could not be eliminated by optimizing the print parameters, like reducing the print head speed and acceleration. The main reason for this was the necessity of a minimum extrusion speed to prevent nozzle clogs. Due to a combination of shininess and a slight translucency the printed PLA surface also turned out to not being suitable for structured-light scanning. A second attempt was made using the Formlabs Form1 SLA (stereolithography) printer with an XY-axis resolution of 300 μm and a minimum layer thickness of 25 μm. This printer works by solidifying layers of fluid resin using a directed UV-laser. The objects produced with the available gray resin material exhibited multiple problematic properties, including distortions on the X- and Y-axis and an even higher degree of translucency than the PLA prints. The non-solidified resin also tended to accumulate in the small concave grooves of the wedges, withstanding the cleaning procedures using isopropyl alcohol. A third 3D printing attempt was made, using an industrial grade SLS (Selective Laser Sintering) printer with a resolution of 100 μm and white nylon as a printing material. This printer works by consecutively placing thin layers of nylon powder on the top of a production area. The areas that are part of the printed object are then melted together by a high power laser. While the SLS printed objects were sufficiently accurate and showed sharp corners even in the concave wedge grooves, the nylon prints had a rough surface with translucency issues and could therefore not be scanned with adequate resolution. Following attempts to fix the scanning properties of the 3D-printed objects by applying matte varnish to the surface failed, because of difficulties to control the varnish thickness on flat parts of the surface and the varnish accumulating in the wedge grooves.

To target the most problematic aspects of the previous production attempts, namely the production of small, sharp and concave wedge features and a non-translucent surface, a clay stamping technique was considered. As a prerequisite, a convex wedge shaped stamp was produced on the milling machine and tested on three different types of clay. The tested candidates included a fine industrial clay with a very low shrinking rate during drying, a non-shrinking polymer clay and a natural clay. The natural clay was supplied by the philologists from Würzburg University, where it is used regularly for cuneiform writing experiments. Of the three clay variants, only the natural clay was able to replicate the sharp concave and convex features of wedges, when processed with the prepared wedge stamp. This resulted from both other types of clay partially exhibiting elastic deformation properties. To prevent the clay from sticking to the stamp surface while lifting the stamp, it was necessary to reduce the moisture of the clay with a short drying pause. The following sections will describe, how



(a) (b)

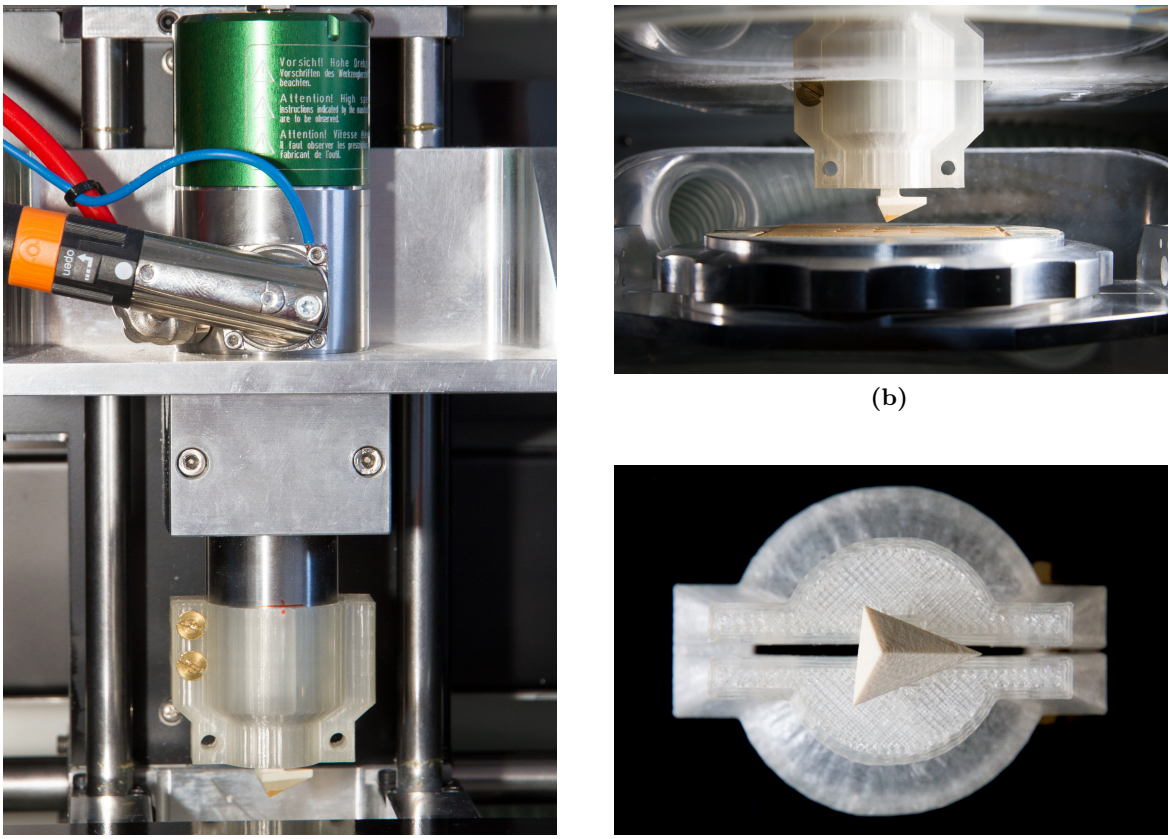
Figure 4.57: A 3D print of the reference object using a Makerbot Replicator 2 with PLA and a layer height of 100 μm. The whole object is shown in **(a)** and a detail image showing the layer structure and printing defects in **(b)**.

the stamping technique was further developed into a reproducible method by converting a milling machine into a high accuracy stamping device.

4.11.1 Milling machine conversion and stamping process

Precise and reproducible control of a wedge stamp requires a method for precise stamp positioning above the clay surface and the ability to produce a linear, rotation free stamping motion without any perpendicular drift. In view of the absence of a suitable robotic arm, an available Roland DWX-30 milling machine was modified to meet these requirements. The machine features a milling head with full XYZ-axis position control and a 1-axis orientable clamping device for holding the workpieces in place, which can be seen in figures 4.58a and 4.58b. Since the precise orientation of the milling heads in the original seating mechanism is not controllable, a stamp holding clamp was produced on the Makerbot Replicator 2 3D printer and mounted on top of the milling spindle. The stamp holding clamp includes screw fastening on the milling machine and the stamp mounting side to ensure a durable connection, which can withstand the unavoidable forces during the stamping process. The mounting hole for holding the stamps is designed with a 5 mm square cross section to avoid any unwanted rotation and stamp shaft deformation. Using this setup, stamps can be switched while precisely preserving orientation and tip position of the stamps.

The supplied orientable workpiece holding device of the DWX-30, which can be seen in the lower part of figure 4.58b, is designed for 98 mm $\times$ 12 mm disk shaped ceramic blanks with a centering groove. These are hold in place by a screw-on aluminum ring. To allow the use of smaller arbitrarily shaped blanks, the machine can be equipped with disk shaped adapter frames. One of the available custom adapter frames, which features a rectangular end-to-end hole with two one sided seating notches, was modified to serve as a creation device for reproducible clay blanks. The adapter frame was extended with a 3D-printed part that

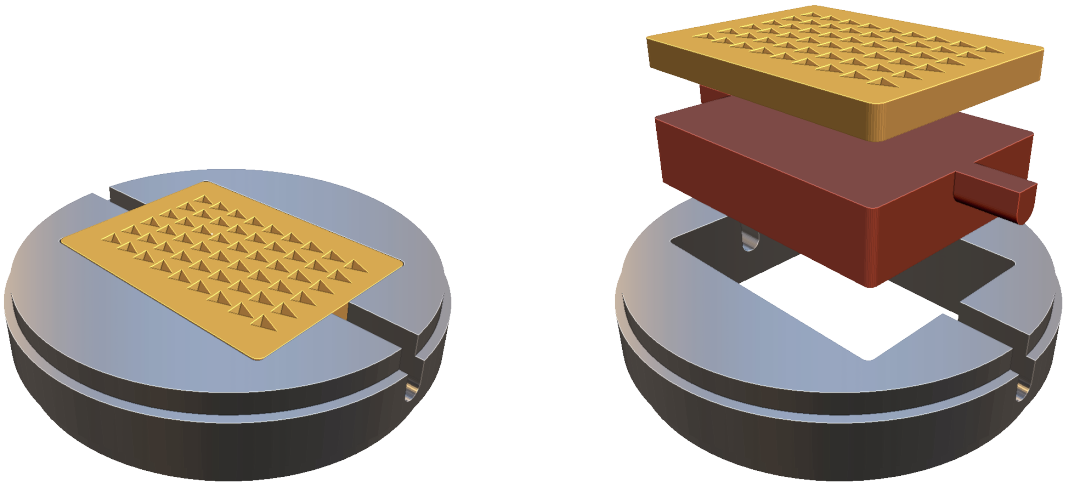


(a) **(b)** **(c)**

Figure 4.58: Detail views of the wedge stamping mechanism. The assembled mount on the milling machine is shown in **(a)** and the stamp above the clay blank during stamping in **(b)**. A top view of the stamp in the mount is shown in **(c)**.

is hold in place by precisely fitting into the seating notches. This converts the adapter into a box shaped container to be filled with moist clay like shown in figure 4.59. To produce a clay blank, the 3D-printed bottom is inserted into the adapter and covered by a thin layer of silicone paper, preventing adhesion of the clay to the surface later on. The created box is then filled with clay, which must be compressed to completely occupy the available space. Excess clay is removed afterwards by steadily and firmly pulling a flat rigid object over the surface, using the borders of the adapter as a guide. This creates a smooth planar surface that is on the same level as the upper side of the adapter. In combination with the well-defined adapter position in the milling machine, a reproducible clay surface for stamping is obtained. Moreover, the stamp, like shown in figure 4.58, is designed with asymmetric wedge faces to produce wedges with an unambiguous rotation and three different inner angles.

Before being able to start the stamping process, the workpiece holder of the machine containing the adapter with the clay blanket has to be leveled. This leveling is usually an automated process where the machine first determines the length of a calibration tool by colliding it with a conductive contact plate of known position. The now known length of the tool is then

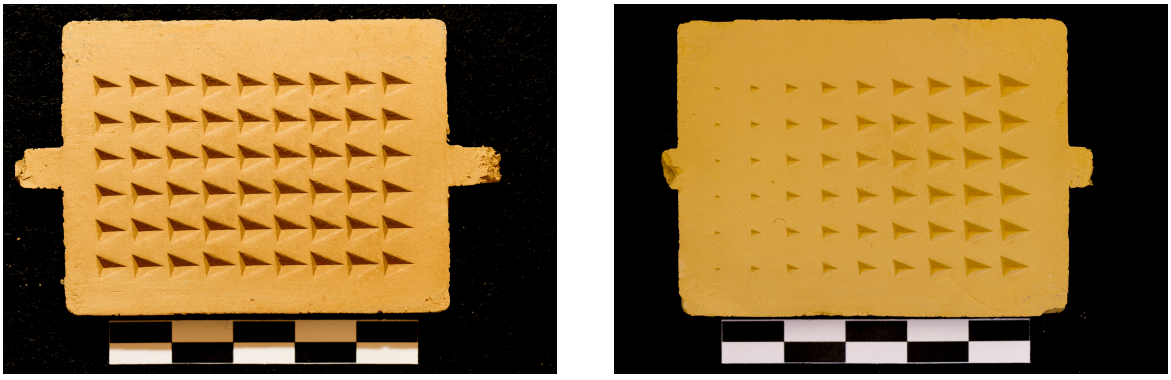


(a) (b)

Figure 4.59: The mounting mechanism for the clay blanks **(a)** and in an exploded view as required for clay removal **(b)**. The assembly includes the adapter frame (gray) the 3D-printed bottom (dark red) and the clay blank (ocher).

used to level the workpiece by palpating its borders with the tool. This process requires an electrically conductive tool, which is why the leveling had to be performed before mounting the stamp holding clamp to the spindle. A precise calibration of the milling machine coordinate system to the workpiece additionally requires precise knowledge of the tool tip position. Since the X- and Y-position can be derived from the known spindle position, the only remaining parameter to be measured is the tool length. However, in case of the ceramic stamps, this could not be measured using the electrical tool length calibration procedure. The tool length of the stamps was therefore measured by moving the stamp tip manually with a high precision jog dial remote to only just touch the clay surface. The respective position of the milling machines Z-axis was then recorded and used to parameterize the stamp motion. This process of leveling and determining the tool length was repeated for every clay blank. In addition to that, every generated stamping control sequence was tested at one-tenth of the real speed beforehand to exclude the possibility of tool collisions with the workpiece holder during travel moves of the machine. At normal travel speed, a grid of 6×9 wedges, as shown in figure 4.60, could be produced in less than 15 seconds.

The high stamping speed was chosen to prevent the clay from adhering to the stamp. To additionally optimize the stamping process with regard to the contact behavior of clay and stamp, three strategies were evaluated. Initiating the stamping process as soon as possible after creating the clay blank, occasionally lead to pieces of clay sticking to the stamp and a muddy pattern on the wedge faces. An attempt to eliminate the stickiness by powdering the tablets with a very subtle layer of flour before stamping was successful, but produced a comparatively rough wedge faces. In a third attempt, the clay was not stamped until the



(a) (b)

Figure 4.60: Stamped wedges on two clay tablets with centimeter scale objects. A grid of 6×9 wedges with uniform scale is shown in **(a)** and grid with increasing wedge scale is shown in **(b)**.

completion of a short drying pause of 5 minutes, which resulted in smooth wedge faces and a sufficiently small stickiness. The duration of the drying pause has to be chosen depending on the clay used and is a trade-off between adhesive surface properties and reduced surface deformation capabilities.

After each stamping process, the adapter frame was removed from the tool holder and the processed clay blank could be extracted by pushing the 3D-printed bottom of the adapter upwards out of the aluminum frame. The clay could then be separated from the bottom piece without causing visible distortions due to the previously inserted silicone paper, resulting in flat clay tablets.

Since there was no opportunity to gain access to the milling machine and the structured-light scanner in one place, the stamped clay tablets had to be dried for safe and damage free transport to the scanner. The issues arising from the drying process will be targeted in the following section.

4.11.2 Clay drying process optimization

Upon drying clay objects, moisture evaporates from the surface that leads to an unavoidable shrinkage of the objects. The observable rate of shrinkage is unique for different types of clay and can be determined by comparing the dimensions of dried and non-dried objects. Although for a homogeneous mass of clay an isotropic rate of shrinkage is expected, the isotropy of shrinkage heavily depends on the shape of the object, the drying speed, and environmental conditions, including temperature, air flow and gravity. To minimize the influence of gravity, a first drying attempt was made by laying the clay tablets on a flat waterproof surface. This strategy resulted in a uniform crescent-shaped distortion of the tablets due to the upper side drying and shrinking much quicker than the bottom side, a result of which can be seen in the right-hand side of figure 4.61. A second attempt of drying the tablets, standing upright on the surface resulted in unpredictable distortions, most likely due to a combination of gravity



Figure 4.61: Result of the optimized clay drying process with an optimized drying result on the left and an unoptimized crescent-shaped drying result on the right.

and non-uniform drying. Increasing the speed of drying by placing the tablet and the drying surface on a radiator only increased the anisotropic drying effects and forced the tablets to crack. Based on these experiments an optimized result without visible distortions was obtained by individually wrapping the tablets in several layers of newspapers and laying them on a flat surface while additionally putting a flat object on top. The 100 g weight of the flat object was carefully chosen to only exert a small but even amount of pressure to the surface to optimize the contact between the newspaper sheets and the clay. In this scenario, the newspaper sheets allowed for a uniform and slow drainage of moisture from the surface, which resulted in a distortion optimized drying process with an approximate duration of one and a half week. An example of a resulting undistorted clay tablet can be seen in the left-hand side of figure 4.61.

4.11.3 Measurements

The relative positions of the wedge depth points and the inner wedge angles are expected to be the features that are least likely to be influenced by possible inaccuracies of the tablet surface, which in turn may influence the wedge size. Therefore, they are the most appropriate features to evaluate the measurement quality of the wedge positions and the wedge angles. While the inner wedge angles are rotation-, scale- and position-invariant with respect to the wedges, the depth point distances are expected to be influenced by the shrinkage of the clay during drying. However, the effects of uniform shrinking can only affect the evaluation of the accuracy, but not the evaluation of the precision.

Regarding the influence of the wedge size on the measurements, a column-wise evaluation of the inner angles on the tablet from figure 4.60b with varying wedge depths form 0.235 mm to 1.25 mm showed that the standard deviation of the angle measurements significantly raises for wedge depths below 0.35 mm in the left-most column with the smallest wedges. The main reason for this is that these wedges are near the size limit up to which the used stamping method produces well-defined wedges with flat faces and sharp edges. Therefore, the measurements to determine the achievable position and angle accuracy are conducted on data-sets with at least two times larger wedges.

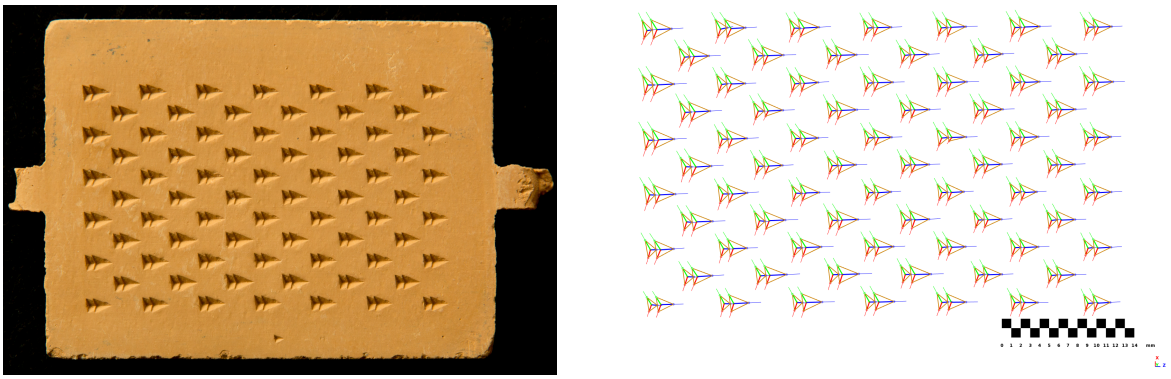
The measured average distances of neighboring depth points, originally stamped on a 6 mm spaced 6×9 4-connected grid with 1.5 mm deep wedges, was 5.7863 mm with a standard deviation of only 0.0256 mm, which indicated a shrinkage to 96.4396% of the original tablet

	Left inner angle	Top angle	Right inner angle
reference	87.88°	88.80°	94.36°
measured	88.39°± 0.80°	90.50°± 0.61°	94.86°± 0.58°

Table 4.6: Measured average inner angles on the tablet with a 6 mm spaced 6×9 grid with 1.6 mm deep wedges form figure 4.60a.

size. This is in good agreement with the overall shrinkage of the tablet to 96.5394% of its original size, measured using a caliper on the tablet boundaries. The low standard deviation confirms a very uniform clay shrinkage and indicates that relative position measurements of the depth points, like used in cuneiform sign comparisons, are both precise and accurate on good quality data. The respective measurements of the inner wedge angles are shown in table 4.6. The observable effect of the measured angles consistently being larger than reference angles of the used stamp has two main causes. On one hand, the wedge models were intentionally fitted without ignoring any data points near the depth point, which results in slightly more shallow wedge edges and consequently, in bigger inner angles. On the other hand, removing the stamp from the moist clay produced a small amount of suction resulting in a small deformation of the wedges, which also leads to shallower wedges and bigger inner angles. The obtained values suggest that an angular accuracy of at least 2° can be achieved.

The influence of superimposed wedges on the precision of the angle measurements was evaluated on the tablet shown in figure 4.62 by separately considering the superimposed (left) wedges and the superimposing (right) wedges. As table 4.7 shows, the standard deviations of the superimposed wedges are higher. This, and the slight change in the measured angles is expected, as the geometry of the superimposed wedges is influenced by the superimposition process after their creation. The increased standard deviation is predominantly a result of slight changes of the direction of the directional edge, which is reflected in the standard deviation of the top angle being much smaller than the standard deviation of the left and right inner angle. As these slight direction changes can also be observed on the real tablet by using optical magnification, it can be assumed that they do not originate from the scanning process, but from an inhomogeneous displacement behavior of the clay during the stamping process. Comparing the absolute angles with the ones obtained for the slightly bigger, non-superimposed wedges in table 4.6, it is noticeable that the top and right inner angle are more closely matched on the superimposing wedges, while the left inner angle is more closely matched on the superimposed wedges. This could be a caused by the atilted top face, which has a bigger contact surface with the clay on the superimposed wedges, and could therefore result in a differing clay displacement. However, this hypothesis could not be verified. The average wedge depth of 0.8 mm for the superimposed wedges, on the other hand, changes only slightly to 0.799 mm for the superimposing wedges, with no significant change in the standard deviation of 0.04 mm.



(a) (b)

Figure 4.62: Stamped superimposed wedges on a clay tablet. On this tablet, a grid of 6×7 double wedges is overlain with a with a 5×6 grid to place more non-interfering samples on the surface. A photo of the tablet is shown in **(a)** and the extracted wedge models in **(b)** with an included millimeter scale.

	Left inner angle	Top angle	Right inner angle
reference	87.88°	88.80°	94.36°
measured (left)	88.64°± 3.41°	91.21°± 1.56°	94.01°± 2.88°
measured (right)	90.83°± 2.12°	89.31°± 0.99°	94.95°± 1.42°

Table 4.7: Measured average inner angles on the tablet with 0.8 mm deep superimposed wedges from figure 4.62. The values are give separately for the superimposed (left) wedges and the superimposing (right) wedges.

A comparison of the positional accuracy with measured values from a real cuneiform tablet is not expedient, because no tablets meeting a precisely known production specification were available. Unfortunately, the only cuneiform documents with a suitable content, namely exercising tablets used by scholars to practice the writing of one sign, are written by scribes which are expected to have a comparatively inconsistent hand writing. However, assuming that a cuneiform tablet is in most cases written using the same writing edge of one specific cuneiform stylus, the precision of the inner angles can be measured for comparison with the values of the stamped tablets. Respective values for tablet *Bo532*, which features comparatively large and clean wedges, are shown in figure 4.8. To consider potential differences, the values were measured separately for each wedge type and combined for all wedge types.

Aside from the differing inner angles that are caused by either a differently shaped stylus or a non-linear stylus motion, the much higher standard deviation of the angles, when compared to the stamped wedges, is immediately noticeable. This suggests that the achievable measurement quality of the wedge extraction pipeline is sufficiently accurate to capture wedge feature variations that are much smaller than the ones produced by a cuneiform scribe. In

	Left inner angle	Top angle	Right inner angle
measured (vertical)	104.40°± 12.25°	107.71°± 14.29°	94.59°± 13.01°
measured (horizontal)	103.42°± 19.05°	105.72°± 18.84°	95.06°± 20.85°
measured (Winkelhaken)	105.34°± 11.60°	109.16°± 12.12°	95.53°± 8.16°
measured (combined)	104.26°± 15.29°	107.30°± 15.86°	95.01°± 15.81°

Table 4.8: Average inner angles on tablet *Bo532* from figure 4.49, measured separately for vertical, horizontal and Winkelhaken-type wedges.

addition to that, the differing standard deviations per wedge type indicate that the hand posture during imprinting a wedge, may have a significant influence on the writing accuracy.

To determine the consistency of the stamped wedges, the measured wedge depths were analyzed. The depths values should exhibit a high correlation to the wedge size, as the orientation and the angle of impression was kept constant for all wedges. As the standard deviations of the measured depth values for multiple stamped tablets in table 4.9 show, the wedge depth were very consistent on each measured tablet. This suggests that the milling machine leveling and the wedge stamping process produced sufficiently uniform and repeatable results. It is noticeable that the measured depth values are consistently lower than the targeted depth values. One reason for this is the clay shrinkage during drying. However, the clay shrinkage measured above does only explain a small amount of the observable wedge depth shrinkage. Another possible reason to explain the observed shrinkage are an imprecise height measurement of the stamping surface, either caused by an imprecise height calibration or an expansion of the clay after flattening the clay surface in the device. While the precise contributions of these factors could not be determined, the measurements on tablets 5 and 6 suggest that potential contributions of these factors are very consistent. However, the clay expansion is expected to be responsible for the large majority of the depth deviation, as the positioning precision of the milling machine adjustment for calibrating the tablet height is 1250 μm. As a last possible factor, the clay expansion while removing the stamp from the clay was examined. For this, the wedges on tablet 7 were stamped multiple times to force a non-reversible clay compression for the wedges. As can be seen by comparing the depth values of the tablets 5 and 6 with the ones of tablet 7, which all share the same wedge depth target value, this did indeed affect the measured wedge depth. This combined suggests that the wedge depth measurements of the presented wedge extraction method are sufficiently accurate. For measurements on real cuneiform tablets, the precision of the depth values is expected to be far less precise than on the stamped tablets. This is not only a result of the expected worse data quality in terms of surface damage, but additionally depends on the curvature of the writing surface and the precise location of the associated Poisson approximation.

In addition to the general suitability of the generated data for statistical script classification on larger amounts of data proven by Michele Cammarosano [38], the measurements on the stamped clay tablets indicate that the achievable accuracy of the wedge extraction pipeline is

Tablet	Depth	Target depth
Tablet 4	1.07 mm ± 0.063 mm	1.5 mm
Tablet 5	1.27 mm ± 0.058 mm	2.0 mm
Tablet 6	1.23 mm ± 0.037 mm	2.0 mm
Tablet 7 (multi)	1.60 mm ± 0.092 mm	2.0 mm
Tablet 8	0.80 mm ± 0.046 mm	1.0 mm
Tablet 8 (right)	0.80 mm ± 0.048 mm	1.0 mm
Tablet 8 (left)	0.80 mm ± 0.044 mm	1.0 mm

Table 4.9: Measured average wedge depth and target wedge depth on multiple stamped tablets.

sufficiently high for obtaining meaningful positional and angular measurements even on small number of wedges. Depending on the data quality of the cuneiform fragments to be examined, the standard deviation of the wedge features of interest should be determined to assess the meaningfulness of single feature comparisons.

4.12 Conclusion

This chapter presented a fast and reliable method for wedge and wedge component extraction on 3D-scanned cuneiform manuscripts. The extraction is supplemented by a similarity-based wedge validation mechanism to identify false detections and to handle fracture faces. It is shown that the extraction rates, robustness and the running time of the method meet the practical requirements to be used interactively in philological research scenarios. The extracted wedge model data can be used for manual script feature comparison and for statistical script analysis in the context of identifying documents with similar script characteristics, which is an important prerequisite for identifying potential fragment joins. Furthermore, the extracted wedge data can be employed to create a digital 3D cuneiform sign inventory.

CHAPTER 5

Cuneiform wedge skeletonization

In this chapter, a skeletonization method is presented for extracting sub-wedge-scale micro-structures from 3D-scanned cuneiform artifacts using a distance-field-based thinning approach. To accomplish this goal, a fast processing pipeline is introduced to extract a skeleton, represented in the form of a directed poly-line graph, from the geometric micro-structure of cuneiform wedges and wedge constellations. The resulting skeletal representation can supplement the segmentation results presented in the previous chapter and provide a base for a 3D cuneiform sign inventory or automated cuneiform sign recognition. After an introduction to the underlying problem in section 5.1 and a definition of application scenarios in section 5.2, section 5.3 will give an overview of existing skeletonization methods including their application to 3D meshes and introduce previous work on graph-based representations of cuneiform script. Subsequently, the extraction pipeline will be outlined in section 5.4, followed by a description of the required construction of a starting area and the approximation of a distance field in sections 5.4.1 and 5.4.2. After that, the actual construction of the skeleton is described in sections 5.4.3 to 5.4.6 and an analysis of skeleton quality and runtime is given in section 5.5.

5.1 Introduction

Model-based cuneiform wedge extraction, as described in chapter 4, works best on cuneiform variants containing a low number of wedge constellations with tightly stacked and largely overlapping wedges, like Akkadian cuneiform. However, this approach does only work to a limited extent when large parts of a wedge are superimposed by other wedge imprints. Although in general, like shown in section 4.10.2, even significantly superimposed wedges may be extracted correctly, segmentation and model fitting become an issue in cases where at least a whole inner edge, more than a whole face, or a depth point of a wedge no longer have an unambiguous geometric correspondence in the 3D model. This includes wedges without a head

and column and paragraph separators which in contrast can still be represented by a skeleton representation. As the initial detection of wedge faces in section 4.8.1 is based on normal clustering, obtaining a usable segmentation of thin wedges, consisting of only two or three rows of triangles, also can be an issue. An additional shortcoming of the tetrahedron-model-based approach is its inability to represent small groove features at sub-wedge-scale and curved inner edges. While the features included in the wedge model are sufficient for identifying scribes based on statistical analysis of the model features [39], they lack an alternative structural representation of micro-features to retain readability in cases of non-successful segmentation. An intuitive wedge micro-structure representation could further be used to conduct reproducible automated measurements on edge curvatures or on the relation of reoccurring micro-features, like markings caused by a damaged stylus. This could help philologists with the analysis of the cuneiform writing process. A compact geometric representation of wedges in form of a directed graph could also serve as a base for a 3D cuneiform sign inventory and for automated cuneiform sign recognition. The former is particularly interesting regarding the fact that traditional 2D cuneiform sign inventories are based on manually copying cuneiform signs in 2D, which always involves a certain amount of subjective interpretation. Data based on an automated 3D extraction process could extend existing inventories with a less biased and, therefore, more trusted representation including depth information.

Computing a stable and clean skeletal representation from 3D-scanned fragments of cuneiform tablets is a challenging task. As described in chapter 3, available datasets acquired under the objective of conducting a philological analysis of the contained cuneiform script often suffer from a variety of geometric deficiencies like overlapping or non-2-manifold meshes, holes, and non-solid meshes. Besides dealing with these deficiencies, a skeletonization algorithm suitable for script analysis has to meet additional philologically motivated requirements. These include the availability of visually distinguishable wedge centers and edge directions and an inherent visual detectability of severe scanning errors. In addition to that, it is desirable to produce a readable representation.

The challenge in achieving these goals is not only connected to technical aspects, like the mesh quality but also to aspects like the structure of cuneiform script itself. Especially superimposed wedges and complex wedge constellations can introduce segmentation relates issues. Figure 5.1 shows some examples for problematic wedge constellations with superimposed wedges which, for some applications, can only be handled insufficiently by the model-based wedge extraction approach presented in chapter 4. A very prominent example in this respect is the Hittite cuneiform sign “nu”, consisting of two superimposed wedges sharing a common depth point as shown in figure 5.1a. Other examples include arrays of horizontal or vertical wedges sharing a common top or side face, like shown in figure 5.1b, and multiplets of tightly packed wedges with multiple missing or very short inner edges, like shown in figure 5.1c. A skeletonization approach could be more suitable in these cases, since it can represent these structures without being restricted by the shape of a tetrahedron wedge model.



(a) (b) (c)

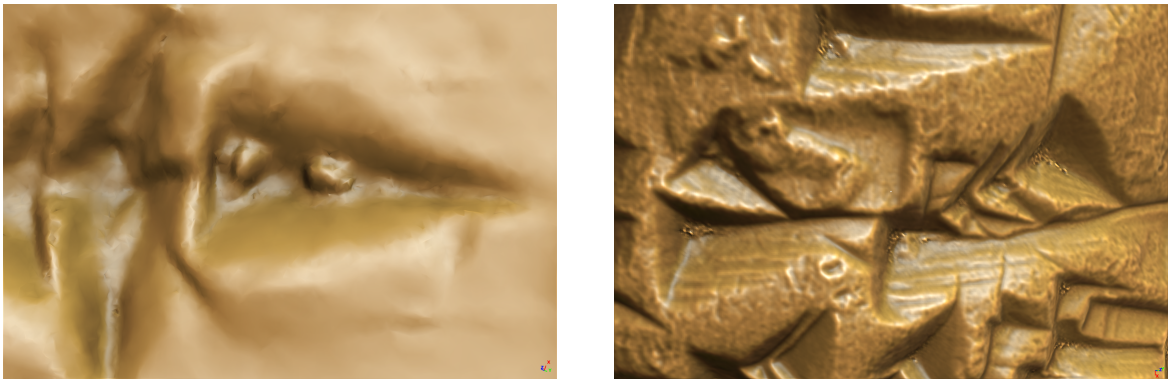
Figure 5.1: Examples of challenging wedge constellations on fragments *1587/u* and *Bo495*. Wedges with common depth point are shown in **(a)**, an array of subtle vertical wedges in **(b)** and tightly packed, nearly indistinguishable horizontal, vertical and Winkelhaken-type wedges in **(c)**.

5.2 Application scenarios

The construction of a suitable skeleton requires defining a set of construction objectives and quality measures, which, for cuneiform research, may vary largely with the intended use case. Skeleton quality should therefore always be assessed with respect to a specific use case or application scenario. The following will therefore define two selected application scenarios in the context of cuneiform script analysis, specifically *wedge micro-feature analysis* and *wedge feature analysis*. Each scenario will be defined including the intended use case and the requirements in terms of desired skeleton features. Adjustments to the skeletonization pipeline to target one specific application scenario will be detailed in the respective parts of section 5.4.

5.2.1 Wedge micro-structure analysis

The first application scenario is wedge micro-structure analysis. Here, wedge micro-structures are defined as sub-wedge-scale features that can not be described by the wedge model used in chapter 4. While some of these features, like dirt crumbs, exhibit a rather coincidental occurrence, others like reoccurring dents on wedge edges or small equidistant grooves on wedge faces may lead to philological insights related to the writing process, in particular, regarding the used stylus. The features of interest are therefore expected to consist of small skeleton branches or dents on branches, whose size or pairwise distance can be measured. Thus, the skeleton should ideally retain all micro-features that are not caused by scanning errors. Some examples of wedges with micro-structures are shown in figure 5.2. While the geometric peculiarities in figure 5.2a consist of dirt crumbs in the center wedge, figure 5.2b also shows ripple markings from a reed stylus in the wedges. In addition to that, the tablet surface in figure 5.2b has a rough texture distinguishing it from the wedge faces. The reed stylus markings and the rough surface structure are much more subtle features than the wedges and were only captured as a result of the much higher scanning resolution in figure 5.2b. This high resolution scan was supplied by the KU Leuven and recorded by using a photogrammetric reconstruction technique [231].



(a) **(b)**

Figure 5.2: Examples of micro structures on cuneiform wedges. **(a)** shows a horizontal wedge with dirt crumbs on fragment *431/s* (Source: [4]), while **(b)** shows multiple wedges with stylus induced ripple structures on fragment *CUNES-49-15-687* (Source: KU Leuven & Cornell University).

5.2.2 Wedge feature analysis

The second application scenario is wedge feature analysis. Although the wedge model extracted in chapter 4 addresses most aspects regarding the shape of wedges, some complex features like curved inner edges or partially filled depth points can not be represented sufficiently. In addition to that, features of wedges without a successful model fitting exhibit no feature representation at all. This affects partially damaged and broken-off wedges, failed model fittings on superimposed wedges, and inverted wedges on cuneiform seals. The features of interest in this application scenario are therefore expected to consist of a clean polygonal representation of the inner edges, including circles in the center of partially filled wedges. Some examples of wedges with complex features are shown in figure 5.3. While most of the wedges can be extracted using the model-based approach, the characteristics of the complex features are not captured, except that they negatively influence the quality of the model fitting. The specific information encoded in these kinds of wedge features varies depending on the type of feature. Characteristic shapes of cuneiform wedges like the large curved Winkelhaken-type wedge in figure 5.3a can be used to directly identify scribes. Other features like the wedge fillings and the crack shown in figures 5.3b and 5.3d can indicate the conservation state of the cuneiform fragment or provide landmarks for joining fractured manuscripts. In contrast, the deformation of the wedges in figure 5.3c allows to determine the order of impression for the participating wedges.

Another important application of wedge-scale features occurs on superimposed wedges, like shown in figure 5.1a, where the wedge extraction from chapter 4 can not separate the individual wedges. A skeleton for the respective wedge area could represent both contained wedges.

Figure 5.4 shows an example for a hypothetical skeleton for the Hittite cuneiform sign “nu”. The skeleton contains six outer branches, one inner branch and two junctions. While the outer branches represent features that correspond to edges of one wedge, the inner branch

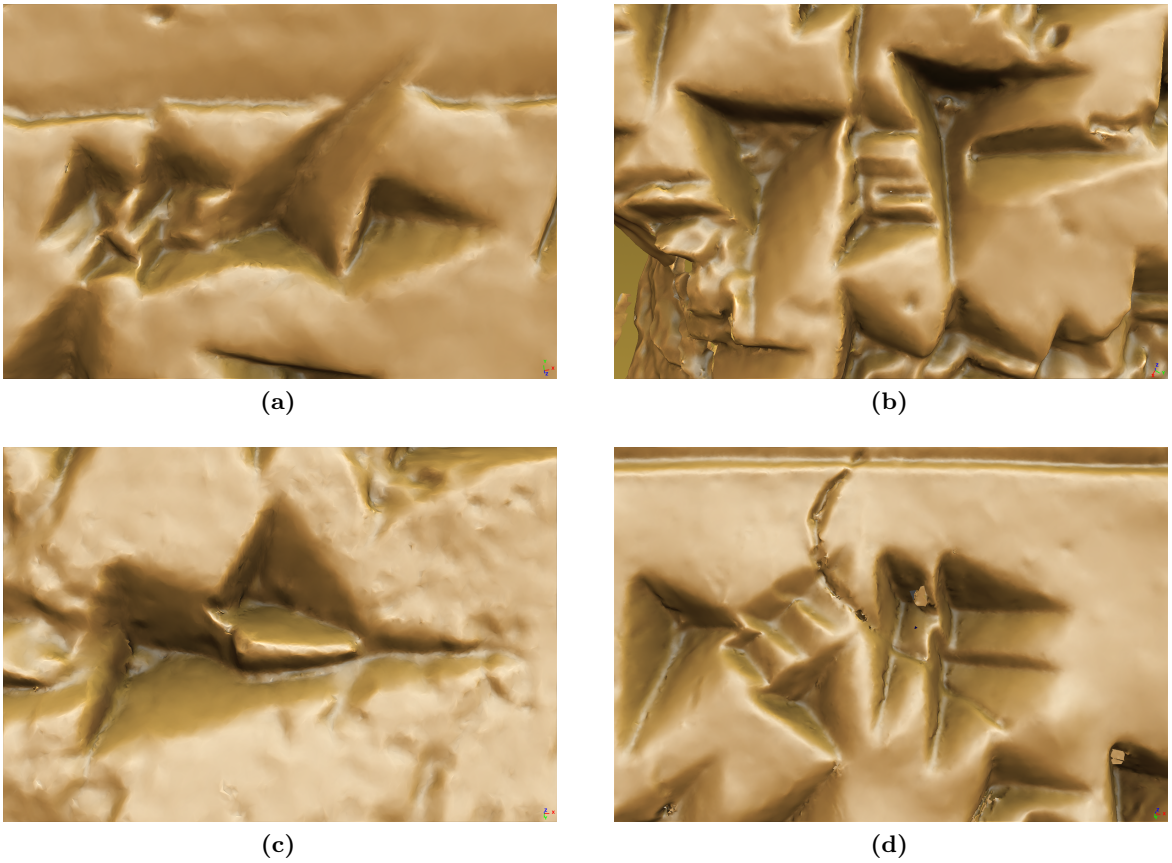
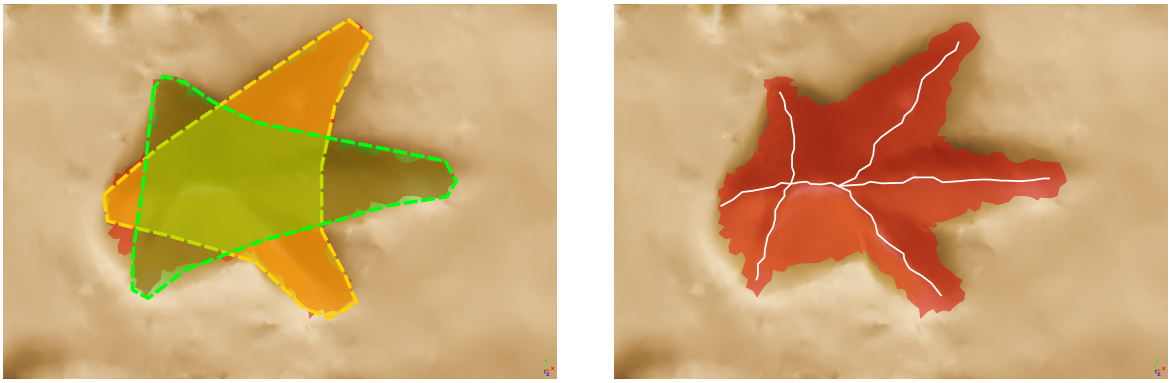


Figure 5.3: Examples of complex wedge features. The long edge of the Winkelhaken-type wedge on tablet *531/f* shown in **(a)**, exhibits a significant curvature, characteristic for the writer Pikku. The two vertical wedges on fragment *Bo77/164* shown in **(b)**, are partially filled and the rightmost one is followed by a horizontal wedge with a missing head. The directional edge of the horizontal wedge on fragment *106/e* shown in **(c)**, is significantly deformed by a superimposed wedge. The wedge group on fragment *Bo83/902* shown in **(d)** is protruded by a curve shaped crack.

represents a section of an edge in both participating wedges. Although the skeleton branches in this example are comparatively straight, the skeleton retains some non-linear features for the wedge edges, which could be used for an extended wedge shape analysis. In addition to that, the shown skeleton is a readable representation. A Hittitologist can identify the sign, even without an explicit branch to wedge assignment or branch direction information and is able to resolve the ambiguity of the inner branch.

5.3 Related work

A skeleton is a thin, in relation to the borders centered representation of a geometric shape that can be used to analyze the structure and the topology of a corresponding geometric object. One of the most common skeleton definitions is closely related to the medial axis that is defined as the set of all points with more than one closest border point on the object [24]. This,



(a) **(b)**

Figure 5.4: Example of a possible skeleton for the Hittite cuneiform sign "nu" on fragment *1587/u*. A wedge overlay with the horizontal (green) and Winkelhaken-type (yellow) wedges is shown in **(a)** and a matching hypothetical skeleton (white) in **(b)**. The red colored area is the wedge area for this sign, extracted with the method from chapter 4.

and equivalent skeleton definitions, like maximally inscribed balls, lead to curve skeletons in a two-dimensional domain and to curve or surface skeletons in a three-dimensional domain [210]. In this context, a curve skeleton can be represented in 2D by polygonal curves and by a set of lines consisting of pixels. Similarly, possible curve skeleton representations in 3D space include polygonal curves and sets of lines consisting of voxels, whereas 3D surface skeletons may consist of polygonal surfaces and thin layers of voxels. The representation of curve skeletons on the surfaces of three-dimensional objects is usually coupled to the representation of the objects. For triangular meshes, this would result in sets of polygonal curves, while voxel-based objects would yield sets of lines, consisting of voxels on the surface. In both cases, it is possible to derive a skeletal representation that does not conform to the representation of the respective object. This however requires the skeleton to not be restricted to the input domain, which for triangular meshes could be defined as the set of mesh edges or in case of a volumetric object, as the set of voxels that the object consists of. A restriction to a discrete input domain, like a triangular mesh, can impose severe impacts on desired skeleton properties, such as centeredness, smoothness and a low noise ratio.

The construction of medial skeletons requires objects with a well-defined surface. For suitable objects $\mathcal{O}$ which are a compact spatial subset of $\mathbb{R}^n$, the surface $\mathcal{S}$ can be defined as $\mathcal{S} = \partial\mathcal{O}$, where $\partial\mathcal{O}$ denotes the boundary of $\mathcal{O}$. The resulting medial skeletons consist of components with at most $n-1$ dimensions. For $n=2$, these components are referred to as medial axis branches, or for $n=3$ as medial axis sheets [210]. Sticking to the medial axis definition, the skeleton components can be further classified into inner and outer branches, junctions, and tips. With regard to continuous curve skeletons, all points $p \in \mathcal{O}$ with at least 2 closest border points are part of the skeleton. Points with more than 2 closest points are called junctions and points with 1 closest border point are called tips. The set of junction points partitions the

skeleton into individual branches, where branches containing a tip are called outer branches, whereas the remaining branches are called inner branches. Removing inner branches always results in a topology change of the skeleton. If skeletons are constructed on a discrete domain, like 3D triangular meshes, the conditions for the aforementioned skeleton components may only be approximately met, especially when employing skeleton simplification methods. Aside from that, skeletons on cuneiform wedges should not necessarily be constructed as medial skeletons with respect to the wedge area borders, but instead with the goal to be in good correspondence with the wedge edges, like in the example in figure 5.4b.

Common applications of skeletons include shape analysis and more precisely, object segmentation, object recognition, and object deformation. In shape analysis, the topological structure of an object is reduced to a skeleton representation with the goal to derive a structural graph from the branches and intersection points of the skeleton. The graph can then be analyzed for specific properties like circles and branch structure or matched against the skeletons of other objects for object recognition [89, 123]. The skeletal structure can also be used to decompose the underlying object into semantic clusters by assigning parts of the geometry to individual skeleton branches [198]. Another application of skeletons is the automated creation of bone structures for object deformation. There, the skeleton is used both for decomposing the model into suitable parts for animation and for providing a sufficiently simple structure for structural manipulation [237]. For three-dimensional objects, the semantic decomposition via a skeleton can rely on a curve or surface skeleton with respect to the object volume, or on a curve skeleton on the object's surface. The second variant can also be used for detecting sharp surface features on 3D models [43, 91, 178].

A common method for constructing 2D skeletons involves the use of morphological operators [194]. Rössl et al. [178] presented an approach to transfer morphological operators from 2D image processing to be applied to the surface of triangular meshes. For this, a curvature-based region extraction is further processed by applying opening and closing operators, modified to work on the new domain. This includes the introduction of a vertex removal rule, building upon the definition of so-called complex vertices and a length dependent pruning scheme. Typical applications of this type of feature extraction are in the field of reverse engineering, where CAD models can be constructed using the extracted feature lines as a base. Surface features on polygonal surfaces can also be extracted using volumetric surface skeletons [91].

Furthermore, skeletons have been used for feature extraction on cuneiform data in the past. With the goal of deriving 2D vector drawings of wedges, Mara presented a method [136, 137] to extract skeletal representations of 3D-scanned cuneiform wedges, based on the computation of integral invariants. The integral invariants are used to extract wedge regions and inner extremal points from the tablet surface and to identify outer extremal points on the wedge contours. The skeletons are then obtained by connecting neighboring inner and outer extremal points with geodesic lines on the mesh surface and serve as a base for constructing 2D spline drawings of the wedges. In this method, mesh errors like holes are dealt with by employing a

mesh repair preprocessing to remove non-2-manifold geometry and fill holes. In addition to that, missing values on the local 2D raster representation of the tablet surface that is used to compute the volume of the spheres for the surface integral invariant, are reconstructed using interpolation. Depending on the mesh error characteristics of the input mesh, mesh repair operations may be an issue, since they construct geometry that is not true to the scanned surface of the cuneiform tablet. Similarly, fixing the intermediate 2D raster representation by using interpolation may lead to data that does no longer represent mesh merging errors, like dual layer surfaces, accurately. However, this should usually only be an issue on meshes with pronounced mesh errors.

Building upon this work, in [29] Bogacz et al. derive a simplified graph representation from the spline drawings. This representation is then processed using graph kernel methods and a similarity metric to perform graph-similarity-based cuneiform sign comparison. On superimposed wedges, these graphs and the intermediate spline representation from [136] require further work to associate their components with individual wedges. This issue was targeted in [30] with methods for wedge-head detection and wedge-arm tracing. The work on vectorizing 2D raster images of cuneiform hand copies was continued in [140] by introducing an extraction method based on bitmap tracing and 2D skeletons.

5.4 Skeleton construction

The skeletonization method presented in this section aims to extract a polygonal line skeleton from cuneiform structures on the surface of a triangle mesh. It uses a thinning approach with morphological and distance-field-based operations and is combined with methods for pruning, junction optimization and simplification. Additional post-processing is applied by reusing the surface distance from section 4.5 and the wedge area assignments from section 4.7 for partitioning the skeletons and deriving directional information. An overview of the skeleton extraction pipeline can be seen in figure 5.5, with the pipeline stages being colored depending on their operational domain. This domain is chosen to accommodate the needs of the respective pipeline stages that can operate on either vertices or edges of the underlying mesh or on derived polygonal lines. The pipeline starts with a computation of the initial skeleton area on the surface. Since the desired wedge surface features are characterized by a positive curvature, a precomputed discrete curvature [145] is used like in [178] to identify this area. A distance field is then computed on the vertices of the initial area. The subsequent thinning algorithm operates on a vertex order derived from the distance field, successively removing suitable vertices until convergence is achieved, which produces an intermediate skeleton in form of a vertex set. This vertex set is then converted into a spanning tree representation consisting of mesh edges. The conversion process includes detection and cleaning of junction points and integrating connectivity and directional information based on the surface distance field from 4.5. Afterwards, the resulting polygonal line skeleton is pruned and its subsets are assigned to detected wedges where available. Depending on the use case, the skeleton can then

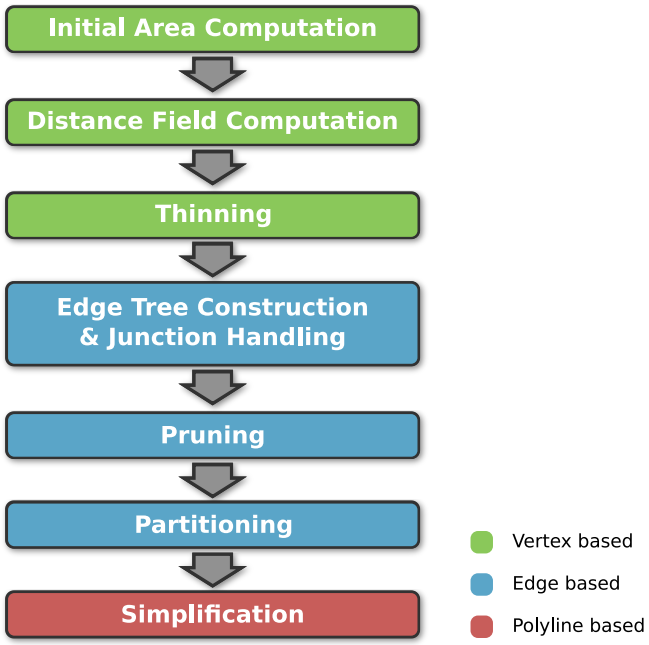
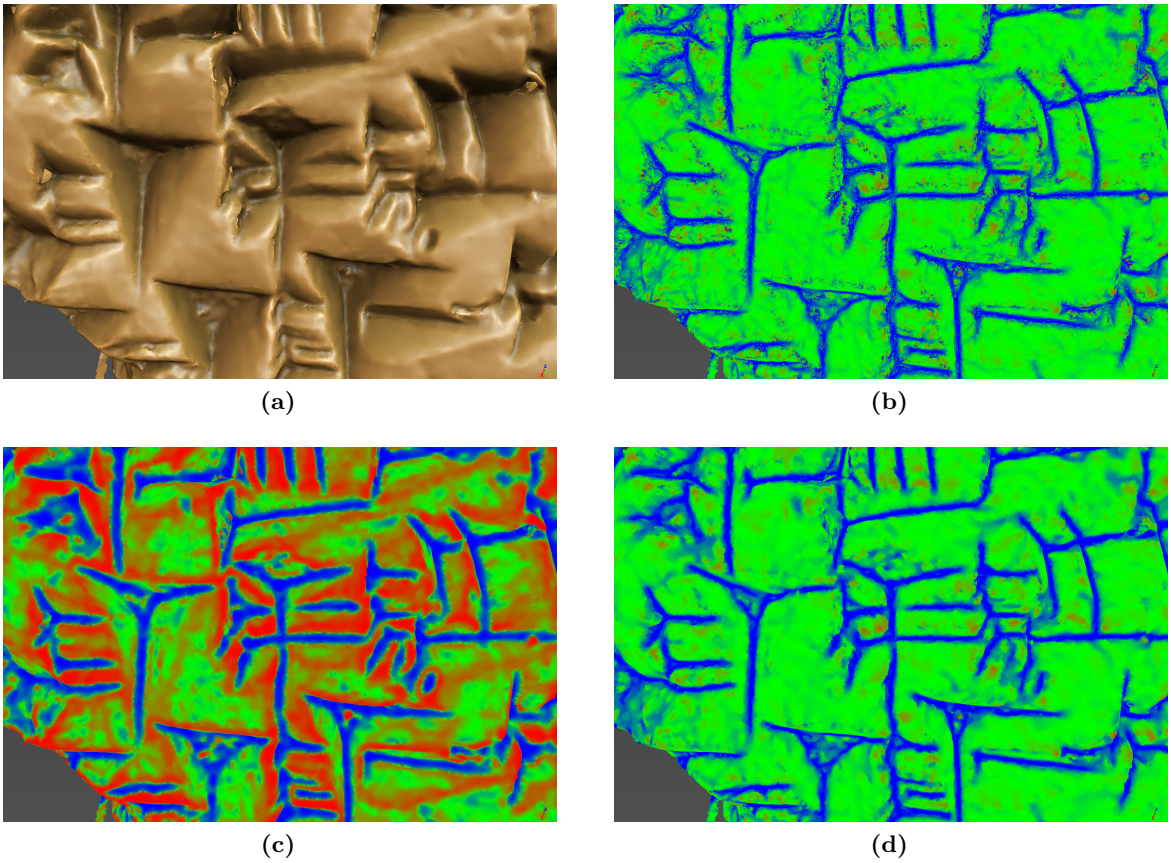


Figure 5.5: The skeleton extraction pipeline, with the pipeline stages colored depending on their operational domain.

be simplified using polygonal reduction methods. All stages of the pipeline will be described in detail in the following subsections.

5.4.1 Initial area construction

Vertex-based thinning on triangle mesh surfaces requires a proper definition of an initial surface area to begin with. This area must contain the complete set of vertices the skeleton will consist of. As the surface features of interest are characterized by a reasonably high negative curvature, a thresholded curvature is used to identify the required subset of vertices. Unlike in [178], the application scenario of scanned cuneiform tablets in this thesis allows a restriction to features with negative curvatures. To counteract the noisy nature of discrete curvatures on 3D-scanned triangle meshes, like shown in figure 5.6b, the curvatures are preprocessed by a median filter. As the comparison of the filtered discrete mean curvature and maximum curvature in figures 5.6c and 5.6d shows, the maximum curvature tends to better reflect small features in the cuneiform wedges and the edges have fewer discontinuities. Therefore, although this comes at the expense of slightly more noise, the maximum curvature is used to obtain stable results. After thresholding, an additional filtering operation is applied to eliminate 1-vertex sized holes in the resulting vertex set. To avoid the removal of long, thin holes and to not modify the overall borders of the initial area, this is realized by a simple 1-ring filter rather than a by a morphological closing. In this process, a vertex is added to the initial surface area if all vertices on the 1-ring are already part of the initial surface area.

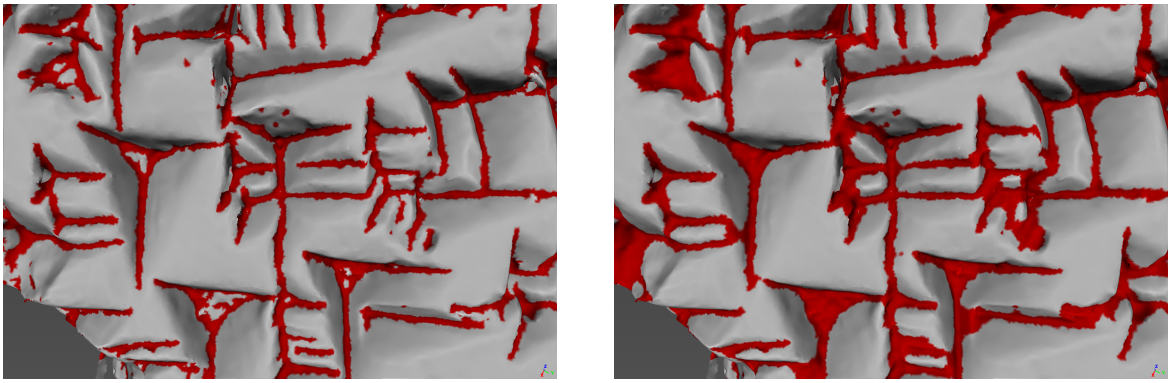


(a) **(b)** **(c)** **(d)**

Figure 5.6: Comparison of discrete curvatures on a section of cuneiform tablet *Bo77/164*. The shaded tablet is shown in **(a)** with a visualization of the discrete maximum curvature in **(b)**. A median filtered version of the discrete mean curvature and maximum curvature are shown in **(c)** and **(d)**. The curvature values are color-coded using a gradient with red for negative, green for zero and blue for positive curvature values.

On particularly noisy meshes or scans containing many soil particles, a moderate use of closing operations can be employed at this point to prevent the resulting skeleton from containing an extensive amount of small circles. As figure 5.7 shows, aggressive use of closing operations is even able to eliminate large circles in the initial area, resulting from partially filled wedges. However, this approach also tends to punctually connect neighboring wedges via low curvature surface areas and to fuse long parallel features of the initial area. The negative effects on the skeleton quality manifest themselves in skeleton branches that are forcefully constructed through the newly created connections and in missing branches. As the resulting branches do not correlate well with curvature features on the mesh, it is suggested to keep the use of closing operations as low as possible.

The required curvature threshold value is determined using a semi-automatic method since it appears to mostly depend on the parameters of the scan reconstruction methods employed by the 3D scanner and the scanner software. These parameters are, in the experience of



(a) **(b)**

Figure 5.7: Visualization of the effect of closing operations on the initial area (red). The unfiltered initial area is shown in **(a)** and a filtered version with a too aggressive 4-step closing in **(b)**.

the author, mostly kept constant for a combination of 3D scanner and application scenario. The threshold is adapted to the characteristics of the 3D scans by computing the standard deviation σ of the curvature values on the mesh and mapping the curvature values from the interval $[-2\sigma, 2\sigma]$ to the interval $[0, 1]$. All values outside the interval are clipped to the endpoints of the target interval. This way, the mapping takes the distribution of curvature values into account, without being influenced by vertex outliers with extreme curvature values. By working on the mapped curvature values, a threshold value yielding satisfactory results on nearly all used cuneiform tablet scans from the Hethitologie-Portal Mainz could be identified. Within this application scenario, the comparatively uniform size of Hittite cuneiform wedges had an additional positive effect on the broad applicability of a constant threshold, which is why for these scans a threshold value of 0.45 is used.

The resulting classification, indicating whether a vertex belongs to the initial area, is handled by introducing a vertex classification scheme for the whole skeletonization process. For this purpose, the vertex set $V = \{v_1, .., v_N\}$ of a mesh with N vertices is extended with a vertex type vector $T = \{t_1, .., t_N\}$. The t_i take on the form of integer values, associated with one of the following vertex types:

$$t_i := \begin{cases} 0 & \text{excluded} \\ 1 & \text{active} \\ 2 & \text{discarded} \\ 3 & \text{junction} \\ 4 & \text{tip.} \end{cases} \tag{5.1}$$

Thereby, excluded vertices are not part of the initial area, while the complementary set of vertices representing the initial area is formed by active vertices. During skeletonization, active vertices are processed and may be either discarded upon removal or classified as junction

vertices or tip vertices. The resulting vertex skeleton thus consists of active vertices and junction and tip vertices. The handling of discarded vertices in a separate class serves the purpose of reducing the number of vertices to be processed in some stages of the skeletonization pipeline.

Considering the mapped thresholding without morphological filtering and the classification scheme, the vertex classification of the initial area can be expressed as

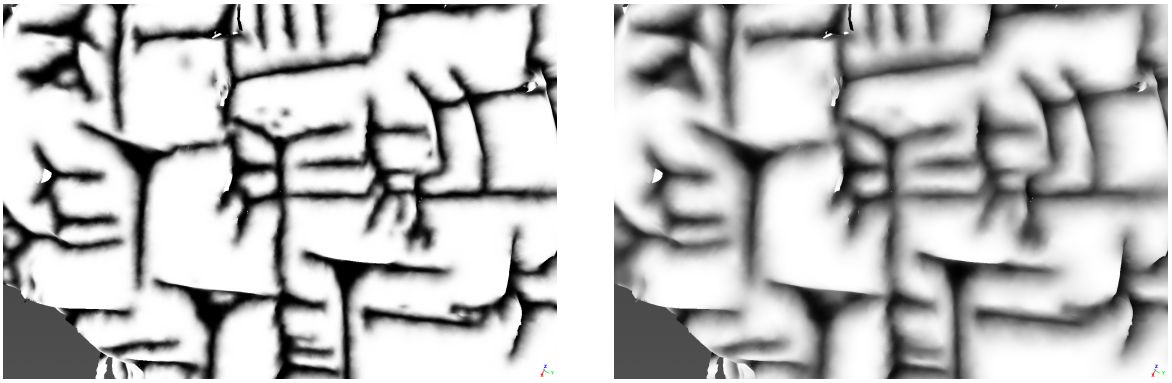
$$t_i := \begin{cases} 1 & \text{for } \min(1, \max(0, \frac{\mathrm{cv}(\boldsymbol{v}_i)}{2\sigma} \cdot 0.5 + 0.5)) < 0.45 \\ 0 & \text{else,} \end{cases} \tag{5.2}$$

where $\mathrm{cv}(\boldsymbol{v}_i)$ is the median filtered maximum curvature at vertex $\boldsymbol{v}_i$.

5.4.2 Distance field approximation

Skeletonization algorithms based on thinning, require mechanisms to enforce several desired skeleton properties, like centeredness and smoothness, and preventing the skeleton branches from uncontrolled shrinkage. While some of these goals can be achieved by an appropriate definition of removable vertices, others can depend on the order of vertex removal. In the particular application scenario of cuneiform wedge skeletonization, target properties would include centering the skeleton in the wedge grooves and extracting long, noise-free skeleton branches that reflect the real length of the grooves. One possible issue of pure morphological thinning methods, like the one of Rössl et al. [178], is that the centering would only depend on the edge distance to the border of the initial area. It has to be noted that for cuneiform scan data, the length of triangle edges on the underlying mesh, as constructed by the scanner software, is usually neither uniform nor strictly correlated to the curvature. Therefore, centering the skeleton in the grooves solely based on edge distances to the borders of the initial area may yield skeletons that do not match well with the perceived center of the wedge edges and have poor correlation with the surface curvature. While geodesic distances can provide a more accurate border distance, their computation in $O(n \log^2 n)$ or $O(n \log n)$ [139] for an approximation, in combination with the required computation of many edge intersections, is comparatively expensive and also does not account for curvature values within the initial area. However, as Tagliasacchi et al. point out for raster images [210], exact centeredness on triangle meshes is additionally constrained by the sampling density and can not be guaranteed anyhow. The proposed approach will therefore approximate a multi-factor distance metric incorporating the border distance of the initial area, the curvature and the surface distance components as needed.

Directly using the maximum curvature values as distance values is not possible without resulting in extremely unsteady skeletons with a very high amount of branches. This is the result of the many local minima of the curvature that favor the formation of many omni-directional small branches in the wedge edges. Due to the round shape of the local minima, this can not be counteracted by smoothing the curvature values or using a scale dependent



(a) **(b)**

Figure 5.8: Color coded visualization of distance fields with too little **(a)** and a sufficient amount of smoothing iterations **(b)**.

curvature like Growing Least Squares [143], without losing the correlation to the shape of the wedge edges. Therefore, a border-distance-based metric is used to obtain stable results. A fast approximation for such a metric can be computed by applying several iterations of smoothing to the 0/1-values of an indicator function like the one of the initial area shown in equation 5.2. For each vertex, this smoothing is realized by averaging the per-vertex function values over a local neighborhood including the vertex itself and its 1-ring. To achieve a good correlation of the distance field with the minima of the maximum curvature, the threshold value should be chosen as small as possible but at the same time big enough to include the wedge edges without interruptions. For this application, using a threshold of 0.3 has been found to produce satisfactory results. In contrast to the initial area, the indicator function here is not post-processed using morphological closing to keep a better correlation with the curvature values. An adaptivity to different mesh characteristics is achieved via the preceding mean deviation controlled remapping of the curvature values. Depending on whether the resulting metric should account for the edge lengths, the smoothing must be applied using appropriate non-uniform edge weights. The number of smoothing iterations is chosen to be as small as possible, but large enough to affect the complete initial area to enforce centeredness. This can be achieved by keeping track of remaining 1-values. A new smoothing iteration is only triggered if the number of remaining 1-values was reduced in the last iteration. This criterion can also terminate the smoothing in cases where the mesh contains isolated 1-only patches which are not affected by the smoothing. In cases of much larger wedge sizes or a strongly increased mesh resolution in the wedges, the threshold value of the indicator function should be decreased to avoid the initial area from requiring to many smoothing iterations. A comparison of two differently smoothed distance fields is shown in figure 5.8. The insufficient smoothing of the field in figure 5.8a prevents a good centering of the skeleton. Since the distance field only influences the processing order of the vertices during thinning, it is no issue that the smoothed field extends over the border of the initial area shown in figure 5.7.

Thinning-based algorithms, like the one described in the following subsection 5.4.3, show a tendency to produce overly shortened branches for pointy features of the initial area. Counteracting such an undesired shortening of skeleton branches during thinning is especially important for branches that represent inner edges of wedges. On meshes, where the wedges exhibit sharp features in terms of maximum curvature near the writing surface, this is naturally achieved by the processing order given by the curvature derived distance field. This results from the fact that vertices with a large curvature at the tip of branches are processed at a time, where vertices near the tip have already been reduced to the maximum possible amount and therefore become irremovable. On meshes with larger areas of uniform curvature in the wedge grooves near the writing surface, this mechanism may however fail. The result are progressively shortening branch areas, violating the skeleton construction target of good coverage. As a countermeasure, the curvature-based border distance values can be combined with a penalty term depending on the surface distance from section 4.5. The penalty term is designed in a way to favor the convergence of the thinning in areas with a bigger surface distance and delay the removal of vertices near the tablet surface. As will be detailed in section 5.4.3, this enables the rules for vertex removal to counteract a too pronounced shrinkage of the branches.

5.4.3 Thinning and vertex classification

After the initial area is defined, an intermediate skeletal representation is constructed by employing a thinning algorithm. As a starting point, the vertices of the initial area are ordered according to their distance values from subsection 5.4.2. The resulting vertex set is then repeatedly processed in ascending order by successively removing border vertices with the procedure being repeated until all vertices meeting a specific border vertex criterion have been removed. Therefore, the order of removal is the essential component for obtaining a suitable skeleton. It depends on the processing order resulting from the distance field and on a condition for identifying valid border vertices for removal. This border vertex condition has to take into account and enforce centeredness, thinness, topology preservation and good coverage as the desired skeleton properties. In the following, the implications of implementing each of this these properties on a triangle mesh will be considered. As detailed in subsection 5.4.2, centeredness is enforced within the limits of the given mesh sampling by the processing order implied by the distance field and can be treated separately from the design of the removal condition. In contrast, a good coverage can be achieved by preventing skeleton branches from uncontrolled shrinkage with respect to their spreading direction. Applied to triangle meshes, this can be translated into the condition to not remove vertices at the tip of branches, while guaranteeing the formation of thin branches before accumulating an unacceptable amount of shrinkage. The closely related property of topology preservation corresponds to the condition of not separating locally connected components, with the connectedness of vertices being equivalent to the presence of a connecting edge. A connecting edge in this case is defined as an edge connecting two active vertices. Lastly, thinness of a vertex skeleton can be achieved by removing as many vertices as possible, without violating the other three design properties.

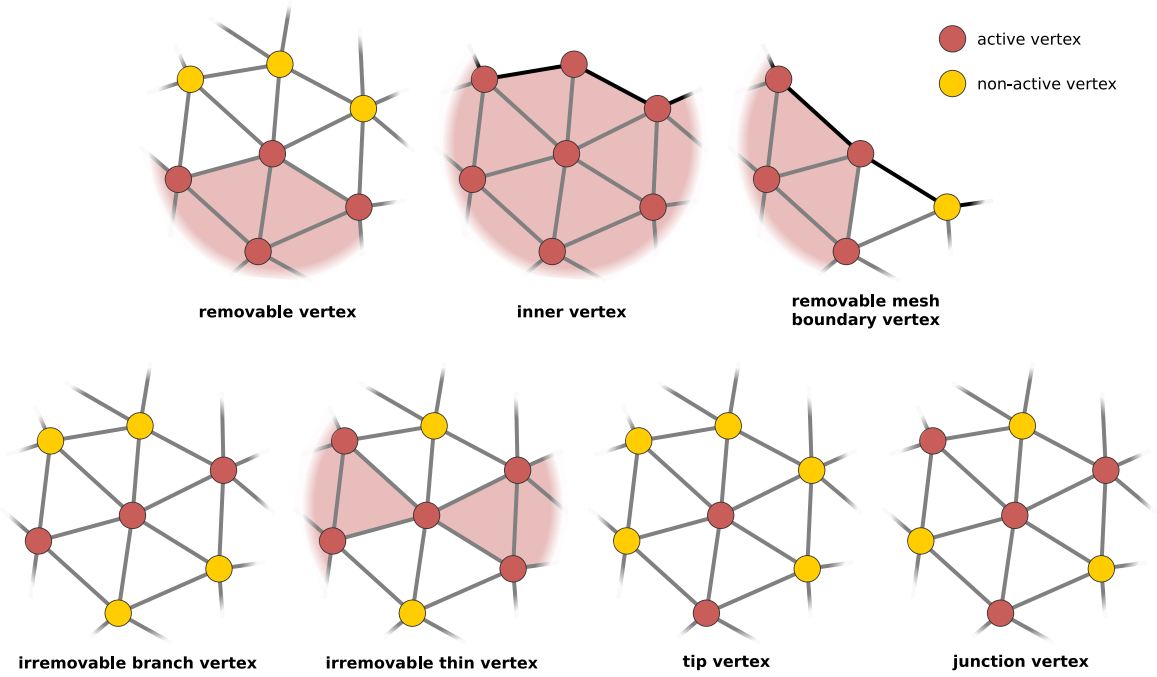


Figure 5.9: Vertex classification examples for thinning, with a visualization of the 1-ring of the respective vertices. Active vertices are shown in red, and inactive vertices in yellow. The associated active triangle area is marked in light-red, the mesh edges in gray, and the mesh boundary edges in black.

Consequently, methods are required to detect whether a vertex forms the tip of a branch or the removal of a vertex will lead to a change in topology. In the following it will be shown that both aspects can be targeted by analyzing the 1-ring neighborhood of a vertex.

To detect whether a vertex $\boldsymbol{v}_i$ can be removed, the values t_i of the vertex type vector are binary filtered to be either classified as active ($t_i = 1$) or as non-active ($t_i \neq 1$). Cycling the 1-ring of a vertex then allows counting the number of transitions ct_i between active and non-active vertices and determining the amount of active neighbor vertices ca_i. Without considering mesh boundaries, it is possible to classify a vertex-based on these two values. As a tip of a branch can be defined as a skeleton vertex that is only connected to one other skeleton vertex, an active neighbor count $ca_i = 1$ is sufficient for detecting an irremovable branch tip vertex. On the other hand, active vertices acting as a local connection of width 1 between two active areas can be detected for a transition count $ct_i = 4$. A simultaneous active vertex count of $ca_i = 2$ then identifies a vertex on a locally complete branch of width 1. Regardless of the active neighbor count, the remaining relevant cases include removable active vertices, with a transition count of $ct_i = 2$ and irremovable junction vertices with $ct_i > 4$. The junction vertices, as will be detailed in section 5.4.4, denote vertices connecting multiple skeleton branches. Some examples for the above mentioned vertex types are visualized in figure 5.9. Each example includes the considered vertex in the center and its 1-ring, as the indicator environment.

While the transition count condition is also used by Rössl et al. [178] as part of their removal condition to determine irremovable, so-called *complex vertices*, they suggest treating all mesh boundary vertices as complex. In the case of 3D-scanned cuneiform wedges with small-sized holes included in the wedge geometry, this would cause issues with skeleton branches sticking to the mesh boundaries. To avoid these issues, the transition count ct_i is modified to take account of mesh boundaries. This is realized by incrementing the transition count in cases, where circulating the 1-ring transitions from an active vertex to a missing face or vice versa. In contrast, transitions from a non-active vertex to a missing face and vice versa must not modify the transition count. As a result, missing segments in the 1-ring of a vertex are treated like segments of non-active vertices, which keeps mesh boundary vertices removable and prevents the skeleton from sticking to the mesh boundaries.

At a point where no further vertices can be removed, the intermediate vertex skeleton consisting of active, junction and tip vertices is regarded as being complete. Thus, according to the definition of the vertex type vector T the vertex skeleton S_v can be expressed as

$$S_v = \{\boldsymbol{v}_i | t_i \in \{1,3,4\}\}. \tag{5.3}$$

To derive the desired polygonal skeleton representation, the following section will describe a method to convert the vertex skeleton into an edge skeleton.

5.4.4 Edge tree construction and junction handling

On a triangle mesh, a trivial edge skeleton S_e can be obtained from a vertex skeleton S_v by defining the edge skeleton as the set of all edges $(\boldsymbol{v}_i, \boldsymbol{v}_j)$ connecting two distinct vertices $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ of the vertex skeleton:

$$S_e = \{(\boldsymbol{v}_i, \boldsymbol{v}_j) | \boldsymbol{v}_i \in S_v \wedge \boldsymbol{v}_j \in S_v \wedge \boldsymbol{v}_i \neq \boldsymbol{v}_j \wedge i < j\}. \tag{5.4}$$

The condition $i < j$ forces all edges to be treated as directed and therefore prevents multiple inclusions of edges into the edge skeleton. S_e can be efficiently constructed by traversing the vertices of S_v in a depth-first search manner to iteratively add valid edges to S_e. It has to be noted that while the underlying vertex skeleton S_v satisfies the thinness condition that a further removal of vertices would break the topology by construction, the trivial edge skeleton S_e does not. Depending on the specific skeleton, it may be possible to remove inner edges from S_e without changing the set of connected components in S_v. This results from the fact that the trivial edge skeleton may contain unwanted circular edge loops in mesh regions with junctions. While, as figure 5.10 shows, the case of three connected junction vertices can be solved intuitively by breaking the longest edge, junctions with more than three associated branches or directly adjacent junctions can get increasingly complex.

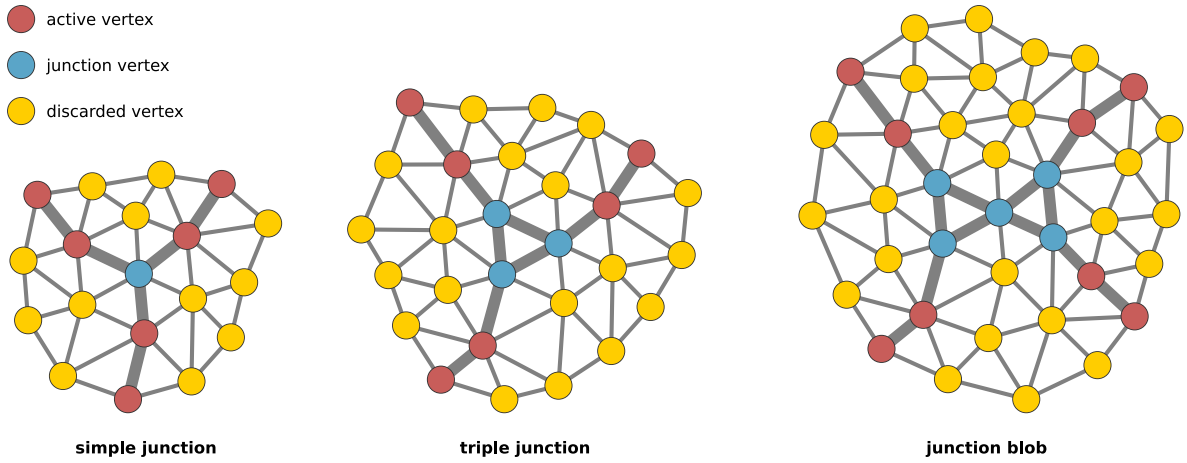


Figure 5.10: Examples of mesh sections with three common junction area types after thinning. A trivial junction is shown on the left, a triple junction in the middle and a complex junction blob on the right. The associated branches of S_e are highlighted in bold.

In this regard, the case of n adjacent junction points forming a circle, with one vertex at the center connected to all outer ones, deserves special attention. While the center vertex could be removed with a special post-processing removal rule and the result would consist of n junctions, it could be more appropriate to exclude the edges forming a circle in the skeleton, resulting in only one junction. The occurrence of this type of junctions has, however, been found to be very rare. This is partly being attributable to the average degree of vertices in a closed 2-manifold triangular mesh being 6. As common remeshing methods, like employed by the scanner software, do not favor vertices with a degree significantly deviating from this average, the amount of vertices with a high degree in a mesh is usually very low. Junctions of this type will therefore not receive any special treatment during junction optimization.

Completely omitting circles to avoid issues in junction areas is not an option, as the constructed edge skeleton is expected to contain valid circles in cases of overlapping or partially filled wedges and impurities of the clay surface. To resolve the issue and meet the skeleton design target of smoothness, the edge skeleton in junction areas is therefore approximated by constructing minimal spanning trees for these regions using Kruskal's algorithm [119]. The required graph for spanning tree construction is determined by applying a special morphological dilation operator on the set of junction type vertices, which expands the set but affects only vertices contained in the thinned vertex skeleton S_v. This results in a vertex set which consists of many small regions of vertices around junctions that are only connected to thin, separated skeleton parts. The required graph is then created by including all edges between these vertices. Using the edge lengths as weights, Kruskal's algorithm therefore yields a circle free set of valid sub-skeletons that connect the remaining skeleton parts in junction areas. After resolving the junctions, the vertex types are reevaluated, since some former junction vertices may no longer fulfill the junction criterion based on the corrected edge skeleton. It has to be mentioned that it is generally possible to construct cases of any desired number of connected junction areas

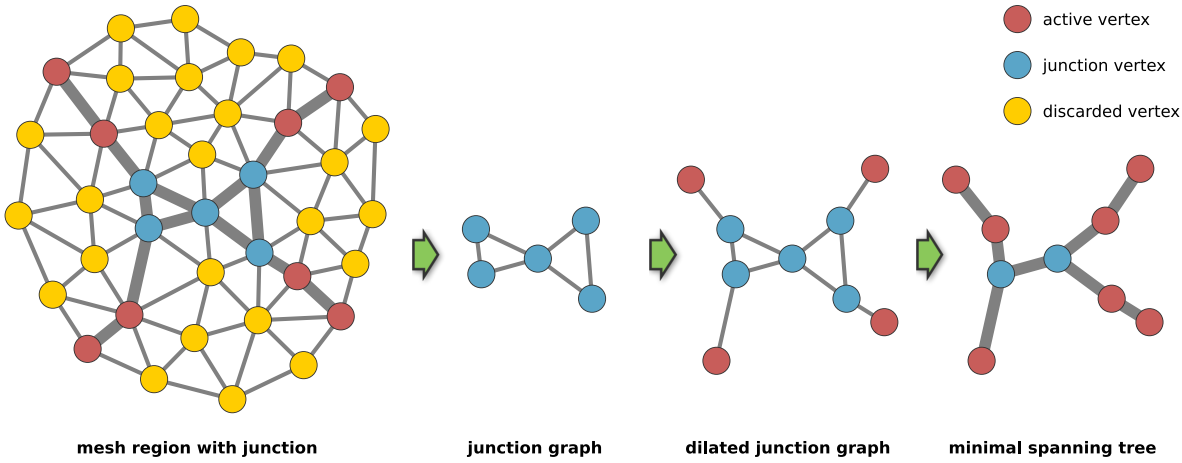


Figure 5.11: Resolution of complex junction areas through the construction of spanning trees on the induced graph.

which as a whole form a big circle. In these cases, Kruskal's algorithm would exclude one important edge from the spanning tree to prevent the formation of a circle. Junction constructs of this type are, however, highly unlikely to occur on 3D-scanned cuneiform meshes and have never been observed on the 3D scans used in this thesis. Resolving only the junction areas by the construction of a spanning tree instead of running Kruskal's algorithm on the whole vertex skeleton serves two purposes. On the one hand, it enables the resulting edge skeleton to contain intended circles and on the other hand, it reduces the amount of necessary edge length computations. An example for such a junction resolution is visualized in figure 5.11. The junction is extracted from the mesh, dilated and then, the spanning tree is constructed on the induced graph, yielding a thin edge skeleton resolving the junction. It has to be noted that the junction blob for this example has been constructed so that the minimum spanning tree includes two junction vertices. A different set of edge length for this junction region could also have led to a spanning tree containing only one junction vertex in the center. For performance reasons, all necessary operations for junction handling are performed on the mesh using vertex labeling.

A real world example for the distribution of junction areas can be seen in figure 5.12, using the same color scheme as in figure 5.11. From the contained 58 junction areas, 32 are non-trivial, with the most common non-trivial junction type being a triple junction, like shown in the middle of figure 5.10.

Depending on the use case, the resulting skeleton at this point usually exhibits a significant amount of over-segmentation in form of skeleton branches representing non-relevant features. In addition to that, the skeleton can be improved by adding directional and connectivity information by fusing branches and extracting a simplified polygonal line representation. These goals will be targeted by the post-processing pipeline stages for partitioning, pruning, and simplification which will be presented in the following sections.

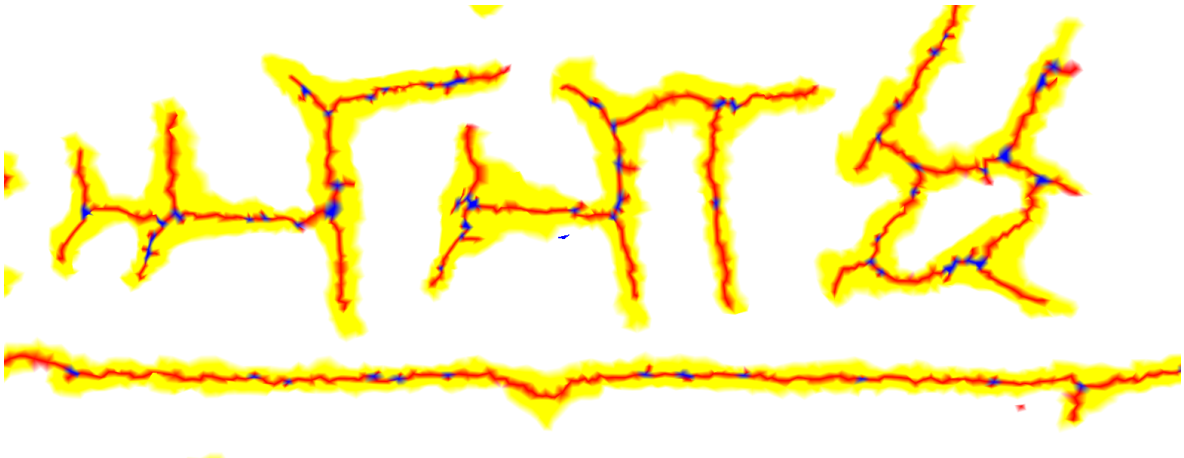


Figure 5.12: Example of an non-pruned skeleton on cuneiform fragment *7/a*, color-coded on the mesh vertices. Discarded vertices are shown in yellow, junction vertices in blue and active skeleton vertices in red.

5.4.5 Skeleton partitioning

The junction resolved edge skeleton created in subsection 5.4.3 consists of a set of labeled mesh edges. Subsequent branch-based processing, like the simplification, requires this edge set to be transformed into a polyline representation. At this point, a branch of the skeleton is represented by a linearly connected set of junction-free skeleton edges, delimited by tip or junction vertices or forming a circle. However, on cuneiform data, this representation may include branches belonging to multiple wedges, constructed as a result of a connected initial area of these wedges. To confine branches to one wedge, the skeleton can be split based on the watershed regions constructed in section 4.7 during wedge extraction. Since the visualization of the pruned and simplified skeleton branches is polyline-based, the partitioning of the skeleton will be described before the pruning and simplification stages although it is executed in between pruning and simplification.

Accordingly, polygonal lines for the skeleton branches can be constructed from the edge skeleton by determining connected sets of edges. This is realized by an iterative depth-first search edge tracing algorithm. Starting at an unvisited active vertex, the algorithm adds all neighboring vertices that can be reached through an unvisited skeleton edge to a stack. For this, the method only considers edges that are part of the spanning tree computed for junction resolution. The stack vertices are processed in the same manner, adding reachable active neighbor vertices to the stack. Processed stack vertices are labeled as visited and appended to the polyline being constructed. Processed skeleton edges are labeled as visited, when adding the respective target vertex to the stack. At a point, where the reachable neighbor vertex is either visited or non-active, the tracing algorithm not only adds the respective edge to the polyline but also switches to backward mode. This results in all following vertices being prepended to the polyline. If a visited or non-active vertex is reached in backward mode, the respective edge is prepended as the last edge to the polyline. Consequently, the polyline

for the branch is complete and the algorithm continues by finding the next unvisited active vertex for another branch and adding it to the stack. This way, all skeleton edges are added to exactly one polyline, representing one branch. Operating the algorithm on the watershed regions from section 4.7 one by one, automatically constructs branches confined to individual wedges. In contrast to tracking a skeleton-edge to watershed-region assignment, this also allows processing manually created overlapping wedge areas, because unambiguous labels can be corrected in beforehand. As a last step, directional information is derived from the surface distance of the polyline endpoints. For the purpose of producing ascending polylines with respect to the tablet surface, the direction of a polyline is reversed if the first point has a smaller surface distance than the last point. A simplified version of the polyline tracing algorithm in pseudo-code form is shown in figure 5.13.

```
1  backward = false
2  find next unvisited active vertex and add it to S
3  if (S.size == 0) abort;
4
5  while (S.size > 0)
6  {
7    get vertex v_i from  S
8    mark v_i as visited
9    if (backward) prepend v_j to polyline
10   else append v_j to polyline
11
12   found_edge = false;
13   find v_j, with unvisited spanning tree edge (v_i,v_j)
14   {
15     found_edge = true
16     if (t_j != active || (v_i,v_j) is visited)
17     {
18       if (backward) prepend v_j to polyline
19       else append v_j to polyline
20       backward = true
21     }
22     else
23     {
24       add v_j to S
25     }
26     mark (v_i,v_j) as visited
27   }
28   if (!found_edge && S.size == 1)
29   {
30     backward = true
31   }
32 }
33 if (surfaceDistance(polyline.start) < surfaceDistance(polyline.end))
34 {
35   reverse polyline
36 }
37 repeat
38
```

Figure 5.13: The polyline tracing algorithm, with $\boldsymbol{v}_i$, t_i, $(\boldsymbol{v}_i,\boldsymbol{v}_j)$ and S denoting vertex i, the type of vertex i, an edge between vertex i and j and the stack S.

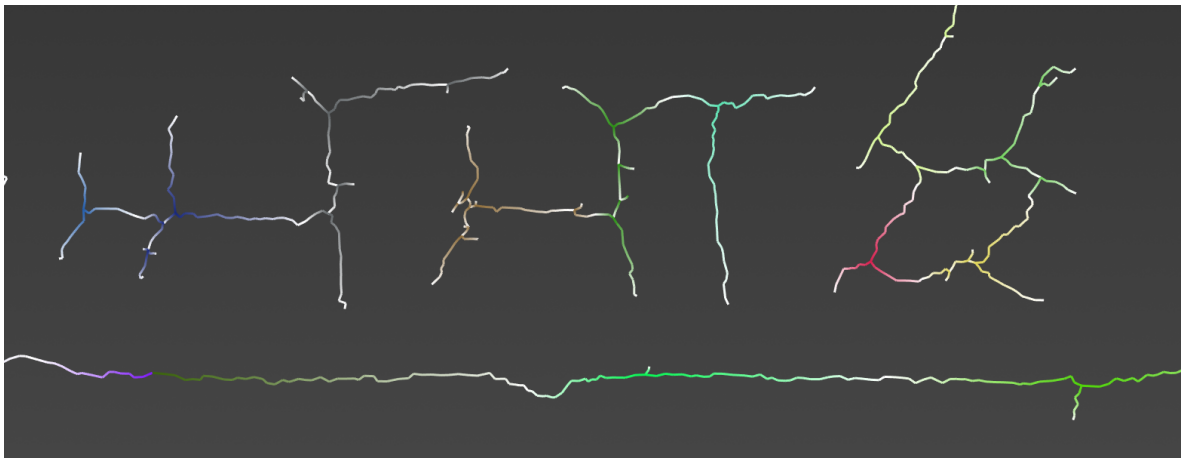


Figure 5.14: Example of a non-pruned skeleton on cuneiform fragment *7/a* in the form of polygonal lines. The branches are colored with a gradient blending from a randomly assigned wedge color to white, based on the surface distance.

Figure 5.14 shows the polygonal line resulting from the skeleton example in figure 5.12. It can be seen that all junctions have been properly resolved, since the edge skeleton does not contain any small circles at the junctions. The directional information is visualized as a color gradient starting with a random per wedge color at the polygon end with a larger surface distance and blending to white. Since the wedge mapping is derived from the wedge areas from chapter 4, associated skeleton branches share the same colors. It is worth mentioning that the paragraph separator at the bottom which consists of multiple wedge areas, has been skeletonized successfully, but is fragmented into four segments. Nonetheless, it has a continuous skeleton representation on the mesh which could be used to fuse the respective wedge areas heuristically. In contrast, the wedge model fitting from chapter 4 can not handle the paragraph separator, due to the missing wedge heads. The numerous tiny branches in the shown skeleton will be handled later during skeleton pruning.

At this point the skeleton has been converted to a branch-based polyline representation. To facilitate further processing, polyline adjacency lists are created for the skeleton junctions, as part of the tracing algorithm. This effectively extends the skeleton representation to a directed polyline graph.

5.4.6 Pruning and simplification

Skeleton pruning describes the process of removing skeleton branches to counteract over-segmentation and is almost always a trade-off against over-simplification. The core element of pruning is therefore to rate the importance of branches and selecting branches for removal. Since the definition of over-segmentation strongly depends on the application scenario, pruning strategies can only be employed to improve a skeleton with respect to application specific quality criteria. This section will therefore describe two pruning strategies that target the application scenarios described in section 5.2. The pruning is performed on the labeled skeleton

edges of the mesh, since structural changes of the skeleton as a result of branch removal can be handled more efficiently on this representation.

The first application scenario considered is wedge micro-structure analysis. Apart from potentially indistinguishable mesh features resulting from errors during the scanning and scan merging process, the micro-features of interest are expected to exhibit a minimum size and a minimum significance regarding the occurring curvatures. Therefore, pruning in this application scenario will be reduced to the definition of a measure of minimum importance which only excludes branches with a very low expected relevance. An aggravating factor of defining a measure for the minimum required relevance is the fact that aside from the small size, no general assumptions about the positions and shapes of the micro-features within the wedges can be made. An exception is the possible presence of circular features which introduces the requirement to not remove inner branches. The pruning in this pipeline will therefore only consider outer branches for removal. Rating the importance of branches is performed depending solely on branch length, which can either be measured as the sum of the edge lengths or as the amount of edges the branch consists of. While both measures require the choice of a threshold, the edge count is the more suitable measure for the data used in this thesis. This results from the skeleton at this point consisting only of mesh edges of a 3D-scanned object. Taking into account that the object was scanned using an appropriate resolution and the scanning software employed some kind of curvature adaptive remeshing, the edge length is related to the local amount of detail in the scan. A branch length measured by its edge count therefore exhibits the advantage of being linked to the local amount of detail in the mesh. This simplifies the choice of a threshold that can be used on a wide range of datasets. In contrast, for scan data with a comparatively uniform vertex density, a branch length based on the sum of edge lengths would be more appropriate. This would, however, require the threshold to be chosen depending on scanning resolution and the average size of wedges in the scan. For the datasets processed in this thesis, thresholds in the range of $[1,3]$ produced satisfactory results. While removing branches of length 1 almost always helped with reducing over-segmentation, choosing a threshold bigger than 2 already showed a tendency to remove many significant micro-features. Only on especially noisy meshes, choosing a larger threshold was necessary.

Figure 5.15 shows the polygonal skeleton from figure 5.14 after pruning with the intent to be used for wedge micro-feature analysis. In the example, most small branches have been eliminated which causes most skeleton branches to represent single inner edges of wedges. However, despite several remaining tiny branches, only the small branch on the right-hand side of the paragraph separator is an example of a useful micro-feature, since it represents a position at which the stylus may have been reoriented during drawing the separator. The micro-branches on the red horizontal wedge in the middle result from scan merging artifacts and the remaining ones are from forceful connections between touching wedges which are cut at the wedge area transitions. Although the inter-wedge connections do not necessarily conform to the definition of a micro-feature, they can be used as in indicator for overlapping

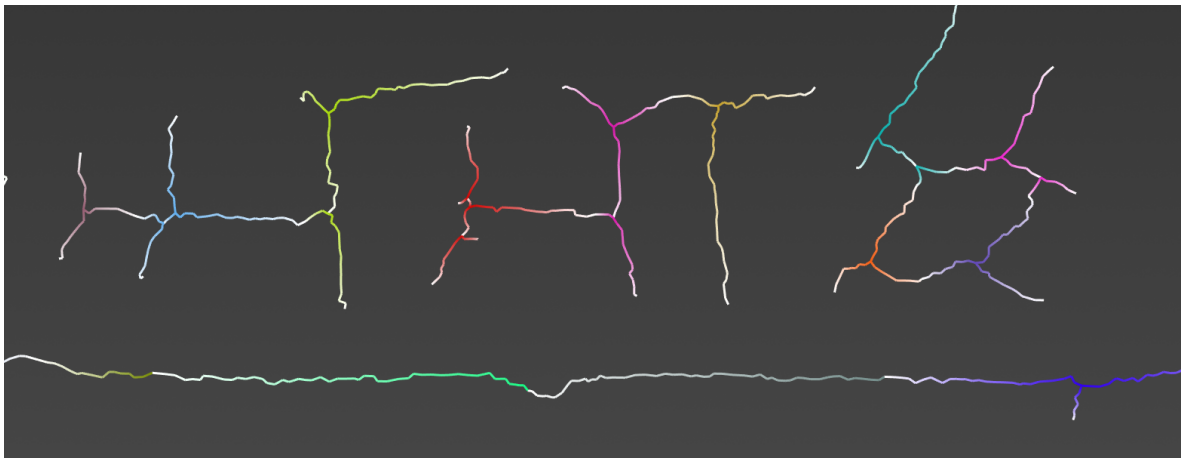


Figure 5.15: Example of an non-pruned skeleton on cuneiform fragment *7/a* in the form of polygonal lines. The branches are colored with a gradient blending from a randomly assigned wedge color to white, based on the surface distance.

wedges. This information could be used to extend the extracted wedges from chapter 4 with connectivity information that can not be derived from the wedge area adjacency with the same precision. It is worth mentioning that methods for automated processing of the generated feature-rich skeletons are not restricted to the geometric information of the branch vertices and edges. The vertices can be extended with additional information derived from the underlying mesh, such as the surface distance, ambient occlusion or curvature values.

The second considered application scenario is wedge feature analysis. To target the requirements mentioned in subsection 5.2.2, the pruning strategy has to enforce the removal of micro features while preserving a clean representation of the inner edges of wedges. This includes having more than three inner edges in wedge areas with superimposed wedges. In a second stage, the skeleton branches are simplified, since long branches no longer need to be densely tessellated to represent micro-features.

A key observation regarding a more aggressive pruning than employed for micro-feature analysis is the fact that the skeletons often contain many small branches of similar size and some of these branches represent important features. The employed pruning strategy will therefore target an ordered removal of branches, beginning with the least important ones. If the importance of branches is rated by their edge count, this can be an issue, because the fulfillment of an edge-count-based removal condition is likely to change for other branches when a branch is removed. This is the direct consequence of removing only outer branches which touch the remaining skeleton at exactly one junction. If this junction connects three branches, the remaining two branches get fused and the junction is removed.

To solve this issue, the pruning method for wedge feature analysis employs multiple pruning iterations to remove branches. Starting with a length of 1, branches of increasing length are removed until a threshold is reached. If, for example, a branch length threshold of 6 is

specified, the method will first remove all branches of length 1, then reevaluate the branch lengths based on the removed junctions and continue with the branches with a length of 2 and so on. As a result, important branches with a length smaller than the threshold length can be preserved, because they get fused with other branches, thus preventing their removal. With respect to the cuneiform wedge features, branch fusion addresses the issue that skeleton branches of the pruned polyline skeleton not necessarily correspond to whole inner edges of wedges. Instead, a single inner edge may represented by multiple branches. Fusing those branches is a simplification that improves the representation of semantically connected features and also enables a more intuitive visualization.

Selecting a suitable threshold is an essential component for obtaining usable skeletonization results. According to the author's experience, the threshold is most sensitive to changes of the scanning procedure, which includes scanning resolution and scan merging parameters. A second important factor is the general roughness of the scanned surfaces. Especially on high resolution scans that capture large amounts of small details on the rough clay surface, separating important wedge-scale features from unimportant noisy surface features becomes increasingly difficult. For the large majority of cuneiform fragments used in this thesis, thresholds between 5 and 7 worked well, because the datasets were recorded with a well evaluated and consistently executed scanning procedure, including the merging of scans in AIKON OptoCat[1]. An automated approach to compute the pruning threshold is not presented, since the requirements for the specific characteristics of a skeletonization for manual processing are highly user specific. An advantage of the low computation times of the presented method is that multiple different pruning parameters can be tried out in a few minutes, even on large cuneiform fragments.

In cases with a large amount of noise, a scale dependent importance measure, like employed by Mara et al. [138], could be superior to the discrete curvatures employed here to determine an initial area that is less susceptible to noise. However, using similar techniques like Growing Least Squares [143], decouples the ROI borders from the curvature adaptive structure of the underlying mesh. This can exert a negative influence on a border-distance-based thinning strategy, resulting in even more unwanted branches. As a countermeasure, the thinning can be coupled to the scale dependent function used to determine the ROI or to geodesic distances on the surface, which can significantly increase the computation times. Likewise, a scale dependent importance measure on the ROI border can also be used for pruning skeleton branches or for the guided construction of branches, like used by Mara et al. [136]. Yet, this introduces the requirement that important branches can be reliably characterized by such features on the ROI border.

An additional aspect regarding the simplification of the skeletons for wedge feature analysis is connected to the resolution of the polyline branches. Regardless of curvature, the polylines are represented by a dense set of points, derived from the vertex positions on the underlying mesh.

[1] www.aicon3d.com

The vertex density on the mesh may depend on the surface curvature, but a low curvature in direction of the extracted polyline does not necessarily correspond to a low vertex density. This is because a significant curvature perpendicular to the polyline direction may have induced a high vertex density. Branch polylines may therefore be represented by a significantly lower amount of points without overly compromising on the feature approximation quality. Consequently, the polyline branches are simplified using the Douglas–Peucker algorithm [58] in a post-processing stage. This divide-and-conquer algorithm generates an approximation of a polyline by successively determining important points with regard to a given error distance ϵ. For this purpose, starting with the complete polyline, the algorithm approximates an interval of the polyline by a line connecting the endpoints of the interval. The quality of this approximation is then evaluated by projecting all polyline points of that interval onto the line to determine a point of maximum distance. If the distance of this point does not exceed the predefined ϵ, the approximation quality of the interval is sufficient, since the point distance is maximal. Otherwise, the interval is split at the respective point and the algorithm continues by finding the next points of maximal distance with respect to the new set of intervals. The process is repeated until all points of the original polyline are within the error distance ϵ. This way, an approximate polyline is constructed, consisting of the points defining the interval borders. The expected complexity of the algorithm is $O(n \log n)$ [90], with a worst-case complexity of $O(n^2)$ [3]. Especially when considering the comparatively low amount of polyline segments in contrast to the amount of edges in the underlying mesh, the worst-case complexity is not expected to have a significant impact on the overall runtime of the skeletonization algorithm.

A result of polyline simplification using two different values of ϵ is shown in figure 5.16 for a curved Winkelhaken-type wedge that was frequently used by the Hittite scribe Pikku. The original polyline with 138 segments in figure 5.16a is reduced to a version with only 12 segments in figure 5.16b that still captures the unusual curved edge. In contrast, the maximally reduced version in figure 5.16c with 3 segments, does not only no longer capture the curved inner wedge anymore but also delivers a much worse approximation of the inner edges than the extracted wedge model shown in figure 5.16e. The worse approximation is a result of the line segments being defined by the end points of the original polyline instead of being a constrained least squares approximation like the lines of the wedge model. However, on straight wedges, the approximation difference is usually negligibly small. Another frequent reason for differences between the wedge skeletons and the extracted wedge models is the center point approximation for filled wedges. Since the skeleton vertices are constrained to the mesh surface, the hypothetical depth points of filled wedges are not captured as well as with the model-based approach. When viewing the wedge skeletons and the wedge model simultaneously, this can be an advantage and be used to identify filled wedges without having to view the mesh.

A combined example of pruning and simplifying a skeleton for wedge feature analysis is shown in figure 5.17 for two parameter values of the pruning threshold. Although the threshold

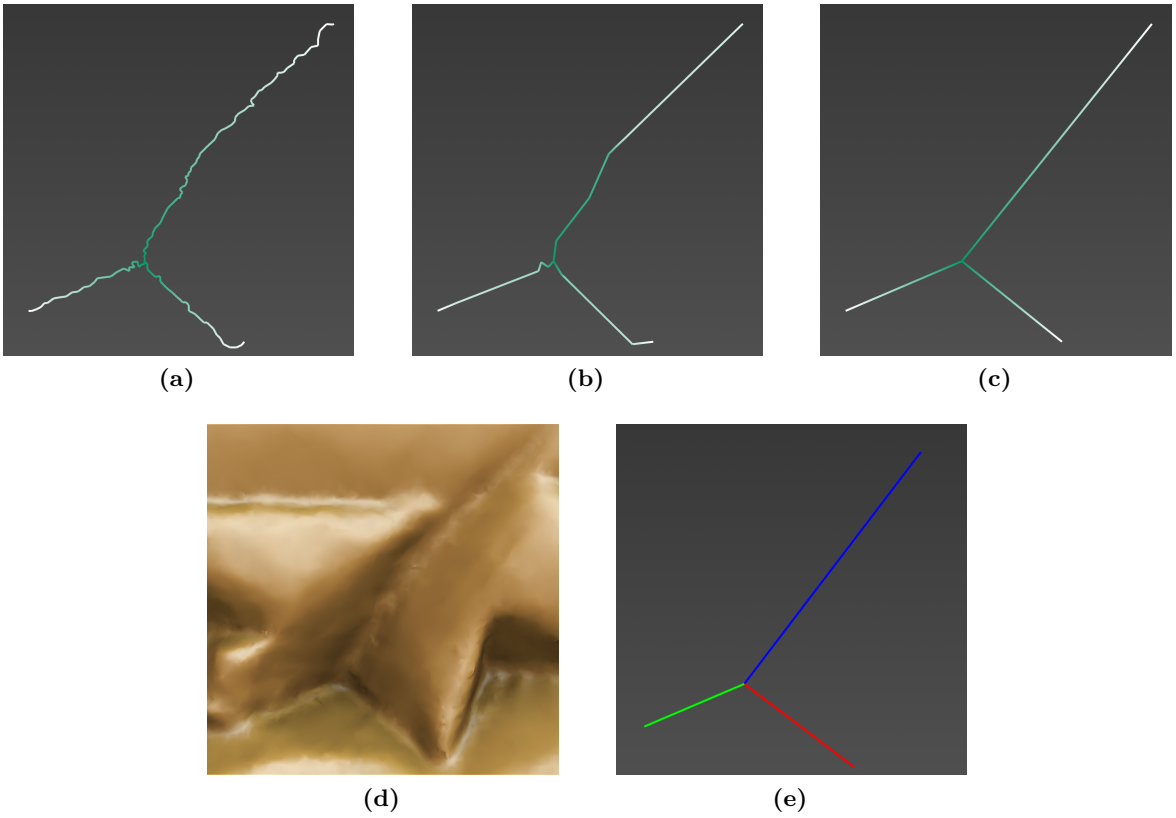
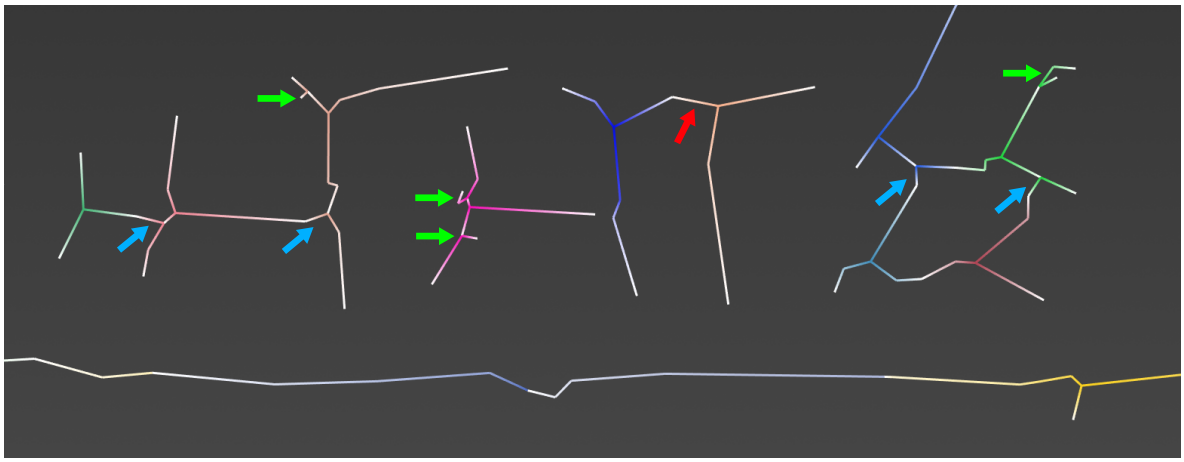


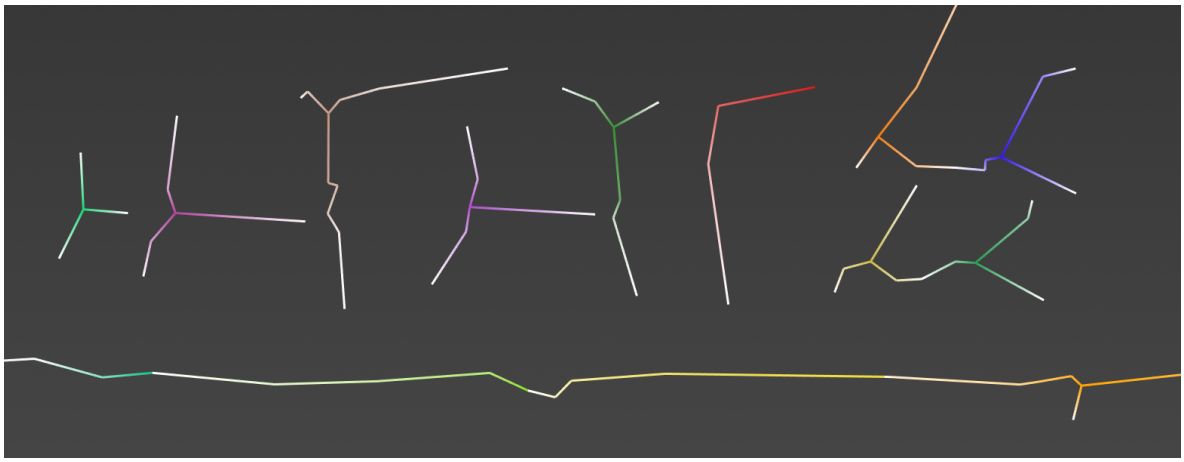
Figure 5.16: Polyline simplification with the Douglas-Peucker algorithm for a curved wedge on cuneiform fragment *531/f*. The non-simplified skeleton is shown in **(a)** and simplified versions with $\epsilon = 0.06$ mm and $\epsilon = 1.0$ mm in **(b)** and **(c)**. The shaded wedge is shown in **(d)** and the corresponding wedge model in **(e)**.

between figure 5.17a and figure 5.17b is only changed from 5 to 6, the skeleton changes significantly. Positive changes include the removal of some small micro-feature branches which are marked by green arrows and the removal of some inter-wedge connections, marked by blue arrows. These connections can be removed because the splitting of the skeleton at the wedge area borders makes them outer branches. One negative change in this example is the removal of a branch representing a whole inner edge, at the position marked by the red arrow. This example demonstrates the difficulty of finding a suitable pruning threshold even on a simple wedge constellation. On cuneiform scans with progressively changing mesh characteristics, on noisy meshes and on meshes with many similarly sized sub-wedge-scale features, finding a suitable threshold gets increasingly difficult. Additional examples for pruning results will be discussed in section 5.5.

Regardless of the intended use case, a pruning of non-simplified skeleton branches consisting only of one edge has proven to be useful for counteracting over segmentation. Assuming the underlying mesh was produced by 3D scanning with a sufficient resolution, removing branches of this size can be safely expected to not remove relevant micro-features. In addition to that,



(a)



(b)

Figure 5.17: Example of two pruned skeletons for wedge feature analysis on cuneiform fragment *7/a*. The skeleton in **(a)** has been pruned with a length threshold of 5 and the skeleton in **(b)** with a threshold of 6 and $\epsilon = 0.06$ mm for simplification. The colored arrows in **(a)** mark the changes to **(b)**.

the skeletons produced for both usage scenarios can be employed to make cuneiform wedge structures accessible to graph-CNN-based methods [114] for conducting content-based retrieval tasks on cuneiform manuscripts. These can not be used directly on the graphs given by the meshes of the wedge areas from chapter 4, because of their large size. The reason for this is the memory consumption of the back propagation algorithm used, which is proportional to the input size in form of the graph elements, multiplied by the number of layers and the number of features. Since the considered meshes can consist of several thousand triangles for one wedge, the graph size quickly becomes non-manageable for currently available sizes of GPU memory. On the other hand, although enough memory is available on current CPUs, the latter lack the processing power required to handle graph CNNs in a reasonable amount of time.

5.5 Extraction results

The evaluation of the wedge skeletonization in this section will cover two main aspects, a discussion of the achievable skeletonization quality with respect to the use cases in subsection 5.5.1 and a presentation of the performance of the approach in terms of running time which will be presented in subsection 5.5.2.

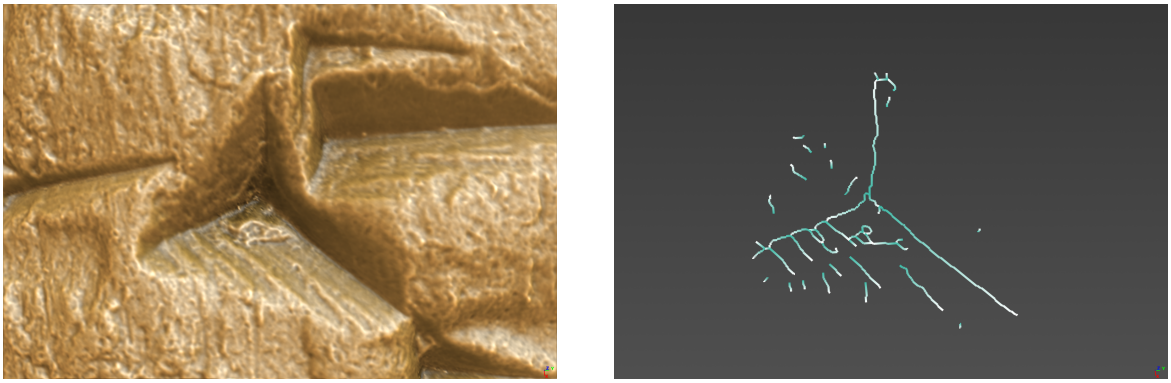
5.5.1 Skeletonization quality

As detailed in section 5.2 assessing the quality of the computed skeletons objectively by means of a quantitative quality measure is not possible. Instead, this section will discuss the quality of the skeletons in the context of the two previously identified application scenarios, wedge micro-structure analysis and wedge feature analysis. For this, exemplary extraction results will be presented and discussed with regard to possible applications and encountered issues.

An example for the intended possibility to capture reed stylus structures in wedges as micro-features is shown in figure 5.18. The reed structures that are visible in figure 5.18a as an array of parallel ridges on the lower face of the wedge are captured as a set of partially unconnected skeleton branches in 5.18b. Although the shown scan exhibits a comparatively high resolution and a good quality, the reed structures are captured only very rudimentary and include other details of the rough clay surface. Comparisons with other similar wedges showed that in most cases, where the scanning resolution is sufficient to capture reed markings, the markings are still very subtle and can in many cases only be seen when enhanced via radiance scaling. Therefore, capturing these subtle structures with the skeletonization approach is not always possible. In addition to that, as can be seen in figure 5.18a, rough structures on the tablet surface often exhibit the same scale as the reed markings and are consequently also captured by the skeleton. This makes these features hard to distinguish and in many cases unusable. An approach to counteract skeleton fragmentation in the wedge areas could be to employ a post-processing pruning filter that only keeps the n largest connected components per wedge area. However, this can lead to new issues in wedge areas with mesh errors.

In contrast, micro structures like dents in the inner edges, wedge fillings or impurities like grains of sand are captured well with the skeletonization approach. The mostly random shape of these features makes an automated processing of the resulting skeletons for segmentation purposes challenging but not impossible. One example for a simple micro-structure-based detection heuristic is scanning the skeletons within each wedge area for larger circles near their center. The occurrence of circles has been found to be a good indicator for the presence of grain like impurities and therefore damaged wedges. However, detecting no circles in the skeleton graph is no proof for the absence of these impurities, since circles may be broken by mesh errors, like shown in the example in figure 5.19.

Aside from cuneiform wedges, cuneiform fragments may contain a variety of other structures like column and paragraph separators or cuneiform seals. While some seals may be composed of extremely subtle low curvature image components that are not captured well by the



(a) **(b)**

Figure 5.18: Reed micro structures on fragment *CUNES-49-15-687* (Source: KU Leuven & Cornell University) **(a)** and the captured skeleton **(b)**.



(a) **(b)**

Figure 5.19: Skeleton of a filled wedge on cuneiform fragment *IM_10865/f*. A shaded image of the wedge is shown in **(a)** and the corresponding skeleton in **(b)**.

skeletonization approach, the structure of other more defined seals can be captured well. Figure 5.20 shows an example for a structure-rich stamped seal with inverted cuneiform wedges, circles and image components that are represented well by a skeleton. Regarding automated detection tasks, the resulting skeletons can be used to detect circular structured elements or reoccurring shapes of ancient symbols and items. For the analysis of seal elements it can be exploited that 3D scans are, in many cases, presorted upon creation with respect to the object type. This can help with restricting the employment of suitable processing methods to objects that are very likely to contain seals. One such object type are clay bullae, small clay objects with stamped seals, which were used for tamper proofing documents or transport containers.

As an in-depth example for the skeletonization of superimposed wedges, figure 5.21 shows skeletons for different instances of the Hittite cuneiform sign *nu*, like already introduced in figure 5.4. The left-hand side of the figure shows a color-coded version of the junction resolved and pruned vertex skeletons, while the corresponding simplified polygonal line skeletons are shown on the right-hand side. The shown skeletons were created by pruning branches smaller



(a) **(b)**

Figure 5.20: Skeleton for the cuneiform seal *Bo91-905*. A shaded image of the seal is shown in **(a)** and the corresponding skeleton in **(b)**.

than 7 edges and with a simplification epsilon of $\epsilon = 0.05$ mm. In contrast to the wedge segmentation from chapter 4, the inner edges of all four samples were captured correctly. The additional branches in figure 5.21f are the results of floating geometry which was erroneously produced during scan merging. The additional branches in figure 5.21h, on the other hand, originate from small structures in the wedge groves that can not be attributed to scanning errors. When comparing the topology of the graphs induced by the skeletons, even when not taking the small branches into account, it becomes apparent that despite their visually similar shape, the structure of all graphs is different. This is predominantly a result of slight position variations between the two participating wedges which cause the junctions to assume different positions or even to merge. Thus, the graphs can not be trivially matched for sign comparison, but require more sophisticated methods that can perform a fuzzy matching. Possible starting points in this regard were evaluated by Bogacz et al. including graph kernels for similarity computation of wedge constellations [29] and wedge head conflict resolution [30]. In addition to that, Kriege et al. evaluated the suitability of spline CNNs for cuneiform sign detection [117] on the wedge model data from chapter 4. This method could be adapted to the more complex skeletonization data. In view of the geometrically more complex representation and the higher degree of data variation it is, however, expected that the CNNs will require significantly more training data.

Aside from matching whole constellations of overlapping wedges, it is desirable to match the contained skeleton branches to individual wedges. This could be done by adapting the model fitting from section 4.8.2 to work on the skeleton vertices of the non-simplified skeletons. For this, the depth point could be constrained to a junction area while fitting the three edges

to the vertices of three connected branches. A match could be detected by monitoring the coverage of the branches in the wedge area by the RANSAC consensus sets.

When considering the use case of wedge feature analysis, the overall similarity of the skeletonization results and the extracted wedge models is of interest. In particular, for cases of non-successful model extraction, the skeletons are a promising alternative representation to maintain readability. Figure 5.22 shows a comparison of the wedge model extraction from chapter 4 and the wedge skeletons obtained with the method described here. Since the wedge models on the shown fragment *Bo532* have been used to determine the wedge type, the skeletons are colored according to the wedge type. It can be clearly seen that the red colored skeletons in figure 5.22a provide a readable representation of missing wedges in figure 5.22b. Aside from that, it is notable that the skeletons in the lowest text line tend to show more branches than the wedge model representation. This is due to the lowest text line containing several damaged wedges with fracture markings that are partially converted to skeleton branches.

It is worth mentioning that using the wedge skeletons as an alternative representation in cases of unsuccessful wedge segmentation works only well on cuneiform fragments with a low amount of mesh errors. However, on meshes with extensive mesh damage the skeletonization will produce chaotic results, since the approach is sensitive to noisy curvature values. In addition to that, the skeletons are forced through the possible pointy connections on non-2-manifold geometry, which causes an even more chaotic behavior. Filtering these damaged wedge areas with the mesh error detection does only work to a limited extent, because the filtering does only exclude the erroneous geometry but not all accompanying curvature artifacts. As a consequence, automatically distinguishing the affected wedge areas from usable ones without using the wedge filtering from section 4.9.2 is only rarely possible. Therefore, using the skeletons on extensively damaged meshes is not recommended. An example of a cuneiform fragment affected by this issue is shown in figure 5.23. The chaotic meshed region in the upper right of figures 5.23a and 5.23b is the result of scanning errors caused by a black inventory number writing covered in clear lacquer that was added by a museum for fragment identification. Figures 5.23c and 5.23d give an example of the possible difference in skeleton quality, caused by mesh merging errors of the scanning software. While the vertical column separator is nearly error free, the horizontal paragraph separator is affected by extensive mesh errors, leading to chaotic skeleton fragments.

5.5.2 Extraction performance

Since the wedge skeletonization is intended to act as a supplementary segmentation approach to the wedge extraction from chapter 4, the evaluation of the extraction performance will only cover aspects which have not already been computed for wedge extraction. Therefore, the wedge areas and the surface distance field are assumed to be provided. In order to have comparable measurements nevertheless, the skeletonization performance is measured on the cuneiform fragments from section 4.10.3.

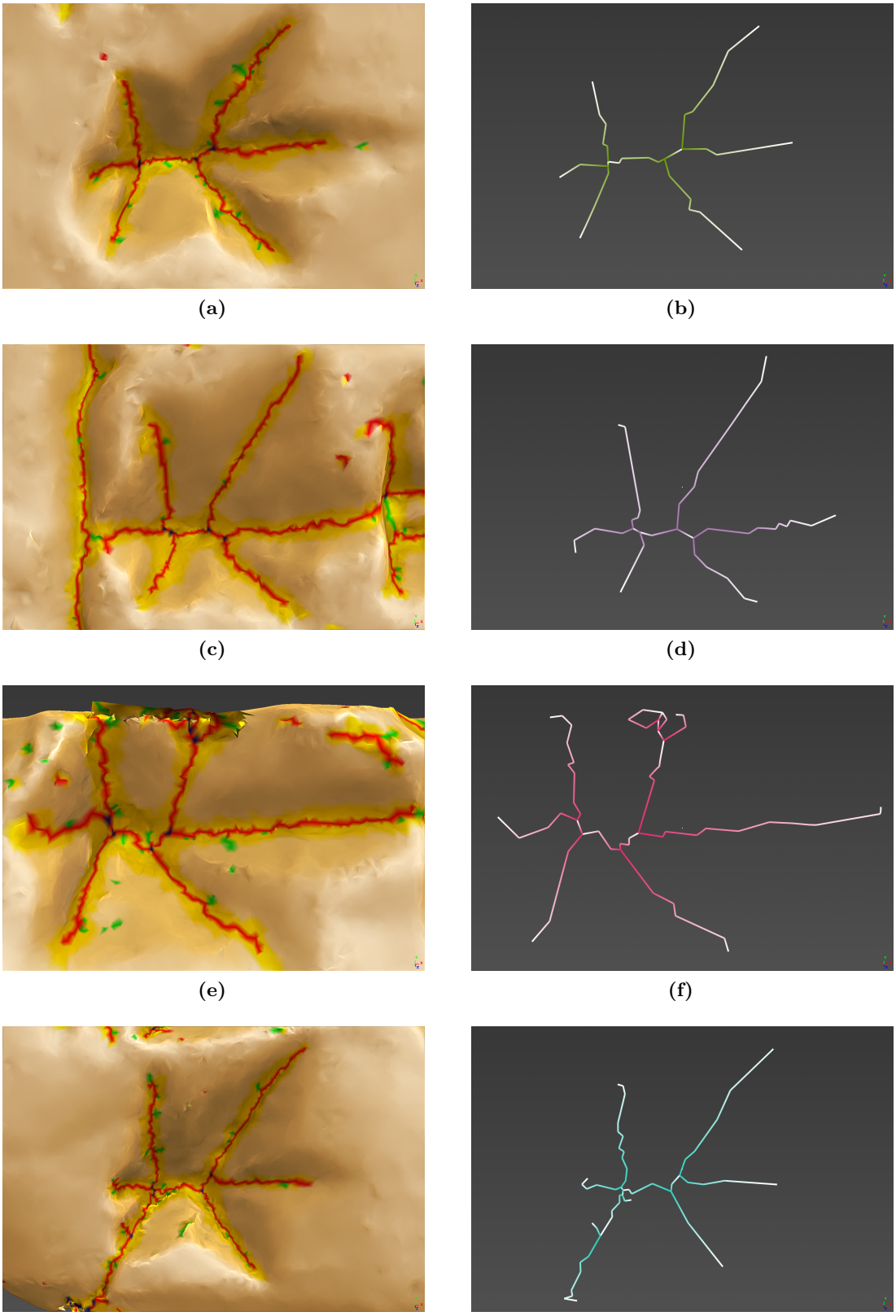
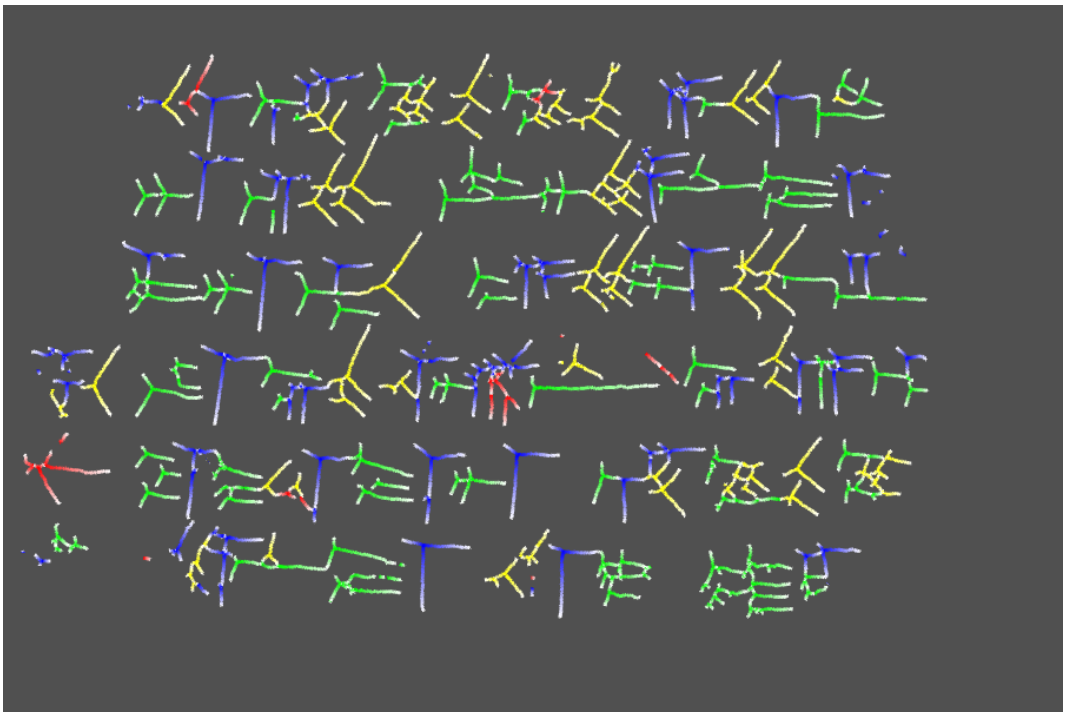
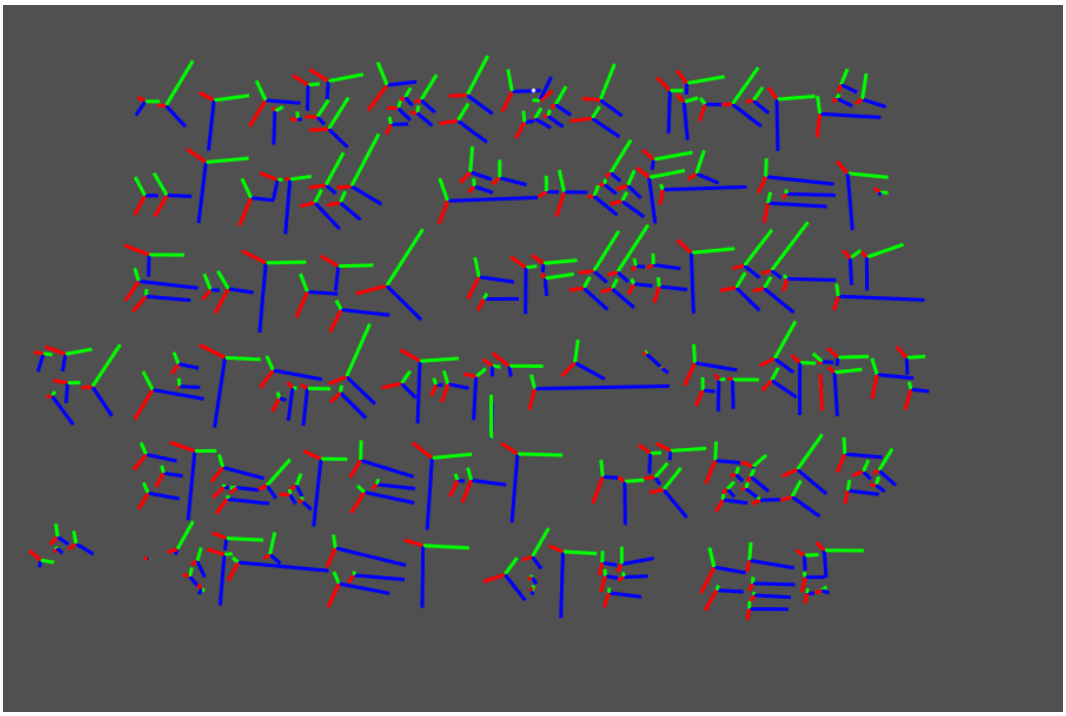


Figure 5.21: Skeletons of superimposed wedges for different occurrences of the Hittite cuneiform sign *nu*. A color-coded version is shown on the left-hand side and the simplified and pruned polyline version on the right-hand side. The green vertices on the left-hand side represent discarded skeleton branches.

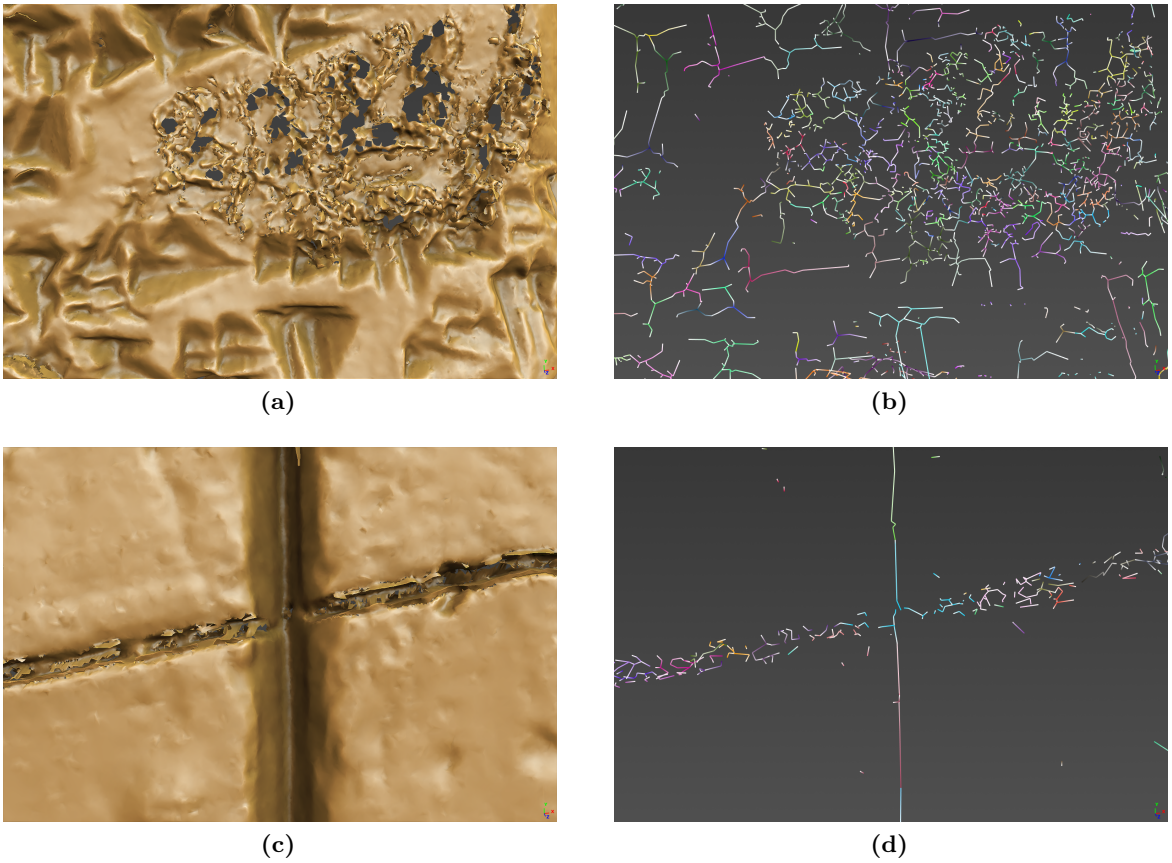


(a)



(b)

Figure 5.22: Comparison of the extracted skeletons **(a)** with the watershed-based segmentation results from chapter 4 **(b)**. The skeletons in **(b)**, are colored according to the wedge type with green for horizontal, blue for vertical, yellow for Winkelhaken-type and red for unknown wedges.



(a) **(b)**

(c) **(d)**

Figure 5.23: Examples of skeletons on a scan of cuneiform fragment *485/b* which contains sections with extensive mesh damage. The tablet surface is shown on the left-hand side and the resulting chaotic skeletons on the right-hand side.

Figure 5.24 shows the overall performance of the skeletonization approach, measured on an Intel Core-i7 4790K. The running times measured in seconds are given in relation to the amount of vertices in the initial area and show the expected linear running time behavior. The linear running time results from the fact that most skeletonization stages simply cycle the input vertices of the initial area or perform a depth-first search on a fraction of the vertices.

It is worth mentioning that the running time with respect to the total amount of vertices of the cuneiform fragments is also linear. As can be seen in figure 5.25, this results from the almost constant ratio of the two sizes. However, this data characteristic may be related to the scanning procedure which is predominantly focused on scanning the text covered regions of the cuneiform fragments. On fragments which contain a small amount of text or a large amount of fracture faces this characteristic may change.

A more detailed measurement of the running times, separating the contributions of the individual stages is shown in figure 5.26. Although the graphs for most stages show a predominantly linear running time behavior, the graph for the thinning stage stands out.

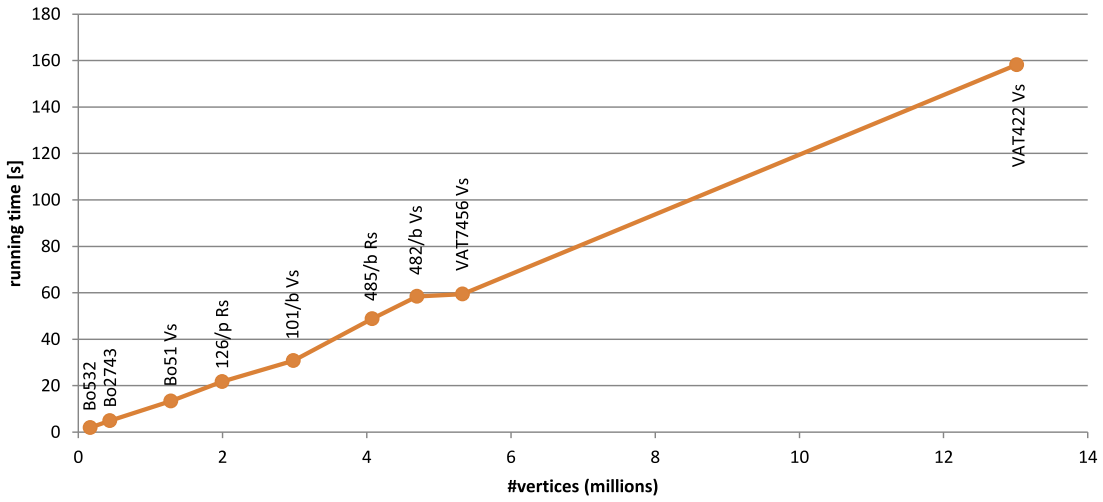


Figure 5.24: Running times of the skeletonization for the fragments from figure 4.51.

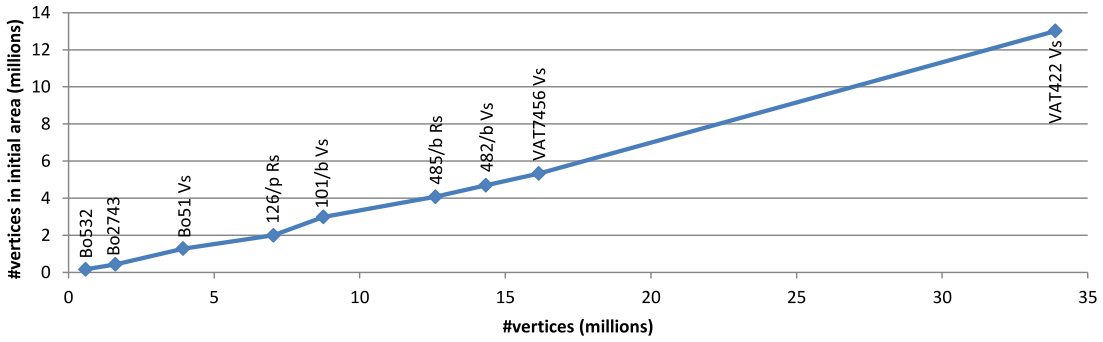


Figure 5.25: Vertex count of the initial area in relation to the fragment vertex count for the fragments from figure 4.51.

The measured irregularities of this pipeline stage result from the repeated processing of the vertices during thinning. Since some vertices can only be removed after other vertices have been removed first, the amount of iterations on the sorted vertex set does not necessarily depend linearly on the amount of vertices. Although each vertex may only be removed once until the final set of irremovable skeleton vertices is determined, a vertex and its 1-ring may be checked multiple times before removal. As a result, the processing times may unpredictably vary to a certain amount depending on the structure of the initial area. Here, the best case is described by the constellation that each vertex can be removed upon primary inspection of its 1-ring, which results in a linear running time. However, the worst case would be that in each iteration over the vertex set only one vertex is removed, which results in a quadratic running time. Considering the shape of the initial areas, the occurrence of this extreme case is highly unlikely, which is also reflected by the measured running times. The observable variations for the pruning and the partitioning and simplification stages also result from geometric

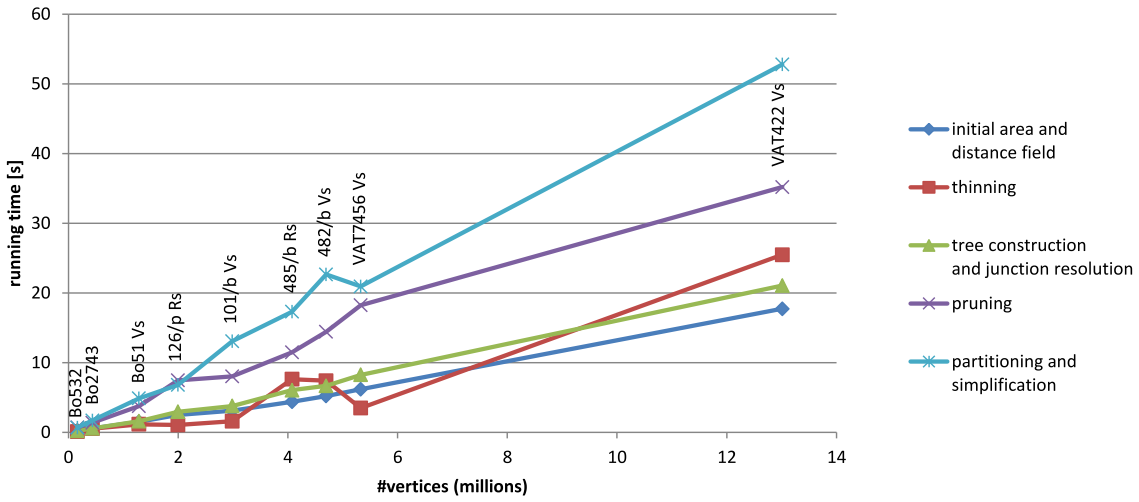


Figure 5.26: Running times of the stages of the skeletonization approach for the fragments from figure 4.51.

peculiarities of the respective meshes. In these stages, the number of wedge areas and the number of created skeleton branches can lead to small deviations from a linear running time behavior.

Regarding the achieved running times, of below 160 seconds on very large meshes of over 33 million vertices, the presented skeletonization approach can still be considered to be usable for interactive processing of cuneiform fragments. Since the measured values are only valid if the wedge areas and the surface distance have been precomputed by the wedge extraction approach from chapter 4, the respective measured running times from section 4.10.3 have to be considered. As can be seen from the extraction graph in figure 4.52, even when considering large fragments, the combined running times for creating polygonal per wedge skeletons are in the range of only a few minutes.

To show the data reduction associated with the generation of a skeletal representation, table 5.1 lists the sizes of the important vertex sets, namely the whole fragment, the initial area, the skeleton and the simplified skeleton. The non-simplified skeletons include only 4.5% to 7.0% of the vertices of the whole cuneiform fragment, the simplified skeletons only 1.3% to 2.2%. This reduction can be essential for making cuneiform structures accessible to graph-CNN-based retrieval methods, which would otherwise not be possible due to memory limits of current GPU hardware. Aside from that, the overhead for storing the simplified skeletons in addition to the wedge models is comparatively small, especially when storing the associated polylines in the form of indices to the original vertices.

Tablet	Fragment	Initial area	Skeleton	Simplified
Bo532	587,053	164,767	30,964	7,780
Bo2743	1,610,845	437,504	74,523	21,778
Bo51 Vs	3,938,454	1,285,040	216,167	62,436
126/p Rs	7,034,882	1,996,055	364,818	97,359
101/b Vs	8,749,368	2,983,627	511,935	148,696
485/b Rs	12,598,242	4,074,815	697,025	223,861
482/b Vs	14,328,507	4,696,265	844,751	250,988
VAT7456 Vs	16,150,133	5,327,426	926,684	235,587
VAT422 Vs	33,889,749	13,014,362	2,347,865	744,570

Table 5.1: Reduction of the considered vertex amount during skeletonization. The shown numbers represent vertex counts for the whole fragments, the initial areas, the resulting skeletons and the simplified skeletons.

5.6 Conclusion

This chapter presented a fast supplementary segmentation approach to the wedge models extracted in chapter 4. The approach can extract polygonal per wedge skeletons on average sized cuneiform fragments in under 60 seconds and on large fragments in only a few minutes. An optimization of the constructed skeletons for several use cases, like wedge micro-feature analysis and wedge feature analysis can be achieved by changing the parameterization of some pipeline stages. For this, the level of detail regarding the amount of branches and the spatial sampling of the branches can be changed. One of the most interesting aspects in terms of a further automated processing of the constructed skeletons is the strong reduction of the size of the representation, when compared to the input meshes of the wedge areas. The achieved reduction may make cuneiform structures accessible to graph-CNN-based retrieval methods, which would otherwise not be possible due to memory limits of current GPU hardware. Aside from retrieval tasks, the skeletons can be used for graph-based decomposition of superimposed wedges to extract individual wedges which could not be retrieved with the wedge extraction from chapter 4.

Regarding the suitability of the skeletons for further processing, the quality of the analyzed cuneiform meshes is of crucial importance, considering both mesh errors and damaged writing surfaces. On damage free and well scanned fragments, a clean readable representation can be obtained. However, on noisy or damaged fragments and, in particular, on fragments with mesh errors, the approach tends to produce chaotic and fragmented skeletons, on which distinguishing cuneiform structures from non-important details is a very challenging task. Especially noise and mesh errors should preferably be targeted by improving the quality of the scan data. Aside from that, the parameterization of the approach requires to be adapted to the outputs of the used scanning pipeline to obtain good pruning results. Once a suitable

set of parameters is obtained, it can be used on a wide range of fragments, scanned with the same scanning procedure.

In view of the largely varying scan characteristics of different cuneiform collections, this approach represents only a starting point focused on extraction speed for creating skeleton representations. Since the resulting skeletons are not yet mapped to cuneiform wedge components as used in the model-based extraction, using these representations for retrieval tasks or script feature analysis would require additional processing. For this, parts of the model fitting techniques from chapter 4 could be adapted. Specialized graph-based post-processing methods are not covered in this thesis.

CHAPTER 6

Visualization concepts for cuneiform segmentation

This chapter presents visualization concepts for efficiently working with 3D-scanned high resolution cuneiform datasets and the associated segmentation data, produced by the methods described in chapters 4 and 5. Starting with a description of general cuneiform visualization requirements in section 6.1, section 6.2 will give an overview of the framework concept including structural and user interface related design choices. Based on this, section 6.3 will present and discuss several design aspects to consider for fast rendering of large cuneiform meshes. All methods that determine the appearance of the visualized mesh and meta-data components will be described in section 6.4. As an extension to sections 6.3 and 6.4.6, section 6.5 focuses on fast and flexible methods for geometry and meta-data selection. The presentation of the methods is complemented by a performance evaluation in section 6.6, which will cover running times and memory consumption. The chapter will close with a conclusion.

6.1 Introduction

An efficient visualization for working with cuneiform 3D scans in a segmentation-centered framework has to address various requirements to be suitable for philological use and at the same time support the design process of segmentation methods. These requirements include that the visualization must be able to handle large meshes, since 3D scans of large cuneiform manuscripts or the simultaneous visualization of many cuneiform fragments, like required for virtual joining, may, in extreme cases, consist of hundreds of millions of triangles. While this may not represent a frequent case, a suitable visualization should at least be able to ensure interactive handling of datasets with tens of millions of triangles. In addition to that, the use case of creating virtual joins introduces the need to be able to handle multiple

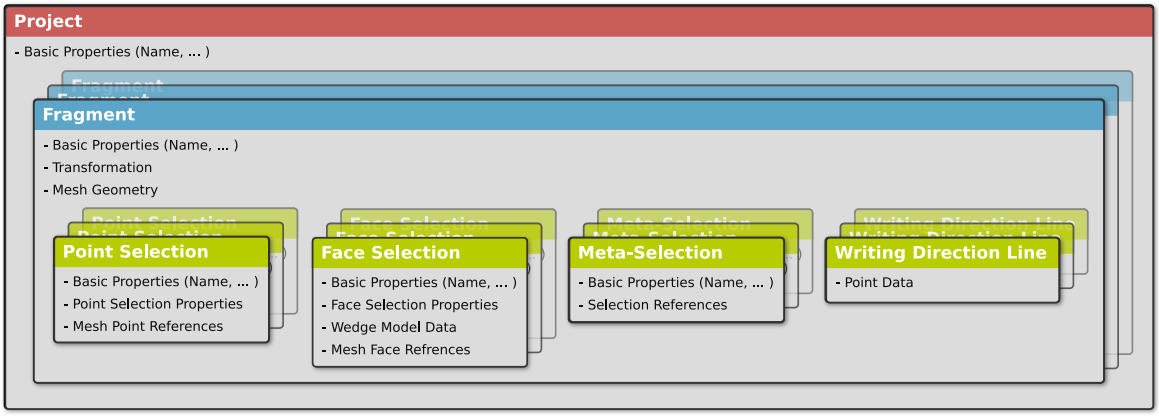


Figure 6.1: The hierarchical data concept.

cuneiform fragments at once with an intuitive visual interface for translating and rotating the fragments into alignment. To ensure a good visual representation of cuneiform features, the visualization should provide methods to optimize depth perception and rendering of subtle details that are frequently found on cuneiform scans. These methods should be based on a realistic representation to enable intuitive access by the philologists. Aside from that, the use of stylized rendering methods, similar to the hand copies mentioned in chapter 2, can offer added value.

Beyond these non-segmentation related requirements, the generated segmentation data should be visualized intuitively. This includes visualization aspects on elements of the processed meshes and the visualization of model-based data in the form of meta-drawings. In order to prevent a time-consuming recalculation of the large amounts of generated segmentation data and to ensure data portability and persistence, a viable concept for storing segmentation data for visualization and external access is required.

6.2 Framework concept

The two main use cases of the segmentation framework, namely fragment joining and cuneiform segmentation, are supported by the framework structure via hierarchically structuring the data into *fragments*, *projects*, *selections* and *meta-selections*. This concept has two components, a hierarchical data concept for handling the data in the framework and a corresponding file concept for persistent storage of the data. An overview of the hierachical data concept, which will be detailed in the following, is shown in figure 6.1.

In contrast to the cuneiform fragment definition from chapter 2, a *fragment* in the framework concept refers to a geometric object that was scanned as a whole and loaded from one geometry file, like a PLY file. Thus, a fragment may correspond to one cuneiform fragment but may also contain only a portion of a cuneiform fragment, multiple already physically joined fragments or a complete manuscript. Fragments are loaded as one triangle mesh that can be transformed in

the 3D viewport and processed for segmentation purposes. In addition to that, each fragment has its own set of visualization parameters. To address the requirement of being able to handle multiple fragments at once and to conduct virtual joins, a fragment is always part of a *project*, which in turn can contain multiple fragments. Since one fragment can potentially be part of multiple projects, some fragment specific properties are managed per project. These properties include a name, a color, a comment and the transformation of the fragment, all sharing the common trait that they must be able to assume different values depending on the project they are located in. This way, a fragment can be loaded in different projects, for example, multiple join proposals with multiple other fragments whereby all fragments are positioned differently in each project but have the same associated segmentation data in all projects.

All segmentation data in the framework is managed per fragment in the form of selections and meta-selections. A *selection* is a subset of geometry on a fragment that has been specified as a processing unit for segmentation methods. This subset of geometry can be point- or triangle-based and can be created in one of two ways. The first way is to manually select geometry in the viewport and convert it to a selection that will be a point- or triangle-selection depending on the current rendering mode. Selections can also be generated automatically by segmentation methods like the wedge extraction presented in chapter 4. There, a selection is created for each extracted wedge candidate region, which then serves as a base for the model fitting. Likewise, the model fitting can also be executed on manually created selections. The rather generic name "selection" was chosen because the content of a selection is not restricted to single wedges. Even when just considering cuneiform related features, a selection can contain any combination of wedges, writing surface, fracture faces, firing holes, seals, and column- or paragraph separators. In future work, each of this content types could be targeted by an own set of segmentation methods, which makes selections very flexible. For face-selections containing cuneiform wedges, each selection holds the extracted set of meta-data, like the wedge model or cluster annotations of the faces and a set of generic properties, like an ID, a name, a color and a comment. As will be detailed in section 6.4.6, these properties are interactively accessible through visual representations in the 3D viewport and via a list-based visualization. As an extension of the selections, *meta-selections* represent ordered groups of selections. Consequently, meta-selections can be used to manage cuneiform signs or words that are composed of multiple wedges. The capability to also manage an order for the included elements comes in handy when comparing cuneiform signs. If the respective meta-selections for a sign are created using the same wedge order, the properties of corresponding wedges can be directly compared. As a cuneiform specific feature, meta-selections can also contain placeholders for missing wedges. This allows inputting incomplete signs while maintaining the comparability with complete signs, which is a particularly useful feature for digital sign inventories. Aside from the ordered list of contained selections, meta-selections only contain a basic property set including an ID, a name, a color and a comment. These basic properties, which are also available for fragments and selections, act as an annotation layer for the elements. They also fulfill the purpose of making the respective elements uniquely recognizable, both

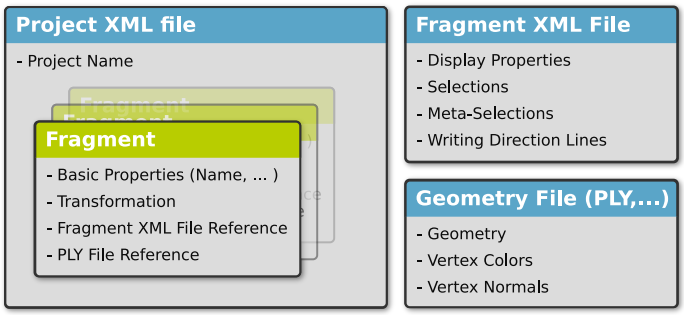


Figure 6.2: The framework file concept.

in the 3D viewport and the list-based visualizations, which will be detailed in section 6.4.6. Beyond selections and meta-selections, fragments may additionally contain writing direction polygonal lines to specify the writing directions required in section 4.8.3.

The mapping of the previously described structural framework concept to persistent data structures takes place using two custom XML-based file formats for projects and fragments and is visualized in figure 6.2. The 3D-scanned geometry of the fragments is always loaded from the original geometry files as created by the scanner software. These files are never modified by the framework to ensure data integrity. Most fragment specific data, like the writing direction polygonal lines, the selections, and meta-selections, is stored in an XML file at the same location and with the same file base name as the geometry file. Upon loading a geometry file containing a fragment, the framework detects the presence of a respectively named XML file and loads its content in addition to the geometry. Since the geometry files are not modified, they do also not include curvature and ambient occlusion values that are computed during wedge segmentation. Consequently, these mesh properties are re-computed either on opening a geometry file or when required by the segmentation methods. The per fragment meta-data can be saved to and loaded from arbitrarily named files to manage multiple data versions. However, since the meta-data format strongly relies on saving the mesh related information based on primitive IDs, the original geometry files may not be modified for the XML files to remain usable. This restriction does only apply to geometry-mapped information, like the points and triangles, contained in the selections. In contrast, the wedges models would not be affected by a reordering of the primitive IDs in the geometry files, either intentionally or due to file corruption. As a direct result of the contained index-based information, the XML files for fragments may get comparatively large. Typical XML file sizes are in the order of one third of the geometry file size in the case of binary PLY files. Although this size could be reduced by about 70% through applying common compression algorithms to the text-based information, no compression is applied to maintain the readability of the files and thus ease of access. This is especially important when considering the intended possibility of the file format to be read from other software for statistic evaluation. The second XML file format stores the project information. This includes the contained fragments in the form of XML elements

that hold the set of basic properties mentioned earlier, including the fragment transformation and references to the geometry and meta-data files of the respective fragments. Loading a project file can therefore initiate the loading process for all associated fragments and their meta-data and restore the respective relative fragment transformations applied during virtual joining. Due to the small amount of contained data, project files are usually no larger than a few kilobytes.

The user interface concept of the framework is centered around a 3D viewport for displaying the loaded geometry and the associated segmentation data. As shown in figure 6.3, the viewport is complemented by a menu bar, a toolbar, a widget for changing the display/rendering properties, and list-based widgets for working with fragments, selections, and meta-selections. Beyond including all necessary GUI elements for covering the use cases described in section 2.3, extended with the requirements of the segmentation methods from chapters 4 and 5, the main goal was to provide a highly intuitive and flexible user interface. This is achieved by supporting widely-used interaction concepts and hotkeys and providing a highly customizable user interface. The position and size of all toolbars and docking widgets, such as the fragment, selection and meta-selection lists, can be adapted by the user. Moreover, the interface supports well-known hotkeys and modifier keys for clicking actions, both in the 3D viewport and the list-based visualizations. For example, click-selecting elements in the viewport uses the same principles as the list-based selection methods. Regarding the free positioning of docking widgets, being able to increase the size of the selection list and move it to a separate screen accommodates the high information density in the list-based visualization. Additionally, it is ensured that most actions can be performed in a few different ways to accommodate the working habits of the users. For instance, actions on selections can be performed using commands in the menu bar, using a context menu in the selection list or in the viewport, or by using hotkeys. Furthermore, the real-time synchronization between the list views and the viewport, which will be detailed in section 6.5, ensures a consistent visualization and enables mixed working patterns. Aside from invoking the wedge extraction and other cuneiform segmentation methods manually in real time, segmentation tasks can also be performed using a batch mode. This mode allows processing a large number of fragments without any further user interaction once the parameterization of the methods is set. The results are then saved to the usual XML files for every fragment. Thus, evaluating the influence of segmentation parameters on statistical script properties, or simply precomputing segmentation results for hundreds of fragments, becomes much less time-consuming.

In the practical implementation, the framework is based on the Qt library [23] for the graphical user interface, on OpenMesh [33] for the mesh data structures, and on OpenGL [200] and the OpenGL Shading Language (GLSL) [179] as the 3D interface. Thus, a platform independent operation is possible. The actual implementation prototype of the framework is called “CuneiformAnalyser”.

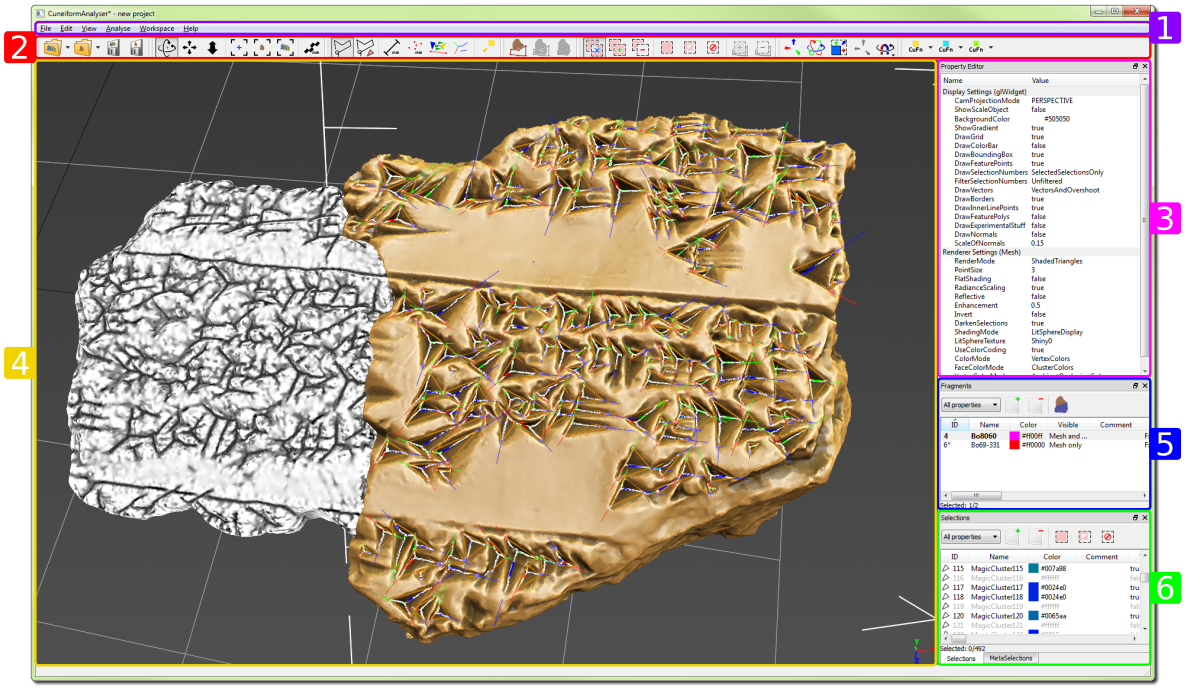


Figure 6.3: The UI concept of the CuneiformAnalyser with highlighted main components including the menu bar (1), the toolbar (2), the display/rendering properties (3), the 3D viewport (4), the fragment list (5) and the selection and meta-selection lists (6).

6.3 Mesh rendering

Aside from using indexed rendering to reduce the memory footprint on the GPU and to maximize rendering speed, a few optimizations for rendering large meshes in OpenGL can be applied to improve the handling of large meshes.

Given the large data sizes of 3D cuneiform meshes that need to be loaded into main memory for geometry processing and into GPU memory for rendering, it becomes apparent that less capable hardware might not be able to handle this amount of data for large fragments. For a cuneiform fragment scan like the obverse of fragment *VAT422* with 33,889,749 vertices and 67,183,948 faces the indexed data structure alone with only vertex positions and indices would use 1.212 GB graphics memory. Adding vertex normals and vertex colors would increase this amount to 1.755 GB, which would already be hard to handle on a GPU with 2 GB of graphics memory. In addition to that, sections 6.4 and 6.5 will detail that additional space is required for deferred shading and fast viewport selection techniques. Beyond the lack of memory, GPU processing power also needs to be considered. Without any further optimizations, the above mentioned cuneiform fragment *VAT422* renders at about 10 frames per second on an Nvidia Geforce GTX 780. Rendering this mesh on a less capable GPU with only 2 GB of memory, like a Nvidia Geforce GTX 560, results in unusable framerates below one frame per second and significantly less if the GPU is forced to swap parts of the mesh to the main memory.

In cases like this, the use of out-of-core rendering techniques can be a solution. Out-of-core rendering describes a rendering approach that loads only a fraction of the whole geometry from main memory into GPU memory. This technique is based on the assumption that not all geometry contributes equally to the resulting rendered image. Geometry may be located outside the clipping volume of the viewport, be occluded or exhibit a very small rasterized size. If a method can determine the importance of all rendered primitives for the final image before rendering, then only the most relevant data needs to be transferred to GPU memory. This allows outputting an image that is a good approximation of the image that would have been rendered using the whole geometry. This advantage, in both reduced GPU memory usage and less required GPU processing power often comes at the expense of a time-consuming precalculation step in order to create a search structure for the visibility computation. An example of such an out-of-core rendering method is the iWalk method presented by Corrêa et al. [51]. In 2002, it could render a 13 million polygon model at 9 frames per second and with 90% approximation accuracy on a 900MHz AMD Athlon CPU with a Nvidia Geforce2. This was achieved by using multi-threaded visibility computations on a precomputed data structure which took about 17 minutes to create. The used models were architectural models with a large amount of exploitable self occlusion. A more modern example for an out-of-core rendering method is the NEXUS format presented by Ponchio et al. [158, 159]. It features a hierarchical mesh reduction with fast decoding and view dependent rendering, which enables a progressive visualization of very large models and can handle non-2-manifold meshes. The NEXUS format was used in the web-based 3DHOP cultural heritage viewer [161] and in the Cuneiform WebGL Viewer [70] developed by the author. In contrast to iWalk, NEXUS decimates large meshes and produces a hierarchical data format that contains geometry patches in multiple resolutions. Appropriate patches for rendering are determined online, based on the current view of the model and a user-defined visualization accuracy and then requested from the server. In addition to that, loss-less and lossy data compression can be applied to the vertex and face data to reduce streaming sizes and ensure a very fast decompression on the client side.

However, regarding cuneiform segmentation, there are several contradictions against using out-of-core techniques. For methods that work with mesh decimation, like the NEXUS format, an accurate representation of per-vertex and per-face attributes representing segmentation results is only possible on patches of the highest resolution. This is especially true for attributes that must not be interpolated like primitive IDs or cluster assignments of points and faces but also negatively affects discrete curvatures. Moreover, GPU-based fast selection methods for primitives on the original full resolution mesh, which will be described in section 6.5, are only available when rendering non-decimated full resolution patches. Yet, forcing patches to be rendered at full resolution would diminish the advantages of out-of-core rendering. This is all the more true when considering that the rendering of occluded primitives and primitives with no rasterized fragments may be necessary for accurate viewport selection results. The mesh approximation also affects the accuracy of the picking-based measuring tools which

will be revisited in section 6.4.4. This is caused either by the positional inaccuracies or by non-matching primitive IDs. Another important factor are the required preprocessing times and the compatibility of the resulting data format. The NEXUS format focuses on fast decompression but processing of very large meshes for conversion into the NEXUS format can take several hours even on an Intel Core-i7 4790K. Therefore, large models can not be converted on-the-fly upon opening a PLY or OBJ file but must be preprocessed before actual use. The NEXUS format is also no widely-used format for mesh storage and is, since it contains redundant and possibly lossy compressed data, an unsuitable format for persistent data archival. Considering these disadvantages, the mesh rendering in this framework uses no out-of-core rendering techniques but instead loads the complete mesh data into GPU memory. This is done with a focus on compact data storage of the required components and the suitability for fast data updates of single attributes. As will be shown in section 6.6, this leads to acceptable rendering speeds and memory consumption on current consumer grade hardware.

On a side note, the principle of out-of-core data processing can also be transferred to the geometric processing of the meshes. There, an out-of-core algorithm is simply an algorithm that does not need to load the whole geometry into the main memory but can work on manageable portions, loaded from slower memory, such as a hard drive or streamed over a network. Those algorithms are often also called external memory algorithms. For processing geometry, not being able to access the whole geometry at once can be an even bigger disadvantage than for rendering. In many cases, geometric processing algorithms must be specially designed to support online processing, which denotes the capability to process the data piece-by-piece serially without being able to access the whole geometry. Such so-called “online algorithms” [104] are particularly problematic in cases that include global quality criteria or a high degree of data interdependence, like, for example, ambient occlusion calculations. Therefore, out-of-core data processing is not employed for the segmentation algorithms in this thesis.

Another option for improving the rendering speed are vertex cache optimizations. These kinds of techniques exploit the vertex cache, a rendering pipeline optimization supplied by modern graphics hardware. Prior to being rasterized as points or triangles, all vertices issued for drawing are transformed with the so-called model-view-projection matrix that transforms the vertices from world coordinates to clip space and projects them further into 2D screen space. Transforming vertices is computationally expensive, which is why transformed vertices are cached for being reused upon triangle generation. Therefore, rendering consecutive triangles that share vertices becomes more efficient and increases the rendering speed. In 1999, Hoppe presented a method to optimize the mesh locality with respect to a better vertex cache utilization [93]. A more advanced linear speed version was presented 2006 by Forsyth [73], which works on indexed triangle lists, like used in this thesis. This technique requires the vertex cache size of the targeted hardware as the only parameter. Even more sophisticated versions with additional overdraw optimization and vertex prefetch cache optimization were

presented by Nehab et al. [151] and Sanders et al. [189] with the Tipsy method. Overdraw denotes the cost of a pixel in terms of how many rasterized fragments need to be drawn to obtain the final pixel color. Here, a fragment refers to a rasterized portion of a primitive that has been mapped to one pixel. Rendering close-to-camera fragments first, can avoid fragment operations for the now early occluded remaining fragments behind. Optimizing for the vertex prefetch cache can be achieved by reordering the vertices according to their occurrence in the triangle index list. Since vertices for drawing are usually fetched in cache lines, this speeds up rendering. This combined method is available through the AMD Tootle framework[1]. In general, vertex cache optimizations can be costly operations. Specifically for cuneiform scans with existing segmentation information, the triangle and vertex reordering requires a costly update of mesh and file data structures for the meta-data to remain valid. Under the condition that the original geometry files must remain unmodified, only fast optimization techniques can be used, because the optimization must then always be performed after loading a fragment. This can be a memory consuming operation, because the half-edge data structure must be regenerated each time. Moreover, the triangle IDs in newly created selections do no longer match the ones in the geometry file but the optimized versions instead, which negatively affects data consistency. In contrast, the optimization could only be performed on the GPU data structures. This would, however, require to store a mapping between the CPU and GPU indices and increase the costs for data synchronization in both directions. Therefore, vertex cache optimizations are only offered as an optional function in the framework. Nonetheless, a cuneiform specific evaluation of the effectiveness of vertex cache optimizations will be given in section 6.6.

In OpenGL, managing custom attributes for faces requires special attention, since primitive attributes are only supported for vertices. Thus, a different mechanism has to be employed to be able to use per-face attributes. A very straight-forward technique includes storing the per-face attribute data in textures and accessing this information by associating each face with a unique texture location. For indexed rendering in OpenGL, this can be achieved by using the so-called “primitive ID”. The primitive ID is available as an input variable in all GLSL shaders that are executed after primitive generation, namely fragment shaders, tesselation control, and tesselation evaluation shaders and can be accessed through `gl_PrimitiveID` [108]. Geometry shaders, on the other hand, include an identically named output variable that must be filled upon primitive generation. Since `gl_PrimitiveID` contains a zero-based primitive index that is restarted on every draw call, it assumes values between zero and the number of points or triangles in the rendered cuneiform mesh depending on the rendering mode. Assuming a square sized texture, the corresponding texture coordinates (t_x, t_y) with the matching attribute information for a primitive can be obtained using

[1] https://github.com/GPUOpen-Tools/amd-tootle

$$\begin{aligned} t_x &= \text{int}(\text{mod}(\text{primitiveID}, \text{texSize})) \\ t_y &= \text{int}(\text{primitiveID}/\text{texSize}), \end{aligned} \tag{6.1}$$

where texSize is the edge length of the texture. Before writing t_x and t_y to `gl_Position.xy`, the coordinates must be mapped from $[0,1]$ to the OpenGL clip coordinate interval of $[-1,1]$ and corrected by an offset of half the pixel width to precisely match the output pixels. However, the code in equation 6.1 may fail to reconstruct the correct pixel positions if the rendered meshes contains more than $2^{23} = 8.388.608$ primitives, which is frequently the case for cuneiform meshes. The reason for this is that many graphics cards use floating-point calculations for division and modulo operations. In particular, the modulo operation in GLSL 3.2 is specified only for single precision IEEE 754 [98] floats and a 32 bit floating-point number in GLSL uses only 23 bits for the mantissa. Therefore, the available precision for division and modulo operations on integers is smaller than expected for 32 bit fixed-point calculations. As a countermeasure, the ID mapping can be separated for IDs bigger than 2^{23} to compute the fraction of the value above 2^{23} separately. On newer GLSL versions, the computation can be forced to operate on integers, or the floating-point accuracy can be manually increased to 64 bit. However, this comes at the cost of a slower running time.

6.4 Visualization techniques

This section extends the author's publication in MDPI Informatics [70] that describes how a subset of the visualization techniques used in this thesis was integrated in a web-based cuneiform viewer for the Hethitologie-Portal Mainz [4]. A part of the utilized techniques was already briefly described in the author's publication at SCCH 2013 [71].

6.4.1 Ambient occlusion

Ambient occlusion, as used for the approximation of the writing surface and already described in section 4.5.1, is a very useful method for real time rendering of archaeological artifacts to enhance the depth perception of complex geometries. Due to the usually high vertex density on 3D-scanned cuneiform meshes, a vertex-based ambient occlusion can be directly used for visualization without introducing visually unpleasant interpolation artifacts. Therefore, the precomputed object-space ambient occlusion from section 4.5.1 is recycled for visualization purposes, instead of applying one of the screen-space techniques mentioned earlier in chapter 4. Since ambient occlusion describes a diffuse illumination component, it can simply be post-multiplied with the surface color, computed by the remaining diffuse shading functions. In contrast, specular, reflective, and refractive shading components should not be attenuated by an ambient occlusion term to maintain a natural appearance. However, negative effects on these shading components are not very pronounced if the resulting shading function is smooth. Ambient occlusion may therefore harmonize well with lit-sphere rendering in cases that do not include sharp reflections, refractions, or highlights like the ones used in the following sections.



(a) **(b)**

Figure 6.4: Improved depth perception of ambient occlusion only shading **(a)** as opposed to Lambert shading **(b)** on a section of cuneiform fragment *485/b*.

An example for ambient occlusion shading is shown in figure 6.4, where the much better depth perception for the fracture faces of the depicted cuneiform fragment is clearly visible when compared to Lambert shading.

6.4.2 Lit-sphere rendering

Regarding the geometrical inspection of cuneiform fragments, an important goal is to improve the spatial perceptibility of small geometric surface details. This can be an issue on real cuneiform fragments and is connected to the small scale of the fragments and the visual appearance of cuneiform script itself. As already detailed in chapter 2, a small light source is usually used to visually enhance the geometric structures by creating contrast-rich shadows in the cuneiform wedges. While this works well in cases of uniform surface textures, feature rich surface textures of cuneiform fragments caused by erosion, or in the form of burn marks are, an issue. This results from the fact that the color structures visually compete with the perceptible spatial structures of the surface. A further aggravating factor is the fact that a uniformly suitable shadow casting can be achieved only on flat writing surfaces. Especially in the case of photographic documentation, this enforces the requirement to take many overlapping photos to obtain a good representation of all parts of a manuscript. However, through 3D scanning of cuneiform artifacts, the surface texture can be separated from the surface geometry. This enables the use of a uniform but at the same time realistic surface shading technique to improve the visual perceptibility of the cuneiform structures. In order to simultaneously offer philologists familiar working conditions, it is desirable to use a technique that can mimic several common variations of real uniform fragment materials.

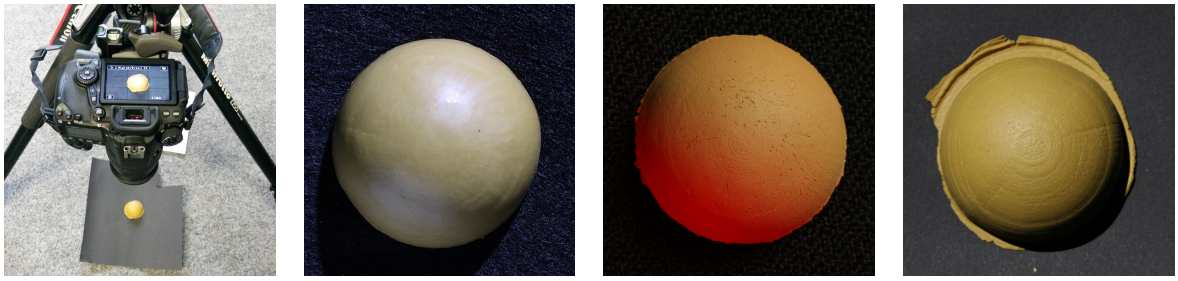
Semi-realistic shading of the cuneiform fragments can be achieved by using precomputed lighting in the form of lit-spheres, as proposed by Sloan et al. [204], sometimes also called MatCaps (MaterialCaptures). In this lighting method, a complex material is applied to a sphere, including the placement and parameterization of an arbitrary number of lights and

a reflective environment. The lighting is then stored by rendering an orthographic image of the shaded sphere, which afterwards acts as a normal-based lookup texture for shading the final geometry. A lookup is realized by centering and projecting the surface normal vector of the final geometry or the reflected eye space light vector (with respect to the surface normal) on the sphere texture. While the former usually yields more realistic shading for diffuse materials, the latter is more suitable for reflective or metal surfaces. An advantage of the method is its low impact on rendering performance and its capability to mimic a wide range of surface materials. This includes non-photo-realistic materials, which have been used by Sloan et al. [204] for stylized rendering. When using lit-sphere textures that include all lighting components, the visual appearance of the resulting surfaces in many cases retains a slightly stylized look resulting from the lack of local surface variations in object space.

Aside from pre-rendering, lit-sphere textures can also be generated by taking photographs of lit-spheres in real world material and lighting conditions. To approximate an orthographic projection, the images should be taken using a long-focus lens. The resulting images can then be cropped to the extends of the contained sphere. A disadvantage of lit-sphere lighting is its inability to directly reproduce materials with sharp and detailed color features that are not part of reflections. The same applies to macroscopically rough objects, since the resulting shading often produces unrealistically moving glossy spots. This unwanted effect of photographic lit-sphere textures with many uneven features can be counteracted by employing appropriate image smoothing methods. As an alternative to smoothing, Todo et al. [216] proposed a method to compute lit-sphere rendering in light space. This extended lit-sphere approach can handle small surface variations and non-view-dependent lighting. However, as the results tend to produce an even more stylized look, the extended method is not used here.

To accommodate the needs of cuneiform script analysis, several types of lit-sphere textures were generated to mimic clay and metal materials in various color and shininess variations. The creation of textures was realized by taking photographs of real world clay spheres, like shown in figure 6.5, including necessary post-processing, and by rendering spheres with complex shaders using a ray tracer with global illumination lighting. The design of the lit-spheres not only considers the material properties but also the lighting conditions to match the ones that are frequently used in traditional photographic cuneiform reproductions and to produce detail enhancing highlights. For this, photographs of the Hethitologie-Portal Mainz [4] were examined to identify common lighting setups. In order to show some possible variations of lit-spheres, figure 6.5b includes a sample for a sharp highlight, the sample in figure 6.5c is illuminated by multiple colored lights and figure 6.5d shows the setup of a light source from a top left direction on a matte clay surface, which can be commonly found in cuneiform photographs.

Optionally, for lit-sphere textures mimicking clay, an inexpensive color shift and saturation modification can be applied during shading to change the basic color of the clay to suit the user's preference. Two examples for the improved shading quality are shown in figures 6.6b



(a) (b) (c) (d)

Figure 6.5: Camera setup **(a)** for photographic creation of lit-sphere textures with a clay half-sphere on a dark background and exemplary resulting, unprocessed images for different clay types, surface finishes and lighting setups **(b-d)**.

and 6.6c as opposed to simple Lambert shading in figure 6.6a. The corresponding lit-sphere textures are shown in figures 6.6d-f. Since Lambert shading can also be realized via lit-sphere shading, the corresponding radially symmetric lit-sphere texture is shown in figure 6.6d. In contrast, figure 6.6e shows an example for a processed photographically generated texture with a pronounced shadow and a diffuse highlight and figure 6.6f shows a synthetically rendered texture with a less pronounced shadow and rim lighting. The extrapolated corners in the photographic texture are a side-effect of the required smoothing to remove surface imperfections of the used clay spheres. Applying the necessary smoothing without extrapolating the corners would have caused the sphere borders to smear up with the black background and cause unpleasant rendering artifacts. Alternatively, the length of the normal vectors for look-up can be reduced by a small amount before projecting them on the texture. However, this will usually result in a lower visual contrast and a more dull rendering of silhouettes.

It has to be noted that the simple lookup-based lighting comes at the expense of a less interactive illumination. Dynamically repositioning the light-sources, like frequently carried out by philologists during real world collation of an artifact, is only possible by rotating the lookup vector for the lit-sphere texture around the viewport z-axis. However, this can still create an effect comparable to a light source rotating on a view-aligned plane around the object.

All in all, the lit-sphere shading can be particularly useful when combined with the other visualization techniques from section 6.4. Its realistic to semi-realistic surface shading compensates for the omission of the non-advantageous real surface textures of the scanned artifacts and has proven to be very useful for the visualization of cuneiform fragments.

6.4.3 Radiance scaling

While ambient occlusion can enhance the depth perception, it can not significantly improve the visual perceptibility of small shallow features, like the geometrical details on cuneiform manuscripts. To visually enhance these details on the scanned surfaces, the radiance scaling

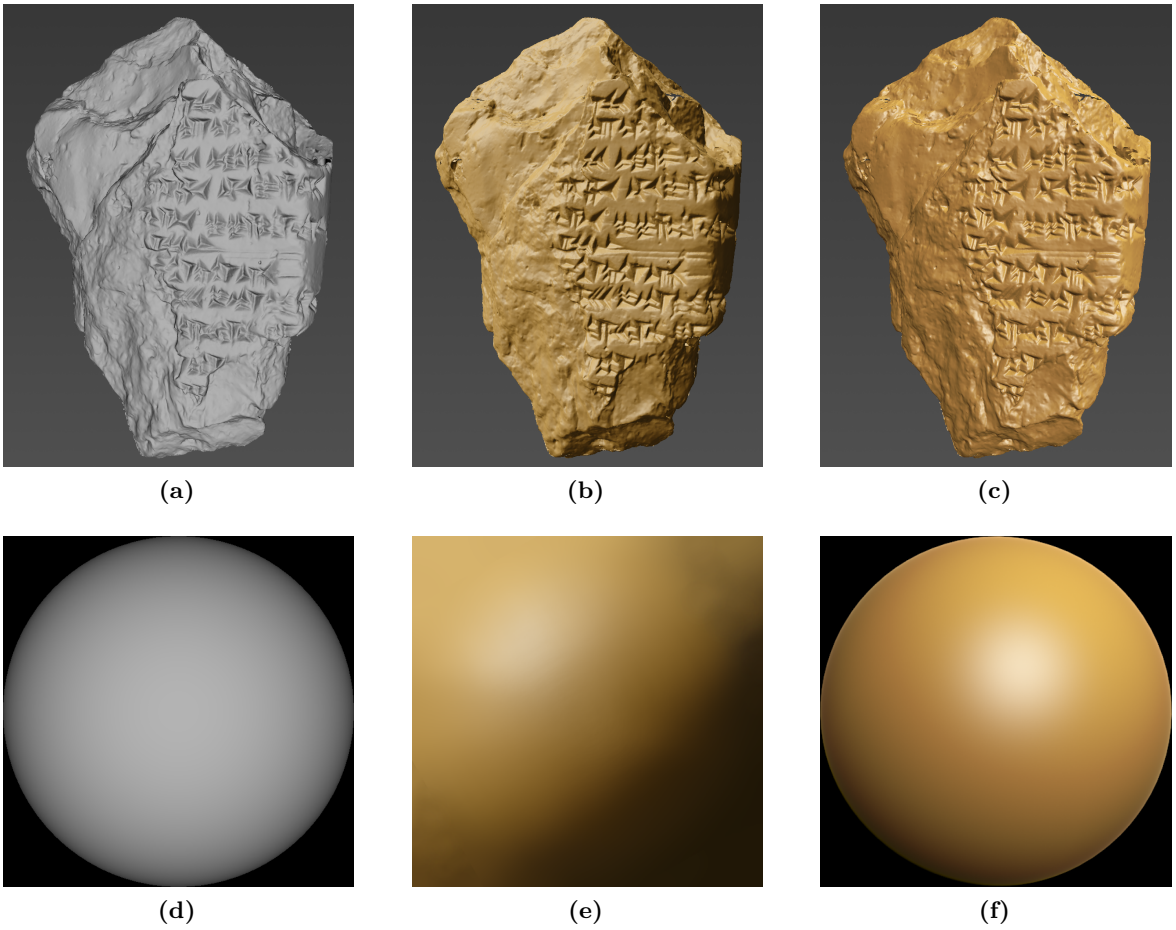


Figure 6.6: Rendering improvement over Lambert-shading **(a)** with lit-sphere shading **(b)** and **(c)** on cuneiform fragment *E971*. The corresponding processed lit-sphere textures are shown in **(d)**-**(f)**.

technique [224] of Vergne et al. is employed that correlates surface variations to aspects of the shading function. As the name suggests, radiance scaling adjusts the reflected light intensity depending on the surface curvature and material properties. Applying this concept to the radiance equation yields the enhanced radiance L' at a surface point $\mathbf{p}$ with surface normal $\mathbf{n}$, an eye direction $\mathbf{e}$ integrated over a hemisphere Ω around $\mathbf{p}$ with incoming radiance L as

$$L'(\mathbf{p} \rightarrow \mathbf{e}) = \int_{\Omega} \rho(\mathbf{e},\mathbf{l})(\mathbf{n}\cdot\mathbf{l})\sigma(\mathbf{p},\mathbf{e},\mathbf{l})L(\mathbf{p} \leftarrow \mathbf{l})dl. \tag{6.2}$$

Here, dl denotes the infinitesimal solid angle of the integration variable $\mathbf{l}$. The scaling function σ is applied independently to the original BRDF (Bidirectional Reflectance Distribution Function) ρ. Complying to the goal of enhancing the perceptibility of surface concavities and convexities, the scaling function is designed to be monotonic and exhibit an invariant point

to not influence planar surface regions. A possible function with parameters controlling the location of the scaling-invariant point $\alpha \in]0,1[$ and the scaling magnitude $\gamma \in [0,\infty[$ is

$$\sigma_{\alpha,\gamma}(\kappa,\delta) = \frac{\alpha\gamma e^{\kappa} - \delta(1-\alpha(1+\gamma e^{\kappa}))}{\alpha + \delta(\gamma e^{\kappa} - \alpha(1+\gamma e^{\kappa}))}. \tag{6.3}$$

The scaling function input parameters are a normalized reflectance δ and a normalized curvature κ. Applied to calculating the lighting in an OpenGL- and GLSL-based environment, δ can be the length of a color vector, computed by an independent shading function, like Phong shading, lit-sphere shading, or even a vertex-based ambient occlusion. κ on the other hand, must be a curvature derived term. Since the feature scale and the vertex density on cuneiform datasets are not known a priori, a screen-space-derived curvature rather than a discrete differential vertex-curvature [145] is employed to optimally exploit the radiance scaling technique. This enables the user to select the scale of features to be enhanced since the size of features in screen space can be varied via zooming the viewport navigation. The curvature term is calculated like in [222] as a hyperbolic tangent mapped mean curvature. In this process, a relative depth gradient is computed as

$$\mathbf{g}(\mathbf{p}) = \begin{pmatrix} -n_x/n_z^f \\ -n_y/n_z^f \end{pmatrix}, \tag{6.4}$$

for all points $\mathbf{p}$ with normals $\mathbf{n} = (n_x, n_y, n_z)$ and a parameter $f \in [0,\infty[$ that controls a curvature falloff depending on the angle between the normals and the view-direction. To obtain the required mapped mean-curvature $\kappa(\mathbf{p})$, the Hessian $H(\mathbf{p})$ of the gradient field is computed from the first order derivatives $\mathbf{g}_{\partial x}$ and $\mathbf{g}_{\partial y}$ of $\mathbf{g}$ in x and y directions as

$$H(\mathbf{p}) = \nabla^T \mathbf{g}(\mathbf{p}) = (\mathbf{g}_{\partial x}, \mathbf{g}_{\partial y}). \tag{6.5}$$

The principal curvatures κ_1 and κ_2 are contained in the trace of H. Thus, the mapped mean curvature $\kappa(\mathbf{p})$ can be obtained with

$$\kappa(\mathbf{p}) = \tanh(\kappa_1 + \kappa_2). \tag{6.6}$$

Since the computation of radiance scaling does not account for object silhouettes, a silhouette detection similar to [223] can be used to counteract excessive scaling at silhouette borders that are characterized by discontinuities in the depth values. This can be realized as a 2D image filter, operating on a 3×3 filter window, which computes an average of thresholded local differences to the center pixel. The smoothness of silhouette detection can be improved by using smoothed thresholding. On cuneiform fragments, the effects of this silhouette correction are usually very subtle because of the lack of strongly inclined silhouettes.



(a) **(b)**

Figure 6.7: Section of cuneiform fragment *E971* without **(a)** and with radiance scaling **(b)**.

In this work, the scaling function is parameterized with a scaling invariant point $\alpha = 0.1$ and a user controllable scaling magnitude $\gamma \in [0,2]$. The scaling is applied only to the color term of the shading function. Additional components, like the ambient occlusion, are applied independently as they have been found to exert a negative influence on the rendering of geometric details within cuneiform wedges.

Both the normal-based gradient and curvature calculations and the depth-based silhouette detection require accessing a local neighborhood of a pixel to be shaded. Consequently, a deferred shading architecture is used, with the first rendering pass storing colors, normals, depth values and the depth gradient values from equation 6.4. Assuming a monochrome colored 3D object and including the precomputed vertex-based ambient occlusion values, the required values can be rendered into two RGBA floating-point textures. In view of the possibly large geometries, this is done in one rendering pass using multiple simultaneous draw buffers, instead of a separate rendering pass for each texture to avoid costly duplicate vertex transformations. The second so-called deferred shading pass can then compute the complete shading, using only the textures from the first rendering pass. This covers lit-sphere shading with ambient occlusion and radiance scaling, including the curvature and silhouette computations, operating on a filter window.

Figure 6.7 shows a representative example of the local detail enhancement for cuneiform wedges that can be achieved using radiance scaling. For the used parameters, convex surface features are brightened and concave features are darkened, increasing the overall contrast of the features. As a result, the detail enhancing effect of the method works particularly well on abraded and corroded surface areas with subtle details.

The improved visual quality is also visible when viewing whole cuneiform fragments as shown in figure 6.8. Despite the larger size of the radiance scaling filter window, when compared to the depicted wedges, the relevant details are enhanced appropriately. However, when taking high-resolution screen-shots for printing, it must be taken into account that the effects

of radiance scaling will get less pronounced when the images are reduced in size. As a countermeasure, the radiance scaling magnitude γ can be increased. When comparing the different shading components in figure 6.8, it becomes apparent that the radiance scaling predominantly results in an edge enhancement while the ambient occlusion improves the depth perception. The combined shading with lit-sphere rendering, ambient occlusion and radiance scaling is shown in figure 6.8d. Since this type of shading usually provides the best detail enhancement and the most attractive visualization, it is used as the default visualization style for mesh rendering. In particular, the radiance scaling technique has proven to be useful for enhancing details that are otherwise very difficult to perceive on real cuneiform artifacts or photographic cuneiform reproductions.

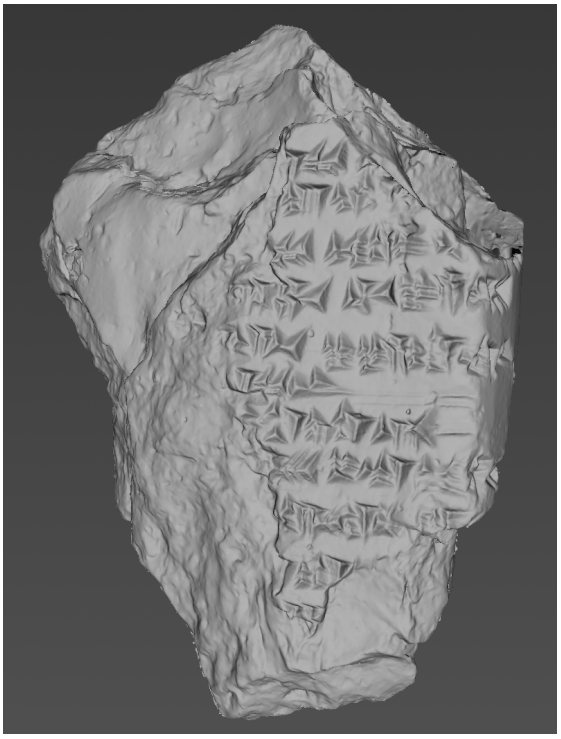
6.4.4 Logarithmic depth buffer

Philological work on 3D-scanned cuneiform manuscripts not only benefits from visually appealing rendering techniques but also requires accurate rendering methods in terms of depth rendering precision. One reason for this is that the 3D-scanned meshes may contain a non-negligible amount of closely overlapping mesh patches which are susceptible to depth precision issues that might cause z-fighting during rendering. Aside from the mesh itself, the Poisson approximation of the original free writing surface from section 4.5.2 is extremely likely to cause z-fighting, because it is located close to the mesh surface by design. Another reason is associated to the fact that the zoom range between viewing an entire cuneiform manuscript and zooming to on one of its wedges can be several orders of magnitude. These size related factors are especially important when working with multiple cuneiform fragments at once to create virtual joins. The process of joining fragments often involves aligning wedge size features while zooming out to view the entire fragment. With this in mind, it is desirable that the user never experiences z-fighting or clipping of the mesh against the front or back plane of the viewing frustum to not limit his interaction capabilities. Therefore, the near plane should be as near to the camera as possible and the far plane should be as far away as possible without compromising on the depth accuracy.

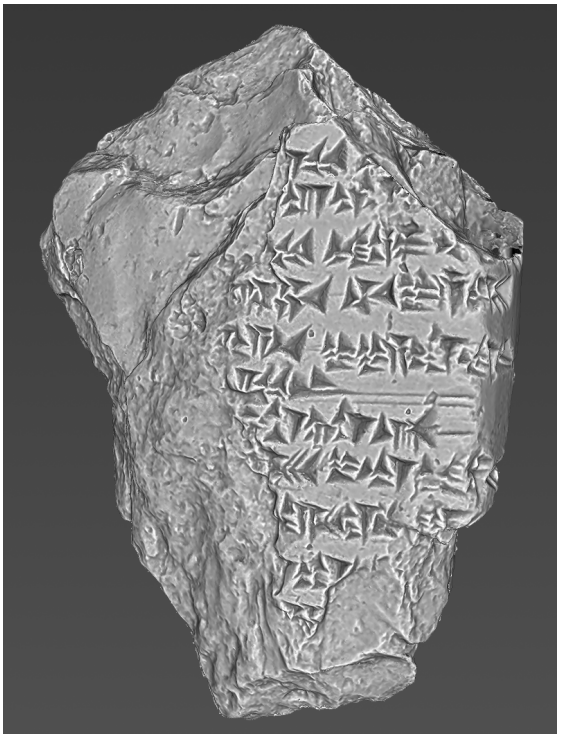
Setting a low distance value of 0.0001 for the near plane and a large distance value of $10,000.0$ for the far plane and using the default 24-bit OpenGL depth buffer, intensive z-fighting can be observed on the cuneiform datasets at usual working distances. This obviously poor depth resolution results from the way the depth values are computed by OpenGL. From a set of eye coordinates (x_e, y_e, z_e, w_e), a near distance n and a far distance f, the clip coordinates z_c and w_c are computed as

$$\begin{aligned} z_c &= -z_e \frac{f+n}{f-n} - w_e \frac{2fn}{f-n} \\ w_c &= -z_e. \end{aligned} \tag{6.7}$$

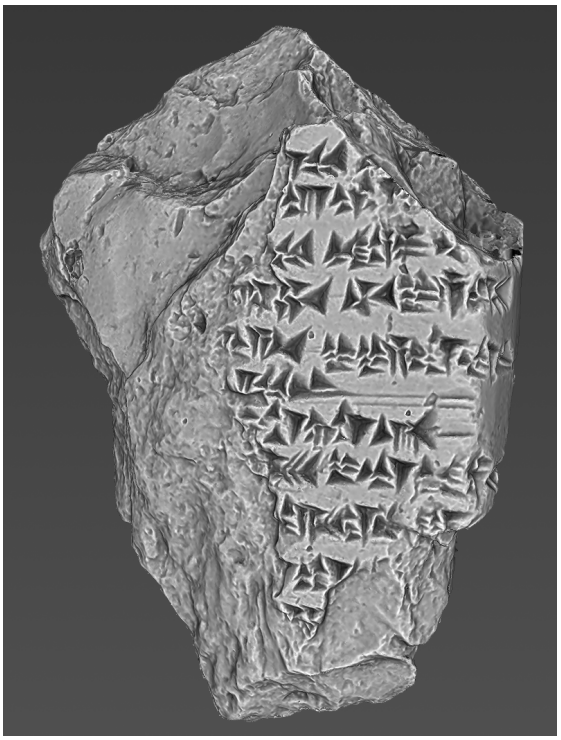
The depth value in normalized device coordinates z_{ndc} is then obtained using a perspective division as



(a) Lambert



(b) Lambert + RS



(c) Lambert + AO + RS



(d) Lit-sphere + AO + RS

Figure 6.8: Rendering improvement over Lambert shading **(a)** with additional radiance scaling(RS) **(b)**, additional ambient occlusion (AO) and radiance scaling **(c)** and additional ambient occlusion, radiance scaling and lit-sphere shading**(d)** on cuneiform fragment *E971*.

$$
\begin{aligned}
z_{\text{ndc}} &= \frac{z_c}{w_c} \\
&= \frac{f+n}{f-n} + \frac{w_e}{z_e}\frac{2fn}{f-n}.
\end{aligned}
\tag{6.8}
$$

The issue arises from the fact that OpenGL interpolates the depth values from the vertex shader linearly, instead of in a perspective corrected way, since it expects the perspective projection matrix to already contain a $1/z_e$ term. The linear interpolated depth values are then used for depth testing, which forces the large majority of available depth resolution to be used for depth values that are close to the near plane. To solve this issue, a logarithmic depth buffer, like described by Brano Kemen [107], is used.

The basic idea of this technique is to compute a logarithmic depth value in the vertex shader, which targets a uniform distribution of the available depth precision on the depth range. A simple implementation of this technique computes the logarithmic depth $z_{\log}$ as

$$
z_{\log} = (2\frac{\log(w_c C+1)}{\log(fC+1)} - 1)w_c \tag{6.9}
$$

in the vertex shader. Here, the factor 2 and the subtraction of 1 account for the OpenGL depth range of $[-1,1]$. The multiplication with w_c compensates the subsequent automatic perspective division by w_c when outputting $z_{\log}$ via `glPosition.z`. The constant C controls the precision distribution and linearizes the depth curve near the camera to avoid issues when linearly interpolating the logarithmic depth values on triangles. For further optimization, the GLSL shader can use the more efficient $\log_2$ instruction, skip the non-necessary transformation into camera space by multiplying with w_c and clip w_c+1 to an epsilon value. The latter avoids issues with triangles crossing the camera plane, which might introduce values smaller than zero into the logarithm and therefore cause unwanted clipping. In addition to that, C can be set to 1. This yields the optimized version of equation 6.9

$$
z_{\log} = 2\frac{\log_2(\max(1^{-6}, w_c+1))}{\log_2(f+1)} - 1. \tag{6.10}
$$

However, this solution still has an issue with large close-up triangles. This results from the fact that the choice of C is a compromise between maximizing depth precision and at the same time minimizing the depth interpolation error. As a countermeasure pixel perfect logarithmic depth values can be computed in the fragment shader after interpolation. For this, the vertex shader has to output a variable containing w_e+1 to the fragment shader that can then compute the depth value as

$$
z_{\log} = (\frac{\log_2(w_e+1)}{\log_2(f+1)}) \tag{6.11}
$$

and write it to `gl_FragDepth`. In the fragment shader, the epsilon clipping can be omitted since the triangles already have been clipped and rasterized at this point. The significantly increased depth precision by using the logarithmic depth values completely eliminates z-fighting in the typically used zooming range when working with cuneiform fragments. This includes viewing fragments from several meters away and zooming in to the triangle level. In close-up viewing, geometry gets only clipped when crossing the camera plane, which enables precise inspection of mesh errors and the visualized meta-data on the mesh.

The precision difference of the logarithmic depth buffer in comparison with the default OpenGL depth buffer in terms of the resolvable depth increments is visualized in figure 6.9. In the double logarithmic graph, the precision for the default OpenGL depth buffer is shown in red and the precision for the logarithmic depth buffer with three different choices of C is shown in blue. Choosing a larger value for C increases the length of the linear section close to the camera. This trades some close-up precision against the possibility to use a vertex shader only version of the logarithmic depth buffer. However, choosing a too small value for C will push the close-up precision in an unusable value range. For cuneiform scans measured in millimeters, the graphs show that the logarithmic depth buffer with $C = 1$ results in an approximately three orders of magnitude increased precision when the camera is 10 centimeters away from the mesh surface. In addition to that, the depth precision is also very usable when viewing cuneiform wedges from a distance of several millimeters. The precision of the default OpenGL depth buffer, on the other hand, becomes unusable at distances above 10 centimeters.

Aside from the visual improvements, the increased depth precision also allows using single depth values, obtained through picking actions, for taking precise measurements on the 3D-scanned meshes. This is important, since determining the location of picked points on the mesh surface by means of intersecting a ray with the mesh triangles on the CPU can get computationally expensive on large meshes. For using the logarithmic depth, the original depth has to be reconstructed by inverting the logarithmic depth formula. It is worth mentioning that measurement points for precise measurements should not be recorded at unusually high camera distances. While measuring the extents of a viewport-filling cuneiform fragment does not suffer significantly from depth buffer precision issues, measuring millimeter-sized wedge depths or distances with a camera located several decimeters away, will. This does, however, not affect distance measurement components parallel to the view plane. The resolution of these is limited by the resolution of the rasterization grid.

The high depth resolution is also used to increase the quality of viewport navigation. In most 3D content creation software, viewport navigation consists of the components for zooming, rotating and panning the viewport. This is usually achieved by activating the respective tool via a button or hotkey and click-dragging in the viewport to navigate the displayed scene. For panning, which triggers a camera motion on a plane perpendicular to the viewing plane, the actual panning distance is often determined by considering a plane p_{target} through the camera target point. This target point is used in many camera models to specify the

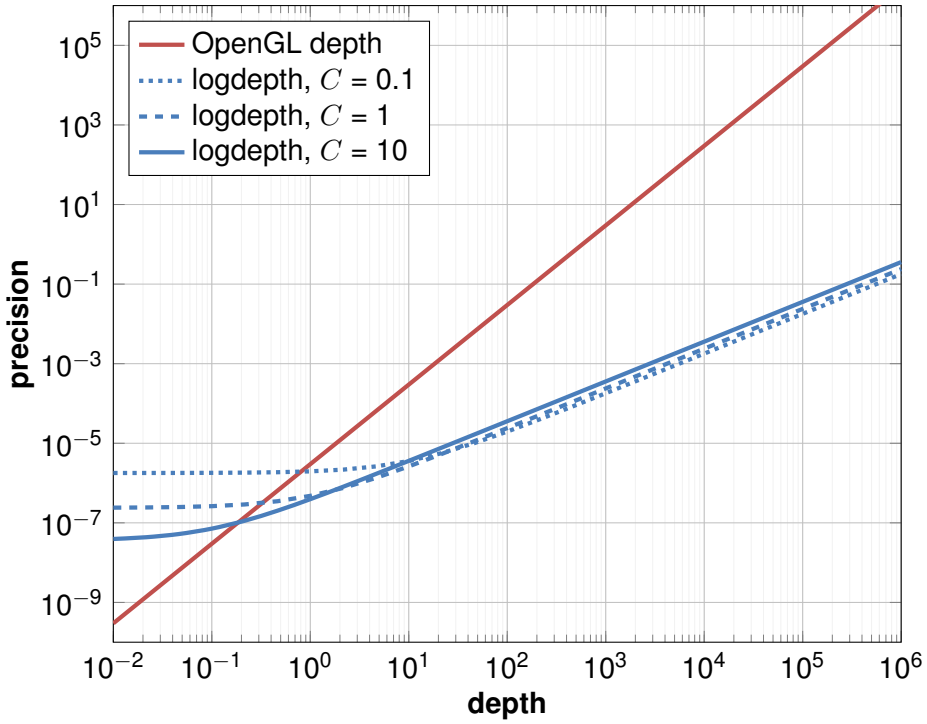


Figure 6.9: Comparison of the precision of the logarithmic depth buffer for 24 bits of available depth resolution.

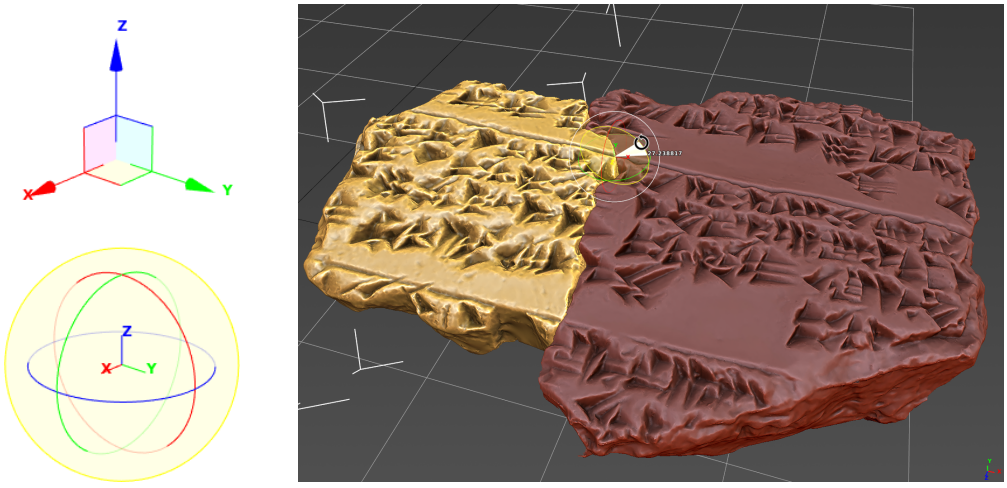
camera focus point and its center for camera rotations. The actual panning distance is now determined by ensuring that the point on p_{target} that was projected at the mouse position when starting the click-drag action stays at the mouse position during dragging the mouse. If the camera uses an orthographic projection, this results in a perfect panning motion, where whatever point of the displayed 3D scene was at the mouse position, stays at the mouse position. For a perspective camera, however, the parallax effect causes points in 3D space to move slower or faster depending on their distance to p_{target}. As a result, the position of the scene that was "grabbed" for panning slides away from the mouse position during dragging. This can be avoided by determining the scene depth at each starting point for a panning operation, reconstructing the corresponding point in 3D space and computing the camera panning distance with respect to a plane through this point. This so-called "pixel perfect panning" greatly improves the precision of camera panning operations as it corresponds better to the subjective expectation for clicking and dragging a point in 3D space. A high depth resolution ensures that the clicked point can be reconstructed well for a large range of depth values. However, when the picked point on the viewport does not correspond to a position on a 3D surface in the scene but belongs to the background, the usual plane p_{target} at the camera target point should be used.

6.4.5 Fragment joining

To support virtual fragment joining, a flexible transformation system has been integrated that supports intuitive and fast alignment of matching cuneiform fragments. For this, the framework utilizes a set of viewport-based interactive alignment helpers called “transform gizmos”, which were inspired by popular animation programs like Blender[2] or Autodesk 3dsmax[3]. These transform gizmos, which are shown in figure 6.10a, are semi-transparent manipulator objects addressing one transformation type, such as translation or rotation. The translation gizmo in the upper half of figure 6.10a consists of a triplet of vectors, representing the x, y and z translation axis and a matching triplet of planes in-between, representing the xy-, yz-, and xz-planes. Moving the mouse cursor over one of these components highlights its graphical representation and changes the cursor to a symbol representing the action. This way, the transformation component and the possibility to start a transformation action by left-clicking and dragging the mouse in the viewport are always indicated in real time. Similarly, the rotation gizmo in the lower half of figure 6.10a consists of a small axis vector representation with a large colored circle for each axis that highlights upon hovering over it with the mouse. Click-dragging such a circle starts a rotation action around the respective axis including the rotation angle being visualized as a solid section on the circle. The larger circle, encompassing the interaction circles for the three axis, can be used to perform rotations around an axis perpendicular to the viewport plane. This enables the user to perform rotations around arbitrary axis by choosing the respective viewing direction. In addition to that, the rotation gizmo can be affected by a snapping function that restricts the allowed angles to multiples of a user definable angle. All transform gizmos use a consistent color coding for the three axis. Since scanned cuneiform meshes are usually represented in real world scale using millimeters, a scale gizmo is not provided by default to prevent the user from mismatching the scale of multiple scans.

Usually an alignment process for two fragments consists of moving one fragment near the intended target fracture face or feature on the other fragment and then adapting the rotation afterwards. Since modifying the rotation around the fragment center almost certainly has a negative effect on the alignment of the fracture faces, this process usually has to be repeated multiple times. To speed up the alignment process, the framework has been extended to be able to place the center of the transform gizmos on any surface position on the respective fragments. This way, the gizmo can be placed on a surface feature, like a paragraph separator or a wedge near a fracture face. The fragment can then be translated so that the feature is near its counterpart on the other fragment, and, as a last step, the fragment is rotated into place. Since the rotation is performed around the manually set rotation center at the surface feature, the alignment is not affected negatively by the rotation and additional translation-rotation iterations are not required. To ensure a scale independent usability of the transform gizmos, the helper objects are rendered at constant screen sizes. In addition to that, any transformation

[2] https://www.blender.org/
[3] https://www.autodesk.de/products/3ds-max



(a) (b)

Figure 6.10: Virtual fragment joining. The transform gizmos for translation and rotation are shown in **(a)** and a joining operation on two cuneiform fragments in **(b)**, with a rotation gizmo positioned at a paragraph separator transition.

in process can be canceled to correct inadvertent transformation actions. All this results in an intuitive and fast transformation system for conducting virtual joins on cuneiform fragments.

6.4.6 Meta-data visualization

An intuitive and flexible visualization of segmentation related meta-data is an important component of cuneiform wedge segmentation. In this context, the term meta-data refers to any kind of data that is derived from or added to geometric features of the 3D-scanned cuneiform meshes. Components of interest not only include the extracted wedge models from chapter 4 and the skeleton data from chapter 5 but also intermediate segmentation data associated with the underlying mesh and manually added annotation-based data. While a significant subset of the considered types of meta-data has a geometric representation that can be visualized in 3D, this is not always the most efficient or the most usable representation. In these cases, list-based visualizations, including access patterns for sorting and selecting data, are employed to complement the geometric visualization in the viewport. Therefore, the meta-data visualization concepts covered in this section can be divided into two main groups, list-based visualization and geometric visualization which will be described in the following.

The structural data components of the framework concept detailed in section 6.2, namely projects, cuneiform fragments, selections, and meta-selections, are each associated with a specific set of meta-data. List-based visualizations are provided for all structural elements that can occur more than once in order to improve handling of large quantities of elements. The affected elements include cuneiform fragments, selections and meta-selections but not projects since the framework is designed to handle only one project at once. Although each of those elements is equipped with a specific set of properties, all elements share several default

properties like an ID, a name, a comment, a visibility attribute and a color. While the ID is an integer, assigned automatically upon creation and used internally as a unique identifier for the elements, the name is a meaningful textual identifier to describe the element. In the case of cuneiform fragments, this is usually a museum inventory number like “Bo532” or “133/r Vs”. For meta-selections on the other hand, an intuitive name could be the name of a cuneiform sign or a word. Since the role of a unique identifier is already taken over by the ID, the name does not need to be unique. However, upon automatic creation of elements, the framework will try to assign a meaningful and unique name. In the case of cuneiform fragments, this role is taken by the filename and for selections and meta-selections a string composed out of a word and a number, like “Cluster 21”, is used. The name can be complemented by a comment in form of a more descriptive piece of text that can include philological findings. The remaining two default properties are related to the viewport appearance of the elements. While the color property holds a user-defined color for displaying the elements in the viewport, the visibility attribute specifies whether an element is visible in the viewport. The latter also controls the ability of elements to be selected via the viewport, which is not only useful for working on only a subset of cuneiform fragments but also for restricting user actions to, for example, one side or one paragraph of a cuneiform fragment. The main role of the color, on the other hand, is visual distinctiveness in the viewport. This is not only important for individual cuneiform fragments but especially for coloring the selections and meta-selection on the meshes. To ensure a high probability of two adjacent elements being colored differently, a random color is assigned to the elements upon creation, which can be manually changed by the user at any given time. Especially for selections, the color can also be generated depending on properties like the wedge type from section 4.8.3 or the amount of similar selections, as computed in section 4.9.2.

Aside from the default properties, each element contains an additional set of element specific properties. While for cuneiform fragments this includes the filenames of the mesh and meta-data files and the number of created selections, the selections include more than 20 direct and derived properties of the wedge model, like the wedge depth, the coordinates of the feature points and the number of similar wedges. To improve the handling of large amounts of properties, each property is displayed in a separate list-column and the columns can be individually hidden to increase visual clarity. This concept is pushed further through the ability to define and store named sets of visible property columns for different tasks. In addition to that, the whole element list can be sorted according to each property, which facilitates selecting specific subsets of elements depending on their meta-data properties. The available data types for the properties include strings, numbers, number lists, colors and named enumerations and all of these properties can generally be edited directly in the list. However, some properties, like IDs and filenames, are marked as read only to prevent unwanted modification by the user. This allows a very intuitive and fast access to all properties. For a rudimentary statistic evaluation, the mean value and the standard deviation of numerical properties is shown for all selected elements when hovering the mouse over the respective column header. This can

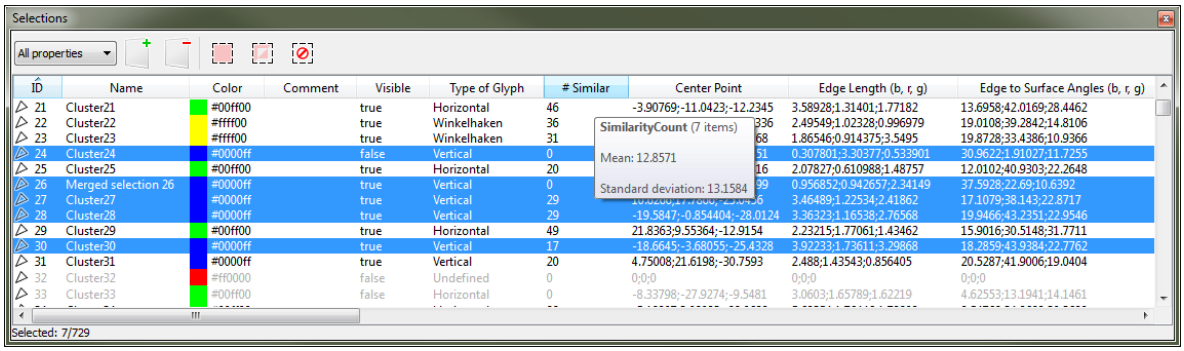


Figure 6.11: Exemplary view of a list visualization of the selections on a fragment with several properties.

help with assessing the statistical significance of the values. Further statistical analyses are supported by an integrated export functionality that can output the properties of all selected elements and all visible property columns in a list to a text-based comma-separated file format, which can then be read in by advanced statistics software.

Figure 6.11 shows an exemplary list visualization of selections, where selected selections are highlighted in blue and hidden elements are rendered with gray text. Here, the gray pop-up box in the center shows the mean and standard deviation for the number of similar selections of the selected elements. The drop-down and the two left-most buttons at the top can be used to manage the named sets of visible property columns while the three right most buttons in the same toolbar are global selection modifiers to select all elements, to invert the selection, or to select nothing. To improve editing speed for larger amounts of elements, editing a property of one element when multiple elements are selected leads to the property being set to the same value for all selected elements. This way, hiding and unhiding large subsets of elements or assigning custom element colors becomes much more efficient.

The 3D-viewport-based visualization of meta-data uses a layered approach in which the visibility of individual components follows the same rules for all displayed elements. This guarantees information consistency on the visualized elements and prevents false interpretations of visualization components in static images. In contrast to the globally defined visibility of visualization components, the local visibility of meta-data is controlled by the visibility of the respective elements. The used visualization components can be grouped into three main categories, namely, segmentation model drawings, point- and triangle-based mesh colorization, and viewport labels. For each selection, the segmentation model drawings include the vertices and edges of the wedge model, like introduced in section 2.2.3, and the skeletons extracted in chapter 5. In addition to that, the edge data points from section 4.8.2 can be shown as white dots to visualize which mesh vertices have been used for fitting the wedge edges. Likewise, the wedge face normals and the interpolated writing direction can be displayed for each selection to extend the possibilities of manually assessing the wedge extraction results with respect to the local mesh geometry.

The colorization of the mesh can be either fragment-, point- or triangle-based and is used as the diffuse base color for the surface shading described in the previous sections. In fragment color mode, one global color is used for each fragment, which can help with distinguishing the fragments during virtual joining operations. Possible colorizations in point color mode include per-vertex curvatures, normal colors, ambient occlusion or the surface colors recorded during scanning the artifact, most of which can be used to assess the surface quality. The triangle-based colorization options include showing a per-selection or per-meta-selection color, the color-coded cluster assignments from wedge face extraction described in section 4.8.1, and an index color mode.

Figure 6.12 shows several examples for different combinations of visualized meta-data. In figure 6.12a, the mesh triangles are colored according to the wedge face clustering results, with valid edge data points visualized as white dots. The dark-gray colored patches on the mesh are hidden selections that have been emphasized by using a darker color than the surrounding geometry. This way, hidden selections can still be spotted and selected in the viewport. Figure 6.12b shows the same wedge configuration with drawn wedge models and the triangles being colored in a selection specific color. Like in the model specification in section 2.2.3 the inner edges are colored in red, green, and blue, while the outer edges are colored in orange. The wedge vertices share the color of the respective edges. Since large parts of the wedge model drawings are occluded by the underlying mesh, the wedge edges are extended by thin extension lines that overshoot the outer vertices. This way, the direction of the edges can be seen more effectively, without just drawing the wedge models on top of the rendered mesh and destroying the perceivable depth relation between the lines and the mesh. For comparison, the same wedge models are shown in figure 6.12c without an interfering mesh.

Figure 6.12d shows meta-drawings for the wedge skeletons extracted with the method presented in chapter 5. For this, the skeleton branches are colored using a gradient starting with the selections' color at the end with the largest distance from the free writing surface and fading to white. Since the skeleton branches are not linear, the issue of line occlusion through the mesh is addressed here by letting the occluded parts of the polylines shine through the mesh with an opacity of 50%. This prevents the depth relation between the lines and the mesh from becoming imperceptible. However, viewing a cuneiform fragment with superimposed renderings of skeletons from the obverse and the reverse side can still become confusing in static images. This should be avoided by hiding the elements on the reverse side manually. To check the correctness of the interpolated writing direction, it can be visualized in the form of a white and red two-line drawing, like shown in figure 6.12e. Here, the white lines are drawn between the depth point of the wedges and the nearest point on the writing surface. At the contact point, a perpendicular line with a red to white gradient indicates the writing direction. Being able to check the interpolated writing direction is particularly useful in regions with oppositely oriented interlocking text lines to decide whether additional writing direction lines must be defined. The small red, green, and blue colored lines in figure 6.12e represent the average normals of the triangle patches associated with the wedge faces.

As a last example for an exotic meta-data component, figure 6.12f shows a mesh triangle colorization that maps the triangle indices in the mesh file onto a multi colored gradient ramp, like already used in section 6.3. As a result, the relation of different mesh parts to the different scans merged to create the cuneiform mesh becomes visible. Adjacent triangle patches with similar colors usually originate from the same scan, while sharp features between such patches are the result of mesh fusion and remeshing operations. This visualization can help with determining geometry patches that have undergone extensive geometric modification during scan merging and are therefore less reliable. However, this only works on scans that have been merged using local remeshing methods since global remeshing methods may reorder all triangles consistently. Comparable results can therefore only be obtained for scans that have been post-processed using the same parameters for scan fusion in the scanning software.

Another essential aspect of viewport-based visualization is the handling of selected elements. In the application case covered here, being able to distinguish selected elements from unselected elements in the 3D viewport is predominantly important for selections and meta-selections. This results from the fact that unlike cuneiform fragments, selections and meta-selections usually occur in larger quantities, which prevents all selected elements from being visible simultaneously in the respective list-views. Also, fragments usually exhibit a highly distinctive shape that can be easily recognized and associated with the fragment name. Due to the high similarity of cuneiform wedges, this does not work for large amounts of selections and meta-selections. With a viewport-based selection indicator, on the other hand, hundreds of selected items can be viewed at the same time when adapting the camera zoom factor accordingly. In addition to that, a viewport-based visualization can also consider the geometrical shape of the selected items and their spatial distribution. However, increasing the extents of the viewport to fit more elements can reduce the rendering area of individual elements down to a size where they become barely visible.

Highlighting selections and meta-selections in the viewport is therefore realized by a combination of color modification and displaying visual element labels. Since selections and meta-selections are almost always associated with a subset of mesh geometry, like vertices or faces, the color of the associated geometry is modified to indicate selected elements. For display modes, where the displayed color of the underlying geometry is already associated with some meta-data, like when displaying the cluster assignment of the selection faces, the existing color is brightened. In all other cases, the color is blended with a saturated red to indicate the selection status more prominently. In addition to the color modification, a textual label of the element ID is displayed near the geometric center of the selected elements. The element ID is used here because it is the shortest unique identifier that is available for all elements and is also used in the list-views to identify the elements. Since meta-selections are composed of multiple selections, the labels are additionally color-coded with black text for selections and blue text for meta-selections. A real-time synchronization of the elements' selection status in the viewport and the list-views allows a very flexible and intuitive handling.

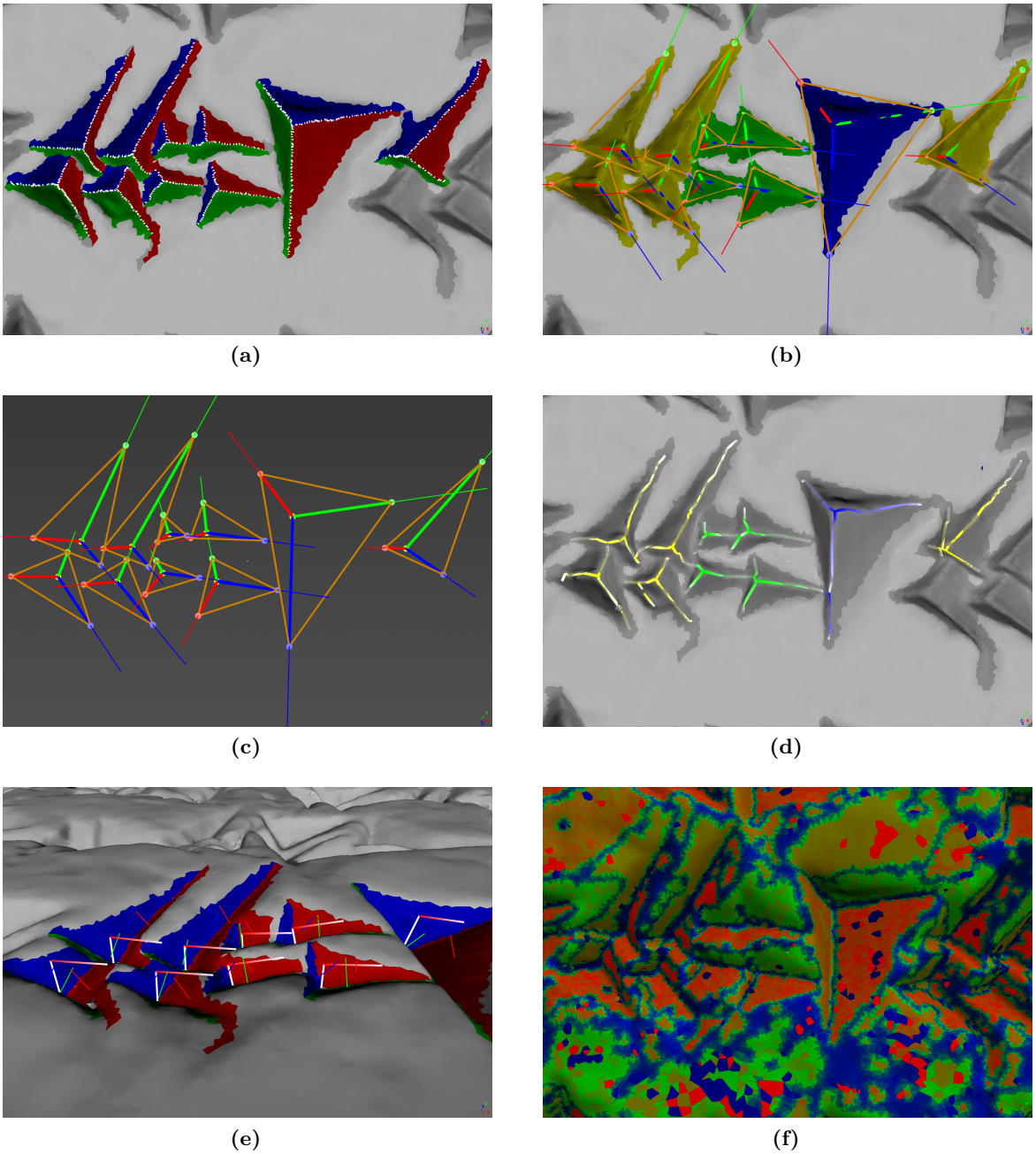
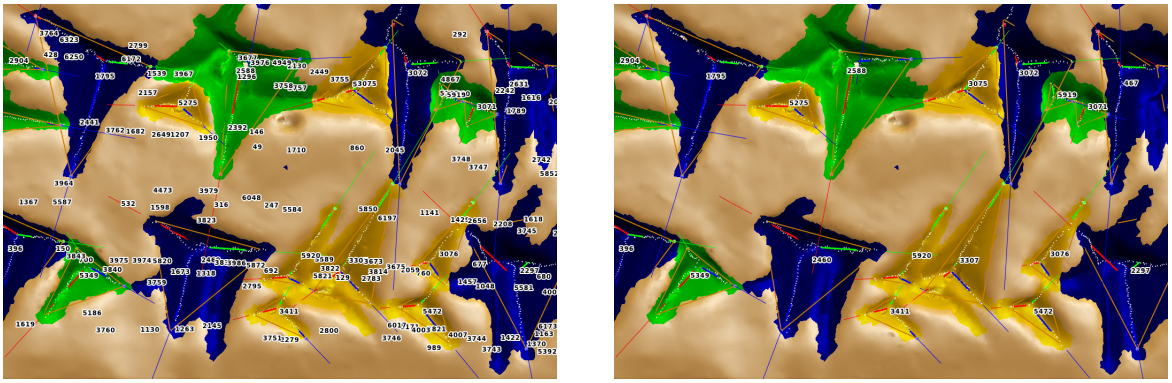


Figure 6.12: Section of cuneiform fragment *Bo532* with different combinations of visualized meta-data.

To optimize visibility, the viewport labels are rendered as dark text with a bright border. This ensures a good readability regardless of the background color. In addition to that, the labels are rendered on top of the displayed geometry to ensure that the labels can not be occluded by mesh geometry regardless of the viewing direction. The issue of multiple overlapping labels occluding each other is not addressed explicitly because the usual creation of selections

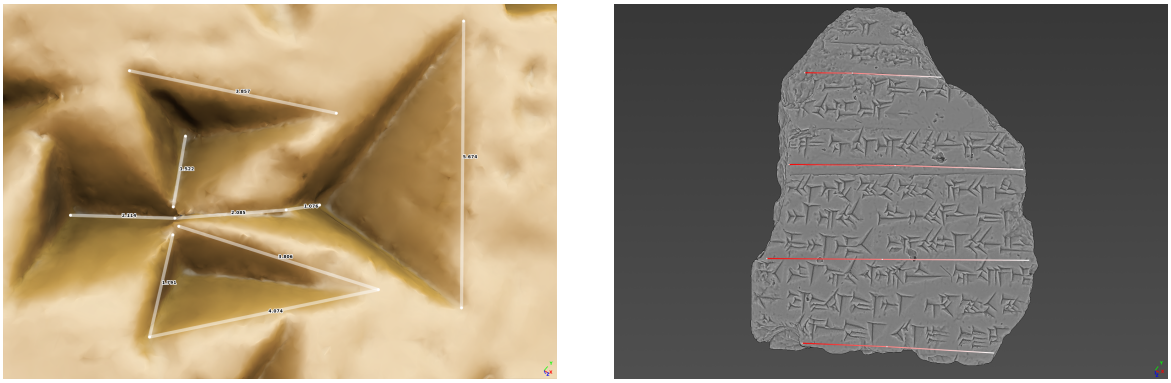


(a) **(b)**

Figure 6.13: Section of cuneiform fragment *133/r* with confusing labels shining through from the reverse side **(a)** and with orientation-based label filtering **(b)**.

during wedge segmentation counteracts an unfavorable placement of labels. This results from the selections being constructed as non-overlapping, neighboring patches of geometry, which naturally sets the labels apart from each other. However, the missing depth component can be an issue on cuneiform fragments that include a reverse side since labels on the obverse and the reverse side can only be distinguished while rotating the viewport. To address this issue, a filter can be activated that controls the visibility of the labels based on the average normal of the associated selection geometry. A label is only displayed if the average normal of the respective selection is oriented towards the viewer. This effectively filters out the selections on the reverse side of a fragment and counteracts confusing label rendering, which is especially useful when orbiting a cuneiform fragment while displaying all selection labels. As can be seen in figure 6.13, this can be essential for a visually unambiguous label placement. In general, the labels of meta-selections are affected by the same issues as the selection labels. However, since meta-selections are usually composed of multiple selections, the labels are packed less dense. More details on the element selection method will be given in section 6.5.2.

In addition to the types of meta-data described up to this point, the framework includes several manual input modes for points and polygonal lines. These can be defined on the mesh surface of each fragment and are used for measurement lines to measure the distance between points, measurement points to measure the writing surface distance of locations on the respective meshes, and for defining the writing direction as required for the wedge classification in section 4.8.3. The input mechanism for points and polygonal lines works by left-clicking locations on the mesh while holding down the Shift key. For polygonal lines, clicking consecutive locations will append segments to the line while releasing and pressing the Shift key again starts a new line. Actions can be undone by right-clicking. This way, lines can be defined very efficiently including a simple correction mechanism. Two examples for measurement lines and writing direction lines are shown in figure 6.14. The measurement distances are shown directly in millimeters at the center on the respective uniformly colored



(a) **(b)**

Figure 6.14: Measurement lines **(a)** and writing direction lines **(b)** on cuneiform fragment *Bo532*.

line segments. In contrast to the measurement lines, the writing direction lines are visualized with a red to white color gradient since the direction of the lines is an essential property.

6.5 Geometry selection

Being able to select geometry directly in the viewport is an important mechanism to manually interact with geometric data in a 3D segmentation framework. Regarding cuneiform segmentation, this includes selecting vertices and triangles of the 3D-scanned meshes and the ability to select the associated meta-data in form of the selections and meta-selections introduced in section 6.2. The primary use case for selecting geometric primitives, like vertices and triangles, on the 3D-scanned mesh is to manually create new selections for segmentation. In addition to that, newly selected patches of primitives can be used to modify existing selections by adding or subtracting these primitives to or from the selections. In the case of overlapping geometry obstructing the view, it is also desirable to be able to hide geometry in the viewport. This allows inspecting surface features that would otherwise be obscured or to hide the reverse side a cuneiform fragment in order to select geometry exclusively on its obverse. In contrast, for working with point and face selections, the most important feature is the ability to click-select single and multiple selections in the viewport for displaying their properties but also for being able to create meta-selections from multiple selected selections. The capability to select selections and meta-selections in a specific area of the cuneiform manuscript, like a text line or a paragraph, is an additional useful feature. The following will discuss the specific requirements for selecting these kinds of data and present a set of fast and flexible concepts to meet these requirements.

Regarding the requirement to select geometry and meta-data in arbitrarily shaped regions of the mesh, a flexible selection mechanism is required. This is realized by providing a selection polygon-based input method for the creation of user defined selection areas in the viewport.



Figure 6.15: Five complex selection polygons (green) in XOR-mode on cuneiform fragment *Bo532*. The selected patches on the mesh are highlighted in red.

These viewport selection areas are then projected onto the displayed geometry to determine the selected geometric elements. To provide maximum flexibility, the polygonal selection area can consist of multiple polygons that are combined according to the rules of a selection mode. This selection mode can be configured to act additive, subtractive or in an XOR way and successively combines multiple polygons according to the rules of constructive solid geometry (CSG) [135]. To facilitate the input of the polygons, the interface supports creating rectangular screen aligned polygons by holding down the Shift key and click-dragging in the viewport. An arbitrary polygon can be created by holding down the Shift key and consecutively clicking in the viewport to add more points to the polygon. Releasing and pushing down the Shift key again completes the current polygon and begins a new one. This way, multiple complex polygons with many vertices can be created with minimal input effort using only one keyboard key and single clicks. An example of five complex selection polygons in XOR-mode is shown in figure 6.15, where the border of the polygon is visualized as a closed green polygonal line segment with green dots at the polygon vertices. The resulting selected area on the mesh is highlighted in red.

6.5.1 Primitive selection

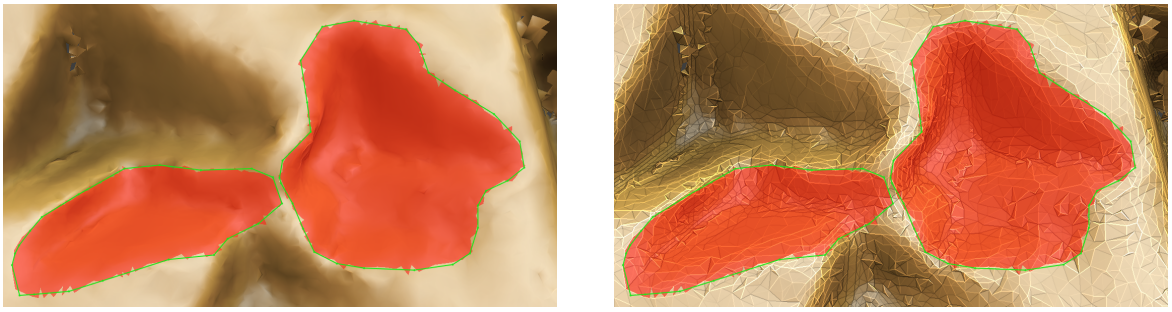
As already mentioned, the requirements for selecting geometric primitives are limited to selecting patches of vertices and triangles on the 3D-scanned mesh. In view of the large size of the processed meshes, this necessitates a GPU-accelerated selection approach to ensure interactive selection feedback. Being able to immediately see the resulting set of selected primitives is especially important in cases where neighboring primitives can not be distinguished well based on the employed visualization. This situation does frequently occur on shaded triangle meshes with smoothly interpolated normals. The immediate display of the selection

result for each newly entered point of the selection polygon instead of displaying only the result for the finished set of selection polygons also allows more precise positioning of the polygon points to obtain the desired result.

Another important requirement for the primitive selection method is the ability to select primitives regardless of their orientation and their occlusion by other primitives. The method should ensure that all primitives inside the selection area get selected to reflect the real topology of the selected geometry. In this respect, issues can arise from the fact that the meshes may contain mesh errors and locally inverted surface patches facing away from the viewing direction, including occluded geometry. These surface patches can not be selected if the selection method only processes front facing non-occluded geometry. Although an OpenGL fragment shader can be configured to process fragments regardless of orientation and occlusion, it can not handle primitives that produced no fragments during rasterization. Since fragment-less primitives frequently occur for triangles that are perpendicular to the viewing direction and for triangles which cover only a very small area on the screen, the selection method should not rely on using the OpenGL rasterization to speed up the computation of the selected geometry. Instead, the proposed method relies on the OpenGL transform feedback extension [113]. This extension allows to capture the output of a vertex shader and stream the results into a buffer object. While usually employed for processing vertices multiple times before rasterization, transform feedback can also be used to process data in form of vertex attributes and write the results to a buffer object while deactivating the rasterization dependent part of the OpenGL rendering pipeline. The per-vertex computation results can then be downloaded from GPU memory to the CPU.

Starting with the selection polygons described earlier, the primitive selection is computed in two passes. The first pass includes drawing the triangulated selection polygons into the OpenGL stencil buffer [112], using the stencil operations to realize the CSG operations of the selection mode. The previously black colored 8 bit stencil buffer then contains a selection mask where all active polygon regions are drawn in white and the remaining area is colored black. In the second rendering pass, the primitives to compute the selection for are drawn using transform feedback with deactivated rasterization. The vertex shader then transforms the incoming primitives into clip space and uses the mask in the stencil buffer, which is bound as a texture, to determine whether a primitive should be selected based on the xy-coordinates. Eventually, the result is streamed into a previously prepared vertex buffer object and, from there, copied to CPU memory to update the mesh data structures.

Since the transform feedback works vertex-based, the mesh vertices can be directly used for the transform feedback pass. However, rendering the triangles directly with transform feedback would result in three results per triangle that would have to be combined. In addition to that, using the three vertices of a triangle for computing the selection could lead to non-intuitive selection results since a large portion of a triangle can intersect with the selection area without any of the triangles vertices being in the selection area. Obtaining a precise result would require



(a) (b)

Figure 6.16: Detail of a multi-patch primitive selection without **(a)** and with flat shading **(b)**.

to intersect the triangles with the selection area. This requires all three triangle vertices to be accessible, which, in a vertex shader, is only possible by duplicating the vertex information into separate vertex attributes. However, aside from tripling the memory consumption, a triangle/polygon intersection is a computationally expensive operation. Moving this operation from the vertex to a geometry shader can eliminate the additional memory consumption but not the computational overhead for invoking a geometry shader and computing the intersection. Rasterizing the triangles and computing the intersection in the fragment shader is also not an option due to the previously mentioned occurrence of fragment-less primitives.

Therefore, a much less complex method is employed that compromises on the selection accuracy. For this, the mid-points of the triangles are pre-computed and rendered instead of the triangle vertices. This has the advantage that the triangles are selected based on their mid-points, which provides a better approximation of the triangles selection area coverage than the triangle vertices. To further increase the performance, one pre-allocated buffer object for the transform feedback results can be reused for both vertices and triangles without wasting too much memory. This results from the fact that, according to Eulers theorem, the number of triangles m in an evenly tessellated mesh is approximately two times the number of vertices n in that mesh. Excluding special cases, it is therefore sufficient to allocate a buffer of $m \times 32$ bytes, when the selection results are stored in a 32 bit data type.

A wedge-scale selection result obtained using the described method is shown in figure 6.16. It can be clearly seen that the computed triangle selection highlighted in red closely follows the green selection polygon without containing an extensive amount of spikes. This indicates that using the triangle centers for sampling the selection polygon is an approximation of sufficient quality for this use case. For cases requiring a spike-free selection border, a morphological closing operator could be employed to smooth the border. In contrast, an increased selection precision of the user can be achieved by rendering the mesh using flat-shading in combination with radiance scaling. As can be seen in figure 6.16b, this makes the transitions between non-coplanar triangles visible to the user, who can then take the triangle positions into account upon creating the selection polygon.

6.5.2 Wedge and meta-data selection

In contrast to the selection of geometric primitives, which is computed in a one-to-one mapping, the selection of wedge areas and the associated meta-data, which are represented by so-called selections, requires a many-to-one mapping. This results from the fact that selections consist of a set of vertices or triangles of the underlying mesh and of a set of meta-data, including the wedge models. These two representation layers of a selection are not necessarily required to be visible at the same time. Instead, hiding the geometric primitives of a selection to exclusively inspect the wedge model and hiding all the meta-data to exclusively inspect the underlying geometry, are common usage scenarios. This leads to the requirement to be able to select the associated vertex- or triangle-selections regardless of the current display mode. More precisely, a selection should get selected if any of its geometric primitives or relevant elements of the meta-data visualization is inside the active area of the selection polygons. In addition to that, selections should be selectable by double-clicking the associated geometric primitives in the viewport, including the ability to select and deselect multiple selections by holding down the Ctrl key. The result mimics the selection capabilities in a list-view and can therefore be used intuitively.

For a hardware accelerated realization of the required many-to-one mapping, OpenGL floating-point blending is used in a similar way like during the computation of the ambient occlusion in section 4.5.1. As a prerequisite, a floating-point texture T_{res} is prepared that can store as many pixels as there are selections on the current cuneiform fragment and is initialized to be filled with zeros. With most OpenGL hardware supporting a maximum texture size of 16384×16384 pixels and the number of selections being typically several orders of magnitude lower than the number of primitives, it is sufficient to use a 2D texture. This way, a maximum of $268,435,456$ selections can be handled. However, in most cases, the number of selections is smaller than $50,000$.

The basic idea of the proposed approach is to use a selection computation rendering pass and render to the prepared floating-point texture with activated additive blending and deactivated depth test. The elements to be rendered include all primitives of the current cuneiform fragment and the four feature points of the wedge models of all existing selections. For this, the primitives, in form of vertices or triangle-centers, can be rendered from the same vertex buffers as used for primitive selection. However, the primitives and the feature points to be rendered are additionally equipped with a vertex attribute that contains a unique, continuous ID for every selection. This is done with the intent to map the result of each primitive test in the vertex shader to a position in the texture that corresponds to the specific selection. Since the selection IDs in the CuneiformAnalyser are not necessarily continuous, a set of continuous IDs has to be generated once at the beginning of each selection process. For this, the set of selections is traversed in ascending order to assign ascending temporary IDs starting with the ID 0. The integer IDs are then packed into a 32 bit attribute and stored in the additional vertex attribute for each rendered primitive and every feature point.

Much like for primitive selection, the vertex shader transforms the incoming primitives into clip space and uses the selection mask in the stencil buffer, which is bound as a texture, to determine whether a primitive is inside the active area of the selection polygons. The result is passed to the fragment shader. In addition to that, the packed temporary selection ID associated with the primitive is decoded. The selection ID is used in combination with the texture size to determine the respective pixel position of the selection in the result texture and the vertex coordinates are modified accordingly and output via `gl_Position` to the fragment shader. This way, all primitives and feature points belonging to one specific selection are mapped to the same texture pixel that accumulates the results. As a last step, the texture is read back to CPU memory and the selection result is reconstructed using the temporary selection IDs. If the pixel value associated with a specific selection is larger than zero, the selection is selected. Otherwise, it is deselected. Thus, a selection is selected if any of its associated primitives is inside the selection polygon, which results in a very intuitive selection behavior.

Special attention has to be paid to the ID-based mapping of the primitives to the positions in the result texture. Usually, the texture coordinates (t_x, t_y) corresponding to the correct pixel positions for a decoded 32 bit integer ID are calculated as

$$\begin{aligned} t_x &= \text{int}(\text{mod}(\text{selectionID}, \text{selTexSize})) \\ t_y &= \text{int}(\text{selectionID}/\text{selTexSize}), \end{aligned} \tag{6.12}$$

where selTexSize is the width of T_{res} and selectionID contains the integer ID. Again, before writing t_x and t_y to `gl_Position.xy`, the coordinates is mapped from $[0,1]$ to the OpenGL clip coordinate interval of $[-1,1]$ and corrected by an offset of half the pixel width to precisely match the output pixels. Although the modulo computation in equation 6.12 is generally susceptible to the same accuracy issues as described in section 6.3, the issues are very unlikely to occur here. The reason for this is the comparatively small number of selections that usually does not exceed several ten thousands.

As already mentioned in section 6.5.1, rendering the primitives as vertices or triangle mid-points that are mapped to the output texture has the advantage that the selection process does not depend on rasterization-based visibilities of the primitives. As a result, even primitives with no rasterized fragments can be selected. The downside of this technique is the fact that the vertex representing a point and the mid-point of a triangle must be in the selection polygon to trigger the selection mechanism. If the rasterized size of the triangles or points covers a significant area on the screen, this could lead to the selection mechanism to be perceived as inaccurate. Cuneiform 3D scans are, however, usually densely tesselated and consist of well-balanced triangles, which makes selection situations like this highly unlikely to occur. An alternative way of handling this would be to map the resulting fragments of the rasterized points and triangles to the selection texture. This would trigger the selection mechanism if any of the primitive's pixels are inside the selection polygon. However, the accuracy advantage with respect to primitive area coverage and the omitted requirement to calculate and store

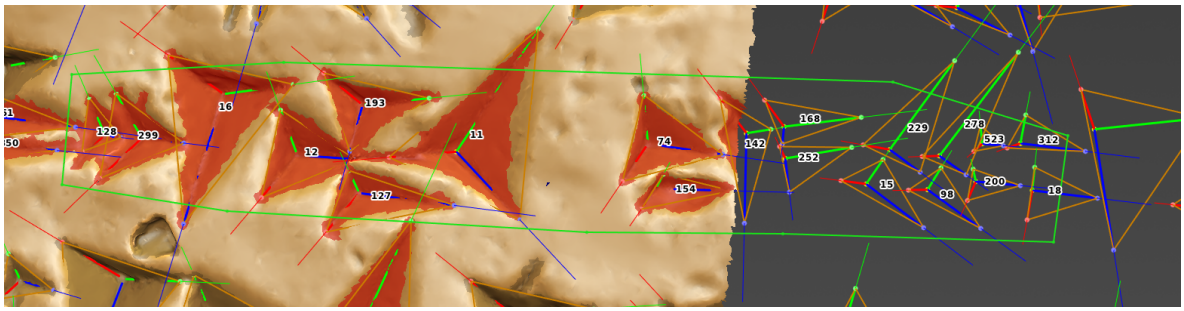


Figure 6.17: Example for a mixed mode viewport selection of selections on cuneiform fragment *Bo532*. Selected selections are highlighted in red on the mesh and their selection ID is shown as a label.

triangle mid-points would have been gained at the cost of the inability to handle fragment-less primitives and a performance loss through the now required rasterization.

An example for a mixed mode viewport selection of selections is shown in figure 6.17, where the mesh in the right half has been hidden via a primitive selection. As intended, the selections within the green selection polygon get selected regardless of the mesh visibility. Selected selections can be distinguished from unselected ones via the red highlighting on the mesh and the displayed selection ID labels. Although the shown selection polygon is intended to select only selections in the text line at the top, one selection in the text line at the bottom gets also selected, because a small fraction of its primitives is inside the selection polygon.

For an improved user experience, selecting selections in the viewport is real-time synchronized with the additionally available list-based visualization of the selections. Selected selections in the viewport are immediately highlighted in the selection list and vice versa. As a result, all selection operations can be performed alternately in the viewport and the selection list.

6.6 Performance evaluation

Relevant visualization related performance aspects of a real-time cuneiform segmentation framework include the mesh rendering performance, which determines the general smoothness of the user interaction with the viewport, and the responsiveness during the selection of geometry. At the same time, the memory consumption for loading cuneiform scans of various sizes is of interest since it determines which mesh sizes can be handled interactively on the available hardware. Aside from the overall rendering speed, the performance with respect to updating the GPU data with new values from the CPU mesh data structures is an equally important factor in order to maintain the impression of interactive and fluid operability. The following subsections will therefore evaluate the above-mentioned aspects on the set of cuneiform fragments already used in section 4.10.3.

Fragment	Vertices	FPS (navigate)	FPS (prim sel)	FPS (meta sel)
Bo532	587,053	417.92	61.37	40.61
Bo2743	1,610,047	235.15	34.72	20.57
Bo51	3,938,454	97.21	17.13	9.32
126/p Rs	7,034,195	63.19	10.21	5.41
101/b Vs	8,742,711	48.82	8.58	4.30
485/b Rs	12,598,242	33.34	6.42	3.18
482/b Vs	14,328,507	31.06	5.64	2.78
VAT7456 Vs	16,141,130	23.12	4.58	2.32
VAT422 Vs	33,889,749	10.54	2.23	1.12

Table 6.1: The mesh rendering performance for the fragments from figure 4.51. Measured are the average frames per second in the viewport for navigating the mesh, for continuous primitive selection (triangles), and for continuous meta-data selection.

6.6.1 Mesh rendering and geometry selection

In order to measure the rendering performance for viewport navigation and geometry selection, the cuneiform fragments from section 4.10.3 were opened in the framework and zoomed to fill the viewport, while at the same time ensuring that no parts of the geometry were outside the viewport. This setup was chosen to guarantee that all primitives were rasterized and no geometry was clipped, which can both have an unwanted positive impact on the measured rendering speed. The loaded fragments were then rotated continuously using the viewport navigation and the resulting frames per second were measured averaged over time intervals of five seconds. For measuring the primitive selection performance, a selection rectangle was drawn in the viewport and continuously changed. This enforces the selection computation being executed for every drawn frame. The performance of the meta-data selection methods was measured in the same way. Table 6.1 and figure 6.18 show the results obtained for the example set of cuneiform fragments in ascending size with respect to the vertex count. It can be seen that the rendering speed of the viewport navigation is still in an interactive range even for the large cuneiform fragment *VAT422 Vs*. Continuous selection actions on the respective mesh are no longer fluid, but still usable. This is mitigated by the fact that selection actions on cuneiform meshes are usually performed by clicking a polygon rather than by continuously click-dragging a rectangle. Since this involves clicking a sequence of polygon points and the selection is only updated when adding a point, the slower computation time is reduced by the time the user takes to click a new polygon point.

As can be seen in figure 6.18, all three performance curves have a similar shape. The significantly lower values for the selection actions can be explained with the necessary transform feedback for primitive selection and the required texture downloads via pixel buffer objects for meta-data selection. It is worth mentioning that both the viewport rendering and the selection methods are vertex-bound for larger fragment sizes. This can be verified by zooming out so that

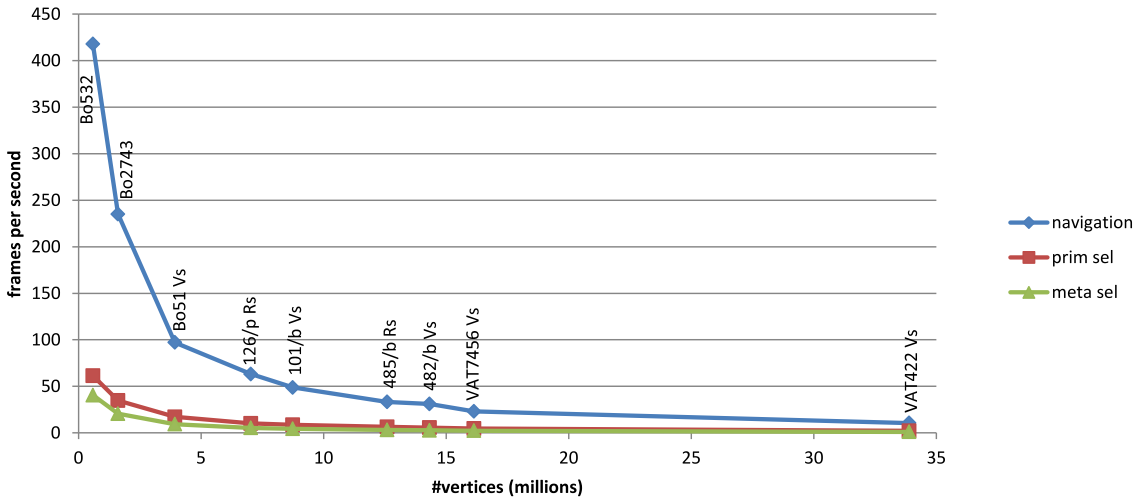
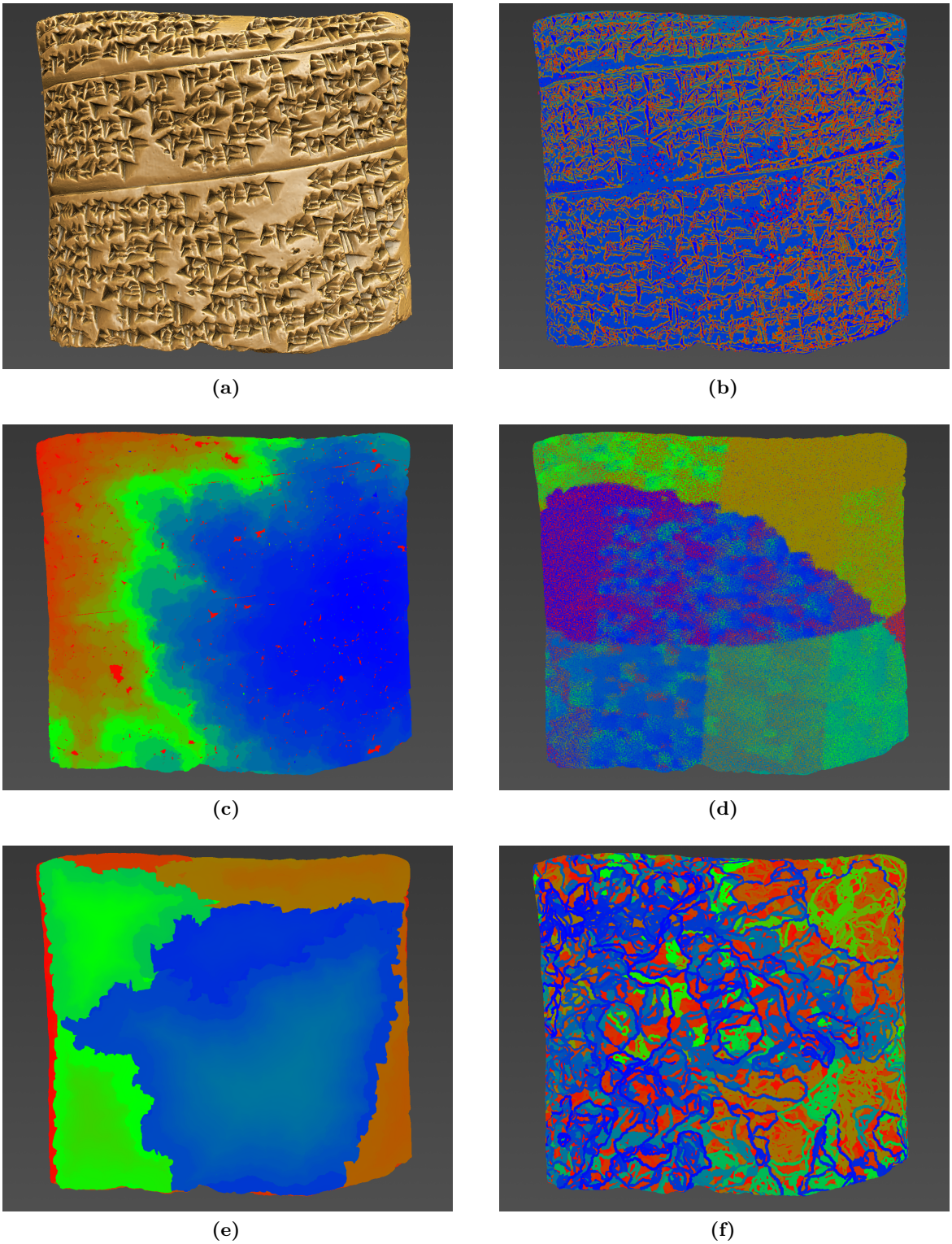


Figure 6.18: The mesh rendering performance for the fragments from figure 4.51. Measured are the average frames per second in the viewport for navigating the mesh, for continuous primitive selection and for continuous meta-data selection.

the fragments only cover a small area on the screen, which has no significant influence on the rendering speed for the larger fragments. A small improvement can only be observed for fragments below 3 million vertices since at this size the fragment operations still have a noticeable impact on the overall performance. This is partially caused by the deferred rendering that executes costly shading operations only once for every viewport pixel. Thus, the shading costs are limited by the usually constant pixel count of the viewport.

As already mentioned in section 6.3, optimizing the vertex order in the triangle index list to better utilize the vertex cache is a way to improve rendering speed. By employing a triangle colorization that maps the triangle indices in the mesh file onto a multi colored gradient ramp, the locality of the triangles can be visualized. In such a rendering, smooth color gradients correspond to good locality whereas color discontinuities correspond to bad locality. A typical locality distribution for a cuneiform scan assembled of multiple individual scans is shown in figure 6.19b. The scan of fragment *Bo71-222* with 3.4 million vertices exhibits many discontinuities at the inner and outer wedge edges and renders at about 101 frames per second. After an optimization with the AMD Tootle framework, which includes the Tipsy method from Sanders et al. [189], the mesh locality is significantly improved, as can be seen in figure 6.19c. The optimized mesh renders at about 220 frames per second. Aside from the scanned meshes, the Poisson reconstructed surface meshes from section 4.5.2 are also affected by unfavorably ordered triangles as can be seen in figure 6.19d. Here, the discontinuities show a different pattern that originates from the mesh construction in a spatial subdivision tree with box shaped elements. Optimizing this mesh, which consists of 3.3 million vertices and renders at only 50 frames per second, leads to the version shown in figure 6.19e. It exhibits several individually optimized subsections and renders at about 230 frames per second. The



(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 6.19: Vertex cache optimization on fragment *Bo71-222*. A shaded rendering is shown in **(a)**, followed by an unoptimized version in **(b)** and a Tootle-optimized version in **(c)**. **(d)** shows an unoptimized Poisson mesh and **(e)** and **(f)** optimized Poisson meshes using AMD Tootle and only the Tipsy method. Similar surface colors correspond to good triangle locality.

Renderer	FPS
CuneiformAnalyser	80
CuneiformAnalyser (+D3DXOptimizeFaces)	141
CuneiformAnalyser (+Tipsy)	200
MeshLab	89
3dsmax 2015 OpenGL	5
3dsmax 2015 DX3D-9	140
3dsmax 2015 DX3D-11	249

Table 6.2: Mesh rendering performance comparison for the CuneiformAnalyser, MeshLab and Autodesk 3dsmax 2015, measured in frames per second.

drawback of the complete Tootle optimization is the fact, that is can take several tenth of minutes or even hours to compute on large meshes. On the other hand, employing the Tipsy method takes only about one second for the meshes shown in figure 6.19. The resulting mesh in figure 6.19f looks like a composition of ant trails and renders at about 160 frames per second for the scanned mesh and 220 frames per second for the Poisson mesh. The largely unordered appearance results from the optimization goal to maximize vertex cache usage for a consecutive strip of triangles. This strip does not necessarily need to be packed densely in compact regions but can roam around freely on the mesh as long as the global fragmentation stays reasonably small. Due to the reasons mentioned in section 6.3, mesh optimization via the Tipsy method is only offered as an option in the CuneiformAnalyser. However, it can push the frame rates even for large fragments. For *VAT422 VS* an increase from 10.54 to 26.1 frames per second can be achieved. All frame rates in this section were measured on an Nvidia Geforce GTX 780.

For a comparison of the framework rendering performance with other mesh processing software the rendering and the selection performance were measured for the CuneiformAnalyser, MeshLab[4] and Autodesk 3dsmax 2015[5]. The measurements were conducted on the exemplary cuneiform fragment *44-a* with approximately 5 million vertices under the same conditions as the measurements in table 6.1. Results for the rendering performance are shown in table 6.2. Here, the CuneiformAnalyser shows a slightly slower rendering performance than MeshLab, which becomes more similar for larger meshes. This is due to the CuneiformAnalyser using more complex shading methods than MeshLab, while both renderers use indexed triangle rendering from vertex buffer objects. However, the rendering performance of 3dsmax 2015 is higher than the performance of the CuneiformAnalyser, both with the DirectX 9 and the DirectX 11 renderer. The reason for this lies in the rendering optimizations employed by 3dsmax. Since version 9, DirectX offers the `D3DXOptimizeFaces` method to apply online vertex cache optimizations to a rendered mesh. This comes with the characteristic behavior

[4] http://www.meshlab.net/
[5] https://www.autodesk.com/products/3ds-max/

Renderer	Mode	FPS
CuneiformAnalyser	points	24.0
CuneiformAnalyser	triangles	15.6
MeshLab	points	6.7
MeshLab	triangles	1.3
3dsmax 2015 DX3D-11	points	1.0
3dsmax 2015 DX3D-11	triangles	0.1

Table 6.3: Selection performance comparison for the CuneiformAnalyser, MeshLab and Autodesk 3dsmax 2015, in point-mode and triangle-mode, measured in frames per second.

that a rendered mesh is first rendered at a slower frame rate and then switches to a higher frame rate, once DirectX has applied the optimizations. This behavior can be observed in 3dsmax for both renderers. The fact that the DirectX 11 version obtains a higher frame rate suggests that additional optimizations are applied. To replicate these optimizations with the CuneiformAnalyser, the mesh was preprocessed using AMD Tootle to apply the optimization of the `D3DXOptimizeFaces` method manually to the mesh. This resulted in a frame rate increase to 140 frames per second, which is very similar to the results of the DirectX 9 renderer of 3dsmax. An even better value could be achieved by applying the Tipsy method in the CuneiformAnalyser before rendering. This could, however, still not reach the speed of 3dsmax' DirectX 12 renderer. The very slow results of the 3dsmax OpenGL renderer were just added for completeness. This renderer uses no modern acceleration techniques like vertex buffers and can therefore not handle large meshes.

Considering the selection performance, the evaluated candidates perform very differently. Since neither MeshLab nor 3dsmax 2015 offer meta-data selection methods similar to the CuneiformAnalyser, the selection performance is evaluated only for primitive selection in point-mode and in triangle-mode. The obtained results can be seen in table 6.3. It becomes apparent that both 3dsmax 2015 and MeshLab are not optimized for selecting primitives on large meshes. In contrast, the CuneiformAnalyser offers interactive frame rates in both modes. The superior selection performance comes, however, at the price of an increased memory consumption, which will be considered in the following subsection.

6.6.2 Memory consumption

As a result of the concurrent management of data structures for geoemetry processing and rendering, the memory consumption of the framework has to be considered on the CPU and the GPU side. On the GPU side, the required data components include a VBO for the vertices with positions, colors, and normals and a VBO for the triangle indices. In addition to that, a VBO for the triangle centers and transform feedback buffers and pixel buffer objects for the selection methods have to be allocated. Beyond this mesh specific data, textures for the framebuffer objects, required for deferred rendering have to be created. Since these last

Fragment	Vertices	GPU_{total}	GPU_{VBOs}	$GPU_{textures}$	CPU_{RAM}
Bo532	587,053	71,329	61,980	9,344	200,292
Bo2743	1,610,047	194,097	169,004	25,088	549,196
Bo51	3,938,454	474,182	412,993	61,184	1,381,864
126/p Rs	7,034,195	849,950	740,249	109,696	2,419,288
101/b Vs	8,742,711	1,051,304	915,875	135,424	3,002,720
485/b Rs	12,598,242	1,505,171	1,311,630	193,536	4,361,904
482/b Vs	14,328,507	1,723,773	1,501,688	222,080	5,096,480
VAT7456 Vs	16,141,130	1,943,318	1,692,945	250,368	5,558,516
VAT422 Vs	33,889,749	4,076,891	3,551,702	525,184	11,659,328

Table 6.4: The memory consumption in megabytes of the fragments from figure 4.51. Measured are the total memory consumption on the GPU and the respective fractions in terms of vertex buffer objects (VBOs), textures, and the memory consumption in CPU main memory.

components do not depend on the mesh size and usually take less than 200 MB of space, they are not included in the following measurements. On the CPU side, the data includes the half-edge data structure of OpenMesh with the included vertex, half-edge, edge and face lists. Aside from the required attributes, like vertex positions, the CuneiformAnalyser allocates a total of 21 vertex attributes and 9 face attributes. Among the vertex attributes are elements such as vertex colors, normals, curvatures, ambient occlusion, and the surface distance but also selection IDs and selection and visibility flags. The face attributes, on the other hand, include values such as triangle mid-points, selection and cluster IDs.

Table 6.4 shows an overview of the measured memory consumption on the GPU and the CPU side. For the GPU data structures, the memory occupied by VBOs and textures is listed separately. It can be seen that the framework uses comparatively high amounts of memory. This is a trade-off for gaining processing speed, which can be seen when looking at the achieved good segmentation performance in chapter 4 and the measured rendering performance in section 6.6.1. Taking into account that the largest fragment requires approximately 4 GB of GPU memory and 12 GB of main memory, it becomes apparent that even these mesh sizes can be handled by current mid- to upper-range gaming hardware. Even more so, considering that the large majority of cuneiform scans of the Hethitologie-Portal Mainz is much smaller than that.

6.6.3 GPU data update

As a last performance aspect, the speed of the GPU data update is of interest. Frequently updated data components include the vertex and face colors of the mesh that are updated every time the display mode is changed or a different set of attributes is been mapped to a primitive color attribute for display. In addition to that, several flags for the visibility and the selection status of triangles and vertices are frequently updated, following selection

actions in the viewport or the list-based visualizations. Since GPU data updates are most performant when the data to be updated is located in large continuous blocks of memory, the frequently updated data is stored in a compact form. The vertex buffer portion that holds the vertex colors in form of a vector of four component RGBA unsigned byte values is mixed with the visibility and selection flags, which are bit-masked into the unused alpha component. In contrast to the primitive colors, the transform feedback buffer and pixel buffer objects for receiving the selection data include no unused components. Measuring the GPU data upload and download performance in an environment that interleaves rendering and data transfer operations is a complex task. Since the additional components using the available bandwidth can often not be determined, it is not possible to give precise values. For the data upload and download operations of the framework, values of 817.64 ± 147.21 MB/s have been measured for the data upload and 1141.78 ± 123.32 MB/s for data download on the test system with an Intel Core-i7 4790K and an Nvidia Geforce GTX 780. An impression of usual update delays can be gained by considering the size of the buffer portion containing the color/flag data on a large fragment like *VAT422 Vs*. It has a size of approximately 467 MB and can thus be uploaded in about 0.6 seconds. Transferred to average sized cuneiform scans, this means that color updates are applied almost immediately.

6.7 Conclusion

This chapter presented a set of visualization concepts for a cuneiform segmentation framework. Included therein are a hierarchical concept for data handling and persistent storage that has proven its suitability in various application scenarios connected to cuneiform script analysis. The concept comprises a project-based management of cuneiform fragments with contained selections and meta-selections that carry the segmentation information. The flexible user interface concept, with its synchronized dual list-based and 3D visualization, can adapt to various work flows and accommodates different working habits of the users. Beyond this, the employed 3D visualization concepts for cuneiform fragments and the associated meta-data includes ambient occlusion, radiance scaling, and lit-sphere rendering to achieve a suitable combination of good depth perception, detail enhancement, and semi-realistic surface shading. As a result, the visualization not only addresses the two main use cases of fragment joining and cuneiform segmentation but also collation related tasks. The handling of meta-data is complemented by a set of fast, GPU-accelerated selection techniques for geometric primitives and meta-data, which facilitates the handling of segmentation results, manual segmentation tasks, and the handling of large meshes. In addition to that, a flexible and intuitive meta-data visualization provides layered access to the generated segmentation results, like the wedge model data. With respect to the evaluation of virtual cuneiform fragment joins the framework provides an intuitive and fast transformation system based on manipulator objects. The performance of the employed rendering techniques is shown to be competitive. This is particularly true for the geometry selection methods that are several times faster than

the competition. The performance advantage comes at the expense of a larger memory consumption, which, however, can still be handled well by mid-range gaming hardware.

CHAPTER 7

Discussion and future work

To emphasize the importance of the contributions of this work, this chapter will discuss their impact and benefits with respect to script feature analysis and the philological working methods and use cases presented in section 2.3. In addition to that, starting points for future improvements will be identified.

Geometric *script feature analysis*, as one of the main use cases targeted by this thesis, benefits largely from the developed methods. Being a sub-aspect of various methods for pursuing different philological research goals, the focus of script feature analysis also varies. Script features in traditional script feature analysis may be connected to the execution and composition of cuneiform sign types, content-based aspects regarding the contained text or the type and language of the used cuneiform script. If script analysis is restricted to a purely geometric level, traditional research methods are in many cases limited to qualitatively describing the appearance of a specific cuneiform script as bold, italic or as being filigree executed. Considering specific shape differences of cuneiform wedges was until now limited to large and eye-catching wedges and could not be quantified well. This was caused by a lack of appropriate measurement and automated processing methods to handle large amounts of wedges. Due to the small script and the delicate original artifacts, many measurements were previously taken using magnified photographic reproductions. Thus, the methods presented in this thesis provide for the first time quantitative access to three-dimensional wedge-level cuneiform script features. As shown by Michele Cammarosano [38], the extracted wedge models are also a promising starting point for automated statistical script analysis and can be used to identify different script types. For handling this new type of data, both automated and manual methods are supplied to accommodate established working habits in the field of cuneiform research. A new and central aspect of the extracted script features is their objectiveness in comparison to the previously used hand copies. Nevertheless, the new set of

methods can not replace but only complement the traditional philological research methods. In this respect, the creation of a digital 3D sign inventory could be a good extension to the traditional sign inventories to promote the accessibility of digital cuneiform representations with the goal of providing content-based retrieval methods. The use case of joining cuneiform fragments will be addressed separately in the following paragraphs. Since the segmentation methods in this thesis have been used in conjunction with a specific cuneiform scan acquisition process and only on Hittite cuneiform, future work should target their application on scans produced by other scanners and with other cuneiform variants. First evaluations in this direction on photogrammetric datasets from the KU Leuven and the Pergamon Museum Berlin containing non-Hittite cuneiform writing produced promising results. With respect to the statistical analysis of the extracted features, the novelty of this kind of metrological data implies that its utility value has so far only been insufficiently evaluated. In particular, the data potential with respect to sign-based retrieval tasks needs to be explored further.

The philological *collation* of cuneiform fragments benefits from the methods developed in this thesis in the sense that many collation tasks can now be performed on 3D scans. This is advantageous for various reasons. A first aspect is tied to the fact that the access to the 3D scans is much less restricted than the access to the original artifacts, both in terms of spatial and temporal availability. In contrast to the original artifact, a cuneiform scan can be accessed by multiple researchers at multiple locations at once. Additionally, the here presented segmentation framework offers various methods for improving the collation accuracy. This includes precise, destruction-free measurement methods for distances and depth that surpass the available manual methods and besides provide the option to store the actual measurement situation for later review. With respect to visual collation aids, the presented methods provide a flexible visual feature enhancement, variable lighting, the simulation of different surface materials and a suppression of the influence of surface textures on the visual recognition of features. Here, especially the capability of radiance scaling and ambient occlusion to enhance the features independently of light positions is an advantage. On the other hand, the texture-less surface rendering helps in cases where geometric details on the original artifact are unrecognizable due to structure rich surface textures, like burn marks. With respect to collaborative work, collation results can now be annotated directly on the 3D scan and shared with other researchers. This eliminates the need to map collation annotations on hand copies to the original artifact given the case that the collation goal is not the verification of a hand copy. To extend the collaborative features in the future, the framework could provide more segmentation independent annotation methods, like the annotation of point and polygonal line markers. A first step to expand the collaborative capabilities regarding collation has already been taken with the Cuneiform-WebGLViewer [70]. An ongoing issue of digital collation is the inability of a researcher to validate the accuracy of the underlying scan data especially in cases where the scan was taken and post-processed by another researcher. This is similar to the issue with the correctness of hand copies that always include aspects of subjective interpretation. However, the influence of subjective errors can be expected to be far less pronounced on 3D

scans than for the case of hand copies. Considering these factors, digital collation can improve many aspects of traditional collation and provide a significant gain in measurement accuracy regarding small script features. However, digital collation can still not replace the collation of the original artifact in all cases. The reason for this can be connected to inaccuracies during the capturing of the artifact, but can also originate from a poor match between the aspects taken into account during scanning and a current research question.

The methods presented in this thesis do not directly target the *dating of manuscripts*. As a result, this philological use case can only benefit from the methods developed in this thesis in the sense that some subtasks, such as collation, the comparison of script features or the determination of similar script characteristics, are improved. An attempt to specifically target the dating of manuscripts could be made by analyzing the extracted script features via machine learning methods. Given large enough training sets, with automatically extracted script features and manual dating annotations, a temporal classification of the fragments could be attempted.

Searching for instances of cuneiform related elements, like wedges, signs and text passages, is a very frequent sub-task for the processing of many philological research questions. Since the geometric feature extraction methods in this thesis are focused on wedges, the extraction results can be used to automatically search for specifically shaped wedges or wedges with derived properties, like the wedge type, a minimum size or a specific direction. Although this does not cover finding instances of cuneiform signs and text passages, highlighting wedge-level retrieval results can significantly lower the time to locate occurrences of cuneiform signs on larger documents. A basic extension to the sign level is achieved by local wedge neighborhood descriptors that manage a list with relative locations of surrounding wedges for each wedge. This makes finding multiple stacked wedges of the same type particularly easy and facilitates locating signs that contain these wedge combinations. Since the generated segmentation data in the XML files can also be searched without loading the actual 3D scans, a cross-document search can be performed. Aside from the segmentation data, manual annotations on wedges and signs can be used for locating respective instances. However, this option is only available on previously annotated datasets. For future work, the retrieval methods could be extended to combine the automatically extracted wedge features to cuneiform sign features that enable searching for sign instances. A step in this direction has already been performed in cooperation with Kriege et al. [117] using graph-based methods and with Rothacker et al. [180] in the two-dimensional domain.

For the *identification of scribes*, framework methods for collation, metrology and statistical evaluation of script features can be employed. However, the extracted wedge models can only be used to a limited extent for scribe identification. This is due to the fact that scribe characteristics are usually not captured by the abstract wedge model but may instead be encoded in subtle wedge details or sign compositions. In addition to that, scribe characteristics are overlain, for example, by characteristic features for writing schools or regional and temporal

script peculiarities and are generally hard to identify. The developed methods can therefore only somewhat extend a philologists set of available methods for identifying and quantifying respective features.

For the use case of *manuscript reconstruction*, the most important contribution of this thesis is the automated extraction of the wedge-based script features. The extraction method is robust against mesh errors and can handle significantly damaged wedge geometries, which guarantees its applicability to a wide range of 3D cuneiform scans. As shown by Michele Cammarosano [38], these features can be statistically analyzed to extract plausible groups of cuneiform fragments with similar script properties. These groups can be used to significantly lower the amount of promising join candidates. A high script similarity or dissimilarity can also be used to support or disprove existing join proposals. In the case of a promising join candidate identification, the framework provides fast and intuitive methods for conducting and verifying a virtual join. This is particularly interesting for large two-sided manuscripts that consist of many fragments on the obverse and reverse side. These fragments do not necessarily share common fracture faces but are often aligned using extrapolated paragraph or columns separators. Since fragment proportions and positions are often distorted in traditional join layout sketches, matching the actual geometry can reveal positional inaccuracies and non-fitting fracture faces for inner sections of the manuscript which are not located near the tablet surface. Completed virtual joins can then be stored as a set of fragment transformations via the project files of the CuneiformAnalyser, which can be easily shared without computing fused mesh files. Here, the framework's capability to handle large meshes comes in handy for working with multi-component joins. For future work, the extracted script features could be integrated into a database, to enable a more systematic search for join candidates. In addition to that, the manual joining methods could be complemented by (semi-)automatic registration methods that use the fracture faces or script features like line patterns and column-/paragraph-separators for alignment. On a higher level, the locations of multiple fragments of a manuscript could be optimized simultaneously to find a globally optimal fragment alignment. A promising but complex method in this regard has been presented by Huang et al. [96].

With respect to cuneiform *stylus shape reconstruction*, the methods developed in this thesis can contribute precise measurement tools that are able to quantify features on wedge- and sub-wedge-scale. Since a stylus may have been rotated or shifted during impression, the measured wedge shapes can not be assigned with certainty to a certain stylus shape. Instead, considering the combined shapes of many wedges could be used to derive some restrictions to a possible stylus shape. As an alternative approach, like performed by Michele Cammarosano, the here presented wedge extraction can be used to determine the wedge angles caused by a variety of self constructed styli [39] for comparison with the ancient originals. On the visualization side, the visual enhancements of radiance scaling make extremely subtle features like impressed reed structures visible. This can be used in some cases to track, which sides of a stylus created which wedge faces to gain insights into the writing process. Without the automated wedge extraction, these comparisons could only be made on a visual level because

manual measurements of wedge angles on 3D scans are less precise than the automatic wedge segmentation.

The *identification of incomplete signs* can be important in cases where parts of cuneiform signs have been severely affected by surface erosion. However, some traces of the wedges on an abraded surface may have been conserved. By considering the surface distance from section 4.5, the depth of indentations can be compared to typical wedge depth on the same document as a plausibility check. In addition to that, constellations of extracted wedge models from other instances of the same sign within the manuscript could be aligned with the incomplete sign to check if the wedge positions match the indentations in position and depth.

Transliteration and transcription is rudimentary supported by the developed framework concept via the collation related methods and the possibility to annotate the extracted wedges with transliteration data. However, annotating the digital 3D representation is less flexible and more time-consuming than working with a self drawn or printed hand copy or a printed photographic representation. Considering this, a stylized rendering of the maximum curvature can be used to generate a representation with a similar appearance like hand copies. This representation can then be printed out and annotated with a pen to accommodate usual working habits. It can also serve as a base for creating a hand copy in the form of a digital vector drawing. An automated approach for the creation of vector drawings from 3D cuneiform scans has already been presented by Mara et al. [138]. In addition to the curvature-based representation, the extracted wedge models in terms of the three inner edges and the skeletonization results can also be used as a preliminary stage for creating hand copies.

References

[1] 3D Systems Inc.: *StereoLithography Interface Specification*. 1988.

[2] R. Abraham, J. E. Marsden, and R. Ratiu: *Manifolds, Tensor Analysis, and Applications: 2Nd Edition*. Springer-Verlag, 1988.

[3] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang: "Near-Linear Time Approximation Algorithms for Curve Simplification". In: *Proceedings of the 10th Annual European Symposium on Algorithms*. ESA '02. Springer-Verlag, 2002, pp. 29–41.

[4] Akademie der Wissenschaften und Literatur, Mainz and Julius-Maximilians-Universität, Würzburg: *Hethitologie Portal Mainz*. 2000. URL: `http://www.hethiter.net/` (visited on 11/16/2018).

[5] B. Allen, B. Curless, and Z. Popović: "The Space of Human Body Shapes: Reconstruction and Parameterization from Range Scans". In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. ACM, 2003, pp. 587–594.

[6] N. Amenta, S. Choi, and R. K. Kolluri: "The Power Crust". In: *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*. SMA '01. ACM, 2001, pp. 249–266.

[7] S. Anderson and M. Levoy: "Unwrapping and Visualizing Cuneiform Tablets". In: *IEEE Computer Graphics and Applications* 22.6 (2002), pp. 82–88.

[8] P. R. Andresen and M. Nielsen: "Non-rigid registration by geometry-constrained diffusion". In: *Medical Image Analysis* 5.2 (2001), pp. 81–88.

[9] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander: "OPTICS: Ordering Points To Identify the Clustering Structure". In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pp. 49–60.

[10] L. Arge, J. S. Chase, P. Halpin, L. Toma, et al.: "Efficient Flow Computation on Massive Grid Terrain Datasets". In: *GeoInformatica* 7.4 (2003), pp. 283–313.

[11] D. Arthur and S. Vassilvitskii: "K-means++: The Advantages of Careful Seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[12] M. Attene: "A lightweight approach to repairing digitized polygon meshes". In: *The Visual Computer* 26.11 (2010), pp. 1393–1406.

[13] P. F. Bang and W. Scheidel: *The Oxford Handbook of the State in the Ancient Near East and Mediterranean*. Oxford Authors Series. OUP USA, 2013.

[14] R. Barnes, C. Lehman, and D. Mulla: "Priority-Flood: An Optimal Depression-Filling and Watershed-Labeling Algorithm for Digital Elevation Models". In: *CoRR* abs/1511.04463 (2015).

[15] B. G. Baumgart: "A Polyhedron Representation for Computer Vision". In: *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*. AFIPS '75. ACM, 1975, pp. 589–596.

[16] J. L. Bentley: "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM* 18.9 (1975), pp. 509–517.

[17] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, et al.: "A Survey of Surface Reconstruction from Point Clouds". In: *Computer Graphics Forum* (2016).

[18] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, et al.: "State of the Art in Surface Reconstruction from Point Clouds". In: *Eurographics 2014 - State of the Art Reports*. Ed. by S. Lefebvre and M. Spagnuolo. The Eurographics Association, 2014.

[19] P. J. Besl and N. D. McKay: "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256.

[20] S. Beucher and C. Lantuéjoul: "Use of Watersheds in Contour Detection". In: *International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation, Rennes, France*. 1979.

[21] W. Bieniecki: "Oversegmentation avoidance in watershed-based algorithms for color images". In: *Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004. Proceedings of the International Conference*. 2004, pp. 169–172.

[22] C. M. Bishop: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Corr. at 8. print. Information science and statistics. Springer-Verlag New York, Inc and Springer, 2006.

[23] J. Blanchette and M. Summerfield: *C++ GUI Programming with Qt 4*. Prentice Hall PTR, 2006.

[24] H. Blum and W. Wathen-Dunn: "A Transformation for Extracting New Descriptors of Shape". In: *Models for the Perception of Speech and Visual Form* (1967), pp. 362–380.

[25] B. Bogacz, N. Howe, and H. Mara: "Segmentation Free Spotting of Cuneiform using Part Structured Models". In: *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)* (2016).

[26] B. Bogacz, M. Klingmann, and H. Mara: "Automating Transliteration of Cuneiform from Parallel Lines with Sparse Data". In: *IAPR International Conference 2017*. Vol. 01, pp. 615–620.

[27] B. Bogacz and H. Mara: "Clustering Fundamental Spatial n-Grams for Large Scale Cuneiform Search". In: *Proc. of the 12th International Workshop on Document Analysis Systems (DAS)* (2016).

[28] B. Bogacz, M. Gertz, and H. Mara: "Character retrieval of vectorized cuneiform script". In: *ICDAR 2015, Nancy, France, August 23-26, 2015*, pp. 326–330.

[29] B. Bogacz, M. Gertz, and H. Mara, eds.: *Cuneiform Character Similarity Using Graph Representations*. Proc. of the 20th Computer Vision Winter Workshop (CVWW'15). 2015.

[30] B. Bogacz, J. Massa, and H. Mara: "Homogenization of 2D and 3D Document Formats for Cuneiform Script Analysis". In: *Proceedings of the 3rd International Workshop on Historical Document Imaging and Processing*. HIP '15. ACM, 2015, pp. 115–122.

[31] R. Borger and F. Ellermeier: *Assyrisch-babylonische Zeichenliste*. Alter Orient und Altes Testament. Butzon & Bercker, 1981.

[32] M. Botsch and L. Kobbelt: "A Remeshing Approach to Multiresolution Modeling". In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. SGP '04. ACM, 2004, pp. 185–192.

[33] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt: "OpenMesh: A Generic and Efficient Polygon Mesh Data Structure". In: *OpenSG Symposium 2002*.

[34] P. S. Bradley, O. L. Mangasarian, and W. N. Street: "Clustering via Concave Minimization". In: *Advances in Neural Information Processing Systems -9*. MIT Press, 1997, pp. 368–374.

[35] E. Brisson: "Representing Geometric Structures in D Dimensions: Topology and Order". In: *Proceedings of the Fifth Annual Symposium on Computational Geometry*. SCG '89. ACM, 1989, pp. 218–227.

[36] A. M. Bronstein, M. M. Bronstein, and M. Ovsjanikov: "Feature-based methods in 3D shape analysis". In: *3D Imaging, Analysis and Applications*. Springer, 2012, pp. 185–219.

[37] M. Bunnell: "Dynamic ambient occlusion and indirect lighting". In: *GPU Gems* 2 (2005), pp. 223–233.

[38] M. Cammarosano: "3D-Joins und Schriftmetrologie: A Quantitative Approach to Cuneiform Palaeography". In: *Current Research in Cuneiform Paleography. Proc. of a Workshop held at the 60th Rencontre Assyriologique Internationale*. University of Warsaw, 2014.

[39] M. Cammarosano: "The Cuneiform Stylus". In: *Mesopotamia XLIX* (2014), pp. 53–90.

[40] M. Cammarosano, G. G. W. Müller, D. Fisseler, and F. Weichert: "Schriftmetrologie des Keils: Dreidimensionale Analyse von Keileindrücken und Handschriften". In: *Die Welt des Orients* 44.1 (2014), pp. 2–36.

[41] S. Campagna, L. Kobbelt, and H.-P. Seidel: "Directed Edges - A Scalable Representation for Triangle Meshes". In: *J. Graph. Tools* 3.4 (1998), pp. 1–11.

[42] M. Campen, M. Attene, and L. Kobbelt: "A Practical Guide to Polygon Mesh Repairing". In: *Eurographics 2012 - Tutorials*. Ed. by Renato Pajarola and Michela Spagnuolo. The Eurographics Association, 2012.

[43] Y.-P. Cao, T. Ju, J. Xu, and S.-M. Hu: "Extracting Sharp Features from RGB-D Images". In: *Computer Graphics Forum* 36.8 (2017), pp. 138–152.

[44] R. I. Caplice and D. C. Snell: *Introduction to Akkadian*. Pubblicazioni Della Classe Di Lettere E Filosofia, Scuola No. Biblical Institute Press, 1988.

[45] *CDLI - Cuneiform Digital Library Initiative*. URL: `http://cdli.ucla.edu/` (visited on 11/16/2018).

[46] CGAFaq: *Evenly distributed points on sphere*. 2012. URL: `https://web.archive.org/web/20120321172612/http://www.cgafaq.info/wiki/Evenly_distributed_points_on_sphere` (visited on 11/16/2018).

[47] D. Charpin: *Lire et écrire à Babylone*. Presses Universitaires de France, 2008.

[48] D. Cohen-Steiner and F. Da: "A Greedy Delaunay-based Surface Reconstruction Algorithm". In: *The Visual Computer* 20.1 (2004), pp. 4–16.

[49] M. F. Cohen, J. Wallace, and P. Hanrahan: *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Inc, 1993.

[50] T. Collins, S. I. Woolley, L. Hernandez Munoz, A. Lewis, et al.: "Computer-Assisted reconstruction of virtual fragmented cuneiform tablets". In: *Virtual Systems & Multimedia (VSMM), 2014 International Conference on*. 2014, pp. 70–77.

[51] W. T. Corrêa, J. T. Klosowski, and C. T. Silva: "iWalk: Interactive out-of-core rendering of large models". In: *Princeton University - Technical Report TR-653-02*.

[52] C. Cortes and V. Vapnik: "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297.

[53] L. Dagum and R. Menon: "OpenMP: An Industry-Standard API for Shared-Memory Programming". In: *IEEE Comput. Sci. Eng.* 5.1 (1998), pp. 46–55.

[54] J. Davis, S. R. Marschner, M. Garr, and M. Levoy: "Filling Holes in Complex Surfaces Using Volumetric Diffusion". In: *Proceedings of the 1st International Symposium on 3D Data Processing, Visualization and Transmission*. 2001, p 428–438.

[55] D. Defays: "An Efficient Algorithm for a Complete Link Method". In: *The Computer Journal* 20.4 (1977), pp. 364–366.

[56] N. Demoli, G. K. Wernicke, S. Krueger, and H. Gruber: "Holographic approach to conserving and analyzing the cuneiform inscription". In: *Proc. SPIE 4402, Laser Techniques and Systems in Art Conservation*. Vol. 4402.

[57] A. P. Dempster, N. M. Laird, and D. B. Rubin: "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38.

[58] D. H. Douglas and T. K. Peucker: "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2 (1973), pp. 112–122.

[59] G. Earl, K. Martinez, and H. Pagi: "Reflectance Transformation Imaging (RTI) System for Ancient Documentary Artefacts". In: *Electronic Visualisation and the Arts (EVA 2011)* 01 (2010).

[60] C. Ericson: *Real-time collision detection*. Morgan Kaufmann series in interactive 3D technology. Elsevier Morgan Kaufmann, 2005.

[61] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu: "A density-based algorithm for discovering clusters in large spatial databases with noise". In: AAAI Press, 1996, pp. 226–231.

[62] R. Estkowski, J. S. B. Mitchell, and X. Xiang: "Optimal Decomposition of Polygonal Models into Triangle Strips". In: *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*. SCG '02. ACM, 2002, pp. 254–263.

[63] A. Fabri: "CGAL - The Computational Geometry Algorithm Library". In: *Sandia National Laboratory*. 2001, pp. 137–142.

[64] Factum Arte: *3D Scanning for Cultural Heritage Conservation: A quick guide*. 2015. (Visited on 11/16/2018).

[65] H. Fadaifard, G. Wolberg, and R. Haralick: "Multiscale 3D feature extraction and matching with an application to 3D face recognition". In: *Graphical Models* 75.4 (2013), pp. 157–176.

[66] A. Falkenstein: *Archaische Texte aus Uruk*. Archaische Texte aus Uruk. Dt. Forschungsgemeinschaft, 1936.

[67] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller: "SplineCNN: Fast Geometric Deep Learning With Continuous B-Spline Kernels". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*. IEEE Computer Society, 2018, pp. 869–877.

[68] H. H. Figulla and E. F. Weidener: *Keilschrifttexte aus Boghazköi*. Keilschrifttexte aus Boghazköi. J.C. Hinrichs'sche Buchhandlung, 1970.

[69] M. A. Fischler and R. C. Bolles: "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (1981), pp. 381–395.

[70] D. Fisseler, G. G. W. Müller, and F. Weichert: "Web-Based Scientific Exploration and Analysis of 3D Scanned Cuneiform Datasets for Collaborative Research". In: *Informatics* 4.4 (2017).

[71] D. Fisseler, F. Weichert, M. Cammarosano, and G. G. W. Müller: "Towards an interactive and automated script feature analysis of 3D scanned cuneiform tablets". In: *Scientific Computing and Cultural Heritage (SCCH 2013)*. 2013.

[72] D. Fisseler, F. Weichert, G. G. W. Müller, and M. Cammarosano: "Extending Philological Research with Methods of 3D Computer Graphics Applied to Analysis of Cultural Heritage". In: *Eurographics Workshop on Graphics and Cultural Heritage*. The Eurographics Association, 2014, pp. 165–172.

[73] T. Forsyth: *Linear-Speed Vertex Cache Optimisation: RAD Game Tools*. 2006. URL: `https://tomforsyth1000.github.io/papers/fast_vert_cache_opt.html` (visited on 11/16/2018).

[74] C. Fraley and A. E. Raftery: "How many clusters? Which clustering method? Answers via model-based cluster analysis". In: *The Computer Journal* 41.8 (1998), pp. 578–588.

[75] J. Friedrich: *Hethitisches Elementarbuch von Johannes Friedrich*. Hethitisches Elementarbuch von Johannes Friedrich. C. Winter, 1960.

[76] J. Friedrich: *Hethitisches Wörterbuch: Kurzegefasste kritische Sammlung der Deutungen hethitischer Wörter*. Hethitisches Wörterbuch: kurzgefasste kritische Sammlung der Deutungen hethitischer Wörter. Carl Winter, 1952.

[77] M. R. Garey and D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, 1979.

[78] M. Geller: "The Last Wedge". In: *Zeitschrift für Assyriologie* 87 (1997), pp. 43–95.

[79] D. Gilbarg and N. S. Trudinger: *Elliptic partial differential equations of second order*. Grundlehren der mathematischen Wissenschaften. Springer, 1998.

[80] A. S. Glassner: *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc, 1994.

[81] E. Grilli, F. Menna, and F. Remondino: "A Review of Point Clouds Segmentation and Classification Algorithms". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W3 (2017), pp. 339–344.

[82] G. Guennebaud, M. Germann, and M. Gross: "Dynamic Sampling and Rendering of Algebraic Point Set Surfaces". In: *Comput. Graph. Forum* 27.2 (2008), pp. 653–662.

[83] M. Gupta, A. Agrawal, A. Veeraraghavan, and S. G. Narasimhan: "Structured light 3D scanning in the presence of global illumination". In: *CVPR 2011*. 2011, pp. 713–720.

[84] M. Guthe, P. Borodin, and R. Klein: "Fast and Accurate Hausdorff Distance Calculation between Meshes". In: *Journal of WSCG* 13.2 (2005), pp. 41–48.

[85] D. V. Hahn, K. C. Baldwin, and D. D. Duncan: "Non-laser-based scanner for three-dimensional digitization of historical artifacts". In: *Appl. Opt.* 46.15 (2007), pp. 2838–2850.

[86] R. Hänsch, T. Weber, and O. Hellwich: "Comparison of 3D interest point detectors and descriptors for point cloud fusion". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-3.September (2014), pp. 57–64.

[87] J. A. Hartigan and M. A. Wong: "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 100–108.

[88] T. Hastie, R. Tibshirani, and J. Friedman: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York, 2013.

[89] L. He, C. Y. Han, B. Everding, and W. G. Wee: "Graph matching for object recognition and recovery". In: *Pattern Recognition* 37.7 (2004), pp. 1557–1560.

[90] J. Hershberger and J. Snoeyink: "An O(n log n) Implementation of the Douglas-Peucker Algorithm for Line Simplification Stony Brook, New York, USA, June 6-8, 1994". In: (1994), pp. 383–384.

[91] M. Hisada, A. Belyaev, and T. L. Kunii: "A Skeleton-based Approach for Detection of Perceptually Salient Features on Polygonal Surfaces". In: *Computer Graphics Forum* 21.4 (2002), pp. 1–12.

[92] D. D. Hoffman and W. Richards: "Parts Of Recognition". In: *COGNITION* 18 (1983), pp. 65–96.

[93] H. Hoppe: "Optimization of Mesh Locality for Transparent Vertex Caching". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co, 1999, pp. 269–276.

[94] P. Hough: "Method and Means for Recognizing Complex Patterns". U.S. Patent 3.069.654. 1962.

[95] S. D. Houston: *The shape of script: How and why writing systems change / edited by Stephen D. Houston*. 1st ed. School for Advanced Research Press Santa Fe, N.M, 2012.

[96] Q.-X. Huang, S. Flöry, N. Gelfand, M. Hofer, and H. Pottmann: "Reassembling Fractured Objects by Geometric Matching". In: *ACM SIGGRAPH 2006 Papers*. SIGGRAPH '06. ACM, 2006, pp. 569–578.

[97] M. Hughes: *KMeansRex: Fast C++ implementation of K-Means*. URL: `https://github.com/michaelchughes/KMeansRex` (visited on 11/16/2018).

[98] "IEEE Standard for Binary Floating-Point Arithmetic". In: *ANSI/IEEE Std 754-1985* (1985).

[99] A. K. Jain and R. C. Dubes: *Algorithms for Clustering Data*. Prentice-Hall, Inc, 1988.

[100] D. M. Jones, ed.: *3D Laser Scanning for Heritage: Advice and guidance to users on laser scanning in archaeology and architecture*. English Heritage Publishing, 2011.

[101] M. W. Jones, J. A. Baerentzen, and M. Sramek: "3D Distance Fields: A Survey of Techniques and Applications". In: *IEEE Transactions on Visualization and Computer Graphics* 12.4 (2006), pp. 581–599.

[102] E. Kalogerakis, A. Hertzmann, and K. Singh: "Learning 3D Mesh Segmentation and Labeling". In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH '10. ACM, 2010, 102:1–102:12.

[103] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, et al.: "A Local Search Approximation Algorithm for K-means Clustering". In: *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*. SCG '02. ACM, 2002, pp. 10–18.

[104] R. M. Karp: "On-Line Algorithms Versus Off-Line Algorithms: How Much is It Worth to Know the Future?" In: *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1 - Volume I*. North-Holland Publishing Co, 1992, pp. 416–429.

[105] M. Kazhdan, M. Bolitho, and H. Hoppe: "Poisson Surface Reconstruction". In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. Ed. by A. Sheffer. SGP '06. Eurographics Association, 2006, pp. 61–70.

[106] M. Kazhdan and H. Hoppe: "Screened Poisson Surface Reconstruction". In: *ACM Trans. Graph.* 32.3 (2013), 29:1–29:13.

[107] B. Kemen: *Outerra: Maximizing Depth Buffer Range and Precision*. 2012. URL: `http://outerra.blogspot.de/2012/11/maximizing-depth-buffer-range-and.html` (visited on 11/16/2018).

[108] Khronos Group: *OpenGL® 4.5 Reference Pages*. URL: `https://www.khronos.org/registry/OpenGL-Refpages/gl4/` (visited on 11/16/2018).

[109] Khronos Group: *OpenGL Extension Registry*. URL: `https://www.khronos.org/registry/OpenGL/extensions/` (visited on 11/16/2018).

[110] Khronos Group: *OpenGL Wiki: Framebuffer Object*. URL: `https://www.khronos.org/opengl/wiki/Framebuffer_Object` (visited on 11/16/2018).

[111] Khronos Group: *OpenGL Wiki: Pixel Buffer Object*. URL: `https://www.khronos.org/opengl/wiki/Pixel_Buffer_Object` (visited on 11/16/2018).

[112] Khronos Group: *OpenGL Wiki: StencilBuffer*. URL: `https://www.khronos.org/opengl/wiki/Stencil_Buffer` (visited on 11/16/2018).

[113] Khronos Group: *OpenGL Wiki: Transform Feedback*. URL: `https://www.khronos.org/opengl/wiki/Transform_Feedback` (visited on 11/16/2018).

[114] T. N. Kipf and M. Welling: "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR* abs/1609.02907 (2016).

[115] J. Kontkanen and S. Laine: "Ambient Occlusion Fields". In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D '05. ACM, 2005, pp. 41–48.

[116] K. Kraus, I. A. Harley, and S. Kyle: *Photogrammetry: Geometry from Images and Laser Scans*. De Gruyter Textbook. De Gruyter, 2011.

[117] N. M. Kriege, M. Fey, D. Fisseler, P. Mutzel, and F. Weichert: "Recognizing Cuneiform Signs Using Graph Based Methods". In: *CoRR* abs/1802.05908 (2018).

[118] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek: "Density-based clustering". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.3 (2011), pp. 231–240.

[119] J. B. Kruskal: "On the shortest spanning subtree and the traveling salesman problem". In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50.

[120] S. Kumar, D. Snyder, D. Duncan, J. Cohen, and J. Cooper: "Digital preservation of ancient cuneiform tablets using 3D-scanning". In: *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings*. 2003, pp. 326–333.

[121] J. M. Lee: *Introduction to Topological Manifolds*. Graduate texts in mathematics. Springer, 2000.

[122] S. Lefebvre and H. Hoppe: "Perfect Spatial Hashing". In: *ACM Trans. Graph.* 25.3 (2006), pp. 579–588.

[123] Lei He, Chia Yung Han, and William G. Wee: "Object Recognition and Recovery by Skeleton Graph Matching and Expo, ICME 2006, July 9-12 2006, Toronto, Ontario, Canada". In: *Proceedings of the 2006 IEEE International Conference on Multimedia and Expo, ICME 2006, July 9-12 2006, Toronto, Ontario, Canada*. IEEE Computer Society, 2006, pp. 993–996.

[124] J. Levallois, D. Coeurjolly, and J.-O. Lachaud: "Scale-space Feature Extraction on Digital Surfaces". In: *Computers and Graphics* (2015), p. 12.

[125] D. Levin: "The Approximation Power of Moving Least-squares". In: *Math. Comput.* 67.224 (1998), pp. 1517–1531.

[126] L. Liang, M. Wei, A. Szymczak, A. Petrella, et al.: "Nonrigid iterative closest points for registration of 3D biomedical surfaces". In: *Optics and Lasers in Engineering* 100 (2018), pp. 141–154.

[127] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou: "Autoencoder for words". In: *Neurocomputing* 139 (2014), pp. 84–96.

[128] S. Lloyd: "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[129] W. E. Lorensen and H. E. Cline: "Marching cubes: A high resolution 3D surface construction algorithm". In: *ACM SIGGRAPH*. Vol. 21. 1987, pp. 163–169.

[130] D. G. Lowe: "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision* 60.2 (2004), pp. 91–110.

[131] W.-C. Ma, F.-C. Wu, and M. Ouhyoung: "Skeleton extraction of 3D objects with radial basis functions". In: *Shape Modeling International 2003*, pp. 207–215.

[132] C. Maes, T. Fabry, J. Keustermans, D. Smeets, et al.: "Feature detection on 3D face surfaces for pose normalisation and recognition". In: *BTAS*. IEEE, 2010, pp. 1–6.

[133] T. Malzbender, D. Gelb, H. Wolters, and B. Zuckerman: *Enhancement of Shape Perception by Surface Reflectance Transformation*. 2004.

[134] A. P. Mangan and R. T. Whitaker: "Partitioning 3D Surface Meshes Using Watershed Segmentation". In: *IEEE Transactions on Visualization and Computer Graphics* 5.4 (1999), pp. 308–321.

[135] M. Mäntylä: *An Introduction to Solid Modeling*. Computer Science Press, Inc, 1987.

[136] H. Mara: "Multi-scale Integral Invariants for Robust Character Extraction from Irregular Polygon Mesh Data". PhD thesis. Universität Heidelberg, 2012.

[137] H. Mara and S. Kromker: "Vectorization of 3D-Characters by Integral Invariant Filtering of High-Resolution Triangular Meshes". In: *2013 12th International Conference on Document Analysis and Recognition*, pp. 62–66.

[138] H. Mara, S. Krömker, S. Jakob, and B. Breuckmann: "GigaMesh and Gilgamesh - 3D Multiscale Integral Invariant Cuneiform Character Extraction". In: *VAST*. 2010, pp. 131–138.

[139] D. Martinez, L. Velho, and P. C. Carvalho: "Computing Geodesics on Triangular Meshes". In: *Comput. Graph.* 29.5 (2005), pp. 667–675.

[140] J. Massa, B. Bogacz, S. Krömker, and H. Mara: "Cuneiform Detection in Vectorized Raster Images". In: *CVWW* (2016).

[141] A. Meijster and Jos B. T. M. Roerdink: "A disjoint set algorithm for the watershed transform". In: *Proc. IX European Signal Processing Conference (EUSIPCO'98), September 8 - 11, 1998, Rhodes, Greece*. Ed. by S. Theodoridis, I. Pitas, A. Stouraitis, and N. Kalouptsidis. Typorama Publications, 1998, pp. 1665–1668.

[142] N. Mellado: "Analysis of 3D objects at multiple scales: application to shape matching". PhD thesis. Université Sciences et Technologies - Bordeaux I, 2012.

[143] N. Mellado, G. Guennebaud, P. Barla, P. Reuter, and C. Schlick: "Growing Least Squares for the Analysis of Manifolds in Scale-Space". In: *Computer Graphics Forum* 31.5 (2012), pp. 1691–1701.

[144] F. Meyer: "Topographic Distance and Watershed Lines". In: *Signal Process* 38.1 (1994), pp. 113–125.

[145] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr: "Discrete Differential Geometry Operators for Triangulated 2-Manifolds". In: *International Workshop on Visualization and Mathematics*. 2002.

[146] M. Mittring: "Finding Next Gen: CryEngine 2". In: *ACM SIGGRAPH 2007 Courses*. SIGGRAPH '07. ACM, 2007, pp. 97–121.

[147] D. M. Mount: *KMlocal: A Testbed for KMlocal: A Testbed for k-means Clustering Algorithms*. URL: `https://www.cs.umd.edu/~mount/Projects/KMeans/` (visited on 11/16/2018).

[148] M. Mudge, C. Schroer, G. Earl, K. Martinez, et al.: "Principles and Practices of Robust, Photography-based Digital Imaging Techniques for Museums". In: *VAST: International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage - Short and Project Papers*. Ed. by A. Artusi, M. Joly, G. Lucet, D. Pitzalis, and A. Ribes. The Eurographics Association, 2010.

[149] D. Müllner: "fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python". In: *Journal of Statistical Software* 53.9 (2013), pp. 1–18.

[150] *Musée du Louvre cuneiform collection at CDLI: Cuneiform Digital Library Initiative*. URL: `https://cdli.ucla.edu/collections/louvre/louvre_fr.html` (visited on 11/16/2018).

[151] D. Nehab, J. Barczak, and P. V. Sander: "Triangle Order Optimization for Graphics Hardware Computation Culling". In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. I3D '06. ACM, 2006, pp. 207–211.

[152] H.-G. Nesselrath: *Einleitung in die griechische Philologie*. Vieweg+Teubner Verlag, 2014.

[153] M. Niepert, M. Ahmed, and K. Kutzkov: "Learning Convolutional Neural Networks for Graphs". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by M. F. Balcan and K. Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. PMLR, 2016, pp. 2014–2023.

[154] S. O'Hara and B. A. Draper: "Introduction to the Bag of Features Paradigm for Image Classification and Retrieval". In: *CoRR* abs/1101.3354 (2011).

[155] D. L. Page, A. Koschan, and M. A. Abidi, eds.: *Perception-based 3D Triangle Mesh Segmentation Using Fast Marching Watersheds*. 2003.

[156] M. Pharr, S. Green, and R. Fernando: "Ambient Occlusion". In: *GPU Gems* (2004), pp. 279–292.

[157] O. Planchon and F. Darboux: "A fast, simple and versatile algorithm to fill the depressions of digital elevation models". In: *CATENA* 46.2 (2002), pp. 159–176.

[158] F. Ponchio and M. Dellepiane: "Fast decompression for web-based view-dependent 3D rendering". In: *Web3D 2015. Proceedings of the 20th International Conference on 3D Web Technology*. ACM, 2015, pp. 199–207.

[159] F. Ponchio and M. Dellepiane: "Multiresolution and fast decompression for optimal web-based rendering". In: *Graphical Models* 88 (2016), pp. 1–11.

[160] M. Pontil and A. Verri: "Support Vector Machines for 3D Object Recognition". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 20.6 (1998), pp. 637–646.

[161] M. Potenziani, M. Callieri, M. Dellepiane, M. Corsini, et al.: "3DHOP: 3D Heritage Online Presenter". In: *Computers & Graphics* 52 (2015), pp. 129–141.

[162] H. Pottmann, J. Wallner, Q.-X. Huang, and Y.-L. Yang: "Integral invariants for robust geometry processing". In: *Computer Aided Geometric Design* 26.1 (2009), pp. 37–60.

[163] C. R. Qi, H. Su, M. Nießner, A. Dai, et al.: "Volumetric and Multi-View CNNs for Object Classification on 3D Data". In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2016.

[164] K. Radner and E. Robson: *The Oxford Handbook of Cuneiform Culture*. Oxford Handbooks in Classics and Ancient History. OUP Oxford, 2011.

[165] M. Rashad, M. Khamiss, and M. MOUSA: "A Review on Mesh Segmentation Techniques". In: *International Journal of Engineering and Innovative Technology (IJEIT)* 6 (2017).

[166] H. C. Rawlinson and T. G. Pinches: *A Selection from the Miscellaneous Inscriptions of Assyria and Babylonia: #35*. Harrison & Sons, 1909.

[167] J. E. Reade: "The Manufacture, Evaluation and Conservation of Clay Tablets Inscribed in Cuneiform: Traditional Problems and Solutions". In: *Iraq* 79 (2017), pp. 163–202.

[168] D. Reniers and A. Telea: "Hierarchical part-type segmentation using voxel-based curve skeletons". In: *The Visual Computer* 24.6 (2008), pp. 383–395.

[169] D. Reniers, J. van Wijk, and A. Telea: "Computing Multiscale Curve and Surface Skeletons of Genus 0 Shapes Using a Global Importance Measure". In: *IEEE Transactions on Visualization and Computer Graphics* 14.2 (2008), pp. 355–368.

[170] A. Richardson and U. Smilansky: "Geometrical Feature Extraction for Cuneiforms". In: *Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pp. 1173–1178.

[171] E. Rieken: *Einführung in die hethitische Sprache und Schrift*. Lehrbücher orientalischer Sprachen / 1: Cuneiform languages. Ugarit-Verlag, 2011.

[172] K. K. Riemschneider: *Lehrbuch des Akkadischen*. Verlag Enzyklopädie, 1984.

[173] H. Riesel: *Prime Numbers and Computer Methods for Factorization*. Birkhauser Boston Inc, 1985.

[174] T. Ritschel, T. Grosch, and H.-P. Seidel: "Approximating Dynamic Global Illumination in Screen Space". In: *Proceedings ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 2009.

[175] J. B. Roerdink and A. Meijster: "The Watershed Transform: Definitions, Algorithms and Parallelization Strategies". In: *Fundam. Inf.* 41.1,2 (2000), pp. 187–228.

[176] L. Rokach and O. Maimon: "Clustering Methods". In: *Data Mining and Knowledge Discovery Handbook*. Ed. by O. Maimon and L. Rokach. Springer US, 2005, pp. 321–352.

[177] A. Roshop and C. Cruse: “Digital Character Recognition of Cuneiform Inscriptions via Neural Networks”. In: *Optical Technologies in the Humanities*. Ed. by D. Dirksen and G. von Bally. Springer Berlin Heidelberg, 1997, pp. 175–178.

[178] C. Rössl, L. Kobbelt, and H.-P. Seidel: “Extraction of feature lines on triangulated surfaces using morphological operators”. In: *Proceedings of the AAAI Symposium on Smart Graphics*, pp. 71–75.

[179] R. J. Rost, B. Licea-Kane, D. Ginsburg, J. M. Kessenich, et al.: *OpenGL Shading Language*. 3rd. Addison-Wesley Professional, 2009.

[180] L. Rothacker, D. Fisseler, G. Müller, F. Weichert, and G. A. Fink: “Retrieving Cuneiform Structures in a Segmentation-free Word Spotting Framework”. In: *International Workshop on Historical Document Imaging and Processing* (2015), pp. 129–136.

[181] M. Ruiz, L. Szirmay-Kalos, T. Umenhoffer, I. Boada, et al.: “Volumetric ambient occlusion for volumetric models”. In: *The Visual Computer* 26.6-8 (2010), pp. 687–695.

[182] S. Rusinkiewicz: “Estimating Curvatures and Their Derivatives on Triangle Meshes”. In: *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2Nd International Symposium*. 3DPVT ’04. IEEE Computer Society, 2004, pp. 486–493.

[183] C. Rüster and E. Neu: *Hethitisches Zeichenlexikon: Inventar und Interpretation der Keilschriftzeichen aus den Boğazköy-Texten*. Hethitisches Zeichenlexikon: Inventar und INterpretation der Keilschriftzeichen aus den Bogazkoy-Texten. Harrassowitz, 1989.

[184] R. B. Rusu, N. Blodow, and M. Beetz: “Fast Point Feature Histograms (FPFH) for 3D Registration”. In: *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*. ICRA’09. IEEE Press, 2009, pp. 1848–1853.

[185] R. B. Rusu and S. Cousins: “3d is here: Point cloud library (pcl”. In: *In Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[186] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi: “Optimization by simulated annealing”. In: *SCIENCE* 220.4598 (1983), pp. 671–680.

[187] E. B. Saff and A. Kuijlaars: “Distributing many points on a sphere”. In: *The Mathematical Intelligencer* 19.1 (1997), pp. 5–11.

[188] M. Salvini: *Keilschrifturkunden aus Boghazköi: Texte des hurritischen Kreises*. Keilschrifturkunden aus Boghazköi. Akademie-Verlag, 1975.

[189] P. V. Sander, D. Nehab, and J. Barczak: “Fast Triangle Reordering for Vertex Locality and Reduced Overdraw”. In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH ’07. ACM, 2007.

[190] M. Sattler, R. Sarlette, G. Zachmann, and R. Klein: “Hardware-accelerated ambient occlusion computation”. In: *Vision, Modeling, and Visualization 2004*. Ed. by B. Girod, M. Magnor, and H.-P. Seidel. Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2004, pp. 331–338.

[191] D. Scharstein and R. Szeliski: "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms". In: *Int. J. Comput. Vision* 47.1-3 (2002), pp. 7–42.

[192] J. L. Schonberger and J. Frahm: "Structure-from-Motion Revisited". In: *IEEE Conference on Computer 2016*. Vol. 00, pp. 4104–4113.

[193] E. Schubert and A. Zimek: *ELKI: Environment for DeveLoping KDD-Applications Supported by Index-Structures*. 2008. URL: `https://elki-project.github.io/` (visited on 11/16/2018).

[194] J. Serra: *Image Analysis and Mathematical Morphology*. Academic Press, Inc, 1983.

[195] A. Shamir: "A survey on Mesh Segmentation Techniques". In: *Computer Graphics Forum* (2008).

[196] P. Shanmugam and O. Arikan: "Hardware Accelerated Ambient Occlusion Techniques on GPUs". In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. I3D '07. ACM, 2007, pp. 73–80.

[197] S. Shantaiya, K. Verma, and K. Mehta: "Article: A Survey on Approaches of Object Detection". In: *International Journal of Computer Applications* 65.18 (2013), pp. 14–20.

[198] L. Shapira, A. Shamir, and D. Cohen-Or: "Consistent mesh partitioning and skeletonisation using the shape diameter function". In: *The Visual Computer* 24.4 (2008), pp. 249–259.

[199] A. Sharf, M. Alexa, and D. Cohen-Or: "Context-based surface completion". In: *ACM Trans. Graph.* 23.3 (2004), p 878–887.

[200] D. Shreiner and The Khronos OpenGL ARB Working Group: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*. 7th. Addison-Wesley Professional, 2009.

[201] R. Sibson: "SLINK: An optimally efficient algorithm for the single-link cluster method". In: *The Computer Journal* 16.1 (1973), pp. 30–34.

[202] D. Sieger and M. Botsch: "Design, Implementation, and Evaluation of the Surface_mesh Data Structure". In: *Proceedings of the 20th International Meshing Roundtable*. Ed. by W. R. Quadros. Springer Berlin Heidelberg and Springer-Verlag GmbH Berlin Heidelberg, 2012, pp. 533–550.

[203] A. Slizewski and P. Semal: "Experiences with low and high cost 3D surface scanner". In: *Quartär* 56 (2009), pp. 131–138.

[204] P.-P. J. Sloan, W. Martin, A. Gooch, and B. Gooch: "The Lit Sphere: A Model for Capturing NPR Shading from Art". In: *Proceedings of Graphics Interface 2001*. GI '01. Canadian Information Processing Society, 2001, pp. 143–150.

[205] D. Smeets, J. Keustermans, D. Vandermeulen, and P. Suetens: "meshSIFT: Local Surface Features for 3D Face Recognition Under Expression Variations and Partial Data". In: *Comput. Vis. Image Underst.* 117.2 (2013), pp. 158–169.

[206] R. Socher, B. Huval, B. Bhat, C. D. Manning, and A. Y. Ng: "Convolutional-recursive Deep Learning for 3D Object Classification". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Curran Associates Inc, 2012, pp. 656–664.

[207] M. Spivak: *A Comprehensive Introduction to Differential Geometry*. A Comprehensive Introduction to Differential Geometry. Publish or Perish, Incorporated, 1975.

[208] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, eds.: *Point Feature Extraction on 3D Range Scans Taking into Account Object Boundaries*. 2011.

[209] C. Stentoumis, L. Grammatikopoulos, I. Kalisperakis, and G. Karras: "On accurate dense stereo-matching using a local adaptive multi-cost approach". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 91 (2014), pp. 29–49.

[210] A. Tagliasacchi, T. Delame, M. Spagnuolo, N. Amenta, and A. Telea: "3D Skeletons: A State-of-the-Art Report". In: *Computer Graphics Forum* (2016).

[211] G. K. L. Tam, Z. Q. Cheng, Y. K. Lai, F. C. Langbein, et al.: "Registration of 3D Point Clouds and Meshes: A Survey from Rigid to Nonrigid". In: *IEEE Transactions on Visualization and Computer Graphics* 19.7 (2013), pp. 1199–1217.

[212] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross: "Optimized Spatial Hashing for Collision Detection of Deformable Objects". In: *VMV 2003*, pp. 47–54.

[213] *The British Museum cuneiform collection at CDLI: Cuneiform Digital Library Initiative*. URL: `https://cdli.ucla.edu/collections/bm/bm.html` (visited on 11/16/2018).

[214] *The Iraq Museum cuneiform collection at CDLI: Cuneiform Digital Library Initiative*. URL: `https://cdli.ucla.edu/collections/im/im_en.html` (visited on 11/16/2018).

[215] R. L. Thorndike: "Who belongs in the family". In: *Psychometrika* (1953), pp. 267–276.

[216] H. Todo, K. Anjyo, and S. Yokoyama: "Lit-Sphere Extension for Artistic Rendering". In: *The Visual Computer* 29.6-8 (2013), pp. 473–480.

[217] C. Tomasi and T. Kanade: "Shape and motion from image streams under orthography: A factorization method". In: *International Journal of Computer Vision* 9.2 (1992), pp. 137–154.

[218] F. Tombari, S. Salti, and L. Di Stefano: "Unique Signatures of Histograms for Local Surface Description". In: *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*. ECCV'10. Springer-Verlag, 2010, pp. 356–369.

[219] P. Torr and A. Zisserman: "MLESAC". In: *Comput. Vis. Image Underst.* 78.1 (2000), pp. 138–156.

[220] *VCG Library*. URL: `http://vcg.isti.cnr.it/vcglib/` (visited on 11/16/2018).

[221] N. Veldhuis: *History of the Cuneiform Lexical Tradition*. Guides to the Mesopotamian textual record. Ugarit-Verlag, 2014.

[222] R. Vergne, R. Pacanowski, P. Barla, X. Granier, and C. Schlick: "Improving Shape Depiction under Arbitrary Rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 17.8 (2011), pp. 1071–1081.

[223] R. Vergne, R. Pacanowski, P. Barla, X. Granier, and C. Schlick: "Light Warping for Enhanced Surface Depiction". In: *ACM Transaction on Graphics (Proceedings of SIGGRAPH 2009)* 28.3 (2009).

[224] R. Vergne, R. Pacanowski, P. Barla, X. Granier, and C. Schlick: "Radiance Scaling for Versatile Surface Enhancement". In: *I3D '10: Proc. symposium on Interactive 3D graphics and games*. ACM, 2010.

[225] L. Vincent and P. Soille: "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 13.6 (1991), pp. 583–598.

[226] M. Vlaminck, H. Luong, and W. Philips: "Multi-resolution ICP for the efficient registration of point clouds based on octrees". In: *Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*. 2017, pp. 334–337.

[227] M. Wagner, U. Labsik, and G. Greiner: "Repairing Non-Manifold Triangle Meshes using Simulated Annealing". In: *International Journal of Shape Modeling* 9.2 (2003), pp. 137–154.

[228] I. Wald and V. Havran: "On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N)". In: *IEEE Symposium on Interactive Ray Tracing*. 2006, pp. 61–69.

[229] L. Watkins JR. and D. A. Snyder: *The Digital HammurabiProject*. 2003.

[230] R. Westermann, L. Kobbelt, and T. Ertl: "Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surfaces". In: *The Visual Computer* 15.2 (1999), pp. 100–111.

[231] G. Willems, F. Verbiest, W. Moreau, H. Hameeuw, et al.: "Easy and cost-effective cuneiform digitizing". In: *The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST 2005)*. 2005, pp. 73–80.

[232] S. Willems: *OpenGL Hardware Database: (Capabilities)*. URL: `http://opengl.gpuinfo.org/listcapabilities.php` (visited on 11/16/2018).

[233] S. I. Woolley, T. R. Davis, N. J. Flowers, J. Pinilla-Dutoit, et al.: "Communicating cuneiform: The evolution of a multimedia cuneiform database". In: *Visible Language*. Vol. 36.3.

[234] S. I. Woolley, N. J. Flowers, T. N. Arvanitis, A. Livingstone, et al.: "Three-dimensional capture, representation, and manipulation of Cuneiform tablets". In: *Proc.SPIE*. Vol. 4298.

[235] H. Xu and J. Barbič: "Signed Distance Fields for Polygon Soup Meshes". In: *Graphics Interface 2014* (2014).

[236] X. Xu, A. Dehghani, D. Corrigan, S. Caulfield, and D. Moloney: "Convolutional Neural Network for 3D object recognition using volumetric representation". In: *First International Workshop on Sensing, Processing and Learning for Intelligent Machines (SPLINE)*. 2016, pp. 1–5.

[237] S. Yoshizawa, A. G. Belyaev, and H.-P. Seidel: "Free-form Skeleton-driven Mesh Deformations". In: *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications*. SM '03. ACM, 2003, pp. 247–253.

[238] A. Zaharescu, E. Boyer, K. Varanasi, and R. Horaud: "Surface feature detection and description with applications to mesh matching". In: *CVPR 2009 - IEEE International Conference on Computer Vision and Pattern Recognition*, pp. 373–380.

[239] W. Zhao, S. Gao, and H. Lin: "A robust hole-filling algorithm for triangular mesh". In: *The Visual Computer* 23.12 (2007), pp. 987–997.

[240] S. Zhukov, A. Inoes, and G. Kronin: "An Ambient Light Illumination Model". In: *Rendering Techniques '98*. Ed. by G. Drettakis and N. Max. Eurographics. Springer-Verlag Wien New York, 1998, pp. 45–56.