technische universität
dortmund

fakultät für informatik

# Projektgruppe
# Smoothed-Particle-Lab

Ein Software-System zur Modellierung und Simulation
dynamischer, physikalischer Systeme

2. Dezember 2010

**Teilnehmer:**
Sedat Colak, Jan Emmerich, Christoph Kilimann, Christian Leszinski,
Thomas Nitschke, Irina Prahova, Bartholomaeus Rudak, Daniel Wippler

**Betreuer:**
Dr. Frank Weichert
Dipl.-Inform. Denis Fisseler
Dipl.-Inform. Marcel Gaspar

Lehrstuhl Informatik VII (Graphische Systeme)
Technische Universität Dortmund

fakultät für
informatik

# Inhaltsverzeichnis

# 1. Mathematische Notation

| | |
|---|---|
| $\mathbf{a} = [a_1, ..., a_n]$ | Allgemeiner Vektor $a \in \mathbb{R}^n$ mit den Elementen $a_i$ |
| $\mathbf{a}_x$ | Element x des Vektors a |
| $\mathbf{p}_i(t) = [\mathbf{pos}_i(t), \mathbf{acc}_i(t), \mathbf{vel}_i(t)]$ | Partikel mit Index $i$ zum Zeitpunkt t |
| $\mathbf{pos}_i(t) = [x, y, z]$ | Position von Partikel mit Index $i$ zum Zeitpunkt $t$ |
| $\mathbf{vel}_i(t) = [x, y, z]$ | Geschwindigkeit von Partikel mit Index $i$ zum Zeitpunkt $t$ |
| $\mathbf{acc}_i(t) = [x, y, z]$ | Beschleunigung von Partikel mit Index $i$ zum Zeitpunkt $t$ |
| $\mathbf{p}_{max}$ | Maximale Partikelanzahl |
| $\Delta t$ | Zeitschritt |
| $\dot{x}$ | Erste Ableitung |
| $\ddot{x}$ | Zweite Ableitung |

# 2. Einleitung

Der Blutfluss durch die Ader eines Menschen, Meeresströmungen, Tsunamis und Vulkanausbrüche sind nur einige Beispiele für Naturphänomene, die auf den physikalischen Vorgang der Strömung zurückzuführen sind. Sie spielen in vielen verschiedenen Bereichen des menschlichen Wissens, wie z.B. der Astrophysik, der Medizin, der Ozeanographie und der Meteorologie eine wichtige Rolle und finden ihre praktische Anwendung in der Technik sowie in den angewandten Wissenschaften wie beispielsweise der Ballistik, der Aerodynamik oder der Bauphysik. Ferner werden Strömungen zur realitätsnahen Abbildung der Wirklichkeit in diversen Computerspielen erfolgreich eingesetzt. Wegen ihres breiten wissenschaftlichen Umfangs und der Allgegenwärtigkeit damit verbundener, natürlicher Gesetze und Phänomene, ist die Flüssigkeitsbewegung ein physikalischer Vorgang, der das menschliche Verständnis von der Aussenwelt weitgehend prägt. Daher ist ein formales Modell zur Beschreibung und Vorhersage von Fluidbewegungen fundamental, um unsere Welt verstehen, erforschen und nachahmen zu können. Das physikalische Verhalten von inkompressiblen Fluidbewegungen wird in der Hydrodynamik im Allgemeinen mit den Navier-Stokes-Gleichungen ausgedrückt. Hierbei handelt es sich um ein partielles Differentialgleichungssystem zweiter Ordnung, welches das Zweite Newtonsche Gesetz der Impulserhaltung sowie den Massenerhaltungssatz verkörpert. Die Navier-Stokes-Gleichungen sind jedoch zum Lösen von realen Problemen in wissenschaftlichen Bereich nicht direkt einsetzbar, da sie global nicht analytisch lösbar sind.

Die Navier-Stokes-Gleichungen sind jedoch numerisch mit Approximationsverfahren wie FEM (Finite-Elemente-Methode) und FDM (Finite-Differenzen-Methode) lösbar. Die Vorgehensweise bei der Lösung des partiellen Navier-Stokes-Differentialgleichungssystems durch die genannten numerischen Methoden besteht im Allgemeinen darin, das topologische Grundgebiet des Problembereichs zunächst zu diskretisieren und darauf aufbauend schliesslich die Ableitungen des Differentialgleichungssystems anzunähren. Die FEM- und FDM-Methoden haben ihren Ursprung in zwei verschiedenen mathematischen Ansätzen: dem Euler- und dem Lagrangeansatz. Der Euleransatz liefert eine raum-basierte und der Lagrangeansatz eine material-basierte Beschreibung der physikalischen Differentialgleichungen (siehe hierzu [LL03, Seite 5]). Infolgedessen wird bei der Diskretisierung des Grundgebiets eines hydrodynamischen Problems nach dem Euleransatz der Raum, der die Flüssigkeit enthält, und nach dem Lagrangeansatz die bewegte Flüssigkeit selbst diskretisiert. Diese Methoden werden als „gitterbasierte“ Methoden bezeichnet, da die Diskretisierung des Grundgebiets in der Form eines Gitters geschieht, das die Stützstellenpositionen zur Messung der Materialeigenschaften angibt. Obwohl FEM und FDM-Methoden häufig zum Lösen von hydrodynamischen Problemen eingesetzt werden und dabei gute Ergebnisse liefern, haben sie trotzdem gewisse Einschränkungen. So eignen sie sich für bestimmte hydrodynamische Problemstellungen, wie z. B. Berechnung von Materialeigenschaften einer Explosion oder eines Hochgeschindigkeitsaufpralls, nicht. Die FDM-Methode führt aufgrund ihres regulären Gitters häufig zu Problemen bei der Ermittlung der physikalischen Eigenschaften von komplexen Geometrien (siehe hierzu [LL03, Seite 12]). Hingegen führen bei der FEM-Methode große Deformationen der Fluidoberfläche zu starken Gitterdeformationen, die die Genauigkeit der Lösung des hydrodynamischen Problems.

Ein aktueller Ansatz zur approximativen Lösung der Navier-Stokes-Gleichungen, der die Nachteile der „gitter-basierten“ Methoden aufhebt, ist die Lagrangemethode SPH (Smoothed-Particle-Hydrodynamics). Bei dieser Methode wird das Grundgebiet der Flüssigkeit nicht durch ein Gitter, sondern durch eine endliche Anzahl an Stützstellen, den so genannten Partikeln, diskretisiert. Die Materialeigenschaften einer dieser Stützstellen werden durch die Interpolation der Materialeigenschaften der benachbarten Partikel ermittelt. Der wesentliche Vorteil dieser Methode liegt in ihrer Adaptivität, welche durch die gitterlose Diskretisierung ermöglicht wird. Die SPH-Methode passt sich gut an komplexe Oberflächengeometrien von bewegten Fluiden an und kann deren Materialeigenschaften problemlos ermitteln, da sich die Stützstellen zusammen mit dem Fluid bewegen und seine Oberfläche dadurch definieren. Außerdem verursachen große Deformationen der Fluidoberflächen bei dieser Methode weniger Ungenauigkeiten in der numerischen Lösung des Differentialgleichungssystems, da die Partikel miteinander nicht verbunden sind (siehe hierzu [LL03, Seite 29]). Ein zusätzlicher Vorteil der SPH-Methode besteht darin, dass keine Rechenzeit zur Erstellung und Aktualisierung eines Gitters benötigt wird. Dies ermöglicht die Interaktivität bei Computersimulationen von Fluidbewegung ist somit durch die SPH-Methode.

Im Wintersemester 2009/2010 wurde am Lehrstuhl 7 der TU Dortmund die Projektgruppe „Smoothed-Particle-Lab“ ins Leben gerufen, die sich das Ziel setzte, ein Softwaretools zur Simulation von Flüssigkeitsbewegung mittels SPH zu entwickeln. Die Projektgruppendauer war auf zwei Semester ausgelegt. Es fanden wöchentlich zwei Diskussionssitzungen sowie eine ausgedehnte Implementierungssitzung statt. Die Projektgruppe hat mit einer Seminarphase begonnen, in der die algorithmischen, softwaretechnischen und physikalische Grundbegriffe und Zusammenhänge einer Simulation der Flüssigkeitsbewegung mit der SPH-Methode erklärt wurden. Hierbei wurde der Projektgruppe aufgabenspezifisches Wissen vermittelt, welches den Projektgruppenteilnehmern eine Übersicht über die Komplexität der zu lösenden Aufgabe gab und das Projektgruppenziel konkreter zu formulieren erlaubte. Die zu simulierende Flüssigkeit sollte aus einer Quelle durch einen Container fließen und in einer Senke enden. Dabei sollte die Flüssigkeitsbewegung durch bestimmte externe und interne physikalische Kräfte sowie durch Kollisionen mit dem Container geprägt werden. Als externe Kräfte sollten die Gravitationskraft und die Oberflächenspannung, als interne Kräfte der Druck, die Viskosität und die Geschwindigkeit berücksichtigt werden. Der Ablauf der Simulation sollte zusätzlich durch globale Systemeigenschaften, wie das Zeitintegrationsverfahren beeinflusst werden können.

Dieser Endbericht dokumentiert die Arbeit dieser Projektgruppe hinsichtlich der Aufgabenstellung. Er gliedert sich in 14 Kapitel, die zu großen Teilen den Aufbaumodulen des Softwaresystems entsprechen. Kapitel 3 erläutert die SPH-Methode, die als Grundlage dieses Softwaretools dient und zeigt den aktuellen Stand der Technik für die Simulation von Fluidbewegungen auf Basis dieser Methode auf. Im 4. Kapitel wird der Aufbau der Szene im Detail beschrieben. Das 5. Kapitel beschäftigt sich mit den Verfahren der sog. Nachbarschaftssuche, die zum Auffinden der Nachbarpartikel eingesetzt werden. Die physikalischen Feldwerte der Nachbarpartikel werden für die Ermittlung der Feldwerte des jeweils untersuchten Partikels in der SPH-Summierung herangezogen. Die physikalische Kräfteberechnung ist Schwerpunkt des Kapitels 6. In Kapitel 7 werden die Zeitintegrationsverfahren behandelt. Diese sind für das zeitdynamische Verhalten des Gesamtsystems verantwortlich. Die Kollisionserkennung, welche die Kollisionen der Partikel mit der umschließenden Wand ermittelt, wird in Kapitel 8 beschrieben. Die objektorientierte Realisierung ist Bestandteil des 9. Kapitels. Kapitel 10 befasst sich anschließend mit der Metavisualisierung, welche die Eigenschaften der Partikel

während des Simulationslaufs visualisiert. In Kapitel 11 wird die Anwendungsvisualisierung erläutert. Diese stellt eine realitätsnahe Visualisierung des Simulationslaufs dar, welche in Form von Snapshots erfolgt. Die Bedienung der im Rahmen dieser Arbeit entwickelten Simulationssoftware wird in Kapitel B anhand eines Benutzerhandbuches detailliert beschrieben. In Kapitel 12 wird das Simulationssystem anhand von Testszenarien verifiziert und die Ergebnisse mit geprüften Vergleichswerten evaluiert. Abschließend werden in Kapitel 13 die wesentliche Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf noch offene Fragen und mögliche Erweiterungen gegeben.

# 3. Smoothed-Particle-Hydrodynamics

In diesem Kapitel wird Smoothed-Particle-Hydrodynamics (SPH) als echtzeitfähiges, numerisches Verfahren zur Fluidsimulation genauer vorgestellt. Neben der Idee und der mathematischen Herleitung des Verfahrens werden in zwei gesonderten Unterkapiteln die beiden wichtigsten Komponenten genauer beleuchtet. Es folgt ein umfassendes Kapitel zum aktuellen Stand der Technik mit einem historischen Einstieg, Anwendungsgebieten und der Betrachtung zweier konkreter Beispielanwendungen. Abschließend wird in Bezug auf das Projekt auf den Ablauf eines SPH-basierten Simulationssystem eingegangen und die notwendigen Komponenten eingeführt.

## 3.1. SPH-Interpolation

SPH ist ein numerisches, gitterloses Lagrange-Verfahren zum Lösen hydrodynamischer Gleichungen und wurde 1977 erstmal von Gingold und Monaghan [GM77] vorgestellt. Ursprünglich entwickelt im Bereich der Astrophysik zur dreidimensionalen Darstellung kompressibler Gasströmungen, hat dieses Verfahren mittlerweile in vielen Anwendungsgebieten Einzug erhalten (siehe Kapitel 3.2).

Die Grundidee von SPH besteht in der Aufteilung des betrachteten Fluids in diskrete Massenpunkte, den sogenannten „Particles“, die die beweglichen Abtaststellen des Verfahrens bilden. Neben der Masse und dem Volumen erhalten diese Partikel alle weiteren physikalischen Eigenschaften des Fluids, wie beispielsweise Druck oder Geschwindigkeit. Diese Eigenschaften werden zwischen den einzelnen Partikeln durch eine sogenannte Kernfunktion geglättet („smoothed“), wobei die räumliche Ausdehnung dieser Funktion die Anzahl der Partikel bestimmt, die in die Berechnung miteinbezogen werden. Die räumliche Mittelung der Feldgrößen und eine anschließende Diskretisierung transformiert das System partieller DGL in ein System gewöhnlicher DGL. Die Berechnung des zu simulierenden Fluids vereinfacht sich dadurch erheblich.

Die mathematische Herleitung des Verfahrens erfolgt über eine Integralinterpolation. Zur Bestimmung des Integralinterpolanten einer beliebigen Feldgröße $A(\mathbf{r})$ an einem bestimmten Ort $\mathbf{r}$ wird von einer Identität ausgegangen. Die *Diracsche Deltadistribution* $\delta(\mathbf{r})$ wird hierbei durch eine Kernfunktion $W$ angenähert:

$$\begin{aligned} A_I(\mathbf{r}) &= (A * \delta)(\mathbf{r}) \\ &= \int_V A(\mathbf{r})\, \delta(\mathbf{r} - \mathbf{r}')\, d\mathbf{r}' \\ &\approx \int_V A(\mathbf{r})\, W(\mathbf{r} - \mathbf{r}', h)\, d\mathbf{r}'. \end{aligned} \tag{3.1}$$

Da es sich um ein numerisches Verfahren handelt und die Anzahl an Abtaststellen endlich

ist, wird Gleichung 3.1 durch eine Summe approximiert:

$$A_S(\mathbf{r}) = \sum_j A_j V_j W(\mathbf{r} - \mathbf{r}_j, h). \tag{3.2}$$

$V_j$ ist das zum Partikel $j$ gehörige Volumen, $r_j$ die Position und $A_j$ eine beliebige Feldgröße.

Die Glättungslänge $h$ bestimmt die räumliche Ausdehnung der Kernfunktion und die damit verbundene Einflussnahme der Partikel untereinander. Eine genauere Beschreibung der Glättungslänge und möglicher Kernfunktionen erfolgt in Kapitel 3.1.1 und Kapitel 3.1.2.

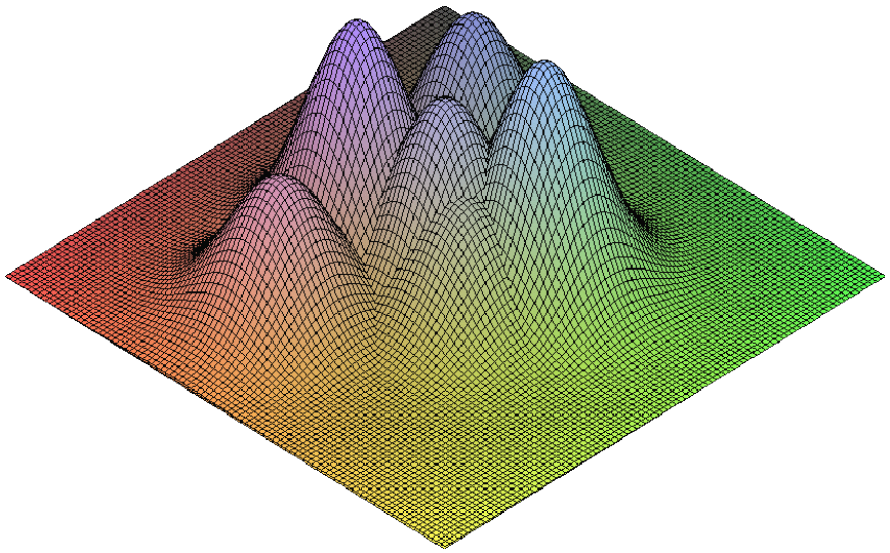


Abbildung 3.1.: Faltung einer räumlichen Größe $A(\mathbf{r})$ mit einer beispielhaften Kernelfunktion an vorgegebenen Abtaststellen

Wird das Volumen durch $V_j = \frac{m_j}{\varrho_j}$ ersetzt, wobei $m_j$ die Masse und $\varrho_j$ die Dichte des Partikels $j$ darstellen, ergibt sich die SPH-Basisgleichung:

$$A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\varrho_j} W(\mathbf{r} - \mathbf{r}_j, h). \tag{3.3}$$

Ausgehend von dieser Gleichung lassen sich sämtliche Feldgrößen an jedem beliebigen Punkt im betrachteten Volumen approximieren. Da einige hydrodynamischen Gleichungen ebenfalls die Ableitungen dieser Feldgrößen erfordern, werden im Folgenden der Gradient $\nabla$ und der Laplace-Operator $\Delta$ der SPH-Basisgleichung bestimmt.

Bei Betrachtung zweier Partikel $i$ und $j$, berechnet sich die partielle Ableitung der $x$-Komponente aus

$$\frac{\partial}{\partial x} A_S(\mathbf{r}_i)) = \frac{\partial}{\partial x} \left( A_j \frac{m_j}{\varrho_j} W(\mathbf{r}_i - \mathbf{r}_j, h) \right). \tag{3.4}$$

Die Anwendung der Produktregel liefert

$$\begin{aligned}\frac{\partial}{\partial x}\left(A_j\frac{m_j}{\varrho_j}W\left(\mathbf{r}_i-\mathbf{r}_j,h\right)\right) &= \frac{\partial}{\partial x}\left(A_j\frac{m_j}{\varrho_j}\right)W\left(\mathbf{r}_i-\mathbf{r}_j,h\right)+A_j\frac{m_j}{\varrho_j}\frac{\partial}{\partial x}W\left(\mathbf{r}_i-\mathbf{r}_j,h\right)\\ &= 0\cdot W\left(\mathbf{r}_i-\mathbf{r}_j,h\right)+A_j\frac{m_j}{\varrho_j}\frac{\partial}{\partial x}W\left(\mathbf{r}_i-\mathbf{r}_j,h\right)\\ &= A_j\frac{m_j}{\varrho_j}\frac{\partial}{\partial x}W\left(\mathbf{r}_i-\mathbf{r}_j,h\right), \end{aligned} \tag{3.5}$$

wobei die Tatsache ausgenutzt wurde, dass der Term $A_j\frac{m_j}{\varrho_j}$ weder von der $x$-Komponente noch von den anderen räumlichen Koordinaten abhängt und somit als konstant angesehen werden kann. Demnach hat die partielle Ableitung nur Einfluss auf die verwendete Kernfunktion und infolgedessen gilt dasselbe auch für den Gradienten

$$\nabla A_S(\mathbf{r}) = \sum_j A_j\frac{m_j}{\varrho_j}\nabla W\left(\mathbf{r}-\mathbf{r}_j,h\right). \tag{3.6}$$

Bei der Bildung der zweiten Ableitung

$$\nabla^2 A_S\left(\mathbf{r}\right) = \Delta A_S\left(\mathbf{r}\right) = \sum_j A_j\frac{m_j}{\varrho_j}\Delta W\left(\mathbf{r}-\mathbf{r}_j,h\right) \tag{3.7}$$

bleibt diese Eigenschaft ebenfalls erhalten. Voraussetzung ist hierbei allerdings die zweifache Differenzierbarkeit der verwendeten Kernfunktion.

### 3.1.1. Die Glättungslänge

Ein zentraler Parameter der SPH-Methode ist die Glättungslänge $h$, die sogenannte „Smoothing Length“. Die Glättungslänge $h$ ist der Radius, in dem andere Partikel Einfluß auf das betrachtete Partikel haben. Sie hat daher starken Einfluß auf die Genauigkeit und den Rechenaufwand der Simulationen, da bei entsprechender Wahl der Kernfunktion, die im Kapitel 3.1.2 erklärt wird, die Anzahl der bei der Berechnung mit einzubeziehenden Nachbarn festlegt wird. An der Abbildung 3.2 lässt sich der Einflußbereich der Glättungslänge $h$ veranschaulichen. Das dunkel eingefärbte Partikel ist das aktuell betrachtete Partikel und daher Mittelpunkt des Radius. Bei gewählter Glättungslänge $h$ werden die eingefärbten zwölf Partikel in die Berechnung einbezogen.

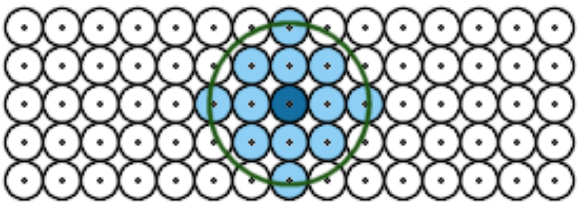


Abbildung 3.2.: Markierte Partikel fließen in die die Kraftberechnungen für das zentrale Partikel ein [Kel06]

Die Annahme, dass ein größerer Radius immer zu einem besseren Ergebnis führen wird, ist

nicht richtig. Bei einem für ein spezifisches Modell zu groß gewählten Radius, kann es sein, dass die Simulation ungenauer wird, da große Teile des Randbereichs ohne Partikel in die Berechnung mit einbezogen werden (siehe Abbildung 3.3). Das Ergebnis wird ungenauer, da Asymmetrien entstehen. Bei der Wahl eines Radius, der das gesamte Modell abdeckt, wird der Einfluss jedes Partikels auf alle anderen berechnet. Die würde dem Rechenaufwand der gitterbasierter Methoden enstsprechen.

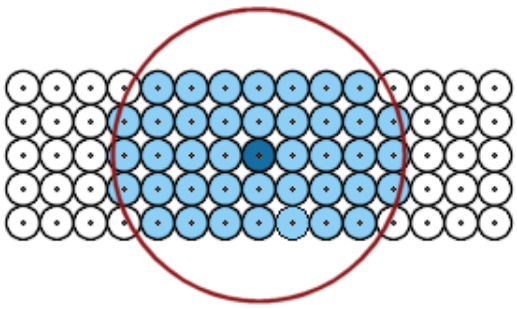


Abbildung 3.3.: Zu groß gewählter Glättungsradius. Randbereiche fließen in die Berechnung ein. $h$ [Kel06]

Ein sehr kleiner Glättungsradius (siehe Abbildung 3.4) führt zu einer Beschleunigung der Simulation, aber zu ungenauen Ergebnissen, da zu wenige Partikel in die Berechnung einbezogen werden.

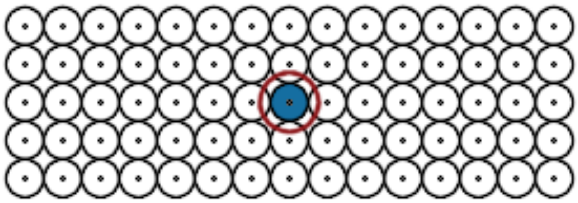


Abbildung 3.4.: Zu klein gewählter Glättungsradius $h$ [Kel06]

Die Glättungslänge kann global für alle Partikel festgelegt werden. Für diesen Fall wird die Glättungslänge $h$ so gewählt, dass bei der gegebenen durchschnittlichen Dichte der simulierten Substanz etwa 15 bis 50 Teilchen im Einflußgebiet des Partikels liegen [Kel06]. Der Radius $h$ hängt also direkt von der Dichte des Mediums ab, wenn alle Partikel dieselbe Masse haben.

Ein weiterer Ansatz ist „variable Smothing Length“, bei der für jedes Partikel, abhängig von seiner Umgebungsdichte, ein eigener Radius h berechnet wird. So kann auch bei kompressiblen Medien garantiert werden, dass immer ausreichend viele Nachbarn zur Karftberechnung vorhanden sind, aber nie so viele, dass die Berechnung verlangsamt werden würde. Dies wirft allerdings neue Probleme auf, da das Impulserhaltungsgesetz verletzt wird und Korrekturterme notwendig werden.

### 3.1.2. Die Kernfunktion

Es gibt drei Bedingungen, die eine gültige Kernfunktion $W$ erfüllen muss [Kel06]. $W$ muss auf 1 normiert sein (siehe Gleichung 3.8). Das heißt, dass der Flächeninhalt unter der Kurve gleich 1 sein muss.

$$\int_V W(|\mathbf{r} - \mathbf{r}'|, h) dV' = 1 \tag{3.8}$$

Weiterhin muss die Kernfunktion das $\delta$-Funktional approximieren (siehe Gleichung 3.9).

$$lim_{h \to 0} W(\mathbf{r} - \mathbf{r}', h) = \delta(\mathbf{r} - \mathbf{r}') \tag{3.9}$$

Die Kernfunktion muss mindestens einmal differenzierbar sein. Je nach Anwendungsfall wird die erste oder auch die zweite Ableitung benötigt. Einige hydrodynamische Feldgleichungen erfordern die erste Ableitung (Gradient) der betrachteten Felder, und die zweite partielle Ableitung des Ortes in Form des Laplace-Operators. Zur Verbesserung der Berechenbarkeit wird der Kern durch den Glättungsradius $h$ (siehe 3.1.1) beschränkt (siehe Gleichung 3.10).

$$W(\mathbf{r}, h) = \begin{cases} W'(\mathbf{r}, h) & 0 \leq ||\mathbf{r}|| \leq h \\ 0 & sonst \end{cases} \tag{3.10}$$

Das heißt, dass der Kern ab dem Glättungsradius $h$ null ist. Dadurch wird die Anzahl der Nachbarn bei der Berechnung beschränkt und deren Gewichtung festgelegt. Erst durch diese Beschränkung auf einen endlichen Wirkungsradius ergibt sich ein Geschwindigkeitsvorteil durch die SPH-Methode.

#### 3.1.2.1. Gauß-Glockenkurve

Die naheliegendeste Kernfunktion für die meisten Probleme ist die Gauß-Glockenkurve.

$$W(\mathbf{r}, h) = \frac{1}{\pi^{\frac{3}{2}} h^3} \exp \begin{cases} c - r^2 \\ h^2 \end{cases} \tag{3.11}$$

Sie ist intuitiv und statistisch gesehen die beste Funktion für die meisten Anwendungsfälle.

Leider eignet sich die Gaußfunktion aber nicht als Kernfunktion, da sie die an eine Kernfunktion gestellten Bedingungen nicht erfüllt. Ein gültiger Kern muss beschränkt sein (siehe Gleichung 3.10). Wenn wir aber die Gaußfunktion beschränken, wird der Flächeninhalt unter der Gaußkurve nicht 1 sein. Die gestellte Bedingung

$$\int_V W\left(|\mathbf{r} - \mathbf{r}'|, h\right) dV' = 1 \tag{3.12}$$

wird offensichtlich verletzt.

#### 3.1.2.2. Spline-Funktionen

Die Vorteile der Gaußkurve motivieren die Konstruktion einer Standardkernfunktion ohne die Nachteile der Gaußfunktion aber mit deren positiven Eigenschaften.Es hat sich herausgestellt, dass die Spline-Funktionen diese Kriterien erfüllen. Spline-Funktionen n-ten Grades sind stückweise aus Polynomen maximal n-ten Grades zusammengesetzte Funktionen. Spline Funktionen

können linear (wenn es sich um eine stückweise lineare Funktion handelt, ein Polygonzug), quadratisch, kubisch usw. sein. Stellen, an denen die Polynomstücke zusammentreffen, werden besondere Bedingungen unterstellt. So muss zum Beispiel der Spline n-1 mal differenzierbar sein.

Ein Beispiel für eine Splinefunktion ist die häufig verwendete Standardkernfunktion „6th degree polynomial“.

$$W(\mathbf{r},h) = \frac{315}{64\pi h^9} \begin{cases} \left(h^2 - ||\mathbf{r}||^2\right) & 0 \leq ||\mathbf{r}|| \leq h \\ 0 & ||\mathbf{r}|| > h \end{cases} \tag{3.13}$$

Sie ist eine Spline-Funktion 6ten Grades und wird aufgrund der Ähnlichkeit zur Gauß-Glockenkurve (siehe Abbildung 3.5) gerne verwendet.

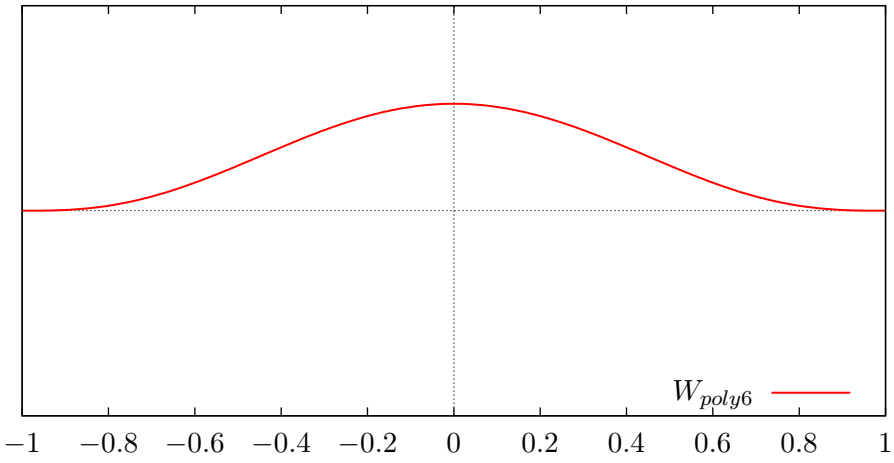



Abbildung 3.5.: „6th degree polynomial“

Die erste Ableitung der „6th degree polynomial“ Spline-Funktion, der Gradient,

$$\nabla W(\mathbf{r},h) = \frac{-945}{32\pi h^9} r \left(h^2 - ||\mathbf{r}||^2\right)^2 \tag{3.14}$$

und die entsprechende Funktionskurve (siehe Abbildung 3.6). Der Gradient ist bei dieser Funktion asymmetrisch und kann deshalb in dieser Form nicht angewendet werden. Für die SPH Anwendung wird der Gradient an der Y-Achse gespiegelt.

Die zweite Ableitung der „6th degree polynomial“ Spline Funktion, der Laplace Operator,

$$\nabla^2 W(\mathbf{r},h) = \frac{945}{32\pi h^9} \left(h^2 - ||\mathbf{r}||^2\right) \left(3h^3 - 7\,||\mathbf{r}||^2\right) \tag{3.15}$$

und die entsprechende Funktionskurve (siehe Abbildung 3.7). Da der Laplace Operator zum Zentrum hin negativ wird, eignet er sich zur Simulation einiger physikalischer Kräfte nicht.

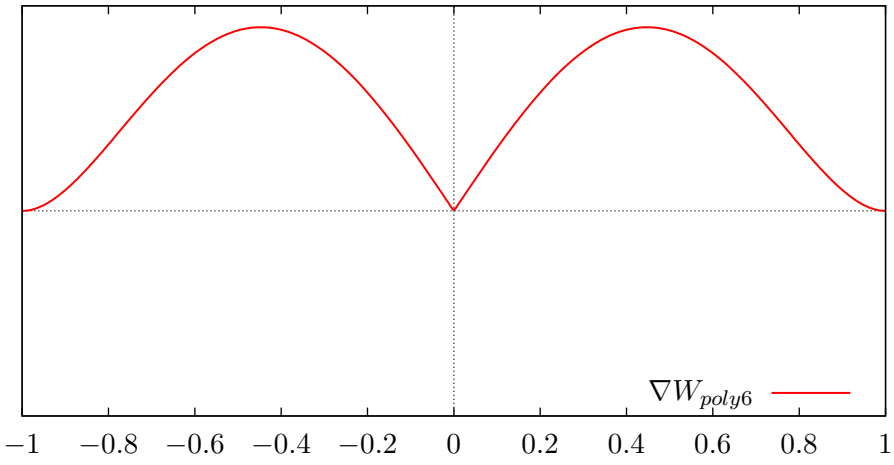



Abbildung 3.6.: Der Gradient der „6th degree polynomial“ Spline-Funktion

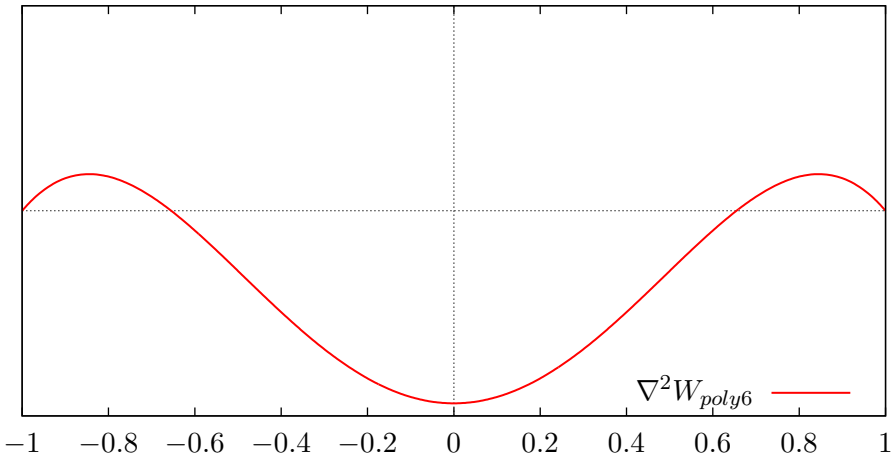



Abbildung 3.7.: Der Laplace Operator der „6th degree polynomial“ Spline-Funktion

## 3.2. Stand der Technik

Im folgenden Kapitel wird der Stand der Technik der SPH-Simulation anhand der historischen Entwicklung, Erweiterungen und Anwendungen der SPH-Methode und an zwei Beispielprogrammen erläutert.

### 3.2.1. Historische Entwicklung der SPH-Methode

SPH wurde ursprünglich entwickelt, um astrophysikalische Probleme im dreidimensionalen, unbegrenzten Raum zu lösen[Luc77], [GM77], da die Bewegung dieser Teilchen vergleichbar ist mit der Bewegung einer Flüssigkeit oder eines Gases. Die Motivation für die Entwicklung

einer neuen Methode rührte daher, das traditionelle gitterbasierte numerische Verfahren wie die Finite-Differenzen-Methoden (FDM) und die Finite-Elemente-Methoden (FEM) besonders Schwierigkeiten bei der Berechnung von komplexen Phänomene in der Astrophysik hatten. Um diese Art von Problemen zu lösen, schien die SPH-Methode eine gute Wahl zu sein. Zahlreiche Veröffentlichung wurden zu diesem Thema gemacht, einschließlich der wichtigen Arbeiten von [Ben89] und [Mon92].

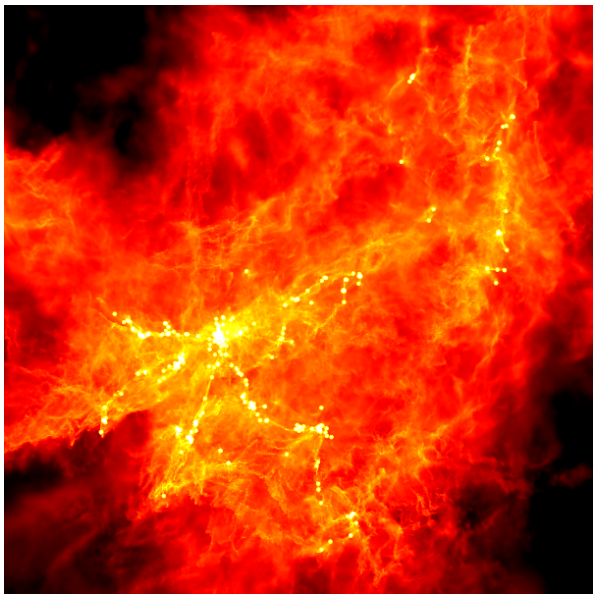


Abbildung 3.8.: SPH-Simulation einer Galaxie mit 20 Millionen Partikeln

Im Gegensatz zu der bekannten Particle-In-Cell (PIC)-Methode, die in den frühen 60er Jahren entwickelt wurde [Har63], braucht die SPH-Methode kein Netz, um die räumlichen Ableitungen zu berechnen. Sie hat den großen Vorteil, dass alle für die Berechnungen erforderlichen Informationen mit den Partikeln verknüpft sind.

### 3.2.2. Erweiterungen der SPH Methode

Heute wird die SPH-Methode in vielen Bereichen, wie der Simulationen von binären Sternen und der Kollision von Sternsystemen [Ben88], [Mon92], Supernovas [HP99], Entstehung und Zusammenbruch von Galaxien [ML91], Vereinigung von schwarzen Löchern mit Neutronensternen [Lee00], Detonation von weißen Zwergen [GSDE99] und der Evolution des Universums [Mon90] verwendet. Die SPH-Methode wurde für ein breites Spektrum von Problemen bei der Berechnung von Fluiden oder der Festkörpermechanik verwendet, aufgrund der Eigenschaft komplizierte physikalische Effekte in SPH-Problemen formulieren zu können. In gewisser Weise kann der Begriff der Hydrodynamik als Mechanik im Allgemeinen interpretiert werden. So wird in einigen Quellen [PHK95], der Begriff „Smoothed Particle Mechanics“ verwendet.

### 3.2.3. Anwendungen der SPH-Methode

Die ersten Anwendungen der SPH-Methode betrafen hauptsächlich die Fluiddynamik verwandten Anwendungsgebiete (Siehe Abbildung 3.9). Dazu gehören elastische Fluide, Magnet-Hydrodynamik [Mor96], mehrphasige Fluide [MK95], quasi inkompressible Fluide [Mon94],

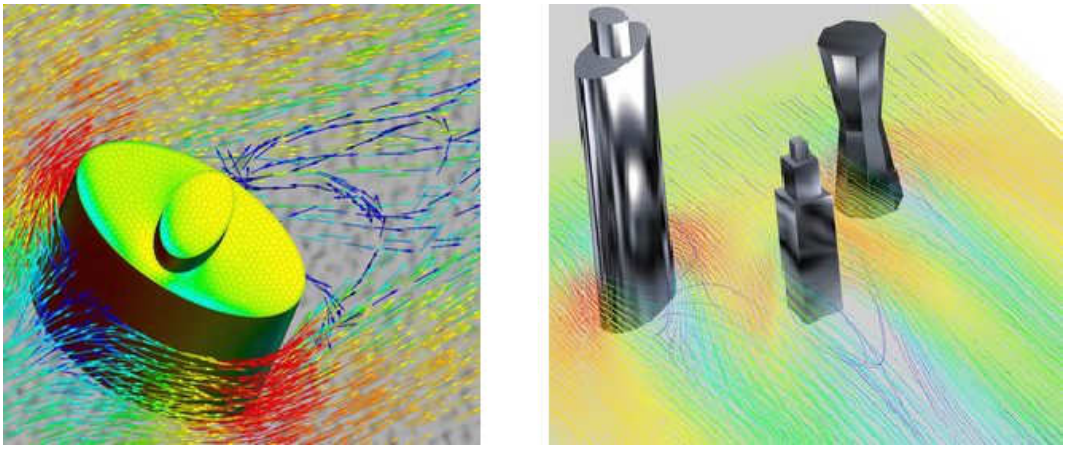


Abbildung 3.9.: SPH-Strömungssimulation

Schwerkraftphänomene [MK95], Fluß durch poröse Medien, Wärmeleitung [MK95], Schock-Simulationen [Mon89], Wärmeübertragung und Massenstrom, Eis und granulare Materie [GS98]. Benz [BA95] erweiterte die SPH-Methode für die Simulation für Brechen von spröden Feststoffen. Bonet und Kulasegaram[BK00] verwendeten SPH-Methoden für die Simulation der Blechumformung.

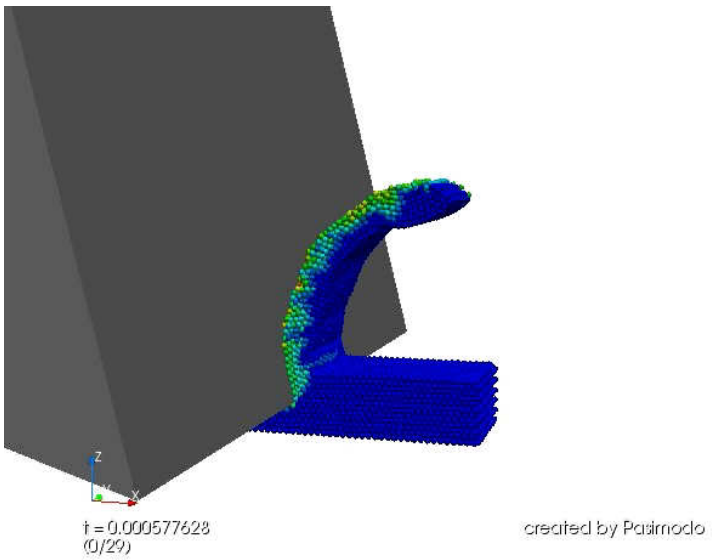



Abbildung 3.10.: SPH-Simulation einer Metallzerspanung [Pas]

Die SPH-Methoden wurde somit auch sehr interessant für die Simulation starker Deformationen (siehe Abbildung3.10). Ein weiterer wesentlicher Anwendungsbereich sind „high (oder

hyper) velocity impact (HVI)“-Probleme. Dies ist die Simulation der Kollision von Projektilen auf Weltraumfahrzeuge (Satelliten, Raumstationen, Raumfähren). Bei HVI-Problemen bewegen sich Schockwellen durch den getroffenen Körper, die sich wie Fluide verhalten. Von Libersky [LP91] und weiteren wurden Beiträge für die Anwendung von SPH-Methoden bei HVI-Problemen veröffentlicht. Eine weitere wichtige Anwendung der SPH-Methoden ist die Simulation von Explosionen, genannt „High Explosive“ (HE). Swegle und Attaway [SHA95] untersuchten die Machbarkeit Unterwasserexplosionen zu mit SPH-Methoden zu berechnen. Liu und sein Team haben die Simulation von Unterwasserexplosionen durchgeführt [LLZL00].

Die Anwendung von SPH-Methoden auf ein breites Spektrum von Problemen hat zu bedeutenden Erweiterungen und Verbesserungen der ursprünglichen SPH-Methode geführt. Die numerischen Methoden wurden schrittweise verbessert, einige Nachteile der SPH-Methode wurden identifiziert und modifizierte Techniken oder korrigierende Methoden wurden ebenfalls vorgeschlagen. Swegle identifizierte das Zug-Instabilitäts Problem [SHA95], das bei Materialien mit Festigkeit von Bedeutung ist. In den vergangenen Jahren wurden verschiedene Änderungen und Korrekturen eingeführt, die die Konsistenz und die Genauigkeit der SPH-Methode nach und nach verbesserten. Diese Änderungen führten zu vielen verschiedenen Versionen der SPH-Methode .

### 3.2.4. Beispiele

Zu Beginn der Projektgruppe wurden einige am Markt befindliche Programme, mit denen SPH-Simulationen durchgeführt werden konnten, näher betrachtet. Es sollten Anregungen für die Projektgruppe gesammelt werden und überprüft werden, ob eventuell die Weiterentwicklung oder Ergänzung eines bestehenden Programms sinnvoll sein könnte. Nachstehend werden zwei Programme kurz vorgestellt, die in den PG-Sitzungen besprochen wurden.

#### 3.2.4.1. Fluids

„Fluids“ (siehe [Flu]) ist ein kleines Open-Source-Programm eines wissenschaftlichen Mitarbeiters der Universität „California Santa Barbara“. Das Programm soll zur Einführung in die Programmierung von SPH-Simulationen dienen. Die Schwerpunkte wurden daher auf eine leichte Lesbarkeit des Quellcodes, Übersichtlichkeit in der Programmstruktur und einfache Anwendbarkeit gelegt. Alle Parameter sind mit, für die Simulation von Wasser, sinnvollen Werten vorbelegt. Eine große Anzahl von getesteten Szenarien ist vorhanden (Siehe Abbildung3.11). Es ist in C programmiert und unterstützt wahlweise Beschleunigungsfunktionen der Grafikkarte. Es ist keine GUI vorhanden, die Steuerrung erfolgt über Tastaturbefehle. Alle Szenarien und Parameter sind fest programmiert. Das Programm diente in manchen Punkten als Anregung für unser eigenes Projekt.

#### 3.2.4.2. PASIMODO

Der Name „PASIMODO“ steht für „PArticle SImulation and MOlecular Dynamics in an Object oriented fashion“. PASIMODO ist ein Programmpaket (siehe [Pas]) zur partikelbasierten Simulation (siehe Abbildung 3.12).

Hauptanwendungsgebiet ist die Simulation granularer Medien wie Sand, Schotter, Granulate der chemischen Industrie, etc. Darüber hinaus kann es zur Simulation beliebiger anderer Lagrange-basierter Methoden wie z.B. Fluid-Simulation mit Smoothed-Particle-Hydrodynamics

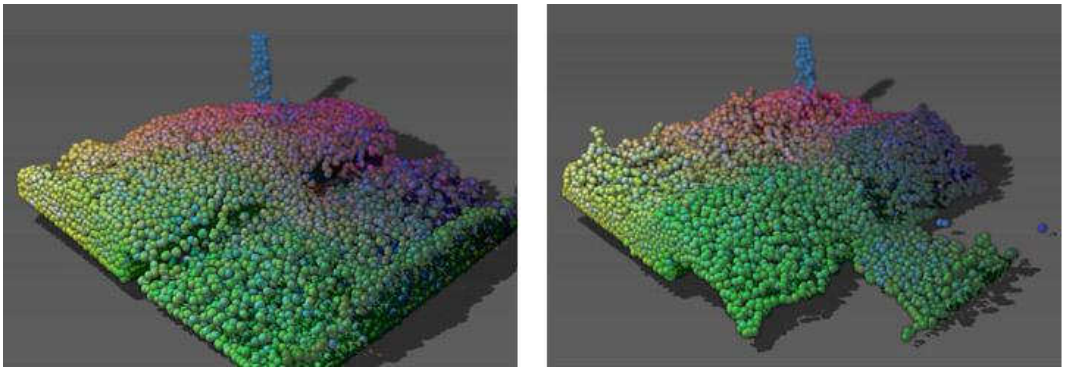


Abbildung 3.11.: SPH-Simulator „Fluids“ in der Version 2

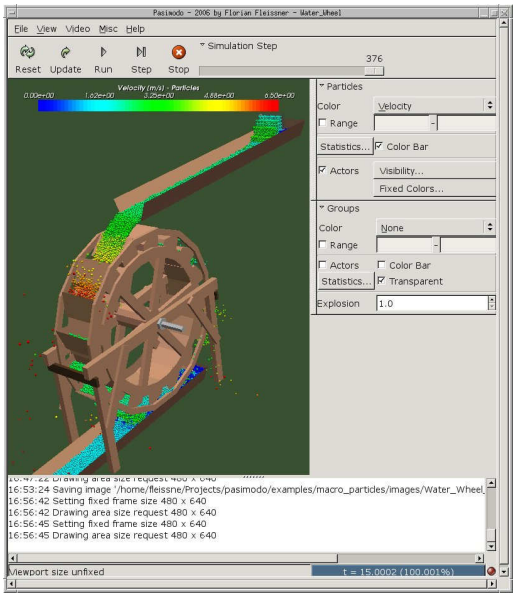



Abbildung 3.12.: SPH-Simulator „Pasimodo“

verwendet werden (vgl.:[Pas]). Einige, von den Entwicklern hervorgehobenen, Eigenschaften des Programmspakets sind

- XML-basierte Ein- und Ausgabedatenschnittstelle
- Parallele Simulation mit MPI
- Verschiedene Partikel-/Objekttypen

- Verschiedene Interaktionen/Potentiale wie Normalkräfte, Coulombreibung und Gleitreibung
- Effiziente cache-optimierte Nachbarschaftssuche
- Hinzufügen und Entfernen von Objekten (Partikeln) während der Simulation
- Plug-in-Schnittstelle zur Erweiterung des Simulators um beliebige Objekte (Partikel), Integratoren und Interaktionen (Potentiale)
- Graphische Benutzeroberfläche

Mit dem Programmpaket PASIMODO stellt somit, laut den Verfassern, ein mächtiges Framework für die Fluid-Simulation zur Verfügung. Leider ist das Programmpaket PASIMODO kein Open-Source-Projekt und auch das kompilierte Programm ist nicht frei erhältlich. Der Quellcode von PASIMODO wird nur im Rahmen von Kooperationen und Forschungsprojekten freigegeben. Ohne den Zugriff auf den Quellcode war leider nicht zu beurteilen, ob PASIMODO sich für die Zwecke der Projektgruppe eignen würde. Die Projektgruppenbetreuer wurden mit der Kontaktaufnahme zum zuständigen Lehrstuhl an der Universität Stuttgart beauftragt. Da aber nicht abzusehen war, ob und wann die Projektgruppe Zugang zum Quellcode von PASIMODO erhalten würde, wurde gegen die Verwendung von PASIMODO entschieden.

## 3.3. SPH-Simulationsablauf

Im folgenden Abschnitt wird der typische Ablauf einer Smoothed-Particle-Hydrodynamics-Simulation kurz vorgestellt (siehe Abbildung 3.13). Alle Simulationsschritte werden in späteren Kapiteln nochmals ausführlich behandelt.

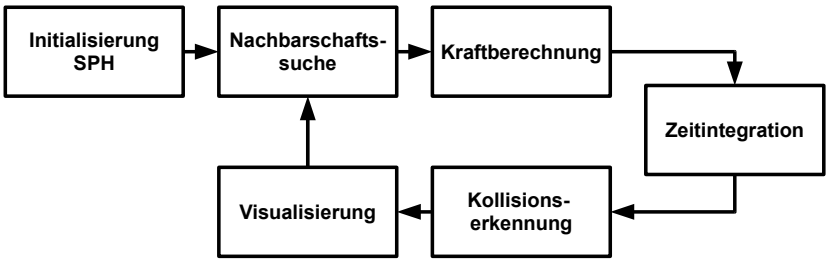



Abbildung 3.13.: SPH Simulationsablauf

### 3.3.1. SPH-Initialisierung

Bei der Initialisierung werden die Startwerte der SPH-Simulation zugewiesen. Dies können materialspezifische Parameter sein, sowie Initialwerte für die Parameter der Partikelgrößen, wie z.B. Position und Geschwindigkeit. Desweiteren wird eine Szene geladen und deren Objekte als Kollisionsobjekte, Quellen, Senken oder reine visuelles Objekt definiert (siehe Kapitel 4). Die Bestimmung des Zeitschritts $\Delta t$ und der Glättungslänge $h$ sind ebenfalls Bestandteil der Initialisierung.

Nachfolgenden Schritte werden typischerweise bei der Initialisierung ausgeführt.

- Definition des Fluidmaterials durch Parameterwahl
- Festlegung von Partikelanzahl, -position und -initialgeschwindigkeit
- Berechnung der Partikelmasse
- Initialisierung der Kernfunktionen und Bestimmung der Glättungslänge
- Initialisierung der Nachbarschaftssuche (Anlegen von Hashtabellen)
- Definiton der Kollisionsobjekte
- Festlegung der Stoßzahl der Kollsionsbehandlung (siehe Kapitel 8)
- Definiton der Quellen und Senken (siehe Kapitel 4.2)
- Definiton der Oberflächeneigenschaften (siehe Kapitel 6.7)
- Festlegung des Zeitschritts und Initialisierung der Integrationsmethode

### 3.3.2. Nachbarschaftssuche

Der erste Schritt jeder SPH-Simulationsschleife ist die Bestimmung der Nachbarschaftsbeziehungen der Partikel. Bei der Nachbarschaftssuche werden (siehe Kapitel 5) für jedes Partikel die Nachbarn, die innerhalb der Glättungslänge des jeweiligen Partikels liegen, bestimmt und in Nachbarschaftslisten gespeichert. Die gewählte Glättungslänge entspricht dem Suchradius der Nachbarschaftssuche. Die Nachbarschaftslisten bleiben bis zum Ende der Simulationsschleife unverändert und werden danach aktualisiert. Die Nachbarschaftsbeziehungen werden für die Berechnung sämtlicher Kräfte und der Dichte benötigt. Es stehen 9 verschieden Varianten der Nachbarschftssuche zur Verfügung. Sie unterscheiden sich in der Geschwindigkeit, Adaptivität und geeigneten Anwendungsfällen.

- Quadratische Suche (siehe Kapitel 5.3.1)
- Reguläre Gitter mit fester Speicherzuodnung (siehe Kapitel 5.3.2)
- Spatial-Hashing (siehe Kapitel 5.3.3)
- Capped-Spatial-Hashing (siehe Kapitel 5.3.4)
- Spatial-Hashing-Vertex Verfahren (siehe Kapitel 5.3.5)
- Spatial-Hashing-Automatic-Smoothing-Length Verfahren (siehe Kapitel 5.3.6)
- Optimized-Spatial-Hashing (siehe Kapitel 5.3.7)
- Union-Spatial-Hashing (siehe Kapitel 5.3.8)
- Symmetry-Aware-Grid (siehe Kapitel 5.3.9)

### 3.3.3. Kraftberechnung

Bei der Kraftberechnung (siehe Kapitel 6) werden alle auf ein Partikel wirkenden Kräfte berechnet und summiert. Dies ist erforderlich, um die neue Beschleunigung eines Partikels zu bestimmen. Zu Beginn der Kraftberechnung werden für jedes Partikel die Parameter Dichte und Druck bestimmt. Kräfte, die bei der Simulation berücksichtigt werden sollen, werden anschließend unter Verwendung unterschiedlicher Kernelfunktionen berechnet. Mögliche zu berechnende Kräfte sind:

- Druckkraft (siehe Kapitel 6.4)
- Viskosität (siehe Kapitel 6.5)
- Gravitation (siehe Kapitel 6.6)
- Oberflächenspannung (siehe Kapitel 6.7)
- Zentralkraft

Für die Berechnung jeder Kraft stehen meist mehrere Alternative Formeln zur Wahl. Die Auswahl der passenden Formeln ist bestimmt durch den konkreten Anwedungsfall der Simulation.

### 3.3.4. Zeitintegration

Im nächsten Schritt der Zeitintegration (siehe Kapitel 7) wird aus der Beschleunigung und der Geschwindigkeit die neue Position des Partikels nach dem nächsten Zeitschritt berechnet. Die neue Geschwindigkeit eines Partikels ist das Resultat aus der Geschwindigkeit, die das Partikel aus des vorherigen Simulationsschleife mitbringt und der Beschleunigung, die aus den einwirkenden Kräften berechnet wurde. Die Zeitintegration erfolgt nicht analytisch durch Lösung der Differentialgleichung, sondern durch Approximation. Hierfür stehen in YASPHS zwei Verfahren zur Wahl, die sich in Geschwindigkeit und Genauigkeit unterscheiden.

- Euler-Verfahren (siehe Kapitel 7.2.1)
- Leap-Frog-Verfahren (siehe Kapitel 7.2.2)

Wahlweise ist es im Zeitintegrationsschritt möglich, das Ergebniss der Approximation durch das XSPH Verfahren zu glätten (siehe Kapitel 7.2.3).

### 3.3.5. Kollisionserkennung und Kollisionsbehandlung

Anschließend wird durch Kollisionserkennung (siehe Kapitel 8) für jedes Partikels überprüft, ob zwischen seiner alten und seiner neuen Position ein Kollisionsobjekt liegt. Dies kann die Wand eines primitven Körpers (siehe Kapitel 8.1) sein oder die Dreiecksfläche eines Meshes (siehe Kapitel 8.2). Falls eine Kollision festgestellt worden ist, muss die neu berechnete Position des Partikels korrigiert werden. Dies kann auf unterschiedlich aufwendige Arten geschehen, die in Kapitel 8 genauer beschrieben werden. Zusätzlich kann bei der Kollisionsbehandlung Reibung simuliert werden, indem dem Partikel bei der Reflexion Energie entzogen wird (siehe Kapitel 8.3).

### 3.3.6. Visualisierung

Der Schritt der Visualisierung ist optional und muss nicht bei jedem Simulationsdurchlauf durchgeführt werden, da er das Ergebnis der Simulation nicht beeinflusst. YASPHS kann die gesamte Szene mit Partikeln, Kollisionskörpern und Szeneobjekten visualisieren, oder auch nur Teile davon. Die angewendeten Methoden zur Visuallisierung der berechneten Partikel können sehr unterschiedliche Ziele haben. Möglich ist es die Partikel sehr einfach und nur zur Kontrolle des Simulationsfortschritts als Punkte darzustellen. Zur Analyse der wirkenden Kräfte bietet sich die Metavisualisierung an (siehe Kapitel 10). Ziel der Metavisualisierung ist die optische Hervorhebung von Information über den Zustand der Simulation. Dies können zum Beispiel die Partikelverteilung oder Partikelattribute wie Masse, Dichte und Geschwindigkeit sein.
Ziel der Anwendungsvisualisierung (siehe Kapitel 11) ist die realitätsnahe Darstellung des Simualtionsergebnisses. Die realistische Darstellung ist insbesondere nötig, um die Plausibilität der Simulation beurteilen zu können, da die simulierten Partikel in der Regel ein Fluid (Wasser, Öl, Gas) darstellen sollen. Die Partikel müssen durch unterschiedliche Methoden als Fluid interpretiert werden, um zu einer realitätsnahen Visualisierung zu gelangen.

# 4. Szene

Die Szene bildet die Datenbasis für alle anderen Komponenten des Programms, sie beinhaltet die Simulationsdomäne mit ihren Objekten, die Einstellungen für Integrationsverfahren, Kraftberechnungen u.Ä., sowie zusätzliche Metainformationen der Simulation. Eine Simulation wird daher vollständig und ausschließlich durch ihre Szene definiert. Eine physikalische Fragestellung muss mit Hilfe von Szeneobjekten in eine simulierbare Szene in YASPHS übersetzt werden, daher ist ein tieferes Verständnis der unterschiedlichen Objekttypen sowie deren Wechselwirkung untereinander für das korrekte Modellieren einer Simulation unerlässlich. Um eine wissenschaftliche Vergleichbarkeit und Reproduzierbarkeit zu gewährleisten, ist das Simulationsergebnis einer Szene in YASPHS immer deterministisch, d.h. zwei identische Szenen haben auch einen identischen Simulationsverlauf.

## 4.1. Bestandteile der Szene

Die Objekte in der Szene werden in einer Menge verwaltet, d.h. es gibt keine Sortierung oder Reihenfolge der Objekte, jedes Objekt ist aber durch eine eindeutige Id-Nummer zu identifizieren. Die Menge der Simulationsobjekte in der Szene lässt sich in zwei grundlegende Typen partitionieren. Die erste Menge, die Menge der Basisobjekte, können für sich alleine in der Szene existieren und werden für den grundlegenden Simulationsaufbau benötigt. Elemente der zweiten Objektmenge können nur im Verbund mit einem Basisobjekt in der Szene vorkommen. Diese, einer Vater-Kind ähnliche, Verbindung drückt eine Spezialisierung aus und beeinflusst das Verhalten beider betroffener Objekte. Durch diese gegenseitige Beeinflussung kann jedes Vaterobjekt nur ein Kind haben und jedes Kind nur einen Vater (siehe Abbildung 4.1 und Abbildung 4.2). Objekte können, bei angehaltener Simulation, aus der Szene entfernt oder hinzugefügt werden, die evtl. vorhandenen Vater-Kind-Beziehungen werden dabei automatisch aufgelöst. Das Entfernen eines Basisobjektes führt daher auch immer zur Löschung des verbundenen Kindobjektes. Um das Arbeiten mit Szeneobjekten intuitiver zu gestalten, kann jedem Objekt ein Name zugeordnet werden, der allerdings nicht eindeutig sein muss.

## 4.2. Quelle und Senke

Quelle und Senke werden verwendet um Partikel in der Simulation hinzuzufügen oder zu entfernen. Beide Objekttypen sind Kindobjekte deren geometrische Form von ihren Vaterobjekten bestimmt wird. Die Objekte selber verfügen über alle weiteren Informationen, die ihr Verhalten bestimmen.

Die Quelle erzeugt abhängig von einer einstellbaren Emissionsrate neue Partikel innerhalb der Geometrie des Vaterobjektes. Die neuen Partikel werden dabei mit vordefinierten Parametern wie z.B. Anfangsgeschwindigkeit etc., von der Quelle initialisiert und werden im bereits im aktuellen Zeitschritt verwendet. Um Simulationszustände mit zu nahen Nachbarschaften unter den Partikeln zu vermeiden, kann die Quelle durch einen optionalen Test überprüfen, ob

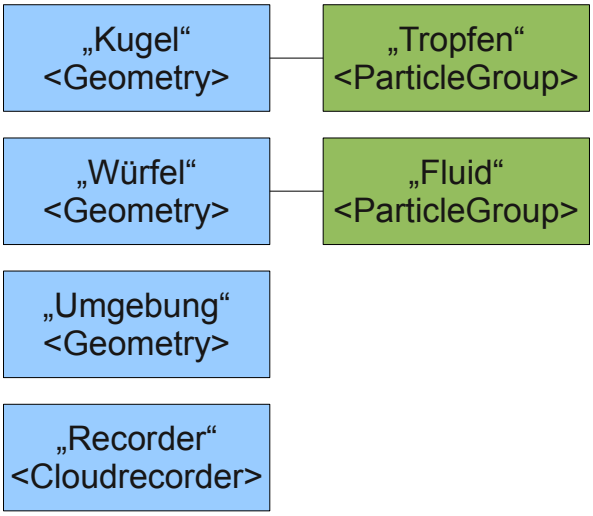



Abbildung 4.1.: Abstrakte Darstellung einer Szene mit 6 Objekten. 3 Geometrieobjekte und ein Cloudrecorder bilden die Menge der Basisobjekte (blau), wobei 2 Geometrieobjekte durch verbundene Kindobjekte (grün) als Partikelgruppen fungieren.

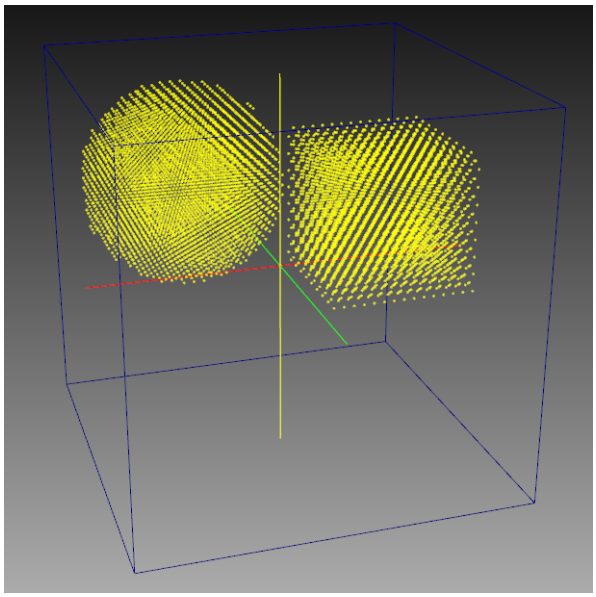


Abbildung 4.2.: Konkrete Darstellung der Szene in Abbildung 4.1. Der, in der abstrakten Darstellung sichtbare, Cloudrecorder wird in der 3D-Vorschau nicht angezeigt. Die zwei Partikelgruppen werden durch die umschließende Umgebungsgeometrie (blau) wie in einem Tank eingeschlossen.

innerhalb der neuen Partikelmenge alle Partikel einen Mindestabstand einhalten. Im Falle von kritischen Nachbarschaften wird versucht eine bessere Position für die betroffenen Partikel zu finden, falls dies nicht gelingt, werden weniger Partikel erzeugt als durch die Emissionsrate eigentlich vorgeschrieben wurde.

Die Senke fungiert als Gegenstück zur Quelle und entfernt alle Partikel aus der Simulation, die die Geometrie des Vaterobjektes berühren oder durchdringen. Senken verfügen, im Gegensatz zur Quelle, über keine weiteren Parameter.

## 4.3. Geometrie

Geometrieobjekte beschreiben räumliche Körper und dienen, sofern sie nicht mit einem Kindobjekt verbunden sind, als statische Kollisionsobjekte (siehe Kapitel 8) für die simulierten Partikel. Das genaue Kollisionsverhalten wird über das, dem Geometrieobjekt, zugewiesene Material definiert (siehe Kapitel 6.7). Verfügt das Objekt hingegen über ein Kind, so wird sein Verhalten über das Kind definiert. Häufig treten Geometrieobjekte mit Kind nur als Datenlieferant für das Kindobjekt auf und haben selbst keinen Einfluss mehr auf die Simulation.

Die Beschaffenheit des Körpers des Geometrieobjektes kann durch impliziert definierte Primitive (Würfel und Sphäre) oder ein Dreiecksmesh definiert werden. Ein Dreiecksmesh ist dabei als eine Menge von 3-Tupeln definiert bei dem jedes 3-Tupel drei Positionen im Raum enthält, die zusammen ein Dreieck ergeben. Dreiecksmeshes stellen die vielseitigste und flexibelste Struktur zur Modellierung von Körpern in YASPHS dar, erfordern dafür allerdings auch einen wesentlichen Mehraufwand bei der Simulation, spez. der Kollisionsbehandlung. Hinzukommt, dass aus einem Dreiecksmesh bestehende Geometrien auch nicht als Partikelquelle oder Partikelgruppe genutzt werden können.

## 4.4. Partikelgruppen

Über Partikelgruppen können Geometrieobjekte kontrolliert mit Partikeln gefüllt werden. Es werden solange Partikel innerhalb des Geometrieobjektes erzeugt, bis eine von der Partikelgruppe vorgegebene Dichte erreicht ist. Nach der Erzeugung der Partikel verhalten sich die Partikelgruppen passiv und haben keinen weiteren Einfluss auf die Simulation. Es stehen zur Verteilung der Partikel in den Geometrieobjekten verschiedene Verfahren zur Verfügung:

- Zufällig: Die Partikel werden zufällig im Volumen angeordnet.
- Kompakt: Es wird versucht so viele Partikel woe möglich in einer Rasteranordnung in das Volumen zu setzen.

Grundsätzlich ist die kompakte Füllmethode zu bevorzugen, da sie stabilere Startsituationen erzeugt und in den meisten Fällen die Silhouette des gefüllten Volumens besser nachzeichnet als die zufällige Verteilung.

## 4.5. Voxelrecorder

Der Voxelrecorder ist ein Basisobjekt, das kein Kindobjekt haben kann. Der Recorder dient der Datenerfassung während der Simulation und definiert in der Szene ein achsenorientiertes Gitter mit den im Voxelrecorder eingestellten Ausmaßen und Unterteilungsgrad. Die Elemente

dieses Gitters, auch Voxel genannt, können jeweils einen Skalarwert speichern und bilden zusammen mit den impliziten Positionsdaten des Gitterpunktes eine räumlich bezogene Datenbasis. Der Voxelrecoder führt in jedem Zeitschritt die SPH-Interpolation für die Dichte (siehe Kapitel 3.1) an jedem der Voxelgitterpunkte durch und speichert die berechneten Skalarwerte in den entsprechenden Voxelzellen. Das fertig berechnete Voxelgitter wird anschließend in einer separaten Datei gespeichert (siehe Kapitel 9.2.4 und Kapitel 11.1) und kann für spätere Datenerfassungen und Verarbeitungsschritte genutzt werden. Die Gitterauflösung und Ausdehnung des Voxelrecorders kann leider nur bei der Erzeugung des Recorders einmalig eingestellt werden und ist danach unveränderbar, da eine Änderung dieser Parameter im genutzten Maya-Datenformat nicht abgebildet werden kann.

## 4.6. Cloudrecorder

Der Cloudrecorder ist, ähnlich wie der Voxelrecorder, ein Basisobjekt ohne Kind und dient ebenfalls zur Datenerhebung während der Simulation. Der Cloudrecorder speichert für jeden Zeitschritt die Partikelpositionen als Liste von Vektoren in eine Ausgabedatei. Geeignete Programme (siehe Kapitel 9.2.4 und Kapitel 9.2.5) können diese Dateien dann für eine Anwendungsvisualisierung laden und verarbeiten (siehe Kapitel 11.2). Der Cloudrecorder erfasst, anders als der Voxelrecoder, immer die gesamte Simulation und ist nicht in seiner Auflösung beschränkt.

## 4.7. Oberflächeneigenschaften

Oberflächeneigenschaften werden in der Szene durch Materialien beschrieben und dienen hauptsächlich zur Parametrisierung der Kollisionserkennung (siehe Kapitel 8). Jedes Material hat einen eindeutigen Namen, kann mehreren Objekten zugeordnet sein und agiert als einfacher Behälter für Strings, Skalarwerte und Vektoren, die über einen Schlüsselnamen angesprochen werden. Welche Informationen in einem Material gespeichert werden, ist daher von der konkreten Simulation abhängig und vom Benutzer beliebig erweiterbar. Die einzelnen Simulationskomponenten suchen in den Materialien nach den vorher vereinbarten Schlüsselnamen für bestimmte Parameter bzw. verwendet voreingestellte Werte, falls ein Material diesen Parameter nicht definiert. Die für jede Komponente bereits vordefinierten Schlüsselnamen und Parameter werden in den jeweils entsprechenden Kapiteln beschrieben.

# 5. Nachbarschaftssuche

Die Nachbarschaftssuche der Partikel ist der erste Berechnungsschritt innerhalb der Simulationsschleife und dient als direkte Grundlage für die Berechnung der Partikelbstände, die in dem nächsten Schritt der Simulationsschleife (siehe Kapitel 6), der Kraftberechnung, verwendet werden . Hierfür werden in jedem Simulationszyklus zu allen Partikeln die Nachbarpartikel, welche innerhalb des Glättungsradius (engl. „smoothinglength“) liegen, bestimmt. Deshalb wird der Glättungsradius bei der Nachbarschaftssuche als Suchradius $r$ bezeichnet. Die auf diese Weise berechneten Nachbarpartikel werden jeweils in eine für jedes Partikel generierte Nachbarliste eingetragen. Diese Nachbarschaftsbeziehungen werden schließlich innerhalb eines Zeitschritts für die Berechnung der Partikelabstände verwendet und müssen anschließend jeweils für den darauf folgenden Zeitschritt aktualisiert werden.

## 5.1. Grundlagen

In der Nachbarschaftssuche gibt es nach [WMK07] im Wesentlichen zwei verschiedene Herangehensweisen. Bei der ersten Methode wird jedem Partikel eine feste Anzahl von zu suchenden Nachbarpartikeln vorgegeben. Dadurch ergibt sich, dass alle Partikel eine einheitliche maximale Anzahl an Nachbarn erhalten. Da der Fokus auf der Anzahl der Nachbarn liegt, rückt die räumliche Distanz zwischen den Partikeln für die Auswahl der Nachbarn zunächst in den Hintergrund. Die zweite Methode teilt den Partikeln einen festen Suchradius zu. Somit werden nur Partikel als Nachbarn klassifiziert, welche sich in dem vorgegebenen Suchradius befinden. Das Prinzip der einheitlichen Nachbaranzahl rückt bei dieser Methode in den Hintergrund und die räumliche Distanz bestimmt die Anzahl der Nachbarpartikel. Beide Methoden haben ihre Vor- und Nachteile, weshalb auch beide zum Einsatz kommen. Der wesentliche Vorteil bei der Nachbarschaftssuche mit fester Anzahl an Nachbarn ist, dass die benötigte Laufzeit unabhängig von der Dichte der Partikel und somit für jedes Partikel gleich groß ist. Der Nachteil ist, dass die feste Anzahl zu bestimmender Nachbarn räumlich gleichverteilt gefunden werden muss, um zu verhindern, dass die später zu berechnenden Kräfte im Wesentlichen nur aus einer Richtung auf das Partikel einwirken. Dies würde dazu führen, dass sich Partikel in der Simulation durch diese ungewollten Kräfte falsch bewegen. Der Nachteil ergibt sich schließlich aus dem Problem, dass einfache Strategien für die Berechnung der räumlichen Gleichverteilung der Partikel zu viele Partikel auf Eignung als Nachbarpartikel untersuchen müssen und somit die Laufzeit stark ansteigt. Der Vorteil eines festen Suchradius ist, dass alle gefundenen Nachbarpartikel innerhalb dieses Suchradius meistens automatisch, bis auf Randfälle, fast gleichverteilt vorkommen. Allerdings ist die Laufzeit hier von der Anzahl der Partikel innerhalb des Suchradius und somit von der Dichte der Partikel abhängig. Somit wirkt sich eine hohe Dichte der Partikel negativ auf die Laufzeit aus. Daher ist es nötig Verfahren zu verwenden, die beide Konzepte so verbinden, dass die Nachteile möglichst ausgeglichen und die Vorteile verstärkt werden. Hierfür gibt es im Wesentlichen zwei Möglichkeiten. Zum Einen kann nach neuen Algorithmen gesucht werden, die den Ansatz des konstanten Suchradius mit dem Ansatz der konstanten Nachbaranzahl kombinieren. Zum Anderen können effiziente

Suchdatenstrukturen verwendet werden, die die schlechte Laufzeit des Ansatzes mit konstantem Suchradius, bei einer hohen Partikeldichte, ausgleichen.

## 5.2. Geeignete Suchdatenstrukturen

Das Ziel einer Suchdatenstruktur ist es, Daten so zu speichern, dass möglichst effizient nach ihnen gesucht werden kann. Außerdem soll auch das Einfügen und Löschen dieser Daten effizient sein. Für dieses Problem gibt es, bezogen auf die zu suchenden Nachbarpartikel, mehrere verschiedene Lösungen. Hierzu gehören Gitterverfahren, wie z.B. reguläre Gitter, aber auch baumartige Suchstrukturen. Bei der Entwicklung der Nachbarschaftssuche wurde ein reguläres Gitter verwendet, da der Suchradius für alle Partikel durch die Verwendung eines für alle Partikel gleichen Glättungsradius identisch ist. Außerdem müssten auch baumartige Suchstrukturen durch die Bewegung vieler Partikel in jedem Zeitschritt neu aufgebaut werden. Somit könnten die Vorteile der Bäume durch den komplizierteren Aufbau eine schnelle Suche zu ermöglichen, kombiniert mit der Unterstützung für verschiedene Suchradien der Partikel, nicht greifen. Auch die Zellen des regulären Gitters müssen in jedem Schritt neu mit den entsprechenden Partikeln gefüllt werden und es ist nötig, häufig Partikel darin zu suchen. Jedoch ist es bei Gitterverfahren deutlich einfacher möglich, zusätzlichen Aufwand, wie er bei der Verwendung dynamischer Datenstrukturen der Bäume entstehen würde, durch eine geeignete Abbildung dieser Daten auf Arrays fester Länge, zu vermeiden. Zur Reduktion der zu speichernden Daten wird das reguläre Gitter zusätzlich mit Hashing kombiniert. Hierzu wird jeder Gitterzelle ein Hashwert zugewiesen und die Partikel der jweiligen Zelle unter diesem Hashwert in einer Hashtabelle abgespeichert. Somit wird kein zusätzlicher Speicher für leere Gitterzellen benötigt, und es kann eine unendliche Ausdehnung des Gitters simuliert werden.

### 5.2.1. Reguläres Gitter

Der gegebene dreidimensionale Raum wird zur möglichen Verringerung der Anzahl der nötigen Nachbarschaftstests in gleichgroße, würfelförmige Gitterzellen, deren Seitenlängen dem Suchradius entsprechen, unterteilt. Somit muss nicht jeweils der gesamte Raum auf mögliche Nachbarpartikel untersucht werden, wenn die Seitenlängen der Gitterzellen dem Suchradius entsprechen, weil so nur die eigene Gitterzelle des Partikels und die 26 Nachbarzellen als Orte für mögliche Nachbarpartikel in Frage kommen. Die einzelnen Schritte bei dieser Vorgehensweise sind für jedes Partikel $p_i(t)$ zu jedem Zeitschritt $t$ die Folgenden:

1. Alle weiteren Partikel in der Gitterzelle des aktuellen Partikels werden als mögliche Nachbarkandidaten ausgewählt

2. Alle Partikel in den 26 Nachbarzellen werden zu Nachbarkandidaten

3. Ein Abstandstest bestimmt die richtigen Nachbarn aus allen gefundenen Nachbarkandidaten

Hierbei ist für den Abstandstest die Unterscheidung der Suchstrategie, ob eine feste Anzahl von Nachbarn oder ein fester Suchradius zur Auswahl verwendet wird, zu treffen. Entscheidend für den Speicherverbrauch ist in jedem Fall, wie die Daten in jeder Gitterzelle gespeichert werden. Die einfachste Variante, jeder Gitterzelle so viel Speicher zur Verfügung zu stellen, dass jedes Partikel gespeichert werden kann, ist durch den hieraus entstehenden kubischen

Speicherverbrauch nur für sehr kleine Gitter durchführbar. Eine Variante diesen Speicherbedarf zu umgehen ist, die Partikel jeder Gitterzelle in einer linearen Liste zu speichern. Hierdurch ist nur linearer Speicherplatz in Bezug auf die Partikelanzahl nötig. Allerdings entsteht ein zusätzlicher Rechenaufwand durch die Verwaltung der dynamischen Datenstruktur der linearen Liste. Daher ist die geeignetste Lösung für die Verbesserung der Laufzeit, reguläre Gitter mit Hashing zu kombinieren und für die Speicherung der Partikel unter ihrem Hashwert auf dynamische Datenstrukturen zu verzichten und stattdessen Arrays fester Länge zu verwenden. (siehe Kapitel 5.3.3).

### 5.2.2. Hashing

Da es bei der Nachbarschaftssuche auch auf den Speicherverbrauch ankommt, welcher durch das Abspeichern von leeren Gitterzellen stark ansteigen würde, ist Hashing als geeignete Ergänzung zu den regulären Gittern ausgewählt worden. Für das Hashing werden Schlüssel $s$ für Objekte einer Menge $U$ berechnet, die als Universum bezeichnet wird. Anschließend werden diese Objekte unter ihrem Schlüsselwert abgespeichert. Hier bilden die auf ganze Zahlen gerundeten Koordinaten der Gitterzellen grid(x,y,z) das Universum $U$, zu denen jeweils ein Schlüssel $s$ berechnet werden muss. Die Partikel werden dann unter dem Schlüssel $s$ der Zelle, in der sie sich befinden, in einer Hashtabelle abgespeichert. Die Menge aller möglichen solcher Schlüssel ist dann die Schlüsselmenge $S$. Die Hauptidee ist nun, die Schlüsselmenge $S$ so zu wählen, dass sie im Verhältnis zum Universum deutlich kleiner ist. Hierfür ist eine Hashfunktion $h$ so zu wählen, dass bei der Abbildung der Gitterzellen auf die Schlüsselmenge möglichst wenige Kollisionen entstehen. Kollision bedeutet, dass für mehrere verschiedene Elemente aus dem Universum $U$ mit der Hashfunktion derselbe Schlüssel berechnet wird, woraus folgt, dass bei der Suche nach einem Partikel mit Hilfe des Schlüssels seiner Gitterzelle auch alle weiteren Partikel mit dem selben Schlüssel aus anderen Gitterzellen gefunden werden. Dies erfordert einen zusätzlichen Aufwand zur Aussortierung dieser doppelt durchsuchten und eventuell auch doppelt als Nachbar eingetragenen Partikel und erzeugt somit eine höhere Laufzeit. Für den Vergleich der Anzahl der durchschnittlich verursachten Kollisionen wurden zwei verschiedene Hashfunktionen verwendet. Die erste Hashfunktion (siehe [Hje06]) lautet:

$$h(grid(x,y,z)) = (P1 \cdot grid_x \oplus P2 \cdot grid_y \oplus P3 \cdot grid_z) \bmod N. \tag{5.1}$$

In der Gleichung 5.1 sind $P1 = 73856093$, $P2 = 55924061$, $P3 = 83492791$ Primzahlen, die größer als das Universum $U$ sind, um mögliche Kollisionen, aufgrund von gemeinsamen Teilern, zu vermeiden. $grid_x$, $grid_y$ und $grid_z$ sind die auf ganze Zahlen gerundeten Koordinaten der jeweiligen Gitterzelle $grid(x,y,z)$, die mit den Primzahlen multipliziert werden. Anschließend wird das Ergebnis mit $XOR$ bitweise verknüpft. $N$ ist die Größe der Hashtabelle und sollte ebenfalls eine Primzahl sein, die ca. 1.5 bis 2.5 fach so groß wie die Anzahl der mit Partikeln gefüllten Gitterzellen ist. Somit ist sichergestellt, dass in einem mathematischen Körper gerechnet wird und keine zusätzlichen Kollisionen in der Hashfunktion durch Nullteiler entstehen können.

Die zweite Hashfunktion wurde von der Projektgruppe entwickelt und ist folgendermaßen definiert:

$$h(grid(x,y,z)) = (grid_x \cdot \lambda^2 + grid_y \cdot \lambda + grid_z) \bmod N. \tag{5.2}$$

Das Ziel bei der Entwicklung der Hashfunktion in der Gleichung 5.2 war es einerseits die

Struktur der Abbildung der Partikel in einem dreidimensionalen Raum auf würfelförmige Gitterzellen zu berücksichtigen und andererseits mit der Abhängigkeit der Funktion von der Partikelanzahl dafür zu sorgen, dass im Durchschnitt kein Worst-Case im Bezug auf die Anzahl der Kollisionen einer Szene entsteht. Hierzu wird die Partikelanzahl $\mathbf{p}_{max}$ in der Gleichung 5.3 als Volumen eines Würfels interpretiert. Anschließend wird mit der dritten Wurzel die Seitenlänge $\lambda$ des Würfels bestimmt. Diese Seitenlänge wird in der Gleichung 5.2 quadratisch mit $grid_x$ und linear mit $grid_y$ multipliziert, wobei $grid_z$ unverändert verwendet wird. Das Ergebnis wird schließlich mit einer Addition verknüpft, da sich diese empirisch als besser erwiesen hat als eine bitweise „XOR“ Verknüpfung. $grid_x$, $grid_y$ sind $grid_z$ die auf ganze Zahlen gerundeten Gitterzellenkoordinaten der Gitterzelle $grid(i)$. $N$ ist auch hier die Größe der Hashtabelle und sollte eine Primzahl sein. Die Würfellänge $\lambda$ sorgt also dafür, dass die Wahl guter oder schlechter Werte in der Hashfunktion nicht nur von der verwendeten Szene, sondern auch von der Anzahl der vorhandenen Partikel abhängt.

$$\lambda = \lceil \sqrt[3]{\mathbf{p}_{max}} \rceil \tag{5.3}$$

## 5.3. Verwendete Suchverfahren

Um die bei großen Partikeldichten zunächst nicht akzeptable Gesamtlaufzeit des Programms zu verbessern, wurden zu den grundlegenden Nachbarschaftssuchverfahren einige Verbesserungen von der Projektgruppe entwickelt. Hierdurch wurde außerdem eine genaue Beurteilung der Effizienz dieser Verfahren im Vergleich ermöglicht. Zu den grundlegenden Verfahren gehören das „naive Verfahren“, welches den geringsten Speicherverbrauch, aber die größte Laufzeit hat. Außerdem das reguläre Gitter mit fester Speicherzuodnung für jedes Partikel in jeder Gitterzelle als Beispiel für ein Verfahren mit kubischen Speicherverbrauch im Tausch für eine geringer Laufzeit. Als Grundlage für Verbesserungen diente aber hauptsächlich das schon stark optimierte „Spatial Hashing“, welches dennoch ohne weitere Verbesserungen keine akzeptabele Gesamtlaufzeit des Programms bei allen Partikeldichten ermöglicht.

### 5.3.1. Naives Verfahren

Wie der Name schon sagt, ist dies ein sehr simples und ineffizientes Suchverfahren. Jedes Partikel bildet jeweils einen Nachbarkandidaten zu jedem anderen Partikel und sofern sie sich in einem Abstand kleiner oder gleich dem konstanten Suchradius befinden, werden sie durch einen Abstandstest als Nachbarn eingestuft. Somit hat jedes der $n$ Partikel zunächst $(n-1)$ Nachbarkandidaten. Das Resultat ist eine Nachbarschaftssuche, die durch den nötigen Abstandstest eine quadratische Laufzeit $O(n^2)$, aber linearen Speicherverbrauch $O(n)$ hat. Diese quadratische Laufzeit bei der Nachbarschaftssuche hat allerdings einen zu hohen negativen Einfluß auf die Gesamtlaufzeit des Programms. Daher dient das „Naive Verfahren“ nur als ein Referenzmodell in den Punkten Laufzeit und Speicherverbrauch für die Nachbarschaftssuche.

### 5.3.2. Reguläres Gitter mit fester Speicherzuodnung

Das „Reguläre Gitter“ ist ein Verfahren, welches die Laufzeit durch starken Speicherverbrauch reduziert. Somit entstammt dieses Verfahren genau der gegenteilige Idee zum „Naiven Verfahren“. Hierzu wird der dreidimensionale Raum mit Hilfe eines regulären Gitters in achsenparallele, gleich große, rechtwinklige Gitterzellen unterteilt, deren Seitenlängen dem Suchradius

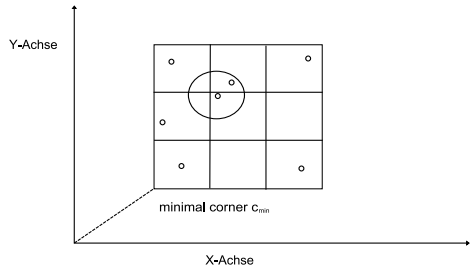



Abbildung 5.1.: Der „Minimal-Corner“ repräsentiert den Abstand zwischen dem Gitterfeld und dem Ursprung des Koordinatensystems.

entsprechen. Somit handelt es sich um ein kartesiches Gitter und jede Gitterzelle entspricht einem Würfel im dreidimensionalen Raum. Durch diese Anordnung entsteht automatisch eine Nachbarschaftsbeziehung der Gitterzellen, wodurch eine Nummerierung der Gitterzellen ermöglicht wird, die zur Speicherung der Partikel jeder Zelle in einem dreidimensionalen Array dient. Für jede Gitterzelle muss genügend Speicher reserviert werden, um eine festgelegte Anzahl Partikel aufnehmen zu können. Hierdurch ist das Gitter auf einen Bereich endlicher Ausdehnung beschränkt. Jedes Partikel wird in die Arrayposition der jeweiligen Gitterzelle im Raum eingetragen. Anschließend wird jedes Partikel durchlaufen und der Index seiner Gitterzelle berechnet. Dann werden mit einem Abstandstest die Kandidaten in der eigenen Gitterzelle und in den 26 Nachbarzellen überprüft und bei bestandenem Test als Nachbarn in die Nachbarliste eingetragen. Somit wird keine dynamische Speicherverwaltung verwendet, die sich negativ auf die Laufzeit auswirken könnte. Allerdings ergibt sich der Nachteil, dass der Speicherverbrauch kubisch bezogen auf die Anzahl der durch Partikel besetzten Gitterzellen ist. Genau wie das „Naive Verfahren“ dient das „Reguläre Gitter“ als Referenzmodell. Es ist ein Verfahren, welches die Laufzeit durch extremen Speichereinsatz verbessert (siehe Kapitel 12.1.7).

### 5.3.3. Spatial-Hashing

Das „Spatial-Hashing“ [Hje06] ist eine um Hashing erweiterte Form der Gittersuche. Das Ziel ist es, den Speicherverbrauch zu senken, indem das Speichern von leeren Zellen entfällt. Außerdem wird auf die Verwendung von dynamischen Speicherstrukturen zur Verbesserung der Laufzeit verzichtet. Der Raum wird analog zum „Regulären Gitter“ in gleich große Gitterzellen unterteilt. Dabei ist zu berücksichtigen, dass die Seitenlänge einer Gitterzelle dem Suchradius entspricht, damit im dreidimensionalen Raum nur in der eigenen und den 26 direkt benachbarten Gitterzellen gesucht werden muss. Nach der Raumunterteilung werden den einzelnen Gitterzellen Indizes zugewiesen, welche aus den jeweiligen Koordinaten der $x-$, $y-$ und $z-$Achse bestehen. Hierbei muss beachtet werden, ob der betrachtete Raum seinen Ursprung an der Null-Koordinate des Koordinatensystems hat oder verschoben ist. Sollte der zweite Fall zutreffen, muss wie in Abbildung 5.1 dargestellt, der so genannte „Minimal-Corner“ $c_{min} = (x_{min}, y_{min}, z_{min})$ bestimmt und in der Hashfunktion verwendet werden. Zur Berechnung des jeweiligen Zellindex $I$, ohne den „Minimal Corner“, wird die Gleichung 5.4 verwendet. Hierbei werden die jeweiligen Koordinaten $x$, $y$ und $z$ durch den doppelten Suchradius $d = 2 \cdot r$ dividiert und auf die nächst kleinere ganze Zahl abgerundet,

welches der Mitte einer Zelle entspricht:

$$I(x, y, z, d) = (\lfloor x/d \rfloor, \lfloor y/d \rfloor, \lfloor z/d \rfloor). \tag{5.4}$$

Sollte ein „Minimal Corner“ vorhanden sein, müssen auch hierbei die jeweiligen Koordinaten $x$, $y$ und $z$ durch den doppelten Suchradius $d = 2 \cdot r$ dividiert und auf die nächst kleinere ganze Zahl abgerundet werden. Zusätzlich muss die Verschiebung des „Minimal-Corners“ mit den jeweiligen $x_{min}$, $y_{min}$ und $z_{min}$ Werten von den Koordinaten subtrahiert werden:

$$I(x, y, z, d) = \left( \left\lfloor \frac{x - x_{min}}{d} \right\rfloor, \left\lfloor \frac{y - y_{min}}{d} \right\rfloor, \left\lfloor \frac{z - z_{min}}{d} \right\rfloor \right). \tag{5.5}$$

Mit dem Index der Zellen und der Hashfunktion (Gleichung 5.1 oder Gleichung 5.2) kann nun der jeweilige Hashwert ermittelt werden. Die Hashwerte für Partikel $p_i$ werden in einem „Hash-Value-Buffer“ (siehe Abbildung 5.2) jeweils an der Postion $hashvalue[i]$ gespeichert.

| 1 | 3 | 5 | 0 | 13 | 3 | 2 | 1 | 8 |
|---|---|---|---|---|---|---|---|---|

Abbildung 5.2.: „Hash-Value-Buffer“ mit den jeweiligen $hashvalue[i]$-Einträgen

Aus dem „Hash-Value-Buffer“ wird ein Historgramm H berechnet (siehe Abbildung 5.3). Die Größe des Histogramms entspricht dem größten Wert $N$ der „Hashtable“. Der jeweilige

| 1 | 2 | 1 | 2 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Abbildung 5.3.: Ein Histogramm, welches mit Hilfe des „Hash-Value-Buffers“ berechnet wurde

Wert, der in dem Histogramm $H(i)$ an Position $i = 0, 1, ...N - 1$ eingetragen ist, ist davon abhängig, wie oft der Wert $i$ in der zuvor genannten „Hashtable“ vorhanden ist:

$$H(i) = \sum_{n=1}^{N} \gamma(i(n) - i) \tag{5.6}$$

$$\gamma(x) = \begin{cases} 1 & f\ddot{u}r \quad \gamma = 0 \\ 0 & sonst \end{cases} . \tag{5.7}$$

Die Funktion $\gamma(x)$ (siehe Gleichung 5.6 und Gleichung 5.7) liefert eine Eins wenn der aktuell zu zählende Wert $i$ mit dem aktuellen Wert in der „Hashtable“ übereinstimmt und ansonsten eine Null. Das Histogramm dient dazu, eine „Offsettable“ (siehe Abbildung 5.4) zu bestimmen. Diese enthält genau so viele Elemente, wie das Histogramm. Im ersten Arrayeintrag wird eine Null eingetragen. Die folgenden Felder werden mit einer kumulativen Funktion (siehe Gleichung 5.8) aus den Einträgen des Histogramms $H(j)$ berechnet. Daher stellt die“ Offsettable“ eine monoton steigende Funktion dar:

$$Offset(N) = \sum_{j=1}^{N} H(j). \tag{5.8}$$

Die Berechnung findet ihren Abschluss in einer neuen „Hashtable“ (siehe Abbildung 5.4). In dieser „Hashtable“ wird an Position $k$ eingetragen, an welchem Index der „Hashtable“ sich der

Hashwert $k$ befindet. Außerdem wird mit Hilfe des Histogramms die Größe des Buckets in der neuen „Hashtable“ für den jeweiligen Hashwert bestimmt. Ein Bucket ist ein Bereich in der neuen „Hashtable“, in dem alle Positionen des Vorkommens eines Hashwertes in der „Hashtable“ eingetragen sind. Die Einträge innerhalb eines Buckets stehen also für Partikel, die sich in der selben Gitterzelle befinden. Um die Anfangsposition eines Buckets für den Hashwert $i$ zu bestimmen, wird der Wert der „Offsettable“ an der Position $i$ verwendet. Mit Hilfe der neuen

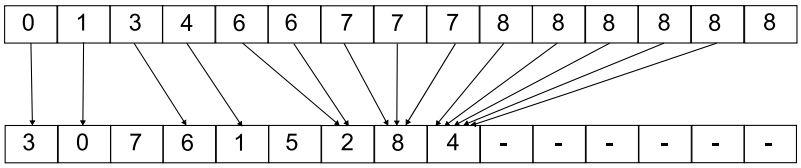



Abbildung 5.4.: Das obere Array stellt die Offsettable dar. Unten ist die neue berechnete „Hashtable“.

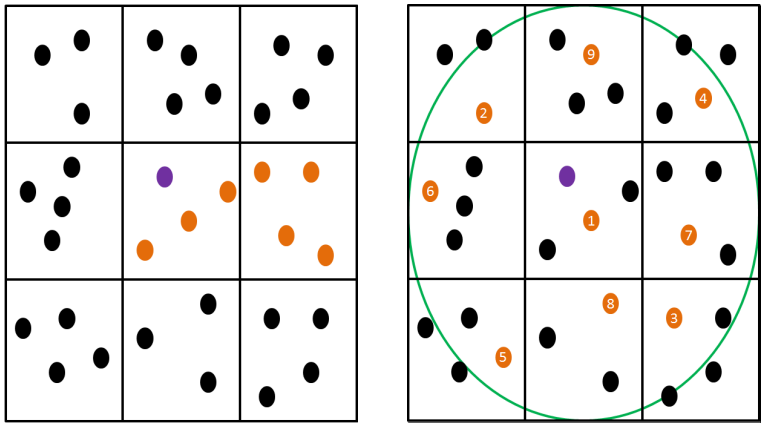
„Hashtable“ beginnt das eigentliche Suchen der Nachbarn eines jeden Partikels im „Spatial-Hashing“. Hierfür wird die Nachbarschaftssuchmethode mit festem Suchradius verwendet. Zunächst wird die eigene Gitterzelle des aktuellen Partikels durchsucht und anschließend die 26 Nachbarzellen im dreidimensonalen Raum. Anhand der neuberechenten „Hashtable“ ist es nun möglich, für einen gegebenen Zellindex alle Partikel dieser Gitterzelle zu bestimmen. Die so gefundenen Nachbarkandidaten werden anschließend mit Hilfe eines Abstandstests als Nachbarn erkannt oder verworfen. Durch diese geschickte Vermeidung von dynamischen Datenstrukturen ist das „Spatial-Hashing“ ein wichtiges und grundlegendes Verfahren, welches weiter Verbesserungen lohnenswert erscheinen lässt. (siehe Kapitel 12.1.1).

### 5.3.4. Capped-Spatial-Hashing

Das „Capped-Spatial-Hashing“ ist ein Verfahren, welches den Ansatz eines festen Suchradius mit dem Ansatz einer festen Anzahl zu suchender Nachbarn kombiniert. Bis auf die Auswahl der zu testenden Nachbarkandidaten ist die Berechnung hierfür vollkommen analog zum „Spatial-Hashing“ (siehe Kapitel 5.3.3). Für die Beschränkung der Nachbaranzahl ist es nowendig zuerst die Nachbarkandidaten aus der eigenen Zelle des aktuellen Partikels, für den die Nachbarn berechnet werden sollen, zu nehmen. Sollten dies nicht genügend Nachbarkandidaten sein, werden weitere Kandidaten jeweils aus gegenüberliegenden Gitterzellen der 26 Nachbarzellen genommen. Dies ist notwendig um zu verhindern, dass alle Nachbarn auf einer Seite des Partikels gefunden werden, wodurch sich die später berechneten Kräfte einseitig auf das Partikel auswirken würden (Siehe Abbildung 5.5). Durch die Kombination dieser beiden Ansätze werden die jeweiligen Vorteile kombiniert, was bedeutet, dass die Nachbarn nahezu gleichverteilt sind und die Laufzeit fast unabhängig von der Dichte der Partikel ist (siehe Kapitel 12.1.4).

### 5.3.5. Spatial-Hashing-Vertex

Eine weitere Variante des „Spatial-Hashing“ ist das „Spatial-Hashing-Vertex“. Der grundlegende Unterschied zwischen dem „Spatial-Hashing“ und dem „Spatial-Hashing-Vertex“ ist die Positionierung des Mittelpunktes des Suchradius. Wie bereits erwähnt, wird beim „Spatial-Hashing“ der Suchradius so positioniert, dass er seinen Mittelpunkt in der Mitte der jeweiligen Gitterzelle




(a) Veranschaulichung des Problems Nachbarn bei einem Gittersuchverfahren mit konstanter Nachbaranzahl gleichverteilt zu finden, wenn nicht alle Partikel durchsucht werden sollen und die Nachbarn vollständig aus wenigen Gitterzellen genommen werden

(b) Veranschaulichung der Lösung Nachbarn bei einem Gittersuchverfahren mit konstanter Nachbaranzahl gleichverteilt zu finden. Hier werden die Nachbarn zufällig in jeweils gegenüberliegenden Gitterzellen gesucht. Außerdem wird der konstante Suchradius (grün) eingehalten

Abbildung 5.5.: Veranschaulichung des Problems Nachbarn bei einem Gittersuchverfahren mit konstanter Nachbaranzahl gleichverteilt zu finden, wenn nicht alle Partikel durchsucht werden sollen und deshalb nicht die Nachbarn mit dem geringsten Abstand verwendet werden können

hat (siehe Kapitel 5.3.3). Somit müssen beim“Spatail-Hashing“ insgesamt 27 Gitterellen, die aktuelle Zelle und die 26 Nachbarzellen, im dreidimensionalen Raum in die Nachbarschaftsuche einbezogen werden. Der Ansatz des „Spatial-Hashing-Vertex“ ist, die Anzahl der Nachbarzellen die durchsucht werden müssen, zu reduzieren. Hierfür müssen zwei wesentliche Veränderungen vorgenommen werden. Als erstes muss die Gitterzellengröße verdoppelt werden. Hatten die Zellen beim „Spatial-Hashing“ noch eine Kantenlänge und somit einen Suchradius von $r$, so beträgt die Kantenlänge der Gitterzellen in „Spatial-Hashing-Vertex“ $2 \cdot r$. Die zweite Veränderung ist die Verlagerung des Mittelpunktes des Suchradius. Der Mittelpunkt wird dorthin gelegt, wo sich vier Gitterzellen berühren, also auf die Ecken des Gitters. Da der Raum in ein kartesisches Gitter unterteilt ist, liegen die Eckpunkte der jeweiligen Gitterzellen in einem äquidistanten Abstand zueinander. Im Vergleich zum „Spatial Hashing“ ist der Raum, welcher beim „Spatial-Hashing-Vertex“ durchsucht werden muss, größer. Beim „Spatial-Hashing“ wird ein Raum der Größe $(3 \cdot r)^3$ durchsucht, dies entspricht $27 \cdot r^3$. Beim „Spatial-Hashing-Vertex“ ist der durchsuchte Raum jedoch $(4 \cdot r)^3$, also $64 \cdot r^3$ groß (siehe Abbildung 5.6). Somit ist beim „Spatial-Hashing-Vertex“ der durchsuchte Raum um den Faktor $2,37$ größer. Die Anzahl der zu durchsuchenden Nachbarzellen reduziert sich jedoch von 26 auf 8 (siehe Kapitel 12.1.5).

### 5.3.6. Spatial-Hashing-Automatic-Smoothing-Length

Die Idee des „Spatial-Hashing-Automatic-Smoothing-Lenght“ ist, die Nachbarschaftssuche mit einer dynamischen Anpassung des Suchradius zu kombinieren. Die Berechnung hierfür verläuft analog zum „Spatial-Hashing“ (siehe Kapitel 5.3.3). Zusätzlich wird jedoch die durchschnittliche Anzahl der gefundenen Nachbarpartikel berechnet:

$$\frac{1}{\mathbf{p}_{max}} \sum_{i=1}^{\mathbf{p}_{max}} \sum_{j=1}^{N} \tau(P_i(t), P_i(t)]) \tag{5.9}$$

$$\tau(i,j) = \begin{cases} 1 & falls \quad P_i(t) \text{ Nachbar von } P_j(t) \\ 0 & sonst \end{cases} \tag{5.10}$$

und anschließend mit einem vom Benutzer vorgegebenen Bereich verglichen. Hierbei ist $\sum_{i=1}^{\mathbf{p}_{max}}$ eine Summierung über alle Partikel und $\sum_{j=1}^{N}$ eine Summierung über alle Nachbarpartikel $N$. Wobei $\tau(P_i(t), p_j(t))$ genau dann eins ist, wenn $P_j(t)$ das Nachbarpartikel von $P_i(t)$ zum Zeitpunkt $t$ ist. Schließlich erfolgt mit $\frac{1}{\mathbf{p}_{max}}$ eine Normierung über die Anzahl der Partikel. Liegt die durchschnittliche Nachbaranzahl unterhalb des angegebenen Bereichs, wird der Suchradius anschließend vergrößert. Liegt die durchschnittliche Nachbaranzahl oberhalb des angegebenen Bereichs, wird der Suchradius anschließend verkleinert (siehe Abbildung 5.7). Bei einer durchschnittliche Nachbaranzahl innerhalb des Bereichs wird der Suchradius unverändert gelassen. Die Anpassung erfolgt hierbei jeweils um einen festen Wert. Da es passieren kann, dass diese Anpassung der Änderung der Nachbaranzahl nicht schnell genug folgen kann (Steigungsüberladung), oder dass die Anpassung zu groß ist und somit abwechselnd ein zu großer und zu kleiner Wert gesetzt wird (Feinrauschen), ist auch die Anpassungsrate vom Anwender einstellbar.

### 5.3.7. Optimized-Spatial-Hashing

Der Ansatz des „Optimized-Spatial-Hashing“ basiert auf der Idee des „Optimized Spatial Hashing for Collision Detection of Deformable Objects“ [Tes03], ohne jedoch den Ansatz für die Kollision zwischen dynamisch verformbarer Geometrie umzusetzen, da dies im Programm nicht vorgesehen ist. Jede Gitterzelle enthält eine lineare Liste zur Speicherung der Partikel unter ihrem Hashwert als Nachbarkandidaten, wodurch der Speicherverbrauch linear zur Anzahl der

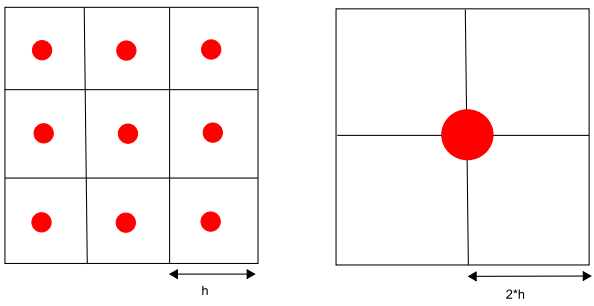
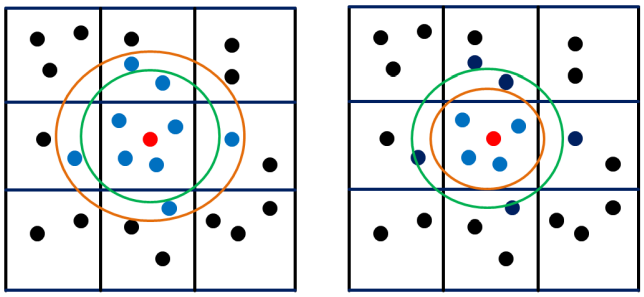



Abbildung 5.6.: Das linke Gitter zeigt Stellen an, auf die das „Spatial-Hashing“ rundet (rote Punkte). Das linke Gitter zeigt an, auf welchen Punkt das „Spatial-Hashing-Vertex“ rundet (roter Punkt).



(a) Verkleinerter Radius: Der braune Kreis zeigt den aktuell verwendeten Suchradius. Der Grüne stellt den verkleinerten Suchradius dar.

(b) Vergrößerter Radius: Der braune Kreis zeigt den aktuell verwendeten Suchradius. Der Grüne stellt den vergrößerten Suchradius dar.

Abbildung 5.7.: Der benutzerdefinierte Nachbarschaftsbereich liegt zwischen 5-6 Nachbarn. Das Bild a zeigt eine nötige Verkleinerung und Bild b eine nötige Vergrößerung.

Partikel ist. Hierbei entsteht ein großer „Overhead“ durch die Verwaltung der verwendeten dynamischen Speicherstruktur, welche sich vor allem durch die dynamische Anforderung von Speicher und dem Geschwindigkeitunterschied der CPU zu dem Speicher ergibt. Daher wird zur Verbesserung der Laufzeit eine Markierung der linearen Listen eingeführt welche angibt, ob eine lineare Liste aktuelle Informationen enthält oder nicht. Hierzu werden die berechneten Zyklen $z$ der Simulationsschleife gezählt und bei jedem Eintrag eines Nachbarkandidaten zuerst überprüft, ob die Markierung der entsprechenden Gitterzelle aktuell ist, also dem aktuellen Zyklus $z$ entspricht. Wenn dies nicht der Fall ist, wird der Inhalt der linearen Liste gelöscht und die Markierung auf den aktuellen Zyklus $z$ gesetzt. Ist die Markierung beim Eintrag eines Partikels jedoch aktuell, kann daraus geschlossen werden, dass in diesem Zyklus schon ein anderes Partikel neu in diese Liste eingetragen wurde, wodurch alte Einträge auf jeden Fall vorher gelöscht wurden. Durch dieses Vorgehen werden nur diejenigen Listen der Gitterzellen in dem jeweiligen Zyklus gelöscht, in deren Bereich Partikel existieren. Zur eigentlichen Berechnung der Nachbarn wird für jedes Partikel der Index der zugehörige Gitterzelle mit seinem Hashwert berechnet. Dann wird in der eigenen Gitterzelle und den 26 Nachbarzellen nach Nachbarkandidaten gesucht. Schließlich wird mit Hilfe einer Abstandsfunktion überprüft, ob diese gefundenen Nachbarkandidaten innerhalb des Suchradius liegen und gegebenenfalls in die entsprechende Nachbarliste eingetragen. Dieser Ansatz hat sich jedoch als ineffizient erwiesen (siehe Kapitel 12.1.6).

### 5.3.8. Union-Spatial-Hashing

Das „Union-Spatial-Hashing“ ist ein auf der Grundlage des „Spatial-Hashing“ weiterentwickeltes Verfahren, welches mehrfaches Durchsuchen von Gitterzellen auf Grund von Kollisionen der Hashfunktion vermeidet. Bei einer Kollision der Hashwerte werden mehreren Gitterzellen der selbe Hashwert zugewiesen. Dadurch werden sämtliche Partikel, welche sich in den Gitterzellen mit selbem Hashwert befinden, unter diesem identischen Hashwert abgelegt. Zusätzlich wird diese große Partikelmenge für jede Gitterzelle mit dem selben Hashwert vollständig durchsucht. Dadurch werden diese Partikel mehrfach durchsucht, was zu einer höheren Laufzeit führt, und

die jeweiligen Partikel werden mehrfach als Nachbarn eines bestimmten Partikels eingetragen. Hierdurch ist außerdem ein zusätzliches Aussortieren mehrfach eingetragener Nachbarn nötig. Um dieses Problem zu beheben, werden zunächst alle Berechnungen analog zum „Spatial-Hashing“ ausgeführt (siehe Kapitel 5.3.3). Zusätzlich werden jedoch alle Hashwerte von der eigenen und den 26 Nachbarzellen vor der Durchsuchung der Nachbarzellen auf Übereinstimmung überprüft. Hierzu wird eine duplikatfreie Liste der Hashwerte dieser Gitterzellen erstellt, was eine mehrfache Durchsuchung der Gitterzellen bei Übereinstimmung der Hashwerte unnötig macht. Somit führen Kollisionen der Hashwerte nicht zu einer Vergrößerung der Laufzeit. Hierdurch wird der Nachteil der größeren Streuung der neuen Hashfunktion ausgeglichen, wodurch der Wert der neuen Hashfunktion seinen negativen Einfluß, bei eventuell vorkommenden „Worst-Case-Situationen“, auf die Ausführungsgeschwindigkeit des Gesamtprogramms verliert (siehe Kapitel 12.1.3).

### 5.3.9. Symmetry-Aware-Grid

Das „Symmetry-Aware-Grid“ ist das beste Nachbarschaftssuchverfahren, welches auf den Erfahrungen aller anderen Verfahren beruht. Die wesentliche Idee ist, den Nachteil des „Spatial-Hashing“, die Symmetrie nicht zu verwenden, auszugleichen. Hierzu ist ein neuer sich vom „Spatial-Hashing“ unterscheidender Anzatz nötig, der die Symmetrie der Nachbarschaftsbeziehung bei der Suche mit einem festen Suchradius ausnutzt. So muß nur noch die Hälfte der Nachbarschaftsbeziehungen berechnet werden. Durch eine neue „Offsettable“, welche nur die Indizes aktiver Gitterzellen verwaltet, ist es hierbei möglich, ausschließlich alle aktiven Gitterzellen, also Zellen mit Partikeln, zu durchlaufen und von diesen aus gesehen 13 anstatt wie bisher 26 Nachbarzellen zu durchsuchen. Es ist bei diesem Verfahren also nicht nötig über die Partikel zu iterieren, sondern nur über die aktiven Gitterzellen, wodurch sich ein deutlicher Gewinn in der Ausführungsgeschwindigkeit ergibt (siehe Kapitel 12.1.8). Die Abbildung 5.8 zeigt ein zweidimensionales Beispiel, das verdeutlicht, dass es ausreicht bei einer Iteration über die Zellen eines Gitters nur die Hälfte der Nachbarschaftsbeziehungen zu testen. In diesem zweidimensionalen Fall reichen vier durch eine individuelle Farbe dargestellte Beziehungen, um die Nachbarn in allen acht benachbarten Gitterzellen zu finden.

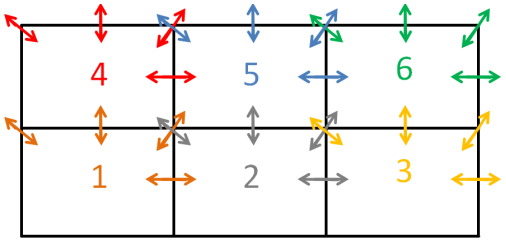



Abbildung 5.8.: Zweidimensionale Verdeutlichung, dass es ausreicht, nur die Hälfte der Nachbarschaftsbeziehungen zu testen

# 6. Kraftberechnung

In diesem Kapitel wird die Kraftberechnung vorgestellt, die zur Berechnung der auf das jeweilige Partikel wirkenden Kräfte dient. Es wird auf die Kontinuitätsgleichung und die Impulsgleichung eingegangen, die ein wesentlicher Bestandteil der Fluiddynamik in YASPHS sind. Ebenfalls werden die verwendeten Kräfte und deren Umsetzung im SPH-Formalismus beschrieben, so dass diese in einem exemplarischen Simulationszyklus verwendet werden können. Die Kräfte bewirken im Simulationszyklus die auf die Partikel wirkende Beschleunigung, so dass nach Anwendung geeigneter Integrationsverfahren die neue Position und Geschwindigkeit der Partikel resultiert.

Die Berechnung der Kräfte verlangt vorher festgelegte bzw. berechnete Konstanten, Variablen und Kernelfunktionen. Zur Anwendung der Kernelfunktionen müssen ebenfalls die im vorhergenenden Schritt des Simulationszyklus berechneten Nachbarschaftssuchverfahren vorhanden sein.

Die Kernelfunktionen werden benötigt, da es sich bei den Kräften um Feldgrößen handelt, die mit Hilfe dieser Funktionen als Gewichtung über benachbarte Partikel den Feldwert an einer bestimmten Position, meistens an der des betrachteten Partikels, bestimmen.

Vor der Berechnung der Kräfte werden die Feldgrößen Dichte und Druck, die in einem Partikel gespeichert werden, bestimmt. Die Dichte kann berechnet werden, indem über die Partikelmassen der benachbarten Partikel, die über eine Kernelfunktion gewichtet sind, aufsummiert wird. Diese kann dann zur Berechnung des Drucks und folglich der Druckkraft verwendet werden. Die Druckkraft ist nicht die einzige Kraft, die auf ein Partikel wirken kann und dessen Bewegung beeinflusst. Weitere Kräfte, die Einfluss auf das Verhalten der Partikel haben können sind unter anderem die Viskosität, die Gravitation und die Oberflächenspannung. Die Summe dieser Kräfte addieren sich zu einer Resultierenden auf, die das betrachtete Volumen beschleunigen. Es ist nicht zwingend erforderlich, alle Kräfte mit in die Simulation einzubeziehen. Die physikalische Genauigkeit der Simulation hängt jedoch von den jeweils verwendeten Kräften ab.

## 6.1. Physikalischer Hintergrund

Die Dynamik von Flüssigkeiten und Gasen wird unter dem Sammelbegriff Strömungslehre zusammengefasst. Diese findet Anwendung in vielen alltäglichen Bereichen, wie in der Wettervorhersage, in der Luftfahrt und in der Medizin. Die Gleichungen, die die Simulation von Fluiden ermöglichen, sind die Navier-Stokes-Gleichungen. Die in Kapitel 3.1 beschriebenen Methoden approximieren diese Gleichungen und ermöglichen somit die Umsetzung der Fluiddynamik als computergestützte Simulation.

Das physikalische Modell, das zur Simulation von Fluiden notwendig ist, ist durch die Navier-Stokes-Gleichungen gegeben, deren Komponenten in diesem Abschnitt hergeleitet werden. Die Navier-Stokes-Gleichungen sind ein System nichtlinearer, partieller Differentialgleichungen zweiter Ordnung. Der Hauptteil der Gleichungen ist der Impulssatz als Anwendung der

Newtonschen Axiome für ein Kontinuum. Die Newtonschen Axiome beinhalten die drei Grundgesetze der Bewegung: Trägheitsprinzip, Aktionsprinzip, Reaktionsprinzip. Der Impulssatz, der im Folgenden genauer erläutert wird, muss noch durch die Kontinuitätsgleichung und den Energieerhaltungssatz ergänzt werden.

Mit den Anforderung und der physikalischen Genauigkeit, die an das Fluid gestellt werden, variiert das vollständige Gleichungssystem, das die Fluiddynamik beschreibt. So sind die inkompressiblen Navier-Stokes-Gleichungen ein Spezialfall der kompressiblen Navier-Stokes-Gleichungen, die für ein allgemeines ideales Gas gelten und auch als vollständige Navier-Stokes-Gleichungen bezeichnet werden.

Im Gegensatz zu den kompressiblen Gleichungen, welche die Kompression des Fluids ermöglichen (z.B. Gase wie Luft), tritt bei inkompressiblen Flüssigkeiten keine Volumenänderung durch Druck auf. Durch die inkompressiblen Navier-Stokes-Gleichungen lassen sich viele Strömungsprobleme wie Wasser simulieren. Diese Gleichungen setzen voraus, dass sich die Dichten der betrachteten Fluide nicht ändern, da die Realität in diesem Fall nicht mehr korrekt durch dieses Modell wiedergegeben werden kann. Dichteänderungen treten z.B. auf, wenn sich Fluide unterschiedlich stark komprimieren, wie dies bei sich stark ändernden Strömungsgeschwindigkeiten bei Gasen oder starken Temperaturdifferenzen der Fall ist. Dies wird teilweise durch Dichtegradienten kompensiert.

Die Umsetzung des Energieerhaltungssatzes in SPH übertrifft bei weitem die Anforderungen an die Projektgruppe, weshalb auf eine Realisierung in YASPHS verzichtet wurde.

Aufgrund der Beschaffenheit des Energieerhaltungssatzes und dessen aufwändige Umsetzung wurde auf dessen Anwendung in YASPHS verzichtet. Dies ist möglich, da der Energieerhaltungssatz nicht unbedingt zum Schließen der Fluiddynamik erforderlich ist und mit dem Impulssatz und der Kontinuitätsgleichung eine ausreichende Approximation des Fluidverhaltens gegeben ist. Für ein über die Zeit isothermes (beständige Temperatur bei Druckvariationen), viskoses (zähflüssiges) und inkompressibles Fluid lässt sich das System bei Ausgrenzung der Energie- und Zustandsgleichung ausdrücken durch die Impulsgleichung und die Kontinuitätsgleichung [Dar07]. Betrachtet man ein makroskopisches Flüssigkeitselement, welches im Vergleich zum betrachteten Volumen klein ist und sich zum Zeitpunkt $t = 0$ im Ort $\mathbf{X_0}$ befand, ist dessen aktuelle Position zum Zeitpunkt $t$ mit

$$\mathbf{X} = \mathbf{X}(t) = \Phi(\mathbf{X_0}, t) \tag{6.1}$$

gegeben, wobei $\mathbf{X} = (X_1, \ldots, X_d)$ die Orts-Koordinaten in einem festen, kartesischen System sind. Der Geschwindigkeitsvektor als Änderung des Ortes über die Zeit ist dann

$$\mathbf{u}(\mathbf{X}, t) = \frac{d}{dt}\Phi(\mathbf{X_0}, t) = \frac{d}{dt}\mathbf{X}(t) \ . \tag{6.2}$$

Es wird angenommen, dass $\Phi$ invertierbar ist. Dies bedeutet, dass zwei unterschiedliche Teilchen mit unterschiedlichen Startpositionen auch im weiteren Verlauf unterscheidbar bleiben.

### 6.1.1. Kontinuitätsgleichung

Die Kontinuitätsgleichung beschreibt das Verhalten eines Fluids, dessen zeitliche Änderung der Masse $m(t)$ eines Fluidelements Null entspricht [HEF07]. Mit dieser Gleichung wir ausgedrückt, dass die Summe der in ein Volumenelement eintretenden Masse derjenigen der austretenden Masse gleicht.

Sei $V$ ein abgeschlossenes Volumen eines Fluids mit der zugehörigen Dichte $\varrho = \varrho(\mathbf{x}, t)$ als Feldgröße. Abgeschlossen bedeutet, dass das Volumen einen klar definierten Rand besitzt, der ebenfalls zur Volumenmenge gehört. Durch Integration der Dichte über das Volumen resultiert die extensive Größe Masse als Funktion der Zeit:

$$m(t) = \int_V \varrho(\mathbf{x}, t) dV. \tag{6.3}$$

Ist $d\mathbf{A}$ ein Oberflächenelement des betrachteten Volumens, dann tritt in einer Zeiteinheit $\Delta t$ aufgrund der Austrittsgeschwindigkeit $\mathbf{u}$ an dessen Oberfläche die Masse $\oint_A \varrho \mathbf{c} \cdot d\mathbf{A}$ aus. Damit kann die Massenänderung des betrachteten Volumens über die Zeit ausgedrückt werden als

$$\frac{d}{dt} \int_V \varrho dV = - \oint_A \varrho \mathbf{u} \cdot d\mathbf{A} \ . \tag{6.4}$$

Durch das Vertauschen von Differentiation und Integration auf der linken Seite von Gleichung 6.4 ist $dV$ in Eulerschen Koordinaten von $t$ unabhängig und Anwenden des Gaußschen Satzes auf die rechte Seite ist die Formulierung der Kontinuitätsgleichung in differentieller Form

$$\int_V \left( \frac{\partial \varrho}{\partial t} + \operatorname{div}(\varrho \mathbf{u}) \right) dV = 0 \ . \tag{6.5}$$

Da dies für beliebige Kontrollvolumina $V$ gilt, kann der Integrand aus Gleichung 6.5 verschwinden und man erhält die Kontinuitätsgleichung:

$$\frac{\partial \varrho}{\partial t} + \operatorname{div}(\varrho \mathbf{u}) = 0 \ . \tag{6.6}$$

Für inkompressible Flüssigkeiten ist die Dichte $\varrho(\mathbf{x}, t)$ konstant. Damit vereinfacht sich Gleichung 6.6 zu

$$\operatorname{div} \mathbf{u} = 0. \tag{6.7}$$

In der Physik wird ein Vektorfeld $\mathbf{u}$ als quellenfrei angesehen, wenn es die Gleichung dieser Form erfüllt.

### 6.1.2. Impulsgleichung

Ein angenommenes Flüssigkeitselement in Lagrange-Koordinaten mit infinitessimalem, d.h. unendlich kleinem Volumen $dV$, der Masse $dm$ und der Oberfläche $dA$, bewegt sich entlang der Fluidströmung. Auf dieses Element wird der Impulserhaltungssatz, welcher aus dem zweiten und dritten Newton'schen Axiom folgt, angewendet. Unter Impuls wird die Bewegung einer Masse verstanden, die ein Körper enthält und die durch äußere Krafteinwirkung beeinflusst wird. Nach dem Impulserhaltungssatz $\mathbf{F} = \dot{\mathbf{p}}$, mit $\mathbf{F}$ als Kraft und $p = m\mathbf{u}$ der Impuls, entspricht die Resultierende aller auf ein Fluidelement einwirkenden Kräfte der totalen zeitlichen Änderung des Impulses oder dem Produkt aus seiner Masse und Beschleunigung $\mathbf{F} = m\dot{\mathbf{u}}$. Für ein Fluidelement gilt somit

$$d\mathbf{F} = dm \frac{d\mathbf{u}}{dt} = \varrho dV \frac{d\mathbf{u}}{dt}, \tag{6.8}$$

wobei die Masse $m = \varrho V$ gegeben ist als das Produkt aus der Dichte $\varrho$ und dem Volumen $V$.

Diese Gleichung kann mit der substantiellen Ableitung für $\frac{d\mathbf{u}}{dt}$ gelöst werden. Ist $\Phi$ eine in Lagrangeschen Koordinaten gegebene, zeitabhängige und dreidimensional ortsabhängige skalare Größe $\Phi(\mathbf{x}, t)$, so ist deren substantielle Ableitung

$$\frac{d\Phi(\mathbf{x},t)}{dt} = \frac{D\Phi(\mathbf{x},t)}{Dt} = \frac{\partial\Phi}{\partial t} + u_x\frac{\partial\Phi}{\partial x} + u_y\frac{\partial\Phi}{\partial y} + u_z\frac{\partial\Phi}{\partial z} = \frac{\partial\Phi}{\partial t} + \mathbf{u}\cdot\nabla\Phi. \tag{6.9}$$

Hierbei ist $\mathbf{u}$ der dreidimensionale Geschwindigkeitsvektor mit den Komponenten $u_x, u_y, u_z$ in die jeweilige Strömungsrichtung $x, y, z$.

Der Term $\frac{\partial\Phi}{\partial t}$ wird als lokalzeitliche Änderung bezeichnet, da sie eine ortsgebundene Größe ausschließlich über die Zeit beeinflusst (z.B. Messung der Temperatur über ein feststehendes Thermometer). Mit $\mathbf{u}\cdot\nabla\Phi$ wird die konvektive Komponente oder der Advektionsterm bezeichnet. Hier erfährt die Größe Änderungen aufgrund von lokalen Einwirkungen wie Krafteinflüssen oder Temperaturdifferenzen. Der Advektionsterm sorgt für den Transport der Größen im Geschwindigkeitsfeld.

Wird die substantielle Ableitung auf die Geschwindigkeitskomponente aus Gleichung 6.8 angewendet, erhält man komponentenweise

$$dF_i = \varrho dV\frac{du_i}{dt} = \varrho dV\left(\frac{\partial u_i}{\partial t} + (\mathbf{u}\cdot\nabla)u_i\right) \tag{6.10}$$

oder

$$\frac{dF_i}{dV} = \varrho\left(\frac{\partial u_i}{\partial t} + (\mathbf{u}\cdot\nabla)u_i\right)\ . \tag{6.11}$$

Bei konstanter Dichte $\varrho \neq 0$ entfällt auf der rechten Seite der Advektionsterm und die Gleichung 6.11 vereinfacht sich zu

$$\frac{dF_i}{dV} = \varrho\frac{\partial u_i}{\partial t}\ . \tag{6.12}$$

Die Kraft $dF_i$, die auf ein Volumenelement wirkt, setzt sich aus folgenden Komponenten zusammen:

- Innere Kräfte wie die spezifische Druckkraft, die auf die Oberfläche des Fluidelements wirkt und Kräfte mit dissipativen Eigenschaften wie Wärmeleitung, innere Reibung oder Viskosität. Dissipative Eigenschaften beschreiben den Energieverlust eines Systems aufgrund von Wärmeverlusten und dergleichen.

- Neben den internen wirken auf einen Körper auch externe Kraftdichten, deren Resultierende die Bewegungsrichtung vorgibt. Die Kraft $\frac{dF_i}{dV} = \mathbf{f}$ aus Gleichung 6.12 kann als Summe verschiedener Kräfte aufgefasst werden

  $$\mathbf{f}^{external} = \sum_n \mathbf{f}^n\ .$$

  Dabei ist $n$ die Anzahl unterschiedlicher Kraftkomponenten, die die physikalischen Eigenschaften des Fluids beschreiben. Einige Kraftdichten wie die Gravitation können direkt auf die Partikel angewendet werden, andere sind wiederum abhängig von den

Feldgrößen benachbarter Partikel und müssen daher erst mittels SPH berechnet werden. Zu diesen gehört z.B. die Druckkraft.

### 6.1.3. Partikelbasierte Flüssigkeitsdynamik mit SPH

Wie bereits erwähnt, handelt es sich bei SPH um eine auf dem Lagrange-Formalismus aufbauende Methode. In der Lagrangeschen Darstellung wird die Flüssigkeit durch Fluidelemente approximiert, wobei die Bewegung der Flüssigkeit aus dem Ort und der Geschwindigkeit dieser Fluidelemente resultiert. Die Fluidelemente können durch Partikel ausgedrückt werden, die die jeweiligen Flüssigkeitseigenschaften wie Ort, Geschwindigkeit, Masse und andere Größen des Flüssigkeitsvolumens speichern. Wird ein inkompressibles, isothermales Fluid betrachtet und die Anzahl und Masse der Partikel, die dieses Fluid beschreiben, als konstant über die Simulation angesehen, kann aufgrund der so implizierten Massenerhaltung die Kontinuitätsgleichung als konstant betrachtet werden. Um die Dynamik einer solchen Flüssigkeit zu beschreiben ist es somit ausreichend, die Impulsgleichung anzuwenden, wie sie in Kapitel 6.1.2 hergeleitet wurde. Da die Partikelgrößen nur noch von der Zeit $t$ abhängen, wird aus der partiellen Differentialgleichung eine einfache Differentialgleichung

$$\frac{dF_i}{dV} = \varrho \frac{du_i}{dt} \ . \tag{6.13}$$

### 6.1.4. Beschleunigung

In Gleichung 6.13 ist $\frac{du_i}{dt}$ die Beschleunigung $a_i$, die ein Partikel aufgrund der resultierenden Volumenkraft $\frac{dF_i}{dV}$ erfährt

$$a_i = \frac{du_i}{dt} = \frac{1}{\varrho} \frac{dF_i}{dV} \ . \tag{6.14}$$

Die Beschleunigung

$$a_i = \dot{u}_i = \ddot{x}_i \tag{6.15}$$

ist die zeitliche Änderung der Geschwindigkeit. Somit kann aus der errechneten Beschleunigung durch geeignete Integrationsmethoden die aktuelle Geschwindigkeit und durch erneute Integration die Position des Partikels berechnet werden. Geeignete Integrationsverfahren sind in 12.4 beschrieben.

## 6.2. Masse

Die Masse $m$ eines Partikels ist eine initial vorgegebene Konstante, die über eine vorgegebene Dichte $\varrho_0$, dem Volumen $V$ und der Anzahl der Partikel $n$ festgelegt ist

$$m = \varrho_0 \frac{V}{n} \ . \tag{6.16}$$

## 6.3. Dichte und Druck

Die grundlegenden physikalsichen Größen eines inkompressiblen, isothermalen und viskosen Fluids sind gegeben durch die Geschwindigkeit $\mathbf{u}$, die Massendichte $\varrho$ und den Druck $p$, die als

Feldgrößen gegeben sind und daher im SPH-Formalismus über eine Kernelfunktion berechnet werden.

### 6.3.1. Dichte

Bei der Massendichte handelt es sich um eine Feldgröße, die explizit berechnet werden muss. Für ein inkompressibles Fluid bleibt das Volumen erhalten. Somit ist die Dichte eine Größe, die allein durch die Masse definiert ist. Für ein Partikel $i$ ist dies

$$\begin{aligned} \varrho_i &= \varrho(r_i) \\ &= \sum_j \varrho_j \frac{m_j}{\varrho_j} W(r_i - r_j, h) \\ &= \sum_j m_j W(r_i - r_j, h) \ , \end{aligned} \tag{6.17}$$

mit $W$ als Kernfunktion und dem Partikelabstand und Glättungsradius als Übergabeparameter. Für $W$ kann die in [MDM03] beschriebene Glättungsfunktion sechsten Grades verwendet werden

$$W_{default}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} \left(h^2 - \|\mathbf{r}\|^2\right)^3 & 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \|\mathbf{r}\| > h \ . \end{cases} \tag{6.18}$$

### 6.3.2. Druck

Zur Berechung der Druckkraft ist es erforderlich, den auf ein Fluidelement wirkenden Druck $p$ zu bestimmen. Der Druck $p = \frac{\mathbf{F}_\perp}{A}$ ist definiert als die auf ein Flächenelement $A$ senkrecht wirkende Kraft $\mathbf{F}_\perp$.

Dieser kann über das ideale Gasgesetz bestimmt werden, welches auf die Zustandsgleichung der Navier-Stokes-Gleichungen hinausläuft.

$$\mathbf{p}V = nRT \tag{6.19}$$

mit $V = \frac{1}{\varrho}$ als Volumen pro Einheitsmasse, $n$ die Anzahl der Gaspartikel in Mol, $R$ die universale Gas-Konstante und $T$ die Temperatur. Für ein isothermales Fluid mit konstanter Masse kann die rechte Seite der Gleichung 6.19 zu einer Konstanten zusammengefasst und durch eine Gas-Steifigkeitskonstante $k$ ausgedrückt werden. Nach Umformen erhält man für den Druck die Gleichung

$$p = k\varrho \ . \tag{6.20}$$

Die Druckkraft, die im nachfolgenden Abschnitt beschrieben wird, führt ausschließlich zu abstoßenden Kräften, so dass sich die simulierte Flüssigkeit ohne Gravitationskraft bzw. Luftdruck ständig ausdehnen würde. Zur Simulation eines Luftdrucks kann ein Ruhedruck $p_0$ eingeführt werden, der bei Unterschreiten einer Schwelle die Vorzeichen umkehrt und für anziehende Kräfte sorgt

$$\begin{aligned}(p + p_0)V &= k \\ p + k\varrho_o &= k\varrho \\ p &= k(\varrho - \varrho_0),\end{aligned} \tag{6.21}$$

mit $\varrho_0$ als vorgegebene Materialdichte. Für Wasser z.B. $\varrho_0 = 998,29\frac{kg}{m^3}$.

## 6.4. Druckkraft

Die Druckkraft ist eine durch den Druck $p$ ausgeübte Kraft auf ein Fluidelement.

Somit ist die durch den Druck auf ein Volumen $V$ mit Oberfläche $\partial V$ und Normalenvektor $\mathbf{n}$ ausgeübte Druckkraft gegeben durch

$$-\int_{\partial V} \mathbf{p}\mathbf{n} dA = -\int_V \nabla \mathbf{p} dV \ . \tag{6.22}$$

Jedes Volumenelement $dV$ erfährt damit aufgrund der umgebenden Flüssigkeit die Druckkraft

$$\frac{dF_i}{dV} = -\nabla \mathbf{p} \ . \tag{6.23}$$

Ist die Druckkraft die einzige wirkende Kraftkomponente, so erhält man die sogenannte inkompressible Eulersche Gleichung als Grundgleichung der Thermodynamik

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\varrho}\nabla \mathbf{p} \ . \tag{6.24}$$

Wurde der Druck für jedes Partikel $i$ berechnet, kann die Druckkraft mit der Gleichung 6.24 bestimmt werden. Es ist wichtig, dass die Kräfte zueinander symmetrisch sind, um das Kräftegleichgewicht der Partikel zueinander zu erhalten. Eine symmetrische Variante, die dies im SPH-Formalismus ausdrückt ist

$$\begin{aligned}f_i^{pressure} &= -\nabla p(\mathbf{r}_i) \\ &= -\sum_{j \neq i} \frac{p_i + p_j}{2} \frac{m_j}{\varrho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \ .\end{aligned} \tag{6.25}$$

Eine weitere symmetrische Formulierung, die ebenfalls verwendet werden kann ist

$$f_i^{pressure} = -\varrho_i \sum_{j \neq i} \left( \frac{p_i}{\varrho_i^2} + \frac{p_j}{\varrho_j^2} \right) m_j \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \ . \tag{6.26}$$

Eine Anforderung an die Kernfunktion ist, dass diese zum Rand hin monoton fallend ist und im Intervall $[0, 1]$ stets positive Werte aufweist. Die in Kapitel 3.1 vorgestellte Basisfunktion kann nicht angewendet werden, da sie für Werte kleiner $\|r\|$ ebenfalls gegen Null tendiert. Dies hat zur Folge, dass sich die Rückstoßungskräfte bei eng beieinander liegenden Partikeln abmildern, was zu einem Clusterverhalten der Partikel führt. Um dies zu verhindern, wird

für die jeweiligen Kräfte eine geeignete Kernelfunktion benötigt, die die entsprechenden Bedingungen erfüllt. Für die Druckkraft ist dies z.B.

$$W_{pressure}(\mathbf{r}, h) = \frac{15}{\pi h^6} \begin{cases} (h - \|\mathbf{r}\|)^3 & 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \|\mathbf{r}\| > h \ , \end{cases} \tag{6.27}$$

mit dem Gradienten

$$\nabla W_{pressure}(\mathbf{r}, h) = -\frac{45}{\pi h^6} \frac{\mathbf{r}}{\|\mathbf{r}\|} (h - \|\mathbf{r}\|)^2 \tag{6.28}$$

$$\lim_{r \to 0^-} \nabla W_{pressure}(\mathbf{r}, h) = \frac{45}{\pi h^6} \ , \lim_{r \to 0^+} \nabla W_{pressure}(\mathbf{r}, h) = -\frac{45}{\pi h^6} \ . \tag{6.29}$$

## 6.5. Viskosität

Ein Fluid ist eine Substanz, die Scherkräften nicht standhalten kann und dementsprechend fließt. Die Moleküle in der Flüssigkeit werden einer Reibungskraft ausgesetzt, die auch als Viskosität bezeichnet wird. Dieser Prozess ist irreversibel, da durch Dissipation kinetische Energie in Wärmeenergie umgewandelt wird. Die durch die Viskosität auf ein Volumen $V$ ausgeübte Kraft ist

$$\int_{\partial V} \sigma \cdot \mathbf{n} dA = \int_V \operatorname{div} \sigma dV \ , \tag{6.30}$$

mit dem zähen Spannungstensor $(\sigma_{ij})_{i,j=1\ldots d}$.
Somit wirkt auf ein Volumenelement $dV$ die innere Kraft

$$\frac{dF_i}{dV} = \operatorname{div} \sigma \ . \tag{6.31}$$

Der Spannungstensor $\sigma$ beschreibt die Vorgänge der inneren Reibung, die auftreten, wenn Flüssigkeitselemente in Relativbewegung zueinander stehen. An diesen Tensor werden zwei Anforderungen gestellt. Zum einen soll er bei konstanter Geschwindigkeit verschwinden, zum anderen muss er von den örtlichen Ableitungen der Geschwindigkeit abhängen. Somit ist $\sigma = \sigma(\nabla u, \Delta u, \ldots)$.

Für kleine Geschwindigkeitsgradienten können die höheren Ableitungen vernachlässigt werden. Der lineare Spannungstensor ist somit

$$\sigma = \mathbf{2}\mu\varepsilon + (\zeta - \frac{2}{3}\mu)\mathrm{Id} \operatorname{div} \mathbf{u} \ , \tag{6.32}$$

mit $\mu$ und $\zeta$ als ersten und zweiten Zähigkeitskoeffizienten. Der Deformationstensor $\varepsilon = (\varepsilon_{ij})_{i,j=1\ldots d}$ ist definiert durch

$$\varepsilon_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \ . \tag{6.33}$$

Bei Annahme des Geschwindigkeitsvektorfeldes als divergenzfrei, fällt der zweite Teil von Gleichung 6.32 weg

$$\sigma_{ij} = \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \ . \tag{6.34}$$

Komponentenweise ist die spezifische innere Kraft

$$\operatorname{div} \sigma_i = \mu \sum_{j=1}^{d} \frac{\partial}{\partial x_j} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \ . \tag{6.35}$$

Der Satz von Schwarz liefert unter der Annahme der Divergenzfreiheit von $u$

$$\operatorname{div} \sigma_i = \mu \left( \sum_{j=1}^{d} \frac{\partial^2 u_i}{\partial x_j^2} + \frac{\partial}{\partial x_i} \sum_{j=1}^{d} \frac{\partial u_j}{\partial x_j} \right) = \mu \Delta u_i \tag{6.36}$$

[Dar07].

Eine symmetrische SPH-Variante der Viskositätskraft aus Gleichung 6.36 ist gegeben durch

$$\begin{aligned} f_i^{viscosity} &= \mu \nabla^2 \mathbf{u}(\mathbf{r}_i) \\ &= \mu \sum_{j \neq i} (\mathbf{u}_j - \mathbf{u}_i) \frac{m_j}{\varrho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \ , \end{aligned} \tag{6.37}$$

mit der Kernfunktion

$$W_{viscosity}(\mathbf{r}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{\|\mathbf{r}\|^3}{2h^3} + \frac{\|\mathbf{r}\|^2}{h^2} + \frac{h}{2\|\mathbf{r}\|} - 1 & 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \|\mathbf{r}\| > h \ , \end{cases} \tag{6.38}$$

$$\lim_{r \to 0} \nabla W_{viscosity}(\mathbf{r}, h) = \infty \tag{6.39}$$

und der Anwendung des Laplaceoperator $\Delta$ auf diese Funktion:

$$\nabla^2 W_{viscosity}(\mathbf{r}, h) \quad = \quad \frac{45}{\pi h^6} (h - \|\mathbf{r}\|) \ . \tag{6.40}$$

Eine symmetrische SPH-Variante der Viskositätskraft aus Gleichung 6.36, ist gegeben durch

$$\begin{aligned} f_i^{viscosity} &= \mu \nabla^2 \mathbf{u}(\mathbf{r}_i) \\ &= \mu \sum_{j \neq i} (\mathbf{u}_j - \mathbf{u}_i) \frac{m_j}{\varrho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \ , \end{aligned} \tag{6.41}$$

mit der Kernfunktion

$$W_{viscosity}(\mathbf{r},h) = \frac{15}{2\pi h^3}\begin{cases} -\frac{\|\mathbf{r}\|^3}{2h^3} + \frac{\|\mathbf{r}\|^2}{h^2} + \frac{h}{2\|\mathbf{r}\|} - 1 & 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \|\mathbf{r}\| > h \ , \end{cases} \tag{6.42}$$
$$\lim_{r\to 0} \nabla W_{viscosity}(\mathbf{r},h) = \infty$$

und der Anwendung des Laplaceoperators auf die Funktion:

$$\nabla^2 W_{viscosity}(\mathbf{r},h) \quad = \quad \frac{45}{\pi h^6}(h - \|\mathbf{r}\|) \ . \tag{6.43}$$

## 6.6. Gravitation

Die Gravitationskraft als externe Kraft führt dazu, dass sich Massen gegenseitig anziehen. Je größer dabei die Masse der entsprechenden Körper und je geringer deren Abstand ist, umso größer ist die auf diese Massen wirkende Anziehungskraft. Nach dem Gravitationsgesetz von Newton ist die auf eine Masse $m$ wirkende Kraft $F$ mit $F = mg$, wobei $g$ die Gravitationskonstante ist. Angewendet auf ein Volumenelement $dV$, mit $V = \frac{m}{\varrho}$ gegeben, ist die Gravitationskraft

$$\frac{dF_i}{dV} = \varrho_i \mathbf{g} \ , \tag{6.44}$$

mit $\mathbf{g} = (0, 9.811\frac{m}{s^2}, 0)^T$ als nach unten gerichtete, auf der Erdoberfläche wirkende Gravitationskraft. Es ist zu beachten, dass in diesen Wert die nach aussen wirkende Beschleunigung durch die Fliehkraft wertmindernd einfließt.

Das Gravitationsfeld beeinflusst alle Partikel gleich. Mit der nach unten gerichteten Gravitationsbeschleunigung $g$ berechnet sich die Gravitationskraft-Dichte für ein Partikel $i$ zu

$$\mathbf{f}_i^{gravitation} = \varrho_i \mathbf{g} \ . \tag{6.45}$$

## 6.7. Oberflächenspannung

Die Oberflächenspannung kann als externe Kraft auf den Rand des Fluids angewendet werden. Da sie nicht auf das gesamte Fluid wirkt, ist sie nicht Teil der Navier-Stokes-Gleichungen, kann aber als Randbedingung einbezogen werden. Ursache der Oberflächenspannung ist Kohäsion. Unter Kohäsion versteht man die Bindungskräfte zwischen Atomen oder Molekülen eines Stoffes. Innerhalb eines Fluids sind diese Bindungskräfte ausgeglichen, da auf ein Fluidteilchen Kräfte aus allen Richtungen wirken. An der Oberfläche jedoch wirken diese Kräfte nur entlang des Randes und in das Fluid hinein, da sich oberhalb der Flüssigkeit keine weiteren Moleküle befinden und sich die Kräfte so nicht mehr gegenseitig aufheben können. Die Folge ist, dass eine Resultierende entsteht, die in das Fluidinnere zeigt und zu dem Phänomen der Oberflächenspannung führt. Bei dem Versuch die Oberfläche zu vergrößern, müssen mehr Moleküle vom Inneren der Flüssigkeit an die Oberfläche, wozu eine gewisse Kraft nötig ist. Physikalische Systeme sind jedoch bestrebt, die energetisch günstigste Form einzunehmen. Im Falle des Fluids wird dies über eine möglichst kleine Oberfläche erreicht. Somit wirkt die Oberflächenspannung in Richtung der Oberflächennormalen in das Fluid hinein.

Die Oberflächenspannung, in Fachterminologie als spezifische Oberflächenarbeit bezeichnet, ist die spezifische Arbeit, die aufgewendet werden muss, um die Oberfläche des Fluids zu vergrößern [Uni09].

$$\sigma := \frac{dW}{dA} = \frac{\text{Arbeit zur Bildung neuer Oberfläche}}{\text{neue Oberfläche}}, \quad [\sigma] = \tfrac{N}{m} \ . \tag{6.46}$$

Ist die Ruhelage der Flüssigkeit eben, so steigt der Kohäsionsdruck für eine konvex gewölbte Oberfläche durch die nach innen gerichtete, auf die Oberflächenmoleküle wirkende Oberflächenkraft. Bei konkaven Oberflächen ist die Resultierende der Oberflächenkräfte nach aussen gerichtet. Um den Normaldruck zu berechnen, der durch die Krümmung der Oberfläche hervorgerufen wurde, wird ein gekrümmtes Oberflächenelement $dA$ betrachtet. Auf die Begrenzung dieses Elements wirken Tangentialkräfte, deren resultierende eine senkrecht zur Oberfläche in das Fluid wirkende Kraft ergibt. Diese kann über die Krümmung $\kappa$, die als Kehrwert des Krümmungsradius definiert ist, bestimmt werden. Die Oberflächenzugkraft $\tau$ ist das Produkt aus der Oberflächenspannung $\sigma$, der Krümmung $\kappa$ und zeigt in Richtung der Normalen

$$\tau = \sigma\kappa\frac{\mathbf{n}}{\|\mathbf{n}\|} \ . \tag{6.47}$$

Die Oberflächenspannung wirkt am Rand des Fluids entlang der Oberflächennormalen in das Fluid hinein

$$\mathbf{f}_i^{surface} = -\sigma\nabla^2 c_i \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|} \ , \tag{6.48}$$

wobei $\sigma$ ein materialabhängiger Spannungskoeffizient ist, der im Falle von Wasser z.B. $0.0728\frac{N}{m}$ ist. Der Vektor $\mathbf{n}_i$ ist der nach innen gerichtete Normalenvektor am Ort des Partikels $i$, der sich nahe an der Oberfläche befindet.

Der Wert $c$ ist eine Feldgröße mit dem Wert $c = 1$ an den Partikelpositionen und $c = 0$ sonst. $c_i$ ist der geglättete Wert des sogenannten Farbfeldes, ausgewertet an Partikel $i$. Mit SPH berechnet sich der Wert des Farbfeldes an Partikel $i$ zu

$$\begin{aligned} c_i &= c(\mathbf{r}_i) \\ &= \sum_j c_j \frac{m_j}{\varrho_j} W(\mathbf{r_i} - \mathbf{r_j}, h) \\ &= \sum_j \frac{m_j}{\varrho_j} W(\mathbf{r_i} - \mathbf{r_j}, h) \ . \end{aligned} \tag{6.49}$$

Die ins Fluid gerichtete Normale $\mathbf{n}_i$ ist dann der Gradient des Farbfeldes

$$\begin{aligned} \mathbf{n}_i &= \nabla c(\mathbf{r}_i) \\ &= \sum_j \frac{m_j}{\varrho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \ . \end{aligned} \tag{6.50}$$

Die Länge des Vektors $\|\mathbf{n}_i\| > 0$ nimmt mit dem Abstand zur Oberfläche ab und ist daher nur in dessen Nähe größer Null.

Die Krümmung $\kappa$ ergibt sich als Divergenz von $\mathbf{n}$

$$\kappa = -\frac{\nabla \mathbf{n}}{\|\mathbf{n}\|} = -\frac{\nabla^2 c}{\|\mathbf{n}\|} \ . \tag{6.51}$$

Das negative Vorzeichen muss gesetzt werden, um eine positive Krümmung für konvexe Fluidvolumina zu erhalten. Mit der Oberflächenzugkraft aus Gleichung 6.47 und einem Skalierungsfaktor $\delta_i = \|\mathbf{n}_i\|$, der für eine gleichmäßige Aufteilung der Kräfte sorgt, ist die Oberflächenspannung

$$\mathbf{f}_i^{surface} = \delta_i \tau_i = \sigma \kappa_i \mathbf{n}_i = -\sigma \nabla^2 c_i \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|} \ . \tag{6.52}$$

Diese Gleichung sollte nur auf Partikel in Oberflächennähe angewendet werden, da für $\|\mathbf{n}\| \to 0$ der Term $\frac{\mathbf{n}}{\|\mathbf{n}\|}$ in Gleichung 6.52 numerisch instabil wird. Daher sollte diese Gleichung nur angewendet werden, wenn $\|\mathbf{n}_i\| \geq \ell$, mit einem von der Partikeldichte abhängigen Grenzwert $\ell > 0$.

# 7. Zeitintegration

Das zu simulierende physikalische Phänomen ist kontinuierlicher Natur. Zur exakten Bestimmung der Partikelposition für einen Zeitschritt $t_n$ müssen die in Kapitel 3 beschriebenen Differentialgleichungen zweiter Ordnung gelöst werden. Eine exakte analytische Lösung der Differentialgleichungen ist nicht möglich. Stattdessen wird zur Bestimmung des Simulationszustandes zu einem bestimmten Zeitpunkt eine approximative Lösung gesucht. Dabei wird die kontinuierliche Simualtionszeit in diskrete Zeitschritte zerlegt. Ausgehend von einem Startwert, Anfangspostion der Partikel mit ihren Initialkräften, wird in jedem Simulationszeitschritt, eine approximative Lösung bestimmt, basierend auf zeitlich davorliegenden Berechnungen. Vorraussetzung für Positionsbestimmung ist, daß die Geschwindigkeiten für alle Partikel bekannt sind. Die Geschwindigkeit eines Partikels im nächsten Zeitschritt ergibt sich aus der Geschwindigkeit und der Beschleunigung im aktuellen Zeitschritt. Die Beschleunigung eines Partikels in einem Zeischritt $t_n$ wiederrum ist das Resultat der Kraftberechnung, welche in der Simulationsschleife unmittelbar vor der Zeitintegration liegt.

## 7.1. Grundlagen

Im Folgenden werden die zum Verständnis nötigen Grundlagen der numerischen Zeitintegration eingeführt. Es wird unterschieden zwischen impliziten und expliziten Integrationsverfahren. Die impliziten Verfahren haben bei der Berechnung im Zeitschritt $t_n$ als Eingabe den Zeitschritt $t_n$ selbst, was zur Lösung eines linearen Gleichungssystems führt. Explizite Verfahren basieren auf bereits berechneten Werten. Implizite Verfahren werden im Folgenden nicht betrachtet, da sie aus Effizienzgründen bei YASPHS nicht zum Einsatz kamen. Explizite Integrationsverfahren gliedern sich in Einschritt- sowie Mehrschrittverfahren. Einschrittverfahren zeichnen sich dadurch aus, dass die approximative Lösung im nächsten Zeitschritt nur von den Werten des aktuellen Zeitschritts abhängt. Dies bringt den Vorteil der hohen Adaptivität, d.h. die Schrittweite kann von Zeitschritt zu Zeitschritt verändert werden. Mehrschrittverfahren betrachten bei der Berechnung des nächsten Zeitschritts die letzten k Zeitschritte. Die Verwertung der bereits berechneten Werte bringt höhere Genauigkeit und Effizienz mit sich.
Die einzelnen numerischen Integrationsverfahren haben unterschiedliche Genauigkeitsordnung, welche in Abhängigkeit von der Schrittweite angegeben wird und die Fehlerabweichung der approximativen von der exakten Lösung repräsentiert. Wichtige Eigenschaften der numerischen Zeitintegrationsverfahren sind neben der Genauigkeit, Adaptivität, im Hinblick auf die Schrittweite und Effizienz. Bei der Wahl des geeigneten Verfahrens gilt es diese abzuwägen.
Formal lässt sich die numerische Zeitintegration als ein Anfangswertproblem auffassen, wobei sie als Eingabe gewöhnliche Differentialgleichungen hat, die nur vom Paramter $t$, der Zeit, abhängen.

### 7.1.1. Numerische Zeitintegration als Anfangswertproblem

Gegeben sei eine gewöhnliche Differentialgleichung zweiter Ordnung:

$$\ddot{x}(t) = f(t, x(t)) \quad , \quad 0 \leq t \leq b \tag{7.1}$$

sowie ein Anfangswert:

$$x(0) = c \tag{7.2}$$

Gesucht wird der Funktionswert $x(t_n)$ zum Zeitpunkt $t_n$ ausgehend vom Anfangswert und gegebener Funktion f. Zu jedem Anfangswert $c$ gibt es genau eine Lösung der Differentialgleichung. Eine exakte Angabe dieser Lösung ist lediglich in theoretischen Fällen möglich. Infolgedessen wird eine approximative Lösung angestrebt. Eine Approximation bedeutet eine Diskretisierung der kontinuierlichen Werte. Das Intervall für die Unbekannte t wird diskretisiert, wobei der Abstand zweier aufeinanderfolgender diskreter Werte durch die Schrittweite $\Delta t$ festgelegt wird. Ausgehend vom Anfangswert werden approximative Lösungen für die diskreten Werte $t_i$ berechnet. Ist der Wert $x(t_n)$ bekannt, kann die zweite Ableitung $\ddot{x}(t) = f(t, x)$ berechnet werden. Für weiterführende Literatur sei auf [AP98] verwiesen.

### 7.1.2. Genauigkeitsordnung

Die verschiedenen numerischen Integrationsverfahren unterscheiden sich hinsichtlich ihrer Genauigkeitsordnung. Diese wird angegeben als $O(\Delta t^x),\ 0 < \Delta t < 1$ , Ordnung $x \in \mathbb{N}^+$, und ist ein Maß für die Abweichung der approximativen von der exakten Lösung. Der Zeitschritt $\Delta t$ ist kleiner als 1. Folglich wird $O(\Delta t^x)$ mit steigender Ordnung kleiner, d.h. die approximative Lösung nähert sich mit steigender Ordnung der exakten Lösung an. Zu bemerken sei, dass mit höherer Ordnung der Berechnugsaufwand steigt.

### 7.1.3. Newtonsche Bewegungsgleichung

Die Berechnung der Partikelpositionen basiert auf der Newtonschen Bewegungsgleichung (7.3) , die durch Differentialgleichungen zweiter Ordnung beschrieben werden. Die Kraft entspricht dem Produkt aus Beschleunigung und Masse. Die Beschleunigung wird aufgefasst als die erste Ableitung der Geschwindigkeit nach der Zeit und als die zweite Ableitung des Ortes nach der Zeit.

$$F = m \cdot a(t) = m \cdot \ddot{x}(t) \quad , \quad a(t) = \dot{v}(t) = \ddot{x}(t) \tag{7.3}$$

## 7.2. in YASPHS verwendete Verfahren

Im Folgenden werden die in YASPHS verwendeten numerischen Zeitintegrationsverfahren vorgestellt, die sich auf explizite Einschrittverfahren beschränken. Der Grund für die Entscheidung gegen Mehrschrittverfahren ist, dass sich nur für statische Partikelanzahlen eignen und nur bedingt eine adaptive Schrittweite erlauben. Wird ein Partikel im Laufe der Simualtion neu generiert, so ist aufgrund fehlender Werte das m-Schritt-Verfahren nicht anwendbar. Für dynamisch erzeugte Partikel müsste eine Sonderbehandlung mit einem k-Schrittverfahren (k<m) erfolgen, wobei die Partikel mit unterschiedlicher Genauigkeit berechnet würden. Dieses Phänomen würde die Werte verfälschen und es ließen sich keine Aussagen über die Genauigkeitsordnung machen. Bei YASPHS bleibt die Anzahl der Partikel statisch, wenn als Kind-Objekte in der Szene ausschliesslich Particle-Groups vorkommen. Bei Verwendung von

Quellen und Senken ändert sich die Partikelanzahl dynamisch während der Simulation, sodass wir Mehrschrittverfahren ausschliessen.
In allen Fällen ist ein Startsimulationszustand gegeben mit initialen Partikelpositionen sowie auf jedes Partikel einwirkenden initialen Kräften. Nach 7.1.3 lässt sich für jedes Partikel daraus die Beschleunigung, folglich die Geschwindigkeit und schließlich die neue Position bestimmen.

### 7.2.1. Euler

Bei der Eulerintegration wird in jedem Simulationsschritt für jedes Partikel die neue Position in Abhängigkeit von der alten Position und dessen aktueller Geschwindigkeit bestimmt. Anschliessend wird die neue Geschwindigkeit in Abhängigkeit von der aktuellen Geschwindigkeit und aktuellen der Beschleunigung berechnet. Die Abbildungen 7.1, 7.2, 7.3 veranschaulichen den Ablauf des Verfahrens. Als Startwert sind die initalen Positionen der Partikel sowie die initialen auf die Partikel einwirkenden Kräfte gegeben.

Position aktualisieren:

$$\mathbf{pos}_i(t_{n+1}) = \mathbf{pos}_i(t_n) + \mathbf{vel}_i(t_n) \cdot \Delta t$$

Geschwindigkeit aktualisieren:

$$\mathbf{vel}_i(t_{n+1}) = \mathbf{vel}_i(t_n) + \mathbf{acc}_i(t_n) \cdot \Delta t$$

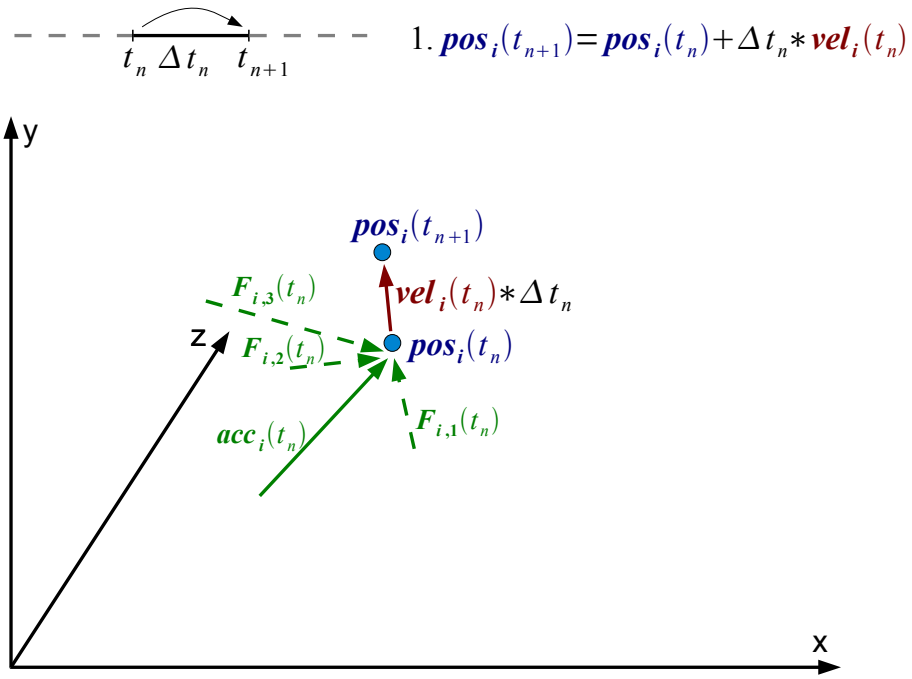



Abbildung 7.1.: Euler: Bestimmung der neuen Position

**Bewertung:** Die Genauigkeitsordnung des Eulerverfahrens beträgt $O(\Delta t)$. Der Fehler ist linear bezüglich des gewählten Zeitschritts. Das Eulerverfahren ist adaptiv, d.h. die Schrittweite

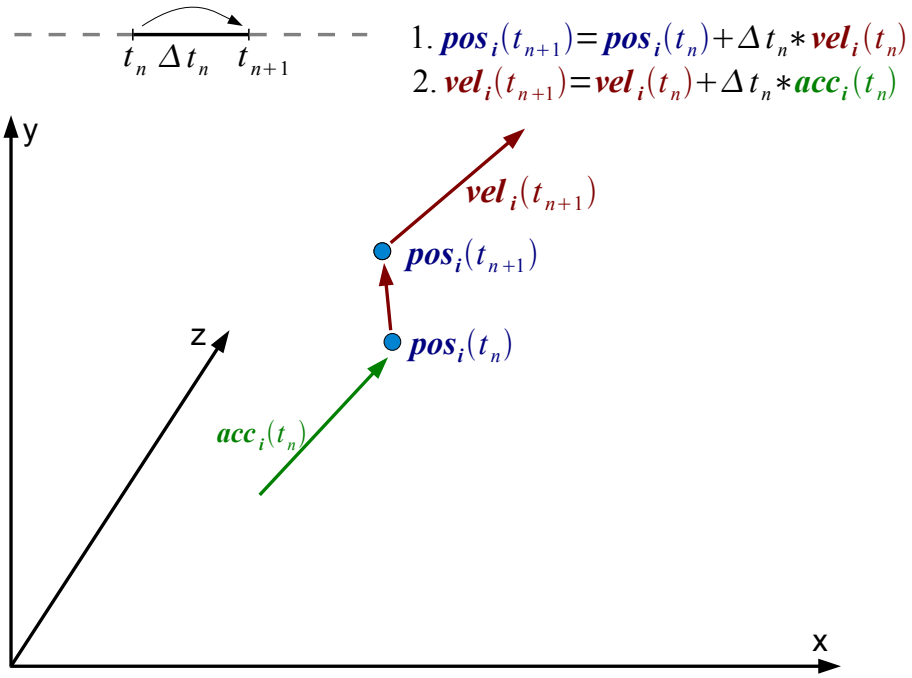



Abbildung 7.2.: Euler: Bestimmung der neuen Geschwindigkeit

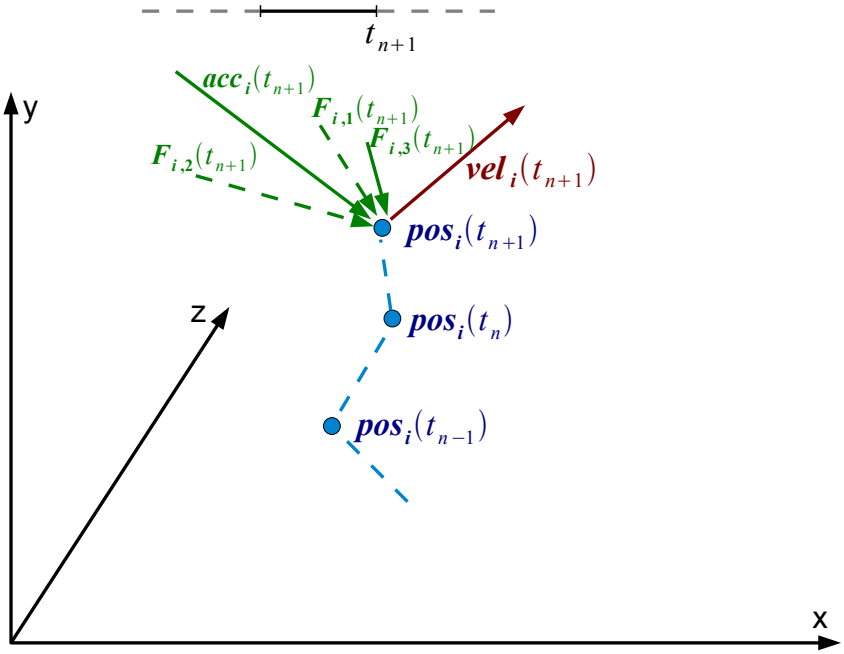



Abbildung 7.3.: Euler: Simulationszustand im Zeitschritt $t_{n+1}$

kann problemlos von Zeitschritt zu Zeitschritt varieren.

### 7.2.2. Leapfrog

Das Leapfrog-Verfahren ist eine Modifikation des Eulerverfahrens. Es berechnet die Position sowie Geschwindigkeit einen halben Zeitschritt versetzt. Als Startwerte werden neben den initialen Partikelpositionen und Kräften auch die Geschwindigkeiten im Zeitschritt $t_{1/2}$ benötigt. Diese kann z.B. mit dem Eulerverfahren ermittelt werden. Abbildung 7.4 veranschaulicht den Ablauf des Verfahrens. Als weiterführende Literatur sei [You09] genannt.

$$\mathbf{pos}_i(t_{n+1}) = \mathbf{pos}_i(t_n) + \Delta t \cdot \mathbf{vel}_i(t_{n+1/2}) \quad (Position) \tag{7.4}$$

$$\mathbf{vel}_i(t_{n+3/2}) = \mathbf{vel}_i(t_{n+1/2}) + \Delta t \cdot \mathbf{acc}_i(t_{n+1}) \quad (Geschwindigkeit) \tag{7.5}$$

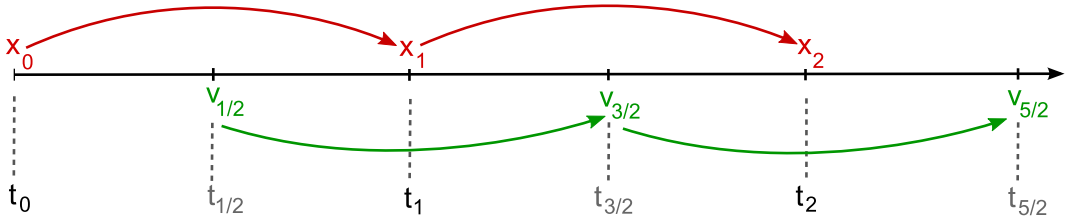



Abbildung 7.4.: Leapfrog-Verfahren, $x_i$: Position zum Zeitpunkt $t_i$, $v_i$: Geschwindigkeit zum Zeitpunkt $t_i$.

**Bewertung** Durch die einen halben Zeitschritt versetzte Berechnung der Position und Geschwindigkeit wird eine Mittelung erreicht. Die Genauigkeitsordnung beträgt $O(\Delta t^2)$. Damit liegen die mit Leapfrog berechneten Werte näher an der exakten Lösung im Vergleich zur Berechnung mit dem Eulerverfahren bei gleichbleibender Effizienz.

### 7.2.3. XSPH

XSPH ist eine Modifikation, speziell für Fluidsimulationen, und baut auf einem klassischen Integrationsverfahren auf. Das Verfahren korrigiert die Bewegung der Partikel und erhöht die Stabilität der Simulation, indem über eine Kernelfunktion die durchschnittliche Bewegung benachbarter Partikel zu der eigentlichen Geschwindigkeit des betrachteten Partikels hinzuaddiert wird. Dadurch werden Partikel geordnet und kontrolliert im Strom mitgerissen. Die XSPH-Formulierung zur Ermittlung der zusätzlichen Geschwindigkeit $\Delta v_i$ für ein Partikel $i$ lautet:

$$\Delta v_i = \epsilon \sum_j \frac{m_j(v_i - v_j)}{\frac{\rho_i + \rho_j}{2}} W(r_i - r_j, h) \;, \epsilon \in [0,1] \,. \tag{7.6}$$

Bei SPH geschieht die Berechnung nach der eigentlichen Integration mit dem Leap-Frog-Verfahren, beschrieben in 7.2.2.

# 8. Kollisionserkennung

Der Begriff „Kollisionserkennung“ bezeichnet den Vorgang, bei dem ermittelt wird, ob ein Zusammenstoss zweier geometrischer Körper im mehrdimensionalen Raum stattgefunden hat. Es handelt sich bei diesem Vorgang in dieser Arbeit um ein analytisches Verfahren, das aufgrund einer formalen Beschreibung der Körper eingesetzt wird. Diese Arbeit behandelt die Kollisionserkennung zwischen einem Partikel und einer Wandgeometrie, die ein Fluid umschließt. Der Partikel wird hierbei formal lediglich durch seine Raumkoordinaten dargestellt, da es sich bei ihm nicht um einen Körper mit Fläche bzw. Volumen handelt, sondern um eine Messstelle, an der Messungen der Materialeigenschaften des untersuchten Fluids vorgenommen. Aus diesem Grund wird in dieser Arbeit eine Kollision zweier Partikel nicht berücksichtigt. Sie führt lediglich dazu, dass sie vorübergehend dieselben Raumkoordinaten und dieselben Materialeigenschaften des Fluids an dieser Stelle im Raum anzeigen. Ein Zusammenstoss der Messstelle mit der Wand ist trotz den zuvor genannten Zusammenhängen möglich, da sie keine Kollision im eigentlichen Sinne darstellt, sondern lediglich eine Einschränkung des Analysebereichs sowie die Erhaltung der gemessenen Materialeigenschaften des Fluids bezeichnet.
Der Begriff „Kollisionsbehandlung“ bezeichnet die Reaktion auf einen Zusammenstoss zweier Körper, d.h. das Verfahren, welches im Falle einer erkannten Kollision eingesetzt wird. Da es sich bei YASPHS um ein Softwaretool zur Simulation des Fluidverhaltens insbesondere in abgeschlossenen Bereichen handelt, womit die Bewegung des Fluids zusammen mit den Messstellen innerhalb der Körperwand gemeint ist, wird ein mit der Körperwand zusammenstoßender Partikel in diesen Bereich zurückgestoßen.

## 8.1. Primitive Körper

Zur Ermittlung einer Kollision eines Partikels mit einem primitiven Körper wird in einem ersten Schritt die Partikelposition mit der impliziten formalen Darstellung des Körpers verglichen. Hierfür wurden in Rahmen des Projekts YASPHS die primitiven Körper Sphäre und Würfel implementiert. Diese werden wie folgt formal dargestellt:

$$F(x)_{Sphäre} = |x-c|^2 - r^2 \tag{8.1}$$

$$F(x)_{Würfel} = [|x| - ext]_{max} \tag{8.2}$$

Hierbei wird die Partikelposition durch x und der Mittelpunkt der Sphäre durch c bezeichnet. Die Variable *ext* steht für den Abstand der Würfelwände vom Mittelpunkt. Der Operator $[.]_{max}$ gibt den Vektorkomponent mit dem grössten Betrag aus. In Abhängigkeit davon, ob der Körper als Container, d.h. als umschließendes Objekt, oder als Hindernis definiert wird, müssen die Kollisionserkennung und -behandlung so gestaltet sein, dass die Partikel sich ausschließlich innerhalb bzw. außerhalb der Körperwand bewegen. In dieser Arbeit werden die primitiven Körper als Container definiert. Es werden drei mögliche Ausgänge der Kollision eines

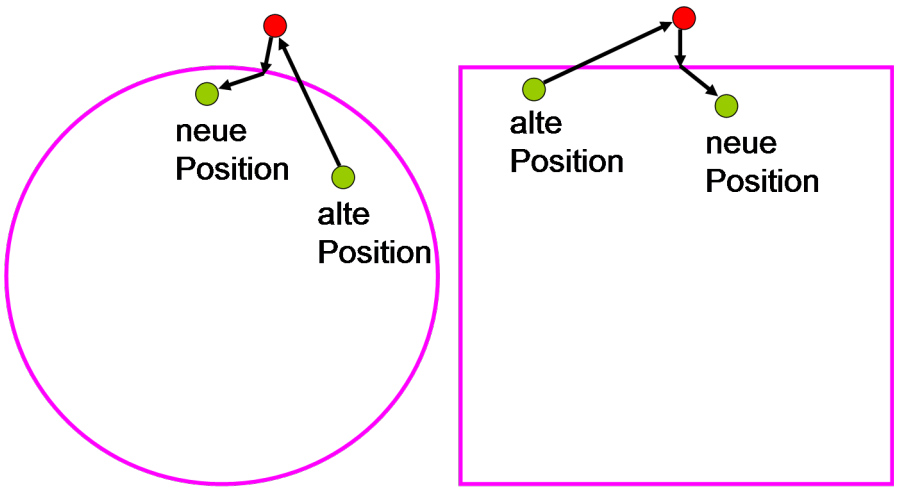



Abbildung 8.1.: Kollisionserkennung bei geometrischen Körpern.

Partikels mit einer Körperwand unterschieden. Im ersten Fall liegt die Partikelposition auf der Wand des Körpers. In diesem Fall hat eine Kollision zwischen dem Körper und dem Partikel stattgefunden und der Partikel wird von der Körperwand reflektiert. Im zweiten Fall befindet sich der Partikel innerhalb des Körpers. Dies impliziert, dass keine Kollision stattgefunden hat. Es wird hierbei keine Kollisionsbehandlung durchgeführt. Schließlich befindet sich der Partikel im letzten Fall außerhalb des Körpers, was bedeutet, dass eine Kollision bereits stattgefunden hat, diese Kollision allerdings nicht rechtzeitig erkannt wurde. Da es sich bei diesem Verfahren um ein rückwirkendes Verfahren handelt, d.h. eine Kollision wird nicht vorhergesehen, sondern erst erkannt, wenn sie stattgefunden hat, kann dieser Fall eintreten. Dazu sei erwähnt, dass es sich in diesem Fall um einen unerwünschten Effekt handelt, welcher von der Wanddicke und der Zeitschrittgröße abhängt. Bei einer Wanddicke, die größer oder gleich der von der Zeitschrittsgröße festgelegten Bewegungsdistanz der Partikel ist, kann dieser Effekt nicht auftreten, da der Partikel nicht weit genug transportiert werden kann, um die Wanddicke zu überschreiten. Ansonsten ist das Auftreten dieses Effektes möglich.
In YASPHS haben die Wände eine minimale Dicke. Wenn jedoch der Zeitschritt hinreichend klein eingestellt wird, befinden sich die Partikel beim Auftreten dieses Effektes nah an der Wand. Der Partikel wird hierbei entlang der Normale zurück auf die Wand gesetzt und von dort aus reflektiert. Der Betrag des Reflexionsvektors entspricht dem Betrag des Richtungsvektors. Da der Partikel sich vor der Kollisionsbehandlung nah an der Wand befunden hat, weicht der so entstandene Reflexionsvektor von demjenigen, der aus dem eigentlichen Berührungspunkt des Partikels mit der Körperwand entstanden wäre, nur unerheblich ab. Bei einem grossen Zeitschritt wird diese Abweichung allerdings zur Ungenauigkeiten bei der Berechnung und der Ausgabe der Materialeigenschaften des Fluids führen.

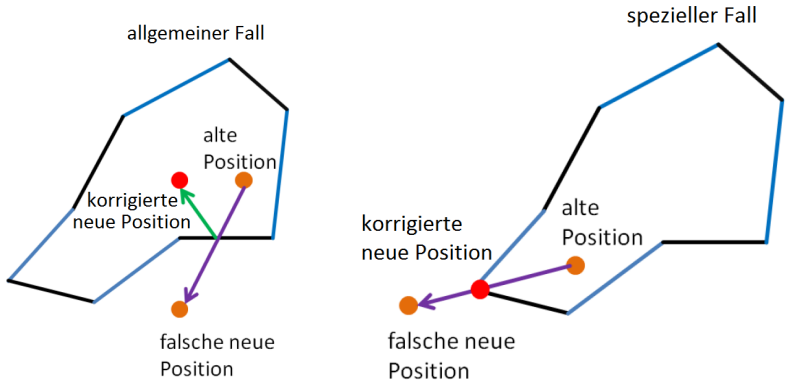
## 8.2. Meshes

Auch die Kollisionserkennung von Partikeln mit Meshes besteht aus der Erkennung, ob eine Kollision stattgefunden hat und der Kollisionsbehandlung. Für die Erkennung, ob eine Kollision stattgefunden hat, muss ein Kollisionstest zwischen einer Strecke und den Dreiecken, aus denen

die Meshes bestehen, möglichst effizient durchgeführt werden. Die Strecke wird hierbei aus der alten und neuen Partikelposition, berechnet in der Zeitintegration, gebildet. Diese Berechnung wird in YASPHS mit Hilfe der Softwarebibliotheken „Coldet“ und „Bullet“ durchgeführt (siehe Kapitel 9.2.2) und liefert die für die Kollisionsbehandlung nötigen Informationen. Dazu gehören die Koordinaten der Ecken des an einer Kollision beteiligten Dreiecks, die Normale des Dreiecks und der eigentliche Kollisionspunkt. Die Kollisionsbehandlung, die eine Approximation des natürlichen physikalischen Verhaltens darstellt, wird anschließend ausgeführt, wenn eine Kollision stattgefunden hat. Um eine möglichst realistische Kollisionsbehandlung zu erreichen, müssen bei einer vorhandenen Kollision zwei Fälle unterschieden werden. Im ersten, dem allgemeinen Fall, wird der normierte Reflektionsvektor am Kollisionspunkt bezüglich des Vektors von der alten zu der neuen Partikelposition berechnet. Anschließend wird die Länge dieses Vektors aus der Länge der Strecke zwischen der alten und der neuen Partikelposition und der eingestellten Dämpfung des Materials bestimmt. Schließlich wird mit diesem Vektor und dem Kollisionspunkt eine korrigierte neue Partikelposition bestimmt. Diese Vorgehensweise wird maximal dreimal rekursiv wiederholt, falls die Strecke zwischen der alten und der neuen korrigierten Partikelposition wieder eine Kollision mit einem der Dreiecke des Meshes verursacht. Ist jedoch keine weitere Kollision vorhanden, wird die korrigierte Partikelposition zur aktuellen Position. Bei einer Überschreitung der Rekursionstiefe von drei Ebenen dieser Berechnung wird davon ausgegangen, dass der zweite, spezielle Fall eingetroffen ist und sich das Partikel in einer spitzen Ecke befindet, in der es erwartungsgemäß zu einer Folge von mehreren Kollisionen kommt. Um die Laufzeit nicht zu stark zu vergrößern (siehe Kapitel 12.3.2) und die spitze Ecke vollständig mit Partikeln ausfüllen zu können, wird in diesem Fall die neue korrigierte Partikelposition auf den Kollisionspunkt verschoben, um einen Korrekturvektor der Länge $\epsilon > 0$, gesetzt. Der Korrekturvektor $\epsilon$ soll verhindern, dass sich das Partikel in der Kollisionserkennung im nächsten Simulationsschritt so nah an einem Dreieck aufhalten kann, dass auf Grund eines Rundungsfehlers in der Erkennung einer möglichen Kollision ein falsches Ergebnis entstehen kann (siehe Abbildung 5.1).

## 8.3. Elastische-/inelastische Kollisionsbehandlung

Bei einer elastischen Kollision zweier Körper bleibt die kinetische Energie dieser Körper nach dem Zusammenstoss erhalten. Aufgrund des Energieaustausches zwischen den Fluidmolekülen, ist eine Kollision im Falle von Flüssigkeiten selten elastisch. Dieser Umstand sollte bei der Kollisionsbehandlung berücksichtigt werden, um eine hohe Genauigkeit der Flüssigkeitssimulation zu erzielen.
Die Elastizität der Kollision wird in YASPHS durch den Parameter *damp* gesteuert. Dieser Parameter bezeichnet die Stosszahl einer Kollision und kann Werte im Bereich zwischen 0 und 1 annehmen. Hierbei bezeichnet ein Wert von 0 eine vollkommen unelastische und ein Wert von 1 eine vollkommen elastische Kollision. Da dieser Parameter in YASPHS als float deklariert wurde, lassen sich hierdurch auch Kollisionen mit einem Elastizitätsgrad gestalten, der zwischen diesen beiden Extremwerten liegt.




(a) Darstellung des allgemeinen Falls der Kollisionserkennung bei Meshes im zweidimensionalen Raum, mit eingezeichneter Strecke (violett) zwischen der alten und neuen Partikelposition (orange), berechnetem Refektionsvektor (grün) und korrigierter neuer Partikelposition (rot).

(b) Darstellung des speziellen Falls der Kollisionserkennung bei Meshes im zweidimensionalen Raum, mit eingezeichneter Strecke (violett) zwischen der alten und neuen Partikelposition (orange) und korrigierter neuer Partikelposition (rot).

Abbildung 8.2.: Verdeutlichung der Unterschiede des speziellen und allgemeinen Falls der Kollisionserkennung bei Meshes mit Beispielen im zweidimensionalen Raum.

# 9. Objektorientierte Realisierung

Die Entwicklungsziele eines SPH-Simulators mit Baukastencharakter, der für viele verschiedenartige Simulationen herangezogen werden kann, stellen besondere Anforderungen an den Entwurf des Programmcodes und die Auswahl des Programmierparadigmas. Ein Programm mit diesen Anforderungen muss modular aufgebaut sein und mit wenig Arbeit um neue Bausteine erweiterbar sein. Die objektorientierte Programmierung bettet diese geforderten Eigenschaften direkt über Abstraktion und Kapselung in ihr Paradigma ein und bietet damit eine optimale Unterstützung für das Entwickeln in Bausteinen und Programmieren gegen Interfaces. Hinzukommt, dass das Programm fast ausschließlich auf ebenfalls objektorientierte Bibliotheken zurückgreift, womit sich der nötige Code zur Anbindung dieser Bibliotheken gering halten lässt und der gesamte Code nur durch ein Paradigma geprägt ist.

Während der Entwicklung hat sich besonders die Datenkapselung als sehr hilfreiche Eigenschaft der objektorientierten Programmierung erwiesen. Da ein Feature oft vollständig von einer Klasse implementiert wird, war es in einem Entwicklungszyklus sehr einfach möglich die Interna einer Klasse zu überarbeiten, ohne dass die Arbeiten an anderen Modulen davon betroffen waren.

Die Eingrenzung der einzelnen Module mit ihren dedizierten Aufgaben, sowie die Bestimmung der Interfaces für die Kommunikation der Module wurden weitestgehend aus dem SPH-Formalismus (siehe Abbildung 3.13) abgeleitet. Der Formalismus beschreibt eine sequenzielle mehrstufige Verarbeitungspipeline, dabei sind die Interfaces zwischen den Pipelinekomponenten über die zu transportierenden Daten klar definiert. Die, so auf ein einheitliches Interface zurückgeführten, Komponenten erlauben eine einfache Veränderung oder Austausch der dahinterliegenden Implementierung.

Die zwei Module, die zweifelsohne am meisten von einem objektorientierten Ansatz profitiert haben, sind die Szeneverwaltung und die Kraftberechnung. Die Szeneverwaltung verfolgt den klassischen Erbungsansatz bei dem ein minimales Interface an der Spitze der Hierarchie durch Erbung schrittweise spezialisiert und erweitert wird. Die eigentlichen Features werden dann in untersten Stufe der Erbungspyramide implementiert. Ein artverwandter Ansatz wurde bei der Kraftberechnung verfolgt, auch hier gibt es ein einheitliches Interface, das sich direkt aus den mathematischen Formeln ableiten lässt. Anders als bei der Szeneverwaltung bleibt die Hierarchie hier allerdings flach, da die Kräfte, aus konzeptioneller Sicht, nicht aufeinander Aufbauen.

## 9.1. Klassendiagramme

Ein Klassendiagramm über alle Klassen von YASPHS wäre weder übersichtlich, noch von hohem Informationsgehalt geprägt, deshalb werden stattdessen nur die Diagramme einzelner Komponenten und die jeweils dem Entwurf zugrunde liegenden Ideen vorgestellt.

### 9.1.1. Szene

Die Modellierung der Szene muss die Vorgaben (siehe Kapitel 4) möglichst präzise wiederspiegeln und bei der weiteren Entwicklung unterstützend wirken. Grundsätzlich verfügt jedes Element der Szene über einen minimalen Satz an Attributen und Methoden, daher wurde diese Funktionalität im abstrakten Typ „Objekt“ zusammengefasst (siehe Abbildung 9.1). Die exklusive Zuordnung eines Szenenelementes als Vater- oder Kindobjekt ist vom Typ abhängig und führt zu zwei Kindklassen von Objekt die als Basis für die eigentlichen Szenenelemente dient. Die Zuordnung der Position in der Hierarchie geschieht damit implizit bereits über die Typen und nicht erst zur Laufzeit.

### 9.1.2. Kräfte

Die Kräfte wurden von Anfang an als Plugins entworfen, um später einer reibungslose Erweiterung des Basisprogrammes zu ermöglichen. Eine Pluginstruktur, die ein nachträgliches einladen weiterer Plugins erlaubt, stellt besondere Ansprüche an das Klassendesign. Alle Aufgaben, die Wissen über die Struktur oder Metadaten des Plugins erfordern, müssen in die Pluginklasse selber integriert werden (siehe Abbildung 9.2). Das Plugin selber erscheint für das Programm damit als eine vollständige Einheit die keine Daten von außen benötigt. Folglich erzeugen die Plugins die GUI für ihre Parameter selbständig und geben die erstellten GUI-Elemente an das Basisprogramm zur Integration in einen Dialog weiter. Entsprechend werden GUI-Events auch direkt an die Plugins weitergeleitet. Das Serialisieren von Einstellungsparametern wird ebenfalls an die Plugins delegiert.
Die restlichen Komponenten im Programm verfolgen keine klar objektorientierte Struktur: Die Nachbarschaftssuche ist eher prozedural geprägt und als Zusammenfassung von Algorithmen nur rudimentär objektorientiert entworfen. Der GUI-Code wurde zu einem großen Teil automatisch über einen GUI-Designer erstellt oder bettet sich, üblicherweise zwecks Eventbehandlung, direkt in diese Vorlagen ein. Die Kollisionserkennung stützt sich zu großen Teilen auf die verwendeten Bibliotheken (siehe Kapitel 9.2) die das Design der selbsterstellten Klassen diktieren.

## 9.2. Tools und Bibliotheken

### 9.2.1. OpenMP

OpenMP ermöglicht die Entwickelung von portablen sowie parallelen C++ und Fortranprogrammen durch Erweiterung der Basissprache. OpenMP selbst ist dabei lediglich eine Spezifikation die eine Reihe von zusätzlichen Schlüsselwörtern in die Sprachen einfügt und deren Semantik definiert, die eigentliche Implementierung dieser neuen Sprachfeatures ist Compiler- und Betriebssystem abhängig. Diese Herangehenweise ist vergleichbar mit OpenGL oder der Standard C-Library. Mit OpenMP werden keine Threads direkt angesprochen oder synchronisiert, vielmehr wird der Code, insbesondere Schleifen, annotiert und als zu parallelisieren markiert. Die Annotation erlaubt dabei eine feingrannulare Bestimmung des Schleifenverhaltens, der Variablenverwaltung und Synchronisation. Dem Programmierer wird daher mit OpenMP nicht direkt die Parallelisierung eines Algorithmus abgenommen, die direkte Programmierung mit Threads wird lediglich von der Betriebssystemebene auf die Sprachebene abstrahiert.


Abbildung 9.1.: UML-Klassendiagramm der Szene. Die Partitionierung in Vater- und Kindobjekte wird bereits in der Erbungshierarchie vorweggenommen.

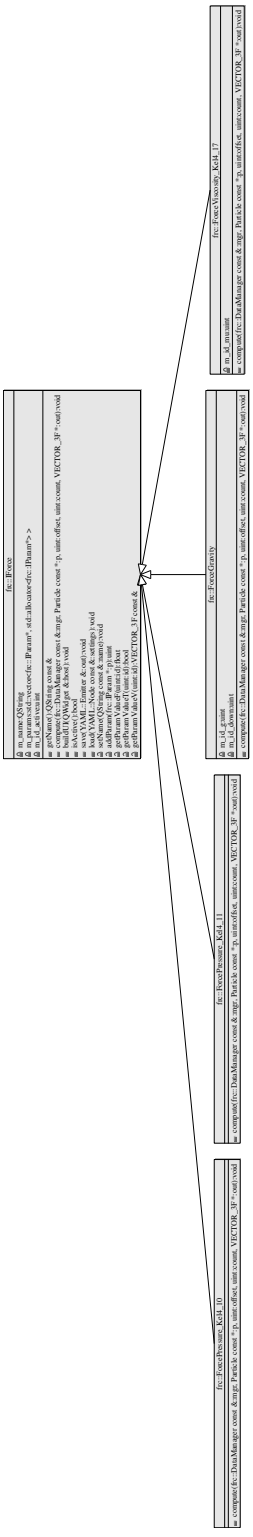



Abbildung 9.2.: UML-Klassendiagramm der Kraftberechnung (Auszug). Alle Kraftplugins implementieren ein gemeinsames Interface und verwalten alle eigenen Daten, wie Parameter, selbst.

Da sich eine manuelle Parallelisierung der einzelnen Schritte im SPH-Formalismus als zu aufwändig und fehleranfällig erwiesen hat, wurde eine einfache Parallelisierung auf Schleifenbasis mit OpenMP erprobt. Es zeigte sich jedoch schnell, dass sich die OpenMP-Implementierungen von Linux/gcc und Windows/VisualStudio an vielen Stellen nicht identisch verhalten und es folglich auf den jeweiligen Platformen zu einem leicht anderem Programmverhalten kommt. Selbst sehr konservative und äußerst restriktive Parallelisierungsversuche konnten auf beiden Plattformen nicht stabil vereint werden, weshalb von einer Parallelisierung mit OpenMP abgesehen wurde. Als weiteres Manko, das gegen eine Verwendung von OpenMP spricht, hat sich das Debugging herausgestellt. Der von OpenMP generierte Code kann im Debugger weder angesprungen noch angezeigt werden, die genaue Ausführungsreihenfolge der Befehle sowie der Inhalt von Variablen kann ebenfalls nur sehr selten verständlich nachvollzogen werden.

### 9.2.2. Coldet und Bullet

Eine Bibliothek für die Erkennung der Kollisionen von Partikeln (siehe Kapitel 8.2) mit Meshes muss einerseits Meshes und andererseits Strecken (bzw. Geradenabschnitte) unterstützen. Die Strecken ergeben sich durch die alten und neuen Partikelpositionen, welche durch die Zeitintegration in jedem Schritt bestimmt werden. Eine solche Strecke kann durch zwei Punkte im Raum oder durch einen Stütz- und einen Richtungsvektor definiert werden, wobei beide Möglichkeiten direkt ineinander umgerechnet werden können. Meshes hingegen werden durch Polygone, in YASPHS durch Dreiecke, definiert. Hinzu kommt, dass für die eigentliche Kollisionsberechnung die Normale des an einer Kollision beteiligten Dreiecks, sowie der Kollisionspunkt selbst benötigt wird. Damit eine Softwarebibliothek als geeignet eingestuft werden kann, ist außerdem die Integration einer effizienten Suchstruktur zur Beschleunigung der Berechnung möglicher Kollisionen eine wesentliche Voraussetzung. Als geeignete Softwarebibliothek wurde zunächst „Coldet“ ausgewählt. Später wurde zusätzlich „Bullet“ verwendet, da Berechnungsfehler, die später auf die endliche Genauigkeit der Gleitkommazahlen zurückgeführt werden konnten, zunächst „Coldet“ angelastet wurden. Außerdem wurde eine zweite Softwarebibliothek zum Vergleich der Effizienz nötig. Für die eigentliche Berechnung besitzen beide Bibliotheken einen Initialisierungsmechanismus, in dem die Koordinaten der Eckpunkte der Dreiecke des Meshes angegeben werden müssen und der Aufbau der Suchdatenstruktur erfolgt. Hiernach werden die Strecken angeben und jeweils eine mögliche Kollision mit dem Mesh berechnet. Hierfür ist keine weitere Initialisierung nötig. Obwohl der allgemeine Aufbau und der Ablauf einer Kollisionserkennung ähnlich sind, ergeben sich im Detail einige geringe Unterschiede.

„Coldet“ ist eine kleine und überschaubare Softwarebibliothek, die fast alle genannten Voraussetzungen erfüllt. Strecken werden als Stütz- und Richtungsvektor angegeben und Meshes können als Teilmodelle eines „Collision - Models“ als Dreiecksnetz angegeben werden. Durch die Möglichkeit Teilmodelle zu bilden und die Verwendung von geschachtelten „Boundig-Boxen“ ist eine geeignete Suchstruktur zur Beschleunigung der Kollisionsdetektion vorhanden. Jedoch ist keine direkte Unterstützung der Ausgabe der Normalen für das an der Kollison beteiligte Dreieck vorhanden. Diese kann jedoch mit Hilfe der Rückgabe der Koordinaten der Eckpunkte des Dreiecks in YASPHS berechnet werden.

„Bullet“ ist eine sehr umfangreiche und modular aufgebaute Softwarebibliothek, die alle Vorraussetzungen erfüllt. Strecken werden durch zwei Punkte im Raum definiert und Meshes können modular als „Kollisionsobjekte“, die Teilobjekte einer „Kollisionswelt“ sind, angegeben werden. Zur Beschleunigung der Kollisionserkennung werden ebenfalls geschachtelte „Bounding-

Boxen“. Die Rückgabe der Ergebnisse der jeweiligen Berechnung erfolgt in einem „Result-Callback“, einer Datenstruktur, die neben dem an einer möglichen Kollision beteiligten Dreieck und dem Kollisionspunkt auch gleich die Normale des Dreiecks mitliefert, wodurch eine zusätzliche Berechnung in YASPHS unnötig ist.
„Coldet“ hat sich als zu ineffizient erwiesen. Jedoch ist „Coldet“ nicht für Fehlberechnungen verantwortlich, die sich als Durchdringung von Partikeln durch Dreiecke bemerkbar gemacht haben. Diese konnten auf eine zu kleine Skalierung der Szene und die damit verbundenen Ungenauigkeiten der Gleitkommazahlen zurückgeführt werden. „Bullet“ hat sich als wesentlich effizienter, aber auch umständlicher in der Verwendung erwiesen. Dies kann jedoch auf die lange Entwicklungsphase und den größeren Umfang dieser Bibliothek zurückgeführt werden.

### 9.2.3. Yafaray

Yafaray [Yaf] ist ein open-source Raytracer, der sich auf die Implementierung von Bildsynthesealgorithmen zur Berechnung von realistischen Szenen spezialisiert hat. Zu den wichtigsten Features gehören u.a.:

- Global Illumination
- Material- und Shadersystem
- Antialiasing
- Caustics
- Volumetrische Effekte

Der Raytracer besticht außerdem durch seine inuitive, XML-basierte Szenebeschreibung und eine sehr gute Verarbeitungsgeschwindigkeit. Yafaray wird in der Anwendungsvisualisierung (siehe Kapitel 11) benutzt um Standbilder und Animationen aus Simulationsddaten zu berechnen.

### 9.2.4. Maya

Maya [May] ist eine vielseitige 3D-Modellierungs- und Animationssoftware mit hoher Marktdurchdringung des Softwareherstellers Autodesk Inc. Übliche Anwendungsszenarien für Maya sind das Erstellen von Computereffekten- und grafiken für Spielfilme, Werbeproduktionen oder Industrie. Das Programm beherrscht eine Vielzahl von Modellierungs- und Renderingtechniken wurde im Rahmen der PG als eine mögliche Endstufe in der Anwendungsvisualisierung genutzt (siehe 11.1).

### 9.2.5. Blender

Blender ist eine open-source 3D- und 2D-Grafikanwendung mit Fokus auf Modellierung und Animation von Polygonmeshes. Die Anwendung besitzt viele Erweiterungsmodule für zusätzliche Features wie Videoschnitt, Benutzerinteaktion, Shaderentwicklung usw.. Während der PG lag Blender in einer stabilen 2.4 Version vor, die jedoch nicht die benötigten Features zur Volumenvisualisierung anbietet (siehe 11.2) und in der sehr instabilen Entwicklungsversion 2.5. Die Entwicklungsversion ist leider für den Produktiveinsatz noch zu instabil, lässt aber das Potenzial der fertigen Version erahnen.

### 9.2.6. Qt

Qt ermöglicht eine objektorientierte, plattformunabhängige Programmierung von graphischen Oberflächen für Softwareanwendungen. Die beiden wichtigsten Konzepte hierbei sind Widgets und die Kommunikation mit dem Signal und Slot Mechanismus. Widgets sind Interaktionselemente von graphischen Oberflächen, z.B. Push-Buttons, Checkboxes. Diese können eigenständig angezeigt werden oder untergeordnet innerhalb eines anderen Widgets vorkommen. Die Qt-Bibliothek bietet vorgefertigte Widgets, der Benutzer hat aber auch die Möglichkeit eigene Widgets zu entwerfen. Der Signal und Slot Mechanismus erlaubt dem Programmierer die Aktion und Reaktion Beziehungen zu definieren, z.B. was passieren soll, wenn ein Push-Button betätigt wird.
Um dem Programmierer die Arbeit zu erleichtern, gibt es den Qt-Designer. Diese Entwicklungsumgebung ermöglicht das Erzeugen von graphischen Oberflächen mit Hilfe des Drag-and-Drop-Mechanismus. Der nötige Code zum Verwalten und Anzeigen der Widgets wird hierbei vom Qt-Designer automatisch erzeugt. Es ist auch möglich die Verknüpfung von Slots und Signals auf diese Art und Weise intuitiv zu gestalten.

### 9.2.7. VTK

Das Vizualisation Tool Kit (VTK) ist ein Open-Source-Softwaresystem für dreidimensionale Computergraphik, Bildverarbeitung und Visualisierung. Es ist eine Ebene oberhalb bekannter Rendering-Bibiliotheken einzuordnen und setzt direkt auf OpenGL auf. VTK ist objektorientiert und beinhaltet eine umfangreiche Sammlung Klassen. Der Schwerpunkt liegt hierbei auf der Visualisierung wissenschaftlicher Daten. So wurde VTK 1993 ursprünglich im Bereich der Medizin mit besonderem Augenmerk auf Volumendatenvisualisierung entwickelt. Heutzutage besteht VTK aus insgesamt über 700 Klassen, die eine Viehlzahl an Einsatzgebieten abdecken. So stehen auch komplexe (und teilweise patentierte) Visualierungsalgorithmen wie beispielsweise Marching Cubes, Delaunay-Triangulierung oder Polygonreduktion zur Verfügung.
Die Systemarchitektur besteht aus einem in C++ kompilierten Kern (compiled core) und einer umliegenden Interpreter-Schicht (interpreted Layer) mit Anbindungen an die Programmiersprachen Tcl, Python und Java (siehe 9.3). Diese sogenannten Wrapper erlauben eine flexible und komfortable Programmierung und ermöglichen eine plattformunabhängige Realisierung. Der Kern hingegen umfasst die Implementierung der eigentlichen Datenstrukturen und Algorithmen und bringt Vorteile bezüglich Effektivität und Geschwindigkeit mit sich. Neben der technischen Umsetzung durch die beiden vorgestellten Schichten beinhaltet VTK zudem zwei abstrakte Konzepte, dem graphischen Modell und der Visualisierungs-Pipeline, die im Folgenden erläutert werden.

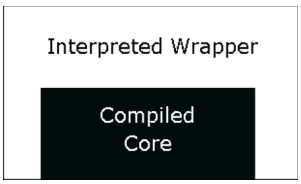



Abbildung 9.3.: Systemarchitekur

#### 9.2.7.1. Das graphische Modell

Das graphische Modell abstrahiert die hardwarenahe Realisierung der eigentlichen Rendering-Bibliothek und ist für die Verwaltung und Darstellung der Szene verantwortlich. Es sorgt für die graphische Ausgabe der von der Visualisierungs-Pipeline bereitgestellten Daten. Durch den erhöhten Abstraktionsgrad ist für den Benutzer eine schnelle und einfache Szenenbeschreibung ohne Kenntnisse der technischen Details möglich. VTK setzt dabei auf ein Klassengerüst, das namentlich an Begriffe der Filmindustrie angelehnt ist und somit einen intuitiven Zugang erlaubt. So gibt es Darsteller (sogenannte Actors), Kameras, Lichter und Requisisten. Für den vollständigen Rendervorgang einer Szene werden hierbei sieben Basisklassen benötigt:

1. vtkRenderWindow stellt das Ausgabefenster dar und enthält einen oder mehrere Renderer und einen optionalen vtkRenderWindowInteractor, der Interaktionsmöglichkeiten für den Anwender bereitstellt (z.B. Reaktion auf Maus- und Keyboardeingaben).

2. vtkRenderer koordiniert den Rendering-Prozess unter Berücksichtigung sämtlicher enthaltenen Szene-Komponenten. Hierzu zählen alle Actors, Kameras, Lichtquellen und die Hintergrundfarbe.

3. vktCamera repräsentiert die Kamera innerhalb der Szene und besitzt Eigenschaften wie Position, Blickwinkel und Brennweite.

4. vtkLight definiert eine Lichtquelle, die innerhalb der Szene beliebig positioniert werden kann. VTK benutzt als Beleuchtungsmodell das Phong-Modell und jede Lichtquelle besitzt somit einen ambienten, diffusen und spekularen Anteil. Die Anzahl der Lichtquellen ist im Gegensatz zu OpenGL nicht begrenzt.

5. vtkActor stellt ein beliebiges graphisches Objekt der Szene dar. Neben der Position können Transformationen mit einem vtkActor-Objekt verknüpft werden, um es im Raum beispielsweise zu verschieben oder zu rotieren. Zudem enthält jedes vtkActor-Objekt eine Referenz zu einem vtkProperty-Objekt.

6. vtkProperty beschreibt die Erscheinungseigenschaften wie die Farbe oder Transparenz der sichtbaren Objekte und legt zudem die Materialeigenschaften für das Beleuchtungsmodell fest.

7. vtkMapper erzeugt für die Eingabedaten geometrische Primitive, die vom Renderer verarbeitet werden können. Die Klasse dient als Schnittstelle zur Visualisierungs-Pipeline und ist offiziell Bestandteil beider Modelle.

#### 9.2.7.2. Visualisierungspipeline

Die Visualisierungspipeline ist für die Erzeugung und Aufbereitung der Daten zuständig und übergibt diese anschließend an das graphische Modell. Die Pipeline unterscheidet zwischen Daten- und Prozessobjekten. Die Datenobjekte beschreiben die Datenstrukturen, die von den Prozessobjekten verarbeitet werden können. VTK bietet verschiedene Datenobjekt-Typen an, die sich in ihrer Topologie stark unterscheiden und mittels Zellen definiert werden. Dazu zählen unter anderem polygonale Daten, Bilddaten oder strukturierte Gitter. Neben der Geometrie und Topologie kann jedes Datenobjekt weitere Eigenschaften wie Normalen, Skalar-, Vektor- oder Tensordaten besitzen.

Die Prozessobjekte werden wiederum in drei Bereiche unterteilt. Source-Objekte stellen die Quellen der Pipeline dar. Datenobjekte werden hier entweder prozedural erzeugt oder mittels Importer bzw. Reader-Klassen aus Dateien oder anderen Programmen eingelesen. Daneben dienen Filter-Objekte der Manipulation und Weiterverarbeitung von Datenobjekten. Sie überführen gegebene Inputs in entsprechende Outputs und besitzen je nach Einsatzgebiet eine Vielzahl an einstellbaren Parametern. Es lassen sich dabei beliebig viele Filter aneinanderreihen, solange die Datenobjekt-Inputs und -Outputs kompatibel sind (siehe Abbildung 9.4). Abschließend bilden die schon im graphischen Modell beschriebenen Mapper die Senken der Visualisierungs-Pipeline und die Schnittstelle zur graphischen Ausgabe. Neben der Erzeugung graphischer Primitive gibt es analog zu den Source-Objekten die Möglichkeit, die Daten direkt in Dateien zu schreiben oder an externe Anwendungen weiterzureichen. Das Zusammenspiel zwischem graphischen Modell und Visualisierungspipeline wird in Abbildung 9.5 schematisch dargestellt.

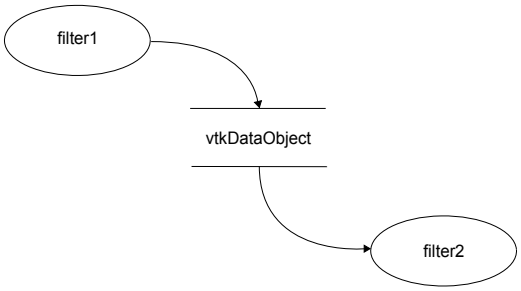



Abbildung 9.4.: Datenflußdiagramm in VTK: Filter 1 erzeugt als Ausgabe ein Datenobjekt, das Filter 2 als Eingabe dient


Abbildung 9.5.: Beispielhafte Darstellung des VTK-Modells mit farblichem Übergang zwischen Visualisierungspipeline und graphischem Modell

#### 9.2.7.3. Vergleich zu OpenGL

Durch das umfangreiche Klassengerüst und die strikte Einhaltung der Pipeline-Struktur ist VTK im Hinblick auf Effektivität und Geschwindigkeit gegenüber OpenGL klar im Nachteil. Allerdings ist VTK nicht unbedingt auf Performance ausgelegt, sondern bietet stattdessen einen riesigen Fundus an Modellierungstechniken zur Verarbeitung und Visualisierung wissenschaftlicher Daten. Jedoch ist diese Komplexität besonders für Einsteiger schwierig zu überblicken und es bedarf einiges an Einarbeitungszeit um das Potential von VTK effektiv nutzen zu können.

# 10. Metavisualisierung

Mit Hilfe der Metavisualisierung werden zur Simulationszeit und zur späteren Nachanalyse die wichtigsten Eigenschaften des Systems graphisch dargestellt. Sie stellt somit die visuelle Komponente der Simulations-Pipeline dar und dient der Bewertung der berechneten Ergebnisse. Im diesem Kapitel werden die grundlegenden Visualisierungskonzepte erklärt und die konkrete Umsetzung im Projekt aufgezeigt.

## 10.1. Konzepte der Metavisualisierung

Das am häufigsten genutzte Verfahren zur Visualisierung skalarer Daten ist die Definition einer Farbtabelle. Diese wird in Form einer Lookup-Tabelle realisiert und enthält $n$ Farben. Zusätzlich werden ein minimaler und maximaler Skalarwert vorgegeben. Es gilt

$$i = \begin{cases} 0 & s \leq s_{min}, \\ int\left((n-1)\frac{s-s_{min}}{s_{max}-s_{min}}\right) & s_{min} \leq s \leq s_{max}, \\ n-1 & s \geq s_{max}. \end{cases} \tag{10.1}$$

Die Farbtabelle stellt somit eine diskretisierte Transferfunktion dar, die jedem auftretenden Skalarwert einen festgelegten Farbwert zuordnet. Die Farbwerte können beispielsweise mit Hilfe des RGB-Farbmodells definiert werden. Zur Visualisierung physikalischer Eigenschaften wie z.B der Temperatur ist eine HSV-Skala die übliche Wahl. Angelehnt an menschliche Erfahrungswerte werden hierbei niedrige Skalarwerte in blau und hohe Skalarwerte in rot dargestellt.
Neben der Verwendung von Farbtabellen ist der Einsatz sogenannter Kontur- oder Isolinien ein weiteres gängiges Verfahren zur Visualisierung von Skalardaten. Die Skalarwerte können im zweidimensionalen Fall als Funktionswerte der Funktion $f(x, y) = z$ aufgefasst werden. Gleiche Funktionswerte werden mit dieser Methode nun als Linien gleicher Farbe dargestellt. Die Isolinien stellen also die Lösung der Funktion $f$ für einen bestimmten Skalarwert $C$ dar. Im dreidimensionalen Fall erweitert sich die Gleichung zu $f(x, y, z) = C$ und es entstehen Isoflächen. Für die Visualisierung von Isoflächen wird häufig der Marching-Cubes-Algorithmus verwendet, der für einen bestimmten Konturwert ein Dreiecksnetz generiert, dass anschließend mit Standardverfahren der Computergraphik gerendert werden kann. Für eine ausführliche Beschreibung des Verfahrens sei an dieser Stelle z.B. auf [LC87] verwiesen.

Zur Visualisierung von Vektordaten wird zumeist auf eine Glyphendarstellung zurückgegriffen. Glyphen sind graphische Objekte, die sich aus geometrischen Primitiven wie Linien, Kugeln oder Zylinder zusammensetzen und ausgehend vom Betrag und der Richtung des betrachteten Vektors bezüglich Größe, Orientierung und Farbe variiert werden können. Diese Darstellung wird verwendet, um den stationären Zustand des Vektorfeld eines gegebenen

Zeitpunkts zu visualisieren.
Eine andere Möglichkeit besteht darin, für ein vorliegendes Vektorfeld mittels numerischer Zeitintegration Trajektorien zu berechnen (sogenannte Stromlinien oder Streamlines).

## 10.2. Anwendung im Projekt

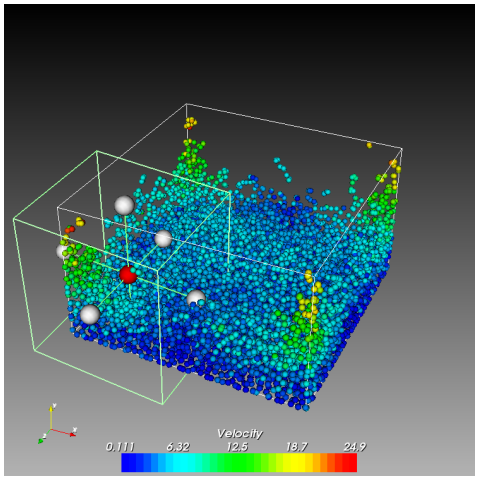
Die wichtigsten Komponenten in einem SPH-basierten Simulationssystem sind naheliegenderweise die Partikel. Dementsprechend ist es wichtig, sämtliche Partikelattribute jederzeit darstellen zu können, um die Qualität der Simulation visuell zu bewerten. Neben den sich ändernden Partikelpositionen und der damit verbundenden Bewegung des dargestellten Fluids, sind auch die Skalar- und Vektordaten der Partikel für eine Analyse wichtig. Zu diesem Zweck wurde für das Projekt auf die im Vorfeld beschriebenen Techniken zurückgegriffen und die Visualisierung der Simulation und der Metadaten mit Hilfe der Bibliothek VTK realisiert. Eine umfrangreiche Beschreibung der Funktionsweise der VTK-Pipeline befindet sich in Kapitel 9.2.7.
Im Folgenden werden kurz die unterschiedlichen Anzeige-Modi des Programms vorgestellt und im Anschluß die Umsetzung der Visualisierung erklärt. Neben den verwendeten Techniken wird hierbei auch ein grober Überblick einiger genutzter VTK-Klassen gegeben. Abschließend folgt eine kurze Bewertung der Ergebnisse.

### 10.2.1. Modus

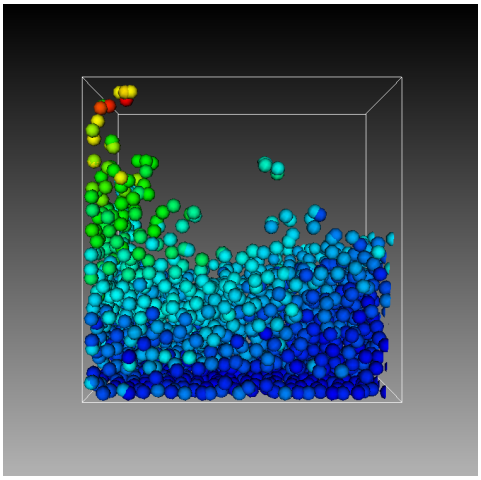
Die Metavisualisierung ist in zwei unterschiedlichen Modi verfügbar. So besteht schon während der Simulation die Möglichkeit eine vereinfachte und echtzeitfähige Ansicht der Metadaten abzurufen. Die Visualisierung beschränkt sich hierbei allerdings auf die Farbkodierung bestimmter Partikeleigenschaften und einer linienbasierten Vektordarstellung (sogenannte Hedgehogs). Zudem können Nachbarschaften ausgewählter Partikel farblich markiert werden.
Der erweiterte Metavisualisierungmodus hingegen basiert auf den Snapshot-Dateien vorausgegangener Simulationsschritte bzw. vollständig abgeschlossener Simulationen. Diese aufgezeichneten Daten beinhalten eine umfassende Beschreibung der Partikel (Position, Skalare und Vektoren). Lediglich die Nachbarpartikel werden aus Speicherplatzgründen nicht mitgespeichert, womit zumindest eine nachträgliche Nachbarschaftsanalyse in diesem Modus entfällt. Stattdessen stehen neben optisch aufwändigeren Visualisierungstechniken auch komfortable Interaktionsmittel zur Verfügung (vgl. Abbildung 10.1).
Als Schnittstelle zur Anwendungsvisualisierung (siehe Kapitel 11) existiert neben dem Modus für die Simulation und die Metavisualisierung zudem eine dritter Modus zur Generierung und Speicherung von Hüllflächen.

### 10.2.2. Realisierung

VTK bietet die gängigen Standardverfahren zur Visualisierung von Skalar- und Vektordaten an. So lässt sich beispielsweise eine Farbkodierung mittels Lookup-Tabellen sehr einfach realisieren. Allerdings wird VTK hauptsächlich für strukturierte und statische Datensätze verwendet. Für unstrukturierte und sich schnell ändernde Daten wie Punktwolken eignen sich viele der angebotenen Filterklassen hingegen nur bedingt. Im Folgenden wird nun auf die einzelnen Teilbereiche eingegangen, die innerhalb des Projekts realisiert worden sind.




(a) Farbkodierte Kugeldarstellung der Partikel und Auswahlrahmen eines Widgets



(b) Resultierendes Volume-of-Interest

Abbildung 10.1.: Beispielhafte Ansicht der erweiterten Metavisualisierung

#### 10.2.2.1. Datenhaltung

Partikel werden ohne explizite Zellstruktur als VTK-Datenobjekt vom Typ `vtkPolyData` instanziiert und erhalten neben der geometrischen Beschreibung als Punktmenge mehrere Arrays für die Skalar- und Vektordaten:

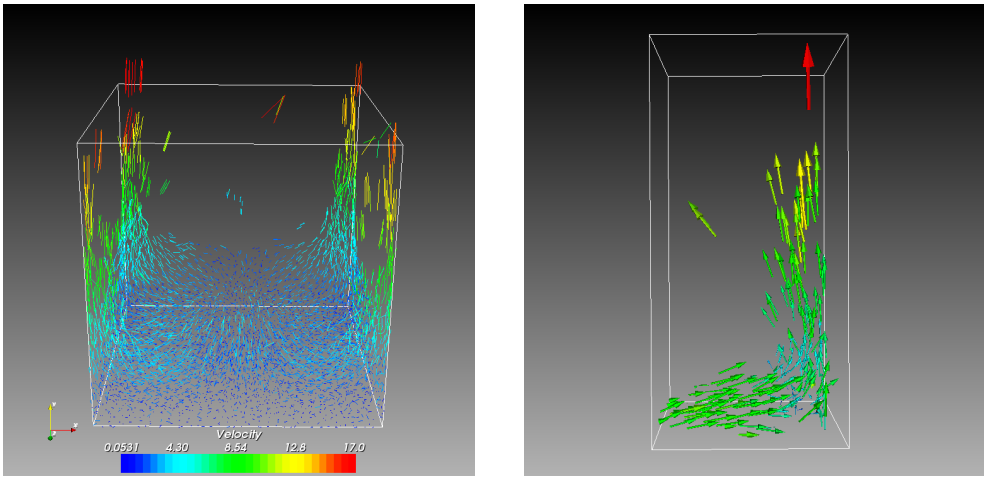
- Dichte
- Druck
- Geschwindigkeit
- Beschleunigung

Zu beachten ist, dass diese Datenobjekte ausschließlich die Visualisierung betreffen und die eigentliche Berechnung der Partikeldaten außerhalb der Rendering-Pipeline geschieht. Die Datenobjekte dienen als Eingabe für die verwendeteten Filterklassen, die die verschiedenen Visualisierungstechniken realisieren und werden mit den Simulationsdaten synchronisiert. Da während der Simulation und dem Snapshot-Modus unterschiedliche Filter benötigt werden und ein Wechseln des Modus jederzeit möglich ist, werden die Datensätze für die Simulation und die Snapshots getrennt organisiert. Dies sorgt zwar für einen höheren Speicheraufwand, gewährleistet aber ein schnelleres Umschalten zwischen den Modi.

#### 10.2.2.2. Vektordarstellung

Die Vektordaten der Partikel werden durch unterschiedliche Glyphen visualisiert. Neben den linienbasierten Hedgehogs (siehe Abbildung 10.2(a)), die auch für Partikelanimationen in Echtzeit geeignet sind, bilden die anderen drei Typen (Kugeln, Pfeile und Zylinder) eher eine optisch anspruchsvollere Variante für einzelne Snapshots (vgl. Abbildung 10.2(b)). Alle Typen lassen sich in ihrer Größe skalieren, außerdem kann durch Ausblendung die angezeigte

Menge der Glyphen verringert werden, um die Performance zu steigern. Die Realisierung von Streamlines erwies sich durch das turbulente Strömungsverhalten der unstrukturierten Partikel hingegen als äußerst schwierig, da diese Technik im Normalfall für stationäre Vektorfelder, die durch reguläre Gitter definiert sind, eingesetzt wird.



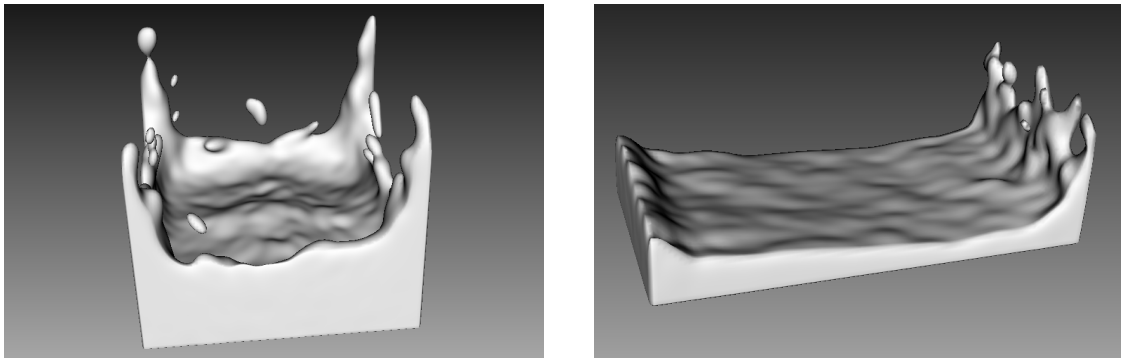

(a) Hedgehogs

(b) Glyphendarstellung mit Pfeilen als Quellobjekt

Abbildung 10.2.: Vektorvisualisierung

#### 10.2.2.3. Contouring

VTK unterstützt verschiedene Klassen, mit deren Hilfe sich eine vorgegebene Anzahl an Isoflächen generieren lässt. Als Parameter werden der gewünschte Skalardatensatz der Partikel und ein oder mehrere Konturwerte übergegeben. Wahlweise lassen sich die Konturwerte auch anhand der gewählten Skalardaten automatisch berechnen. Um ein sichtbares Resultat zu erzielen, ist allerdings im Vorfeld eine Topologisierung der Partikel notwendig, da die angebotenen Filter zum größten Teil nur mit strukturierten Datensätzen arbeiten (vgl. [BGM08]). Eine Delaunay-Triangulierung (mit eingeschränktem Distanzradius) erwies sich als ungeeignet, ebenso die sogenannte Shepard-Methode, die unstrukturierte Punktdaten auf eine strukturierte Punktmenge abtastet. Stattdessen wurde als Basis die Klasse `vtkGaussianSplatter` verwendet, die aus einer gegebenen unstrukturierten Punktwolke Volumendaten erzeugt. Dabei wird ausgehend von den Partikelposition der betrachtete Skalarwert in einem vorgegebenen Radius in umliegendende Voxel „verkleckert“. Eine leicht modifizierte Variante dieser Klasse mit geglättetem Verhalten an der Partikelwolkenoberfläche wird auch im Hüllflächen-Modus verwendet. Die Auflösung des Abtastvolumens wurde im Hinblick auf die Anwendungsvisualisierung hier besonders hoch gewählt. Die anschließend mit dem Marching-Cubes-Algorithmus erzeugten Dreiecksnetze lassen sich auf dem Bildschirm angezeigen (vgl. Abbidlung 10.3) und wahlweise auch für externe Programme im Wavefront-Format abspeichern.

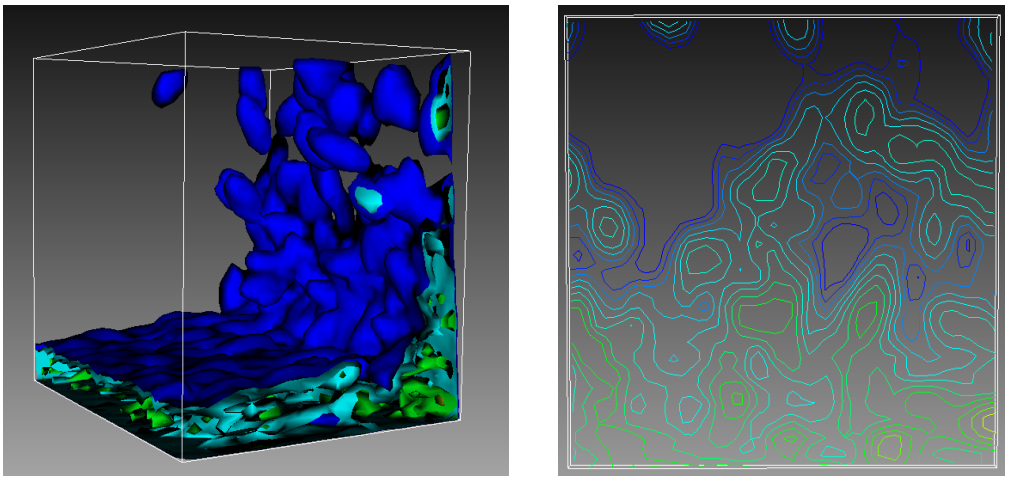


(a) Fallender Tropfen (b) Brechender Damm

Abbildung 10.3.: Hüllflächenberechnung mittels Marching-Cubes-Algorithmus

#### 10.2.2.4. Interaktion

VTK bietet sogenannte Widgets an, die die Interaktion mit dem Benutzer erleichtern und auch im Projekt YASPHS zum Einsatz kommen. Instanzen der Klassen `vtkPlaneWidget` und `vtkBoxWidget` werden verwendet, um bestimmte Bereiche des Bildausschnitts zu markieren und den Inhalt in einem separaten Renderer anzuzeigen. So erlaubt zum Beispiel ein Objekt der Klasse `vtkPlaneWidget` die Markierung einer Schnittebene bereits generierter Isoflächen. Das Resultat ist ein zweidimensionaler Ausschnitt mit entsprechenden Isolinien (vgl. Abbildung 10.4). Es stehen insgesamt vier verschiedene Widget-Varianten zur Verfügung, die unterschiedliche Effekte erzielen.



(a) Farbliche Darstellung der generierten Isoflächen (b) Markierte Schnittebene mit resultierenden Isolinien

Abbildung 10.4.: Beispiel für die Verwendung der Klasse `vtkPlaneWidget`

### 10.2.3. Bewertung

Bezüglich Geschwindigkeit und Effizienz hat die Realisierung durch VTK nur bedingt überzeugen können. Während der Simulation führt die Aufbereitung der Datenobjekte und die damit verbundene Aktualisierung der Pipeline in jedem Zeitschritt bei einer sehr großen Anzahl an Partikeln zu merkbaren Geschwindigkeitsverlusten gegenüber einer rein OpenGL-basierten Implementierung. Für die Visualisierung der Snapshots bietet VTK auf der anderen Seite eine Fülle an Möglichkeiten, wenngleich auch einige Standardverfahren für YASPHS über Umwege realisiert werden müssen. Die Berechnung der Isoflächen kann bei ungünstig gewählten Parametern extrem viel Rechenzeit und enormen Speicherplatz beanspruchen. Die Ergebnisse überzeugen allerdings optisch und stellen somit eine gute Grundlage für die Anwendungsvisualisierung dar.

# 11. Anwendungsvisualisierung

Mit der Anwendungsvisualisierung wird die Simulation visuell in den Kontext ihres, aus der Realität abgeleiteten, Ausgangsproblems gesetzt. Ohne Anwendungsvisualisierung währe nicht erkennbar ob eine Simulation Wasser, Öl oder ein anders geartetes Fluid bzw. Gas simuliert. Erst durch die realistische Darstellung als eine, aus der realen Umgebung bekannten Größe, kann auch intuitiv die Plausibilität der Simulation beurteilt werden.

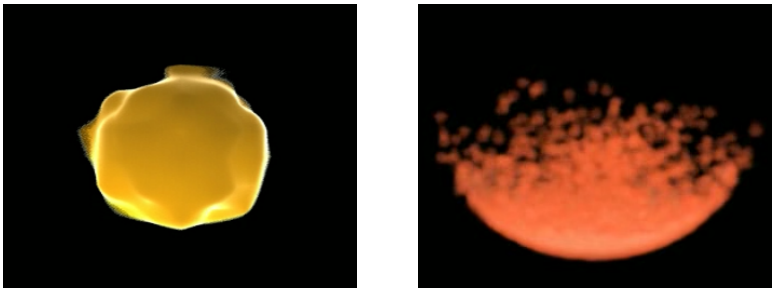
Da die eigentliche Visualisierungsberechnung nicht mit YASPHS sondern durch externe Programme (siehe Kapitel 9.2) übernommen wird, konzentriert sich dieses Kapitel auf die Möglichkeiten zur Gewinnung, Verarbeitung und Speicherung von Daten aus der Simulation in Formaten, die von den Programme gelesen werden können. Die genutzten Visualisierungsarten sind abhängig von den verwendeten Programmen, weshalb jede Methode parallel mit dem zugehörigen Programm vorgestellt wird.

## 11.1. Voxelvisualisierung mit Maya

Maya verfügt über eine integrierte Gas- und Fluidsimulation die finite Elemente als Lösungsverfahren verwendet und die das, den Berechnungen zugrunde liegende Gitter, als Voxelvolumen visualisieren kann. Prinzipiell ist es in Maya nicht vorgesehen, dass externe Voxeldaten in das Programm importiert werden, alle Daten sollen von der internen Simulation erzeugt werden. Glücklicherweise bietet Maya die Möglichkeit eine Simulation zu cachen, d.h. die Ergebnisse der Simulation werden für jeden Frame auf der Festplatte gespeichert und nachher zur Visualisierung nur abgerufen.

Durch den Umweg über einen Mayafluidcache ist es YASPHS möglich die Simulation über den Voxelrecorder (siehe Kapitel 4.5) ab zusamplen und diese Daten im Maya-Voxelformat zu speichern. Diese Voxeldaten können dann in Maya in einen „Fluidcontainer“ eingeladen und abgespielt werden.

Es fällt insgesamt jedoch schnell auf, dass die Voxeldarstellung nicht zur Visualisierung von SPH-Datensätzen geeignet ist (siehe Abbildung 11.1): Viele Partikel „fallen“ durch das Voxelraster und werden so in der Darstellung nicht erfasst, bei dünn besetzten Stellen im Fluid kann es sogar vorkommen, dass die Voxelzellen leer bleiben. Der Speicheraufwand, insbesondere bei Animationen, ist um ein Vielfaches größer als der eigentliche Datensatz. Hinzukommt, dass sich der Maya Voxelrenderer als sehr langsam und schlecht konfigurierbar erwiesen hat, so kann z.B. die Dichte im Volumen nur über Farbgradienten visualisiert werden. Die Möglichkeit einer Beleuchtung der Voxeldaten oder die Berechnung komplexerer Oberflächeneigenschaften fehlt leider. Maya kann zwar für ein gegebenes Voxelgitter eine Hüllfläche ausrechnen, leider ist der verwendete Algorithmus so langsam, dass ein Einsatz in der Praxis undenkbar ist. Insgesamt muss daher von einer Visualisierung mit Maya abgeraten werden, als bessere Alternative haben sich Blender und Yafaray herauskristalisiert.



(a) Oberflächendarstellung (b) Voxeldarstellung

Abbildung 11.1.: Voxelbasierte Anwendungsvisualisierung mit Maya. Abb. (a) visualisiert die, aus den Voxeldaten, generierte Isofläche. Durch die unzureichende Auflösung des Voxelgitters verhält sich die Oberfläche allerdings nicht wie das Fluid. Abb. (b) zeigt die direkte Darstellung der Voxelinformationen über ein Raycastingverfahren. Die einzelnen Partikel sind erkennbar, jedoch ist es schwer einen Gesamteindruck über das Fluid zu gewinnen.

## 11.2. Punktvisualisierung mit Blender

Blender bietet ab Version 2.5 die Möglichkeit, ein, von Punktwolken emitiertes, Dichtefeld zu Visualisieren. Dabei ist die von den Punkten ausgestrahlte Dichtefunktion nicht frei wählbar, sondern der Benutzer muß sich zwischen einer Menge fest definierter Funktionen entscheiden (siehe Abbildung 11.2 und Abbildung 11.3). Diese Visualisierungsart ist daher nicht für eine hochpräzise Veranschaulichung gedacht, eignet sich aber ausgezeichnet zur Darstellung der SPH-Partikel als Gas- oder Staubwolke. Die Daten werden mit Hilfe des Cloudrecorders (siehe Kapitel 4.6) während der Simulation erfasst und dann als Wavefront-Dateien in Blender geladen. Da Blender leider keine Funktion zum Abspielen von Wavefront-Sequenzen hat, ist die Punktvisualisierung auf Standbilder beschränkt.

## 11.3. Hüllflächenvisualisierung mit Yafaray

Weder die Voxel- noch die Punkvisualisierung ist in der Lage eine plausible Darstellung von bekannten Fluiden wie Wasser oder Öl zu erzeugen. Aus diesem Grund wurde YASPHS, mit Hilfe von VTK (siehe Kapitel 9.2.7), um den Export von Isoflächen als Dreiecksmesh erweitert. Die für jeden Frame erzeugte Oberfläche wird als Wavefrontdatei gespeichert und kann anschließend in Yafaray (siehe Kapitel 9.2.3) oder anderen externen Programmen weiterverarbeitet werden (siehe Kapitel 11.4). Yafaray verfügt über eine Reihe von vordefinierten Shadern für verschiedenartige Oberflächen die von Wasser, Glas, Plastik bis hin zu Metallen reicht. Jeder Shader kann über eine Vielzahl von Parametern angepasst werden, so ist es mit dem „Glass“ Shader möglich über die Einstellung des Brechungsindex, der Absorbtionsrate- und farbe sowie des Reflextionsverhaltens Eis, Wasser, durchsichtiges Plastik und viele andere Materialien zu simulieren. Durch hinzufügen von Lichtbrechung kann auch das Schillern eines Ölfilms visualisiert werden.

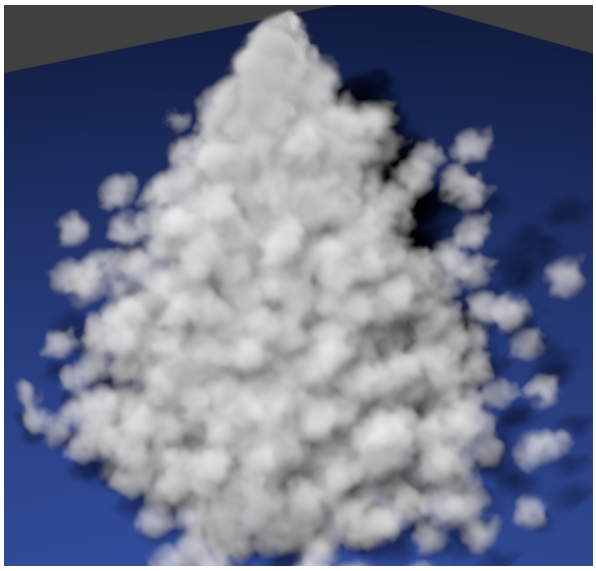


Abbildung 11.2.: Emmitierende Partikelquelle dargestellt mit der Punktvisualisierung und einer einfachen Dichtefunktion.

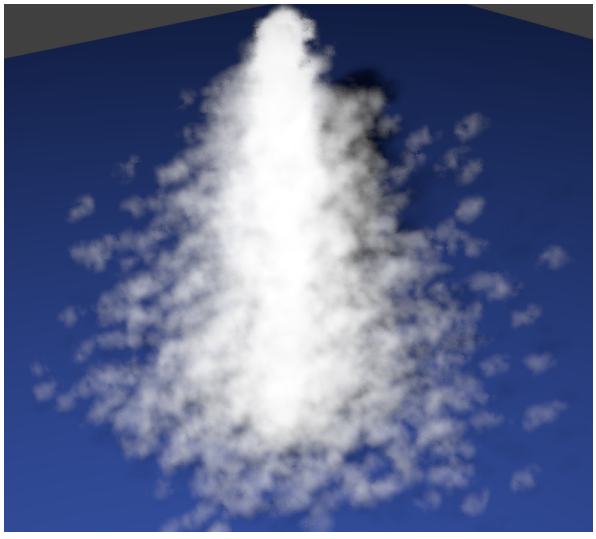


Abbildung 11.3.: Emmitierende Partikelquelle dargestellt mit der Punktvisualisierung. Die Dichtefunktion wird durch eine zusätzliche Rauschfunktion verfremdet um ein Bild zu erhalten, dass mehr an Dampf oder Rauch erinnert.

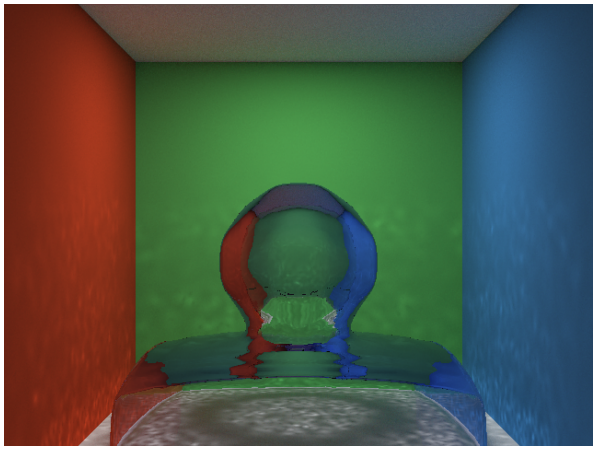


Abbildung 11.4.: Oberflächevisualisierung mit Yafaray: Ein Tropfen fällt in ein ruhendes Fluid. Alle Darstellungsparameter wurden so gewählt, dass das Fluid wie leicht getrübtes Wasser wirkt.

## 11.4. Animationen

Da die meisten gängigen 3D-Programme nicht über die Möglichkeit verfügen eine Sequenz von Wavefrontdateien als Animation zu laden, mußte zur Erzeugung von Videos zusätzliche Software geschaffen werden. Mit Hilfe eines Pythonscripts wird für jede Wavefrontdatei der Isoflächenanimation eine Yafaray-Szene erzeugt, in die das betreffende Mesh eingeladen wird. Die konstruierte Szene wird an Yafaray übergeben und das anschließend synthetisierte Bild dem entsprechenden Frame zugeordnet. Die resultierende Serie von Bildern wird im letzten Schritt mit einem Videoprogramm zu einer abspielbaren Videodatei kodiert.

# 12. Evaluierung

Die Evaluation stellt ein Resume über die, während der PG, gewonnene Praxiserfahrung und vorgenommenen Untersuchungen der theoretischen Hintergründe dar. Die gewonnenen Erkenntnisse gehen über das Wissen der eigentlichen Verfahren hinaus und erlauben eine abschließende Berurteilung der eingesetzten Algorithmen, sowie deren Eigenarten. In einigen Fällen wurden Fehler im theoretischen Ausgangsmaterial gefunden, auf Basis der gewonnenen Erfahrung neue Verfahren entwickelt oder Algorithmen als nicht praktikabel verworfen.
Ein weiterer großer Bestandteil der Evaluierung stellt die Verfikation dar, mit die Verwendbarkeit der Simulationsergebnisse gesichert wird. Eine eingehende Untersuchung der Simulationskomponenten, insbesondere der Kraftberechnung, auf physikalische Plausibilität und Stabilität stellt die Einsetzbarkeit des Programmes sicher oder schränkt diese auf bestimmte Fälle ein.

## 12.1. Nachbarschaftssuche

Im Vordergrund der Evaluierung der Nachbarschaftssuche steht der Vergleich der Verfahren unter klassischen Simulationsbedingungen sowie das Herausarbeiten von bestimmten Vor- und Nachteilen der Verfahren. Um eine objektive Bewertung der Geschwindigkeitsunterschiede vornehmen zu können, wird jedes Verfahren mehrfach zur Simulation von zwei als „klassisch“ anzusehenden Testfällen verwendet und die dabei entstehenden Plots der Framezeiten sowie deren arithmetische Mittelung verglichen. Um eine Verzerrung der Messergebnisse durch äußere Faktoren zu minimieren, wird jeder Testlauf fünfmal wiederholt und die Ergebnisse zu gleichen Teilen gewichtet. Diese Plots ergänzen damit die, im folgenden, geschilderten allgemeinen Erfahrungen die mit jedem Suchverfahren gemacht wurden. Suchverfahren, die als nicht praxistauglich eingestuft wurden, wurden aus dem Programm entfernt und werden in den späteren Detailbetrachtungen ausgelassen.

### 12.1.1. Spatial-Hashing

„Spatial-Hashing“ (siehe Kapitel 5.3.3) hat sich über den gesamten Entwicklungszeitraum der PG als durchschnittlich sehr stabiles und hinreichend schnelles Suchverfahren erwiesen. Im Falle von Partikelballungen kommt es jedoch zu einer spürbaren Verlangsamung bis hinzu quadratischen Laufzeiten. Als besonderes Hindernis bei der Implementierung muß hier das Papier [Hje06] welches als Grundlage des Verfahrens herangezogen wurde erwähnt werden. Die, für die Geschwindigkeit, besonders wichtige Hashfunktion wird im Papier mit einer falschen Primzahl vorgestellt und als Hashfunktion in keiner Weise analysisiert oder diskutiert. Wie sich im Laufe der Entwicklungsarbeit herausgestellt hat, ist die vorgestellt Hashfunktion auch nicht in der Lage, Hashkollisionen zwischen verschiedenen Buckets ausreichend zu minimieren. Durch die Aufstellung einer eigenen Hashfunktion konnte die Anzahl der inneren Schleifendurchläufe (siehe Abb. Abbildung 12.1) wesentlich verringert werden. Insgesamt gehört „Spatial-Hashing“ in fast allen, während der Entwicklung betrachteten, Simulationen zu den schnellsten Verfahren.

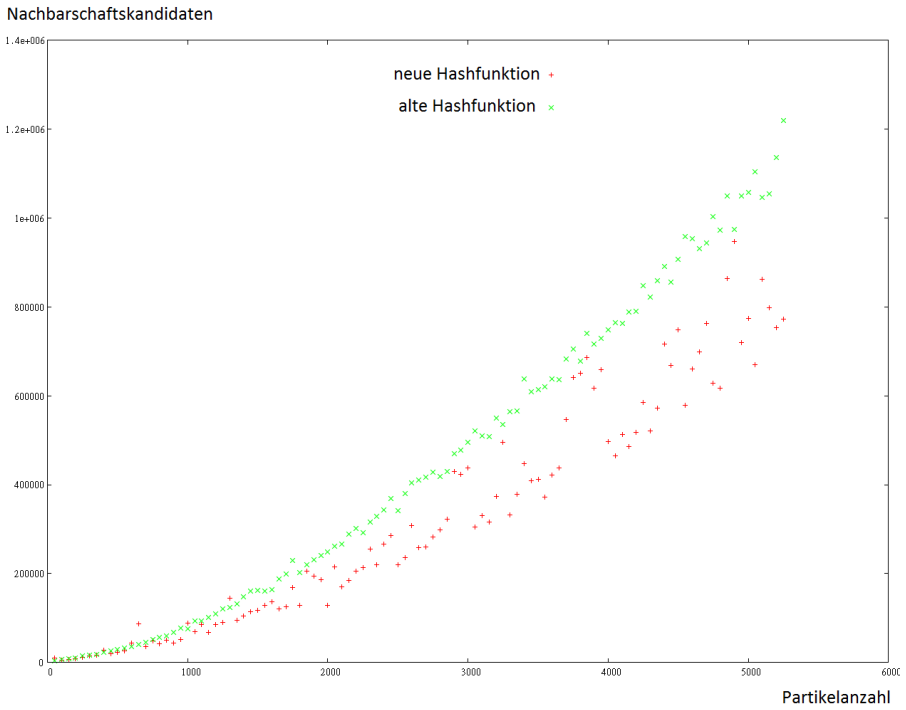



Abbildung 12.1.: Nachbarschaftskandidaten für die alte und neue Hashfunktion. Die neue Hashfunktion streut stärker, bleibt, bis auf wenige Ausnahmen, aber weit unter der alten Hashfunktion.

### 12.1.2. Spatial-Hashing-Automatic-Smoothing-Length

Dieses, von der PG entwickelte, Verfahren basiert auf dem „Spatial-Hashing“ und erbt damit grundsätzlich alle seine Eigenschaften und Geschwindigkeitsverhalten (siehe Kapitel 5.3.6). Im Gegensatz zu seinem Ursprungsverfahren hat das „Spatial-Hashing-Automatic-Smoothing-Length“ dabei die Chance durch die dynamische Anpassung der Glättungslänge mit der Zeit schneller zu werden und auf eine sich stark verändernde Szene zu reagieren. Effektiv wird durch die Veränderung der Glättungslänge zwar die physikalische Berechnung beeinflusst, bei passend gewählten Werten sollte dieser Nebeneffekt allerdings nicht ins Gewicht fallen. Das „Spatial-Hashing-Automatic-Smoothing-Length“ ist das einzige Verfahren dass sich, in begrenztem Rahmen, der zu simulierenden Szene anpassen kann und genießt daher unter allen Verfahren eine Sonderrolle.

### 12.1.3. Union-Spatial-Hashing

„Union-Spatial-Hashing“ (siehe Kapitel 5.3.8) stellt eine Variation des „Spatial-Hashing“ dar und kompensiert den negativen Effekt von Hashfunktionen, die zuviele Hashkollisionen für unterschiedliche Buckets erzeugen. Der Geschwindigkeitsvorteil, der durch „Union-Spatial-Hashing“ erzielt werden kann, ist daher direkt von der Quälität der verwendeten Hashfunktion abhängig. Je mehr Kollisionen während der Suche auftreten, desto mehr wird sich das Verfahren durch die vermiedenen Distanztest vom Spatial-Hashing absetzen.
Nachdem die Hashfunktion für alle Hashingverfahren ausgetauscht wurde, ist ein Vorteil dieses Verfahrens allerdings nicht mehr erkennbar. Es ist jedoch nicht auszuschließen, dass auch die neue Hashfunktion in bestimmten Szenarien zuviele Kollisionen erzeugt, womit das „Union-Spatial-Hashing“ zumindest noch ein Hashingverfahren wäre, das mit der Szene verwendet werden könnte.

### 12.1.4. Capped-Spatial-Hashing

Diese weitere Variante des „Spatial-Hashing“ versucht die temporär auftretenden quadratischen Laufzeiten während der Nachbarschaftssuche durch eine feste Begrenzung der Nachbarschaften für jeden Partikel zu verhindern (siehe Kapitel 5.3.4). Um eine möglichst symmetrische Nachbarschaftsverteilung für jeden Partikel zu erzielen, ist jedoch sehr viel Programmflusslogik in der zentralen Schleife der Nachbarschaftssuche nötig, wodurch der konstante Geschwindigkeitsmalus in den meisten Situationen den Gewinn durch die vermiedene quadratische Laufzeit überwiegt. In speziellen Szenarien mit besonders dichten Partikelhäufungen sollte dieses Verfahren jedoch alle anderen Hashingverfahren überholen. Alternativ kann das Verfahren zur schnellen Vorschausimulation genutzt werden in dem die Anzahl der max. Nachbarschaften auf einen sehr kleinen Wert gesetzt wird.

### 12.1.5. Spatial-Hashing-Vertex

Dieses Verfahren wurde bereits sehr früh wieder aus dem Programm entfernt, da sich der versprochenen Geschwindigkeitsvorteil nicht manifestiert hat. Der Algorithmus entspricht weitestgehend dem „Spatial-Hashing“ mit modifizierter Positionsrundung der Partikel (siehe Kapitel 5.3.5) und skaliert damit ähnlich wie das Grundverfahren. Der größere Suchraum führte allerdings zu einem noch schneller und häufiger auftretenden quadratischen Entwicklung in der Laufzeit bei dichten Partikelgruppen. Ob dieser Ansatz auch ohne Partikelballungen

schneller ist als das normale „Spatial- Hashing“ ist ebenfalls zu bezweifeln, da zwischen den Verfahren sonst keine Unterschiede bestehen. Da „Spatial-Hashing-Vertex“ in allen, während der Projektgruppe simulierten, Szenen zu den langsamste Verfahren zählte und gegenüber den anderen Verfahren kein Alleinstellungsmerkmal besitzt wurde es entfernt und wird somit auch nicht in den folgenden Vergleich einbezogen.

### 12.1.6. Optimized-Spatial-Hashing

„Optimized-Spatial-Hashing“ (siehe Kapitel 5.3.7) wurde genauso wie „Spatial-Hashing-Vertex“ wieder aus dem Programm entfernt, da es für Simulationen von seriöser Größe viel zu langsam ist. Das Verfahren ist für Szenen mit animierter Geometrie ausgelegt, was für alle, von YASPHS simulierbaren, Szenen nicht zutrifft. Der Algorithmus kann so nie seine Stärke auspielen, erleidet jedoch durch die komplexere interne Struktur einen konstanten Geschwindigkeitsmalus. Der vermutlich gewichtigste Anteil an diesem Malus entsteht durch die Verwendung von Listen zum Speichern von Partikeln. Die Einfüge- und Löschoperationen in diese Listen während der Suche lösen sehr viele Allokationen und Deallokationen aus, deren Geschwindigkeit maßgeblich von den verwendeten Compilern und Bibliotheken abhängig ist. Hier ist besonders die GNU STL hervorzuheben, die Allokationen von kleinen Objekten augenscheinlich wesentlich schneller verarbeiten kann als das Gegenstück des Visual Studio Compiliers und so das Verfahren unter Linux wesentlich schneller laufen ließ als unter Windows. Analog zum „Spatial-Hashing-Vertex“ wird dieses Verfahren im Geschwindigkeitsvergleich nicht mehr betrachtet.

### 12.1.7. Reguläres Gitter mit fester Speicherzuodnung

Das „Reguläre Gitter“ (siehe Kapitel 5.3.2) ist, neben der naiven Suche, zweifelsohne das am einfachsten strukturierte Verfahren und gleichzeitig in vielen Fällen äußerst schnell. Das „Reguläre Gitter“ wurde implementiert als sich abzeichnete, dass die Geschwindigkeit, die mit den Hashingverfahren (siehe Kapitel 5.2.2) erreicht werden kann, für große Simulationen nicht ausreichend ist. Wie fast alle Gridverfahren profitiert das „Reguläre Gitter“ durch seinen intuitiven Aufbau und der ausnutzbaren geometrischen Struktur des Gitters. Darüberhinaus wird hier der typische Tausch von Speicher gegen mehr Geschwindigkeit vorgenommen, denn der Speicherverbrauch des „Regulären Gitters“ wächst kubisch mit der Ausdehnung der Simulation. Als weitere Einschränkung kommt hinzu, dass jede Zelle, aufgrund der Implementierung, im Gitter nur eine begrenze Anzahl von Partikeln aufnehmen kann, es bei dichten Partikelwolken also durchaus zu falschen Berechnungen kommen kann. Insgesamgt stellt das „Reguläre Gitter“ damit eher ein Nischenverfahren dar, da seine zahlreichen Einschränkungen eine Einsatz für physikalisch belastbare Simulationen weitestgehend verhindern.

### 12.1.8. Symmetry-Aware-Grid

Das „Symmetry-Aware-Grid“ (kurz SA Grid) wurde als letztes Verfahren implementiert und profitiert somit automatisch von den, aus den anderen Verfahren, gewonnenen Erfahrungen. Der Hauptvorteil des Verfahrens ist die Ausnutzung der Symmetrie des Distanztests, sowie die intuitive Unterteilung in eine Gitterstruktur. In der Tat hat sich das SA Grid sofort als mit Abstand schnellstes Suchverfahren herauskristalisiert. Grundsätzlich wird dieser Geschwindigkeitsgewinn zwar auch mit einem kubisch wachsenden Speicherverbrauch erkauft, im Falle des SA Grids betragen die Kosten pro Zelle jedoch nur 12 Byte. Ein 200x200x200 Gitter, mit einem Partikel in jeder Gitterzelle, würde damit gerade einmal 91 MB Speicher für das Gitter, jedoch

2502 MB Speicher für die Partikel verbrauchen. Selbst bei großen Simulationen dominiert der Speicherverbrauch der Partikel alle anderen Komponenten. Anders als das „Reguläre Gitter“ hat das SA Grid keine Beschränkung für die maximale Anzahl der Partikel in einer Zelle und kann daher uneingeschränkt für jede Simulation genutzt werden. Wie jedes Verfahren hat aber auch das SA Grid Schwachstellen, da das SA Grid bei seiner Suche über die aktiven Zellen iteriert, statt über die Partikel, kann es zu einer spürbaren Verlangsamung kommen, wenn sich die simulierten Partikel sehr weit im Raum ausdehnen aber keine Nachbarpartikel mehr haben. Der Vorteil eines Zell- gegenüber eines Partikelbasierten Tests hebt sich somit langsam auf. Dieser Effekt tritt jedoch nur sehr selten auf, da die meisten Simulationen durch ein umschließendes Boundingvolumen begrenzen werden. Insgesamt ist das SA Grid damit als das schnellste Verfahren hervorgegangen und sollte, sofern in der Simulation keine Sonderbedingungen vorliegen, auch als Standardverfahren genutzt werden.

### 12.1.9. Geschwindigkeitsvergleich

Wie in Kapitel 5 beschrieben, besteht der Vergleich der Verfahren aus der Auswertung von gemittelten Framezeiten und Rechenzeiten pro Testszene. Als Testumgebung wurde ein Rechner mit vier Rechenkernen, 8 GB Arbeitsspeicher und 64 Bit Linux ausgewählt. Das Programm wurde entsprechend für 64 Bit und aktivierten Optimierungen mit gcc compiliert. Während der Simulation sind alle sonstigen, zur korrekten Berechnung nötigen, Komponenten wie Kraftberechung, Kollisionserkennung oder Zeitintegration ebenfalls aktiv, auf eine Darstellung der Simulation über das 3D-Fenster wird allerdings verzichtet um den Einfluss der Grafikkarte auf die Messungen zu minimieren. Um die Vergleichbarkeit der Verfahren untereinander zu gewährleisten, werden alle Simulationen mit den selben Parametern durchgeführt und für Nachbarschaftssuchverfahren mit einstellbaren Parametern einmalig sinnvolle Werte gesetzt. Verwendete Kräfte:

- Gravitation mit normaler Erdbeschleunigung
- Druckkraft
- Viskosität ($\mu$ ist szeneabhängig)

Einstellungen für die Nachbarschaftsverfahren:

- „Capped Spatial Hashing“
  - Max. Particlechunks: 1
- „Spatial Hashing automatic smooth length“
  - min. neighbours: 15
  - max. neighbours: 25
  - upward gradient: 0.00001

Folgende Testszenen wurden verwendet, für genau Beschreibung der Szene mit Partikelzahlen siehe Kapitel 12.2.2:

- Fallender Tropfen
- Brechender Damm
- Fallender Volumenkörper

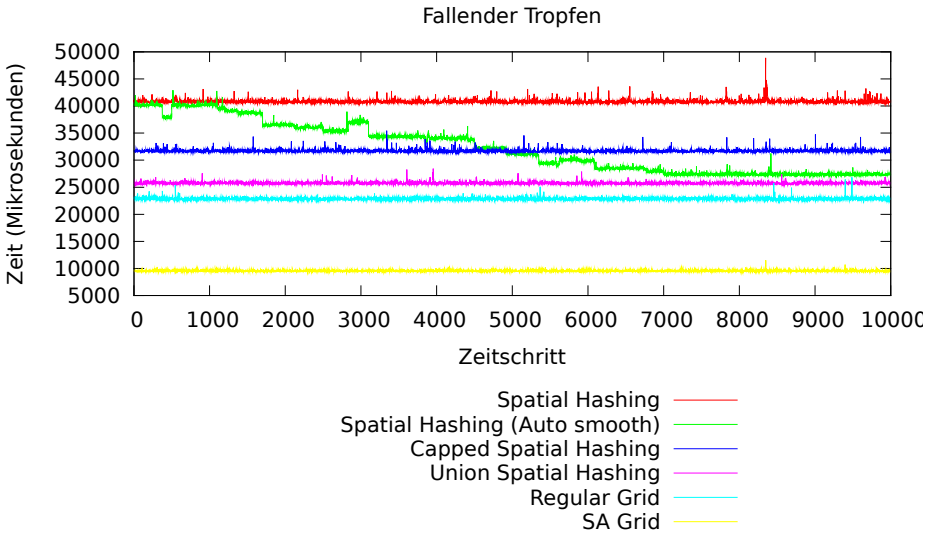



Abbildung 12.2.: Framezeiten für „fallender Tropfen“.

### 12.1.10. Interpretation

Die Plots (siehe Abbildung 12.2, Abbildung 12.3 und Abbildung 12.4) spiegeln das aus den vorherigen Absätzen Erwartungen wieder. Auffällig ist besonders, dass sich das dynamische „Spatial-Hashing“ nur in Testfällen verbessern kann, die größere Partikelbewegungen und Umschichtungen enthalten. Da das „Union-Spatial-Hashing“ in allen drei Testfällen immer noch das schnellste Hashingverfahren ist, deutet darauf hin, dass die verwendete Hashfunktion immernoch viel Potential für Verbesserungen bietet. Da alle Testfälle durch einen recht dichten Partikelhaufen geprägt sind, bzw. sich diese Haufen während der Simulation ausbilden, kann das SA Grid sein volles Potenzial ausspielen und setzt sich als schnellstes Verfahren vom Rest ab. Positiv für alle Verfahren ist die fast konstante Rechenzeit pro Simulationsschritt, keines der Verfahren zeigt übermässige Ausreißer oder eine stetige Verlangsamung.

## 12.2. Kraftberechnung

Durch die Dynamik der Flüssigkeit und durch externe Einflüsse wirken auf die einzelnen Partikel und somit auf das gesamte Fluid Kräfte. Diese Kräfte verursachen eine Beschleunigung der jeweiligen Partikel und führen somit zu einer Bewegung des Fluids. Durch geeignete Integrationsverfahren resultiert aus der Beschleunigung eine neue Geschwindigkeit und Position der Partikel. Bei den Kräften handelt es sich um Feldgrößen, die mit SPH berechnet werden. SPH verwendet hierzu Kernelfunktionen, deren Berechnung die benachbarten Partikel des Betrachteten voraussetzt. Diese werden in der vorhergehenden Nachbarschaftssuche bereitgestellt.

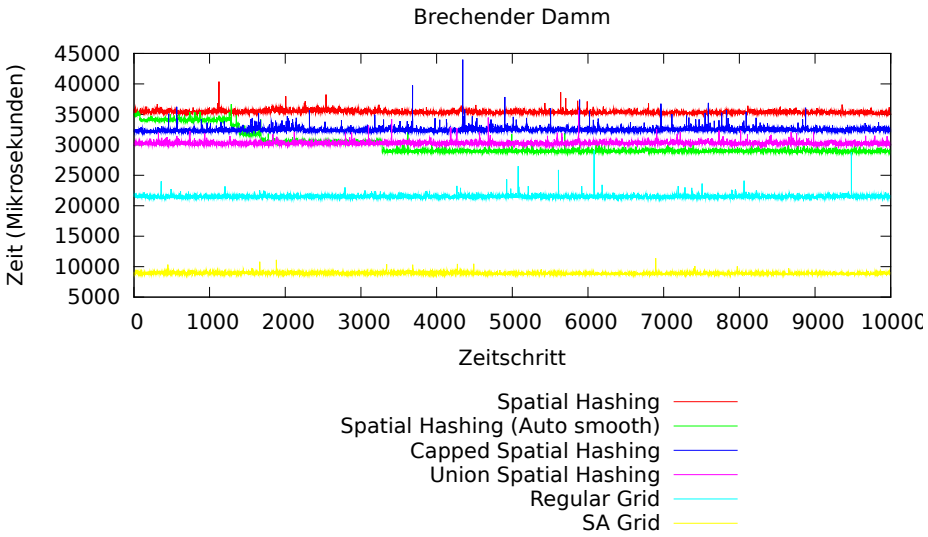



Abbildung 12.3.: Framezeiten für „brechender Damm“.

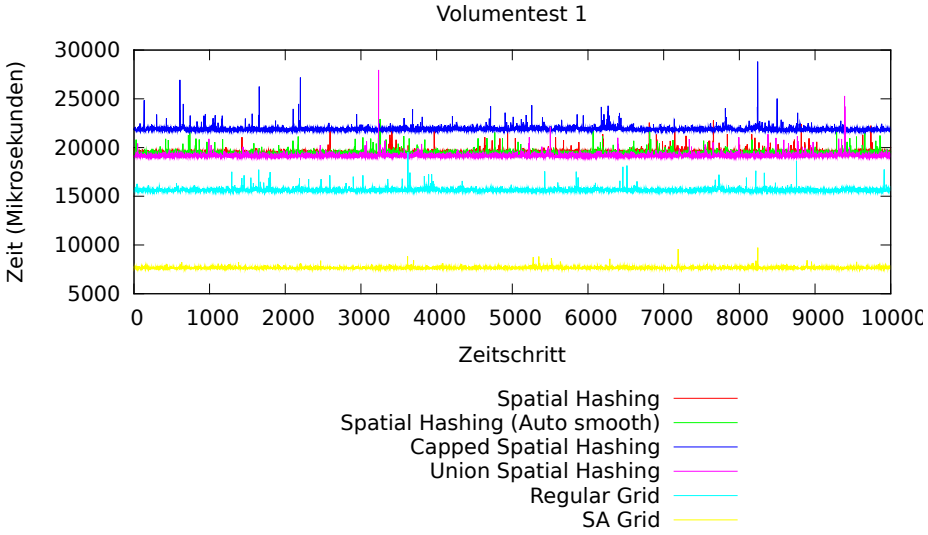



Abbildung 12.4.: Framezeiten für „Volumentest 1“.

### 12.2.1. Komponenten

In einem für die Kraftberechnung zur Verfügung gestellten Verwaltungsteil, der als Singleton realisiert wurde, wird die Berechnung der Dichte, des Druckes und der Kräfte umgesetzt. Dies passiert threadbasiert, so dass die Berechnungen über Multicore-Systeme auf einzelnen Prozessoren ausgeführt und somit schneller durchgeführt werden können. Die in der Simulation aktiven Partikel werden in Einheiten unterteilt und den jeweiligen Threads übergeben, die die Berechnung durchführen. Ebenfalls werden alle zur Berechnung nötigen Variablen zur Verfügung gestellt, sei es als konstant angenommene Werte wie der Ruhedichte, oder über das Benutzerinterface variierbare Größen. Die Partikelmasse und der Einflussradius, die für alle Partikel den selben Wert annehmen, werden für primitive Körper als Partikelgruppe abhängig von deren Volumen berechnet und dann auf alle weiteren Partikel, die z.B. in einer Quelle erzeugt werden, angewendet. Wird in einer Szene ausschließlich eine Partikelquelle verwendet, so werden Initialwerte für die Partikelmasse und den Einflussradius vergeben. Alle für die Simulation notwendigen Kraftplugins können über die Benutzungsschnittstelle ausgewählt werden, so dass nur die aktiven Kräfte in die Berechnung einbezogen werden. Die Berechnung der physikalischen Größen wird separat für jedes Partikel umgesetzt, wobei einige Kräfte Feldgrößen sind, die unter Einbeziehung der Nachbarpartikel über eine Kernelfunktion berechnet werden. Hierbei stehen für jede Feldgröße geeignete Kernelfunktionen und deren Ableitungen zur Verfügung. Da für die Berechnung der jeweiligen Kräfte die Dichte und der Druck vorhanden sein müssen, werden diese vor der Berechnung der Kräfte ausgeführt. Um Luftdruck zu simulieren, wird eine sogenannte Ruhedichte verwendet, die bei Unterschreiten dieser für negativen und bei Überschreiten für positiven Druck sorgt, was auf die Druckkraft angewendet anziehende bzw. abstoßende Kräfte bewirkt. Neben der Berechnung der Druckkraft, die eine wesentliche Komponente hydrodynamischer Flüssigkeitssimulationen ist, können optional Viskosität, Gravitation, Oberflächenspannung und eine Zentralkraft in die Berechnung einfließen. Aufsummiert kann daraus die Beschleunigung und über Integrationsverfahren aus 12.4 die neue Position und Geschwindigkeit der Partikel ermittelt werden.

### 12.2.2. Physikalische Verifikation

Die physikalische Verifikation dient der sachgerechten Verifikation und zur Überprüfung des korrekten Verhaltens der physikalischen Parameter und der Simulationspipeline, indem in vordefinierten Testfällen physikalische Parameter variiert und das plausible Verhalten des Fluids geprüft wird. Dies garantiert das Funktionieren des Programms unter vorgegebenen, standartisierten Testbedingungen. Die nachfolgenden Testfälle untersuchen in diversen Szenarien, ob das erwartete Volumen und somit die Dichte und der Druck angenommen werden. Desweiteren wird die Kompressibilität durch Variation der Gaskonstante und die Geschwindigkeit aufgrund der Bewegung des Fluids durch Änderung der Viskosität untersucht.

#### 12.2.2.1. Volumen

**Kugel**
Als Testfall zur Verifikation der physikalischen Größen Dichte und Druck wird zunächst das Volumen einer Kugel herangezogen. Dieser Test soll die korrekte Implementierung und den Simulationsablauf der eingestellten physikalischen Parameter und den berechneten physikalischen Größen Masse, Dichte und Druck zeigen. Tabelle 12.1 zeigt die verwendeten Parameter.

Tabelle 12.1.: Parameter Testfall Kugel

| **Geometrie** | |
|---|---|
| Radius | 1 m |
| Partikel | 4169 |
| Partikelanordnung | reguläres Gitter |
| Rasterabstand | 0,1 m |
| **Kräfte** | |
| Druckkraft | |
| Zentralkraft | 1 N(konstant) |
| **Integration** | |
| Methode | Leap Frog |
| Zusatz | XSPH( $\varepsilon = 0,5$ ) |
| Zeitschritt | 0,00001 s |
| **Physikalische Parameter** | |
| Gaskonstante | 500 J |
| Ruhedichte $\varrho_0$ | 998,28998 $\frac{kg}{m^3}$ |
| Einflussradius $h$ | 0,19306183 m |
| **Nachbarschaftssuche** | |
| Suchverfahren | Spatial Hashing |

Ergebnis: Nach 10000 Simulationsschritten pendelte sich das berechnete Volumen aus Partikelmasse und Partikeldichte bei 4.19197 ein.. Das Referenzvolumens ist 4.18879, so dass die Abweichung bei 0,0759% liegt. Die durchschnittliche Dichte betrug 997.9, die angestrebte Ruhedichte 998,29. Die durchschnittliche Anzahl der Nachbarpartikel beträgt 24 und liegt etwas unter der Angestrebten von 30 Partikeln. Zurückzuführen ist dies auf die relativ große Kugeloberfläche und der geringen Anzahl an Partikeln. Obwohl die Gaskonstante sehr hoch angesetzt wurde, blieb die Simulation über den gesamten Simulationsverlauf stabil. Dies ist jedoch nicht immer der Fall. Sobald weitere externe Kräfte wie die Gravitationskraft hinzugenommen wurden, musste der Zeitschritt oder die Gaskonstante entsprechend angepasst werden, damit die Simulation nicht kollabierte.

**Füllkörper**
Im zweiten Testfall wird erneut das Volumen und damit die Dichte und der Druck geprüft. Hierzu wurde ein Quader mit der Kantenlänge 2 mit dem entsprechenden Volumen befüllt. Es wurden die Parameter aus 12.2 verwendet.

Ergebnis: Nach 10000 Simulationsschritten wurde eine durchschnittliche Dichte von 997,3 erreicht. Das Volumen wurde mit 8,01259 ermittelt, das tatsächliche ist 8.

Mit diesen beiden Testverfahren konnte zum einen gezeigt werden, dass das berechente Volumen nur eine kleine Abweichung vom Tatsächlichen aufwies. Ebenfalls konnte gezeigt werden, dass die Dichte des Fluids ebenfalls nur geringfügig von der tatsächlich vorgegebenen Ruhedichte $\varrho_0$ abwich. Da sich die Dichte und der Druck nur durch die konstanten Werte Masse und Gaskonstante vom Volumen unterscheiden, ist davon auszugehen, dass diese Werte ebenfalls korrekt berechnet wurden.

Tabelle 12.2.: Parameter Testfall Füllkörper

| **Geometrie** | |
|---|---|
| Maße | 2x2x2 m |
| Partikel | 8000 |
| Partikelanordnung | reguläres Gitter |
| Rasterabstand | 0,1 m |
| **Kräfte** | |
| Druckkraft | |
| **Integration** | |
| Methode | Leap Frog |
| Zusatz | XSPH( $\varepsilon = 0,5$ ) |
| Zeitschritt | 0,00001 s |
| **Physikalische Parameter** | |
| Gaskonstante | 500 J |
| Ruhedichte $\varrho_0$ | 998,28998 $\frac{kg}{m^3}$ |
| Einflussradius $h$ | 0,19275731 m |
| **Nachbarschaftssuche** | |
| Suchverfahren | Spatial Hashing |

#### 12.2.2.2. Kompressibilität

**Variation der Gaskonstante und schrittweise Erhöhung der Gravitation**
Mit diesem und dem folgenden Testfall wird die Kompressibilität eines Fluides getestet. Dazu wird die Gaskonstante variiert, was im Falle einer höheren Gaskonstante für höhere Inkompressibilität sorgen sollte. Als Behälter dient ein Quader mit der Kantenlänge 2, der zu $\frac{3}{4}$ mit Partikeln befüllt wurde und durch die eingestellte Gravitation am Boden festgehalten wird. Die Parameter aus Tabelle 12.3 kamen zur Anwendung.
Die Gravitation wurde vom Wert 0 schrittweise auf 1, 4 und 9,81 in negativer y-Richtung angehoben, die Gaskonstante in separaten Durchläufen auf 50, 30 und 20 gesetzt. Dabei wurden die Werte aus Tabelle 12.4 verwendet.
Das beobachtete Minimum bei einer Gaskonstante von 50 lag im gesamten Simulationsablauf bei 5,96096, was einer Abweichung von 0,593% entspricht.
Bei einer Gaskonstante von 30 wurde das Minimum 5,88999 erreicht, eine Abweichung von 1,868%.
Die Gaskonstante bei 20 angesetzt lieferte ein minimales Volumen von 5,77089. Dies ist eine Abweichung von 3,97%.
Deutlich sind die Auswirkungen der Gaskonstanten auf das Fluidverhalten zu beobachten. Wie bereits angenommen führt eine höhere Gaskonstante zur erwarteten, steiferen Fluidbewegung. Insgesamt war die Abweichung auch mit einer Gaskonstanten von 20 sehr gering. Ein weiterer Testfall, der das Fluid entsprechend mehr verformen sollte, wurde ebenfalls durchgeführt.

**Fallender Volumenkörper**
In diesem Testfall wird wie in 12.5 ein 2 Einheiten hoher Behälter im oberen Teil zur Hälfte mit Partikeln befüllt, der beim Simulationsverlauf durch die Gravitation nach unten fällt.
Für die Gaskonstante wurden nacheinander die Werte 5, 20, 30 und 50, ansonsten die Werte aus Tabelle 12.5 verwendet.

Tabelle 12.3.: Parameter Testfall Kompressibilität

| **Geometrie** | |
|---|---|
| Maße | 2x1,5x2 m |
| Partikel | 8228 |
| Partikelanordnung | reguläres Gitter |
| Rasterabstand | 0,1 m |
| **Kräfte** | |
| Druckkraft | |
| Viskosität | 1 $Pa \cdot s$ |
| Gravitation | schrittweise Anhebung |
| **Integration** | |
| Methode | Leap Frog |
| Zusatz | XSPH( $\varepsilon = 0,5$ ) |
| Zeitschritt | 0,00001 s |
| **Physikalische Parameter** | |
| Gaskonstante | 50 J, 30 J, 20 J |
| Ruhedichte $\varrho_0$ | 998,28998 $\frac{kg}{m^3}$ |
| Einflussradius $h$ | 0,17 m |
| **Nachbarschaftssuche** | |
| Suchverfahren | Spatial Hashing |

Die Volumenänderung und die dazugehörige Geschwindigkeitsänderung im Verlauf der Simulation kann in 12.8 betrachtet werden. Am Anfang der Kurve nähert sich das Volumen bei großer Gaskonstante dem geforderten Volumen viel schneller an, als bei entsprechend kleinerer Gaskonstante. Deutlich ist zu erkennen, dass der Ausschlag bei kleinerer Gaskonstante beim Auftreffen auf die Oberfläche sehr viel größer ist, als bei Wahl einer größeren Gaskonstante. Der Kurvenverlauf ist bei kleiner Gaskonstante auch viel Träger, was sich im ruhigeren Schwingungsverhalten zeigt. Die Wahl der Volumenausdehnung scheint im Verhältnis zur Gaskonstante zu stehen, was hier jedoch nicht weiter untersucht wurde. Die prozentuale Abweichung bzw. Kompressibilität zum Referenzvolumen 8 beträgt 43%, 20%, 13% und 7,5% für eine Gaskonstante von 5, 20, 30 und 50, was relativ hohe Abweichungen sind, die jedoch auch aus einer hohen Geschwindigkeit resultieren. Hier ist es nötig, die Volumenänderung im Verhältnis zur Partikelanzahl, die mit 8.000 relativ gering ist, der Geschwindigkeit und Gravitation zu sehen. An diesem Graphen kann ebenfalls sehr gut beobachtet werden, dass die Wahl einer zu hohen Gaskonstante auch Probleme verursacht. So wird das System am Ende der Simulation mit einer Gaskonstante von 50 unstabil, was an dem unregelmäßigen Volumenverlauf und den hochfrequenten Geschwindigkeitsausschlägen zu erkennen ist. Hier sollte daher zwischen Stabilität bzw. Zeitschritt und Verhältnismäßigkeit der Größe der Gaskonstante unterschieden werden.

#### 12.2.2.3. Viskosität

Dieser Testfall dient der Untersuchung des Flüssigkeitsverhaltens bei Änderung der Viskosität. Annahme ist, dass das Fluidverhalten bei höherer Viskosität zu einem trägeren Fluidverhalten führt. Ein 4 Einheiten langer Behälter bildet zusammen mit einem im mittleren Bereich der

Tabelle 12.4.: Parameter Testfall Gravitation

| Gaskonstante | Zeitschritt | Gravitation | Volumen | Dichte |
|---|---|---|---|---|
| 50 J | 10000 s | 0,0 $\frac{m}{s^2}$ | 6,0 $m^3$ | 998,2 $\frac{m}{s^2}$ |
| 50 J | 20000 s | 1,0 $\frac{m}{s^2}$ | 5,999 $m^3$ | - |
| 50 J | 30000 s | 4,0 $\frac{m}{s^2}$ | 5,98749 $m^3$ | - |
| 50 J | 40000 s | 9,81 $\frac{m}{s^2}$ | 5,96441 $m^3$ | 1004 $\frac{m}{s^2}$ |
| 30 J | 10000 s | 0,0 $\frac{m}{s^2}$ | 6,01164 $m^3$ | 997 $\frac{m}{s^2}$ |
| 30 J | 20000 s | 1,0 $\frac{m}{s^2}$ | 6,00042 $m^3$ | 998,7 $\frac{m}{s^2}$ |
| 30 J | 30000 s | 4,0 $\frac{m}{s^2}$ | 5,96974 $m^3$ | 1004 $\frac{m}{s^2}$ |
| 30 J | 40000 s | 9,81 $\frac{m}{s^2}$ | 5,90856 $m^3$ | 1004 $\frac{m}{s^2}$ |
| 20 J | 10000 s | 0,0 $\frac{m}{s^2}$ | 6,01526 $m^3$ | - |
| 20 J | 20000 s | 1,0 $\frac{m}{s^2}$ | 6,00199 $m^3$ | - |
| 20 J | 30000 s | 4,0 $\frac{m}{s^2}$ | 5,92772 $m^3$ | - |
| 20 J | 40000 s | 9,81 $\frac{m}{s^2}$ | 5,81245 $m^3$ | 1032 $\frac{m}{s^2}$ |

Länge 1 über die gesamte Höhe mit Partikeln befüllten Flüssigkeitsvolumen die Grundszene. Die Startbedingung ist in Abbildung 12.7 aufgeführt.

**Einfluss der Viskosität auf das Fluid**

Für die Viskosität wurden in mehreren Durchläufen die Werte 0,05, 1 und 10 verwendet.
In 12.7 ist der Verlauf der maximalen Geschwindigkeit des Fluids über einen Zeitraum von 50 Sekunden aufgetragen. Zum einen ist zu erkennen, dass bei höherer Viskosität der maximale Ausschlag geringer ausfällt, zum anderen bilden die Kurven kleinere Amplituden aus. Auffällig ist die geringere Endgeschwindigkeit bei hoher Viskosität. Eine Beruhigung des Fluids auch über einen längeren Zeitraum beobachtet, ist nicht zu beobachten. Mit diesem Ergebnis konnte gezeigt werden, dass die Viskosität direkten Einfluss auf das Verhalten des Fluids hat und sich den Erwartungen entsprechend verhält. Es wäre interessant, den Zusammenhang zwischen Viskosität und finaler, maximaler Geschwindigkeit zu untersuchen.

#### 12.2.2.4. Brechender Damm

Um YASPHS mit bekannten Veröffentlichungen wie [MBRA10] zu vergleichen, wurde das Problem des brechenden Damms herangezogen. Der verwendete Behälter ist 4 Einheiten lang, das initiale Volumen Wasser ist 1 Einheit lang und 2 Einheiten hoch. Weitere Parameter sind in Tabelle 12.7 ersichtlich. Aufgrund der Zweidimensionalität des Problems wird die Anzahl der Schichten auf eine beschränkt.

Leider sind nicht alle in der Simulation von [MBRA10] verwendeten Daten bekannt, so dass eigene Parameter für die Simulation so angepasst wurden, dass sie das Ergebnis aus der referenzierten Literatur entsprechend approximierten.
In 12.9 wird die maximale Ausbreitung der Fußsohle in x-Richtung der Volumenspitze aufgezeichnet. Diese verhält sich so wie der Kurvenverlauf in [MBRA10]. Der etwas flachere Verlauf kann aus den unterschiedlich gewählten Parametern resultieren.

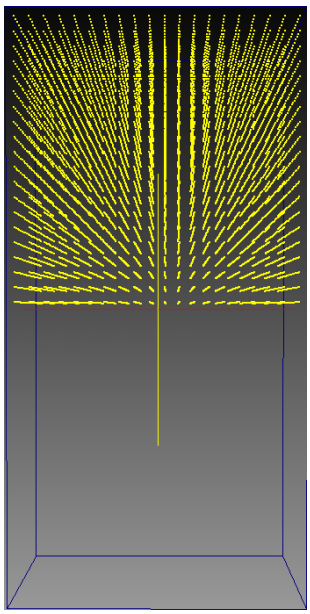


Abbildung 12.5.: Testfall Kompressibilität: Fallender Volumenkörper, Initialstatus

### 12.2.3. Ergebnis

Die durchgeführten Versuche ergaben, dass sich das Fluid den Erwartungen entsprechend verhielt. Ein Problem während der Tests bestand in der korrekten Auswahl der Initialbedingungen. Viele Parameter konnten zwar aufgrund physikalischer Zusammenhänge vorher berechnet und in der Simulation verwendet werden, Parameter wie eine nicht korrekt gewählte Smoothing-Length führten aber zu Nebeneffekten und dem Versagen der Simulation. In [Kel06] wird zwar die Gleichung

$$\sqrt[3]{\frac{3Vx}{4\pi n}}, \tag{12.1}$$

mit $V$ als Volumen, $x$ als Anzahl der Nachbarpartikel und $n$ als Partikelanzahl genannt, diese gilt aber nur für eine feste Anzahl an Nachbarpartikeln. Das Problem, welches schnell erkannt wurde war, dass bei dieser Formulierung der Einflussradius bei steigender Partikelanzahl kleiner wird und somit nicht genügend Partikel abdeckt, um genaue Messungen durchführen zu können. Es stellte sich heraus, dass die zu wählende Smoothing-Length vom initialen Partikelabstand dx abhängig war. Dieses Verhalten kann anhand der Abbildung 12.10 nachvollzogen werden, in der eine Referenzfunktion bei einem Verhältnis von 1:2 zwischen Partikelabstand dx und Smoothing-Length $h$ approximiert wird. Größere bzw. kleinere Werte führen zu einer Verschlechterung der Approximationsgüte. Der genaue Zusammenhang zwischen Smoothing-Length und Partikelabstand ist nicht Gegenstand dieser Projektgruppe.

Tabelle 12.5.: Parameter Testfall fallender Volumenkörper

| **Geometrie** | |
|---|---|
| Maße | 2x4x2 |
| Partikel | 8000 |
| Partikelanordnung | reguläres Gitter |
| Rasterabstand | 0,1 |
| **Kräfte** | |
| Druckkraft | |
| Viskosität | 1 |
| Gravitation | (0;-9,81;0) |
| **Integration** | |
| Methode | Leap Frog |
| Zusatz | XSPH( $\varepsilon = 0,5$ ) |
| Zeitschritt | 0,0002 |
| **Physikalische Parameter** | |
| Gaskonstante | 50, 30, 20, 5 |
| Ruhedichte $\varrho_0$ | 998,28998 |
| Einflussradius $h$ | 0,19275731 |
| **Nachbarschaftssuche** | |
| Suchverfahren | Spatial Hashing |

## 12.3. Kollisionserkennung

Die Evaluierung der Kollisionserkennung zielt darauf ab, zu ermitteln ob eine Kollision eines Partikels mit einer Wand als solche erkannt wird. Desweiteren wird überprüft, ob sich der Partikel nach der Kollision auf der richtigen Position ínnerhalb der Wandgeometrie befindet. Schließlich wird der aktuelle Partikelrichtungsvektor auf seine Korrektheit überprüft.

### 12.3.1. Primitive Körper

Für die Evaluierung der Kollisionserkennung bei primitiven Körpern wurde die Methode collisionExact der Klasse sphere mit der Unit Testing der Klassenbibliothek Qt getestet. Es wurde eine Sphäre mit einem Radius von 1 und einem Mittelpunkt mit den Koordinaten (1, 1, 1) initialisiert. Es wurde zusätzlich ein Partikel mit der aktuellen Position (2.1, 1, 1) und der vorherigen Position (1.9, 1.8, 1) initialisiert. Die Stosszahl wurde auf 0.8 gesetzt. Das Testergebnis bestätigte die erwartete neue Partikelposition (2.0, 1, 1) und den Partikelrichtungsvektor (-0.16, -0.8, 0).

### 12.3.2. Meshes

Die Evaluierung der Kollisionserkennung mit Meshes besteht aus der Evaluierung der Kollisionserkennung selbst und der Evaluierung der Kollisionsbehandlung. Für die Kollisionserkennung wurden die Softwarebibliotheken „Coldet“ und „Bullet“ verwendet (siehe Kapitel 9.2.2). Hierbei hat sich „Coldet“ als eine überschaubare und leicht zu verwendende Bibliothek erwiesen, die jedoch als Suchstruktur nur verschachtelte „Bounding-Boxen“ zur Verfügung stellt. Die umfangreichere Bibliothek „Bullet“ hingegen erforderte eine deutlich längere Einarbeitungszeit,

Tabelle 12.6.: Testfall Viskosität

| **Geometrie** | |
|---|---|
| Maße | 1x2x2 m |
| Partikel | 4000 |
| Partikelanordnung | reguläres Gitter |
| Rasterabstand | 0,1 m |
| **Kräfte** | |
| Druckkraft | |
| Viskosität | 0,05 $Pa \cdot s$; 1 $Pa \cdot s$; 10 $Pa \cdot s$ |
| Gravitation | (0;-9,81;0) $\frac{m}{s^2}$ |
| **Integration** | |
| Methode | Leap Frog |
| Zusatz | XSPH( $\varepsilon = 0,5$ ) |
| Zeitschritt | 0,001 s |
| **Physikalische Parameter** | |
| Gaskonstante | 5 J |
| Ruhedichte $\varrho_0$ | 998,28998 $\frac{kg}{m^3}$ |
| Einflussradius $h$ | 0,19275731 m |
| **Nachbarschaftssuche** | |
| Suchverfahren | Spatial Hashing |

dafür ist hier eine wesentlich effizientere Realisierung der Suchdatenstruktur mit verschachtelten „Bounding-Boxen“ vorhanden. Bei der Verwendung von „Coldet“ ergibt sich ein linearer Anstieg der Laufzeit im Bezug auf die Anzahl der Partikel. Jedoch ist die Streuung der Laufzeit bezogen auf die Anzahl der Partikel gering. Bei „Bullet“ hingegen ist eine deutlich stärkere Streuung der Laufzeit vorhanden. Jedoch ist die Berechnung im Durchschnitt schneller als bei „Coldet“. Eine empirische Bewertung zeigt, dass „Coldet“ und „Bullet“ Kollisionen zuverlässig erkennen, wenn die Strecke zwischen der neuen und alten Partikelposition ein Dreieck des Meshes schneidet. Es kann allerdings vorkommen, dass sich Partikel so nah an den Dreiecken eines Meshes befinden, dass aufgrund von Rundungsfehlern beide Endpunkte der Strecke so liegen, dass keine Kollision vorhanden ist, obwohl es eigentlich eine Kollision geben müßte. Dieser Fehler ist auf die endliche Genauigkeit von Gleitkommazahlen zurückzuführen. In solchen Fällen kann es vereinzelt dazu kommen, dass Partikel ein Mesh durchdringen, ohne das eine Kollision erkannt wird. Deshalb sollten keine zu klein skalierten Meshes verwendet werden, da sich hierdurch die Wahrscheinlichkeit für falsche Berechnungen erhöht, insbesondere wenn sich das Mesh nicht in der Nähe des Koordinatenursprungs befindet. Eine „Primitive Box“ ist in allen Fällen durch die wesentlich geringere Laufzeit der Verwendung von Meshes vorzuziehen, wenn eine solche Geometrie verwendet werden soll. Insgesamt eignen sich Meshes zur Modellierung von komplexen Kollisionsobjekten. Da eine Schwankung der Laufzeit in den jeweiligen Iterationen durch eine stärkere Streuung der Laufzeit in der Kollisionserkennung zu keinem sichtbaren Problemen führt, ist die Verwendung von „Bullet“ vorzuziehen. Es sollten Meshes, bezogen auf die Laufzeit, nicht als Ersatz für die verwendeten geometrischen Primitiven verwendet werden (siehe Abbildung 12.11).

Für die Evaluierung der Kollisionsbehandlung wurde das Verhalten der Partikel (siehe Kapitel 8.2) bei verschiedenen Testfällen mit dem gewünschten Verhalten verglichen. Hierbei ist zu

Tabelle 12.7.: Testfall Brechender Damm

| | |
|---|---|
| **Geometrie** | |
| Maße | 1x2x2 |
| Partikel | 8911 |
| Partikelanordnung | reguläres Gitter |
| Rasterabstand | 0,1 |
| **Kräfte** | |
| Druckkraft | |
| Viskosität | 1 |
| Gravitation | (0;-9,81;0) |
| **Integration** | |
| Methode | Leap Frog |
| Zusatz | XSPH( $\varepsilon = 0,5$ ) |
| Zeitschritt | 0,0001 |
| **Physikalische Parameter** | |
| Gaskonstante | 30 |
| Ruhedichte $\varrho_0$ | 998,28998 |
| Einflussradius $h$ | 0,1215 |
| **Nachbarschaftssuche** | |
| Suchverfahren | Spatial Hashing |

erkennen, dass die Partikel je nach eingestellter Dämpfung des Materials des Kollisionsobjektes elastisch oder unelastisch kollidieren können. Bei der elastischen Kollision ist eine Bewegung in Richtung des Reflektionsvektors bei gleichzeitigem Einwirken der Kräfte, wie z.B. der Gravitation, zu erkennen. Bei einer unelastischen Kollision ist ein solches Verhalten nicht vorhanden. In diesem Fall bekommen die Partikel eine Geschwindigkeit von Null und werden auf die Position nahe dem Kollisionspunkt gesetzt. Allerdings können Kräfte, beispielsweise die Gravitation, in den weiteren Iterationen auf das Partikel einwirken, so dass sich eine Bewegung entlang des Dreiecks des jeweiligen Kollisionsobjektes ergeben kann (siehe Abbildung 12.12(a) und Abbildung 12.12(b)).
Im Zusammenhang mit Meshes ist bei der Evaluierung aufgefallen, dass einzelne Partikel die Dreiecke der Meshes, bei der Verwendung von „Coldet“, durchdringen konnten. Zur möglichen Behebung dieses Problems wurden Teile des Quellcodes von „Coldet“ untersucht. Später wurde jedoch erkannt, dass dieses Problem bei „Bullet“ in gleicher Weise auftritt. Schließlich konnte der Fehler auf die Verwendung von „OpenMP“ und der fehlenden Threadsicherheit von „Coldet“ und „Bullet“ zurückgeführt werden. Außerdem wurden zu klein skalierte Meshes in den Szenen und daraus enstehende Rundungsfehler bei Fließkommazahlen als Fehlerquelle erkannt. Da Partikel bei starken Dämpfung beliebig nahe an die Dreiecke der Geometrie gelangen können und dies die Wahrscheinlichkeit von Rundungsfehler erhöht, sollte die Verwendung von zu starker Dämpfung vermieden werden.

## 12.4. Zeitintegration

Vel omnis malis putent an, iriure dolores ne vix, nec ad inermis tincidunt intellegam. Has at diceret molestiae theophrastus, vim id pertinax evertitur vulputate, delectus delicata

mediocrem no nam. Mei id quaeque dolorem maiestatis. Duo partem partiendo eu, quaeque alterum singulis ea ius, et assum dicit vituperatoribus per. Ex elitr gubergren incorrupte qui, illud errem percipit ut mel, movet dicunt inimicus ut has. Pro ei vocibus minimum, eu munere libris impedit est. In quo affert accumsan iudicabit, libris labore sea ad.

## 12.5. Diskussion

Wie aus der Evaluierung der verschiedenen Teilbereiche von YASPHS hervorgeht, setzte die Projektgruppe bei der Lösung von Problemen nicht nur vorgegebene Verfahren ein, sondern entwickelte auch eigene Methoden. Besonders deutlich wird dies im Bereich der Nachbarschaftssuche. Eine von [Hje06] vorgeschlagene Hashfunktion stellte sich als weniger geeignet für die Fälle der Projektgruppe heraus. Da die Projektgruppe bei der Nachbarschaftssuche einen großen Fokus auf Gitterzellen als Suchstruktur legte, wurde eine neue Hashfunktion erstellt und in die Simulation eingebunden, welche genau auf diese Suchstruktur angepasst wurde (siehe Kapitel 5.2.2). Des weiterem wurden Nachbarschaftssuchverfahren weiter bzw. neu entwickelt. Das bereits verfügbare „Spatial Hashing“ wurde als Referenzverfahren eingestuft und daher zum Laufzeitvergleich mit den, von der Projektgruppe, weiterentwickelten Verfahren verwendet. Da das Spatial Hashing nicht auf die eventuell eintretenden Sonderfälle (siehe Kapitel 12.1.1), während der YASPHS Simulation, optimiert ist, kann in speziellen Fällen, eine höhere Laufzeit auftreten. Basierend auf dem „Spatial Hashing“ wurden weitere Nachbarschaftssuchverfahren entwickelt, welche auf diese speziellen Fälle optimiert wurden und somit unter bestimmten Voraussetzungen schneller als das „Spatial Hashing“ sind (siehe Kapitel 5). Besonders erwähnenswert ist weiterhin das „Symmertry Aware Grid“-Verfahren. Es wurde als letzten Suchverfahren von der Projektgruppe entwickelt und profitierte somit von den Erfahrungen aus den Vorgänger Verfahren., was sich in der Tatsache, dass es das schnellste Verfahren ist widerspiegelt. Das Thema „Nachbarschaftssuche“ ist bei weitem noch nicht erschöpft. So können weitere Hashfunktionen erforscht und entwickelt werden, welche möglichst eine geringe Streuung mit einer schnellen Laufzeit kombiniert. Andere Datenstrukturen stellen eine weitere Herausforderung dar. Sollte sich das Interesse dahingehend entwickeln Partikel mit unterschiedlicher Smootihng-Length zu berechnen, so sind Datenstrukturen wie Suchbäume, insbesondere KD-Bäume, im Bereich der Nachbarschaftssuche, eine potenzielle Alternative zu dem Gitterverfahren. Bei der Evaluierung der Kraftberechnung bezog sich der Fokus zunächst auf die Parameter Dichte und Druck. Es wurden mehrere Simulationen durchgeführt, in denen überprüft wurde, ob das Variieren von physikalischen Parametern das plausible Verhalten des Fluids zur Folge hatte. Hierfür wurden zur Überprung der Dichte und des Drucks zwei verschiedene Körper gewählt, welche mit Partikeln befüllt wurden. Das Ergebnis dieser Simulationen war, dass es nur sehr geringe Abweichungen zu den Prognosen und den Simulationergebnissen gab. Für die Kompressibilität und die Viskosität wurden bei der Evaluierung Simulationen mit den bekannten Szenarien „fallender Volumenkörper“ (Abbildung 12.5) und „brechender Damm“ (Abbildung 12.9) durchgeführt. Interessant ist hierbei das Verhalten des Fluids bei Varriation der Gaskonstante. Anhand des Verhaltens wurde die These aufgestellt, dass die Kompressibilität im Verhältnis zur Gaskonstante steht. Bei dem Simulationstest mit dem eine Verbindung zwischen Viskosität und Fluidbewegung hergestellt werden sollte, stellte sich heraus, dass eine höhere Viskosität eine kleinere Fluidausbreitung bewirkte. Hier wäre es Interessant das Verhältnis zwischen Viskositätswert und Ausbreitungsgeschwindigkeit zu ermitteln. Beim setzen der Initialwerte stellte sich heraus, dass

die Smoothinh-Length abhängig von dem gewählten Partikelabstand ist. In welche Relation diese beiden Werte genau stehen, wäre ein interessantes Forschungsgebiet. Für die Evaluierung der Kollisionserkennung wurden Test mit primitiven Körpern und mit Meshes durchgeführt. Die Evaluierung der Kollisionserkennung mit primitiven Körpern wurde recht simpel gehalten. Es wurde getestet ob bei einer Kollision die Partikel den erwarteten Verhalten entsprachen. Bei den Meshes wurde eine detaillierte Evaluierung vorgenommen. Die Kollisionserkennung wurde zunächst separiert zwischen Kollisionserkennung und Kollisionsbehandlung. In beiden Bereichen wurde Überprüft ob das Verhalten der Partikel bei Kollisionen, den erwarteten Verhalten entsprach. Die Softwarebibliotheken „Bullet“ und „Coldet“ wurden für die Kollisionserkennung verwendet und verglichen. Bei der Kollisionsbehandlung wurde zwischen einer elastischen und unelastischen Kollision unterschieden. Es stellte sich heraus, das bei der Kollisionserkennung mit Meshes einzelne Partikel die Dreiecke durchdringen. Dies ist auf die fehlende Threadsicherheit der Softwarebibliotheken, der Verwendung von „OpenMP“ und Rundungsfehlern bei den Fließkommazahlen zurückzuführen. Der Bereich der Kollisionserkennung mit primitiven Körpern sollte um weitere primitive Körper erweitert werden, da diese Aufgrund ihrer geringeren Laufzeit den Meshes vorgezogen werden sollten. Für die Kollisionserkennung mit Meshes ist sicherlich interessant, das Verhalten der Partikel in Verbindung einer Softwarebibliotheken mit Threadsicherheit zu beobachten.

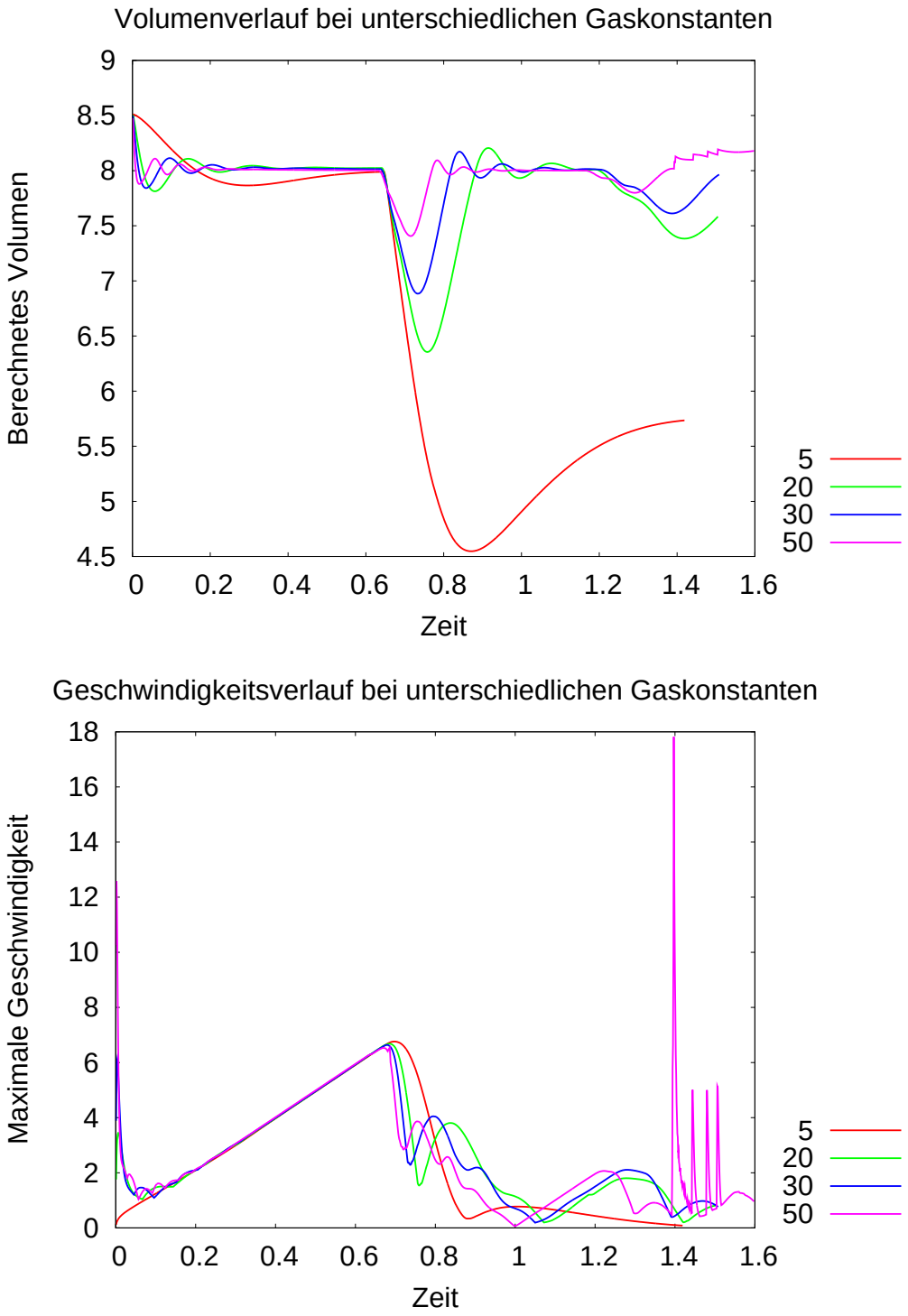



Abbildung 12.6.: Testfall Kompressibilität: Fallender Volumenkörper, Volumen (oben) und Geschwindigkeit (unten)

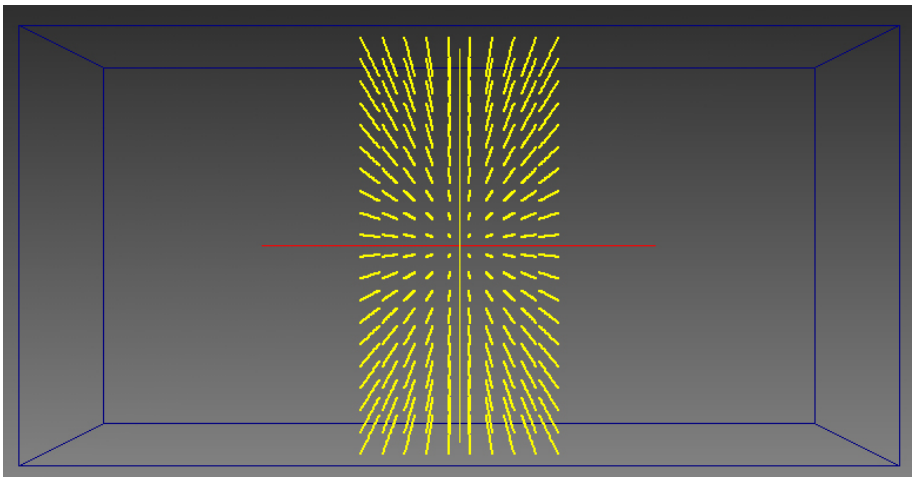


Abbildung 12.7.: Testfall Viskosität: Ausbreitender Volumenkörper, Initialstatus

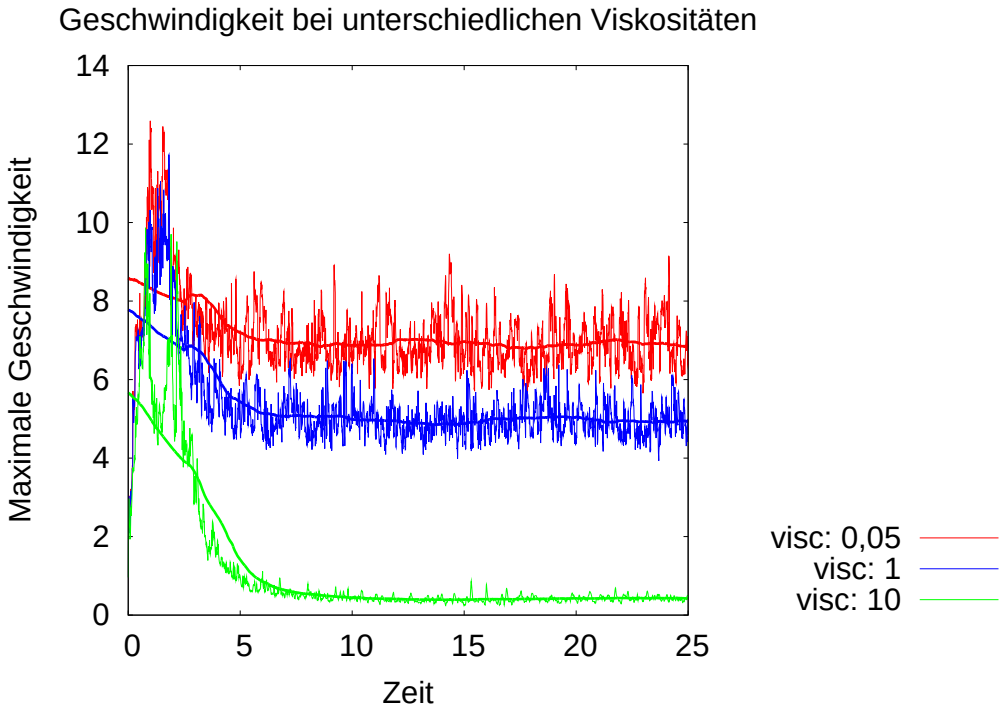



Abbildung 12.8.: Testfall Viskosität: Ausbreitender Volumenkörper, Volumen (oben) und Geschwindigkeit (unten)

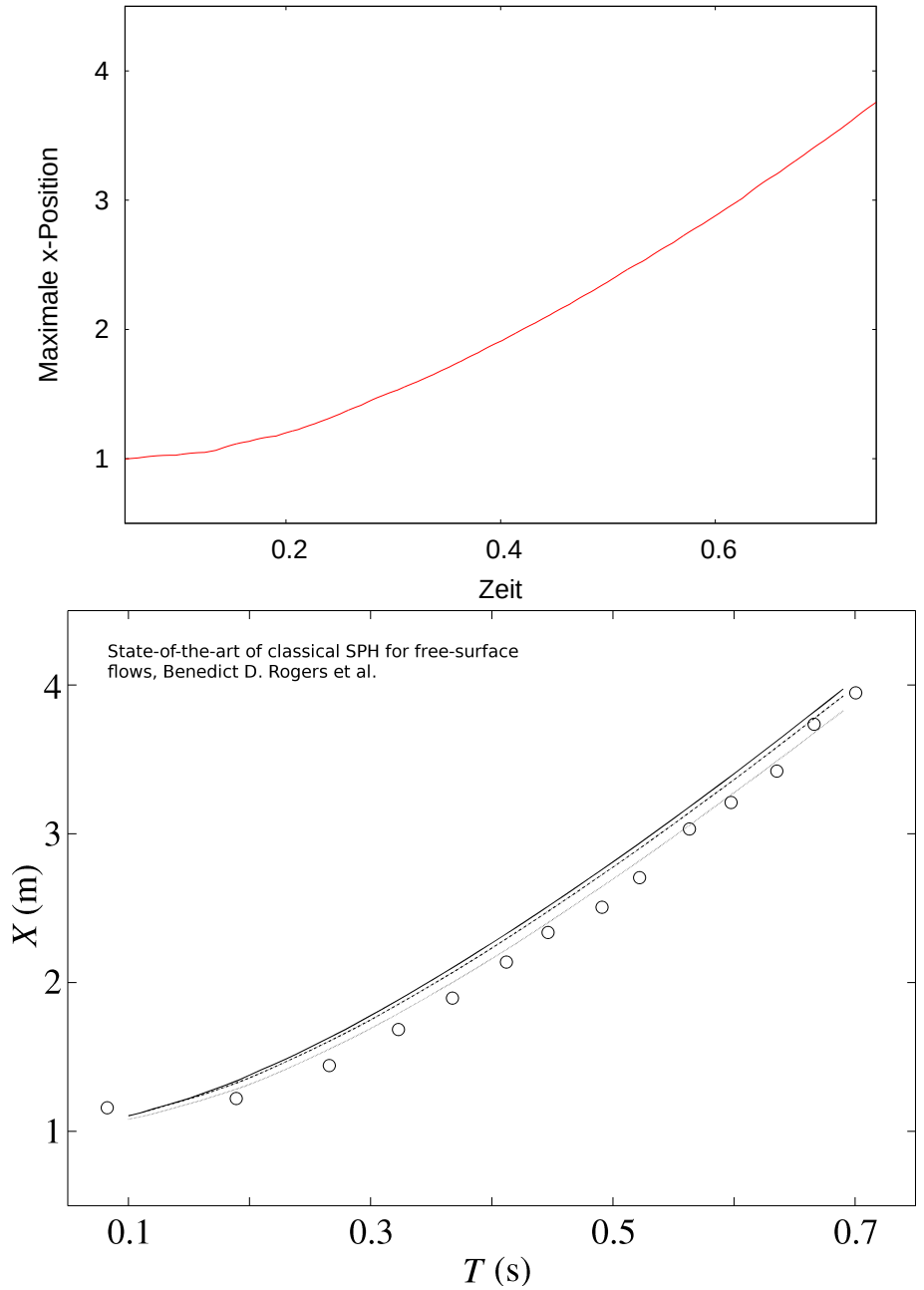



Abbildung 12.9.: Testfall Ausbreitender Volumenkörper

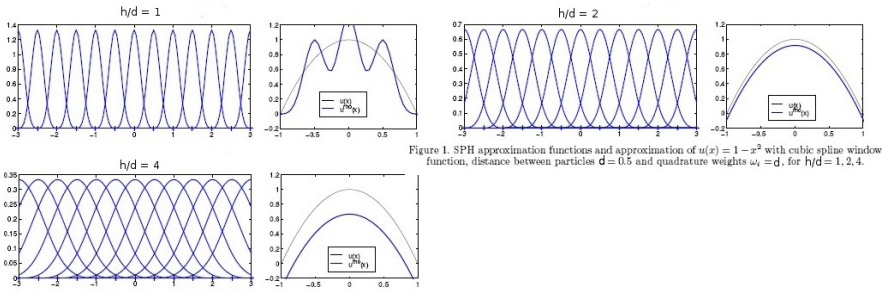



Figure 1. SPH approximation functions and approximation of $u(x) = 1 - x^2$ with cubic spline window function, distance between particles $d = 0.5$ and quadrature weights $\omega_i = d$, for $h/d = 1, 2, 4$.

Abbildung 12.10.: Simulation Fallender Tropfen, Quelle: [Ben07]

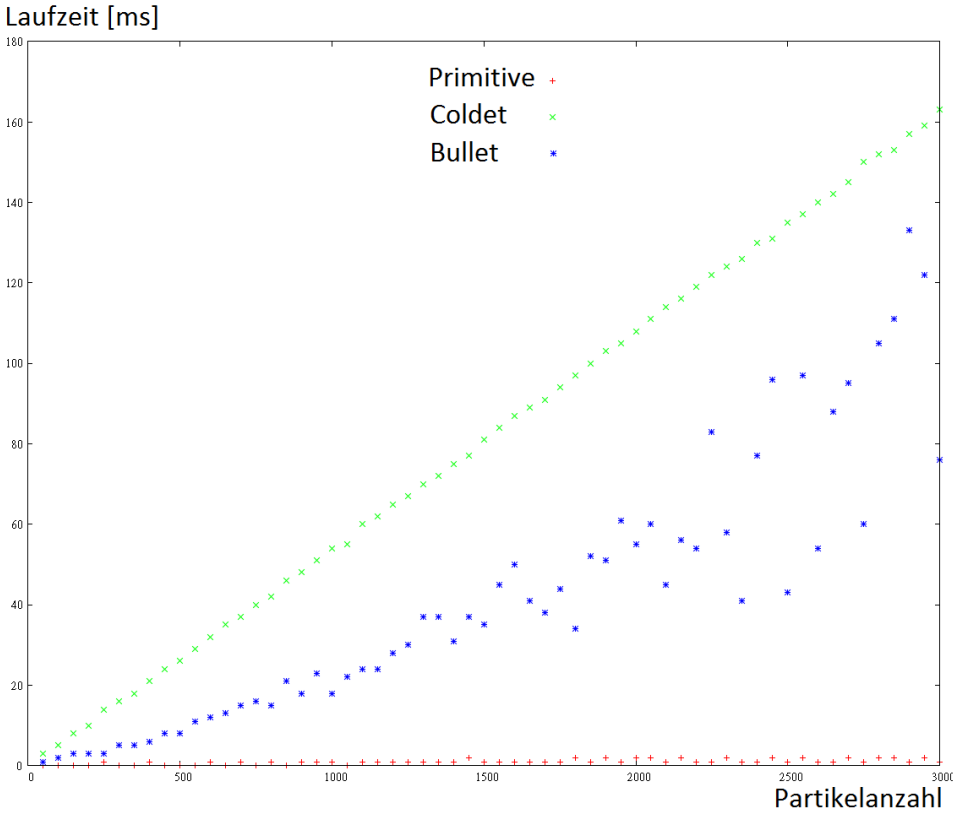
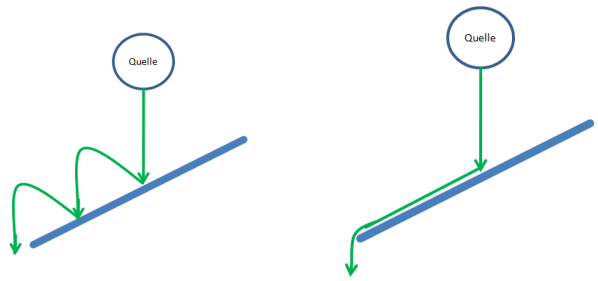



Abbildung 12.11.: Vergleich der Laufzeit der Kollisionserkennung mit einer als Mesh definierten Box von „Coldet“ (grün) mit „Bullet“ (blau) und einer Primitiven Box (Rot)




(a) Darstellung des Verhaltens der Partikel (grüne Pfeile) bei niedriger Dämpfung. (elastische Kollision)

(b) Darstellung des Verhaltens der Partikel (grüne Pfeile) bei hoher Dämpfung. (unelastische Kollision)

Abbildung 12.12.: Darstellung der unterschiedlichen Dämpfung von Partikeln, die in einer Quelle erzeugt werden, und durch die Gravitation auf eine schräge Ebene treffen.

# 13. Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse der Projektgruppenarbeit nochmals erläutert und Vorschläge hinsichtlich der Weiterentwicklung dieses Softwaresystems gemacht.

## 13.1. Zusammenfassung

Ziel dieser Projektgruppe war es, ein Softwaretool zur Simulation des Fluidverhaltens zu entwickeln, das die physikalischen Eigenschaften des Flüssigkeit an verschiedenen Stellen eines Raumes mit Hilfe der Lagrangemethode SPH berechnet und die Ergebnisse dieser Berechnung bezüglich bestimmter Partikelattribute ausgibt. Dieses Softwaretool sollte eine virtuelle Werkstatt darstellen, die dem Benutzer die Möglichkeit gibt, die Berechnung der Materialeigenschaften des Fluids individuell zu gestalten. Ein weiteres Ziel der Projektgruppe war es, eine Metavisualisierung des Fluidverhaltens zu erstellen, welche während des Simulationsverlaufs eine Auskunft über den Fortschritt der Berechnung sowie den Systemzustand liefert. Ferner sollte das Softwaresystem zusätzlich in der Lage sein, eine Anwendungsvisualisierung des Fluidverhaltens in Form von realitätsnahen Snapshots zu ermöglichen. Zu diesem Zweck wurde die Projektgruppe „Smoothed Particle Hydrodynamics Lab“ im Wintersemester 2009 aus acht Teilnehmern gebildet, die die genannten Aufgaben innerhalb dem Zeitraum eines Jahres bewältigen sollte. Aus den wissenschaftlichen Arbeiten der Projektgruppenteilnehmer ist schließlich das Softwaretool YASPS entstanden, das eine softwaretechnische Lösung der genannten Aufgabenstellung realisiert.
Entwickelt wurde das Softwaretool YASPHS mit der objektorientierten Programmiersprache C++, die eine schnelle und hardwarenahe Implementierung der betreffenden Inhalte erlaubt. Die GUI wurde mit der C++-Klassenbibliothek Qt erstellt und bietet eine Vielzahl an Möglichkeiten für die Berechnung des Fluidverhaltens bezüglich der physikalischen und systemtechnischen Eingangsparameter sowie der eingesetzten algorithmischen Verfahren. Diese lässt sich ferner so konfigurieren, dass sie dem jeweils vorliegenden Anwendungsproblem entspricht.
Die Metavisualisierung wurde mit der C++-Klassenbibliothek VTK erstellt, welche häufig für Visualisierungen von dreidimensionalen graphischen Inhalten eingesetzt wird. Im YASPS wird durch den Einsatz der Visualisierung die Bewegungsbahn jedes einzelnen Partikels während des Simulationsverlaufs dargestellt. Außerdem bietet das System die Möglichkeit, die Messwerte der Partikelattribute, wie z.B. Dichte und Druck, anzuzeigen. Letzteres geschieht durch das Einfärben der Partikel während der Simulation und dem zeitgleichen Aufbau einer entsprechenden Referenztabelle.
Die realitätsnahe Visualisierung des Simulationsablaufs wurde durch die Möglichkeit realisiert, die vorliegenden Inhalte mit Hilfe von OBJ-Files zu exportieren und durch den Einsatz externer Software z.B. Maya in Form von Snapshots auszugeben.
Die Systemdynamik wurde durch die drei Zeitintegrationverfahren Eulerverfahren, Leap-Frog-Verfahren und Runge-Kutta-Verfahren implementiert, die auf Anwenderoberfläche anwählbar sind.

Die physikalische Kräfteberechnung in YASPHS basiert auf den wissenschaftlichen Erkenntnissen im Bereich der Hydrodynamik von Micky Kelager, die unter dem Titel „Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics“ publiziert wurden. Die Kräfteberechnung nach der SPH-Methode benötigt für die SPH-Summierung ein Nachbarschaftssuchverfahren um die Partikel, die sich im Einflussradius des aktuellen Partikels befinden, zu ermitteln. Im YASPHS stehen neun solcher Verfahren zur Verfügung, die ebenso wie die Zeitintegrationsverfahren zu Beginn der Simulation auf der Anwenderoberfläche einstellbar sind. Dabei handelt es sich um die folgenden Verfahren: Naives Verfahren, Spatial-Hashing-, Spatial-Hashing-Automatic-Smoothing-Length-, Union-Spatial-Hashing-, Capped-Spatial-Hashing-, Spatial-Hashing-Vertex-, Optimized-Spatial-Hashing-, Reguläres-Gitter-mit-fester-Speicherzuordnung- und Symmetry-Aware-Grid-Verfahren.
Für die Kollisionserkennung wurde ein analytisches Verfahren eingesetzt, welches den Vorteil einer geringen Rechenzeit hat. Bei diesem Verfahren wird die Partikelposition mit der formalen Darstellung der Raumposition des Körpers verglichen und im Falle einer Kollision reflektiert. Es können Kollisionen mit beliebigen Körpern erkannt werden, die allerdings mit einem externen Programm als Meshes erstellt und in YASPHS importiert werden müssen.
Es wurden die Aspekte der physikalischen Kräfteberechnung, der Nachbarschaftssuche sowie der Kollisionserkennung des Softwaresystems anhand von Testszenarien getestet und evaluiert. Die Testergebnisse waren innerhalb der akzeptablen Toleranzgrenze (siehe hierzu Kap. 12). Es konnte allerdings beobachtet werden, dass es bei einer steigenden Partikelanzahl zu Schwierigkeiten kommen kann, einen geeigneten Einflussradius für die Nachbarschaftssuche zu finden. Dies kann wiederum zu erheblichen Ungenauigkeiten bei den Messungen der Partikelattribute zur Folge haben.

## 13.2. Ausblick

YASPHS ist ein voll funktionsfähiges Softwaresystem zur hydrodynamischen Simulation in einem abgeschlossenen Bereich mittels der SPH-Methode. Diese kann im medizinischen Bereich bei der Simulation von Blutzirkulationen in menschlichen Körpern eingesetzt werden. Darüber hinaus könnte in Zukunft YASPHS für weitere Anwendungsbereiche in Anspruch genommen werden.
An bestimmten Stellen innerhalb des Systems müssen weitere Anstrengungen hinsichtlich der Optimierung unternommen werden, um die Genauigkeit der Messungen der Materialeigenschaften des Fluids zu erhöhen. Ferner könnten die nachfolgend aufgeführten Anforderungen in das erarbeitete Softwaresystem integriert werden, um seine Einsatzmöglichkeiten zu erweitern:

- die Erstellung eines Modells zur Einbindung der Temperatur als zusätzlicher physikalischer Parameter,
- die Bestimmung eines geeigneten Einflussradiuses für die Simulationen einer sehr großen Partikelanzahl,
- die Erarbeitung eines Verfahrens zur Berücksichtigung von den SPH-Randbedingungen,
- die Berücksichtigung zusätzlicher Turbulenzmodelle zur grundsätzlichen Turbulenzbehandlung durch die Navier-Stokes-Gleichungen,
- die Implementierung von Streamlines zur Erweiterung der Metavisualisierung,

- die Integration einer hydrodynamischen Simulation für Flüssigkeiten, welche sich innerhalb von beweglichen Wandgeometrien befinden.

Das Einfügen dieser zusätzlichen Funktionalitäten würde die Arbeitsweise des Softwaresystems YASPHS verbessern und ihre Anwendungsbereiche erweitern.

# Literaturverzeichnis


[AP98] Ascher, U. ; Petzold, L.: *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, 1998. – ISBN 0–89871–412–5

[BA95] Benz, W ; Asphaug, E.: Simulations of brittle solids using smooth particle hydrodynamics. In: *Comput. Phys. Commun.* 87 (1995), S. 253– 265

[Ben88] Benz, W.: Applications of Smoothed Particle Hydrodynamics (SPH) to Astrophysical Problems. In: *Computer Physics Communications* 48 (1988), S. 97–105

[Ben89] Benz, W: Smooth Particle Hydrodynamics: A Review / Harvard-Smithsonian Center for Astrophysics. 1989. – Forschungsbericht

[Ben07] Benjamin, G.: *Smoothed Particle Hydrodynamics*, Universität Bonn, Diplomarbeit, 2007

[BGM08] Biddiscombe, J. ; Graham, D. ; Maruzewski, P.: Visualization and analysis of SPH data. In: *ERCOFTAC Bulletin* 76 (2008), S. 9–12

[BK00] Bonet, J. ; Kulasegaram, S.: Correction and stabilization of smooth particle hydrodynamic methods with applications in metal formingsimulations. In: *Int. J. Numer. Methods Eng.* 47 (2000), S. 1189 – 1214

[Dar07] Dario, G.: Eine Einführung in das Navier-Stokes-Problem (HA). (2006/2007), S. 1–6. – Herleitung: Bewegungsgleichung und Kontinuitätsgleichung

[Flu] *Fluids.* `http://www.rchoetzlein.com/eng/graphics/fluids.htm`

[GM77] Gingold, R. A. ; Monaghan, J.: Smoothed Particle Hydrodynamics: Theory An Application to Non-spherical stars. In: *Royal Astronomical Society* 181 (1977), S. 375–389

[GS98] Gutfraind, R. ; Savage, S. B.: Flow of fractured ice through wedge-shaped channels: Smoothed particle hydrodynamics and discrete-element simulations. In: *Mech. Mater.* 29 (1998), S. 1–17

[GSDE99] Garcia-Senz ; D.Bravo ; E.Woosley, S. E.: Single and multiple detonations in white dwarfs. In: *Astron. Astrophys.* 349 (1999), S. 177 – 188

[Har63] Harlow, F. H.: The particle-in-cell method for numerical solution of problems in fluid dynamics. In: *Symposia in Applied Mathematics*, 1963

[HEF07] Heinz, S. ; Ewald, K. ; Frank, K.: *Strömungslehre.* 3rd. Walter de Gruyter, 2007. – ISBN 3110189720

[Hje06] Hjelte, N.: *Smoothed Particle Hydrodynamics on the Cell Broadband Engine*, Umeå University, Diss., 2006. – 22–26 S.

[HP99] Hultman, J. ; Pharayn, A.: Hierarchical, dissipative formation of elliptical galaxies: Is thermal instability the key mechanism? Hydrodynamical simulations including supernova feedback multiphase gas and metal enrichment in CDM: Structure and dynamics of elliptical galaxies. In: *Astron. Astrophys.* 347 (1999), S. 769–798

[Kel06] Kelager, M.: *Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics*. Kopenhagen, Dänemark, Universität Kopenhagen, Diplomarbeit, 2006

[LC87] Lorensen, W. E. ; Cline, H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In: *SIGGRAPH Comput. Graph.* 21 (1987), Nr. 4, S. 163–169

[Lee00] Lee, W. H.: Newtonian hydrodynamics of the coalescence of black holes with neutron stars. In: *Mon. Not. R. Astron. Soc.* 318 (2000), S. 606 – 624

[LL03] Liu, G. R. ; Liu, M. B.: *Smoothed Particle Hydrodynamics: a Meshfree Particle Method*. World Scientific Publishing Co. Pte. Ltd, 2003

[LLZL00] Liu, M. B. ; Liu, G. R. ; Zong, Z. ; Lam, K. Y.: Numerical simulation of underwater explosion by SPH. In: *Advances in Computational Engineering & Science* (2000), S. 1475–1480

[LP91] Libersky, L. D. ; Petschek, A. G.: Smooth particle hydrodynamics with strength of materials. In: *Advances in the Free-Lagrange Method* (1991), S. 248–257

[Luc77] Lucy, L.: A Numerical Approach to Testing the Fission Hypothesis. In: *Astronomical Journal* 82 (1977), S. 1013–1024

[May] *Maya*. `http://www.autodesk.com/maya`

[MBRA10] Moncho, G. ; Benedict, D. R. ; Robert, A. D. ; Alex, J.C.: State-of-the-art of classical SPH for free-surface flows. In: *International Association of Hydraulic Engineering and Research* (2010)

[MDM03] Müller, M. Charypar ; D. ; M., Gross: *Particle-Based Fluid Simulation for Interactive Applications*. Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation, 2003. – 154–159 S.

[MK95] Monaghan, J. J. ; Kocharyan, A.: SPH simulation of multiphase flow. In: *Comput. Phys. Commun.* 87 (1995), S. 225

[ML91] Monaghan, J.J. ; Lattanzio, J.C.: A refined particle method for astrophysical problems. In: *Astron. Astrophys.* 149 (1991), S. 135–143

[Mon89] Monaghan, J. J.: On the Problem of Penetration in Particle Methods. In: *Journal of Computational Physics* 82 (1989), S. 1–15

[Mon90] Monaghan, J. J.: Modeling the universe. In: *Proc. Astron. Soc. Aust.* 18 (1990), S. 233 – 237

[Mon92] Monaghan, J. J.: Smoothed Particle Hydrodynamics. In: *Annual Review of Astronomy and Astrophysics* 30 (1992), S. 543–574

[Mon94] Monaghan, J. J.: Simulating free surface flow with SPH. In: *J. Comput. Phys.* 110 (1994), S. 399

[Mor96] Morris, J. P.: Stability properties of SPH. In: *Publ. Astron. Soc. Aust.* 13 (1996), S. 97

[Pas] *Pasimodo*. `http://www.itm.uni-stuttgart.de/research/pasimodo/pasimodo_de.php`

[PHK95] Posch, H. A. ; Hoover, W. G. ; Kum, O.: Steady-state shear flows via nonequilibrium molecular dynamics and smooth-particle applied mechanics. In: *Phys. Rev.* 52 (1995)

[SHA95] Swegle, J.W. ; Hicks, D. L. ; Attaway, S. W.: Smoothed particle hydrodynamics stability analysis. In: *J. Comput. Phys.* 116 (1995), S. 123 – 134

[Tes03] Teschner, M.: Optimized Spatial Hashing for Collision Detection of Deformable Objects). (2003)

[Uni09] Universität Koeln (Hrsg.): *Versuch 3: Oberflätchenspannung von Flüssigkeiten*. Version: 07 2009. `http://typo3-8443.rrz.uni-koeln.de`. – Definition: Oberflächenspannung

[WMK07] Wroblewski, P. ; M., Kopec ; K., Boryczko: SPH - A comparison of neighbor search methods based on constant number of neighbors and constant cut-off radius. In: *Task Quarterly 11* 3 (2007), S. 273–283

[Yaf] *Yafaray*. `http://www.yafaray.org`

[You09] Young, P.: The leapfrog method and other symplectic algorithms for integrating Newton's laws of motion. In: *Physics* 115/242 (2009), S. 1–14

# Abbildungsverzeichnis

# A. Pflichtenheft

## A.1. Zielbestimmungen

YASPHS - Yet-Another-SPH-Simulator ist ein an der TU Dortmund am Lehrstuhl VII im Rahmen der PG 541 entwickeltes Simulationsprogramm zur approximativen Lösung hydrodynamischer Gleichgungen. YASPHS löst die hydrodynamischen Gleichungen durch eine Lagrange Methode auf Basis von SPH. Vor dem Simulationszyklus müssen physikalische Parameter gesetzt und Start- und Randbedingungen festgelegt werden. Dazu gehört das Bereitstellen eines geometrischen Körpers als Begrenzung der Simulation bzw. als möglichen Kollisionskörper, die Definition des physikalischen Modells und die Wahl geeigneter Kernels. Die Simulation erfolgt in einer Schleife, in der in einem Löser die Krafteinflüsse auf die jeweiligen Partikel und unter Einbeziehung der Navier-Stokes-Gleichungen die auf diese Partikel wirkenden Beschleunigungen berechnet werden. Wie bereits erwähnt, handelt es sich bei einigen Werten um Feldgrößen, die von den benachbarten Partikeln abhängen. Daher ist es nötig, benachbarte Partikel über eine Nachbarschaftssuche zu identifizieren, um diese der Kernelfunktion zur Verfügung zu stellen. Über geeignete Zeitintegrationsverfahren können aus den resultierenden Werten die neuen Partikelgeschwindigkeiten und -positionen berechnet werden. Eine Kollisionserkennung erlaubt die Interaktion des simulierten Fluids mit vordefinierten, geometrischen Objekten. [Kel06]
Im Laufe der Simulation ändern die Partikel ständig ihre Positionen, so dass das Fluid zu fließen beginnt. Um diesen Prozess zu verfolgen und den korrekten Ablauf der Simulation zu überwachen, steht in der GUI ein Ansichtsfenster zur Verfügung, das die aktuellen Partikelpositionen wiedergibt. Es besteht weiterhin die Möglichkeit, berechntete Metadaten wie Dichte, Druck und Geschwindigkeit darstellen zu lassen. Zur realitätsnahen Visualisierung in einem 3D-Anwendungsprogramm wie Maya steht ein 3D-Austauschformat zur Verfügung.
Der modulare Aufbau von YASPHS ermöglicht eine schnelle und einfache Erweiterbarkeit und Anpassung des Programms, so dass es in unterschiedlichen Bereichen Anwendung finden kann. In dem Testszenario „Blutflusssimulation“ in Umgebung eines Aneurysmas wird das Strömungsverhalten diverser Katheter veranschaulicht. Ein weiteres Testfeld ist ein robotergestütztes „Spritzverfahren“ zur Simulation von Oberflächenbeschichtungen. In der Abbildung fig:pneu wird der modulare Aufbau von YASPHS skizziert und die Zielbestimmungen für die Bereiche Modell, Simulation, Visualisierung und GUI festgelegt. Für jeden Bereich werden im Folgenden die Musskriterien, Wunschkriterien und Abgrenzungskriterien bestimmt.

### A.1.1. Modell

Das Modell liefert die Geometrie und die physikalischen Parameter, die für die Simualtion verwendet werden. Diese stehen zum einen über primitive Objekte zur Verfügung, zum anderen können vorliegende Netze in das Programm geladen werden.

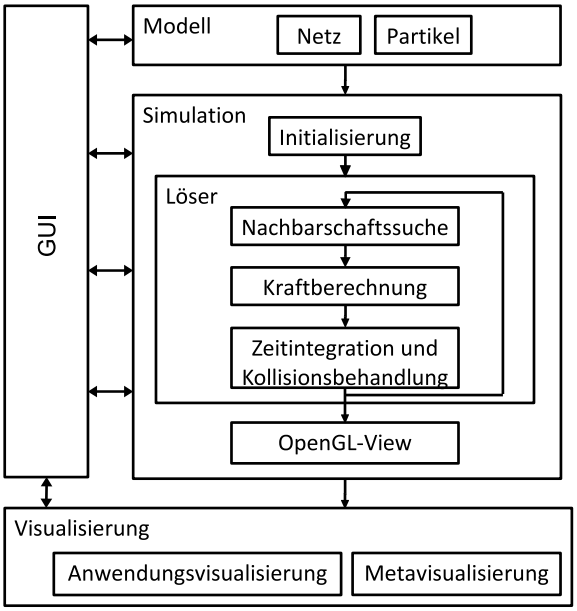



Abbildung A.1.: Modulares Aufbauschema von YASPHS

#### A.1.1.1. Musskriterien

- Statische Geometrie aus Datei laden
- Partikel aus Datei laden
- Statische Partikelzahlen
- Setzen von Quellen und Senken
- Standardgeometrie mit Partikeln füllen
- Geometrie in Maya erstellen
- Physikalische Größen materialspezifisch zuweisen an Netze (Vererbung an Partikel, Kollisionserkennung)
- Physikalische Größen materialspezifisch zuweisen an Partikel-Domänen/-Cluster
- Physikalische Systemgrößen setzen (z.B. Gravitationspunkte)
- Laden und speichern eines Simulationszustandes (z.B. in einer XML-Datei)

#### A.1.1.2. Wunschkriterien

- Dynamische Geometrie aus Datei laden (statische Animation)
- Dynamische Partikelzahlen
- Netzgeometrie mit Partikeln füllen
- Spezialpartikel erzeugen (z.B. Ghostpartikel im Rand, Abstoßungs- Attraktorzentren)

- Implementierung ohne Metrik

#### A.1.1.3. Abgrenzungskriterien

- Keine Softbody-Dynamik auf geom. Objekten (interaktive Verformbarkeit)

### A.1.2. Simulation

In der Simulation werden die Navier-Stokes-Gleichungen mit SPH ausgewertet und über Integrationsverfahren die neuen Partikelpositionen und -geschwindigkeiten berechnet. Über Kollisionserkennung findet eine Interaktion des Fluids mit angrenzenden Körpern statt.

#### A.1.2.1. Musskriterien

- Nachbarschaftssuche mit Standardverfahren, Spatialhashing und Optimized Spatial Hashing
- Kraftberechnung: Druck, Viskosität für isothermale, nahezu inkompressible Fluide
- Gravitation: Feste Implementierung
- Zeitintegration: Eulerverfahren, Leap-Frog-Verfahren
- Kollisionsbehandlung: Partikelkollision mit Geometrie

#### A.1.2.2. Wunschkriterien

- Nachbarschaftssuche über Kd-Bäume
- Kraftberechnung: modulare Kraftimplementierung, Oberflächenspannung und Auftriebskräfte vorimplementiert
- Zeitintegration: Verfahren höherer Ordnung, Mehrschrittverfahren, XSPH
- Kollisionsbehandlung: Alternative Partikelkollisionsverfahren (Wand- und Geisterpartikel)
- Geometrie-Geometriekollision
- Physikalische Berechnungen auf der GPU
- Multicoreunterstützung

#### A.1.2.3. Abgrenzungskriterien

- Berechnung erfolgt nicht in Echtzeit
- Ausschließlich explizite Zeitintegrationsverfahren

### A.1.3. Visualisierung

Die Visualisierung der Partikel und Metadaten erfolgt Programmintern über OpenGL und VTK. Zur Visualisierung werden die aktuellen Partikelpositionen in ein 3D-Austauschformat gespeichert, so dass die realitätsnahe Visualisierung über einen exteren Raytracer erfolgen kann. Die Speicherung von Metadaten in ein Austauschformat wird angestrebt.

#### A.1.3.1. Musskriterien

- Anwendungsvisualisierung über eine Schnittstelle (Dateiaustauschformat COLLADA) zu einer 3D-Applikation (z.B. Maya)
- Metavisualisierung von ausgewählten physikalischen Größen mit VTK
- Aktueller Simulationsfortschritt visuell über OpenGL

#### A.1.3.2. Wunschkriterien

- Volumenvisualisierung (Partikelhülle bzw. Volumenwolke)
- Metavisualisierung von Datenstrukturen (z.B. für Nachbarschaftssuche benachbarte Partikel hervorheben)

#### A.1.3.3. Abgrenzungskriterien

- Keine Echtzeitvisualisierung

### A.1.4. GUI

Das Modell und die physikalischen Einstellungen können vollständig über die GUI gesteuert werden. Die internen Visualisierungsverfahren werden hier ebenfalls bereitgestellt.

#### A.1.4.1. Musskriterien

- Sämtliche Muss-Funktionen in der GUI verfügbar
- OpenGL-View
- Setzen physikalischer Parameter
- Modulverwaltung
- Modell laden
- Simulation starten/beenden

#### A.1.4.2. Wunschkriterien

- Partikelanzahl festlegen
- Hilfe zu den Funktionen
- Speichern/Laden von Sitzungen

## A.2. Produkteinsatz

### A.2.1. Anwendungsbereiche

YASPHS ist ein Programm, das zur Lösung hydrodynamischer Gleichungen den Ansatz der SPH-Methode verwendet. Somit findet es in Bereichen Anwendung, die ihre Berechnungen auf Grundlage dieser Lagrange-Methode durchführen. Der modulare Aufbau ermöglicht die Erweiterbarkeit des Programms und erlaubt somit die flexible Anpassung an gegebene Problemstellungen. Mit diesen Eigenschaften deckt YASPHS ein breites Spektrum möglicher Einsatzgebiete ab, wie u.a. den Maschinenbau, die Spritztechnik, die Medizin, die Biologie und die Luft- und Schifffahrt.

### A.2.2. Zielgruppen

Vorraussetzung zum Betrieb von YASPHS sind technisch versierte Mitarbeiter und Angestellte, die sich durch die nachfolgend aufgeführten Kenntnisse auszeichnen:

- Grundkenntnisse im Umgang mit Computersystemen
- Modellierung der Szene in einem geeigneten 3D-Programm (vorzugsweise Maya)
- Kenntnis physikalischer Zusammenhänge aus den Bereichen der Hydrodynamik
- Verständnis zur Beschreibung des Problems aus dem Kernbereich in SPH

### A.2.3. Betriebsbedingungen

Die Mindestanforderungen an YASPHS sind:

- CPU: 2GHz
- RAM: 4GB
- HDD: 1TB
- Videocard: NVidia GeForce 1024MB, 520MHz

Vor Durchführung der Simulation muss YASPHS das geometrische Modell bereitgestellt und geeignete Parameter gesetzt werden.

## A.3. Produktübersicht

Im Allgemeinen soll YASPHS die Strömungssimulation von Partikeln in der Umgebung eines starren Modells, das durch eine Netzstruktur gegeben ist, ermöglichen.

**Anwendungsfall Medizin:** Im speziellen Anwendungsfall Medizin wird die Strömung von Blut durch eine Ader, in die ein Katheter eingesetzt wurde, simuliert. Wahlweise können Ablagerungen oder Aneurysmen eingearbeitet werden. Durch Druckvisualisierungen können die Einflüsse des Katheters visualisiert und dieser entsprechend optimiert werden.

**Anwendungsfall industrielle Robotik:** In einem weiteren Anwendungsfall aus dem Bereich industrielle Robotik wird das Programm an einer Spritzsimualtion getestet, die den Lackiervorgang mit einer am Roboter befestigten Spritzpistole simuliert.

## A.4. Produktfunktionen

Die folgende Auflistung liefert eine Übersicht der bereitgestellten Funktionen. Funktionen sind mit **/F *Funktionsnummer*** eindeutig gekennzeichnet. Ein nachgestelltes **W** kennzeichnet ein Wunschkriterium:

**Modell:**

- **/F100/ Statische Geometrie laden**: Laden von vorgefertigten statischen Geometriedaten aus einem Standard 3D-Format wie z.B. Collada.
- **/F101W/ Dynamische Geometrie laden**: Laden von vorgefertigten dynamischen Geometriedaten aus einem Standard 3D-Format wie z.B. Collada.
- **/F102/ Partikel laden**: Partikel aus einer Datei laden.
- **/F103/ Quellen und Senken**: Setzen von Partikelgeneratoren und -Abflüssen.
- **/F104W/ Partikel dyn. hinzufügen**: Partikel dynamisch während der Simulation hinzufügen.
- **/F105/ Stand. Geometrie hinzufügen**: Einfügen von Standard-Geometrien in die Simulation.
- **/F106/ Füllen der Stand. Geometrie**: Standard-Geometrien mit festgelegter Anzahl Partikeln füllen.
- **/F107/ Physikalische Größen (an Partikel) zuweisen**: Materialspezifische Zuweisung Physikalischer Größen an Partikel. Die Zuweisung unterscheidet zwischen Partikelclustern und einzelnen Partikeln.
- **/F108W/ Physikalsiche Größen (an Netze) zuweisen**: Materialspezifische Zuweisung Physikalischer Größen an Netze.
- **/F109W/ Netzgeometrie füllen**: Füllen der Netzgeometrie mit Partikeln.
- **/F110W/ Spezial-Partikel erzeugen**: Hinzufügen von ausgezeichneten Partikeln.
- **/F111W/ Modell speichern**: Speichern von Benutzereinstellungen und Modell.

**Simulationsparameter:**

- **/F200/ Kernelfunktion auswählen**: Es stehen diverse Kernelfunktionen zur Auswahl bereit.
- **/F201/ Physikalische Systemgrößen setzen**: Einstellen physikalischer Parameter wie Materialeigenschaften.
- **/F202/ Nachbarschaftssuche festlegen**: Verschiedene Suchverfahren stehen zur Auswahl bereit.
- **/F203/ Zeitintegration festlegen**: Verschiedene Integrationsmethoden stehen zur Auswahl, deren Parameter (z.B. Schrittweite) festgelegt werden kann.
- **/F204W/ Kollisionsbehandlung festlegen**: Auswahl zwischen diversen Kollisionsbehandlungen.
- **/F205/ Auswahl von Standardsimulationstypen**: (z.B. schnelle bzw. genaue Simulation) als Grundlage zum Aufbau einer komplexeren Simulation. Vorgefertigte, neu definierbare Simulationstypen sind wählbar.

**Benutzerschnittstelle:**

- **/F300M/ Steuerung über die GUI**: Modell sowie Simulationsparameter werden über die GUI geladen bzw. modifiziert.
- **/F301W/ Festlegung der Maßeinheit**: Möglichkeit zwischen europäischen und amerikanischen Maßeinheiten für die Ausgabe zu wählen.
- **/F302/ Simulationsstand**: Mittels OpenGL-View kann der aktuelle Zustend der Simulation überprüft werden. Gespeicherte Metadaten können geladen und visualisiert werden.

**Ergebnisausgabe:**

- **/F400/ Realitätsnahe Darstellung**: Anzeige der Simulation als realitätsnahe Animation.
- **/F401/ Metavisualisierung**: Bestimmter physikalischer Größen.
- **/F402/ Ergebnisse speichern**: Speichern der Metadaten als Frames.
- **/F403/ Simulationsparameter speichern**: Speichern sämtlicher Benutzereingaben wie z.B. Simulationsparameter.

## A.5. Produktdaten

Zu den aus Benutzersicht langfristig zu speichernden Daten gehören die Simulationsergebnisse (als Frames) mit den zugehörigen Simulationsparametern.

## A.6. Produktleistungen

Je nach Auflösungsstufe (Partikelanzahl) werden genauere Ergebnisse erzielt. Dies erfordert einen Mehraufwand an Rechenkapazitäten, wobei die Berechnung zeitnah erfolgen soll. Die Simulation soll physikalisch möglichst genaue Ergebnisse liefern.

## A.7. Benutzeroberfläche

Die GUI wird mittels Qt erzeugt, soll menüorientiert und für den Anwender möglichst intuitiv aufgebaut sein. In der GUI soll eine einfache Visualisierung mittels OpenGL-View möglich sein. Metavisualisierung erfolgt mit VTK.

## A.8. Nichtfunktionale Anforderungen

**Zuverlässigkeit:** YASPHS muss 24h in voller Funktionalität verfügbar sein. Fehlende bzw. falsche Eingaben werden automatisch korrigiert.

**Fehlertoleranz:** Die Robustheit aufgrund falscher Eingaben wird mit Exceptions behandelt. Die Einstellungen werden temporär gespeichert, um bei einem Systemabsturz wiederhergestellt zu werden.

**Look and Feel:** YASPHS verwendet zur Steuerung der Parameter die Graphikbibliothek Qt, so dass eine GUI zur Simulationsvorbereitung zur Verfügung steht. YASPHS beinhaltet ein Standardmenü, ein OpenGL-Fenster und mehrere Buttons zum bedienen der Simulation. Werte können über die Tastatur oder einen Schieberegler eingestellt werden.

**Benutzbarkeit:** Die Benutzeroberfläche ist verständlich aufgebaut und durch die intuitive Anordnung der Elemente leicht zu erlernen. Alle zur Simulation erforderlichen Einstellungen können über die GUI gemacht werden.

**Leistung und Effizienz:** Die Simulation erfolgt nicht in Echtzeit. Je nach Auflösungsstufe benötigt ein Simulationsdurchlauf unterschiedliche Zeitspannen. Der Ressourcenbedarf ist in den Minimalanforderungen in Abschnitt A.2.3 aufgeführt.

**Portierbarkeit:** Das Programm benötigt zur Ausführung die in Abschnitt A.10 aufgeführte Produktumgebung. Da es modular aufgebaut ist, kann es den jeweiligen Anforderungen angepasst werden. YASPHS kann mit den benötigten Daten auf jeden Rechner übertragen werden, der die entsprechenden Anforderungen erfüllt.

**Sicherheitsanforderungen:** Die in der Simulation berechneten Daten sind nicht geschützt. Eine individuelle Bedienung des Programms in Form von einer Benutzeranmeldung ist nicht vorgesehen.

**Korrektheit:** Die SPH-Methode ist eine Approximationsmethode. Dies hat zur Folge, dass reale, physikalische Phänomene nicht exakt abgebildet werden können. Zur genaueren Berechnung stehen Verfahren wie die FEM oder Eulermethoden zur Verfügung. Desweiteren treten Rundungsfehler auf, die ebenfalls negativ in die Berechnungsergebnisse einfließen. YASPHS ist jedoch so implementiert, dass das Fehlverhalten des Programms minimiert wird. Ebenfalls kann die Genauigkeit durch entsprechende Anpassbarkeit gesteigert werden. Die technischen Berechnungen sind mindestens auf 4 Nachkommastellen genau. Nach dem Stand der Technik ist es nicht möglich, Computer-Software so zu erstellen, dass sie in allen Anwendungen und Hardware- Kombinationen fehlerfrei arbeitet.

**Flexibilität:** YASPHS basiert auf OpenSource-Technologien wie dem Datenaustauschformat Collada oder OpenGL und VTK.

**Skalierbarkeit:** k.A.

**Partikelanzahl:** k.A.

**Interoperabilität:** Es wird ein 3D-Austauschformat bereitgestellt.

**Ordnungsmäßigkeit:** k.A.

**Analysierbarkeit:** Bei Fehlern wird ein Hinweis mit Fehlernummer eingeblendet.

**Modifizierbarkeit:** OpenSource

**Konformität:** YASPHS richtet sich nicht nach einer spezifischen Norm

## A.9. Technische Produktumgebung

### A.9.1. Software

- Betriebssystem: Windows (ab XP 2002 SP3), Linux v2.6.31.6
- Treiber für OpenGL 3.2
- Qt 4.5.3
- Maya 2008
- VTK 5.4.2
- OpenGL Library 3.2

### A.9.2. Hardware

- siehe Spezifikitaion in Abschnitt A.2.3

### A.9.3. Schnittstellen

- Schnittstelle zu OpenGL
- IO über Standard 3D-Format (z.B. Collada)

## A.10. Spezielle Anforderungen an die Entwicklungsumgebung

### A.10.1. Software

- VisualStudio 2008
- Qt 4.5.3
- Eclipse 3.0
- C++-Compiler
- OpenGL 3.2
- UML-Tool
- Maya 2008

### A.10.2. Hardware

- siehe Spezifikitaion in Abschnitt A.2.3

## A.11. Testszenarien

Die Funktionsfähigkeit von YASPHS wird an diversen Anwendungsfällen getestet. Weitere Details sind in Abschnitt A.3 aufgelistet.

## A.12. Schlussbemerkung

Im Laufe der Programmentwicklung ist es möglich, die in diesem Pflichtenheft aufgeführten Anforderungen abzuändern oder zu erweitern. Gründe können neue wissenschaftliche Erkenntnisse sein oder Programmteile, deren Implementierung sich als nicht umsetzbar herausgestellt haben.

# B. Handbuch

## B.1. Vorwort

Die Naturphänome der realen Welt zu simulieren ist eine Errungenschaft des technologischen Fortschritts. Ein Teilbereich der Naturphänomene sind die inkompressiblen Fluidbewegungen. Viele Bereiche der technischen Wissenschaften, wie z.B. die Medizin, Metereologie und Astrophysik, sind oft daran interessiert, bestimmte inkompressible Fluidbewegungen virtuell zu simulieren, bevor reale Experimente unternommen werden. Neben der Nachfrage aus der Industrie zeigt die Unterhaltungsbranche Interesse an visuell, möglichst real wirkenden Effekten, die in Filmen und Spielen zur Anwendung kommen. Die bisherigen FEM-Verfahren sind sehr rechenaufwendig und stoßen schon bei der Festkörpersimulation oft an ihre Grenzen. Mit der SPH-Methode werden dynamische Fluidsimulationen in der Praxis möglich.

## B.2. Was ist YASPHS

YASPHS ist ein Softwaretool, welches eine Simulation von isothermalen Fluidbewegungen mittels SPH erlaubt. Dem Benutzer stehen Werkzeuge zur Verfügung, um eine Szene zu erstellen sowie die physikalischen Rahmenbedingungen festzulegen und geeignete Simulationsparameter zu setzen. Die Simualtion kann in Echtzeit betrachtet und in Form von Snapshots gespeichert werden, um zum späteren Zeitpunkt Analysen mit vorhandenen Analyse-Werkzeugen durchführen zu können. Desweiteren können die Snapshots als Datenexport dienen, um z.B. mit externen Renderern eine realitätsnahe Visualisierung des simulierten Szenarios zu liefern.

## B.3. Installation

Dieses Kapitel beschreibt alles Notwendige, um das Softwaretool YASPHS auf einem Computer mit Microsoft Windows zu betreiben. Für das Softwaretool YASPHS werden Programmbibliotheken von Drittherstellern benötigt, deren korrekte Konfiguration in den folgenden Kapiteln beschrieben wird.

### B.3.1. Qt

Qt ist eine C++ Klassenbibliothek für die Programmierung von grafischen Benutzeroberflächen. Qt wurde von dem norwegischen Unternehmen Trolltech entwickelt. Anfang 2008 wurde Trolltech von Nokia aufgekauft. Qt steht in allen Versionen und für verschiedene Betriebssysteme auf dem Internetauftritt von Nokia zur Verfügung. Für den Betrieb von YASPHS ist die Installtion von Qt in der Version 4.5.3 notwendig. Es ist ausreichend die unter „LPGL“ lizensierte Open Source Variante für die Installation zu wählen. Nach der Installation muss die Systemvariable „QTDIR“ vorhanden sein und auf das Wurzelverzeichnis der „QT“ Installation zeigen.

### B.3.2. VTK

Das Visualization Toolkit (VTK) ist eine Open-Source-C++-Klassenbibliothek, entwickelt von der Forschungsabteilung von General Electric. Die Firma Kitware leitet momentan die weitere Entwicklung der unter „BSD“ Lizenz stehenden Software. Die Software ist frei verfügbar auf dem Internetauftrit der Firma „Kitware“ für die Betriebssysteme Linux, Windows, Mac und Unix. Für den Betrieb von YASPHS ist die Installation von VTK in der Version 5.4.2 mit Qt Einbindung notwendig. Nur für das Betriebssystem Windows existiert eine fertig compilierte Version. Allerdings hat diese Installationsversion keine Qt Einbindung weshalb das Übersetzen des VTK Quellcodes zwingend notwendig ist. Beim Übersetezen des VTK Quellcodes sind folgende, von den Standardeinstellungen abweichende, Einstellungen zu beachten:

- Die Option „VTK_USE_GUISUPPORT“ verwenden.
- Die Option „VTK_USE_QVTK“ verwenden.
- Die Option „VTK_USE_QVTK_QTOPENGL“ verwenden.
- Die Option „DESIRED_QT_VERSION = 4“ verwenden.

Zusätzlich muss das Paket „VTK Data“ geladen werden, welches auch auf dem Internetauftrit der Firma „Kitware“ zu finden ist. Nach der Installation müssen die Systemvariablen „PG_VTK_ROOT“, „QT_PLUGIN_PATH“ und „PG_VTK_DATA“ angelegt werden. Die Systemvariable „PG_VTK_ROOT“ muss auf den „VTK“ Wurzelordner , die Systemvariable „QT_PLUGIN_PATH“ auf den im Programmpfad von „VTK“ vorhandenen Unterordner „plugins“ und die Systemvariable „PG_VTK_DATA“ auf den Ordner in dem sich der Inhalt des Pakets „VTK Data“ befindet, zeigen.

### B.3.3. YASPHS

Das Softwaretool YASPHS wird als „ZIP“ komprimiertes Paket zur Verfügung gestellt. Nach dem Dekomprimieren findet der Benutzer das zur Ausführung erforderliche Programm in Form der startbaren Datei „YASPHS.exe“ im Wurzelordner vor. Im Wurzelordner sind weitere Dateien und Ordner vorhanden die für den Betrieb von YASPHS erforderlich sind. Das Programm YASPHS kann deshalb nicht ohne diese weiteren Dateien und Ordner gestartet werden.

## B.4. Programmbedienung

In dem Kapitel Programmbedienung wird eine Übersicht über die wichtigsten Funktionen des Programms geboten. Nach einer Einführung in das Erstellen, Laden und Speichern einer Szene, die das zu simulierende, reale Phänomen repräsentiert, folgt eine Einweisung zur Festlegung der wichtigsten physikalischen Parameter sowie der globalen Simulationsparameter. Nach diesen Schritten kann die Simulation beginnen. Der Benutzer hat die Möglichkeit zwischen drei Modi zu wechseln. Der Simulationsmodus ist standardmäßig eingestellt und bietet eine Vorabansicht des Simulationszustands. Jeder Simulationsschritt kann in Form eines Snapshots gespeichert werden. Die Modi, VTK-Metavisualisierung und Hüllflächenberechnung erlauben eine genauere Analyse dieser Snapshots.

### B.4.1. Die Benutzeroberfläche

Die Benutzeroberfläche ist in vier Bereiche gegliedert. Im oberen Bereich ist das „Hauptmenü“ mit den Icons zu finden, darunter das „Ansichtsfenster“, unten das „Meldungs-Fenster“ und rechts der „Info-Bereich“. (siehe Abbildung B.1)

Das Hauptmenü besteht hauptsächlich aus Dialogen und ist in vier Bereiche gegliedert. Der Bereich „Simulation“ stellt dem Benutzer drei Modi zur Auswahl und bietet eine Möglichkeit zur Navigation in den einzelnen Modi. Abhängig von der Wahl des Modus sind nicht alle Dialoge verfügbar. Der Bereich „View“ ermöglicht dem Benutzer eine detaillierte analytische Betrachtung des aktuellen Snapshots bzw. Simulationszustands. Im Menüpunkt „Settings“ kann der Benutzer alle zur Simualtion nötigen Einstellungen vornehmen. Der letzte Menüeintrag „Scene“ bietet dem Benutzer die Möglichkeit, eine Szene zu erstellen, zu bearbeiten, zu laden und zu speichern.
Das „Ansichtsfenster“ erzeugt im Simulationsmodus eine graphische dreidimensionale Ausgabe des aktuellen Simualtionszustands. In den Modi „VTK-Metavisualisierung“ und „Hüllflächenberechnung“ werden hier Snapshots eingeblendet.
Das „Meldungsfenster“ protokolliert wichtige Ereignisse und Meldungen.

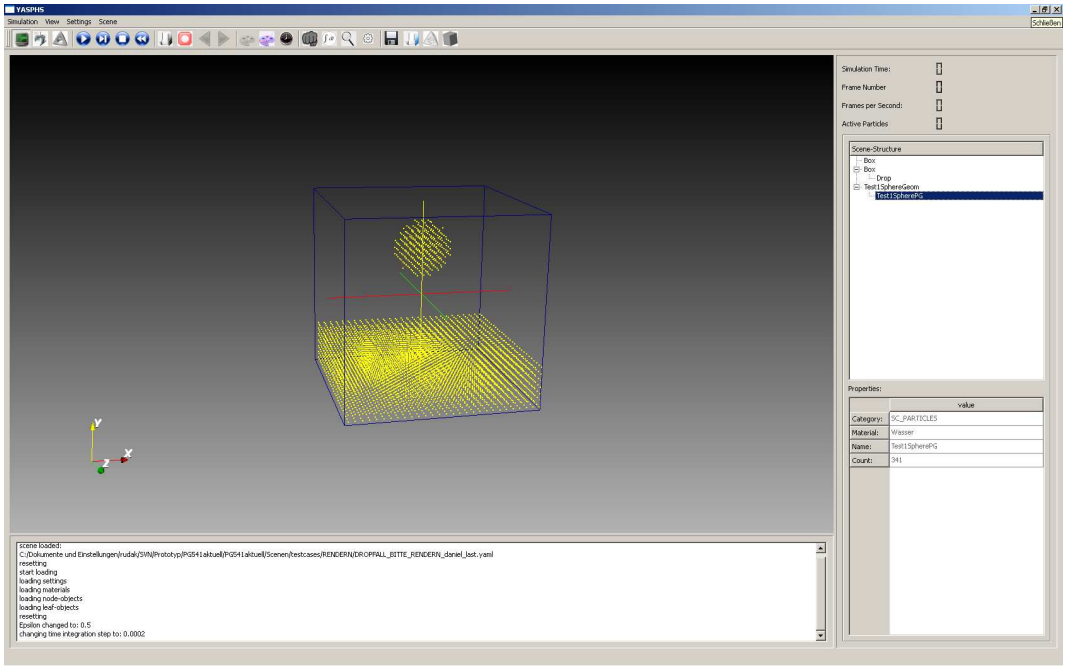



Abbildung B.1.: Die Benutzeroberfläche

Der „Info-Bereich“ gibt Auskunft über die aktuelle Simulationszeit, die aktuelle Frame-Anzahl, die durchschnittliche Frame-Rate sowie die Anzahl der aktuell aktiven Partikel. Desweiteren wird hier die Szenenstruktur visualisiert. Bei Auswahl eines Szenenelements durch Mausklick werden bei „Properties“ Details zu diesem Szenenobjekt eingeblendet.

### B.4.2. Erstellen/Laden/Speichern einer Szene

Der Benutzer hat die Möglichkeit über die Dialoge „Add Primitive“, „Geometry“ sowie „Load Mesh“ eine komplette Szene zu erstellen, die das zu simulierende physikalische Phänomen repräsentieren soll. Über den Dialog „Add Primitive“ können die Primitiven, Würfel und Kugel, an beliebigen Stellen in beliebiger Skalierung in die Szene eingefügt werden. Der Benutzer legt fest, ob die Primitiven als reine Kollisionsobjekte, als Quellen, Senken oder Partikelgruppen fungieren. Jeder Primitiven wird genau ein Material zugeordnet, welches die Dämpfungseigenschaft für die Kollision festlegt. Über den Dialog „Geometry“ können bestehende Geometrien gelöscht, verschoben oder im Falle einer Kugel bzw. eines Würfels skaliert werden. Komplexe Netze können über den Dialog „load Mesh“ der Szene hinzugefügt werden. Die erstellte Szene kann über den Dialog „Save Scene“ gespeichert werden, dabei werden auch die festgelegten physikalischen Parameter sowie Simualtionsparamter mitgespeichert. Der Benutzer hat auch die Möglichkeit bereits existierende Szenen zu laden.

### B.4.3. Festlegen der physikalischen Parameter

Über den Dialog „Forces“ im Menü-Eintrag „Settings“ kann der Benutzer die in der Simulation wirkenden Kräfte aktivieren bzw. deaktivieren sowie justieren. Zusätzlich können im Dialog „Global Parameters“ im Menü-Eintrag „Settings“ die Werte für die Parameter: Restdichte, Gaskonstante sowie Smoothing-Length eingestellt werden. Durch Manipulation dieser Parameter wird das Verhalten des zu simulierenden Fluids verändert.

### B.4.4. Festlegen der Simulationsparameter

Der Benutzer hat die Möglichkeit die Methode zur Nachbarschaftssuche über den Dialog „Neighbour Search“ und die zu verwendende Zeitintegrationsmethode sowie den Zeitschritt über den Dialog „Time Integration“ auswählen. Beide Entscheidungen haben Einfluss auf die Effizienz und Genauigkeit der Simulation.

### B.4.5. Die drei Modi

Standardmäßig ist der Modus „Simulation“ eingestellt. Nach Absolvieren der zuvor genannten Schritte kann die Simulation gestartet werden. Der Benutzer kann zudem zu einem beliebigen Zeitpunkt der Simulation den Record-Button betätigen, um jeden folgenden Simulatinsschritt als Snapshot zu speichern. Zudem stehen dem Benutzer im Simulationsmodus Analyse-Werzeuge unter dem Menüeintrag „View“ zur Verfügung.
Im Modus „VTK-Metavisualization“ bietet sich dem Nutzer die Möglichkeit die zuvor gespeicherten Snapshots bezüglich der Metadaten zu analysieren sowie Isoflächen darzustellen.
Der Modus „Hüllflächenberechnung“ dient zur Berechnung einer Hüllfläche über den Partikeln eines Snapshots zur realitätsnahen Visualisierung.

## B.5. Programmbedienung im Detail

Dieser Abschnitt stellt die Funktionen des Programms detalliert dar.

### B.5.1. Das Menü und die Icons

Das Hauptmenü gliedert sich in vier Bereiche, „Simulation“, „View“, „Settings“ und „Scene“. Der Bereich „Simulation“ bietet die Auswahl zwischen den drei Modi, die Navigation in den einzelnen Modi sowie das Speichern bzw. Laden der Snapshots. Aus dem Menüeintrag „View“ sind Dialoge zur Metavisualisierung in den einzelnen Modi zu finden sowie der Dialog „Timer“

zur Performanceanalyse. Der Bereich „Settings“ bietet dem Benutzer die Möglichkeit alle zur Simualtion nötigen Einstellungen vorzunehmen. Der Bereich „Scene“ bietet Dialoge zum Erstellen, Bearbeiten, Laden und Speichern einer Szene. Jeder Eintrag eines Bereichs löst einen Dialog aus. Jedem Menüeintrag ist ein eindeutiges Tastaturkürzel zugewiesen. Zusätzlich sind die meisten Menüeinträge über Tastaturkürzel aufrufbar. Tabelle B.1 liefert einen Überblick über die Icons, Tastaturkürzel und zugehörigen Aktionen.

| **Icon** | **Aktion** | Tastaturkürzel |
|---|---|---|
|  | Alt+S | In den Simulationmodus wechseln |
|  | Alt+M | In den VTK-Metavisualisierungs-Modus wechseln |
|  | Alt+C | In den Hüllflächenkonstruktions-Modus wechseln |
|  | S | Simulation starten |
|  | O | Simulation stoppen |
|  | P | Simulation schrittweise ausführen |
|  | R | Simulation in den Startzustand zurücksetzen |
|  | F | Ersten Snapshot laden |
|  | L | Vorherigen Snapshot laden |
|  | N | Nächsten Snapshot laden |
| | Shift+Ctrl+L | Snapshot aus Datei laden |
|  | Alt+R | Snapshots aufzeichnen |
|  | Alt+F | Verzeichnis zum Speichern der Snapshots wählen |
|  | Shift+M | Dialog „Metavisualisierung“ anzeigen |
|  | Shift+V | Dialog „VTK-Metavisualisierung“ anzeigen |
| | Shift+S | Dialog „Show Neighbours of“ anzeigen |
|  | Shift+T | Dialog „Timer“ anzeigen |
|  | Ctrl+F | Dialog „Forces“ anzeigen |
|  | Ctrl+N | Dialog „Neighbour Search“ anzeigen |
|  | Ctrl+I | Dialog „Time integration“ anzeigen |
|  | Ctrl+G | Dialog „Global Parameters“ anzeigen |
|  | Alt+Ctrl+L | Szene aus Datei laden |
|  | Alt+Ctrl+S | Szene in Datei speichern |
|  | Alt+Ctrl+M | Mesh aus Datei laden |
| | Alt+Ctrl+G | Dialog „Geometry“ anzeigen |
|  | Alt+Ctrl+P | Dialog „Add Primitive“ anzeigen |
| | Alt+Ctrl+C | Szene leeren |

Tabelle B.1.: Icons und Tastaturkürzel

### B.5.2. Modi

Dem Benutzer stehen drei Modi zur Auswahl, die im Folgenden vorgestellt werden.

#### B.5.2.1. Modus: Simulation

Im Simualtionsmodus kann der Benutzer die Simualtion starten, schrittweise ausführen und eine laufende Simulation stoppen. Bei Aktivierung des Record-Buttons wird in jedem Simulationsschritt ein Snapshot, in dem vorher festgelegten Ordner gespeichert. Während der laufenden Simulation stehen dem Benutzer die Dialoge „Metavisualization“ und „Show Neighbours of“ zur Verfügung.

#### B.5.2.2. Modus: VTK-Metavisualisierung

In diesem Modus können Snapshots geladen und analysiert werden. Der Benutzer kann zwischen den Buttons vor und zurück navigieren. Der Dialog VTK-Metavisualisierung gibt dem Benutzer die Möglichkeit, ausgewählte Metadaten der Snapshots zu visualisieren.

#### B.5.2.3. Modus: Hüllflächenberechung

Dieser Modus stellt eine Schnittstelle zur Anwendungsvisualisierung dar. So können Snapshots geladen, Hüllfächen berechnet und im Anschluß für externe Programme gespeichert werden. Grundlage für die Berechnung ist der Marching-Cubes-Algorithmus, mit dessen Hilfe ein Polygonnetz zur Nachbildung der Partikelwolkenoberfläche gebildet wird. Die gewählten Parameter zur Partikelabtastung und anschließender Netzgenerierung zielen auf möglichst feinmaschige Hüllflächen ab, um zum Beispiel auch einzelne „Partikeltropfen“ darstellen zu können. Komplette Snapshot-Sequenzen können durchlaufen und im Wavefront-Format gespeichert werden. Aufgrund der hohen Auflösung ist die Berechnung allerdings sehr zeitaufwändig und beansprucht enormen Speicherplatz.

### B.5.3. Snapshots

Während der Simulation kann jeder vollendete Simulationsschritt als Snapshot gespeichert werden. Standardmäßig werden die Snapshots im Home-Verzeichnis abgelegt, der Benutzer hat aber auch die Möglichkeit vorher ein Verzeichnis dafür anzugeben. Die Snapshots enthalten für jedes Partikel dessen Metadaten: Position, Druck, Geschwindigkeit sowie Dichte. Im Modus VTK-Metavisualisierung können die Snapshots geladen und untersucht werden. Desweiteren können die Snapshots als Datei-Export für externe Programme dienen, z.B. Renderer.

### B.5.4. Dialoge

Die Interaktion des Benutzers mit YASPHS erfolgt hauptsächlich über Dialoge, die im Folgenden näher erläutert werden.

#### B.5.4.1. Dialog: Global Parameters

Hier hat der Benutzer die Möglichkeit die globalen Parameter: Gaskonstante, Restdichte und Smoothing-length festzulegen. (Siehe Abbildung B.2)

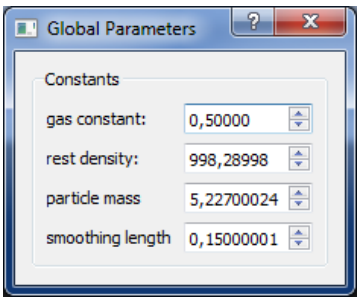



Abbildung B.2.: Dialog: Global Parameters

#### B.5.4.2. Dialog: Time Integration

Über diesen Dialog werden die Methoden für die Zeitintegration und die Kollisionsbehandlung sowie der Zeitschritt bestimmt. Der Parameter $\epsilon$ bietet zusätzliche Kontrolle bei der XSPH-Methode. Bei der Wahl des Zeitschritts ist zu beachten, dass mit kleinerem Zeitschritt eine höhere Genauigkeit der Simulationsergebnisse erzielt wird, der Berechnungsaufwand jedoch linear ansteigt. Die beiden Kollisionsmethoden unterscheiden sich hinsichtlich der Effizienz. (Siehe Abbildung B.3)

#### B.5.4.3. Dialog: Neighbour Search

Dieser Dialog bietet dem Benutzer die Möglichkeit eine geeignete Methode zur Nachbarschaftssuche für die Partikel auszuwählen. Die einzelnen Methoden unterscheiden sich hinsichtlich der Effizienz und sind auf bestimmte Problemstellungen zugeschnitten. Als Standardeinstellung empfielt sich „spatial Hashing“. Der globale Wert für die Hashtable-Scale sollte standardmäßig bei 0.7 liegen. (Siehe Abbildung B.4)

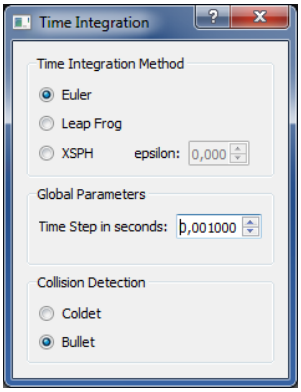



Abbildung B.3.: Dialog: Time Integration

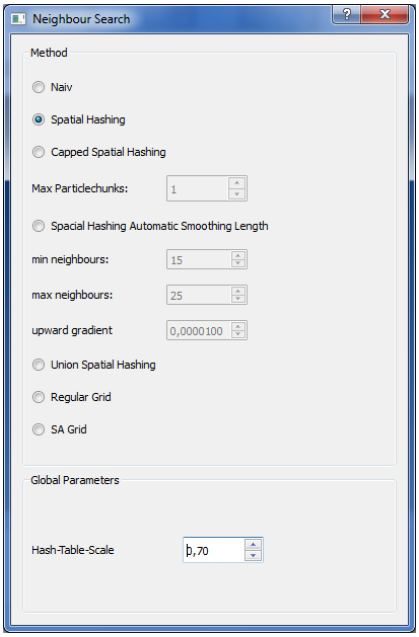



Abbildung B.4.: Dialog: Neighbour Search

#### B.5.4.4. Dialog: Metavisualization

Dieser Dialog ist im Simualtionsmodus verfügbar und wird benutzt zur Visualisierung bestimmter Metadaten, wie Druck, Geschwindigkeit sowie Dichte, für die einzelnen Partikel. Der Benutzer kann die Anzahl der anzuzeigenden Partikel reduzieren, indem er auswählt, dass nur jedes n-te Partikel dargestellt wird. Bei Aktivierung der Checkbox „Velocity Vector“ wird an jedes Partikel der zugehörige Geschwindigkeitsvektor gezeichnet. Die Länge dieser Vektoren ist skalierbar. Der Benutzer hat, bei Aktivierung der Checkbox „Color Gradient“, die Auswahl zwischen der farbkodierten Visualisierung von Druck, Geschwindigkeit oder Dichte. Zusätzlich werden in jedem Simulationsschritt für Druck, Dichte und Geschwindigkeit, der minimale, maximale sowie durchschnittliche Wert angezeigt. (Siehe Abbildung B.5)

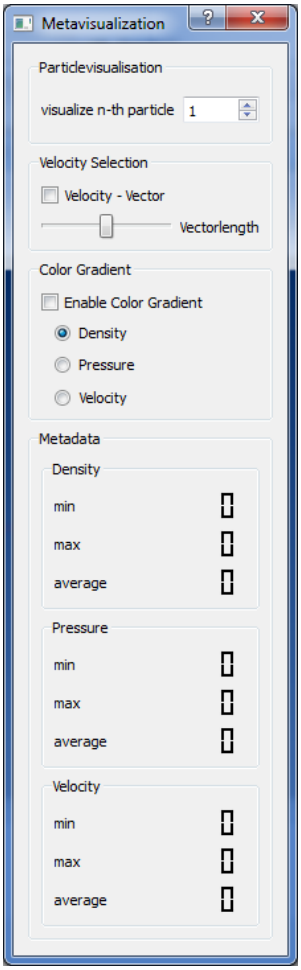



Abbildung B.5.: Dialog: Metavisualization

### B.5.4.5. Dialog: VTK Metavisualization

Dieser Dialog ist nur im Modus VTK-Metavisualisierung aufrufbar und bietet ausgehend von gespeicherten Snapshot-Dateien eine umfassende Möglichkeit zur Nachanalyse vergangener Simulationsberechungen. So kann neben der auch im Simulationsmodus verfügbaren Partikel- und Vektorvisualisierung auf weitere, optisch aufwändigere Techniken zurückgegriffen werden. Neben verschiedener Glyphentypen (Kugeln, Zylinder, Pfeile) und zugehöriger Skalierungsmethoden stehen hier auch Möglichkeiten zur Erzeugung von Isolinien und Isoflächen zur Verfügung. Die Anzahl und der Typ der Generierung (automatisch oder manuell) ist für diese Zwecke jederzeit neu einstellbar. Außerdem ist in diesem Modus der Benutzer in der Lage, durch den Einsatz sogenannter Widgets bestimmte Teilabschnitte des Renderbereichs auszuwählen und gesondert zu betrachten. Es gibt hierbei vier unterschiedliche Typen, die genutzt werden können, um dem Anwender eine intuitive Interaktion zu ermöglichen. So kann beispielsweise mit Hilfe des ContourPlane-Widgets mittels Mausbewegungen eine Schnittebene zur Isoliniendarstellung gezogen oder per GlyphBox-Widget ein Teilvolumen zur Vektorvisualisierung markiert werden. Allerdings ist immer nur ein Widget-Typ zur gleichen Zeit zugelassen. Eine Visualisierung der Partikelnachbarschaften ist in diesem Modus hingegen nicht vorhanden, da diese Informationen in den Snapshot-Daten nicht gespeichert werden.

(Siehe Abbildung B.6)

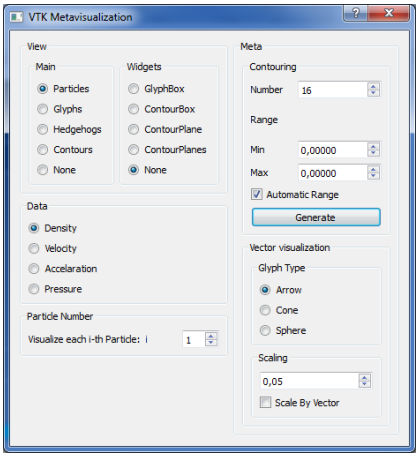



Abbildung B.6.: Dialog: VTK-Metavisualization

### B.5.4.6. Dialog: Show Neighbours Of

Durch Aktivieren der Checkbox „Visualization active“ werden die Nachbarn des ausgewählten Partikels innerhalb des ausgewählten Radius angezeigt. Zusätzlich wird die minimale, maximale sowie durchschnittliche Anzahl der Nachbarn der bisher betrachteten Partikel angezeigt. (Siehe Abbildung B.7)

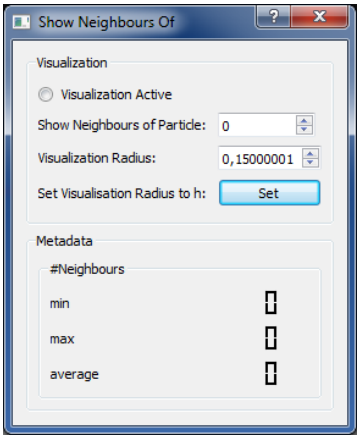



Abbildung B.7.: Dialog: Show Neighbours Of

### B.5.4.7. Dialog: Timer

Dieser Dialog visualisiert prozentual den Berechnugsaufwand der einzelnen Phasen der Simulationspipeline. (Siehe Abbildung B.8)

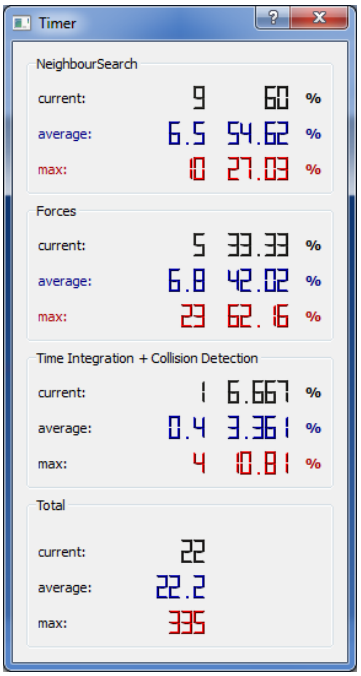



Abbildung B.8.: Dialog: Timer

### B.5.4.8. Dialog: Add Primitive

Über diesen Dialog kann der Benutzer eine neue Primitive, Würfel oder Kugel, in die bestehende Szene einfügen. Die Größe und Position des Würfels wird über die Angabe der linken unteren sowie rechten oberen Ecke definiert. Bei der Kugel wird der Radius sowie die Lage des Mittelpunkts festgelegt. Die neue Primitive kann ein reiner Kollisionskörper, eine Senke, eine Quelle oder eine Partikelgruppe sein. Bei Zuweisung der Partikelgruppe zu der neuen Primitiven wird diese mit Partikel, in äquidistantem Abstand dx, ausgefüllt. Bei einer Quelle kann der Benutzer die Generierungsraterate, Anzahl der Partikel pro Minute, festlegen. (Siehe Abbildung B.9)

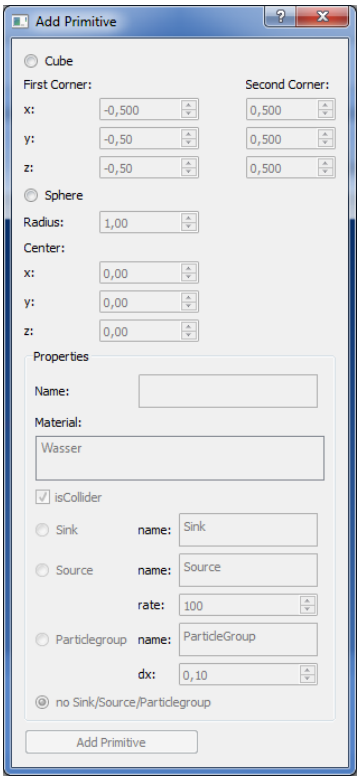



Abbildung B.9.: Dialog: Add Primitive

### B.5.4.9. Dialog: Geometry

Dieser Dialog ermöglicht dem Benutzer die Geometrien aus der Szene zu löschen, zu verschieben bzw. zu skalieren. (Siehe Abbildung B.10)

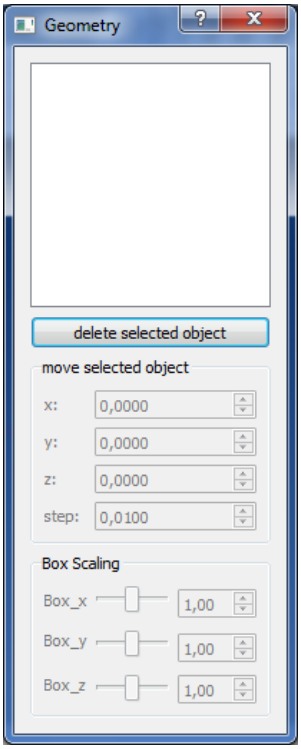



Abbildung B.10.: Dialog: Geometry

#### B.5.4.10. Dialog: Forces

Hier kann der Benutzer die Kräfte in der zu simulierenden Szene festlegen. (Siehe Abbildung B.11)

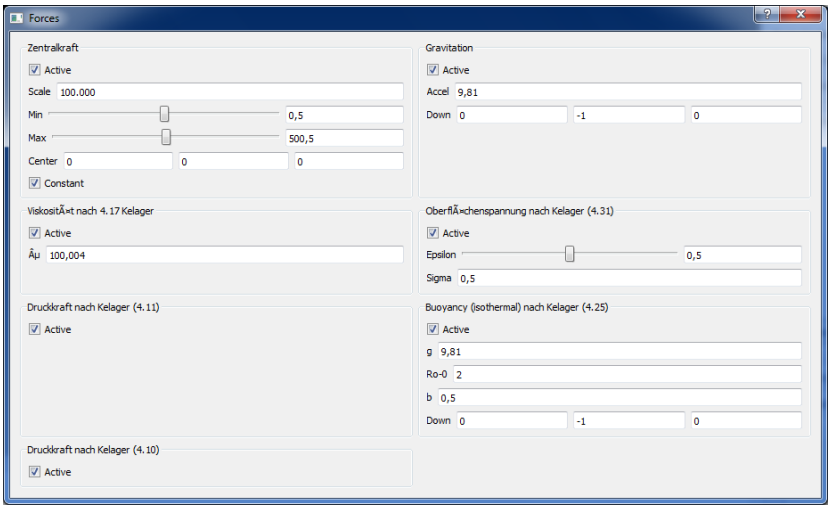



Abbildung B.11.: Dialog: Forces