# Discrete Displacement Fields: A Versatile Representation of Geometry for Simulation in Computer-Aided Manufacturing

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik
von

**Dipl.-Inform. Jörg Ayasse**

Lehrstuhl VII - Graphische Systeme
Fachbereich Informatik
Universität Dortmund

Dortmund
2003

Tag der mündlichen Prüfung: 24.07.2003

Dekan: Prof. Dr. Bernhard Steffen

Gutachter: Prof. Dr. Heinrich Müller, Prof. Dr. Berthold Vöcking

# Contents

# Chapter 1

# Introduction

The shapes of mechanical parts become more and more complex. One reason is the increasing demand on the quality of design and usability visible in many goods of daily use, for example sanitary armatures, domestics, and car bodies. Another source of complexity are technical requirements on goods, like they occur in particular for aircrafts, ship bodies, medical instruments, or turbine blades. As a consequence the complexity of the related processes of planning and production increases, too. The hope is that the introduction of computers helps to cope with the complexity, and, even more, to decrease the time required for the development of new products.

Two important technologies of production are *milling* and *grinding*. In milling, a milling cutter is moved through a block of material and erases material until a desired shape is reached. In grinding, a workpiece is pressed against a rotating grinding disc or belt in order to remove small surface artifacts and to polish the surface. Both processes have in common that a tool (milling cutter, grinding disc/belt) is moved along the surface of a workpiece, removing material from the workpiece. Usually computer-controlled machines are used for this purpose. A program is executed which yields a motion path leading to the desired final shape of the workpiece. A central task of production planning, known as *path planning problem*, is to find a suitable motion path for a given desired shape and an initial workpiece automatically.

Another technology of production is *deformation*. For instance, a planar metal plate is pressed into a mould in order to bring it into a desired shape. Many parts of car bodies are produced in this way. A problem arising with deformation is to find shapes and moulds which do not cause damages during the process of deformation and which yield a resulting workpiece satisfying the requirements with respect to its mechanical behavior and the approximation of the desired shape.

The path planning problem and the mould planning problem are usually solved iteratively. First, a solution is developed according to some procedure or algorithm. Second, the solution is validated by execution of the solution on a real machine or by computational simulation. Disadvantages of execution on a real machine are that errors in the solution may cause a damage of parts of the machine, and that the machine cannot be used for production during the tests. For such reasons, simulation has found increasing interest, and has meanwhile also penetrated the practice of production. However, there are still many unsolved problems. The problems in particular concern the data transfer between the real world and the virtual world of simulation, the accuracy of simulation, and the efficiency of simulation.

Simulation is based on a model of the real process which is responsible for the accuracy of the simulation. A model is represented by a data structure which should efficiently support the operations applied to the model during simulation. Usually, the core of such data structures is the representation of the geometry of the environment in which the production process takes place. Parts of the environment are the workpiece, the tool, and the components of the production machine. Many approaches to the representation of geometric shapes have been developed in the past. A reason is that a universal representation satisfying the requirements of different types of shapes, and in particular of different types of operations for modification of a shape, has not been found yet. Each representation supports some sort of operation canonically and efficiently, while other operations are quite artificially and thus complex to implement. For example, a B-spline representation is well suited for deformations, whereas subtraction of material requires some effort, for instance trimming and pasting of patches. On the other hand, regular cell structures support removal of material quite well while deformations cannot be immediately implemented.

In this thesis we propose to use so-called *displacement fields* (DFs) as the basis of representation of geometric shapes in simulations of production processes. A displacement field consists of a supporting surface $S$ on which a vector field $\mathbf{V}$ is given. The vector assigned to every point on $S$ defines a displacement of the point in space. The union of the displaced points define a new surface $F$ which is the surface represented by the displacement field. Displacement fields have been used before in computer graphics for representation of surface details. The idea is to represent the rough shape of $F$ by $S$, and the details of $F$ by $\mathbf{V}$. If normal vectors exist on $S$, they may be a natural choice for the displacement vector field.

The interesting point of DFs is that they allow further interpretations. For example, if the displacement vectors are seen as line segments, their union will define a solid fringe on $S$. Material can be removed from the solid fringe by cutting displacement vectors, for instance by a milling tool or a grinding disc. The discrete version we favor in the following is possibly more intuitive. In the discrete

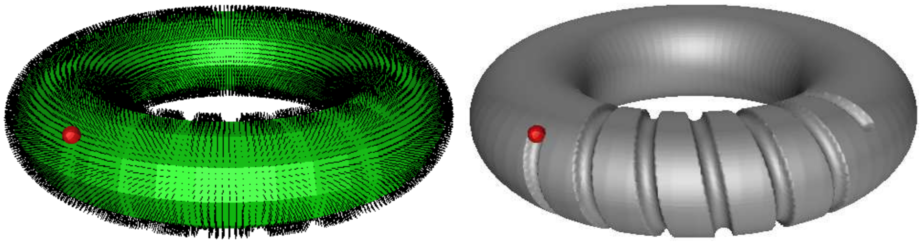


**Figure 1.1:** *A ball moves around a torus and engraves a groove into the surface. The torus is represented by a discrete displacement field (DDF) and the ball in motion cuts the stubbles of the DDF.*

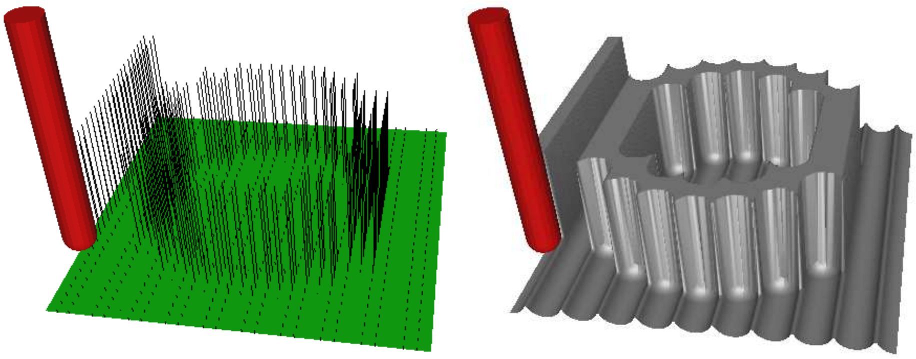


**Figure 1.2:** *Simulation of a 3-axis milling process. The workpiece is represented by a discrete displacement field and the milling cutter in motion shortens the stubbles.*

version, $S$ is rasterized into discrete points. Every point has a displacement vector which can now be seen as a stubble in a field of stubbles defined of $S$. Material removal means to cut stubbles. If a tool is moved along the surface, the parts of stubbles intersected by the tool will be cut off.

Figures 1.1-1.3 give some examples for discrete displacement fields. The support surface is a triangle mesh depicted in green color. The surface represented by the tips of the black stubbles is shown in gray. In figure 1.1, a ball moves around a torus and engraves a groove into the surface by cutting off stubbles of a discrete displacement field. In figure 1.2 a 3-axis milling simulation is established in a similar way. The workpiece is represented by a planar triangular mesh with parallel stubbles. The cutter is moved along the workpiece and cuts off stubbles. Both examples demonstrate the usefulness of discrete displacement fields for the simulation of material-removing manufacturing processes.

A further interpretation of displacement fields is to understand the vector field $\mathbf{V}$ as a function between the supporting surface $S$ and the represented surface $F$.

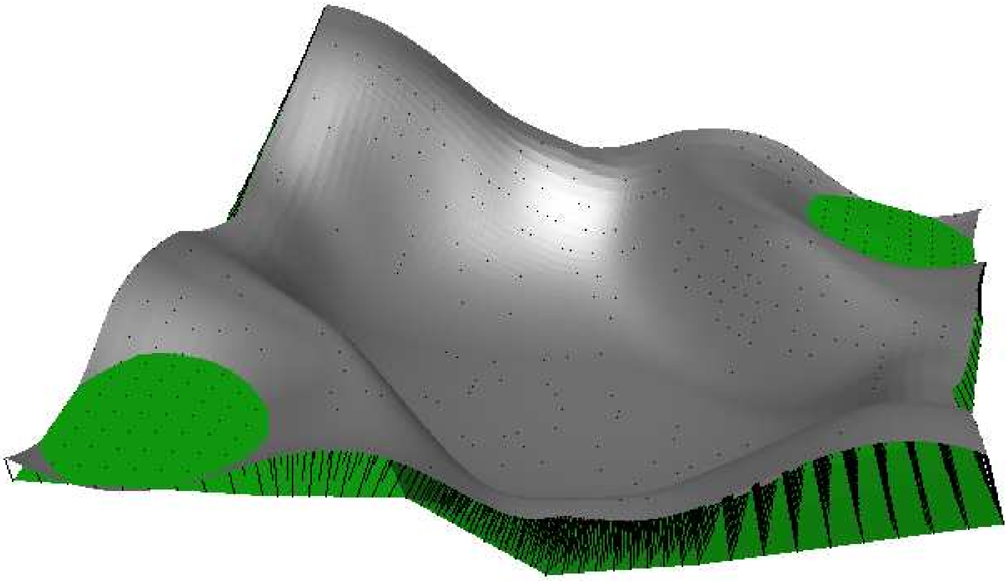


**Figure 1.3:** *Example of a deformation. The gray surface can be interpreted as deformation of the green surface. A penetration of both surfaces is possible.*

The function defines a deformation of $S$ into $F$. Figure 1.3 shows an example. $S$ is represented in green and $F$ in gray. From the two surfaces, a continuous family of surfaces $F(\lambda)$, $\lambda \in [0, 1]$ can be obtained by $F(\lambda) := \{\mathbf{p} + \lambda\mathbf{v}(\mathbf{p}) \mid \mathbf{p} \in S\}$ where $\mathbf{v}(\mathbf{p})$ is the displacement vector of $\mathbf{p}$. Surface families like that can be used to find surfaces well-suited for moulds in deformation-based production. In that case, $S$ might be the ideal mould, and $F$ might be the result of deformation of a given plate simulated with this mould.

In this thesis, we give a new definition of discrete displacement fields (DDFs) which is suited to using displacement fields as central data structure in production processes in the way just outlined. In contrast to other definitions it emphasizes the view of the displacement field as a crust and the properties of the displacement fields as a mapping between two surfaces. The central underlying concept is the unique-projection environment.

The supporting surface of our DDFs is a triangular mesh. The triangles are regularly rasterized. A vector is assigned to every raster point which is obtained by interpolation from three displacement vectors assigned to the three vertices of every triangle. The length of the vector is defined by a scaling factor which is stored in a raster array assigned to the triangle. Thus the displacement vector field is specified by vectors at the vertices of every triangle and raster arrays assigned to the triangles.

For the application of our definition of DDF we present several new algorithms which can be basic building blocks of DDF-based simulation of production processes.

**DDF depth-buffer algorithm**

The DDF depth-buffer algorithm allows to update efficiently the scaling factors of the vectors of the displacement vector field which are intersected by an arbitrary triangle in space. The updating consists in taking the minimum of the scaling factor of the triangle-vector-intersection point and the scaling factor of the vector and in replacing the latter with the new value. The main difference to the conventional depth- or z-buffer-algorithm [Fol90] is projection along the displacement vector field. This complicates the calculation of depth, and the additional problem of searching for the supporting triangles involved in projection has to be treated. The DDF depth-buffer algorithm can be understood as a special version of Breadth-First-Raytracing [Mue88, Nak97].

For the calculation of depth we will show that among two alternative approaches "**V**-shooting" is more efficient than "**V**-projection". The advantage of **V**-shooting is that it can be carried out incrementally over a raster which leads to a valuable gain in efficiency as known from several algorithms in computer graphics.

Search space restriction for determination of the supporting triangles involved in projection is achieved by regular grids. In contrast to the standard representation of the grid by a multidimensional array which represents a one-to-one mapping between grid cells and array elements we propose to use a hashing structure in which just the occupied grid cells are stored. This approach saves memory space since the space requirements are linear in the volume of the DDF, and not in the volume of the bounding box which might be significantly, that is up to an order, more. Furthermore, shape modifications of the DDF are efficiently supported by the more efficient possibilities of dynamic updating of the hashing structure.

**Unique-projection test**

A region in space has the unique-projection property with respect to a supporting triangle $s$ of a DDF if no two displacement vectors of $s$ induce a line traversing the same point of the region. We present an algorithm which checks a type of "crust" around a triangle $s$ for this property. The problem is formulated as an optimization problem to which an algorithm of solution is presented. This approach allows to extend or shrink a given "crust" to a maximum feasible size.

**Conversion of triangular meshes into DDFs**

The goal of conversion is to derive a supporting mesh of a DDF of a low number of triangles which allows to represent a given triangular surface mesh of a given shape (figure 6.2). We present an algorithm which is based on conventional edge contraction known from other mesh reduction problems. However, particular constraints have to be considered which take into consideration the particular requirements of our problem. Besides bounding the size of the angles as required in many other applications, too, an important constraint is to ensure a one-to-one mapping between the reduced and the original mesh by the displacement vector field. The DDF depth-buffer and the unique-projection test are applied as efficient tools.

**Removal of material from a DDF-represented surface**

We solve this problem in a straightforward way by positioning the tool at a sufficiently dense sequence of locations along a path, and applying the DDF depth-buffer algorithm at every location. The tool is given by its surface which consists of triangles. We demonstrate that the DDF depth-buffer algorithm is fast enough to make this approach practically feasible.

**Curvature calculation on a DDF-represented surface**

The DDF-representation can be understood as a discretized parametric representation of the represented surface. We use this view for approximate calculation of the main curvatures and the directions of main curvatures at the raster points of the DDF by evaluation of the classical formulas of differential geometry by a finite-difference approximation. For calculation of curvature lines we propose an algorithm for stream-line calculation in the vector fields adapted to the rasterized representation. The algorithm resembles that of [Roe00]. However, the difference is that just raster points are used by our algorithm for approximation of the stream-line due to the high resolution of the raster. This requires an error correction which is achieved by addition of a compensation vector derived from the error of the preceding step. Furthermore, rasterization artifacts are eliminated by smoothing the resulting stream-line by a low-pass filter.

Curvature is important for production processes because the curvature of a workpiece influences the possible size of a tool. Furthermore, lines of curvature, that is stream-lines in the vector fields of main curvature directions, are favorable tool paths with respect to surface approximation by the tool.

The DDF depth-buffer algorithm, the unique-projection test, and the approach to simulation of material removal have been published in [Aya02]. The DDF depth buffer has also been presented in an application-oriented paper [Wei02]. In [Aya01], alternative definitions of rasterized height fields have been described and discussed. Curvature calculation and other aspects of path planning were addressed in [Aya03].

A particular goal is to offer algorithms which are useful and efficient in practice. Almost all presented algorithms have been implemented and their practical performance has been analyzed empirically. In several cases, an improved efficiency is achieved by minimizing the number and type of arithmetic operations, mainly by incremental evaluation of functions at consecutive parameter values, which is a basically well-known approach in computer graphics. Problems of location where the asymptotic behavior of algorithms is relevant, are solved by heuristics, based on approaches known from computer graphics, and adapted and further developed to the requirements of the problems treated here. This approach has been preferred against the alternative to develop sophisticated worst-case efficient algorithms, based on results of worst-case-oriented computational geometry.

The thesis is organized as follows. Chapter 2 gives a survey on related work, in particular concerning the concept of displacement fields and other representations of workpieces related to production processes. Chapter 3 is devoted to the precise definition of discrete displacement fields in our sense. Chapter 4 presents concepts and algorithms related to projection, in particular **V**-projection, **V**-shooting, and the unique-projection property. Chapter 5 describes the depth buffer algorithm on discrete displacement fields. Chapter 6 turns to the problem of conversion of surface representations, in particular of triangular meshes, into a representation by discrete displacement fields. Chapter 7 deals with the adaptive triangulation of a surface represented by a discrete displacement field. The application of discrete displacement fields for tool path planning and for deformation is considered in chapter 8 and chapter 9. Chapter 10 presents conclusions.

# Chapter 2

# Related Work and Concepts

Displacement fields have been mentioned probably for the first time in literature by Cook [Coo84], as a means for improved rendering, similar to texture, Phong shading, and bump mapping [Fol90]. These concepts allow to specify surface details without modeling of all details by conventional geometry, thus saving modeling efforts, space, and rendering time. Textures influence reflection properties of the surface, but do not influence the geometry. Phong shading and bump mapping modify the geometry virtually in the rendered image by variation of the surface normal vector. The fake can be recognized at silhouettes of the surface which are still represented by the coarse geometry. This problem can be remedied by displacement fields which yield the necessary detailed geometry for rendering along the silhouette.

For surfaces having a normal vector at every point, fields of normals are canonical displacement fields. Some normal vector field can often be immediately calculated from the surface, like $\mathbf{f}_u \times \mathbf{f}_v$ for a parametric surface $\mathbf{f}(u, v)$, or $(F_x, F_y, F_z)$ for an implicit surface $F(x, y, x) = 0$. A displacement field is found by combining the normal vector field inherent to the surface with a scalar height function. The height function yields a factor by which the calculated normal has to be multiplied. The scalar height field can be provided analogously to a texture map.

Efficient rendering of displacement fields including hardware implementation has been investigated recently by Gumhold et al. [Gum99] and Dogget et al. [Dog01].

The aspect of conversion of a given surface into a displacement field representation over a more or less coarse support surface has been treated by several authors with different motivations [Ped94, Kri96, Cig99, Oli00, Lee00, Kob00]. Aspects are data reduction and rendering efficiency.

The possibility of editing surfaces by subtraction and addition of material has been briefly outlined for displacement fields by Lee et al. [Lee00], however without going into the details how it is performed. The details concern the question for the space where editing is performed (in "texture"-space or in 3D-space), but also the question for efficient algorithmic implementation. These are questions emphasized in this thesis.

Discrete normal vector fields have been used for calculation of the error of numerically controlled milling of surfaces by cutting the vectors by the swept volume of the milling tool [Jer89, Oli90, Hua94]. Algorithmic efficiency was mainly gained by simplification, for example by replacing the normal vector field by a vector field perpendicular to a plane, which is a classical height field. In contrast to those approaches, the displacement fields are treated immediately without simplification in this thesis.

For simulation of the whole milling process two main streams of approaches can be distinguished. One stream is to consider milling as subtraction of the tool from the workpiece, and using the operation of subtraction provided by models based on constructive solid geometry (CSG) [Req80]. Instead of iterative subtraction of the tool at densely chosen locations on the tool path, the volume swept by the surface over a segment of the tool path may be calculated and subtracted from the workpiece geometry. This approach reduces the number of CSG-subtractions to be performed, but rises the problem of calculation of the swept volume [Hui94]. For CSG-systems based on representations of geometry by polygons or polynomal patches, the surface details often arising in milling and grinding simulations cause a high number of polygons or patches which might lead to inefficient space and time requirements.

The second stream is the rasterized representation. The possibly most preferred class is defined by space-filling volume representations which include dexel volumes and voxel volumes [Hoo86, Hui94, Hua94, Kra96].

The original dexel model (dexel = depth element) has been developed by van Hook [Hoo86] for real-time shaded display of the milling process of a solid. The view point dependency of this approach has been overcome by Huang and Oliver [Hua94]. These representations of solids can also be understood as a representation by rays which has been the view of Roth [Rot80] and Ellis and al. [Ell91]. Dexel and ray representations have also been used by them for efficient calculations in the context of constructive solid modeling (CSG), also see Hui [Hui94]. The mentioned representations differ somewhat in the data structures used, and in the additional information, like color, normals, etc., which is stored explicitly. The G-buffer of Saito and Takashi [Sai91] maintains particularly comprehensive information.

Dexels represent volumes or surfaces over just one reference plane. This fact causes samples of heterogenous densities since the density of the sampling points caused by the dexels depends on the slope of the surface relative to the reference plane. This problem can be overcome by using multi-dexel models. A typical incarnation of this idea is the three-dexel-model with mutual perpendicular reference planes, whose dexel lines form a virtual 3D-grid [Ben97, Mue03].

A classical approach is to use an explicit 3D grid, leading to the representation of volumes by voxel models. Voxel models are widely used for representation and visualization of volumetric image data, but they have been used for solid modeling purposes as well [Gal91, Kra96]. In the simplest case, the vertices of a spatial grid represent volume elements which are labeled by 1 and 0, dependent on whether the vertex is inside the solid or not (point sampling version). An advantage of the voxel representation over the representations just outlined is the relatively uniform distribution of the voxels representing the solid and its surface. However, a disadvantage is that usually none of the vertices is located exactly on the boundary of the solid because of the quantization onto the grid, so precise information on the surface of the solid is not available. This problem does not occur with dexel models if the dexel vertices are stored by real numbers.

One approach to diminish this problem is to store the signed distance of every grid vertex to the boundary of the solid, for instance with a positive sign for vertices outside the solid [Per01]. Then the surface can be approximated as the iso-surface of distance 0, but the reconstruction still does not to be exact. An advantage of signed distance fields, however, is that they can be used for collision detection and response. A particular problem of distance fields which makes them less suited to simulation of milling and grinding is the difficulty of updating the distance field if the shape of the solid is modified.

Another difficulty of voxel models are the space requirements for highly resolved grids. The space requirements can be reduced by applying a compression scheme to the voxel model. The representation of a solid by an octree can be understood as a compressed voxel representation [Mea82]. Analogously, the dexel representation of a solid can be understood as a run-length compression of a voxel representation, at least if the dexel vertices are quantized onto grid vertices. Octrees have also been used for a hierarchical representation of distance fields [Fri00].

Besides the two main streams, further possibilities exist. A problem with space-filling regular rasters is that a considerable number of elements are required even in the compressed version. An alternative might be the usage of irregular space-filling meshes which adapt to the shape of a represented solid. An important example are the tetrahedral meshes.

Another possibility is to use clouds of points. Clouds of points have recently found interest for modeling and visualization purposes [Zwi02]. One reason is that dense point clouds are the natural output of many scanning devices, so immediate processing without conversion in one of the classical representations is attractive. A disadvantage is that irregularly located points require more data to be stored in order to describe their locations than regularly located vertices on rasterized representations.

The discrete displacement fields of this thesis can be seen as a sort of "semi-discrete" representation between the two main streams. They store the information where it is required and do not cover the whole space. They are rasterized as well, so methods like incremental evaluation of expressions can also be used.

Another example of a semi-discrete representation are the height field patches of Ivanov [Iva01]. The height field patches cover the surface of a solid by local height field maps which have supporting planes approximately tangent to the surface. A height field map is a height function over a region of a plane which may be given by an array of height values corresponding to a rasterization of the region. An advantage of the height field patches against discrete displacement fields is that height fields can be manipulated more efficiently because of the constant direction of displacement. A disadvantage, however, is, that the height field patches need to overlap, and that the overlaps may cause visual artifacts at rendering.

Wavelet representations which came up in the last few years can also be seen as a semi-discrete representation of height fields. The wavelets define a sequence of displacements at scales of different resolution to be applied to a given supporting surface. Duchaineau et al. [Duc03] exemplify how this representation can be used for milling simulations. The example, however, is restricted to a regular mesh as support surface. An extension of the discrete displacement fields in this direction may be of interest, but is outside the scope of this thesis.

This survey shows that discrete displacement fields are a useful alternative to existing representations of geometry. Another advantage against many of the other representations is that displacement fields are also well-suited for deformation. Deformation disturbs regularity, and at least resampling is necessary for re-establishing.

The aspect of shape modification by deformation has particularly been addressed by Kobbelt et al. [Kob00]. By combining the deformation approach of Kobbelt et al. with the discrete displacement field, a representation may be derived which supports operations of deformation and material removal and application equally well.

# Chapter 3

# Definition of Discrete Displacement Fields

A *discrete displacement field* (*DDF*) $D = (S, \mathbf{V}, h)$ is defined by an oriented triangle mesh $S$, called *support*, a *continuous vector field* $\mathbf{V}$, and a *discrete height field* $h$.

A triangle mesh in space is composed of vertices, edges, and triangles. Every triangle has three edges, and every edge has two vertices. The three edges of a triangle yield the three vertices of a triangle. We assume that the mesh defining the support of a DDF is manifold, that is, every edge has at most two incident triangles. A manifold mesh is called orientable, if a front and a back side can be assigned to every triangle so that, starting at any point on the front side of a triangle, no point on the backside can be reached when walking on the surface, without traversing a boundary edge of the mesh. A boundary edge is an edge with just one incident triangle. For example, if $S$ bounds a solid, the outer side of S could be taken as front side, and the inner side as back side.

The definition of DDFs can be applied to non-manifold meshes as well, but a usefully represented surface is not guaranteed in that case.

The vector field $\mathbf{V}$ assigns a vector $\mathbf{v}$ to every surface point $\mathbf{p}$ of the triangle mesh $S$. $\mathbf{V}$ is defined by vectors $\mathbf{v}_i$ assigned to the vertices $\mathbf{p}_i$ of every triangle $s \in S$, $i = 1, 2, 3$. The vector $\mathbf{v}(\mathbf{p})$ at an arbitrary point $\mathbf{p} \in s$ is calculated by barycentric interpolation,

$$\mathbf{v}(\mathbf{p}) = \sum_{i=1}^{3} \mu_i \mathbf{v}_i,$$

where

$$\mathbf{p} = \sum_{i=1}^{3} \mu_i \mathbf{p}_i, \ \sum_{i=1}^{3} \mu_i = 1, \ \mu_i \geq 0, \ i = 1, 2, 3.$$

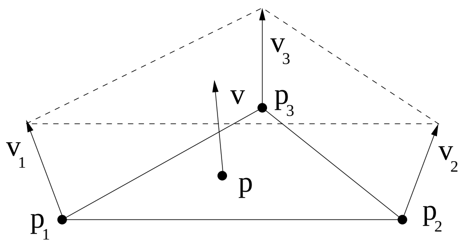



**Figure 3.1:** *Definition of a vector field over a triangle. The point* $\mathbf{p}$ *and the vector* $\mathbf{v}$ *are obtained from* $\mathbf{p}_i$ *and* $\mathbf{v}_i$*, respectively,* $i = 1, 2, 3$*, by barycentric interpolation.*

Figure 3.1 illustrates the definition.

The vector field $\mathbf{V}$ assigns a displacement direction to every surface point $\mathbf{p}$ of $S$. The amount of displacement is given by the height field $h$. In contrast to $\mathbf{V}$ the height field $h$ is discrete, not continuous. On every triangle $s$ of $S$ a regular grid of height values $h$ is established. The vertices $\mathbf{g}_{i,j,k}$ of a grid of *resolution* $m$ are

$$\mathbf{g}_{i,j,k} := \frac{i}{m-1}\mathbf{p}_1 + \frac{j}{m-1}\mathbf{p}_2 + \frac{k}{m-1}\mathbf{p}_3$$

where $i+j+k = m-1$, $i, j, k \geq 0$, and $\mathbf{p}_i$, $i = 1, 2, 3$, the vertices of $s$ (Figure 3.2). The corresponding height values are denoted by $h_{i,j,k}$.

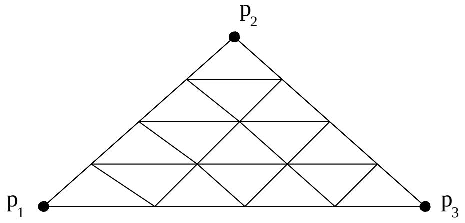



**Figure 3.2:** *A regular grid of a triangle with resolution* $m = 5$.

A surface $f$ is *represented* by a triangle $s$ of $S$ if the points

$$\mathbf{q}_{i,j,k} := \mathbf{g}_{i,j,k} + h_{i,j,k} \cdot \mathbf{v}_{i,j,k},$$

are on $f$, where

$$\mathbf{v}_{i,j,k} := \frac{i}{m-1}\mathbf{v}_1 + \frac{j}{m-1}\mathbf{v}_2 + \frac{k}{m-1}\mathbf{v}_3,$$

and $\mathbf{v}_i$, $i = 1, 2, 3$, are the vectors at the vertices of $s$. The surface $F$ *represented by a DDF* is the union of the surfaces represented by the triangles of $S$.

In order to obtain useful continuous surfaces $F$ the following three constraints have to be satisfied:

**Constraint 1.**
$\mathbf{v}_i$ is directed to the front side of $s$ and $|\mathbf{v}_i| \neq 0$.

**Constraint 2.**
The vectors $\mathbf{v}(\mathbf{p}, s)$ assigned to a vertex $\mathbf{p}$ of $S$ for every triangle $s$ incident to $\mathbf{p}$ are identical.

**Constraint 3.**
No two line segments of the type $\overline{\mathbf{rs}}$, $\mathbf{r} = \mathbf{p} - \mathbf{v}(\mathbf{p})$, $\mathbf{s} = \mathbf{p} + \mathbf{v}(\mathbf{p})$, $\mathbf{p} \in s$, intersect each other.

A vector $\mathbf{v}_i$ is directed to the front side of $s$ if the ray with origin $\mathbf{p}_i$ in direction of $\mathbf{v}_i$ traverses that one of the two halfspaces induced by the plane spanned by $s$ which is on the front side of $s$. *Constraint 1* guarantees that all displacement vectors $\mathbf{v}$ have a length $\neq 0$. If the vectors $\mathbf{v}_i$ would not be directed to the same side of the triangle, an interpolated vector of length 0 might exist (Fig. 3.3).

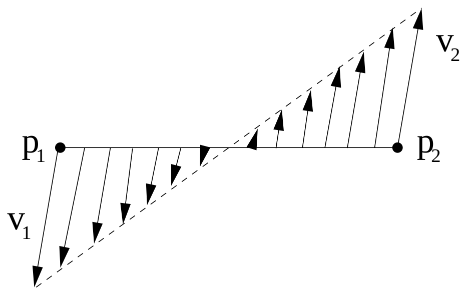



**Figure 3.3:** *The interpolation of the vectors* $\mathbf{v}_1$ *and* $\mathbf{v}_2$ *yields a zero vector in the middle of the line segment* $\overline{\mathbf{p}_1\mathbf{p}_2}$.

A canonical choice of the direction of the vectors $\mathbf{v}$ of the vertices of $S$ is to take the direction of the average of the unit normals of the incident triangles, possibly weighted according to some heuristics, such as the relative size of the incident angle or area.

*Constraint 1* of the DDF-definition is also a constraint on the arrangement of triangles incident to a vertex of $S$. The constraint implies that the intersection of the positive half-spaces induced by the triangles have a non-empty common intersection. The intersection can be efficiently calculated by an algorithm e.g. described in the book of Preparata et al. [Pre88].

*Constraint 2* demands that neighboring triangles have the same displacement vectors in their common vertices. This ensures a continuous transition of the vector field $\mathbf{V}$ over the edges of the support triangles $s$, as depicted in Fig. 3.4 (left). In Fig. 3.4 (right) the two neighboring support triangles use vectors $\mathbf{v}$ perpendicular to the triangle plane, so different vectors are assigned to the

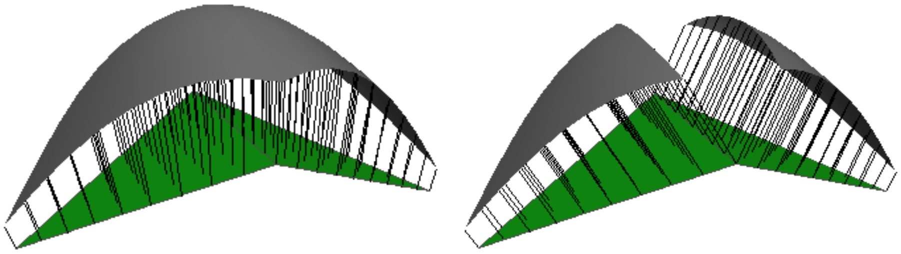


**Figure 3.4:** *Continuous vector field and surface transition over the edge of two neighboring triangles (left). Perpendicular displacement vectors on both triangles lead to a non-continuous vector field and surface transition (right).*

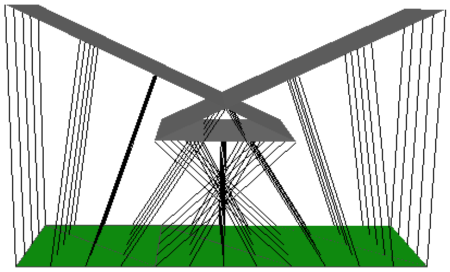


**Figure 3.5:** *Intersecting line segments along the vectors* $\mathbf{v}(\mathbf{p})$ *may cause foldovers of the represented surface.*

shared vertices. The continuous vector field induces a smooth surface $F$ while the surface is non-continuous otherwise.

The background of *Constraint 3* is to avoid foldovers of the represented surface $F$ (Fig. 3.5). This problem is discussed in more detail in section 4.3.

# Chapter 4

# Projections

In order to cut off stubbles of a displacement field by a tool, it has to be known which stubbles are reached by the tool in its current position. For a polygonal tool this task can be reduced to the problem of finding those raster points on triangles of the support $S$ whose associated line segments intersect the triangles $t$ of the tool. This is related to the problem of finding for a raster point $\mathbf{r}$ on a triangle $s$ of $S$ whether the line in direction of the associated vector $\mathbf{v}(\mathbf{r})$ hits the tool triangle $t$, and, if so, the point of intersection. We call this operation $\mathbf{V}$-shooting.

Another problem related to cutting stubbles is to determine for a point $\mathbf{q}$ on the tool a corresponding point $\mathbf{p}$ on $S$ for which $\mathbf{q}$ lies on the line in direction $\mathbf{v}(\mathbf{p})$ through $\mathbf{p}$. We call this operation $\mathbf{V}$-projection. The reason for this definition is that the operation can be understood as a generalization of conventional projections like parallel or perspective projection. $S$ plays the role of the image plane, and $\mathbf{V}$ defines the direction of projection. For conventional projections, a vector of projection can be assigned to every point on the image plane, too. In the case of parallel projection this vector is opposite to the direction of projection and thus is equal for all points of the image plane. For perspective projection, the vector of a point $\mathbf{p}$ of the image plane is a vector from $\mathbf{p}$ towards the view point $\mathbf{e}$ of the perspective projection.

For the conventional projections, every point in space, except the view point $\mathbf{e}$ of a perspective projection, has a unique image on the image plane. This does not hold for the "curved" $\mathbf{V}$-projection. There might be points in space which are traversed by several lines induced by $\mathbf{V}$ and points of $S$. However, in a sufficiently thin fringe around $S$ the unique-projection property can still be observed. An interesting problem is to determine such fringes for a given discrete displacement field.

Our interest in the unique-projection property comes from the fact that foldovers of the surface $F$ represented by a DDF are related to points in space with more than one image under **V**-projection. The reason is that in the case of a self-intersection of $F$, two different points $\mathbf{q}'$ and $\mathbf{q}''$ of $F$ fall together, $\mathbf{q} := \mathbf{q}' = \mathbf{q}''$. The points $\mathbf{p}'$ and $\mathbf{p}''$ from which $\mathbf{q}'$ and $\mathbf{q}''$ originate are **V**-projections of the same point $\mathbf{q}$.

The following two sections 4.1 and 4.2 are devoted to formulas and algorithms for the calculation of **V**-projection and **V**-shooting. In section 4.3, unique projection crusts are defined and an algorithm is presented for testing of this property. Based on these results, approaches to the choice of the displacement vector fields are presented in section 4.4.

## 4.1 V-Projection

Let $\mathbf{q}$ be a point and $s$ be a triangle in space with the vertices $\mathbf{p}_i$, $i = 1, 2, 3$, and vectors $\mathbf{v}_i$, $i = 1, 2, 3$, assigned to these vertices (Figure 4.1). The task of **V**-*projection* is to find a point $\mathbf{p}$ on $s$ so that the vector $\mathbf{v}(\mathbf{p})$ barycentrically interpolated from $\mathbf{v}_i$, $i = 1, 2, 3$, induces a line through $\mathbf{q}$.

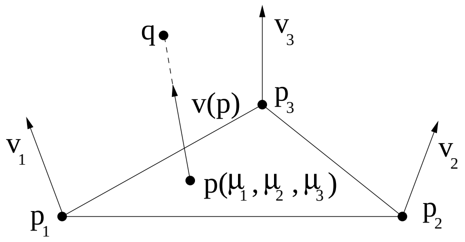



**Figure 4.1:** **V**-*projection of a point* $\mathbf{q}$ *onto a triangle* $s$ *in direction of the vector field defined on* $s$*. The task is to find the barycentric coordinates* $(\mu_1, \mu_2, \mu_3)$ *of the projected point* $\mathbf{p}$.

This condition is equivalent to finding a value $h$ and a point $\mathbf{p}$ on $s$ so that $\mathbf{p} + h\mathbf{v}(\mathbf{p}) = \mathbf{q}$. Let $(\mu_1, \mu_2, \mu_3)$, $\sum_{i=1}^{3} \mu_i = 1$, be the barycentric coordinates of $\mathbf{p}$. With

$$\mathbf{p} = \sum_{i=1}^{3} \mu_i \mathbf{p}_i, \;\; \mathbf{v}(\mathbf{p}) = \sum_{i=1}^{3} \mu_i \mathbf{v}_i,$$

the condition is

$$\sum_{i=1}^{3} \mu_i(\mathbf{p}_i + h\mathbf{v}_i) = \mathbf{q}, \;\; \sum_{i=1}^{3} \mu_i = 1,$$

Rewriting in matrix representation yields

$$\begin{pmatrix} \mathbf{p}_1 + h\mathbf{v}_1 & \mathbf{p}_2 + h\mathbf{v}_2 & \mathbf{p}_3 + h\mathbf{v}_3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} = \begin{pmatrix} \mathbf{q} \\ 1 \end{pmatrix}$$

Considered as a system of equations with the unknown $\mu_1$, $\mu_2$ and $\mu_3$, a solution only exists, if the three column vectors of the matrix and the vector on the right side are linearly dependent, or, equivalently, if the determinant of the $4\times4$-matrix with these four vectors as columns is 0,

$$\begin{vmatrix} \mathbf{p}_1 + h\cdot\mathbf{v}_1 & \mathbf{p}_2 + h\cdot\mathbf{v}_2 & \mathbf{p}_3 + h\cdot\mathbf{v}_3 & \mathbf{q} \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0.$$

This is a cubic polynomial equation in $h$ which can be solved analytically [Bro91]. The desired coordinates $(\mu_1, \mu_2, \mu_3)$ are finally obtained by solving the original linear system of equations for the resulting $h$ by Cramer's rule [Bro91].

In non-degenerating cases, the polynomial equation can have up to three solutions for $h$. This means that up to three **V**-projections of **q** on the plane spanned by $s$ exist. Those falling into $s$ are characterized by $\mu_i \geq 0$, $i = 1, 2, 3$.

## 4.2 V-Shooting

Let $s$ and $t$ be two triangles in space, and **p** be a point in $s$. The task of **V**-*shooting* is to find an intersection point **q** of the line through **p** along the vector $\mathbf{v}(\mathbf{p})$ with the plane spanned by $t$ (Figure 4.2), that is $\mathbf{p}+h\mathbf{v}(\mathbf{p}) = \mathbf{q}$ for a suitable value $h$.

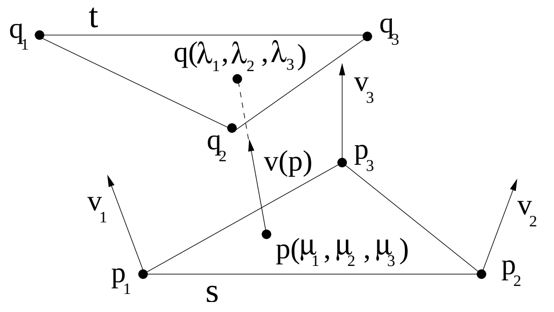



**Figure 4.2:** **V**-*shooting from a point* **p** *represented by barycentric coordinates* $(\mu_1, \mu_2, \mu_3)$ *on a triangle* $s$ *to a triangle* $t$ *in the direction* $\mathbf{v}(\mathbf{p})$. *The task is to find the barycentric coordinates* $(\lambda_1, \lambda_2, \lambda_3)$ *of the hit point* **q** *on* $t$.

With $(\mu_1, \mu_2, \mu_3)$ the barycentric coordinates of **p**, and $(\lambda_1, \lambda_2, \lambda_3)$ the unknown barycentric coordinates of **q** we get

$$\sum_{j=1}^{3} \lambda_j \mathbf{q}_j = \sum_{i=1}^{3} \mu_i(\mathbf{p}_i + h\mathbf{v}_i), \ \sum_{j=1}^{3} \lambda_j = 1, \ \sum_{i=1}^{3} \mu_i = 1.$$

Rewriting the equations for $\lambda_j$, $j = 1, 2, 3$, in matrix representation yields

$$\begin{pmatrix} \mathbf{p}_1 + h\mathbf{v}_1 & \mathbf{p}_2 + h\mathbf{v}_2 & \mathbf{p}_3 + h\mathbf{v}_3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{3} \mu_i(\mathbf{p}_i + h\mathbf{v}_i) \\ 1 \end{pmatrix}.$$

Considered as a system of equations with the unknown $\lambda_1$, $\lambda_2$, and $\lambda_3$, a solution only exists if the three column vectors of the matrix and the vector on the right side are linearly dependent, or, equivalently, if the determinant of the $4 \times 4$-matrix with these four vectors as columns is 0,

$$\begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \sum_{i=1}^{3} \mu_i(\mathbf{p}_i + h\mathbf{v}_i) \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0.$$

The equation system can be transformed into

$$\sum_{i=1}^{3} \left( \begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \mathbf{p}_i \\ 1 & 1 & 1 & 1 \end{vmatrix} + \begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \mathbf{v}_i \\ 1 & 1 & 1 & 0 \end{vmatrix} \cdot h \right) \mu_i = 0.$$

Solving for $h$ yields

$$h = \frac{-\sum_{i=1}^{3} \begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \mathbf{p}_i \\ 1 & 1 & 1 & 1 \end{vmatrix} \cdot \mu_i}{\sum_{i=1}^{3} \begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \mathbf{v}_i \\ 1 & 1 & 1 & 0 \end{vmatrix} \cdot \mu_i}. \tag{4.1}$$

This is a rational linear expression for the calculation of $h$. The desired coordinates $(\lambda_1, \lambda_2, \lambda_3)$ of the intersection point are obtained from the original system of equations for $\lambda_i$, with the calculated $h$, by Cramer's rule, for example

$$\begin{aligned} \lambda_1 &= \frac{\begin{vmatrix} \sum_{i=1}^{3} \mu_i(\mathbf{p}_i + h\mathbf{v}_i) & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix}}{\begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix}} \\ &= \frac{\sum_{i=1}^{3} \mu_i \begin{vmatrix} \mathbf{p}_i & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix} + h \cdot \sum_{i=1}^{3} \mu_i \begin{vmatrix} \mathbf{v}_i & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix}}{\begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix}}. \end{aligned} \tag{4.2}$$

$\lambda_2$ has a similar formula. $\lambda_3$ is calculated as $\lambda_3 = 1 - \lambda_1 - \lambda_2$.

Because of the resulting rational expressions, the possibility of a zero denominator has to be discussed. The determinant $|\mathbf{q}_1\mathbf{q}_2\mathbf{q}_3|$ becomes 0, if and only if the vertices $\mathbf{q}_i$, $i = 1, 2, 3$, are in a common plane through the origin of the coordinate system (figure 4.3). If this configuration occurs in an implementation, the two triangles $s$ and $t$ are translated virtually to some amount perpendicularly to this plane, that is in direction $(\mathbf{q}_2 - \mathbf{q}_1) \times (\mathbf{q}_3 - \mathbf{q}_1)$. This correction does not influence the result because the solution is expressed relatively to the involved triangles.

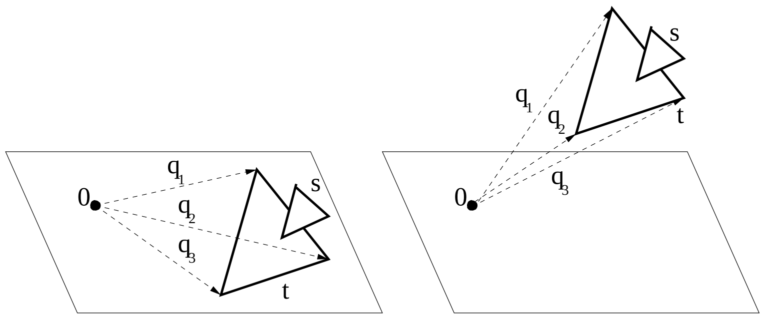



**Figure 4.3:** *The triangles $s$ and $t$ are translated simultaneous to guarantee the linear independence of the vectors $\mathbf{q}_1$, $\mathbf{q}_2$ and $\mathbf{q}_3$.*

In the same way it can be avoided that the $4 \times 4$-determinants in the denominator of the rational expression of $h$ become 0.

In comparison to $\mathbf{V}$-projection, $\mathbf{V}$-shooting is easier since $h$ can be calculated by just evaluating a rational linear expression, while a cubic polynomial equation has to be resolved for $\mathbf{V}$-projection. In section 5 the rational linear expressions will be used to establish a fast incremental algorithm for cutting of the stubbles of a support triangle $s$ by a tool triangle $t$.

## 4.3 Unique Projection Crusts

We now formalize the notion of a fringe or crust around the support $S$ of a DDF. The *$h$-crust*, $h$ a real number, of a triangle $s$ of $S$ with respect to $\mathbf{V}$ is defined as the point set obtained from the union of all line segments $\overline{\mathbf{pq}}$, where $\mathbf{p} \in s$ and $\mathbf{q} = \mathbf{p} + h \cdot \mathbf{v}(\mathbf{p})$. The union of the $h$-crust and the $-h$-crust is denoted as $\pm h$-crust. By this definition, the shape and thickness of the crusts are determined by the lengths of the vectors $\mathbf{v}(\mathbf{p})$ which do not need to be normalized.

An alternative definition can be given based on so-called $h$-offsets of $S$. The *$h$-offset* $S'$ of $S$, $h$ a real number, is a mesh which is obtained from $S$ by replacing vertices $\mathbf{p}$ of $S$ with $\mathbf{p}' = \mathbf{p} + h \cdot \mathbf{v}(\mathbf{p})$ (Figure 4.4). With this definition, the $h$-crust can also be seen as the union of all $h'$-offsets with $0 \leq h' \leq h$. Constraint 1 of chapter 3 implies that the crust is "volumetric" without degenerations to zero thickness.

An $h$-crust of a triangle $s$ of $S$ has the *unique-projection property*, if no two line segments of the type $\overline{\mathbf{pr}}$, $\mathbf{r} = \mathbf{p} + h\mathbf{v}(\mathbf{p})$, $\mathbf{p} \in s$, intersect each other. A $\pm h$-crust has the unique-projection property, if the $h$-crust and the $-h$-crust have the unique-projection property. If the crusts of all triangles $s$ of $S$ have the unique-projection property, and the crusts of all pairs of non-adjacent support triangles have an empty intersection, the crust of $S$ has the unique-projection property. With this definition, constraint 3 of chapter 3 means that the $\pm 1$-crusts of the support triangles of a DDF must have the unique-projection property.

The following investigations focus on $h$-crusts for $h > 0$. The case $h < 0$ can be treated analogously.

A point $\mathbf{q}$ in a $h$-crust of a triangle $s$ with unique-projection property has exactly one image on $s$ with respect to $\mathbf{V}$-projection. It has an image, because it is on one of the line segments defining the $h$-crust and the directions of $\mathbf{V}$-projection. Because of the unique projection property there is not more than one line segment of this property.

Another observation is that if an $h$-crust, $h > 0$, has the unique-projection property, then the $h'$-crusts with $0 \leq h' < h$ have the unique-projection property, too. In particular, the 0-crust, which is the triangle $s$, trivially has the unique projection property. We are now interested in the biggest $h > 0$ for which the $h$-crust has the unique-projection property.

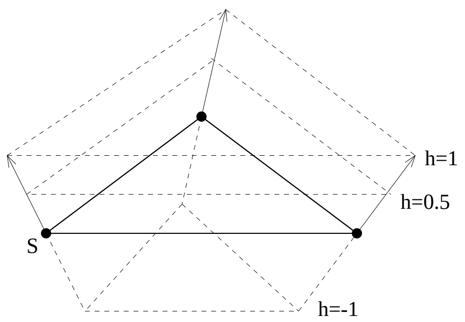



**Figure 4.4:** *The $h$-offsets of a triangle $s$ for $h = -1$, 0.5, 1.*

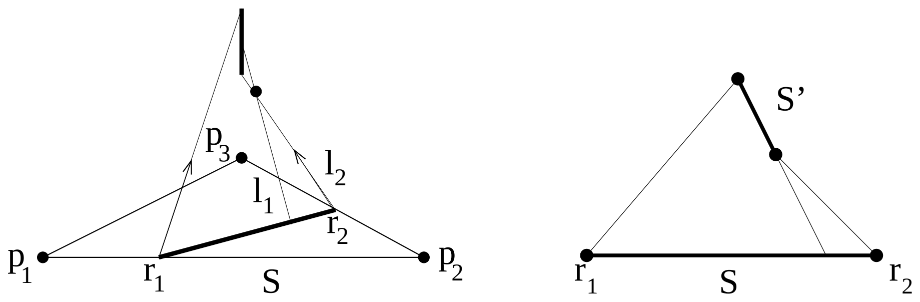



**Figure 4.5:** *Two intersecting line segments $l_1$ and $l_2$ above a triangle (left), and a 2D-configuration corresponding to this configuration (right).*

If $h$ is continuously increased starting with 0, an intersection between two line segments $l_1$ and $l_2$ may occur (Figure 4.5, left). In this case, $l_1$ and $l_2$ span a common plane. The plane intersects $s$ in a line (drawn fat in the figure). The line intersects the boundary of $s$ in two points $\mathbf{r}_1$ and $\mathbf{r}_2$. The vectors of the two intersecting line segments are linear interpolations of the $h$-scaled vectors at $\mathbf{r}_1$ and $\mathbf{r}_2$ (in figure 4.5 it accidentally happened that $l_2$ goes through $\mathbf{r}_2$, too). They are in the common plane, too. Thus, we can restrict the discussion to the 2D-case.

In the 2D-case (Figure 4.5, right) it can be shown that the condition for an occurrence of intersecting line segments is that the line segment corresponding to $s'$ (the line segments corresponding to $s$ and $s'$ are drawn fat) is in parallel to a connecting line segment for some $h$. This means that a connecting line segment exists which is on a common line with $s'$. The direction of the connecting line segment is a barycentric interpolation of the vectors $h \cdot \mathbf{v}_i$ at $\mathbf{r}_i$, $i = 1, 2$.

Thus, in the 3D-case, we have to look for the smallest $h$ for which a barycentric combination of the $h \cdot \mathbf{v}_i$ and two edge vectors of the triangle spanned by $\mathbf{p}_i + h \cdot \mathbf{v}_i$, $i = 1, 2, 3$, are co-linear:

$$\begin{gathered} h_0 := \min\{h > 0 \mid \\ \left| \begin{matrix} \sum_{i=1}^{3} \mu_i \mathbf{v}_i & \Delta\mathbf{p}_{12} + h\Delta\mathbf{v}_{12} & \Delta\mathbf{p}_{13} + h\Delta\mathbf{v}_{13} \end{matrix} \right| = 0, \\ \sum_{i=1}^{3} \mu_i = 1, \mu_i \geq 0\}, \end{gathered}$$

where

$$\begin{aligned} \Delta\mathbf{p}_{12} = \mathbf{p}_2 - \mathbf{p}_1,\ \Delta\mathbf{p}_{13} = \mathbf{p}_3 - \mathbf{p}_1 \\ \Delta\mathbf{v}_{12} = \mathbf{v}_2 - \mathbf{v}_1,\ \Delta\mathbf{v}_{13} = \mathbf{v}_3 - \mathbf{v}_1. \end{aligned}$$

This is a linear optimization problem with non-linear constraints. The exact solution can be obtained as follows.

First, $\mu_3$ is eliminated in the equation for $h$, using $\mu_3 = 1 - \mu_1 - \mu_2$. Elimination of $\mu_3$ implies that the constraint $\mu_3 > 0$ is replaced with $1 - \mu_1 - \mu_2 \geq 0$. The resulting optimization problem is solved by first determining the minimum without the constraints on the $\mu_i$. For this purpose, the quadratic equation for $h$ is solved. The resulting explicit expression for $h$ is minimized by setting its partial derivatives with respect to $\mu_1$ and $\mu_2$ to zero, and solving the two resulting equations analytically for $\mu_1$ and $\mu_2$.

The next step is to look for minima on the boundary of the triangle defined by the constraints $\mu_i \geq 0$, $i = 1, 2$, and $1 - \mu_1 - \mu_2 \geq 0$. For this purpose, $\mu_1 = 0$, $\mu_2 = 0$, and $\mu_2 = 1 - \mu_1$ are put one after the other into the equation of $h$. In all three cases, the resulting equation, which now depends on just one variable, is solved as before.

The resulting values $\mu_1$ and $\mu_2$ which satisfy the constraints are put into the formula of $h$. The smallest resulting positive value is taken for $h_0$.

In practice, an approximative solution usually is sufficient. It can be obtained by discretizing the triangle defined by the constraints $\mu_1 \geq 0$, $\mu_2 \geq 0$, $1 - \mu_1 - \mu_2 \geq$

0 by a regular grid. For every grid vertex $(i/n, j/n)$, $0 \leq j, j \leq n$, $i + j \leq n$, the quadratic equation for $h$ is resolved. Among the positive solutions, the smallest one is taken as approximate solution.

The algorithm for searching for the biggest $h$ which satisfies the unique-projection property can also be used for checking constraint 3 for a given DDF. If $h \geq 1$ is found for every triangle of $S$, constraint 3 will be satisfied.

## 4.4 Choice of the Vector Field

According to the definitions in section 4.3, the thickness of the crusts is determined by the lengths of the vectors at the vertices. The goal should be a crust of optimal thickness in which foldovers do not occur. This means that according to constraint 3 of the DDF-definition, the $\pm 1$-crust has to have the unique-projection property. In many cases, this property limits the possible lengths of the vectors. In particular, the worse side of the surface determines the thickness of the crust.

We do not know a natural formal definition of optimal thickness. The reason is that the shape of an $h$-crust of a triangle $s$ may vary considerably dependent on the length of vectors at the vertices of $s$. We propose the following heuristic algorithm for construction of a reasonable crust of a triangle mesh $S$, if the direction of the vectors at the vertices of the mesh are given, and just their length has to be determined. For each triangle the vectors $\mathbf{v}_i$, $i = 1, 2, 3$, are normalized to length 1. Then the height $h_0$ of section 4.3 is calculated for both sides of the triangle. If the minimum of both $h_0$-values is finite, the $\mathbf{v}_i$ are scaled by the minimum. Otherwise an arbitrary length, sufficient for the application, is chosen. Among the vectors of different lengths which may afterwards occur at a mesh vertex to which several triangles are incident, we take the shortest one. In this manner the vertex vectors of some triangles may be shortened. For those triangles the $h_0$-values are calculated again. If the minimum of the new $h_0$-values is less than 1, the vectors at the vertices are reduced by the factor given by the minimum. Then the step of adaptation on the whole mesh $S$ is iterated again. The iteration is continued until $h_0 \geq 1$ is achieved everywhere.

For DDFs of polygonal chains in the plane, an iteration is not required. For meshes in space we do not know whether the iteration is necessary or not. We leave this question as an open problem and recommend to implement the iteration.

# Chapter 5

# Depth Buffer Algorithm on Discrete Displacement Fields

The classical depth buffer solves the problem of visibility calculation during rendering of a three-dimensional scene of e.g. triangles [Fol90]. It operates on a so-called depth buffer which is an array of the same resolution as the raster image to be calculated. Its array elements correspond to the pixels of the image and contain the depth of the scene at the pixels. Depth can for instance be the distance of the point of the scene displayed by a pixel from the image plane along the line of projection. Initially, the depth buffer is set to infinity. The triangles of the scene are processed one after the other. For each triangle the pixels covered by the triangle are determined, that is, the triangle is located on the image plane. Furthermore, the depth of the triangle at every pixel is calculated. Then a stored depth value will be replaced with a calculated depth value, if it is bigger than the calculated value. In this case the triangle is rendered as visible in the pixel of the image, too. Figure 5.1, left, gives an illustration.

In our case, the role of the depth buffer is taken over by the DDF (Figure 5.1, right). The pixels correspond to the raster points of the support triangles, the depth value corresponds to the vector length at every raster point represented indirectly by the height function $h$, and the direction of projection corresponds to the direction of the displacement vector field $\mathbf{V}$. The scene $T$ is the tool or a solid swept by the tool. We assume that it is bounded by triangles $t$, too. The task is to find the minimum of the height values of the intersection points of the surface $T$ with the displacement vector field $\mathbf{V}$ and the stored height values $h$.

The DDF-depth-buffer algorithm presented in the following proceeds analogously to the classical depth-buffer algorithm. The triangles of $T$ are processed one after the other. Two tasks have to be solved for every triangle $t$: *localization* of $t$ on $S$, and calculation and updating of the depth values, called *cutting* in the following. In comparison to the classical algorithm the situation is more complicated for DDFs because the displacement vectors can be much more irregular than in the case of parallel or perspective projection. In the following we give solutions for the problem of localization and the problem of cutting.

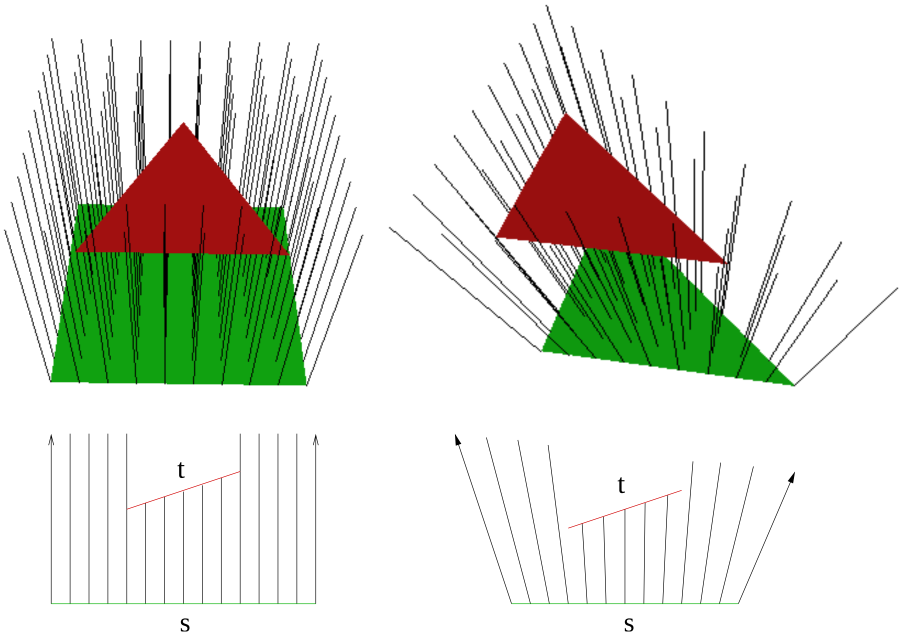



**Figure 5.1:** *A classical depth buffer application (left) in comparison to a DDF-depth-buffer application (right). In both cases a set of "stubbles" has to be cut by a triangle. In the classical application with parallel projection the stubbles are in parallel to each other, while in the case of a DDF the mutual orientation can be more general.*

## 5.1 Localization

The support mesh $S$ of a DDF may consist of many support triangles $s$, and normally just a small subset of them is affected by a tool triangle $t$. The task of localization is to find all triangles $s$ affected by $t$. For all affecting pairs $(s, t)$ a fast cutting algorithm is invoked to update the height values on $s$. The cutting algorithm is subject of the next section.

A support triangle $s$ can only be affected by a cut triangle $t$, if $t$ intersects the $\pm 1$-crust of $s$. The reason is that the surfaces represented by a DDF have to lie within the $\pm 1$-crust. Thus, the task of localization leads to the new task of finding all support triangles with a $\pm 1$-crust intersected by $t$. We propose a grid-based hashing algorithm for this search. For this purpose we determine a rectangular box containing all $\pm 1$-crusts of the DDF. The box is subdivided into the cells of a regular grid $G$. For every triangle $s$ of $S$ the grid cells of $G$ are determined which intersect the $\pm 1$-crust of $s$. $s$ is inserted into lists of triangles of these cells which contain the intersecting triangles.

The grid cells with non-empty list are stored in a one-dimensional hash table. As key, a linear combination of the indices of the grid cell is used. An advantage of this approach are less expected storage requirements than for the representation of the grid cells in a three-dimensional array. The reason is that the crust of a DDF usually is thin, so the majority of the cells of $G$ remains empty. A further advantage of the hashing approach is that the grid can easily be extended without changing the data structure, if the DDF or its crust are enlarged. An extension of an array beyond its initially defined index range is usually not supported by programming languages. Trees of bounding boxes which might be an alternative [Got96] are also more complex to handle in a dynamic environment than the hashing structure.

The data structure is built up in a preprocessing phase during which all support triangles are processed. In the subsequent query phase, the support triangles $s$ affected by an arbitrary query triangle $t$ are found by determining a set of cells of $G$ covering $t$. The grid cells of the set are checked for membership in the preprocessed data structure by hashing. For the cells successfully tested, the crusts of the triangles stored in their lists are checked for an intersection with $t$. For the intersecting ones, the cutting operation of section 5.2 is performed.

We have not yet explained how to calculate the set of cells of $G$ into which a triangle $s$ is inserted in the preprocessing phase. The probably fastest but least precise method is to use the axes-parallel bounding box of the crust of $s$ and find the grid cells intersected by it. The extreme vertices of the crust are induced by the vectors of $\mathbf{v}$ at the vertices of $s$. A better approximation can be achieved by an explicit scan-conversion of the crust into grid cells, which, however, is more complicated to implement. As a compromise we suggest to subdivide $s$ into subtriangles, and the crust over each subtriangle into height segments, in order to obtain a decomposition of the crust into sufficiently small sub-volumes (figure 5.2). For the sub-volumes, the grid cells intersecting the axes-parallel bounding boxes of the subvolumes are determined (in figure 5.2 one of the bounding boxes is indicated), and $s$ is inserted into those cells. By this approach, calculation time can be traded against precision of the set of grid cells.

For the localization of $t$ the indices of a set of cells of $G$ covering $t$ are calculated. We do this analogously to the localization of the crust. The procedure, however, is more simple since the thickness of a crust has not to be considered. This means that a triangle is subdivided into congruent subtriangles whose bounding boxes are used to determine the relevant grid cells as before.

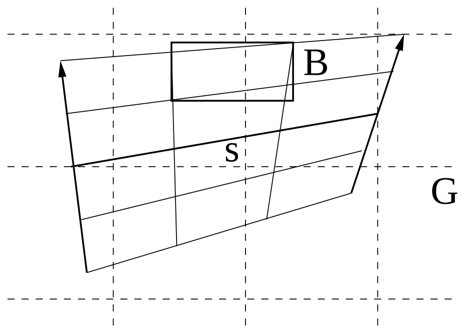



**Figure 5.2:** *Associating a support triangle $s$ and its $\pm 1$-crust to a grid $G$. The crust is divided into smaller parts and bounding boxes around these parts are combined with the grid.*

Figure 5.3 gives a 2D example for the localization process. In a precalculation step the three support triangles $s_1$, $s_2$ and $s_3$ are associated with the grid cells intersecting their $\pm 1$-crust, for example $s_3$ with the cells $D1$, $D2$, $D3$, $E2$ and $E3$. The cut triangle $t$ intersects the three grid cells $C1$, $C2$ and $D1$. Since the support triangles $s_2$ and $s_3$ are associated with these three grid cells they are reported as affected. In fact $s_3$ causes an overhead because it is indeed not affected by $t$.

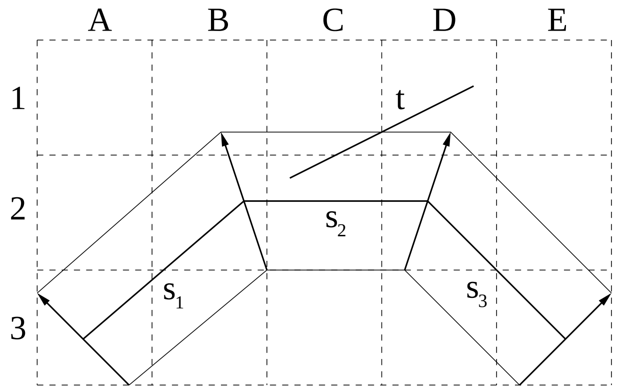
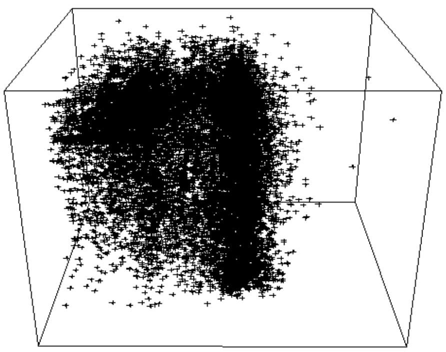



**Figure 5.3:** *Using a grid to find pairs $(s, t)$ of triangles where $t$ intersects the $\pm 1$-crust of $s$.*

Another issue is the choice of the resolution of grid $G$. A suitable edge length of the grid cell can be determined by using a so-called *triangle extension space* $E(S)$. $E(S)$ is three-dimensional. A triangle $s$ of $S$ is represented in $E(S)$ by a point $\mathbf{e}$. The coordinates of $\mathbf{e}$ are the x-, y-, and z-extension of $s$. If these points in $E(S)$ form a compact cluster - that can be expected for a reasonably triangulated surface $S$ - the center point of this cluster induces a suitable grid cell size. The coordinates of the center point define an average value of the triangle extensions and can be used as reasonable grid cell length.

If the points in $E(S)$ do not form a compact cluster a cluster analysis might be performed which determines more than one cluster center. For each cluster center, a separate grid hash structure may be preprocessed into which the triangles are distributed according to their extensions. A query has to be performed on each of those structures. This may slow down the query processing but saves space. The number of hash structures to be processed for a query can be reduced, if clusters in $E(S)$ correspond to triangles restricted to certain parts of $S$. In this case an additional spatial subdivision may be stored which refers to the hash structures relevant for each of its cells. We have not implemented these ideas since the majority of the meshes which we got from applications could be handled reasonably by one uniform grid.

Figure 5.4 visualizes the triangle extensions of two well-known benchmark data sets [Sta03], the Stanford Bunny (see Figure 6.13) and the Stanford Buddha (Figure 6.15). It can be noticed that, apart from very few exceptions, the points are concentrated into a cluster. The extension of the bounding boxes of the Bunny and the Buddha shown in the figure are $[0, 5.33] \times [0, 4.20] \times [0, 4.76]$ and $[0, 2.66] \times [0, 2.62] \times [0, 2.58]$, respectively. The means and standard deviations are $[2.25, 2.21, 2.06]$ and $[0.88, 0.97, 0.93]$ for the Bunny, and $[0.52, 0.53, 0.50]$ and $[0.29, 0.28, 0.26]$ for the Buddha. The extensions of the original meshes are $[-95, 60] \times [33, 187] \times [-62, 59]$ for the Bunny and $[-46, 35] \times [50, 248] \times [-47, 34]$ for the Buddha (all values in $10^{-3}$ length units). If the mean values are taken as grid cell sizes, grids of resolution $70 \times 100 \times 59$ and $166 \times 563 \times 162$, respectively, result.



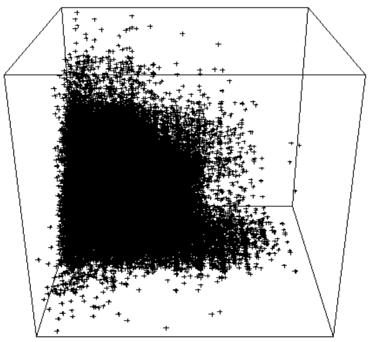


**Figure 5.4:** *Point sets of the Stanford Bunny (left) and the Stanford Buddha (right) in the triangle extension space.*

## 5.2 Cutting

Given a support triangle $s$ and a cutting triangle $t$, the task of cutting is to calculate the height values corresponding to $t$ at those raster vertices of $s$ onto which $t$ **V**-projects, and to update the height values stored for $s$ by the minimum of the found and stored height values.

The basic strategy of the following algorithm is to process the raster of the triangle row by row, in one of the three possible directions. At every raster point $\mathbf{p}$ of $s$ the **V**-shooting operation described in section 4.2 is applied yielding the barycentric coordinates $\lambda_1$, $\lambda_2$, $\lambda_3$ of the intersection point of the ray starting at $\mathbf{p}$ in direction $\mathbf{v}(\mathbf{p})$ with the plane spanned by $t$, and the height value $h$ of the intersection point with the plane. The triangle $t$ itself is hit, if all the $\lambda$-values are $\geq 0$. In this case, $h$ is compared to the value $h(\mathbf{p})$ stored at $\mathbf{p}$. If $h < h(\mathbf{p})$ then $h(\mathbf{p})$ is updated with $h$.

Our goal is to reduce the number of arithmetic operations of the algorithm. The reason is that the number of **V**-shootings is considerably high since it depends on the resolution of the rasterization of $s$ which might be high in order to achieve a given precision. We use three approaches in order to reach the goal, pre-calculation, identification of common sub-expressions, and incremental evaluation of a sequence of operations.

A closer look to the formulas (4.1) and (4.2) of **V**-shooting shows that the determinants are independent of the $s$-raster position $(\mu_1, \mu_2, \mu_3)$. Thus, they have to be evaluated just once for every pair $s$ and $t$. With the pre-calculated values

$$a_i = \begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \mathbf{p}_i \\ 1 & 1 & 1 & 1 \end{vmatrix}, \quad b_i = \begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \mathbf{v}_i \\ 1 & 1 & 1 & 0 \end{vmatrix} \tag{5.1}$$

$$c_i = \frac{\begin{vmatrix} \mathbf{p}_i & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix}}{\begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix}}, \quad d_i = \frac{\begin{vmatrix} \mathbf{v}_i & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix}}{\begin{vmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 \end{vmatrix}}, \quad i = 1, 2, 3,$$

the formulas (4.1) and (4.2) can be rewritten as

$$h = \frac{\sum_{i=1}^{3} a_i \cdot \mu_i}{\sum_{i=1}^{3} b_i \cdot \mu_i}, \quad \lambda_1 = \textstyle\sum_{i=1}^{3} c_i \cdot \mu_i + h \cdot \sum_{i=1}^{3} d_i \cdot \mu_i. \tag{5.2}$$

The formula of $\lambda_2$ has the same structure, with different coefficients $a_i$, $b_i$, $c_i$ and $d_i$. $\lambda_3$ is calculated as $\lambda_3 = 1 - \lambda_1 - \lambda_2$.

Time can be saved during the pre-calculation of the coefficients (5.1) by noticing that most of the determinants have common sub-determinants, like

$$\begin{vmatrix} q_{2,y} & q_{3,y} \\ q_{2,z} & q_{3,z} \end{vmatrix}, \quad \begin{vmatrix} q_{2,x} & q_{3,x} \\ q_{2,z} & q_{3,z} \end{vmatrix}, \quad \begin{vmatrix} q_{2,x} & q_{3,x} \\ q_{2,y} & q_{3,y} \end{vmatrix}.$$

Thus, a speed-up is achieved by evaluating a suitable set of these determinants and using the resulting values in order to replace common subexpressions in later calculations. A reasonable approach is to develop the determinants of $a_i$ and $b_i$ with respect to the last row, since three of the resulting eight sub-determinants can then be used in the two further expressions for $c_i$ and $d_i$. The resulting $3 \times 3$-determinants are developed with respect to one of the columns. For instance, for the three mentioned sub-determinants which define the nominators and denominators of $c_i$ and $d_i$ a development with respect to the first column is reasonable, since the resulting $2 \times 2$-subdeterminants occur several times in the three $3 \times 3$-determinants.

Row-wise processing of the $s$-raster points means to fix one of the barycentric coordinates $\mu_i$, $i = 1, 2, 3$. Let us fix $\mu_3$ and consider two consecutive raster points $(\mu_1, \mu_2, \mu_3)$ and $(\mu_1', \mu_2', \mu_3') := (\mu_1 + \frac{1}{m-1}, \mu_2 - \frac{1}{m-1}, \mu_3)$ on such a row. $m$ is the resolution of the raster. **V**-shooting at these points yields

$$\begin{aligned}\lambda_1 &= c_1\mu_1 + c_2\mu_2 + c_3\mu_3 + h \cdot (d_1\mu_1 + d_2\mu_2 + d_3\mu_3) \\ &=: v_1 + h \cdot f_1\end{aligned}$$

and

$$\begin{aligned}\lambda_1' &= c_1\mu_1' + c_2\mu_2' + c_3\mu_3' + h' \cdot (d_1\mu_1' + d_2\mu_2' + d_3\mu_3') \\ &= c_1\left(\mu_1 + \frac{1}{m-1}\right) + c_2\left(\mu_2 - \frac{1}{m-1}\right) + c_3\mu_3 \\ &\quad +h'\left[d_1\left(\mu_1 + \frac{1}{m-1}\right) + d_2\left(\mu_2 - \frac{1}{m-1}\right) + d_3\mu_3\right] \\ &= c_1\mu_1 + c_2\mu_2 + c_3\mu_3 + \frac{c_1 - c_2}{m-1} + h' \cdot \left(d_1\mu_1 + d_2\mu_2 + d_3\mu_3 + \frac{d_1 - d_2}{m-1}\right) \\ &= v_1 + \frac{c_1 - c_2}{m-1} + h' \cdot \left(f_1 + \frac{d_1 - d_2}{m-1}\right) \\ &=: v_1' + h' \cdot f_1'\end{aligned}$$

This implies that $v_1'$ and $f_1'$ can be calculated from $v_1$ and $f_1$ by adding a constant increment which can be pre-calculated and which is independent from the particular row. Thus, given $\lambda_1$, $\lambda_1'$ can be calculated by three incremental additions and one multiplication, under the assumption that $h'$ is already calculated.

For $h$ and the succeeding value $h'$ the following holds

$$h = \frac{a_1\mu_1 + a_2\mu_2 + a_3\mu_3}{b_1\mu_1 + b_2\mu_2 + b_3\mu_3} =: \frac{n_h}{d_h}$$

$$
\begin{aligned}
h' &= \frac{a_1\mu_1' + a_2\mu_2' + a_3\mu_3'}{b_1\mu_1' + b_2\mu_2' + b_3\mu_3'} \\
&= \frac{a_1\left(\mu_1 + \frac{1}{m-1}\right) + a_2\left(\mu_2 - \frac{1}{m-1}\right) + a_3\mu_3}{b_1\left(\mu_1 + \frac{1}{m-1}\right) + b_2\left(\mu_2 - \frac{1}{m-1}\right) + b_3\mu_3} \\
&= \frac{a_1\mu_1 + a_2\mu_2 + a_3\mu_3 + \left(\dfrac{a_1 - a_2}{m-1}\right)}{b_1\mu_1 + b_2\mu_2 + b_3\mu_3 + \left(\dfrac{b_1 - b_2}{m-1}\right)} \\
&= \frac{n_h + \left(\dfrac{a_1 - a_2}{m-1}\right)}{d_h + \left(\dfrac{b_1 - b_2}{m-1}\right)} \\
&=: \frac{n_h'}{d_h'},
\end{aligned}
$$

Thus, $n_h'$ and $d_h'$ can be calculated from $n_h$ and $d_h$ by adding a constant value, and hence $h'$ can be calculated from $h$ by two incremental additions and one division.

In summary with this strategy 10 additions/subtractions and 3 multiplications/divisions are required for every step of the algorithm for the calculation of $\lambda_1$, $\lambda_2$, $\lambda_3$, and $h$: two additions and one division for the calculation of $h$, three additions and one multiplication for $\lambda_1$, again three additions and one multiplication for $\lambda_2$, and two subtractions in order to calculate $\lambda_3$ by $\lambda_3 = 1 - \lambda_1 - \lambda_2$. For updating the height array $h$ of $s$, one memory access is required in order to read the stored value at the current raster position. If the calculated height value is smaller than the stored one, a second memory access will be necessary in order to overwrite the old value. Checking the three $\lambda$-values for $\geq 0$ and comparing the new and old $h$-values requires four comparisons.

Figure 5.6 shows timings for an implementation of the algorithm. Times have been taken on a PC with AMD Athlon XP1700 processor and 1 GB RAM. The programming language has been Java 1.3i. The program has been applied to the configuration shown in figure 5.5 at different resolutions of the raster, $m = 2\ 000, 4\ 000, 6\ 000, 8\ 000$ and $10\ 000$. The number $n$ of raster points processed at resolution $m$ is $n := m(m+1)/2$. The measurements concern just cutting, localization has not been taken into account. The processing time seems to be linear in $n$ since the scale of the horizontal axis of figure 5.6 is quadratic in the resolution. The extrapolation of the straight curve to resolution 0 would intersect the vertical axis above the coordinate origin because of the constant initialization time required for pre-calculation of coefficients, like $a_i$, $b_i$, $c_i$, and

$d_i$. In the configuration of figure 5.5 all elements of the height array have to be updated. Approximately 100 ns are required per raster point.

Timings for the entire DDF depth-buffer algorithm are presented in section 6.

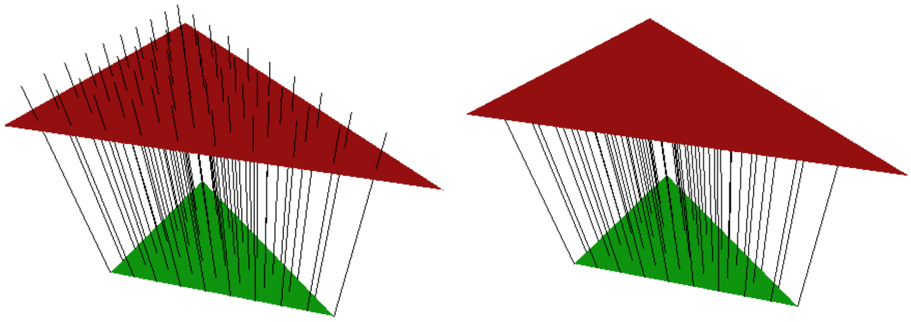


**Figure 5.5:** *The vectors on the green support triangle are cut at the red triangle. The red triangle is chosen large enough to intersect all the vectors.*

If the triangle $t$ is small in comparison to the support $s$ or if $t$ intersects the crust of $s$ only partially, then the number of affected raster positions on $s$ is small, and it might be worthwhile to find a possibility to reduce the number of rays to be shot. According to figure 5.7 a triangle $t$ projects on a point set $t'$ on the support triangle $s$, and only raster positions inside $t'$ are affected by the cutting operation. A difficulty is that $t'$ is not necessarily a triangle since its boundary might be curved.

The following modification of the cutting algorithm again processes the raster points of $s$ row by row. Every row is checked for an intersection with $t'$. If the row does not intersect $t'$, the algorithm continues with the next row. If the row intersects $t'$, the $\mathbf{V}$-shooting operations are restricted to the intervals of the row inside $t'$. The intervals are defined by the three inequalities $\lambda_i(\mu_1, \mu_2, \mu_3) \geq 0$, $i = 1, 2, 3$. If we again take rows of constant $\mu_3$ for illustration, and take into account that $\mu_2 = 1 - \mu_1 - \mu_3$, the inequalities have just one unknown, $\mu_1$. Let us have a closer look at the inequality $\lambda_1(\mu_1) \geq 0$, the other inequalities can be solved analogously. According to (5.2),

$$\lambda_1 = \frac{\sum_{i=1}^3 c_i\mu_i \sum_{i=1}^3 b_i\mu_i + \sum_{i=1}^3 a_i\mu_i \sum_{i=1}^3 d_i\mu_i}{\sum_{i=1}^3 b_i\mu_i} \geq 0.$$

For a constant $\mu_3$ and $\mu_2 = 1 - \mu_1 - \mu_3$, the sums in this formula can be rewritten as

$$\sum_{i=1}^{3} a_i\mu_i \quad = \quad a_1\mu_1 + a_2\mu_2 + a_3\mu_3$$

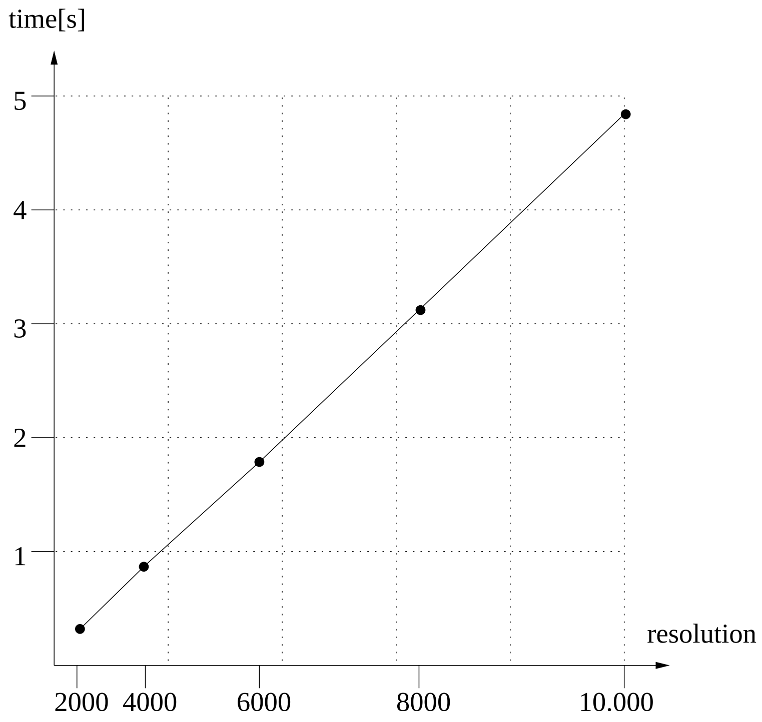



**Figure 5.6:** *Calculation times for cutting all the vectors of a support triangle. The horizontal axis is scaled quadrically in the resolution, that is, linear to the number of raster points on the support triangle. Evidently, the calculation time increases linearly in the number of raster points.*

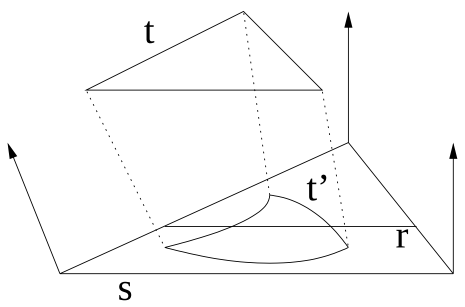



**Figure 5.7:** *The projection $t'$ of a triangle $t$ on $s$ is not necessarily a triangle.*

$$\begin{aligned}
&= a_1\mu_1 + a_2(1-\mu_1-\mu_3) + a_3\mu_3 \\
&= (a_1-a_2)\mu_1 + (a_3-a_2)\mu_3 - a_2 \\
&=: a_f\mu_1 + a_c, \\
\sum_{i=1}^{3} b_i\mu_i &=: b_f\mu_1 + b_c, \\
\sum_{i=1}^{3} c_i\mu_i &=: c_f\mu_1 + c_c, \\
\sum_{i=1}^{3} d_i\mu_i &=: d_f\mu_1 + d_c.
\end{aligned}$$

Replacing the sums in the formula of $\lambda_1$ with the new expressions, and rearranging for $\mu_1$ yields

$$\begin{aligned}
\lambda_1 &= \frac{(c_fb_f + a_fd_f)\mu_1^2 + (c_cb_f + c_fb_c + a_cd_f + a_fd_c)\mu_1 + (c_cb_c + a_cd_c)}{b_f\mu_1 + b_c} \\
&=: \frac{g_1\mu_1^2 + g_2\mu_1 + g_3}{b_f\mu_1 + b_c} \geq 0.
\end{aligned}$$

The endpoints of the intervals inside $t'$ are determined by the roots of the nominator of this expression for $\lambda_1$, and the analogous expressions for $\lambda_2$ and $\lambda_3$. The roots are sorted, and the intervals in-between two consecutive values are checked for the signs of $\lambda_i$, $i = 1, 2, 3$. The raster points of the intervals with positive signs are incrementally evaluated as before.

# Chapter 6

# Conversion of Meshes into DDF-Representation

Usually the surfaces of workpieces are not given in a DDF-representation, so a conversion into a DDF-representation is necessary in order to apply the DDF-based algorithms. A widely used surface representation which allows a straightforward conversion is the representation by triangular meshes. In this case, the given mesh $F_d$ can be immediately used as support surface $S$. If pseudo-normal vectors are supplied with the vertices of $F_d$, they can be used to fix the direction of the displacement vector field $\mathbf{V}$, if not, a pseudo-normal vector can be calculated by averaging the normals of incident triangles as indicated in figure 6.1. For simple surfaces, the height-values $h$ may be set to a suitable constant value which is simplified, if the vertex displacement vectors are unified to an equal length which guarantees the unique-projection property. A more advanced initialization is to use the algorithm presented in section 4.4.

Since algorithms of conversion into an approximating triangular mesh representation are known for many surface representations, we restrict our interest to conversion of triangular meshes into DDFs. The goal is to find a more economical approach of conversion than the one just outlined. "More economical" means to reduce the number of support triangles against the number of triangles of the given mesh $F_d$. Well-approximating triangular meshes often have a huge number of triangles, and if the overhead of the DDF in form of the displacement vectors and the height information is added, the storage requirements may exceed practicability.

In figure 6.2 the initial mesh $F_d$ is an approximation of a sphere consisting of a large number of triangles. A corresponding DDF uses a spherical mesh with a smaller number of triangles as support. The goal is to transform any input mesh $F_d$ into a suitable DDF support mesh with a small number of triangles.

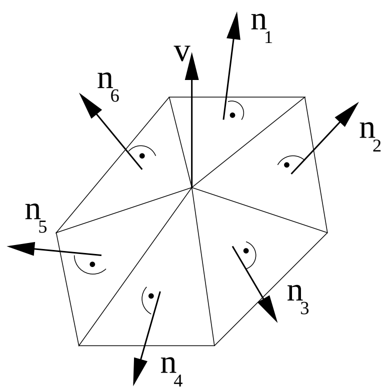
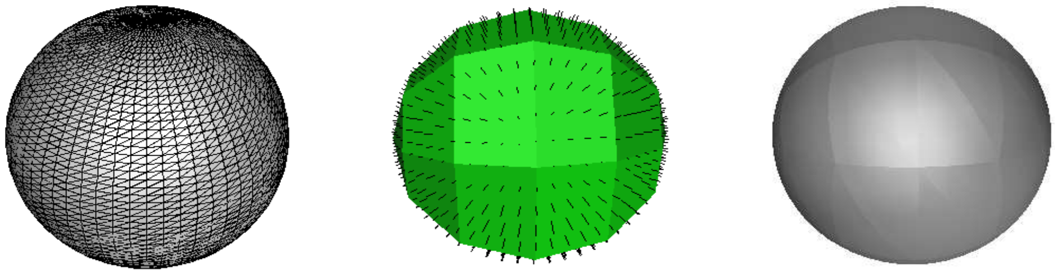



**Figure 6.1:** *Calculation of a pseudo-normal vector* $\mathbf{v}$ *of a vertex of a triangular mesh.* $\mathbf{v}$ *is calculated as average of the unit normals* $\mathbf{n}_i$, $i = 1, .., n$, *of the* $n$ *incident triangles. The normals* $\mathbf{n}_i$ *also may be weighted according to some heuristics like the relative size of the incident angles or areas.*



a) b) c)

**Figure 6.2:** *An approximation of a sphere by a triangular mesh (a), a corresponding DDF visualized by the support and the scaled displacement vectors (b), and the sphere as represented by the DDF (c). The raster resolution is* $m = 6$.

Support mesh reduction for discrete displacement fields has been treated in literature before. In [Oli00], the displacement fields are always perpendicular on the underlying support surface. [Cig99] proposes to decouple mesh reduction and recovery of the old mesh with the displacement fields. This implies that constraints are not introduced on the reduced mesh in order to guarantee that the reduced mesh is indeed a feasible support mesh of the input mesh. Lee et al. [Lee00] outline an approach which is similar to the one we present in the following. The difficulty with the approach by Lee et al. is that its details are not sufficiently described in the paper. The main difference of our approach are additional constraints imposed during mesh reduction in order to get a support mesh which indeed represents the input mesh. Lee et al. mainly compare the directions of the vector fields of the new and the old mesh, and do not analyze in detail whether this heuristic is sufficient.

The problem of DDF support reduction is related to the well-known mesh reduction problem. Mesh reduction means to convert a given mesh, usually step-wise, into a mesh of less vertices, edges, and faces, which still approximates the shape of the initial mesh according to some error criterion. The error criterion is based on so-called error metrics. According to the type of topological operations applied, roughly three classes of mesh reduction approaches can be distinguished, *vertex decimation* [Sch92, Tur92], *edge decimation* [Hop96, Hop99], and *vertex clustering* [Ros93, Lue97]. The elementary operation of vertex decimation is to remove a single vertex and fill the resulting polygonal hole by triangulation. The elementary operation of edge decimation is *edge contraction* which means to contract the end points of an edge into a new point under elimination of the triangles incident to the edge. A special case is *half edge contraction* where the new point is chosen as one of the vertices of the edge. The effect of half edge contraction is similar to that of vertex decimation, with a special triangulation of the hole. Vertex clustering consists of replacing a set of closely located vertices with a new vertex, in one step.

From the topological point of view, our approach of support mesh reduction is based on edge contraction. The main difference to the classical edge contraction problem lies in the error metric. In our case, the support mesh needs not to be close to the given surface, but has to have the property that it allows a DDF representation of the given surface. Support mesh reduction is performed step by step. Every step is executed only tentatively for the first, in order to check whether a set of constraints is satisfied for the resulting mesh. The constraints may concern the containment of $F_d$ in the new crust, the orientation of the triangles of $F_d$ relative to the displacement vector field, or the sufficiently dense sampling of $F_d$.

In order to save space we do not store the DDF explicitly during construction, but transfer the concept of an item buffer known from conventional depth-buffering to our case [Weg84]. Item buffering in our case means to store with every triangle $s$ of the current support those triangles of the original mesh $F_d$ which contain a point $\mathbf{V}$-projecting on $s$. From this information, the height field of the DDF can be explicitly calculated with not too much efforts. If $s$ is large and thus refers to many items, it can be rasterized to some extend, and the corresponding items can be stored for every subtriangle of the raster.

Section 6.1 gives the basic framework of mesh reduction by edge contraction. Section 6.2 presents possible constraints. Section 6.3 discusses the results of empirical investigations.

## 6.1 Mesh Reduction by Edge Contraction

Mesh reduction by edge contraction starts with an initial triangular mesh $F_d$. A vector is given at each vertex which approximates the normal vector of a smooth surface approximated by $F_d$.

Contraction of an edge $e$ of a mesh means to replace $e$ with the center $\mathbf{p} := (\mathbf{p}_a + \mathbf{p}_b)/2$ of its two vertices $\mathbf{p}_a$ and $\mathbf{p}_b$ (figure 6.3). The two triangles that are at most incident to $e$ are removed from the mesh, and the vertices $\mathbf{p}_a$ and $\mathbf{p}_b$ are replaced with $\mathbf{p}$ on all the remaining edges and triangles to which they are incident. The vector $\mathbf{v}(\mathbf{p})$ of $\mathbf{p}$ is calculated as the average of the vectors $\mathbf{v}_a$ and $\mathbf{v}_b$, $\mathbf{v}(\mathbf{p}) := (\mathbf{v}_a + \mathbf{v}_b)/2$.

Another possible choice for the new vertex $\mathbf{p}$ is to set $\mathbf{p} = \mathbf{p}_a$ or $\mathbf{p} = \mathbf{p}_b$ with the advantage that all points of the reduced mesh coincide with points of the original mesh. A disadvantage however is a more unbalanced form of the resulting triangles (figure 6.4). If $\mathbf{p}$ is chosen as center point, approximative adherence of the reduced mesh to the original mesh can be achieved by executing a $\mathbf{V}$-shooting operation into the direction $\mathbf{v}(\mathbf{p})$ and substituting $\mathbf{p}$ by the intersection point with the original mesh.

Mesh reduction by edge contraction is achieved by applying the operation of contraction subsequently edge by edge, as long as predefined criterions are fulfilled. The criterions usually concern the quality of approximation and the quality of the mesh.

The edges for contraction have to be selected with care, if the manifold property has to be preserved. An edge is critical, if it belongs to a closed path on the mesh of length 3 which does not induce a triangle of the mesh. Figures 6.5 and 6.6 give illustrations of this case. If edge $e$ is contracted, two edges fall together. Whether or not these two edges are replaced by one edge in the resulting mesh, this situation leads to pathological configurations. Thus, edges of this type are excluded from contraction.

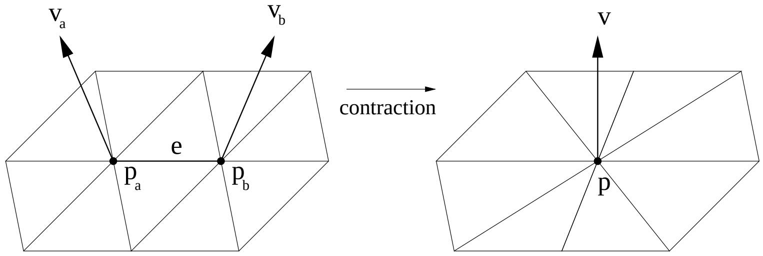



**Figure 6.3:** *Contraction of an edge e. The vertices* $\mathbf{p}_a$ *and* $\mathbf{p}_b$ *and their vertex vectors* $\mathbf{v}_a$ *and* $\mathbf{v}_b$ *are replaced with the new vertex* $\mathbf{p}$ *and the new vector* $\mathbf{v}$*. Edge e and its two incident triangles are removed.*

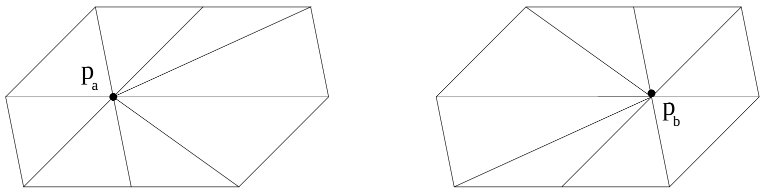



**Figure 6.4:** *The choice of* $\mathbf{p} = \mathbf{p}_a$ *or* $\mathbf{p} = \mathbf{p}_b$ *as new vertex produces unbalanced triangles.*

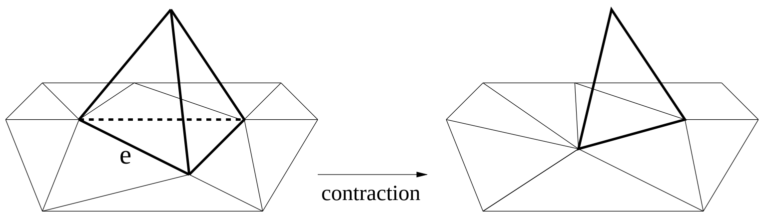



**Figure 6.5:** *The contraction of the base edge e of a pyramid results in a pathological mesh. Two side triangles of the pyramid become identical and a pseudo-2-dimensional corner occurs.*

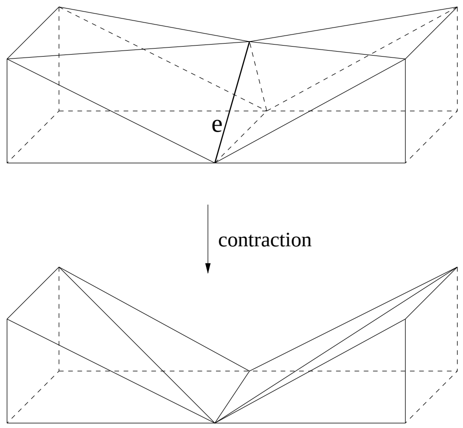



**Figure 6.6:** *The contraction of edge e causes two edges falling on each other which is a geometrically unfavorable configuration.*

There are many possibilities of fixing an order according to which the edges are contracted. We choose the strategy of contracting the shortest feasible edge first. In order to do so, the edges of the initial mesh are sorted in ascending order with respect to their lengths. The edges are processed in this order, but all edges which are modified by an edge contraction are labeled as non-feasibly for contraction in the list and are not contracted while traversing the list. The modified edges are those which become incident to a new vertex caused by edge contraction. If all edges in the list are processed, the remaining edges in the mesh are again sorted according to increasing edge length, and processed analogously as the list before. The algorithm stops, if no more edge feasible for contraction exists.

The strategy has the advantage that triangles with extremely different lengths of their edges should be avoided. However, since we exclude edges on the mesh boundary from contraction, such triangles may occur close to the boundary. Boundary edges are not contracted in order to maintain the boundary of the supporting mesh. This is necessary since the mesh represented by the modified DDFs must not change.

## 6.2 Constraints on Contraction

The mesh reduction algorithm of the preceding section is now extended to a reduction algorithm of the support of a DDF. For every triangle $s$ of the current support, the set of triangles of the initially given surface mesh $F_d$ is stored which have at least one point $\mathbf{V}$-projecting on $s$. As already mentioned, we call this set the item set of $s$. If an edge is contracted, the items have to be re-assigned to the triangles of the new support, for instance by application of the DDF depth-buffer algorithm. In order that the reduced mesh can play the role of a reasonable support and for sake of efficiency, some constraints have to be satisfied. If they do not hold, the edge will not be contracted and is postponed for later processing.

Constraints of interest concern the following aspects:

**Candidate sets of triangles for re-assignment**

For the initial mesh, the candidate set of a triangle consists just of the triangle itself. The candidate set of a triangle resulting from an edge contraction is taken as the union of the triangle sets of the triangles affected by the edge contraction. This approach works since the $\pm 1$-crusts of the non-modified support triangles are not modified, too. Thus, the DDF resulting from an edge contraction coincides with the old one on this part.

The candidate sets obtained in this way might contain more elements than required, in particular after some iterations. This causes computational overhead and slows down the algorithm after a number of iterations. An alternative is to determine only those triangles as candidate set for a support triangle $s$ which intersect the crust of $s$. Crust intersection is tested by splitting up the crust of $s$ into smaller parts surrounded by axis parallel bounding boxes (figure 5.2) and checking for intersections between candiate triangles and bounding boxes. The reduced size of the canditate sets is gained for the additional time requirements of the crust intersection test.

A faster intersection test uses the curved depth buffer algorithm of section 5. The candiate triangle $t$ intersects the crust of the support triangle $s$ if one of the stubbles on $s$ hits the triangle $t$ with a corresponding height value $h$, $h \in [-1, 1]$. However, it turns out that this approach might lead to missing candidate triangles, caused by discretization. In figure 6.7, a triangle $t$ intersects the crusts of the non-modified support triangle $s_a$ and of the non-modified support triangle $s_b$. Assume that $t$ has been hit by discrete $\mathbf{V}$-rays of $s_a$, but not of $s_b$. This means that $t$ has been assigned to $s_a$ before edge contraction, but not to $s_b$. After the edge contraction, the $\mathbf{V}$-rays starting at raster points of $s_b$ usually are different from those before, and now one or more of them might hit $t$. However, $t$ has not been considered in the candidate set of $s_b$.

Experiments have shown that this case indeed happens occasionally in practice. To our experience, it can be practically avoided by taking into consideration the candidate sets of the immediate neighboring triangles of the region modified by edge contraction when updating the displacement vector field and the candidate sets according to the alternative approach. Figure 6.7 depicts the region of triangles affected by a contraction of edge $e$, and their immediate neighbors.

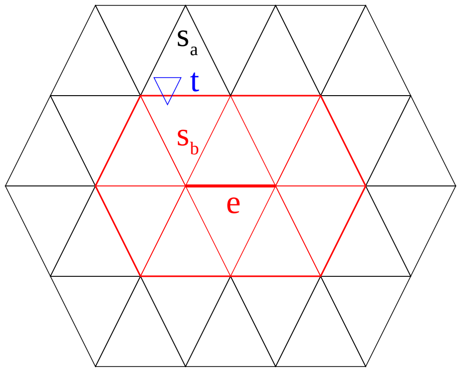



**Figure 6.7:** *The contraction of edge $e$ affects all support triangles incident to the vertices of $e$ (red color) and their crusts. All other support triangles and their crusts remain unchanged. A triangle $t$ of the initial mesh $F_d$ intersects the crusts of an affected support triangle $s_b$ and an unchanged neighboring support triangle $s_a$.*

**Coverage by sampling vectors.**

This constraint has two parts,

a. Every triangle of the input surface $F_d$ has to be hit by at least one **V**-ray.

b. Every **V**-ray starting at a raster point of $S$ has to hit a triangle $F_d$.

Constraint a) can be implemented by maintaining a counter for every triangle $t$ of $F_d$ which stores the number of support triangles to which $t$ is assigned. After a tentative edge contraction, the old hits of the modified support triangles are removed from the involved item triangles. In the **V**-shooting phase of the DDF depth-buffer for re-assigned items, the counters relevant for ray hits are incremented again. If none of the counters is 0 afterwards, the constraint is satisfied.

An implication of constraint a) is that the degree of reduction depends on the resolution $m$ of the raster on the support triangles. If $m$ is fixed during reduction, the distance of the raster points increases with the growing size of the triangles in the reduced mesh. This implies that the sampling rays become less dense, so the danger increases that a triangle of $F_d$ is no longer hit.

Another interesting observation is that condition a) to some extent prevents an unfavorable mutual location of a triangle $t$ of $F_d$ and the support triangles onto which it is **V**-projected. An unfavorable mutual location is given, if the normal vectors of both triangles are nearly perpendicular to each other (figure 6.8). In this case, flanks are smoothed in the mesh represented by the DDF. The "projected" area of $t$ with respect to **V** and $S$ is small, so the chance of a hit by a ray is low.

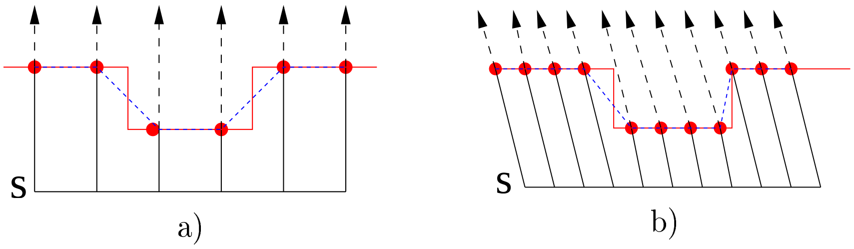



**Figure 6.8:** *Two illustrations of unfavorable surface sampling.*

In order to ensure constraint b) the depth buffer algorithm for re-assigning the items to the modified support triangles is enhanced to mark every raster position with a **V**-ray hitting one of the item triangles. If a raster position remains unmarked after the re-assignment phase, the edge contraction will be rejected.

**Containment in the crust**

It is obvious that the surface $F_d$ has to be in the crust of the DDF to guarantee a satisfying approximation. However, there is a second reason for ensuring the containment in the crust. If the surface $F_d$ exceeds the crust somewhere, this may cause trouble for the reassignment of item triangles to support triangles. Figure 6.9 gives an example. An edge of the support "triangle" $s_3$ is contracted and as a consequence $s_3$ disappears in the contracted mesh. The "triangles" $s_2$, $s_3$ and $s_4$ are modified, but $s_1$ is not. Thus, triangle $t$ is not assigned to the modified region, although it intersects the crust of $s_2$.

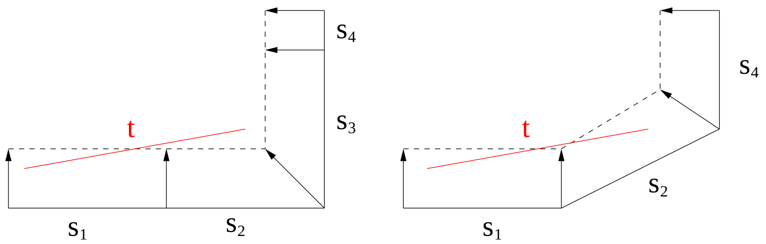



**Figure 6.9:** *Problem of re-assigning a triangle t partially outside the crust to a support triangle.*

This critical case cannot happen, if the surface represented by the DDF is completely within the ±1-crust, that is, if the absolute height values of the DDF are less or equal to one, and if the support has the unique projection property. If a triangle $t$ that does not intersect the original crust of the modified region intersects the crust after modification, that crust would intersect the crust of the non-modified region after modification, because $t$, by assumption, is completely in the crust of the unmodified region. But this contradicts to the assumed unique-projection property after contraction.

The constraint of *coverage by sampling vectors* is not restrictive enough to ensure the containment in the crust. Figure 6.10 gives a contradicting example. The crust containment can be ensured, if the $h$-values of all items assigned to the modified region of the support are between $-1$ and 1, determined with respect to the updated displacement vector field.

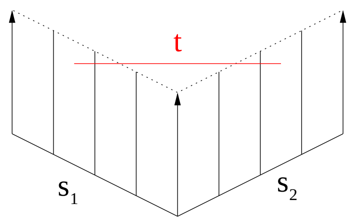



**Figure 6.10:** *The triangle $t$ is hit by discrete $\mathbf{V}$-rays of the support triangles $s_1$ and $s_2$. Nevertheless, $t$ is not entirely in the crust.*

**Unique projection property**

The support of a DDF has the unique-projection property, if every support triangle has this property, and if the crusts of two support triangles do not intersect, except at a shared boundary. Edge contraction changes one of the three vertices of every remaining triangle and its displacement vector. The new point and its vector are obtained by averaging the two end points of the contracted edge and their vectors.

The crusts of the resulting triangles have to be checked for the unique-projection property, and have possibly to be adapted by modifying the length of the new displacement vector according to the strategy described in section 4.4. If the unique-projection property can not be realized with an adaptation of the vector length, the edge contraction will be rejected.

More time-consuming is the investigation and possible correction of the resulting crusts with respect to intersection-freeness with the crusts of all other triangles of the support. The hashing structure of section 5.1 can be used for accelerating of the search for intersecting pairs of crusts.

An interesting point of observation is that even if the unique-projection property does not hold for $S$, a reasonable behavior of the algorithm is still possible. This concerns the case that the crusts of two non-neighboring support triangles intersect. According to the strategy of the DDF depth-buffer algorithm, this might be critical, as figure 6.11 illustrates for two cases. The reason is that the DDF depth-buffer algorithm locates a triangle $t$ to all support triangles whose crusts intersect $t$. The constraint on the candidate set of triangles for re-assignment restricts the application of the DDF depth buffer algorithm to a local support. This restriction ensures a reasonable assignment of item triangles to support triangles, because an assignment to non-neighboring triangles is impossible.

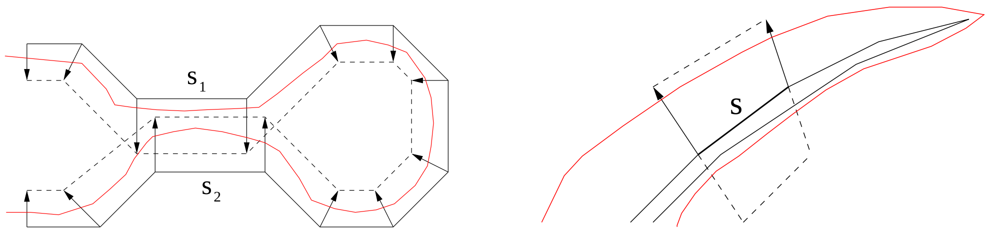



**Figure 6.11:** *If the crusts of two support triangles overlap, the DDF depth-buffer algorithm may assign triangles of the input surface to more than one support triangle. The drawings show two typical situations. The input surface is indicated in red color, and the support surface in black color.*

**Angle between displacement vector and pseudo normal vector**

As already explained by figure 6.8, the DDF-represented surface might not adequately follow the original surface, if the angle between the normals of the input surface and the displacement vector is large, in particular close to 90°. A favorable situation like the one shown in figure 6.12 can be achieved, if an edge contraction is accepted only if a given upper bound on the angle, like e.g. 45° is not exceeded.

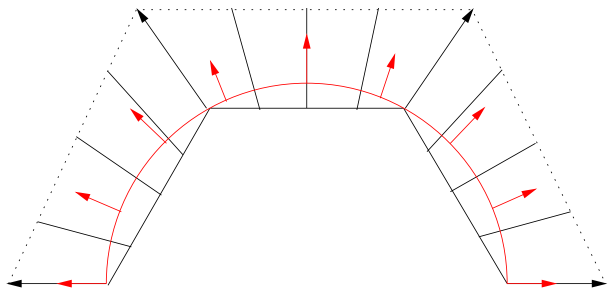


**Figure 6.12:** *A favorable surface representation will be achieved by a DDF, if the angles between the vectors of the displacement vector field (black color) and the normals of the input surface $F_d$ (red color) are small.*

A related issue is the possibility of foldover of the input surface relative to the displacement vector field. Figure 6.8, right, illustrates the problem.

At the left flank of the input surface, a displacement ray may intersect the surface more than once. This problem is also diminished by the threshold on the angle and the requirement that every surface triangle has to be hit by at least one ray.

## 6.3 Empirical Investigations

We have applied versions of the support reduction algorithm to several data sets. Two data sets known as benchmarks for meshing problems have been used for this purpose, the Stanford Bunny and the Stanford Happy Buddha [Sta03]. Furthermore, a B-spline patch has been used as an example of a smooth surface which is closer to the requirements of production. A difficulty with the Bunny data is that the mesh is not manifold everywhere and that some triangles are missing.

The algorithms have been implemented in Java 1.3i. For the performance measurements we have used a personal computer with an AMD Athlon XP1700 processor and 1 GB RAM.

Because the evaluation of many of the constraints compiled in section 6.2 are time-consuming, versions of the algorithm which do not satisfy all constraints have been investigated, too.

Figure 6.13 shows the result, if just the basic mesh contraction algorithm of section 6.1 without any constraints is applied to the Bunny data. Figure 6.13 a) shows the original mesh which consists of 16 301 triangles. Figure 6.13 b) displays the result of mesh reduction which consists of 503 triangles. The resulting mesh shows difficulties at the ears. The reason is an incorrectness of the input mesh. The original mesh has many holes, and about 300 edges of the mesh have more than two incident triangles. Figure 6.13 c) visualizes a discrete displacement field nevertheless calculated with the reduced mesh as support. The resolution of the height field raster is $m = 10$. Figure 6.13 d) shows the surface represented by the DDF. In comparison to figure 6.13 a) the ears are destructed, and the lower chest of the Bunny is connected by a non-correct thin surface piece to the feet. The latter is caused by a hole of the input mesh which opens the way for some rays to the closest intersection point with the surface, which is at the feet. Nevertheless, a quite good representation is achieved in many regions of the surface. In figure 6.13 e), figures 6.13 c) and 6.13 d) are overlaid. The support surface $S$ and the represented surface $F$ of the DDF penetrate each other. Vectors pointing to the inside of the Bunny are caused by negative height values $h$.

The mesh reduction algorithm required 7 seconds to reduce the Bunny mesh from 16 301 to 503 triangles. It needed 1 second to establish the DDF from the

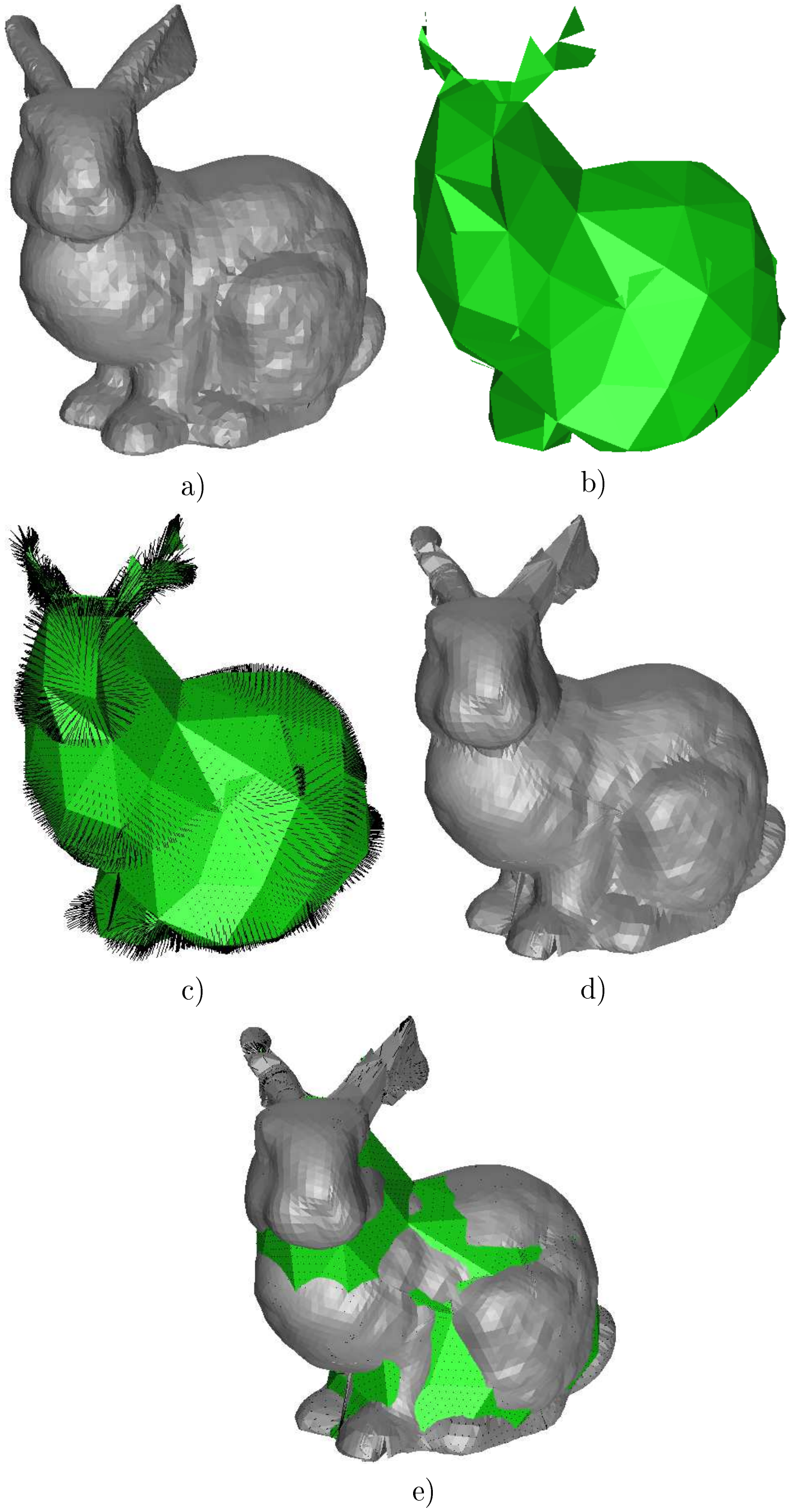



**Figure 6.13:** *Conversion of the Bunny mesh into a DDF-representation using a simple mesh reduction algorithm. a) The original mesh (16 301 triangles). b) The reduced triangle mesh (503 triangles). c) The DDF. d) The surface represented by the DDF. e) An overlay of the support surface and the represented surface.*

reduced mesh and to cut the vectors at the original mesh by the DDF depth-buffer algorithm of chapter 5. The depth buffer algorithm has to perform the localization of item triangles over the whole support because a relation of item and support triangles is not directly known.

Figure 6.14 shows the result achieved by a version of the support reduction algorithm which takes into consideration the constraint on the candidate set of triangles for re-assignment, the constraint of covering of the input mesh by sampling vectors, and containment in the crust, cf. section 6.2. This time the reduction algorithm yields a mesh with 1 675 triangles which does not have further contractable edges. In comparison to figure 6.13 the ears are approximated well with the exception of a small dent inside the bottom edge of the right ear. A closer look to this area shows that the support mesh $S$ is still extremely broken here as consequence of the destructed original mesh.

This version of the mesh reduction algorithm required 150 seconds to reduce the mesh from 16 301 to 1 675 triangles. 3.4 seconds were necessary to calculate the DDF explicitly on the reduced mesh. In comparison to the mesh reduction algorithm without constraints, the time of mesh reduction is a factor of 21 higher. The reason is the expensive test of the constraints. The time of DDF generation is higher, too, because the number of support triangles of the reduced mesh has grown.

Because of the defects of its mesh, the Bunny is a bad example in order to demonstrate the correctness of DDF-generation, but it is a good example to demonstrate the robustness of the algorithm.

We have applied the same version of the reduction algorithm to the Stanford Buddha as an example for a large mesh. The mesh of the Happy Buddha consists of 293 232 triangles. Figure 6.15 a) shows the original mesh. It needed about 6 hours to produce the DDF which is visualized in figure 6.15 b), c) and d). The reduced support mesh has 17 061 triangles.

The surface represented by the DDF consists of 6 159 021 triangles. This is much more than for the original surface. The number of triangles can be reduced with the adaptive surface triangulation of section 7. Results are shown in figure 6.16. The number of triangles depends on an error bound $\epsilon$, as will be explained in section 7. Figure 6.16 b) and c) give two different examples. In figure 6.16 d) the triangle edges are explicitly displayed in order to demonstrate the adaptivity of the triangulation to the shape of the surface.

The calculations have been performed on the same PC as before. The input data set has been an unorganized list of triangles, and it has taken 96 minutes to produce a structured mesh including all required neighboring informations. 224 minutes were necessary to reduce the mesh, and 8 minutes to establish the explicit DDF.

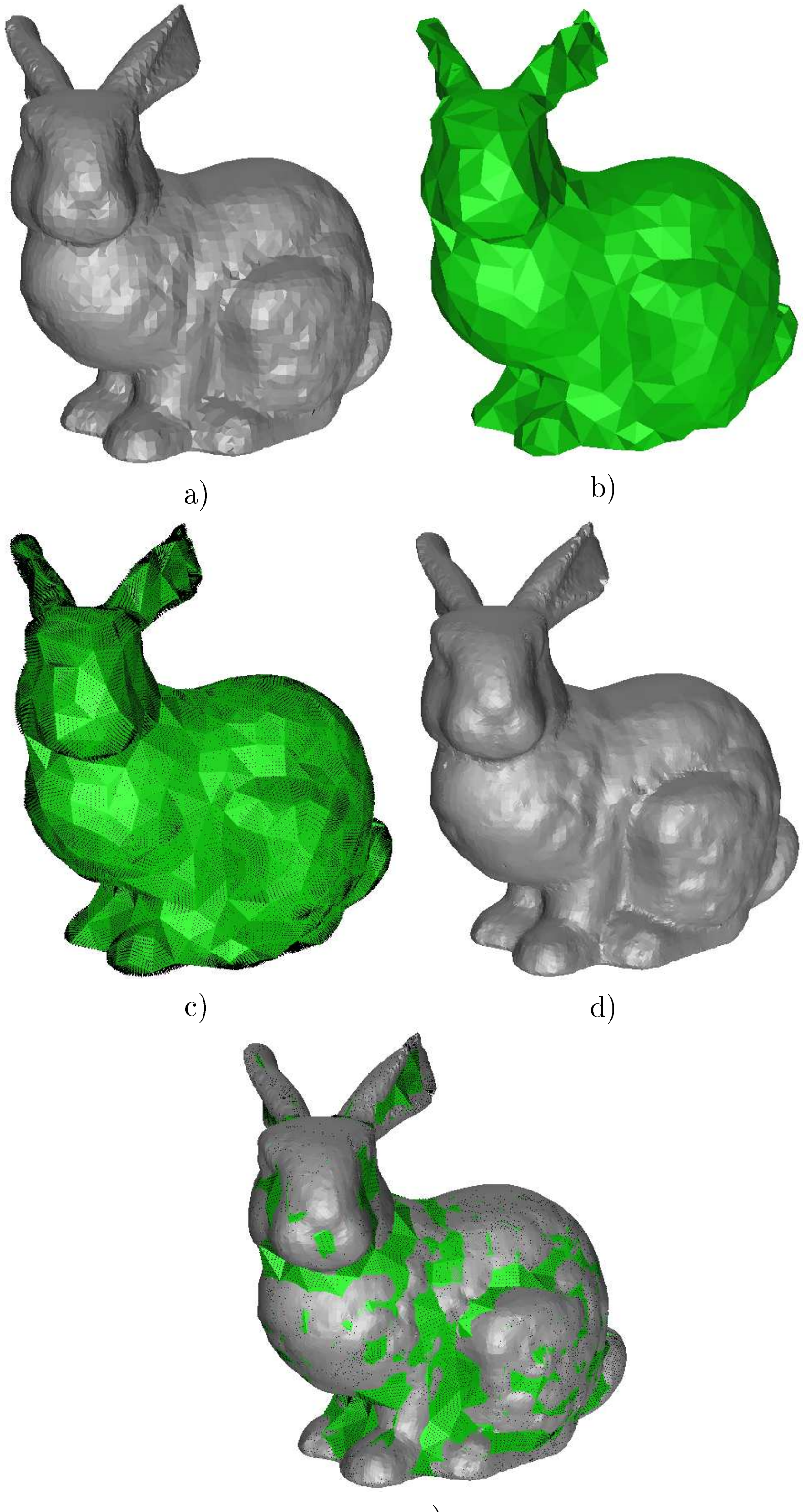



**Figure 6.14:** *Conversion of the Bunny mesh into a DDF representation by taking into consideration the constraints on selection of the candidate items and on the coverage by the sampling rays. a) The original mesh (16.301 triangles). b) The reduced triangle mesh (1675 triangles). c) The DDF d) The surface represented by the DDF. e) An overlay of the support surface and the represented surface.*

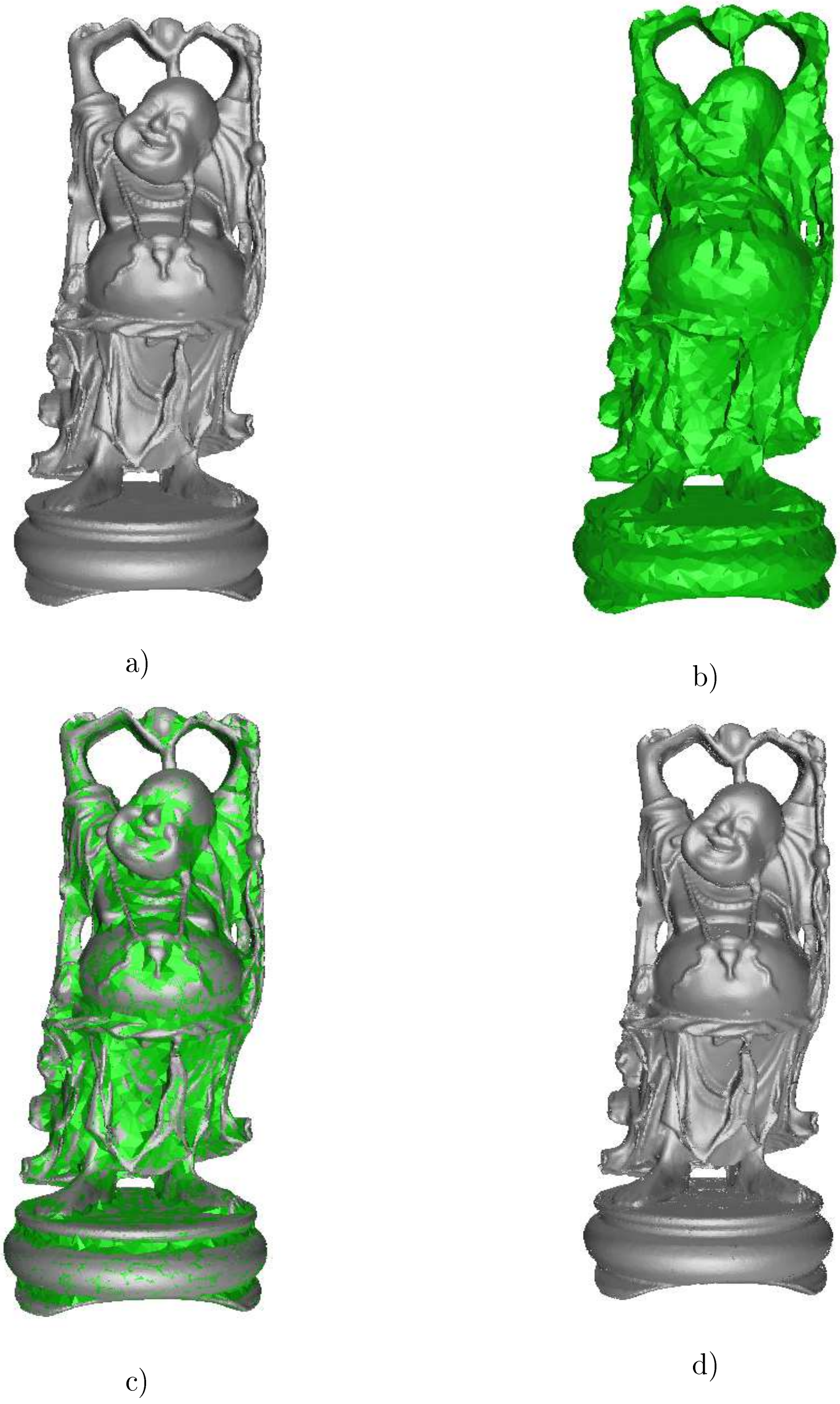



**Figure 6.15:** *A large example for the conversion into a DDF. The version of the algorithm which takes into consideration the constraints on selection of the candidate items and on the coverage by the sampling rays is used. a) The original mesh (293 232 triangles). b) The reduced mesh (*17 061 *triangles). c) Overlay of the support surface and the represented surface. d) The surface represented by the DDF.*

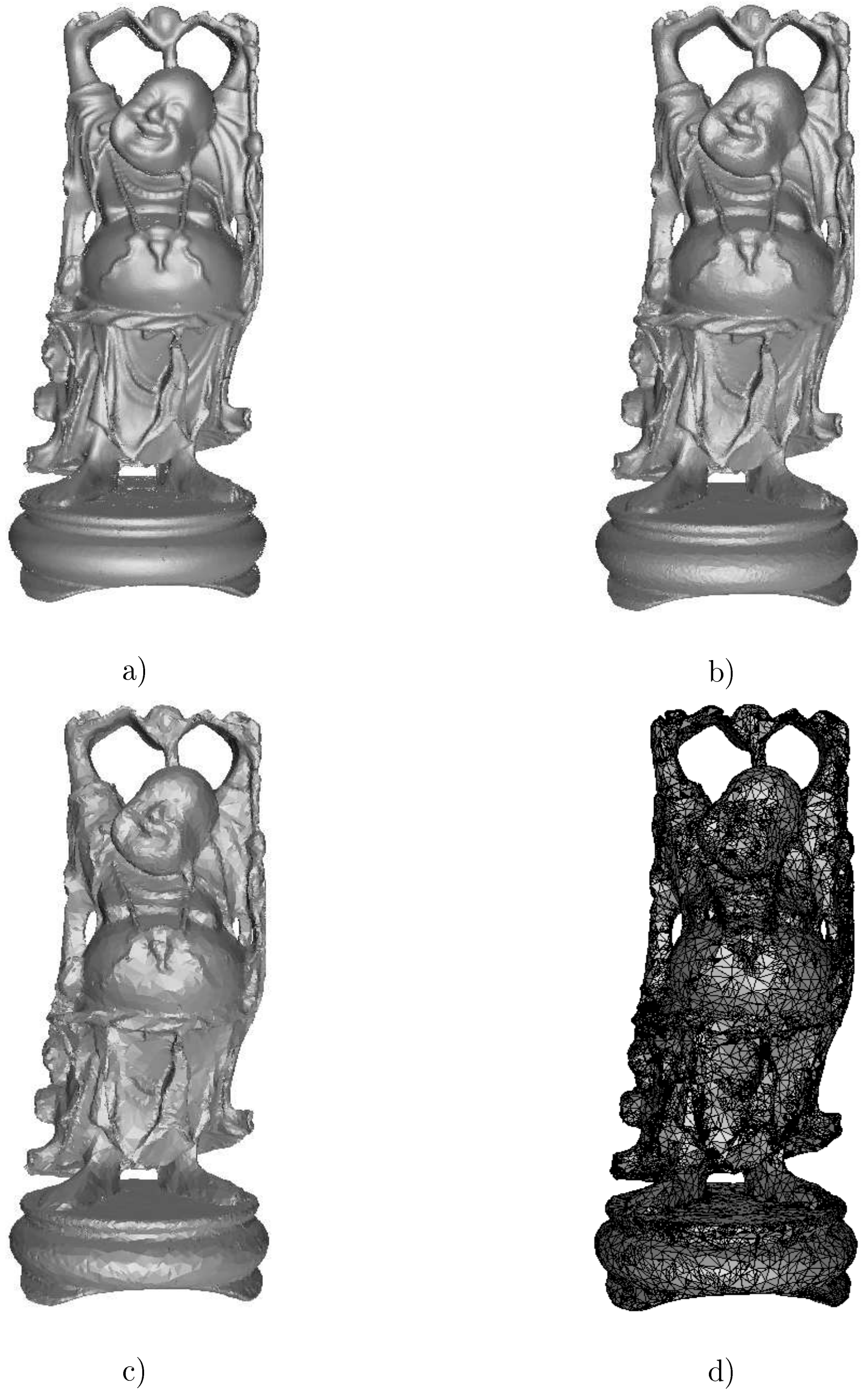



**Figure 6.16:** *Example of an adaptive triangulation of the surface represented by a DDF. a) The surface represented by all the 6 159 021 triangles of the DDF rasters. b) A fine adaptive triangulation with 438 459 triangles. c) A rough adaptive triangulation with 156 657 triangles d) The rough adaptive triangulation with visualization of the edges of the triangle.*

We now turn to a further data set which is an approximation of a smooth B-spline patch by a triangle mesh of 20 000 triangles (figure 6.17). The reason for choosing this example is that the smoothness of the surface enables an easier recognition of defects than the complex surfaces used up to now. Another reason is that workpieces with such smooth free-formed surfaces are of particular interest in production, and are not yet proberly supported by commercial software.

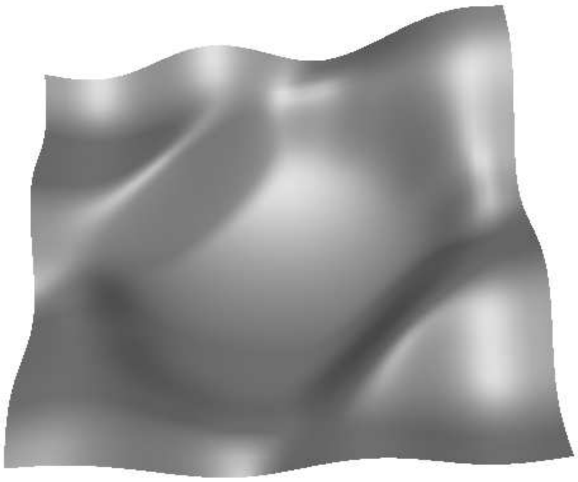
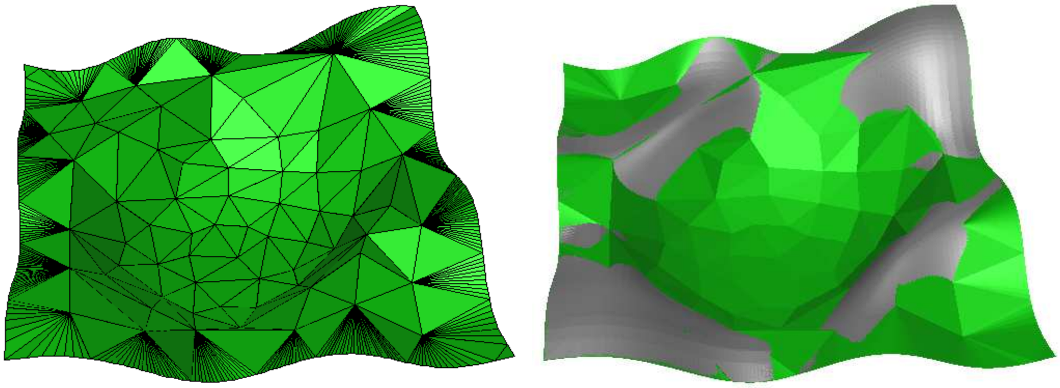


**Figure 6.17:** *A triangulated B-spline patch approximated by a mesh of* 20 000 *triangles.*

Figure 6.18 shows the result of the application of the support reduction algorithm with the same constraints as before at a resolution of $m = 30$. The resulting support mesh has 526 triangles. Near the border the reduced mesh has several very thin and long triangles because edges on the boundary are excluded from contraction.

We have also applied the same version of the algorithm for different resolutions of the raster of the height field, $m = 10, 20, 30, 40, 50$. As we know from section 6.2, the resolution influences the amount of reduction. Figure 6.19 compiles the number of achieved support triangles and the computation time. The third row corresponds to the example of figure 6.18.

In figure 6.20 (left), the time for net reduction is plotted over the resolution. The reduction time increases quadratically with the resolution $m$. In figure 6.20 (right), the number of triangles in the reduced support is plotted over the resolution. Convergence towards an amount of 400 triangles can be noticed. The reason is that 400 triangles are a lower bound of any mesh since the boundary edges are excluded from contraction. The boundary, however, has $4 \cdot 100 = 400$ edges.



a) b)

**Figure 6.18:** *Conversion of the B-spline mesh into a DDF representation by taking into consideration the constraints on the selection of the candidate items and on the coverage by the sampling rays. a) The reduced support mesh (526 triangles). b) Overlay of the support mesh and the represented surface.*

| **Resolution** | **Time [minutes]** | **Triangles** |
|---|---|---|
| 10 | 3 | 1420 |
| 20 | 10 | 662 |
| 30 | 25 | 526 |
| 40 | 43 | 478 |
| 50 | 82 | 452 |

**Figure 6.19:** *Conversion of the B-spline mesh into a DDF representation by taking into consideration the constraints on the selection of the candidate items and on the coverage by the sampling rays, for several resolutions of the height field raster.*

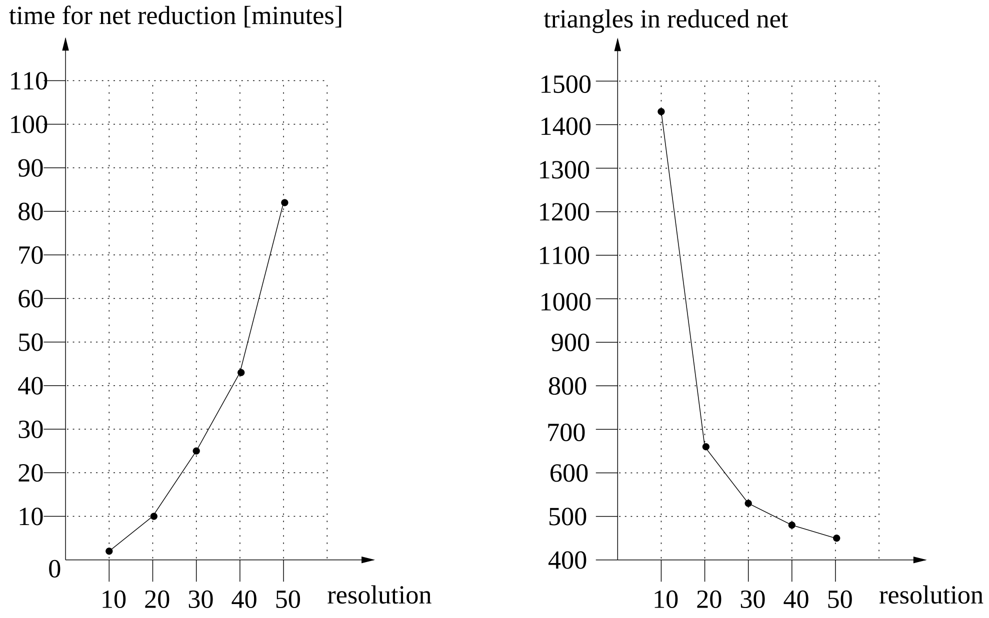



**Figure 6.20:** *The time for mesh reduction (left) and the number of triangles in the resulting support mesh (right) plotted against the resolution of the height field.*

The lower bound of 400 triangles is reached by the version of the algorithm without any constraints. Figure 6.21 shows the resulting mesh which consists of a fan of triangles emerging at a common center point. Mesh reduction needs 10.4 seconds. On the reduced mesh a DDF has been calculated in 56 seconds, which is also shown in Figure 6.21 overlaid to the support.

Figure 6.22 shows the surface represented by the DDF in comparison to the original mesh and a mesh calculated by the version with constraints.

The artifacts which can be noticed on the surface generated by the unconstraint algorithm are caused by rendering. A closer inspection of the surface has shown that it is free of holes.

For simple surfaces where parameters like the length of the displacement vectors can be preset, so the constraints of section 6.2 are automatically fulfilled, the algorithm without explicit constraints should be preferred. The example of the unconstraint version has needed 66.4 seconds for the calculation of the example of figure 6.21 which has a resolution of $m = 50$, of which 10.4 seconds were required for mesh reduction and another 56 seconds for the calculation of the displacement field. In comparison, the version with constraints needs about 82 minutes for the same resolution, of which just 3 seconds have been required for DDF calculation from the reduced mesh (fifth row of the table in figure 6.19). This is a factor of 74.

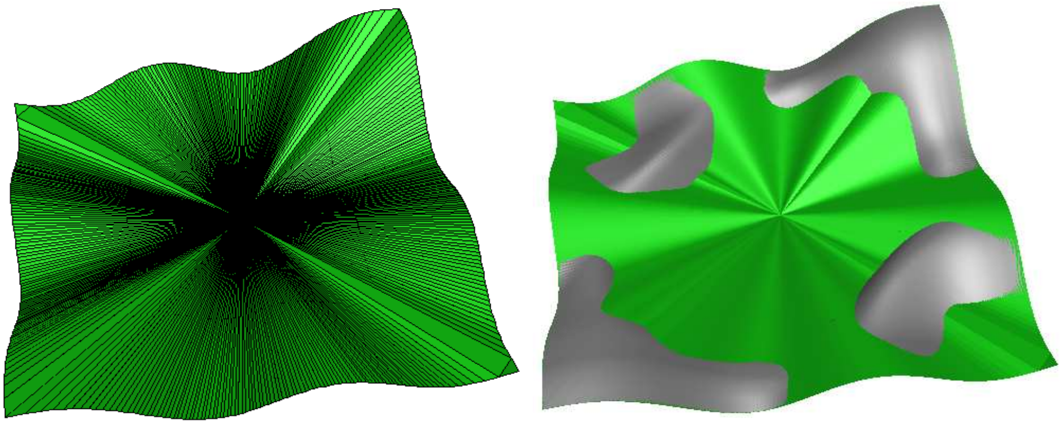
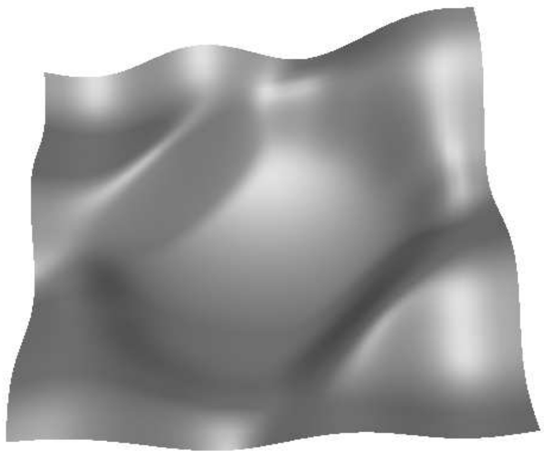
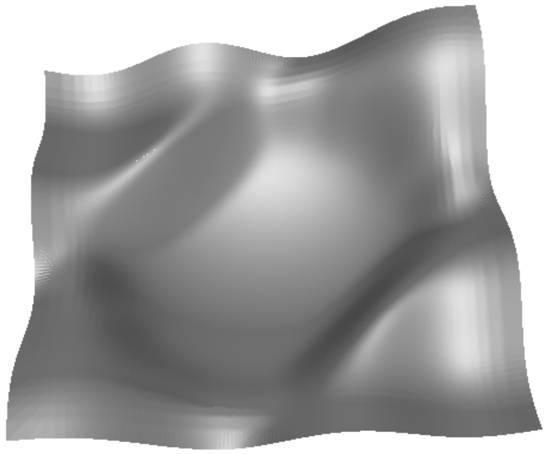


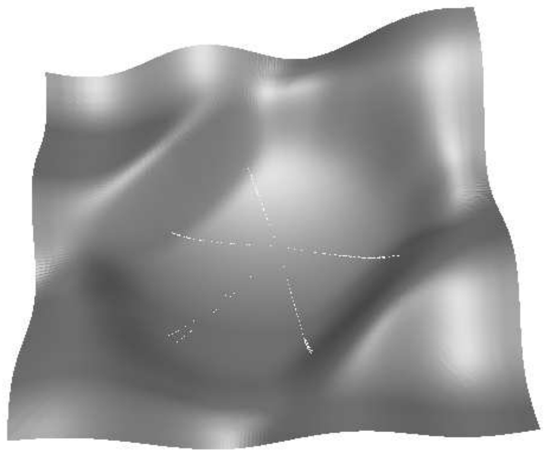
**Figure 6.21:** *Conversion of a B-spline surface into a DDF description by support reduction without constraints. Left: The reduced mesh. Right: An overlay of the support mesh and the represented surface.*



a)



b)



c)

**Figure 6.22:** *Comparison of the surfaces generated by the different versions of the algorithm. a) The original surface. b) The surfaced generated by the version with constraints. c) The surface calculated without constraints. The line artifacts visible on the surface are discretization errors during rendering caused by extremely small triangles.*

# Chapter 7

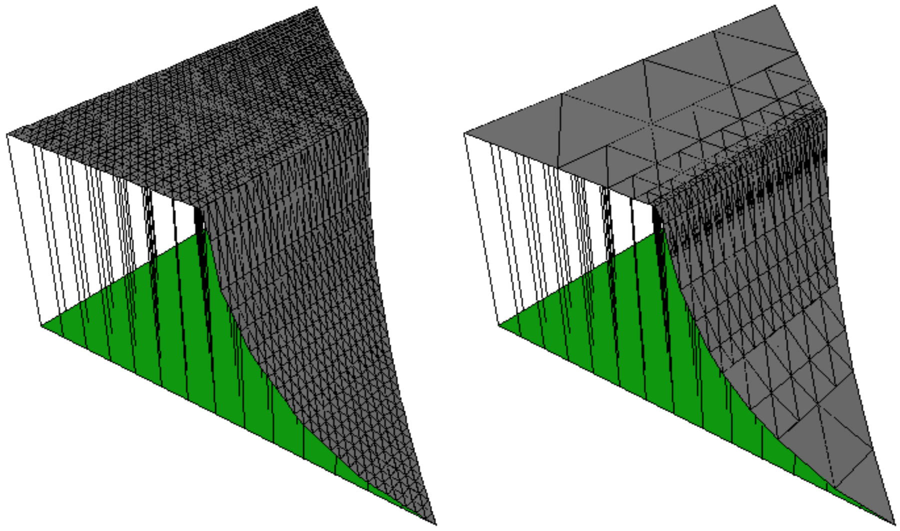
# Adaptive Triangulation of DDF-represented surfaces

A triangular mesh approximating the surface represented by a DDF can be obtained immediately by connecting the points $\mathbf{q}_{i,j,k}$ corresponding to raster points of the discrete height fields according to the structure of the raster grid, cf. chapter 3. Figure 7.1 a) shows a surface mesh derived on this way from one support triangle.

A difficulty of this approach is that the number of triangles may become large, as we already noticed in the example of figure 6.16. The reason is that the number of triangles generated from one support triangle is quadratic in the raster resolution $m$. Another observation is that the resulting triangular mesh may require considerably more space than the DDF representation. The reason is that a raster vertex is roughly represented by just one height value by the DDF, but by three coordinate values by the mesh representation. For rendering where the triangles can be forwarded immediately into the graphics pipeline after generation and thus have not necessarily to be stored, a large number of triangles reduce the rendering performance, and may, for instance, prohibit interactive inspection of a surface on the screen.

Basically, the difficulty can be diminished by employing a mesh reduction approach like edge contraction used in section 6.1 for reduction of the support mesh of a DDF. In the following we choose a different way. The reason is that the meshes of DDF-represented surfaces are composed of quite regularly structured submeshes emerging from the support triangles. These meshes can be reduced by applying an approach originally developed for surfaces in parameter representation by Clay et al. [Cla88].

The algorithm starts with the vertices $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ of the support triangle $s$ and the surface points $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$ assigned to the vertices (figure 7.2). The surface



a) b)

**Figure 7.1:** *Adaptive triangulation of a DDF surface. a) Non-adaptive triangulation by taking the raster cells of the discrete height field. b) A triangulation adapted to the represented surface. In both figures a few of the displacement vectors are indicated in order to depict the relation between the support triangle and the represented surface.*

defined by the triangle $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$ is examined for the quality of approximation of the surface over $s$. For this purpose an error criterion $E_t(\varepsilon)$ is used where $\varepsilon > 0$ controls the admitted error. If the triangle $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$ satisfies the error bound, it is reported as approximation of the surface. Otherwise the support triangle is subdivided into four sub-triangles induced by the centers $\mathbf{p}_{12}, \mathbf{p}_{13}, \mathbf{p}_{23}$ of its edges. The surface points assigned to $\mathbf{p}_{12}, \mathbf{p}_{13}, \mathbf{p}_{23}$ depend on a further error criterion $E_e(\varepsilon)$ for the quality of approximation by edges. If $E_e(\varepsilon)$ is fulfilled for the edge $\mathbf{q}_i, \mathbf{q}_j$, then the center point $\mathbf{s}_{ij} := \frac{1}{2}\mathbf{q}_i + \frac{1}{2}\mathbf{q}_j$ is assigned to $\mathbf{p}_{ij}$ as surface point, $i < j$, $i, j = 1, 2, 3$. Otherwise the DDF-point over $\mathbf{p}_{ij}$, that is $\mathbf{q}_{ij} = \mathbf{p}_{ij} + h_{i,j}\mathbf{v}(\mathbf{p}_{ij})$ is taken. The resulting triangles are processed recursively in the same manner, as long as they are not degenerated to a point. Figure 7.2 shows a configuration resulting from subdivision.

The error criterions $E_t(\varepsilon)$ and $E_e(\varepsilon)$ have to satisfy the condition that if $E_e(\varepsilon)$ does not hold for an edge of a triangle, $E_t(\varepsilon)$ will not hold for the triangle, either. This condition is canonical since it is not reasonable that a triangle is accepted as a good approximation if at least one of its edges is not. On the other hand, all the edges might approximate the surface well, but the triangle does not.

Furthermore, if $E_e(\varepsilon)$ holds for an edge $e$, it also has to hold for all sub-

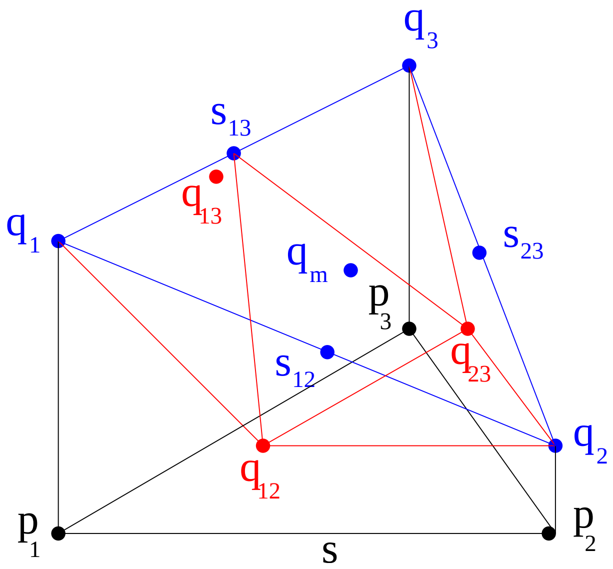



**Figure 7.2:** *Subdivision of an approximating triangle over a sub-triangle of the support. A point* $\mathbf{s}_{ij}$ *is taken as a vertex of a subdivided triangle if the edge is well approximating the original surface, and a point* $\mathbf{q}_{ij}$ *is used if the corresponding edge is not well approximating the original surface.*

edges of $e$. That means that the sub-edges of a well approximating edge is well approximating, too.

A straightforward choice of $E_t(\varepsilon)$ is to return "true", if and only if the maximum of the distances between the approximating and the true surface at every raster point of $s$ is less than $\varepsilon$. For the initial triangle, the corresponding condition is

$$\max\left\{ \left\|\mathbf{q}_{i,j,k} - \left(\mathbf{p}_{i,j,k} + h_{i,j,k} \cdot \mathbf{v}_{i,j,k}\right)\right\| \mid i+j+k = m-1,\ i,j,k \geq 0\right\} < \varepsilon,$$

where

$$\mathbf{q}_{i,j,k} = \frac{i}{m-1}\mathbf{q}_1 + \frac{j}{m-1}\mathbf{q}_2 + \frac{k}{m-1}\mathbf{q}_3,$$
$$\mathbf{p}_{i,j,k} = \frac{i}{m-1}\mathbf{p}_1 + \frac{j}{m-1}\mathbf{p}_2 + \frac{k}{m-1}\mathbf{p}_3,$$
$$\mathbf{v}_{i,j,k} = \frac{i}{m-1}\mathbf{v}_1 + \frac{j}{m-1}\mathbf{v}_2 + \frac{k}{m-1}\mathbf{v}_3.$$

Analogously, $E_e(\varepsilon)$ can be defined by taking the maximum along an edge.

For higher resolutions $m$, this calculation may become somewhat time consuming. The number of distances to be evaluated is of order $O(m^2 \log m)$ for every support triangle of the given DDF. The worst case occurs, if all the sub-triangles of the hierarchy of depth $O(\log m)$ have to be tested. A heuristic approach which often works quite well is to check just the distance of the center of an edge against the true surface point, instead of taking all the raster points along an edge. Figure 7.3 shows calculation times for both approaches applied to the surface of figure 7.1 with different resolutions of the support triangle $s$. The triangulation time for the sophisticated approach increases quadratically whereas the time for the heuristic approach retains constant. Both approaches produce a constant number of 415 triangles for all resolutions of the support triangle.

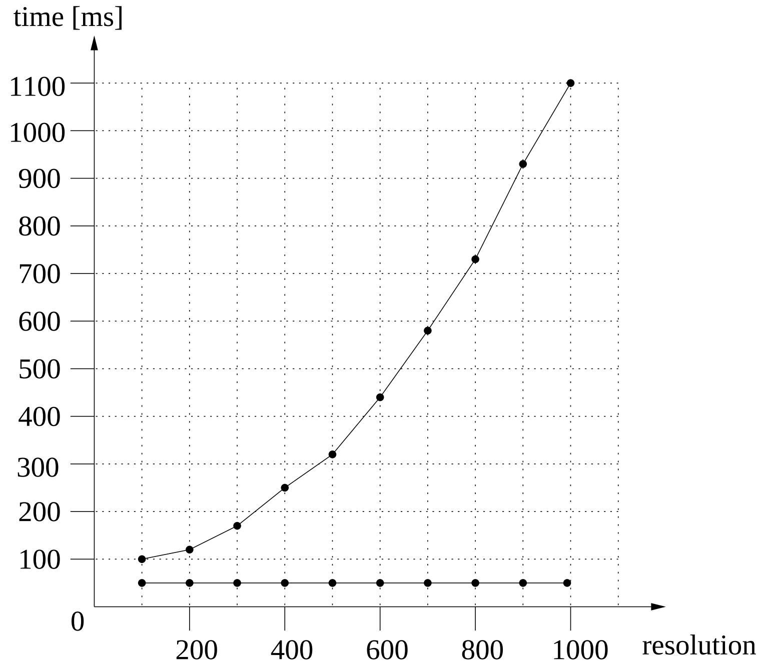



**Figure 7.3:** *Triangulation times for different resolutions of the support triangle s. The time for the heuristic approach retains constant (bottom curve) whereas the time for the sophisticated approach increases (top curve).*

The reason for distinguishing between error criterions for triangles and edges, respectively, is to avoid cracks in the surface. Candidates for cracks are locations where triangles of different levels of subdivision are incident to each other. Such locations can be noticed in Figure 7.1 b) which has been generated with the algorithm. There are line segments which have one triangle on one side, and two triangles on the other. A crack would occur if the vertex shared by the two triangles would not be located on the line segment. However, this configuration only happens, if an edge has been accepted as a good approximation. In this case, the subdivided edges are recognized as good approximations, too, so the subdivision point is chosen as a center point which canonically lies on the straight line.

Even though no cracks occur, meshes of the resulting type, with edges of different lengths lying on each other, are sometimes undesirable. An example is rendering by Gouraud shading where the interpolation of colors on the triangles along a line segment may lead to different results on both sides, resulting in undesirable visible artifacts. This difficulty can be resolved by eliminating the so-called T-vertices by subdividing the larger triangle into sub-triangles, so the new edges match completely with the shorter ones. This approach, however, requires information about the incidence of triangles which is not required by the original algorithm. This requires more efforts for the implementation. Another approach is to use an algorithm as described by Ruprecht et al. [Rup98]. This algorithm subdivides triangles according to the subdivision structure of their edges, that is, the subdivision is chosen so that an edge which is recognized as well approximating is not subdivided. Subdivision of such edges because of the regular subdivision pattern of the presented algorithm is responsible for the T-vertices. A disadvantage of the algorithm by Ruprecht et al. is, that it may lead to thin triangles which again may cause troubles for rendering.

# Chapter 8

# Tool Path Planning with Discrete Displacement Fields

Path planning is a central task of computer-aided production, and considerable work has been performed in the past, cf. e.g. the surveys by Marshall/Griffiths [Mar94], Dragomatz/Mann [Dra97], and the book by Choi/Jerard [Cho99]. Nevertheless, the problem cannot be considered as solved yet, in particular not for complex free-formed shapes which require manipulation with five degrees of freedom.

The task of path planning is to find a motion path of a tool which yields a good approximation of a desired workpiece, if applied to an initially given workpiece. A tool path consists of a continuous sequence of tool configurations. A tool configuration is given by a location and orientation of the tool in space. The tool in motion along the tool path has to transform the workpiece from its given shape into a desired shape. A tool path is suitable to perform this transformation, if the following constraints are fulfilled:

1. To every point $\mathbf{p}$ on the desired shape a point $\mathbf{q}$ exists on the cutting part of the tool, so the distance between $\mathbf{p}$ and $\mathbf{q}$ is less than a given bound $\varepsilon > 0$.
2. The tool never penetrates the desired workpiece with its cutting part.
3. The tool never collides with the current workpiece outside the cutting part.

The cutting part of the tool is the region of the tool which can erase material, if it is in contact with the workpiece, in contrast to the rest of the tool which should not come into contact with the workpiece. Usually the top of a milling cutter is used as cutting part whereas the shank of the milling tool is not. The bound $\varepsilon > 0$ defines the precision achieved by a path of this type.

Two situations can be distinguished: path planning *close to the desired workpiece surface* and planning of paths *far from the desired workpiece surface*. The *close-to-surface situation* means that the superfluous material can be erased by a path along the desired surface, that is the layer of material to be removed is thin enough, so the tool can always be in contact with the desired surface. In the *far-from-surface situation*, the difference between the current and the desired workpiece is so thick that the tool generally cannot reach the desired surface. Since discrete displacement fields focus on the boundary of a surface, our interest lies on the close-to-surface case. Most contributions to the field concern the close-to-surface case. This is not surprising since it is of higher importance for the final quality of the produced surface than the far-from-surface process. The aspect of far-from-surface removal of material has been addressed for example by Tangelder et al. [Tan98] and Flutter/Todd [Flu02].

The typical approaches to close-to-surface path planning are characterized by *global reduction* of the space of all paths by heuristic constraints, and local path selection, within the constraints, by *local optimization*. Typical constraints concern the selection of the tool [Jen02, Gla99], the milling approach and the milling strategy [Hos92, Hel91]. One approach is to subdivide a given surface into processing objects each of which satisfies a special set of constraints [Sto99]. Path planning is then performed on each processing object separately, with a strategy suitable to the constraints. The approach of processing objects is particularly useful, if standard processing objects can be identified which can be pre-processed and re-used [Mey99]. The possibility of re-usage, however, is more likely for regular shapes than for free-formed shapes. Another aspect is adaptation of the desired shape to the requirements of milling [Elb97], in particular if the shape cannot be produced as it is.

*Local optimization* can have several goals: local tool fitting, avoidance of collisions, choice of a distance between neighboring segments, minimization of idle path length, and compensation of tool deformation.

The goal of *local tool fitting* is to approximate the workpiece surface well by the cutting part of the tool, under avoidance of penetration of the workpiece. Approaches to reaching the goal have been tilting of the tool along a given path [Lee97, Li94, Yan99], and a priori avoidance of the problem by analytical exclusion [Pot99]. There is also a relation to tasks of assembling, like for instance drilling in of a screw, where motion paths can be generated by online collision detection and response [Len99].

Solutions proposed for the problem of *suitable distances between neighboring path segments* have been the choice with respect to error analysis [Lee98], surface-adaptive determination of the distance of iso-planar path segments using isophotes [Din03], and a vector field approach [Kim02].

By *idle motion* we mean those parts of the overall motion of the tool where the tool is not in contact with the workpiece. These parts, which connect segments of the milling path, have to be minimized. This task seems to be related to the NP-hard traveling salesman problem [Gar79], a recent treatment can be found in [Par02].

Usually path planning assumes rigid tools. However, in practice milling or grinding tools are deformed during the process. An approach to taking into consideration *tool deformation* is to perform path planning with a rigid tool. With the resulting path a milling or grinding simulation is carried out which takes into consideration deformation based on the calculation of acting forces and tool reaction. Approaches to modeling of tool engagement conditions can e.g. be found in [Wei01] for the case of a milling simulation. The result of simulation can be used for correction of the tool path. Further simulation and possibly iteration of this approach can be used in order to verify and improve the result. However, procedures like this can be time-consuming, so they are not in widespread use up to now.

*Collision avoidance* concerns the tool outside the cutting part and other parts of the milling machine which must not get in touch with the workpiece. Two types of approaches can be distinguished: *offline planning* and *online planning* of collision avoidance.

Methods of *offline collision planning* explicitly pre-calculate a representation of the free-space which is then used for finding a suitable path [Lat91]. *Online planning* means that, during calculation of a path, tests for intersection with the obstacle are performed and, dependent on the result, the next step on the path is chosen collision-free. For quick intersection tests and distance calculations, algorithms and data structures based on bounding volumes like spheres, axes-parallel bounding boxes, and oriented bounding boxes are applied [Len99]. Moreover the computing power of graphics hardware available for e.g. z-buffering is sometimes used. Online collision avoidance has been used by several authors for path planning of milling processes by simulation [Ho01, Elb94, Li94]. Other suggestions use potential fields [Chi02] or interpret collision avoidance as a visibility problem [Elb94].

In the following we show how several of these aspects can be treated using discrete displacement fields. The basic observation is that two items are crucial for path planning: the curvature and the accessibleness of the workpiece surface. We will focus on curvature, and exclude accessibleness, that is the aspect of global collisions. We will, however, give some hints at the end of the chapter how collision may be taken in account as well.

Section 8.1 compiles some concepts of differential geometry for analyzing the curvature of surfaces, and presents an approach to curvature-based tool path planning based on those concepts. Section 8.2 is devoted to curvature analysis of surfaces represented by DDFs. Section 8.3 describes an algorithm for streamline calculation on DDF surfaces. Section 8.4 shows how DDFs can be used for grinding simulation and discusses the application of simulation to the choice and correction of tool paths. The chapter is concluded with section 8.5 on possibilities of taking into account collisions.

## 8.1 Surface Curvature and Tool Path Planning

The analysis of curve and surface curvature is a central topic of differential geometry [Bla73]. The curvature $\kappa(t)$ of a curve $\mathbf{r}(t) = (x(t), y(t), z(t))^T$, $t \in I$ where $I$ is a finite real interval, in three-dimensional space is defined by

$$\kappa = \sqrt{\frac{\mathbf{r}'^2\mathbf{r}''^2 - (\mathbf{r}'\mathbf{r}'')^2}{(\mathbf{r}''^2)^3}}$$

where $\mathbf{r}' = (x'(t), y'(t), z'(t))^T$ and $\mathbf{r}'' = (x''(t), y''(t), z''(t))^T$ are the derivations of first and second order of the function $\mathbf{r}(t)$ and $\mathbf{r}'^2$ and $\mathbf{r}''^2$ are the dot products of the vectors $\mathbf{r}'$ and $\mathbf{r}''$ with themselves. $\kappa(t)$ has an intuitive geometric interpretation: its inverse value is the radius of a tangent circle in point $\mathbf{r}(t)$ which is obtained as the limit of a sequence of circles through three points of the curve, the middle one among them $\mathbf{r}(t)$ when the other two points are moved towards $\mathbf{r}(t)$ (figure 8.1).

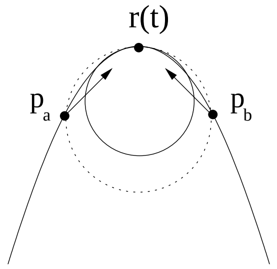



**Figure 8.1:** *Geometric interpretation of the curvature $\kappa$.*

For surfaces the curvature in a point $\mathbf{p}$ is analyzed by considering the family of planes containing the line through $\mathbf{p}$ in direction of the surface normal $\mathbf{n}$ in $\mathbf{p}$. These planes are called normal cutting planes (figure 8.2). The curvatures of the intersection curves of the normal cutting planes with the surface are used for characterization of the surface curvature at $\mathbf{p}$. Typical characterizations are the *maximum* and the *minimum* curvature at $\mathbf{p}$ which means the respective extremal values of the curvatures induced by the normal cutting planes. These curvatures are called *main curvatures*. Further measures are the average curvature which is the average of the two main curvature values, and the Gaussian curvature which is the squareroot of the product of the two main curvatures.

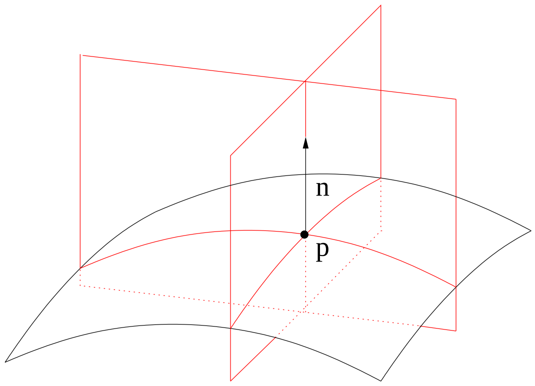



**Figure 8.2:** *Normal cutting planes at a surface point* **p** *with normal* **n**. *The curvatures of the intersection curves are used for characterizing the curvature at the surface point* **p**.

If the maximum and minimum curvature are different, it is known that exactly one normal cutting plane exists for each of the values. The tangent of the two cutting curves define two directions, called the direction of maximum and minimum curvature. The directions can be expressed by vectors of the maximum and minimum curvature to the surface points (figure 8.3). It is known that the two directions are perpendicular to each other.

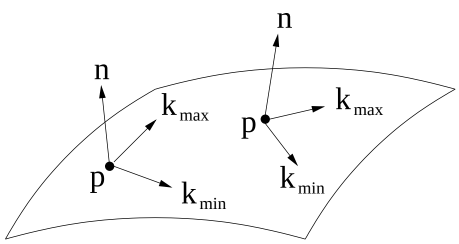



**Figure 8.3:** *To every point* **p** *of the surface where the maximum and the minimum curvatures are different, a direction* $\mathbf{k}_{min}$ *of minimal curvature and a direction* $\mathbf{k}_{max}$ *of maximum curvature can be assigned.* $\mathbf{k}_{min}$ *and* $\mathbf{k}_{max}$ *are perpendicular to each other.*

This observation is important for the choice of the direction of the motion path of a tool. In particular for tools for which the surface-close boundary of a cross section perpendicular to the motion direction is straight, the direction of the main curvature with higher absolute value should be preferred. A grinding disc or belt is an example of a tool of this type. The reason is that the deviation between the tool and the surface, measured between the cutting curves on the cross section, should be less than for other directions.

However, the choice just recommended is dominated by another aspect in regions of saddle points. A surface point is a saddle point, if the minimum and maximum curvature have different signs. In that case, the cross section perpendicular to the chosen direction should show the tool on the convex side of the intersection curve. The reason is that for this choice, a tool of appropriate size does not need further correction of the milling path, as explained later.

The lines of curvature are another concept of differential geometry. A line of maximal curvature follows the direction of maximum curvature in every point of the surface. Analogously, a line of minimal curvature follows the direction of minimum curvature. For every point with different maximum and minimum curvature values, a maximum and a minimum line of curvature exist which traverse the point. According to the observations before, lines of curvatures, or parts thereof, are well-suited as segments of milling paths. Since only a finite number of milling path segments is possible, the surface has to be covered with a finite number of such segments, so the deviation of the surface produced with those segments nowhere deviates more than a given error bound from the desired surface. Simulation of the milling process can be used in order to check a given set of path segments for whether the desired approximation is achieved, and to correct or insert further path segments in regions which have not been sufficiently approximated yet.

An interesting observation is that the lines of maximum and minimum curvature can be understood as streamlines in the vector fields defined by the vectors of maximum and minimum curvature, respectively. Since the vector field of favorable milling directions does not need to be one of those fields throughout the whole surface, but can be composed of vectors from both or even other vectors, the view of streamlines is very useful in order to calculate milling paths for this more general "tool flow vector field", too.

The surface curvature is also relevant for the choice of the tool and its size. Large tools can remove a lot of material in a short time but they cannot reach narrow concave parts of the workpiece surface. The maximum of the absolute values of the minimum and maximum curvature are an upper bound of the tool size for ball-shaped tools, except in regions where both main curvatures have the same sign, and the tool is on the convex side of the surface. This observation can be concluded from the interpretation of curvature given in figure 8.1. Larger tools would penetrate the surface.

Since larger tools allow faster removal of material than small tools the usage of a tool which has the maximum allowed size over the whole surface may be inefficient if significant parts of the surface allow a larger tool. A solution is to decompose the surface into segments by considering the curvature behavior and the size of curvature values so that a tool can be chosen for every segment which is almost equally appropriate over the whole segment.

These observations lead to a path planning approach consisting of four steps (figure 8.4). The first step is to calculate the main curvatures of the desired surface and the corresponding directions (figure 8.4 a). The second step is segmentation under consideration of the curvature type of the surface, that is saddle or not, convex or not from the side of the tool access, and the amount of curvature (figure 8.4 b). The third step is the definition of a field of vectors which expresses the desired direction of the tool path in every point, by using the vectors of main curvature, e.g. the direction of maximum curvature is chosen (figure 8.4 c). The fourth step is the calculation of a family of path segments which satisfy the constraints on the direction in every point (figure 8.4 d).

## 8.2 Curvature Calculation on DDFs

Explicit formulas of the minimum and the maximum curvature at a point of a surface in parameter representation are known in differential geometry [Bla73]. The formulas depend on the so-called first and second fundamental forms of differential geometry which in turn can be calculated from the first and second derivatives of the formula of the surface. Because every surface patch $\mathbf{f}$ represented by a DDF over a support triangle $s$ can be considered as a surface in parameter representation with parameter domain $s$, the formulas of curvature can be approximately evaluated for the raster points on $\mathbf{f}$ by calculating discrete finite-difference approximations of the partial derivatives of $\mathbf{f}$.

A DDF-surface patch $\mathbf{f}$ is given at discrete parameter values $(u, v)$, $u = j/(m-1)$, $v = k/(m-1)$, $j, k = 0, \ldots m-1$, $j + k \leq m-1$, by

$$\begin{aligned} \mathbf{f}(u,v) &\approx (1-u-v)\mathbf{p}_1 + u\mathbf{p}_2 + v\mathbf{p}_3 \\ &\quad + h(u,v)\left[(1-u-v)\mathbf{v}_1 + u\mathbf{v}_2 + v\mathbf{v}_3\right] \end{aligned}$$

where $h(u, v)$ is the discrete height function on the grid of the support triangle $s$.

An approximation of the partial derivation $\partial\mathbf{f}/\partial u$ with respect to the first parameter is for example given by

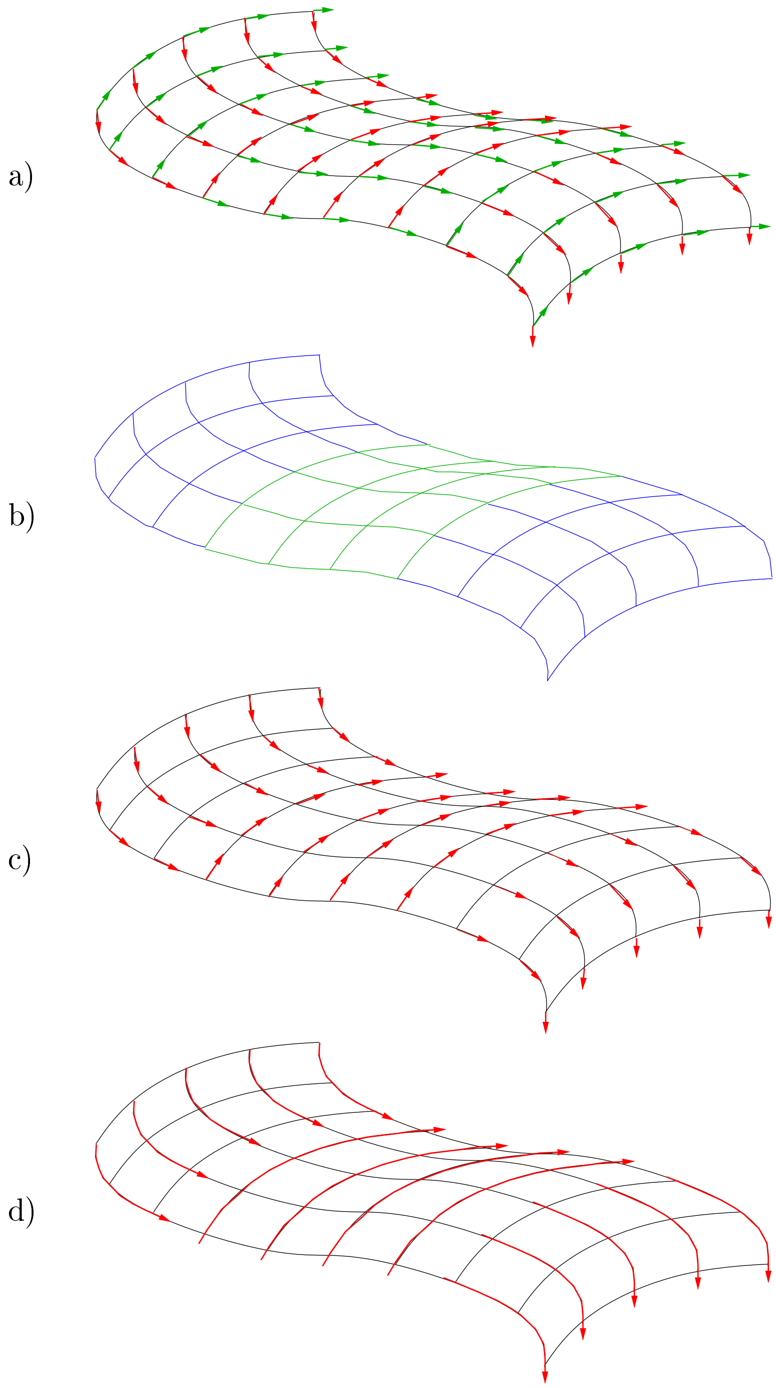



**Figure 8.4:** *Illustration of the steps of the tool path planning. a) Curvature calculation,* $\mathbf{k}_{min}$ *is depicted in green,* $\mathbf{k}_{max}$ *in red color b) Segmentation of the surface according to curvature c) Choice of the tool motion direction, e.g.* $\mathbf{k}_{max}$ *d) tool path selection by calculation of streamlines in the tool flow vector field.*

$$\frac{\partial \mathbf{f}(u,v)}{\partial u} \approx (\mathbf{p}_2 - \mathbf{p}_1) + h(u,v)(\mathbf{v}_2 - \mathbf{v}_1) + \frac{\partial h(u,v)}{\partial u}\left[(1-u-v)\mathbf{v}_1 + u\mathbf{v}_2 + v\mathbf{v}_3\right]$$

A discrete approximation of the partial derivation $\partial h(u,v)/\partial u$ required in the formula is

$$\frac{\partial h(u,v)}{\partial u} \approx \frac{1}{2}\left[h(u+1/(m-1),v) - h(u-1/(m-1),v)\right] \tag{8.1}$$

Figure 8.5 visualizes the maximum and minimum curvature of a DDF represented surface. The range of curvature values is divided into a finite number of intervals to which different colors are assigned. The points on the surface are drawn in the color of the interval of their curvature value. The coloring splits the surface into segments which can be understood as an example of a subdivision of the surface according to the value of the curvature, similar to the approach proposed for tool selection in section 8.1.

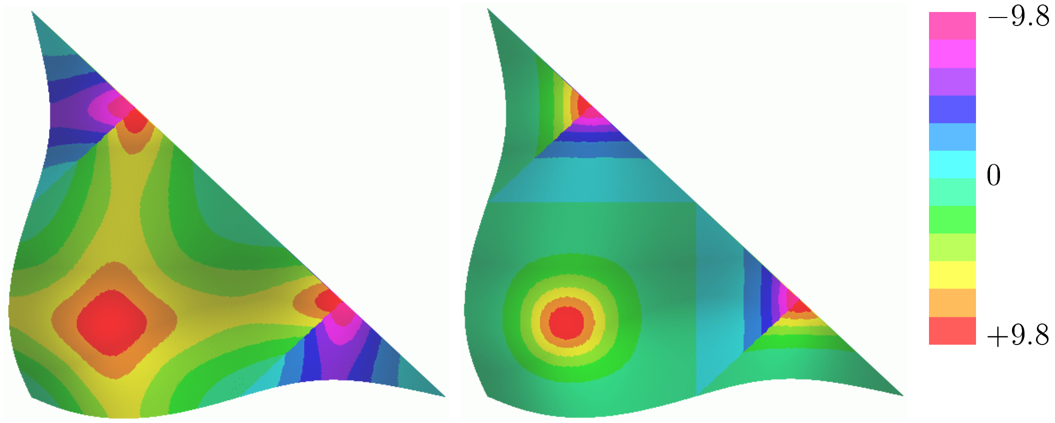



**Figure 8.5:** *Visualization of the maximum curvature (left) and the minimum curvature (right). The values are divided into intervals which are visualized by different colors.*

Besides the calculation of the main curvatures, we use these discrete derivatives also for the calculation of the directions of the main curvatures. This is possible since closed formulas exist for the directions, too, which again depend on the first and second fundamental forms.

Figures 8.6 and 8.7 visualize the directions of main curvature calculated at selected raster points of a DDF surface. At every surface point, the directions are given by two tangential vectors pointing into the direction of minimal and

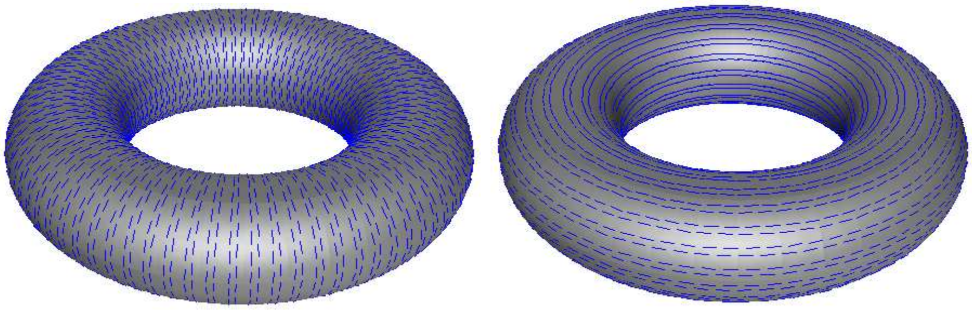


**Figure 8.6:** *Directions of maximum (left) and minimum curvature (right) on a DDF-represented torus, indicated at selected points by short blue line segments.*

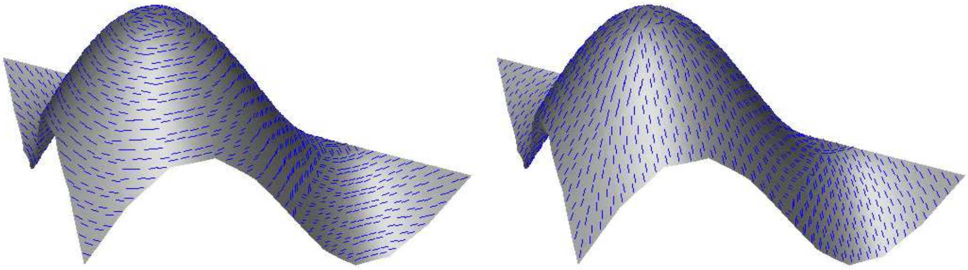


**Figure 8.7:** *Directions of maximum (left) and minimum curvature (right) on a two-dimensional sine surface represented by a DDF. The blue line segments indicate the directions at selected points.*

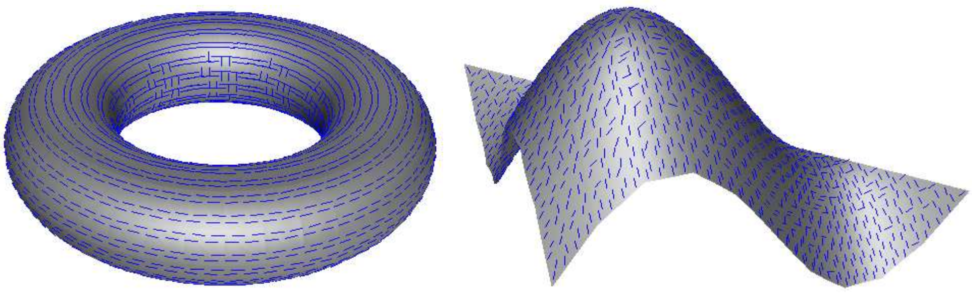


**Figure 8.8:** *For higher resolutions of the DDFs some curvature directions e.g. inside the torus are corrupted.*

maximal curvature. Short blue line segments represent the vectors in the pictures.

The discrete approximation of the partial derivation $\partial h(u,v)/\partial u$ in formula (8.1) can not be calculated at the boundary of the triangle because height values outside the triangle, e.g. $h(-1/(m-1), 0)$, are required. A way out is to substitute the derivations at the border with the derivations of neighboring grid positions inside the triangle. Alternatively, a quotient of one-sided differences may be used in equation (8.1).

Another problem is that the difference of the height values of neighboring grid positions can be extremely small, if the resolution of the DDF is high. In this case rounding errors can severely affect the quality of the approximation of the derivatives, in particular that of the derivatives of second order. The result can be corrupted directions of curvature. Figure 8.8 illustrates such troubles for an example where the minimum curvature on the torus and the sine function is calculated for DDFs higher resolved than for the previous pictures. A solution to this problem is to extend the difference considered in the discrete approximation of the partial derivation $\partial h(u,v)/\partial u$, e.g. formula (8.1) is rewritten as

$$\frac{\partial h(u,v)}{\partial u} \approx \frac{1}{2}\left[h(u + d/(m-1), v) - h(u - d/(m-1), v)\right] \tag{8.2}$$

where $d$ expresses the difference and is choosen according to the resolution $m$.

The torus of figure 8.6 is represented by 800 DDFs of resolution $m = 7$. In figure 8.8 a resolution of $m = 25$ has been used. The sine function in figure 8.7 consists of 100 DDFs with a resolution of $m = 10$, in comparison to figure 8.8 where a resolution of $m = 50$ has been used. The calculation of the curvature directions requires less than one second on all surfaces, on an AMD Athlon XP1700 processor with 1 GB RAM.

## 8.3 Calculation of Streamlines on DDF-Surfaces

A vector field $\mathbf{w}$ on a surface is given by a continuous function which assigns a vector $\mathbf{w}(\mathbf{p})$ to every surface point $\mathbf{p}$. $\mathbf{w}(\mathbf{p})$ is tangential to the surface $\mathbf{F}$. For a parametrized surface $\mathbf{F} : P \to I\!R^3, P \subseteq I\!R^2$ the vector field can be defined alternative as a two dimensional field $\overline{\mathbf{w}}$ in the parameter domain. The related three dimensional vector field on the surface is then given by $\mathbf{w} := \nabla\mathbf{F} \cdot \overline{\mathbf{w}}$ where $\nabla\mathbf{F}(u,v) := (\frac{\partial\mathbf{F}(u,v)}{\partial u}, \frac{\partial\mathbf{F}(u,v)}{\partial v})$. If the vectors of the field $\mathbf{w}$ are interpreted as velocity of a stream, a streamline is defined by the path of a particle inside the stream. Figure 8.9 gives an example for a streamline in a vector field.

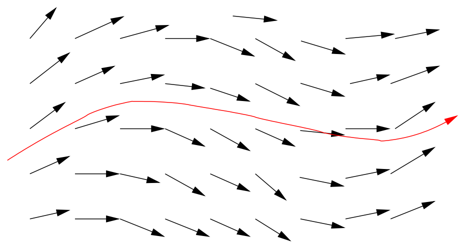


**Figure 8.9:** *A streamline (red color) which follows the direction of a vector field.*

Streamlines can be described by a simple differential equation of first order. Classical techniques to solve this equation numerically can be used for an approximative calculation of streamlines [Pos93]. In [Roe00] this kind of technique is used for the special case of a curvature field on a triangular mesh.

The DDFs can be understood as discrete parametrized functions on a two-dimensional parameter space. To define a discrete vector field on the DDF a two dimensional vector $\overline{\mathbf{w}}$ is assigned to every raster position. The discrete vector field can be extended to a continuous field by barycentric interpolation of the raster point vectors into the interior of the raster triangles. A discrete vector field on a DDF surface is for example given by the directions of one of the main curvatures, as explained in section 8.2.

On DDFs, a streamline can be found by walking from one raster position to a next one as follows. Let $\overline{\mathbf{w}}$ be a vector field defined on the two-dimensional DDF raster. Let $\mathbf{p}$ be the current raster point at which the streamline has to be extended, and $\mathbf{w}(\mathbf{p})$ be the vector at $\mathbf{p}$ which defines the direction of extension. We consider the ray with origin $\mathbf{p}$ in direction of $\overline{\mathbf{w}}(\mathbf{p})$. Among the raster points adjacent to $\mathbf{p}$ in the raster, the point $\mathbf{p}'$ closest to the ray is chosen as successor of $\mathbf{p}$ on the streamline. The resulting direction $\overline{\mathbf{w}}_r = \mathbf{p}' - \mathbf{p}$ deviates from the desired direction $\overline{\mathbf{w}}(\mathbf{p})$ (figure 8.10). The deviation is compensated by adding the error vector $\overline{\mathbf{w}}(\mathbf{p}) - \overline{\mathbf{w}}_r(\mathbf{p})$ to the vector $\overline{\mathbf{w}}(\mathbf{p}')$ of the next point $\mathbf{p}'$.

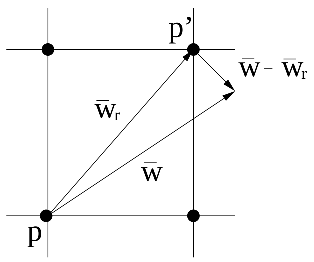



**Figure 8.10:** *Compensation of the error between the raster-induced direction and the desired direction of the streamline.*

The raster-based approach is feasible, if the raster resolution is high. A rasterization effect of the resulting curve, which might nevertheless be noticed, may be diminished by low-pass filtering. Low-pass filtering can for example be achieved by moving a window covering an odd number of consecutive points along the curve, and reporting at every location of the window a weighted average of the points within the window. A simple example is $\mathbf{p}'_i = \frac{1}{3}\mathbf{p}_{i-1} + \frac{1}{3}\mathbf{p}_i + \frac{1}{3}\mathbf{p}_{i+1}$ where $\mathbf{p}_i$ is the point to be replaced and $\mathbf{p}'_i$ is the new location.

Figure 8.11 shows a family of streamlines of maximum curvature. For its generation, a streamline of minimum curvature shown in green has been calculated. On the minimum curvature streamline, a finite number of points in a certain distance were selected which were used as seed points for the calculation of the streamlines of maximum curvature by the streamline algorithm. The streamline algorithm stops, if the boundary of the surface is reached, or if a surface point is reached which already belongs to a streamline. It requires less than one second to calculate the family of streamlines in figure 8.11 on a AMD Athlon XP1700 processor with 1 GB RAM.

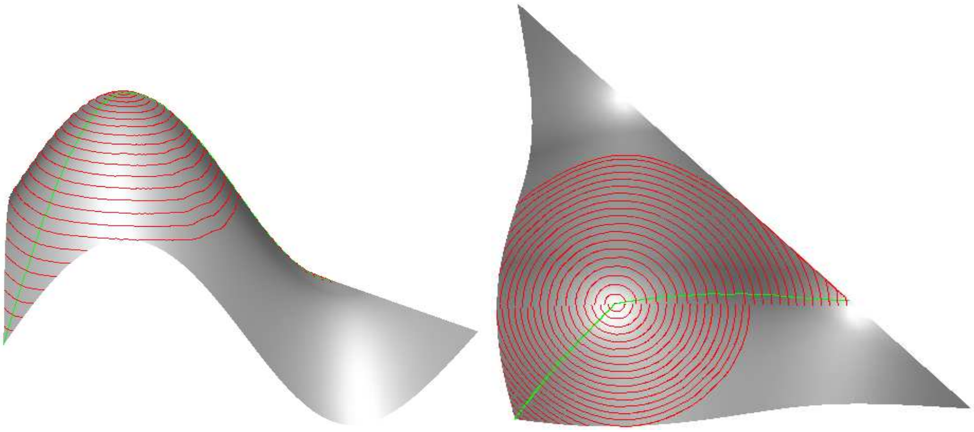


**Figure 8.11:** *A DDF surface with streamlines in direction of maximum curvature (red). The seed points of the streamlines are taken from a streamline of minimum curvature (green). The left image shows a side view, the right figure a top view of the surface.*

The alternative solution for the calculation of streamlines presented in the following is independent of fixed raster positions and allows to follow streamlines beyond the boundaries of the support triangles with less computational effort. According to figure 8.12, a point $\mathbf{q}$ close to the DDF surface is chosen as starting point. $\mathbf{q}$ is projected onto the support mesh. The support triangle onto which $\mathbf{q}$ projects is found using the same grid structure as for the localization process of the curved depth buffer algorithm (chapter 5.1). Let $\mathbf{s}$ be the result of projection of $\mathbf{q}$, and $\mathbf{p}$ be the corresponding DDF surface point. $\mathbf{p}$ is the starting point of

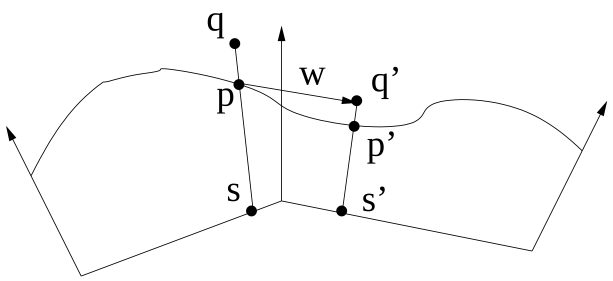



**Figure 8.12:** *Calculation of two consecutive points* $\mathbf{p}$ *and* $\mathbf{p}'$ *of a streamline.*

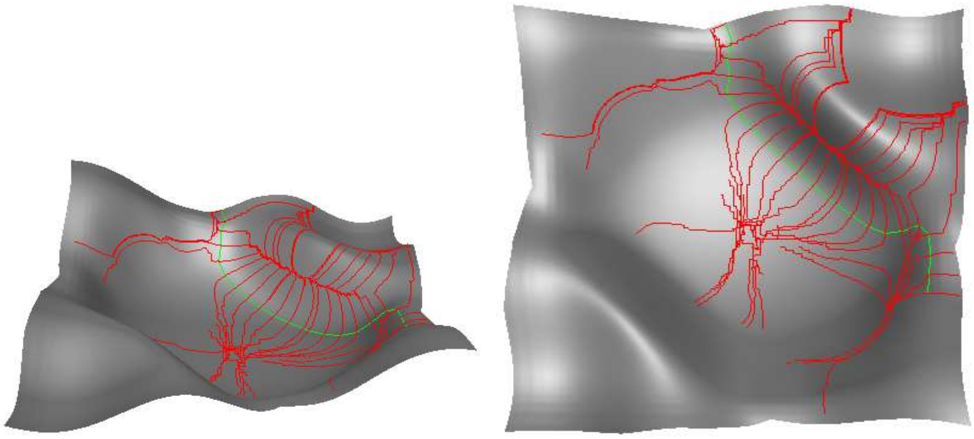


**Figure 8.13:** *A DDF surface with streamlines in direction of maximum curvature (red). The seed points of the streamlines are taken from a streamline of minimum curvature (green). The left image shows a side view, the right figure a top view of the surface.*

the streamline. $\mathbf{q}$ and $\mathbf{p}$ do not need to be raster points, and are represented by barycentric coordinates.

The direction $\mathbf{w}$ of the vector field at $\mathbf{p}$ is obtained by barycentric interpolation using the barycentric coordinates of $\mathbf{q}$. Multiplying $\mathbf{w}$ with a suitable factor and moving into the direction of $\mathbf{w}$ yields a new point $\mathbf{q}'$ close to the DDF surface. $\mathbf{q}'$ is processed in the same way like $\mathbf{q}$ which yields a subsequent point $\mathbf{p}'$ on the approximative streamline.

Figure 8.13 shows an example calculated according to this approach. The streamlines follow the direction of maximum curvature (red). The seed points of the streamlines are taken from a streamline of minimum curvature (green). The left picture shows a side view, the right picture a top view of the surface.

## 8.4 Simulation of Material Removal on DDF-Surfaces

Let us given a workpiece $W_s$ in its current state of processing, a tool path $P$, and a tool $T$. The goal of a simulation of material removal is to calculate the state $W_g$ of the workpiece after moving $T$ along $P$ where the regions are removed from $W_s$ which are reached by $T$.

For the simulation of material removal by DDFs, $W_s$ is represented by a DDF. $T$ is given by a closed triangular mesh which defines the boundary of $T$. Simulation is performed by placing $T$ at every point of a sufficiently dense sequence of sampling points on the tool path, in an orientation available from the tool path planning. At every position, material is erased by applying the DDF depth buffer algorithm to $T$. The depth buffer algorithm replaces the stored DDF height values with the minimum of the heights of the triangles of $T$ and of the stored height values and thus achieves the desired reduction of the workpiece. The resulting DDF with updated height fields is reported as DDF-representation of $W_g$.

Figure 8.14 shows results of a simulation of material removal caused by a cylindric tool moving along lines of curvatures of the surface. The surface is represented by a DDF with a resolution of $m = 400$ on a single support triangle. It requires 189 seconds to place the tool at 216 positions of the line of maximum curvature and to cut the stubbles of the DDF at the 200 tool triangles. For the measurements an AMD Athlon XP1700 processor with 1 GB RAM has been used.

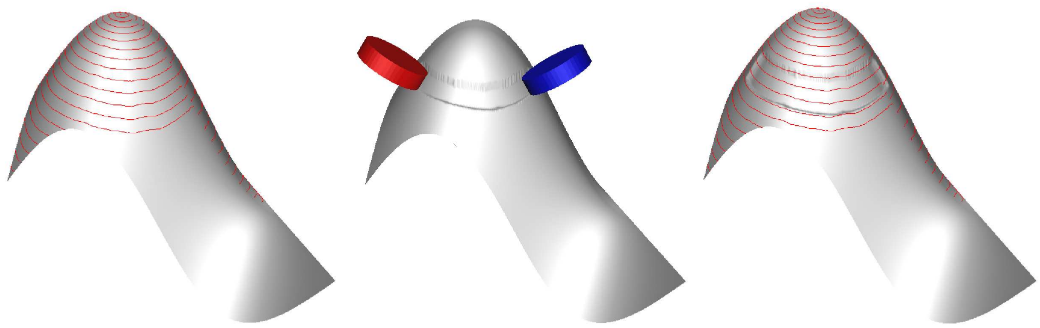


**Figure 8.14:** *The left picture shows lines of maximal curvature calculated on a DDF representation. The picture in the middle shows the result of a simulation of material removal by a cylindric tool. The tool is shown in two locations on its path. The right picture shows the result after simulation of the motion of the tool along two neighboring path segments.*

Some care must be taken with respect to the precision of the resulting surface. A necessary condition for correctness of the result is that the vectors which start at the surface of the workpiece $W_s$ represented by the DFF, which are co-linear to $\mathbf{V}$, and which are directed into the interior of $W_s$, intersect $T$ at most once. We call this condition of $T$ $\mathbf{V}$-*convexity*. For instance, $\mathbf{V}$-convexity might be violated when "drilling" a hole into the DDF. The result is a bad approximation of the hole. Figure 8.15 shows a violation of $\mathbf{V}$-convexity and illustrates this effect. Difficulties of this type are also known from conventional height fields. A possibility to remedy the problem is to use dexel fields [Hua94] instead of height fields, also for DDFs, if the application requires such unfavorable subtraction. However, often the milling tool, and even more a grinding tool, does not penetrate very deeply into the surface of a workpiece. Furthermore, the error mainly occurs at the flanks of the groove generated by the tool. The flanks, however, are usually removed by the neighboring path segments. Thus, the final surface is normally covered by regions located closely around the center line of the grooves. In these regions, the troubles are negligible if the displacement vectors do not deviate too much, say more than 45°, from the normals of the surface of $W_g$.

If the amount of material, possibly removed in several steps, exceeds the thickness of the crust of the support of $W_s$, the support of the workpiece in its current state has to be updated from time to time. A straightforward approach is to use the currently represented surface as new support, and reduce it by the approach of chapter 6.

The required resolution of the height-field-raster depends on the "degree" of the details of $T$ and the occurrence of sharp edges. One measure of detail is the size of the triangles of $T$. Small triangles lead to a higher degree of detail than large triangles. A general rule is that each triangle $t$ of $T$ should be hit by at least a small number of $\mathbf{V}$-rays shot from the raster on $S$. Thus, the choice of the raster resolution depends on the size and direction of $t$ with respect to $\mathbf{V}$. It also depends on the "curvature" of $\mathbf{V}$ which is characterized by the amount of deviation of the direction of neighboring vectors. For example, on a concave side of $S$, the density of vector tips is less than the density of their starting points on $S$.

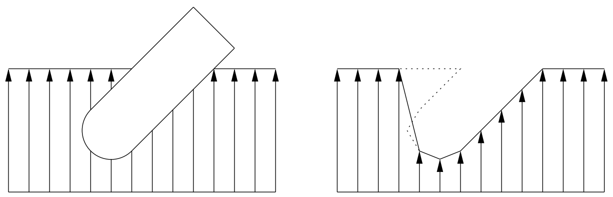


**Figure 8.15:** *A violation of* $\mathbf{V}$*-convexity (left) and the resulting bad approximation of a "drilled hole" indicated by the dashed contour in the right drawing.*

The concept of an item buffer used by the algorithm of conversion into a DDF-representation of chapter 6 can be used for simulation, too. The tool triangles which have an effect on the resulting surface might be stored as items. The information on the items can be used to choose the resolution for explicit extraction of the surface represented by the DDF according to the requirements of application, and to calculate the mesh of the surface by the adaptive approach of chapter 7. Instead of extremely high resolutions, an adaptation of approaches to feature sensitive surface reconstruction by insertion of additional points [Bot01] may also be useful.

### 8.4.1 Distance Between Tool Path Segments

The tool path segments have to be arranged in a distance so that the difference between the produced surface and the desired surface is everywhere less than a given error bound.

A possibility to find tool paths satisfying this property is to start with a family of tool path segments with a distance which has been determined according to some heuristics. An example is the approach used for the generation of the path segments of figure 8.14. Then the difference between the produced and the desired surface is determined. The difference can be determined by using the same support and resolution for the representation of the produced and the desired surface by DDFs. This approach leads to two discrete height fields of the same resolution. From the difference of the heights at the same raster point, the difference between the two surfaces in direction of the displacement vector can be determined. If the displacement vectors do not deviate too much from the normal vectors of both surfaces, this difference is a reasonable measure of the deviation of the two surfaces.

Figure 8.16 shows the result of a grinding simulation and a visualization of the resulting error measured by the amount of the differences just outlined. Because the cylindric tool is flat and the desired surface is curved, the tool is in contact with the desired surface only close to the centerline of the tool path. The error increases with increasing distance from the centerline, and becomes worst along the borderlines of the grooves. The next path can be started at a seed point which is in a region where the deviation becomes too large.

In the grinding simulation of figure 8.16 the surface is represented by 100 DDFs with a resolution of $m = 50$. It requires 24 seconds to place the tool at 217 path locations and to cut the stubbles of the DDFs at the 200 tool triangles.

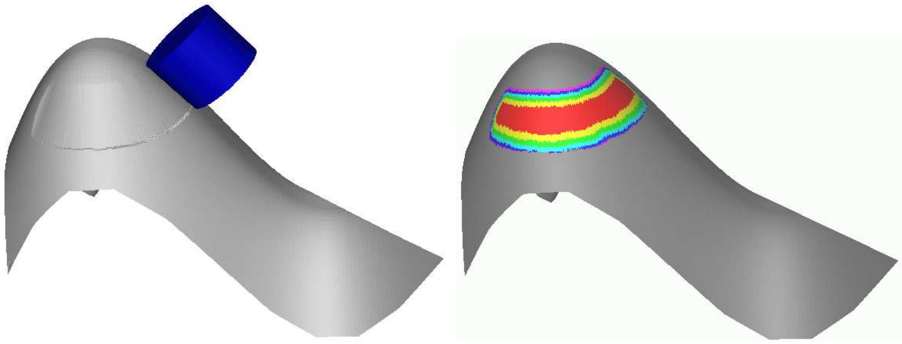


**Figure 8.16:** *Simulation of grinding with a cylindric tool. The left picture shows the resulting groove generated by the tool shown in the figure. The amount of resulting error between the desired surface and current surface is visualized by colors. For this purpose, the range of error values has been divided into intervals to each of which a color has been assigned.*

### 8.4.2 Correction of Local Surface Penetrations

An approach to positioning of a tool $T$ is to define a contact point $\mathbf{t}$ on the surface of $T$. $T$ is moved, so that $\mathbf{t}$ follows the tool path which is a path on the goal surface. Furthermore, $T$ is oriented so that, in the environment of the contact point $\mathbf{t}$, the tool and the surface share a tangent plane through $\mathbf{t}$. In many situations this configuration ensures that the tool does not penetrate the surface (figure 8.17 a). However, there are also many situations in which this approach causes a penetration of the workpiece and the tool (figure 8.17 b).

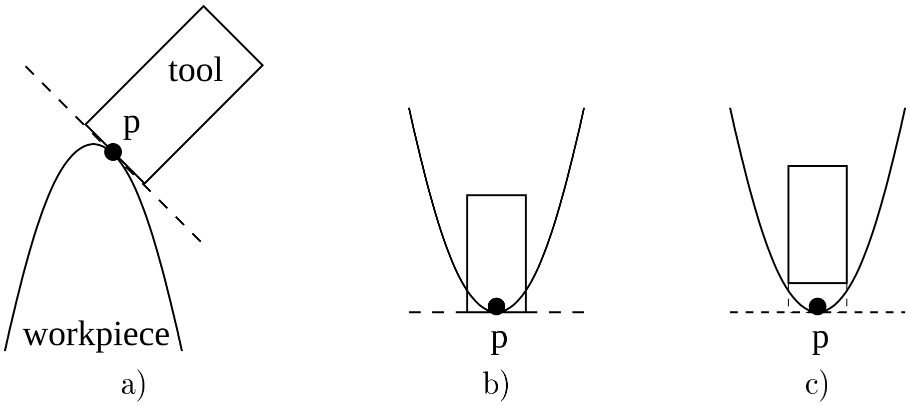



**Figure 8.17:** *(a) A penetration-free contact of a tool and a workpiece with a common separating tangent plane. (b) The tool and the workpiece intersect although they share a tangent plane in the contact point* $\mathbf{p}$*. (c) The penetration has been resolved by translating the tool perpendicularly to the tangent plane.*

A possibility resolving the problem sometimes is to change the position of the tool locally in order to avoid penetration. A common approach is to change the declination of the tool axis [Li94, Lee98] but to leave the workpiece in contact at the points of the given tool path. Another solution is to leave the orientation unchanged, but to give up the contact point by moving the tool perpendicularly to the tangent plane until a non penetrating contact is reached (figure 8.17 c). This is a good solution especially for grinding processes with a grinding disc as tool.

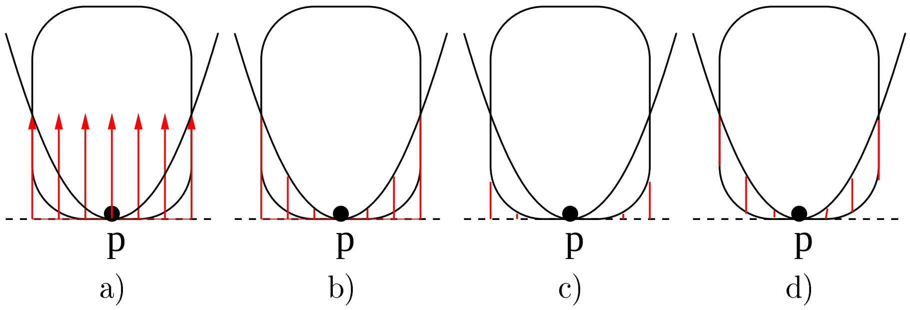


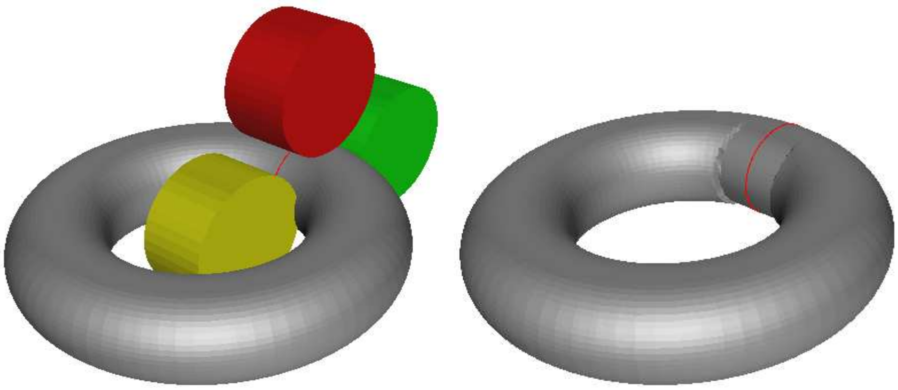

**Figure 8.18:** *(a) A workpiece (indicated by the parabola) and a tool in penetrating contact. The vectors indicate a tangential DDF which is established on the tangent plane of the contact point. (b) The height field $h_W$ which is obtained by cutting the displacement vectors with the workpiece surface. (c) The height field $h_T$ which is obtained by cutting the displacement vectors with the tool surface. (d) The difference between the two height fields. The longest vector determines the amount by which the tool has to be translated in order to resolve the penetration.*

The amount of tool displacement can be calculated with the help of a *tangential DDF*. On the tangent plane of the contact point of the surface, a DDF is established which has a subset of the tangent plane as support surface and a displacement vector field perpendicular to the support surface (Figure 8.18 a). The vectors of the tangential DDF are cut at the triangles of the workpiece surface represented by the workpiece-DDF (figure 8.18 b). The result is a height field $h_W$. Then the original vectors of the tangential DDF are cut for a second time by the surface of the tool (figure 8.18 c). The result is a height field $h_T$. The maximum difference between corresponding height values of $h_W$ and $h_T$ determines the amount by which the tool has to be translated in order to resolve the local penetration of both surfaces (figure 8.18 d).

Figure 8.19 a) shows a three-dimensional example. A cylindric grinding roll is moved around a torus. The torus is the goal surface, and the circular tool path lies on this surface. Figure 8.19 b) illustrates the result of a simulation of grinding. The circular tool path is indicated in red color on the torus surface. Because of the non-convex shape of the inner region of the torus, the roll penetrates the surface and material is removed in an undesirable way.

Figure 8.20 illustrates one of the tangential DDFs used for the necessary correction of the path. Figure 8.20 a) shows a box surrounding the tool in which the tangential DDF is established, that is the crust has the thickness of the box. Figure 8.20 b) shows the result after cutting the crust with the surface of the tool. Figure 8.20 c) shows the differences of the tool and surface height fields, as well as the parts of the two surfaces within the box.

Vector cutting can be performed by the DDF depth buffer algorithm. However, it is more efficient to use the classical depth buffer algorithm for parallel projection in this case. It is more efficient, in particular if a hardware implementation of a three-dimensional graphics board is employed to execute this frequently called operation.

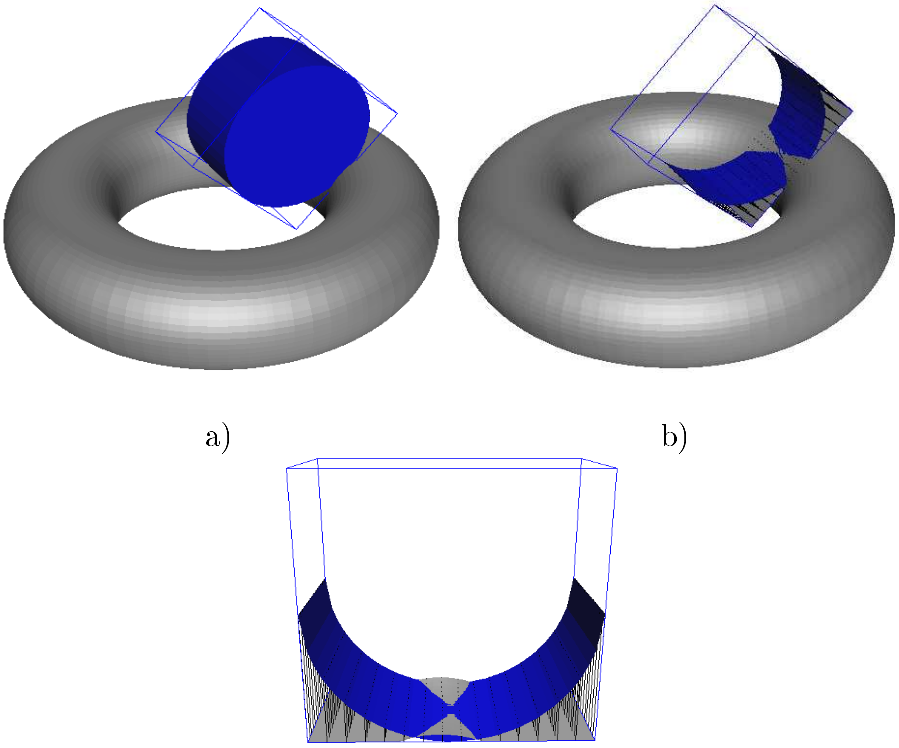


a) b)

**Figure 8.19:** *(a) A grinding roll is moved around the torus. The torus is the desired surface and the tool path is on the surface. (b) The result of the grinding simulation shows that the roll penetrates the surface in an undesirable way.*

For simulation of the the entire tool path, the tool has to be placed and corrected for numerous points of the tool path. The number of points, and thus the calculation time, can be reduced by interpolating the amount of displacement at points in-between of some selected points chosen for precise calculation.

We have measured the performance of an implementation for the example of Figures 8.19 and 8.20. The circular tool path around the torus consisted of 400 tool positions. The resolution of the tangential DDFs has been $50 \times 50$. The DDF depth bufer has been used for cutting. For evaluation of the tangential DDF at all of the 400 positions of the path, 9.8 seconds were required. Using just each 25th



a) b)

c)

**Figure 8.20:** *Illustration of one of the tangential DDFs used for path correction. (a) A bounding box surrounding the tool is used to define the thickness of the crust of the tangential DDF. (b) The result after cutting the crust with the surface of the tool. (c) The differences of the tool and surface height fields, as well as the parts of the two surfaces within the box.*

point reduced the computation time to 0.6 seconds, without visible penetration of the torus by the tool on the interpolated path. A further reduction to every 50th point causes penetrations of the torus surface (figure 8.21). Timings have been measured on a personal computer with AMD Athlon XP1700 processor with 256 MB RAM. The programming language has been Java 1.3i.

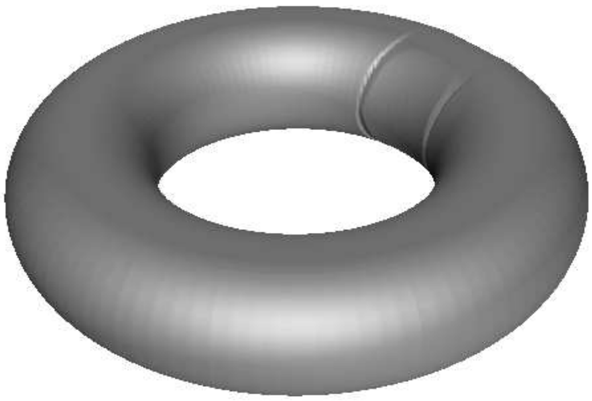


**Figure 8.21:** *A path correction reduced to every 50th point of the tool path causes penetrations of the surface.*

## 8.5 Collision Detection

For the tool path planning on NC machines it is important to avoid undesired collisions of the tool and the workpiece, or the machine and the workpiece as well. For example in a milling process it is desired that the top of the milling cutter is in contact with the workpiece. The shank of the milling cutter and the machine part holding the milling cutter should not come in contact. Methods for the avoidance of collisions can be found in studies about motion planning in robotics [Lat91].

If the workpiece is given in a DDF-description, the DDF depth-buffer algorithm of section 5 can be used to detect collisions. Let $T$ be a triangle set describing the surface of the collision object. The triangles $T$ are used as input for the DDF depth-buffer of the workpiece-DDF. The first step of the DDF depth-buffer, *localization*, detects pairs of workpiece triangles and triangles $T$ possibly affecting each other. This can be used for a fast but rough collision detection. For a correct detection the $\mathbf{V}$-shooting operation has to be performed on the triangle pairs to determine whether the vectors on a workpiece triangle really intersect a triangle of the collision set T.

# Chapter 9

# Deformation

In computer graphics, two sorts of deformations can be distinguished, warping and morphing. Warping means to deform a given shape continuously. Morphing means a continuous sequence of shapes which transfers a given start shape into a given goal shape. Figure 9.1 shows snapshots from a morphing sequence between a sphere and a cube. Extensive surveys on warping and morphing have been given by Ruprecht [Rup94], and, in particular for images, by Wolberg [Wol90].

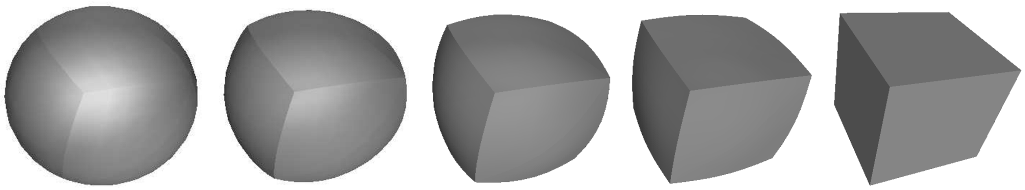


**Figure 9.1:** *Morphing between a sphere and a cube.*

Warping of a surface represented by a DDF can easily be achieved by modifying the height function $h$.

Another possibility is to modify the vectors of the displacement field at the vertices of the support mesh. If $h$ is left unchanged, the effect can be a deformation of the given shape which, however, does not change the details of the shape, up to a certain distortion. Figure 9.2 a) depicts a flat DDF-surface with engravings. An increase of the displacement vector length enlarges the engraving depth (figure b) and an inversion of the vector directions inverts the direction of the engravings, too (figure c).

A third possibility is to warp the support mesh by changing the location of its vertices, like Kobbelt et al. [Kob00] did for a geometric representation which separates the rough shape from the details of a surface. In this way, a

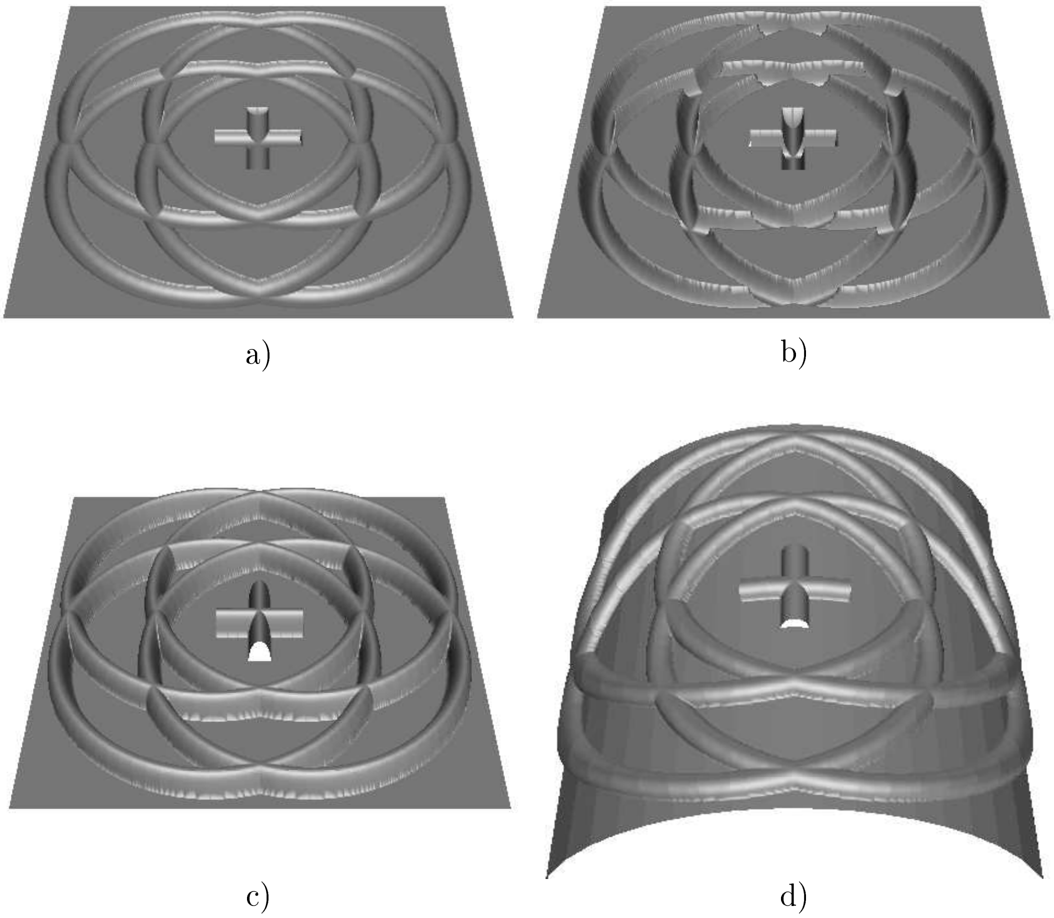



**Figure 9.2:** *Effects of manipulations of the displacement vector field and of the support mesh. (a) A flat DDF-surface with engravings. (b) Result of enlarging the displacement vectors by a factor +3. (c) Enlarging the displacement vectors by a factor −3 reverses the surface details. (d) The points of the support mesh are transferred on a cylindric patch.*

deformation can be achieved which modifies the crude shape but lets the surface details unchanged, up to distortions. In figure 9.2 d) the points of the support mesh are transferred on a cylindric patch.

Morphing sequences can immediately be generated, if the start shape and the goal shape are represented over the same support and the same displacement vector field. Figure 9.3 shows a sphere and a cube which are represented in this way. A continuous sequence of intermediate shapes is obtained by applying a method of continuous interpolation between the height field of the start and the goal shape. The most straightforward approach is linear interpolation. In this case, a height value $h(t)$ at "time" $t$, $t \in [0,1]$, is obtained from a given height value $h_0$ of the start shape and the given height value $h_1$ of the goal shape by $h(t) := (1-t) \cdot h_0 + t \cdot h_1$. The intermediate shapes between the sphere and the cube shown in figure 9.1 have been obtained by linear interpolation with $t \in \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$, $h_0 = 0$ and $h_1 = 1$.

The main difficulty of this approach is to find a support mesh and a displacement vector field which allow the DDF-representation of both given surfaces. For the example of figure 9.3, a DDF-representation of the cube has been determined first, which has then been used for the sphere by cutting the dispacement vectors with the polygons of an approximation of the sphere using the DDF depth-buffer algorithm. This approach can be applied with reasonable efforts for simple shapes, e.g. convex or star-shaped shapes, or start and goal shapes which do not deviate too much from each other. The latter situation is usually given in the case of mould planning for production by deformation.

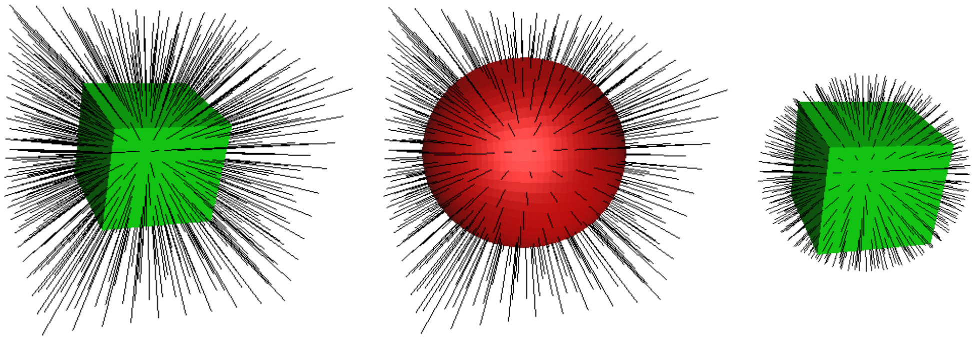


**Figure 9.3:** *Morphing with DDFs. A cube (left) and a sphere (middle) are represented by DDFs with the same support and the same displacement vector field. In the picture on the right, the surface of the cube is represented by polygons, and the sphere is represented implicitly by the points at the tips of the vectors.*

Simulation-based mould planning can be performed as follows. First the desired goal shape is defined by the shape of the mould. Furthermore, a start shape is given. By numerical simulation of the real deformation process, the start shape is deformed. The simulation takes into account physical parameters of the material of the start shape. One goal of the simulation is to detect critical locations on the surface of the workpiece caused by the flow of material during deformation. The flow of material may cause thin regions on the final shape, or even cracks. Another issue is that the deformed shape does not necessarily coincides with the desired shape. Figure 9.4 shows the result of a simulation carried out at the LFU, University of Dortmund. The red surface has been deformed by pressing it from above into a mould given by the green surface. Because of characteristics of the material, the red surface does not match completely with the surface of the mould. The strong bendings of the desired shape are not reached by the red surface.

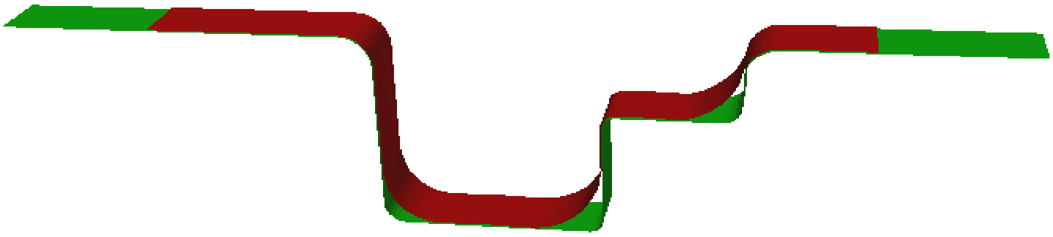


**Figure 9.4:** *The result of a computational simulation of the deformation of a plate carried out at the LFU, University of Dortmund. The originally flat plate has been pressed into a mould represented by the green surface. The red surface is the result of the deformation.*

If troubles of this kind occur, one approach is to modify the goal shape, so that it meets the requirements of the deformation process. A possibility is to take a goal shape which is in-between the original shape and the one resulting from simulation. Such an intermediate shape can be obtained from a morphing sequence between the two given shapes. Figure 9.5 illustrates this approach for the example of figure 9.4. A DDF has been established for the green surface in a straightforward manner. The triangular mesh representing the green surface is the support. The vectors of the displacement field at the vertices are averages of the normals of the incident triangles of the vertices. The lengths of the vectors have been chosen so that the red surface is in the crust. As can be seen from intersecting line segments in the figure, the unique-projection property is hurt. A height field representing the red surface has been generated by applying the DDF depth-buffer algorithm with the red surface. For the example shown in the figure, a resolution of 2 of the raster on the support triangles has been used. This means that just the vectors at the vertices have been considered. The blue intermediate surface in the figure has been obtained by interpolation between the height field of the green surface and the height field of the red surface with a

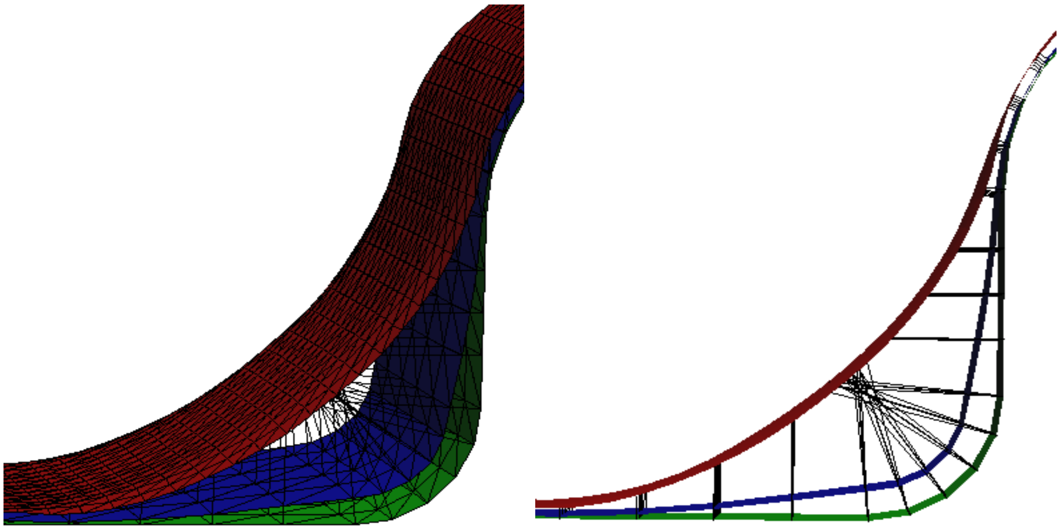


**Figure 9.5:** *Calculation of an intermediate surface, drawn in blue, between the green and the red surface.*

blending function $d$,

$$\mathbf{f}(u,v) = \mathbf{s}(u,v) + d(u,v) \cdot h(u,v) \cdot \mathbf{v}(u,v)$$

where $\mathbf{s}$ corresponds to the green surface, $\mathbf{f}$ to the blue surface, and $h$ is the height field defining the red surface at a location $(u,v)$ of the support surface. In the example, $d(u,v)$ is a constant function.

In figure 9.5, $d(u,v)$ has been chosen so that the green surface is still in the unique-projection region of the displacement vector field. If deformations are desired which enter the region of crossing vectors, a possibility is to split the deformation into a sequence of smaller deformations. Every deformation of the sequence is in the unique-projection region of its DDF. The resulting surface of a deformation is taken as the support surface of the next deformation in the sequence. In this way, the displacement vector field adapts its direction to the falling curvature, and thus extends the region in which the unique-projection property holds.

# Chapter 10

# Conclusions

We have demonstrated the usefulness of DDFs for the simulation of processes of material removal and deformation in mechanical engineering. The usefulness is based on efficient algorithms which have been presented in this thesis. The central algorithm is the DDF depth-buffer algorithm. It transfers the idea of the classical depth-buffer or z-buffer algorithm to a "curved" image plane and a "curved" projection. We have shown that rasterization can still be performed to a large extent incrementally. Incremental evaluation is one source of speed of the conventional depth-buffer algorithm. The depth-buffer algorithm is used by today's 3D graphics boards for visibility calculation, which achieve their impressive performance by hardware adapted to the requirements of the z-buffer algorithm. An interesting question for further research is whether hardware-supported features of OpenGL-based graphics can be immediately used for the implementation of the DDF depth-buffer algorithm, or which hardware extensions are necessary or useful for its efficient hardware support.

Another field we have treated is vector fields on DDFs and calculation of streamlines in the vector fields. Algorithms for streamline calculation have been presented which follow the usual approach of incremental calculation, but which use the special structure of DDFs. As particular example, the vector fields related to maximum and minimum curvature have been used. In section 8.1 we have briefly mentioned that vector fields may serve as a uniform mechanism of specification of constraints on milling paths. The curvature of the surface is one important constraint which is considered by the curvature vector fields. This basic idea should be further pursued. An important aspect in particular is to take into consideration possible collisions between the tool and the workpiece.

The problem of collision detection and avoidance and the possibility of using mechanisms of the DDF depth-buffer algorithm for its treatment have been outlined in section 8.5. However, we have not treated the aspect of global collision detection in this thesis. "Global collision" means collision between the tool and the workpiece outside the cutting part of the tool, which of course has to be avoided. An interesting question of future research is how the constraint of collision avoidance can be considered in the tool flow vector field.

In chapter 9, we have described the application of DDFs for the specification of warpings and morphings. As we have seen in figure 9.4, it may happen in real applications that the displacement vector field does not have the unique-projection property in regions of high curvature. We have indicated a possible approach to overcome this difficulty. This approach, and possibly other solutions, should be worked out further, in particular in the context of real applications in mechanical engineering. Constraints on deformations in this application should be identified together with specialists of this field, and taken into account in the algorithms of deformation.

# Bibliography


[Aya01] J. Ayasse, H. Müller, Höhenfelder auf gekrümmten Flächen, Begleitband zum Kolloqium der DFG-Forschergruppe 366 "Simulationsgestützte Offline-Prozessplanung und -optimierung bei der Fertigung von Freiformflächen", Universität Dortmund, ISBN 3-00-007642-5, 2001, 3–14

[Aya02] J. Ayasse, H. Müller, Sculpturing on discrete displacement fields, Proc. Eurographics 2002, Computer Graphics Forum 21(3), 2002, 431–440

[Aya03] J. Ayasse, H. Müller, Vector-field-based Path Planning with Discrete Displacement Fields, Begleitband zum 2. Kolloqium der DFG-Forschergruppe 366 "Simulationsgestützte Offline-Prozessplanung und -optimierung bei der Fertigung von Freiformflächen", Universität Dortmund, ISBN 3-00-010891-2, 2003, S.2-20

[Ben97] M.O. Benouamer, D. Michelucci, Bridging the gap between CSG and brep via a triple ray representation, Proc. 4th ACM Symp. on Solid Modeling and Appl., pp. 68-79, 1997

[Bla73] W. Blaschke, K. Leichtweiss, Elementare Differentialgeometrie, Springer-Verlag, Berlin, 1973

[Bot01] M. Botsch, L. Kobbelt, Resampling feature and blend regions in polygon meshes for surface anti-aliasing, Proc. Eurographics 2001, Computer Graphics Forum 20(3), 2001, C-402–C-410

[Bro91] I.N. Bronstein, K.A. Semendjajew, Taschenbuch der Mathematik, B.G. Teubner Verlagsgesellschaft 1991, ISBN 3-8154-2000-8

[Chi02] I.J. Chiou, Y.S. Lee, A machining potential field approach to tool path generation for multi-axis sculptured surface machining, CAD 34, 2002, 357–371

[Cho99] B.K. Choi, R.B. Jerard, Sculptured Surface Machining: Theory and Applications, Kluwer Academic Pub., 1999

[Cig99] P. Cignoni. C. Montani, C. Rocchini, R. Scopignio, M. Tarini, Preserving attribute values on simplified meshes by re-sampling detailed textures, The Visual Computer 15, 1999, 519–539

[Cla88] R.D. Clay, H.P. Moreton, Efficient adaptive subdivision of Bezier surfaces, Proc. Eurographics 1988, pp. 357–372, North-Holland, Amsterdam, 1988

[Coo84] R. Cook, Shade trees, Poc. SIGGRAPH 1984, Computer Graphics 18(3), 1984, 223–231

[Din03] S. Ding, M.A. Mannan, A.N. Poo, D.C.H. Yang, Z. Han, Adaptive isoplanar tool path generation for machining of free-form surfaces, CAD 35, 2003, 141–153

[Dog01] M. Doggett, A. Kugler, W. Straßer, Displacement mapping using scan conversion hardware architectures, Computer Graphics Forum 20(1), 2001, 13–26

[Dra97] D. Dragomatz, S. Mann, A classified bibliography of literature on NC milling path generation, CAD 29, 1997, 239–247

[Duc03] M. Duchaineau, K. Joy, Progressive Precision Surface Design, to appear in: Geometric Modeling and Visualization (G. Brunnett, B. Hamann, H. Müller, L. Linsen, eds.), Springer-Verlag, 2003

[Elb94] G. Elber, E. Cohen, Accessibility in 5-axis milling environment, CAD 26, 1994, 490–496

[Elb97] G. Elber, R. Fish, 5-axis freeform surface milling using piecewise ruled surface approximation, ASME Journal of Manufacturing Science and Engineering 119, 1997, 383–387

[Ell91] J.L. Ellis, G. Kedem, T.C. Lyerly, D.G. Thielman, R.J. Marisa, J. Menon, H.B. Voelcker, The ray casting engine and ray representations: a technical summary, International Journal of Computational Geometry and Applications 1(4): 347–380, 1991

[Flu02] A. Flutter, J. Todd, A machining strategy for toolmaking, CAD 33, 2001, 1009–1022

[Fol90] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, Fundamentals of Interactive Computer Graphics, Addison-Wesley, 1990

[Fri00] S.F. Frisken, R.N. Perry, A.P. Rockwood, T.R. Jones, Adaptively sampled distance fields: A general representation of shape for computer graphics, Proc. SIGGRAPH 2000, pp. 249–254, 2000

[Gal91] T.A. Galyean, J.F. Hughes, An interactive volumetric modeling technique, In Proc. SIGGRAPH 1991, pp. 267–274, 1991

[Gar79] M.R. Garey, D.S. Johnson, Computers and intractability – a guide to the theory of NP-completeness, W.H. Freeman and Comp., New York, 1979

[Gla99] G. Glaeser, J. Wallner, H. Pottmann, Collision-free 3-axis milling and selection of cutting tools, CAD 31, 1999, 225–232

[Got96] S. Gottschalk, M.C. Lin, D. Manocha, OBBTree: A hierarchical structure for rapid interference detection, Proc. SIGGRAPH 1996, pp. 171–180, 1996

[Gum99] S. Gumhold, T. Hüttner, Multiresolution rendering with displacement mapping, Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware, 1999, 55–66

[Hel91] M. Held, On the computational geometry of pocket machining, Lecture Notes in Computer Science 500, Springer-Verlag, 1991

[Ho01] S. Ho, S. Sarma, Y. Adachi, Real-time interference analysis between a tool and an environment, CAD 33, 2001, 935–947

[Hoo86] T. van Hook, Real-time shaded NC milling display, Proc. SIGGRAPH 1986, 15–20

[Hop96] H. Hoppe, Progressive meshes, Proc. SIGGRAPH 1996, pp. 99–108, 1996

[Hop99] H. Hoppe, New quadric metric for simplifying meshes with appearance attributes, Proc. IEEE Visualization 1999, 59–66

[Hos92] J. Hoschek, D. Lasser, Grundlagen der geometrischen Datenverarbeitung, 2. Aufl., B.G. Teubner, Stuttgart, 1992

[Hua94] Y. Huang, J.H. Oliver, NC milling error assessment and tool path correction, In Proc. SIGGRAPH 1994, 287–294

[Hui94] K.C. Hui, Solid sweeping in image space – application in NC simulation, The Visual Computer 10, 1994, 306–316

[Iva01] D. Ivanov, Y. Kuzmin, Spatial patches – a primitive for 3D model representation, Proc. Eurographics 2001, Computer Graphics Forum 20(3), 2001, C-511–C-521

[Jen02] C.G. Jensen, W.E. Red, J. Pi, Tool selection for five-axis curvature matched machining, CAD 34, 2002, 251–266

[Jer89] R.B. Jerard, S.Z. Hussaini, R.I. Drysdale, B. Schaudt, Approximate methods for simulation and verification of numerically controlled machining programs, The Visual Computer 4, 1989, 329–348

[Kim02] T. Kim, S.E. Sarma, Toolpath generation along directions of maximum kinematic performance: a first cut at machine-optimal paths, CAD 34, 2002, 453-468

[Kob00] L. Kobbelt, T. Bareuther, H.-P. Seidel, Multi-resolution shape deformation for meshes with dynamic vertex connectivity, Eurographics 2000, Computer Graphics Forum 19(3), 2000, C-249–C-259

[Kra96] F.L. Krause, J. Lüddemann, Virtual clay modeling, Proceedings IFIP WG 5.2, Geometric Modeling for CAD, M. Pratt, London, Chapman and Hall, 1996

[Kri96] V. Krishnamurthy, M. Levoy, Fitting smooth surfaces to dense polygon meshes, Proc. SIGGRAPH 1996, 313–324

[Lat91] J.C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, Boston, 1991

[Lee00] A. Lee, H. Moreton, H. Hoppe, Displaced subdivision surfaces, Proc. SIGGRAPH 2000, 85-94

[Lee97] Y.S. Lee, Admissible tool orientation control of gouging avoidance for 5-axis complex surface machining, CAD 29, 1997, 507–521

[Lee98] Y.S. Lee, Non-isoparametric tool path planning by machining strip evaluation for 5-axis sculptured surface machining, CAD 30, 1998, 559–570

[Len99] C. Lennerz, E. Schömer, T. Warken, A framework for collision detection and response, In Proc. 11th European Simulation Symposium (ESS'99), 309–314

[Li94] S.X. Li, R.B. Jerard, 5-axis machining of sculptured surfaces with a flat-end cutter, CAD 26, 1994, 165–178

[Lue97] D. Luebke, E. Carl, View-dependent simplification of arbitrary polygonal environments, In Proc. SIGGRAPH 1997, 199–208, 1997

[Mar94] S. Marshall, J.G. Griffiths, A survey of cutter path construction techniques for milling machines, International Journal of Production Research 32, 1994, 2861–2877

[Mea82] D. Meagher, Geometric modeling using octree encoding, Computer Graphics and Image Processing 19:129–147, 1982

[Mey99] H.W. Meyer, F. Albersmann, Wochenpensum in zwei Stunden: Features und NC-Sets verkürzen die NC-Programmierung im Formenbau, Special Tooling 5, 1999, 42–48

[Mue88] H. Müller, Realistische Computergrafik - Algorithmen, Datenstrukturen und Maschinen, Informatik-Fachberichte 163, Springer-Verlag, 1988, p. 146

[Mue03] H, Müller, T. Surmann, M. Stautner, F. Albersmann, K. Weinert, Online Sculpting and Visualization of Multi-Dexel Volumes, In: Proc. 8th ACM Symp. on Solid Modeling and Appl., 2003

[Nak97] K. Nakamaru, Y. Ohno, Breadth-First Ray Tracing Utilizing Uniform Spatial Subdivision, IEEE Trans. on Visualization and Computer Graphics, 4(4), 1997, 316–326

[Oli00] M. Oliveira, G. Bishop, D. McAllister, Relief texture mapping, Proc. SIGGRAPH 2000, 359–368

[Oli90] J.H. Olivier, E.D. Goodman, Direct dimensional NC verification, CAD 22, 1990, 3–10

[Par02] S.C. Park, Y.C. Chung, Offset tool-path linking for pocket machining, CAD 34, 2002, 299–308

[Ped94] H. Pedersen, Displacement mapping using flow fields, Proc. SIGGRAPH 1994, 279–286

[Per01] R.N. Perry, S.F. Frisken, Kizamu: A system for sculpting digital characters, Proc. SIGGRAPH 2001, pp. 47–56, 2001

[Pos93] F. Post, T. van Walsum, Fluid flow visualization, in: H. Hagen, H. Müller, G.M. Nielson, Focus on Scientific Visualization, Springer-Verlag, Berlin, 1993

[Pot99] H. Pottmann, J. Wallner, G. Glaeser, G. Ravani, Geometric criteria for gouge-free three-axis milling of sculptured surfaces, ASME J. of Mechanical Design 121, 1999, 241–248

[Pre88] F. Preparata, M.I. Shamos, Computational geometry – an introduction, 2nd ed., Springer-Verlag, New York, 1988

[Req80] A.A.G. Requicha, H.B. Voelker, Representations for solids: theory, methods, and systems, ACM Computing Surveys 12, 1980, 437–464

[Roe00] C. Rössl, L. Kobbelt, H.-P. Seidel, Line art rendering of triangulated surfaces using discrete lines of curvature, Proc. 8th International Conference in Central Europe on Computer Graphics (WSCG'00), 2000, 168–175

[Ros93] J. Rossignac, P. Borrel, Multi-resolution 3D approximations for rendering complex scenes, Modeling in Computer Graphics, Springer-Verlag, 1993, pp. 445–465

[Rot80] S.D. Roth. Ray casting for modeling solids, Computer Graphics and Image Processing 18: 109–144, 1980.

[Rup94] D. Ruprecht, Geometrische Deformationen als Werkzeug in der Graphischen Datenverarbeitung, Dissertation, FB Informatik, Universität Dortmund, Shaker Verlag, 1994

[Rup98] D. Ruprecht, H. Müller, A scheme for edge-based adaptive tetrahedron subdivision, Mathematical Visualization (H.-C. Hege, K. Polthier, eds.), pp. 61–70, Springer-Verlag, 1998

[Sai91] T. Saito, T. Takahashi, NC machining with G-buffer method, Proc. SIGGRAPH 1991, pp. 207–216, 1991

[Sch92] W.J. Schroeder, J.A. Zarge, W.E. Lorensen, Decimation of triangle meshes, Proc. SIGGRAPH 1992, pp. 65–70, 1992

[Sta03] Stanford data archive, http://graphics.stanford.edu

[Sto99] A. Storr, H. Ströhle, Rechnerunterstütze Produktion mit fünfachsigen spanenden Bearbeitungseinrichtungen, wt Werkstattstechnik 89, 1999, 78–82

[Tan98] J.W.H. Tangelder, J.S.M. Vergeest, M. Overmars, Interference-free NC machining using spatial planning and Minkowksi operations, CAD 30, 2000, 277–286

[Tur92] G. Turk, Re-Tiling polygonal surfaces, Proc. SIGGRAPH 1992, pp. 55–64, 1992

[Weg84] H. Weghorst, G. Hooper, D.P. Greenberg, Improved computational methods for ray tracing, ACM Transactions on Graphics 3(1), 1984, 52–69

[Wei01] K. Weinert, T. Surmann, Approaches for Modeling Engagement Conditions in Milling Simulations, Proc. 4th CIRP International Workshop on Modelling of Machining Operations, C.A. van Lutterfeld (ed.), Morgan-Kaufman Publ., 2001, 67–70

[Wei02] K. Weinert, H. Müller, W. Kreis, T. Surmann, J. Ayasse, T. Schübstuhl, K. Kneupner, Diskrete Werkstückmodellierung, ZWF 97, 2002, 385–389

[Wol90] G. Wolberg, Digital image warping, IEEE Computer Society Press, Los Alamitos, CA, 1990

[Yan99] D.C.H. Yang, Z. Han, Interference detection and optimal tool selection in 3-axis NC machining of free-form surfaces, CAD 31, 1999, 303–315

[Zwi02] M. Zwicker, M. Pauly, O. Knoll, M. Gross, Pointshop 3D: An Interactive System for Point-Based Surface Editing, Proc. SIGGRAPH 2002, pp. 322–329, 2002