Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Maximilian Reinhardt

Partikel-basierende Flüßigkeitensimulation mit Smoothed Particle Hydrodynamics

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Maximilian Reinhardt

Partikel-basierende Flüßigkeitensimulation mit Smoothed Particle Hydrodynamics

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Andreas Meisel

Eingereicht am: 14. September 2017

Maximilian Reinhardt

Thema der Arbeit
Partikel-basierende Flüßigkeitensimulation mit Smoothed Particle Hydrodynamics

Stichworte
Simulation, Flüssigkeiten, Smoothed Particle Hydrodynamics, Bachelorarbeit, Behälter Kollision

Kurzzusammenfassung
Dieses Dokument zeigt das Verhalten von Flüssigkeiten in einer Simulation mit Smoothed Particle Hydrodynamics. Die Simulation findet in einem Behälter statt, den die Flüssigkeit nicht verlassen kann. Um dies zu realisieren haben Behälterwände einen Reflektierenden Einfluss auf die Flüssigkeit. Die Kollision mit dem Behälter wird durch Raytracing ermittelt. Smoothed Particle Hydrodynamics wurde erfunden von L.B. Lucy und weiter entwickelt von R.A. Gingold und J.J. Monaghan im Jahre 1977. Es basiert auf den Navier-Stokes Gleichungen, die in der ersten Hälfte des 19. Jahrhunderts von C.L.M.H. Navier, S.D.P.B. de Saint-Venant und G.G. Stokes unabhängig voneinander erfunden wurde.

Maximilian Reinhardt

Title of the paper
Particle-Based Fluid Simulation with Smoothed Particle Hydrodynamics

Keywords
Simulation, Fluids, Smoothed Particle Hydrodynamics, Bachelorthesis, Container collision

Abstract
This document shows the behavior of simulated fluids with Smoothed Particle Hydrodynamics. The fluid ist placed in a container that is complitly closed. It will be reflected when colliding with one of the container walls. The kollision will be detected through ray tracing. Smoothed Particle Hydrodynamics was invented by L.B. Lucy and further developed from R.A. Gingold and J.J. Monaghan in 1977. It is based on the work of C.L.M.H. Navier, S.D.P.B. de Saint-Venant and G.G. Stokes called the Navier-Stokes equasions invented in the first half of the 19. century.

Inhaltsverzeichnis

1 Einleitung

Die Physik von Flüssigkeiten aus der realen Welt nachzubilden ist zur heutigen Zeit schwierig. In der Theorie ist es kompliziert und in der Praxis verbrauchen genaue Simulationen und hohe Auflösungen, in unserem Fall kleinere Kugeln und mehr Dreiecke pro Kugel, zu viel Rechenleistung, so dass es wenig Frames gibt oder die Berechnungen vorher statt finden müssen. Um Fluid-Simulationen im Computer durchzufüren gibt es verschiedene Methoden. Alle diese Methoden basieren auf einer vereinfachten From der Navier-Stokes Gleichung. Eine der Methoden ist zum Beispiel die Eulersche Gitter-basierte Methode [4], wie der Name schon sagt, teilt diese den Raum in ein Gitter auf. Jedes Feld des Gitters hat bestimmte Werte wie Druck, Temperatur usw., je nachdem was Einfluss auf den Fluid nehmen soll. Die Berechnungen finden dann von einem Quellfeld zu einem Zielfeld statt, die vertauscht werden, nach dem das Zielfeld erreicht wurde. Die Elemente in den Feldern des Gitters werden durch eine Differenzialgleichung mit Hilfe der Euler-Methode gelöst, um das nächste Feld zu finden. Noch eine Methode ist die Lattice-Boltzmann-Methode, diese teilt den Raum ebenfalls in ein Gitter, hat aber in den Feldern nur Geschwindigkeitsvektoren. Es gibt wahrscheinliche Geschwindigkeiten zu den anliegenden Feldern und zu dem eigenen Feld. Die Geschwindigkeiten werden durch Kollisions und Strömungsschritte berechnet. Der Kollisionsschritt bestimmt die Größe der Geschwindigkeiten zu den Nachbarfelder und der Strömungsschritt überträgt diese Geschwindigkeiten in die Nachbarfelder. So wird die Energie, von ihrem Ursprung, an die Nachbarfelder weiter geleitet. In diesem Dokument stelle ich die Smoothed Particle Hydrodynamics Methode vor. Die SPH Methode baut, wie der Name schon sagt, auf sogenannten Partikeln und der Arbeit von Navier und Stokes auf. Die Partikel beschreiben einen Punkt in der Flüssigkeit und besitzen jeweils eine Masse, eine Position und eine Geschwindigkeit. Die Position wird in jedem Intervall durch die Geschwindigkeit neu berechnet und die Geschwindigkeit durch die Beschleunigung. Dies geschieht mit der Einbeziehung umliegender Partikel in einem vorbestimmten Radius h.

1.1 Motivation

Im Realen Leben sind Flüssigkeiten alltäglich und ein großer Bestandteil des Lebends. Die Simulation der Flüssigkeiten hilft in vielen Lebenslagen: Bei Aerodynamik, um zum Beispiel den Luftwiederstand und die Strömung der Luft bei Fahrzeugen oder Flugzeugen zu Testen. Bei Wetterberechnungen, um das Verhalten von Wolken und Luftströmungen vorherzusagen. Aber auch bei Flut− und Katastrophen Vorhersagen, wie Überschwemmungen, Tsunamis aber auch Verbreitung von Vulkanasche oder Radioaktivität, da diese durch Winde beeinflusst werden. Im Rahmen dieser Arbeit soll eine Szene entstehen, die so genau wie möglich der Realität nahe kommt. Dazu muss die Szene, um anschaulich zu sein, auch flüssig dargestellt werden.

1.2 Zielsetzung

Dieser Bachelor hat sein Ziel erfüllt, wenn im Rahmen des Frameworks von Herrn Professor Dr. Philipp Jenke in C #, eine Szene durch Smoothed Particle Hydrodynamics entsteht, die sich so realistisch wie möglich verhält und ein beliebiger konvexer Körper, der aus Dreiecken besteht und in dem Framework erstellt wurde, als Behälter gewählt werden kann. Als realistisch wird eine Szene angesehen, wenn die Gravitation, sowie die Bewegungseigenschaften eines Fluids eingehalten werden. Die Anwendung sollte mindestens auf 30 frames per second laufen können, da dieses als flüssig angesehen wird.

1.3 Aufbau der Arbeit

Nach dieser Einleitung werden erstmal ein paar grundlegende Dinge geklärt. Wie sind die physikalischen Eigenschaften in der realen Welt und wie sieht die Navier-Stokes Gleichung, so wie Smoothed Particle Hydrodynamics Methode aus. Danach wird geklärt wie die Grundlagen in Code umgesetzt werden kann und danach, wie dieser mit einigen Beispielen umgesetzt aussieht. Zu guter Letzt kommt ein Rückblick, was sollte man beachten, was gab es für Probleme, was hätte man anders oder weiter machen können wenn man mehr Zeit gehabt hätte und was hält die Zukunft für Smoothed Partikel Hydrodynamics bereit.

2 Grundlagen

2.1 Flüssigkeiten

Als Flüssigkeiten werden Liquide und auch Gase gesehen. Flüssigkeiten sind z.B. Wasser, Rauch und auch Plasma. Flüssigkeitenteilchen bewegen sich nichtperiodisch und mit einer Nahordnung, das bedeutet der Bezugsbereich zu umliegenden Teilchen ist klein. Flüssigkeiten sind volumenbeständig aber nicht formbeständig.

2.2 Navier-Stokes

$$\rho(\frac{\partial v}{\partial t} + v \bullet \bigtriangledown v) = - \bigtriangledown p + \rho g + \mu \bigtriangledown^2 v$$

[9] [1]
Das ist die erste Navier-Stokes Gleichung. Aber was genau sagt sie aus? Eine bestimmte Masse, die in jeder Einzelformel Einfluss durch Summierung mal Gewichtung hat, bewegt sich in einer bestimmten Zeit t in einer bestimmten Geschwindigkeit v. Dadurch kann die Strecke berechnet werden, die die Masse zurücklegt. Die Gleichung besteht quasi aus 5 Teilen. Auf der linken Seite der Gleichung ist die Beschleunigung, diese spaltet sich in zwei Teile $\frac{\partial v}{\partial t}$, dies ist die Beschleunigung über eine bestimmte Zeiteinheit (t) und der zweite Teil, die konvektive Beschleunigung (v $\bullet \bigtriangledown$ v), diese ist zuständig für Flüssigkeiten die durch kleinere Öffnungen fließen. Je kleiner die Öffnung, desto schneller muss die Flüssigkeit fließen, um die gleiche Menge an Masse zu bewegen, die bewegt werden würde ohne die Verengung. Auf der Rechten Seite der Gleichung sind die Kräfte, die von außen wirken, wie die Erdanziehung (g), außerdem der Druck (p) und die Zähflüssigkeit (μ).

$$\rho(\bigtriangledown * v) = 0$$

Dies ist die zweite Navier-Stokes Formel, sie steht für den Masseerhaltungssatz. Dieser besagt, dass Masse weder erschaffen noch zerstört werden kann.

2.3 Smoothed Particle Hydrodynamics

Die Smoothed Particle Hydrodynamics Methode wurde erfunden von L.B. Lucy [7] und weiterentwickelt von R.A. Gingold und J.J. Monaghan [3]. SPH ist nicht nur nutzbar für Flüssigkeiten sondern auch für alle anderen deformierbaren Körper. Es war erst für die Astrophysik gedacht um z.B. Meteoriteneinschläge zu simulieren [12], aber dann wurde entdeckt, dass die Methode vielseitiger verwendet werden kann. In SPH können wir die zweite Navier-Stokes Formel

außer acht lassen, da die Masse konstant bleibt ist dies gegeben, solange wir keine Partikel während der Berechnung löschen oder neue erstellen. Auch die konvektive Beschleunigung wird im Weiteren vernachlässigt, da dies nur ein Teil der berechneten Beschleunigung ist und uns interessiert nur die gesamte Beschleunigung. Somit ergibt sich für die Beschleunigung eines Partikels (i) die Formel:

$$\frac{\partial v_i}{\partial t} = g - \frac{1}{\rho_i} \bigtriangledown p + \frac{\mu}{\rho_i} \bigtriangledown^2 v$$

[9] [8] Die SPH Methode selbst besteht nur aus einer Funktion hier F, die für unsere Fälle angepasst werden kann und noch für viele weitere Fälle, wenn gewünscht.

$$F_i(r_i) = \sum_j F(r_j) W(r_i - r_j, h)$$

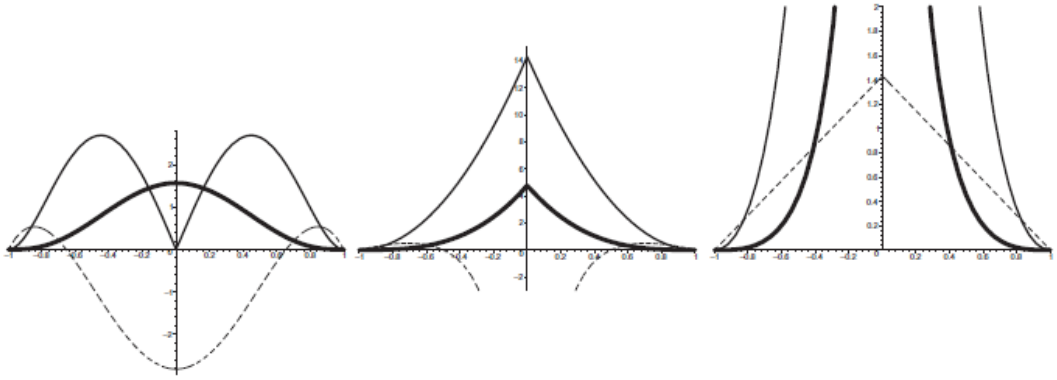


Abbildung 1: Quelle: Particle-Based Fluid Simulation for Interactive Applications von Matthias Müller, David Charypar and Markus Gross. Zeigt verschiedene smoothing kernel und jeweils die ersten zwei Ableitungen.

Das Besondere an der Funktion, ist der smoothing kernel W. Dieser besitzt einen Einflussradius h, alle Partikel die außerhalb des Einflussradius sind, werden in der Summierung ausgeschlossen, da W dann den Wert 0 annimmt und mit W multipliziert wird. In der Abbildung (1) sieht man verschiedene smoothing kernel. Die x-Achse ist der Wert h von -1 bis +1 und die y-Achse der Einflusswert, den der Kernel mit entsprechendem h haben wird. Außerdem sind jeweils 3 Linien zu sehen, die dickste ist die normale Funktion nicht abgeleitet, die etwas dünnere ist der Gradient und die gestrichelte ist der Laplacian. In dem Abbildungsfall ist links der smoothing kernel für die Dichte. Für uns ist die dicke Linie von Interesse, man kann sehr gut sehen das der Einfluss erst langsam abnimmt und dann schnell gegen 0 geht. Die mittlere Abbildung ist der smoothing kernel für den Druck. Hier ist die erste Ableitung wichtig. Je dichter sich die Partikel nähern umso größer der kernel Faktor. Diese Funktion nimmt fast gleichmäßig ab und nähert sich dann langsam 0. Bei der dritten Abbildung ist die zweite Ableitung interessant,

denn diese beeinflusst die Viskosität. Die Ableitung nimmt einfach nur gleichmäßig ab, bis sie 0 ist. Die Kernel kann man sich selbst aussuchen. Die meisten wurden durch Testen oder wie in der Abbildung 1 durch Veranschaulichung, was man sich genau von dem Kernel erhofft, erstellt. Gut veranschaulicht ist dies bei dem Druckabbild. Erst gibt es wenig Druck, ein Körper der unter Druck steht würde sich leicht verformen. Aber je dichter sich Partikel kommen oder je mehr sich ein Körper verformt, um so schneller steigt der Druck. Wie in der Navier-Stokes Gleichung schon zu sehen war, muss unsere Funktion mehrfach abgeleitet werden. Da unsere Funktion sich auf die Position (r) von dem Partikel (i) bezieht und dies nur im smoothing kernel benutzt wird, bezieht sich die Ableitung auch nur auf W den smoothing kernel, da alles andere als Konstante angesehen wird. Die Formel für ρ, die Dichte, sieht demnach wie folgt aus:

$$\rho_i \approx \sum_j m_j W(r_i - r_j, h)$$

Die Formel für p, den Druck Gradienten, wird so beschrieben:

$$\frac{\bigtriangledown p_i}{\rho_i} \approx \sum_j m_j (\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) \bigtriangledown W(r_i - r_j, h)$$

Der Druckgradient hat eine Abhängigkeit von beiden betroffenen Partikeln (i und j) für ein zuverlässigeres Ergebnis. Die letzte Formel für μ, die Viskosität oder Zähflüssigkeit, ist wie folgt:

$$\frac{\mu}{\rho_i} \bigtriangledown^2 v_i \approx \frac{\mu}{\rho_i} \sum_j m_j (\frac{v_j - v_i}{\rho_j}) \bigtriangledown^2 W(r_i - r_j, h)$$

Auch hier ist die Funktion abhängig von der Geschwindigkeit beider Partikel.

2.4 Dichte

Die Dichte beschreibt das Verhältnis von Masse zum Volumen eines Körpers, in unserem Fall die Masse an einer Koordinate. Es wird gemessen in Gramm zu cm^3 also je höher die Dichte, umso schwerer das Element z.B. Wasser hat eine Dichte von 1 und Gold eine Dichte von 19,32.

2.5 Druck

Druck ist der Maßstab für einen Körper, in wie weit er auf eine Komprimierung reagiert. Wenn man einen Reifen aufpumpt, muss sich immer mehr Luft den gleichen Platz teilen. Deswegen wird der Druck größer, bis der Reifen die Kraft die der Luftdruck ausübt nicht mehr halten kann und platzt.

2.6 Viskosität

Viskosität ist für die Zähflüssigkeit von Flüssigkeiten zuständig. Je höher die Viskosität, desto zähflüssiger die Masse. Wasser hat eine niedrige Viskosität und fließt einem direkt von der Hand. Aber eine Flüssigkeit mit höherer Viskosität hat einen stärkeren Zusammenhalt zwischen den Molekülen im realen Leben und zwischn den Partikeln in unserem Fall. Dies bedeutet, dass ein Teil der Flüssigkeit z.B. schon von der Hand hängen könnte, aber die Partikel auf der Hand hindern diese noch am Fallen, durch den Zusammenhalt.

2.7 Beschleunigung

Die Beschleunigung gibt an, wie weit sich die Geschwindigkeit in der nächsten Zeiteinheit vergrößert. Ein Fahrzeug beschleunigt solange sich die Geschwindigkeit erhöht oder hat eine negative Beschleunigung wenn das Fahrzeug gebremst wird.

2.8 Geschwindigkeit

Die Geschwindigkeit ist die Strecke, die eine Masse in einer bestimmten Zeit zurück legen kann. Die gemessene Geschwindigkeit eines Farzeugs ist die, die man zu einer bestimmten Zeit vom Tachometer ablesen kann.

3 Konzept

3.1 Partikel und Partikel Cloud

Der Partikel ist eine einfache Datenstruktur die zur Visualisierung und für die Berechnungen eines Bereiches benutzt wird. Er beinhaltet nur eine Masse, die Geschwindigkeit und eine Position. Zu Hilfszwecken wird auch die root Node aus der Szenenklasse übergeben. Die Nodes einer Szene sind wie ein Baum aufgebaut, deswegen wird nur die root Node benötigt, um eine weitere Node hinzuzufügen, die in der Szene gezeichnet werden soll. Die root Node wird benötigt, weil in dem Partikel die Shpere erstellt wird, die für die Visualisierung eines Berechnungsbereiches da ist. Die Partikel Cloud fast alle Partikel zusammen für spätere Berechnungen. Die Cloud beinhaltet auch die für alle Partikel in der Cloud gültigen Werte wie Gravitation und den Einflussradius h.

3.2 Partikel Nachbar

Es werden nur die Partikel Nachbarn in die Berechnung mit einbezogen, die in der Reichweite h sein können, um Rechenzeit zu sparen. Wie beim smoothing kernel schon bekannt, wird bei einer Partikelentfernung die größer ist als h, 0 als smoothing kernel Gewicht genommen. Das smoothing kernel Gewicht bringt die Summierung auf 0, wir haben also eine Menge Summierungen mit 0 die wir weg optimieren können. Dazu teilen wir die Szene in ein Raster auf, welches eine Feldergröße von 2 mal h hat. Bei allen n Rechenschritten werden die Partikel in das Raster einsortiert, ein Rechenschritt wird gesehen als eine Fluidbewegung, also alle Partikel einmal bewegt. Das Finden des zugehörigen Rasters ist umgesetzt durch eine Interpolation über die Größe der Szene und die Anzahl der Rasterfelder. Bei jedem Berechnungsschritt findet auch eine Interpolation statt, um das Rasterfeld in dem sich der Partikel (i) befindet zu ermitteln. Die Interpolation ist beim Einsortieren des Rasters die selbe, wie die Interpolation um das Raster zu einem Partikel zu finden ($Partikelposition/Scenengre*AnzahlderRasterfelder$ das Ergebnis wird aufgerundet). Es werden außerdem alle umliegenden Rasterfelder in die Berechnung mit einbezogen. Denn sollte der Partikel nicht zentral in der Mitte liegen, wird der Radius h außerhalb des Rasterfeldes enden. Sollte der Radus h außerhalb des Rasters enden, werden nicht alle berechnungsrelevanten Nachbarn in die Rechnung mit einbezogen.

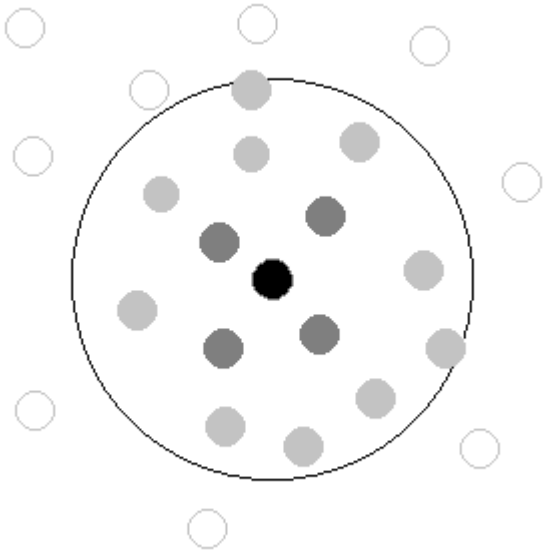


Abbildung 2: Erstellt in Paint. Die Punkte stellen die Partikel dar und der Kreis den Einflussradius h.

In Abbildung 2 ist ein Beispiel für die Interaktion von Partikeln mit dem Radius h zu sehen. Der schwarze Punkt in der Mitte ist der Partikel für den die Berechnung statt findet. Der große Kreis darum ist der Einflussradius h. Je heller ein Partikel wird, umso weniger Einfluss hat er auf die Berechnungen für den schwarzen Partikel, bis hin zu den weißen Partikeln, die gar keinen Einfluss mehr auf die Berechnungen haben.

3.3 Smoothing Kernel

Es gibt 3 verschiedene smoothing kernel jeweils für die Dichte, den Druck und die Viskosität. Die Methoden der 3 smoothing kernel sind alle gleich aufgebaut. Sie bekommen zwei Parameter übergeben: Erstens einen Vektor, der die Distanz zwischen den beiden Partikeln beschreibt (r) und zweitens den Radius h. Der smoothing kernel für die Dichte sieht wie folgt aus:

$$W(r_i - rj) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - |r|^2)^3 & 0 \leq |r| < h \\ 0 & otherwise \end{cases}$$

[13] Bei der Implementierung der smoothing kernel gibt es keine Besonderheiten, sie werden so implementiert, wie es die Formel beschreibt. Der Smoothing Kernel für den Druck sieht wie folgt aus:

$$W(r_i - rj) = \frac{-45}{\pi h^6} \begin{cases} (h - |r|)^2 \frac{r}{|r|} & 0 \leq |r| < h \\ 0 & otherwise \end{cases}$$

Zu guter Letzt der smoothing kernel für die Viskosität:

$$W(r_i - rj) = \frac{45}{\pi h^6} \begin{cases} (h - |r|) & 0 \leq |r| \leq h \\ 0 & otherwise \end{cases}$$

3.4 Dichte

Die Dichte eines Partikels ist die Summe der Masse von allen benachbarten Partikeln mal smoothing kernel.

$$\rho_i = \sum_j m_j * W(r_i - r_j, h)$$

[5]

3.5 Druck

Der Druck wird für ein Partikel wie folgt berechnet: k($\rho - \rho_0$). Hierbei ist die Konstante k nur ein Gewicht. ρ ist wie immer die Dichte und ρ_0 ist die Dichte im ruhenden Zustand, was bedeutet, dass keine anderen Partikel in dem Bereich h sind. Die ruhende Dichte ist also die Masse unseres Partikels und der Druck die Differenz zwischen dem Partikel ohne Nachbarn und mit Nachbarn. Für die Beschleunigungsberechnung wir allerdings der Druckterm $- \bigtriangledown p$ gebraucht, der sich wie folgt berechnet: Die Summe der Masse aller benachbarten Partikel mal $\bigtriangledown$ smoothing kernel mal $\frac{Druck_{Partikel}}{Dichte^2_{Partikel}} + \frac{Druck_{Nachbar}}{Dichte^2_{Nachbar}}$ [5]. Der Druck ist dafür da, dass die Partikel einen gewissen Abstand zueinander halten.

3.6 Viskosität

Die Berechnung der Viskosität sieht wie folgt aus: μ mal der Summe der Masse aller Benachbarten Partikel mal $\frac{Geschwindigkeit_{Nachbar} - Geschwindigkeit_{Partikel}}{Dichte_{Nachbar}}$ mal $\bigtriangledown^2$ mal smothing kernel [5]. Die Berechnung dient dazu, alle Partikel in der Flüßigkeit in der selben Geschwindigkeit zu bewegen. Da dieser Vektor die Geschwindigkeit des Partikels umliegender Partikel annähert.

3.7 Geschwindigkeit

Bei der Berechnung der Geschwindigkeit kommen alle vorherigen Berechnungen zusammen in der Navier-Stokes Gleichung. Diese ergibt die Beschleunigung, mit der dann die Geschwindigkeit aktualisiert wird.

3.8 Behälter Kollision

Auch bei der Behälter Kollision gibt es mehrere Herangehensweisen.[11] Die Herangehensweise in dieser Arbeit ist eine eigene Idee abgeleitet von der Reflextionsmethode. Die Reflextions-

methode funktioniert, wie der Name schon sagt, durch Reflextion der Bewegung, wenn eine Kollision statt finden sollte. Der SPH Berechnungsklasse wird das Behälter Objekt mitgegeben, dieses Objekt ist eine einfache Geometrische konvexe Figur wie z.B ein Würfel oder eine Kugel. Um alle geometrischen Figuren zu unterstützen wird in einer Liste die kleinst mögliche Fläche, der Oberfläche der Figur, gespeichert. Dies ist ein Dreieck das durch drei Positionen beschrieben und gespeichert wird. Diese Dreiecke entstehen in der Node der Figur schon durch die Erstellung und müssen nur in der Liste gespeichert werden. Dann kann durch Raytracing von einem Vektor auf eine Ebene (jeweils eine Behälter Oberfläche nach der anderen) eine Kollision festgestellt werden. Da der Vektor und die Ebene aber ins Unendliche gehen, muss dann noch die Länge des Vektors des nächsten Schrittes im Verhältnis zum Vektor bis zum Schnittpunkt verglichen werden. Es muss außerdem geprüft werden, ob die beiden Vektoren in die selbe Richtung zeigen. Die Gleichung zur Berechnung des Schnittpunktes sieht wie folgt aus:

$$\frac{-(\frac{Ev_1 x Ev_2}{|Ev_1 x Ev_2|} * Scord) + (\frac{Ev_1 x Ev_2}{|Ev_1 x Ev_2|} * Ecord)}{\frac{Ev_1 x Ev_2}{|Ev_1 x Ev_2|} * Sdir}$$

Wobei Ecord der Startpunkt der Oberfläche ist, Ev1 und Ev2 die Vektoren, die die Oberfläche als Ebene beschreiben, Scord ist der Startpunkt für den Vektor, der die Oberfläche schneiden soll und Sdir die Richtung, in die der Strahl zeigt. Die Gleichung ergibt dann Lambda, die Länge bei dem der Strahl die Oberfläche schneidet. Jetzt muss nur noch geprüft werden, ob Lambda zwischen 0 und 1 liegt, um zu sehen ob der Schnitt im nächsten Berechnungsschritt statt findet. Außerdem werden alle Ebenen geschnitten, außer die, die Parallel zum Vektor verlaufen. Deshalb darf nur der Schnitt mit dem kürzesten Lambda genommen werden. Die Berechnungen werden mit einem neuen Strahl wiederholt. Sollte ein Schnitt gefunden werden, wird die Bewegung gespiegelt, bis kein Schnitt mehr gefunden wird. Die Spiegelung des Strahles wird durch folgende Formel berechnet:

$$2 * ((v * -1) * e) * e - (v * -1)$$

v steht für den Vektor der Bewegung des Partikels und e für die Normale der Ebene auf die der Vektor trifft. Diese Berechnung findet nur statt, wenn der ïn bound check"vorher positive ist. Es wird geprüft ob die neu berechnete Position innerhalb des Behälter ist. Für die Berechnung des ïn bound checks"wird die Ebenennormale gebraucht. Die Formel sieht so aus:

$$pos \bullet n - n \bullet epos$$

Hierbei ist pos die Position die geprüft werden soll, n die Normale der Ebene und epos ist ein beliebiger Punkt auf der Ebene. Wir haben die drei Eckpunkte der Ebene. Alle drei Eckpunkte sind potentielle Kandidaten für epos von unserer Formel, da alle Ecken auf der Ebene liegen. Sollte die Position nicht innerhalb des Behälters sein, gibt es noch alternative Szenarien. Bei der Position wird auch beachtet, dass der Vektor nicht in der gleichen Länge gespiegelt wird,

sondern die Strecke die er bis zur Wand zurück gelegt hat, beachtet wird. Die Länge des gespiegelten Vektors wird wie folgt berechnet:

$$ll - (ll/100) * (la * 100)$$

ll ist am Anfang die gesamte Länge des Vektors und nimmt dann pro Spiegelung je nach Länge von Lambda (la), welches die prozentuale Länge bis zum Schnittpunkt an gibt, ab.

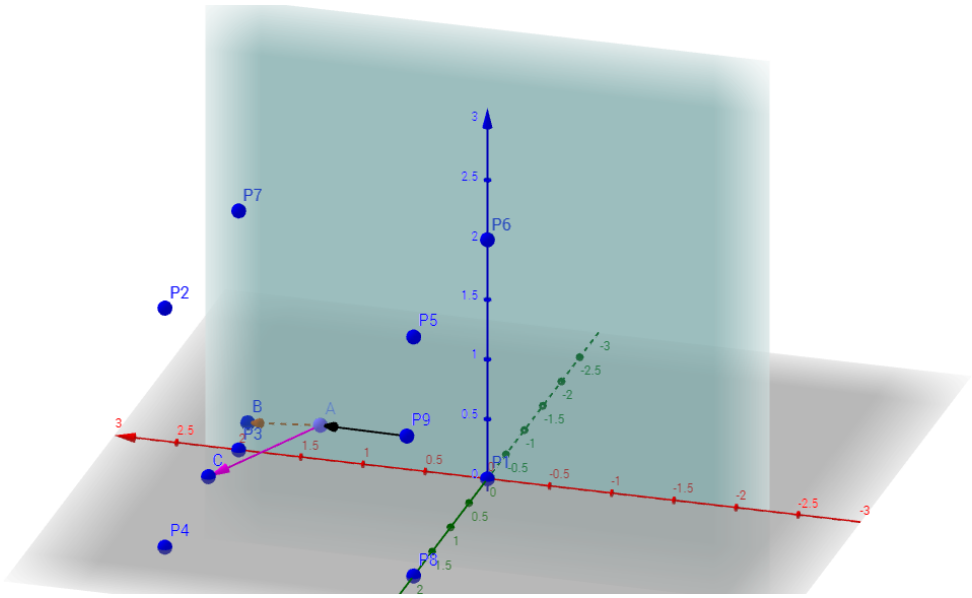


Abbildung 3: Erstellt durch GeoGebra. In grau und in grün sind zwei Ebenen zu sehen. Die blauen Punkte A,B und C sind mögliche Bewegungen des Partikels P9. Alle anderen blauen Punkte P1-8 begrenzen den Behälter. Die Pfeile sind die Bewegungsvektoren der möglichen Schritte.

Die Abbildung 3 dient zur Veranschaulichung des beschriebenen. Es werden 2 Ebenen angezeigt, eine Graue und eine Grüne. Der Schwarze Vektor ist der nächste Berechnungsschritt des Partikels P9. Der Orangene Vektor wäre ein zweiter Schnitt der fehlerhaft genommen werden könnte, da schon eine Ebene davor einen Schnitt hat. Der Pinke Vektor ist die Spiegelung des Schwarzen Vektors. Schneidet aber auch noch die zweite Ebene und muss also selbst auch gespiegelt werden.

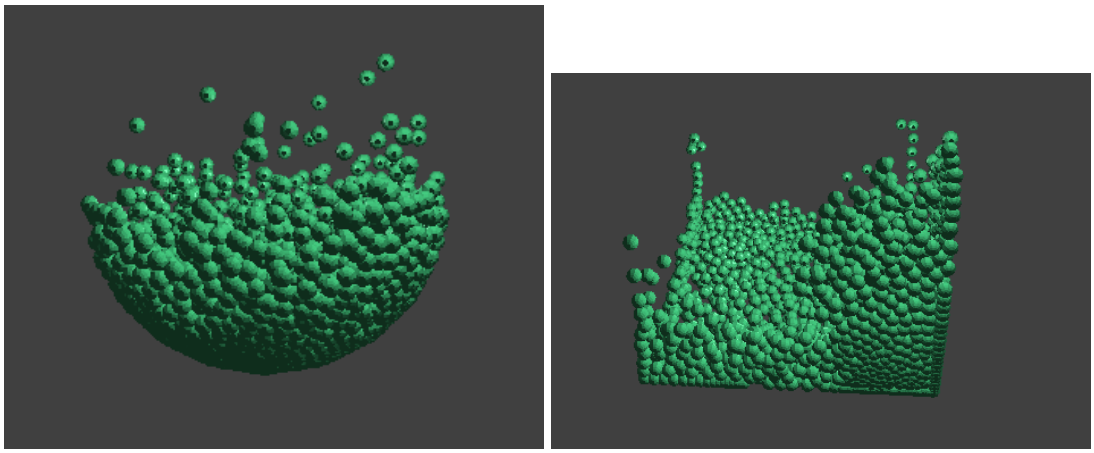


Abbildung 4: Ausschnitt aus dem Programm mit verschiedenen Behältern.

In Abbildung 4 ist gut zu sehen, wie der Behälter die Partikel aufhält, da die Behälter keine Einfärbung haben, sind sie nur durch die Partikel zu sehen.

4 Implementierung

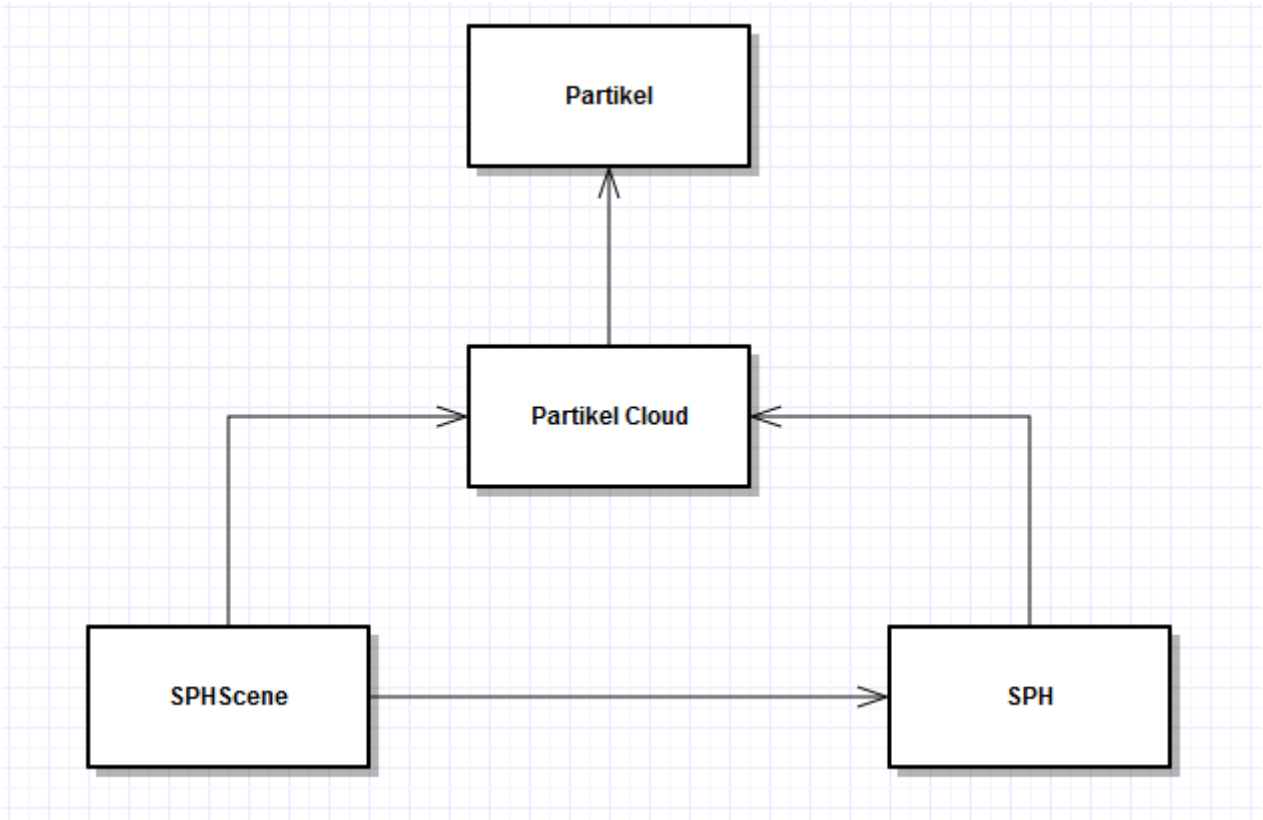


Abbildung 4: Ein UML Diagramm, um die Zusammenarbeit der Klassen darzustellen. Was in den Klassen genau enthalten ist, wird im folgenden Text erklärt.

4.1 Partikel und Partikel Cloud

Die Partikel Klasse bekommt im Konstruktor eine Position als Vector3. Vector3 ist eine Datenstruktur, der drei floats übergeben werden. Diese Datenstruktur besitzt Methoden für Vektorberechnungen. Außerdem bekommt sie eine Masse als float, eine Geschwindigkeit als Vektor3 und die Rootnode übergeben. Die übergebenen Parameter werden gespeichert und eine Shperenode erstellt, die durch eine Translationsnode an die gewünschte Position gebracht

wird. Bei Änderung der Position wird auch die Translation geändert. Die Partikel Cloud bekommt folgendes übergeben: Die Gravitation als Vector3, den Einflussradius h als float, die Zähflüssigkeit auch als float und eine Liste aller Partikel. Die Partikel Cloud besitzt nur geter und seter, da die Klasse nur zur Strukturierung beiträgt, falls es mehr als eine Flüssigkeit in der Szene gibt.

4.2 Partikel Nachbarn

Um das Raster zu realisieren gibt es eine Datenstruktur, RasterUnit, die ein Rasterfeld darstellt. Die RasterUnit bekommt einen x, y und z Wert als int übergeben. Diese geben an, an welcher Position sich die RasterUnit im Raster befindet. Das Raster selber ist eine Liste von RasterUnits, welche als globale Variable in der Berechnungsklasse gespeichert ist. Die Variablen der RasterUnit sind eine Liste von Partikeln, die gefüllt und geleert werden kann. Die Position der RasterUnit und eine Liste von RasterUnits, welche alle Nachbarn beinhaltet, zählt also maximal 26 Stück. Da man sich selber nicht als Nachbar zählt, 3 x 3 x 3 minus 1. Wenn die RasterUnit am Rand oder in einer Ecke ist, hat sie weniger Nachbarn. Die Position wird beim erstellen übergeben als eigene Datenstrucktur. Die Liste der Nachbarn wird bei der Inizialisierung gefüllt. Dafür wird über alle RasterUnits iteriert und die Positionen verglichen (x, y, z +- 1). Jetzt wo das Raster inizialisiert ist, können die Partikel einsortiert werden, dies wird alle x Berechnungsschritte gemacht und ist durch eine Variable in der Berechnungsklasse frei wählbar. Um für einen Partikel die passende RasterUnit zu finden wird eine Position ausgerechnet. Dafür wird x, y und z für die Position einzeln als int berechnet. Es wird der Betrag von der Partikel Position (x,y,z) / die Würfelgröße(x,y,z) (die Größe des gesamten Rasters) - σ, das Ganze $*$ der Anzahl der RasterUnits (x,y,z) (Würfelgröße (x,y,z) / h $*$ 2), das ergibt dann jeweils das x, y und z der Position der RasterUnit im Raster.

4.3 Smoothing Kernel

Bei jedem kernel gibt es einen Teil der Berechnung die unabhängig ist vom Partikel und nur abhängig von h, welches sich in unserem Fall nicht ändert. Diese Berechnungen werden bei der Erstellung der Berechnungsklasse gemacht und gespeichert. Dies sind 3 float Variablen für den Dichte kernel: $315/(64 * \pi * h^9)$, für den Druck kernel: $-45/(\pi * h^6)$ und für den Viskositäts kernel: $45/(\pi * h^6)$. Der Rest des kernels wird in einer eigenen Methode berechnet.

4.4 Dichte

Die Dichte wird zur Berechnung von $\bigtriangledown p$ und $\mu \bigtriangledown^2 v$ aus der Navier-Stokes Gleichung gebraucht. Für die Berechnung der Dichte wird eine Liste von relevanten Partikeln übergeben und der Partikel selbst, für den die Berechnung statt findet. Das Ergebnis das zurück gegeben wird ist ein float und wird mit 0 inizialisiert. Danach wird über alle Partikel der Liste iteriert und die Masse mal dem smoothing kernel summiert. Wenn der Partikel alleine in seiner Umgebung ist,

ist die Dichte gleich der ruhenden Dichte. Da die beiden Dichten für die Druckberechnung subtrahiert werden ergibt dieses 0. Es herrscht also kein Druck.

4.5 Druck

Zur Druckberechnung gehören zwei Teile. Zum einen der Druck selbst, dieser wird nach der Dichte berechnet und ist ein Floatwert. Die Berechnung ist k $*$ (Dichte - RuhendeDichte). Zum anderen der Druckterm, dies ist der $\bigtriangledown p$ Part der Navier-Stokes Gleichung. Das Ergebnis ist ein Vector3 und braucht den Druck Floatwert für die Berechnung. Die Druckterm Methode bekommt, wie die Dichte, eine Liste relevanter Partikel und den betreffenden Partikel übergeben. Es wird ebenfalls, wie bei der Dichte, über alle Partikel der Liste iteriert. Dieses Mal wird der Partikel den es betrifft allerdings außer acht gelassen. Durch eine equals Prüfung wird dies realisiert. Außerdem werden Partikel ignoriert, die die selbe Position haben wie der betroffene Partikel. So etwas sollte allerdings nie vorkommen, da wir die nicht komprimierbare Navier-Stokes Gleichung benutzen. Ansonsten wird wie folgt summiert: Masse $*$ dem Druck beider Partikel, summiert durch das Quadrat der Dichte von beiden Partikeln, summiert $*$ den smoothing kernel.

4.6 Viskosität

Die Viskosität ist dieser Teil der Navier-Stokes Gleichung $\mu \bigtriangledown^2 v$. Die Methode der Viskosität hat die gleichen Übergabewerte wie die Dichte und der Druck, eine Partikel Liste mit allen relevanten Partikeln und den betreffenden Partikel. Das Ergebnis ist wieder ein Vector3. Es wird wieder über alle Partikel iteriert und der betreffende Partikel wird ignoriert. Die Summierung ist Masse $*$ Nachbar Partikel Geschwindigkeit, minus Partikel Geschwindigkeit durch Nachbar Partikel Dichte $*$ smoothing kernel. Nach der Summierung wird das Ergebnis noch mit der Viskosität, die in der PartikelCloud gespeichert ist, durch die Partikel Dichte multipliziert.

4.7 Geschwindigkeit

Die neue Geschwindigkeit ist die Beschleunigung plus die vorher schon bestehende Geschwindigkeit. Die Beschleunigung wird berechnet durch die vereinfachte Navier-Stokes Gleichung, die jetzt durch die vorher gelösten Formeln zu lösen ist $\rho g - \bigtriangledown p + \mu \bigtriangledown^2 v$.

4.8 Behälter Kollision

Die Behälter Kollision wird in einer Schleife wiederholt berechnet. Am Anfang dieser Schleife wird berechnet ob eine Kollision statt findet. Dafür wird die Methode ContainerCut benutzt. Diese bekommt die Position des Partikels als Vektor3 übergeben und die Richtung in die der nächste Schritt gehen soll (oder was davon übrig ist falls schon mindestens einmal reflektiert wurde). Auch dies ist ein Vektor3. In der ContainerCut Methode befindet sich eine for Schleife

um über alle Dreiecke des Behälters zu iterieren. Die Liste der Dreiecke ist eigentlich eine Liste von Positionen. Also eine Liste mit jeweils 3 Vektor3, die dann ein Dreieck bilden. Dass bedeutet, die Schleife läuft die Länge der Liste durch 3 mal durch. Die einzelnen Dreiecke werden dann in der Methode PlainVertices auf einen Schnitt getestet. Hier zur Veranschaulichung ein pseudo Code der Methode ContainerCut:

```
bool ContainerCut(Vector3 position, Vector3 strahlRichtung)
    Vector3 tempNormale = (0,0,0)
    float tempLambda = 2 //So initialisiert, das es auf jeden fall größer ist als ein richtiges
Lambda
    for(Behälter Dreieckspunkteliste / 3)
        bool cut = PlainVertices(Dreieckpunkt 1, Dreieckpunkt 2, Dreieckpunkt 3,
        position, strahlRichtung)
        if(cut)
            if(globalLambda != 0)
//Die cut Berechnung speichert das errechnete Lambda in einer Globalen Variable
                if(tempLambda > globalLambda or tempLambda == 0)
                    tempLambda = globalLambda
                    tempnormal = globalNormal
            else
                if(tempLambda == 2)
                    tempLambda = globalLambda
                    tempnormal = globalNormal
    if(tempLambda != 2)
        globalLambda = tempLambda
        globalNormal = tempNormale
        return true
    return false
```

Die PlainVertices Methode bekommt alle Dreiecke übergeben als Vektor3, die Position des Partikels auch als Vektor3 und die Richtung des nächsten Schrittes ebenfalls als Vektor3. PlainVertices leitet alle Dreiecke einzelnd weiter an die Methode PlainCrossed. Hilft aber schon mal aus dem Dreieck eine Ebene zu machen, indem die Ebenennormale gebildet wird. PlainCrossed bekommt dann die Ebenennormale als Vektor3, die Position des Partikels und Richtung als Vektor3 und einen Punkt auf der Ebenen als Vektor3 übergeben. Wenn das Skalarprodukt aus der Ebenennormalen und der Richtung 0 ist, verläuft der Strahl parallel zur Ebene und es wird false zurück gegeben. Ansonsten wird das Lambda wie folgt berechnet: minus Skalar von Ebenennormale und Partikel Position plus Skalar von Ebenennormale und Punkt auf der Ebene durch das Skalar von Ebenennormale und Richtung des Schrittes. Der pseudo Code von PlainCrossed der Berechnung von Lambda sieht so aus:

```
bool PlainCrossed(Vector3 ebenennormale, Vector3 partikelPosition, Vector3 strahlRichtung,
Vector3 punktAufDerEbene)
if(ebenennormale • strahlRichtung == 0)
        return false
globalLambda = (- (ebenennormale • partikelPosition) + ebenennormale • punktAufDerEbene)
/ ebenennormale • strahlRichtung
if(globalLambda <= 1 and globalLambda >= 0)
        globalNormal = ebenennormale
        return true
return false
```

Wenn das berechnete Lambda jetzt zwischen 0 und 1 liegt, hat der nächste Schritt die Ebene geschnitten. Es findet noch eine zweite Prüfung statt, ob das Lambda kleiner ist als das vorher gefundene. Das kleinste Lambda wird gespeichert. Sollte beides zutreffen, wird true zurück gegeben, anderenfalls false. Danach wird Lambda und die dazu gehörige Normale nochmal local gespeichert, falls die globale Variable in kommenden Berechnungen verändert werden sollte. Diese beiden Methoden sind dazu da um den Schnitt zu erkennen. Der Rest der Schleife ist dafür da, zu entscheiden was bei dem Schnitt mit dem nächsten Schritt passieren soll.

Wenn kein Schnitt statt gefunden hat, wird der Rest der Schleife übersprungen. Es wird also mit if geprüft, ob cut true ist. Da die Schleife auch über cut prüft springt die Berechnung der Kollision gleich an ihr Ende. Das bedeutet der Behälter wurde nicht geschnitten und der nächste Schritt kann ohne Probleme ausgeführt werden. Wenn ein Schnitt gefunden wurde, gibt es noch eine Fallunterscheidung, zwischen Lambda != 0 oder Lambda == 0. Wenn Lambda != 0 ist, wird erstmal die neue Länge des Schrittes ausgerechnet, die der Strahl, der die Bewegung des Partikels darstellt hat, sobald er von der Wand reflektiert wird. Dies ist die Berechnung $uebrigeLaenge = uebrigeLaenge - (uebrigeLaenge/100) * (lambda * 100)$. Danach wird die neue Position des Partikels und der gespiegelte Strahl berechnet, um zu prüfen ob es einen weiteren Schnitt einer Wand gibt. Die neue Position befindet sich dann auf der Ebene, wo der Strahl eine Wand geschnitten hat. Deswegen auch die Fallunterscheidung Lambda == 0, weil nun immer ein Schnitt mit Lambda == 0 gefunden wird. Die Berechnung für die neue Position ist einfach, da wir das Lambda ja bereits haben $neuePosition = alteposition + strahlRichtung * lambda$. Unser neuer Strahl ist die schon bekannte Formel zur Reflektierung des Strahls $*$ der übrig gebliebenden Länge des Schrittes $2 * ((strahl * -1) * (normal * -1)) * (normal * -1) - (strahl * -1) * uebrigeLaenge$ in unserem Fall zeigen die Normalen, vom Behälter aus gesehen, nach außen und der Strahl zur Wand. Deswegen werden sowohl Strahl als auch Normale umgedreht, weil diese in umgekehrter Richtung vorliegen müssen. Jetzt kann auf einen zweiten Schnitt geprüft werden, in dem die Methode ContainerCut mit der neuen Position und dem neuen Strahl aufgerufen wird. Da Lambda beim zweiten Schnitt mindestens einen Schnitt findet und zwar den bei Lambda = 0, wird dieser außer acht gelassen. Sollte aber ein anderer Schnitt gefunden werden, wird die Position vorerst auf die neue Position gesetzt und der Strahl wird auf den neuen Strahl gesetzt, allerdings ohne die Kürzung durch übrigeLänge. Die Position wird sich nur gemerkt, der Schritt wird noch nicht ausgeführt und die Länge wird noch nicht aktuali-

siert aber die übrigeLänge wird kleiner. Danach wird der nächste Schleifendurchlauf ausgeführt.

Wenn es nur den Lambda = 0 Schnitt gibt, gibt es 4 verschiedene Szenarien. Der Schritt mit dem gespiegelten Strahl befindet sich innerhalb des Behälters, da die Position und der Strahl schon vorher berechnet waren. Für den zweiten Schnitt werden die Werte so übernommen und cut, die Variable für die Schleifenwiederholung wird auf false gesetzt. Die nächste Unterscheidung ist für Kanten und Ecken gedacht. Sollte der Partikel genau in einer Kante/Ecke landen, so wird auch nur der Lambda = 0 Schnitt gefunden. Aber der Partikel ist nicht mehr in dem Behälter nachdem der Schritt gemacht wurde, also wird der Schritt hier zurück geworfen. Es wird Geprüft, ob der zurückgeworfene Strahl in dem Behälter ist und wenn dem so ist, wird die Position durch die neu berechneten Position ersetzt und der Strahl wird mit -1 $*$ genommen, sowie mit der übrigen Länge. Auch hier wird cut auf false gesetzt, da kein weiterer Schnitt vorhanden ist. Bei der vorletzten Prüfung wird getestet, ob sich der Partikel auf dem Boden befindet. Hierfür wird geprüft, ob sich der Partikel im Behälter befindet, sollte er sich nur auf der x und z-Achse bewegen können. Die y-Achse (Erdanziehungs-Achse) wird hier außer acht gelassen. Auch hier wird die cut Variable wieder auf false gesetzt. Wenn keiner der genanten Fälle zutrifft, wird angenommen, dass sich der Partikel fast im ruhenden Zustand befindet. Der Strahl wird dann auf einen nuller Vektor gesetzt, da zu kleine Werte Fehler bei der Schnittberechnung hervorbringen können und cut wird auf false gesetzt. Sollte bei der Prüfung des ersten Schnittes Lambda = 0 sein, dann ist dies entweder der mindestens zweite Durchlauf der Schleife, da sich die Position dann auf der Ebene befindet oder der Schritt endet genau auf der Ebene. In diesem Fall wird cut auf false gesetzt, da kein richtiger Schnitt vorhanden ist, danach werden fast die gleichen Prüfungen gemacht wie vorher. Erst wird geprüft, ob sich Position plus Strahl im Behälter befinden, falls dies nicht der Fall ist, wird statt dem Strahl mit dem gespiegeltem Strahl getestet. Danach kommt die Prüfung ohne die y-Achse, ansonsten der default Fall mit dem nuller Vektor.

4.9 Berechnungsklasse

In der Berechnungsklasse geht es hauptsächlich um die Methode StartCalculation. Diese enthÄ¤lt die Dauerschleife für die Berechnung. Die restlichen Methoden sind nur Hilfsmethoden. Die Berechnungen werden wie folgt durchgeführt: Die gesamte Methode besteht nur aus einer while Schleife die immer läuft. Diese Methode wird in einem eigenen thread gestartet, damit die Berechnungen nebenbei laufen können. Am Anfang jedes Schleifendurchlaufs wird geprüft, ob das Raster, wo die Partikel einsortiert sind, aktualisiert werden muss. Danach startet eine von zwei for Schleifen. In beiden wird über alle Partikel iteriert und in der ersten Schleife erst die Dichte und dann der Druck, der für den Druck Gradienten benötigt wird, berechnet. Bei der zweite for Schleife, wird der Druck Gradient und die Zähflüßigkeit berechnet. Nach den Schleifen kommt der Kollisionscheck und wenn die Position danach sicher ist, wird diese für den Partikel übernommen. Wenn das flag gesetzt ist, wird die Position auch in der Save Datei gespeichert für späteres Laden.

4.10 Szene Speichern

Das Speichern der Szene findet nebenbei in der Berechnungsklasse statt. Dafür kann beim Erstellen der Berechnungsklasse ein Boolen Wert übergeben werden, der bestimmt, ob gespeichert wird oder nicht. Sobald die nächste Position des Partikels fest steht, wird diese in die Datei geschrieben, dessen Pfad in der Berechnungsdatei gespeichert ist.

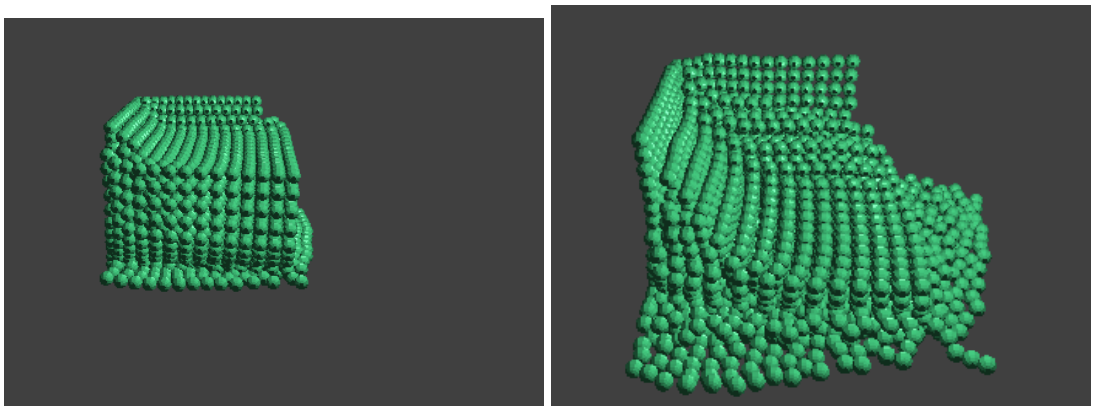
4.11 Szene Laden

Die load Klasse ist eher klein. Sie bekommt einen Datei Pfad als String übergeben und die Rootnode. Dann liest die Klasse die erste Zeile der Datei, um zu wissen, wie viele Partikel es in der Simulation gibt. Danach liest sie so viele male wie es Partikel gibt, um die Szene zu füllen und wartet dann bis die Visualisierung für den Nutzer geladen ist. Wenn die Visualisierung geladen ist, werden nach und nach alle Schritte aus der Datei gelesen und die Partikel in der Szene entsprechend bewegt.

5 Evaluation

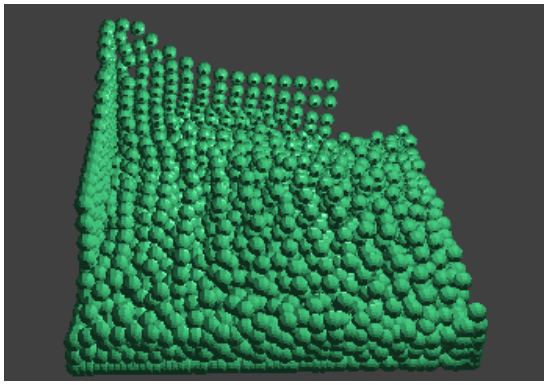
8	27	64	125	206
70 - 80	20 - 30	13	5	3

Die obere Reihe der Tabelle zeigt die Partikel in der Szene und die untere Reihe die FPS mit der jeweiligen Partikelanzahl. Da die Frames der Draw Methode nichts mit der Bewegung des Fluids zu tun haben, ist dies keine zuverlässige Quelle. Ein Frame wird deshalb gesehen als eine Bewegung des Fluids, also eine Bewegung aller Partikel ist ein Frame. Die Frames wurden bei einer Szene der Größe 2 x 2 x 2 und einem h von 0.1 aufgenommen. Da die erwünschte Anzahl an Frames nicht ereicht werden konnte, wurde das Save und Load Prinzip eingeführ. Die Frames bestimmen in einer gewissen Weise die Geschwindigkeit, in der sich die Flüssigkeit bewegt. Sollte die Flüssigkeit also zu viele Frames haben, muss die Schrittgröße verringert werden, um ein realistisches Ergebnis zu bekommen. Das die Framerate so rapide sinkt, dafür ist hauptsächlich das neu zeichnen der Kugeln schuld. Das könnte reduziert werden indem nur die Flächen gezeichnet werden, die die Oberfläche des Fluids ausmachen. Somit müssen die Flächen, die sowiso nicht zu sehen sind, nicht mehr gezeichnet werden und diese überwiegen in den meisten Fällen. Die Berechnungen der Dichte, Druck und Viskosität gehen auch exponentiell in die Höhe, was die Berechnungszeit in die Höhe treibt. Dies soll durch das Gitter der RasterUnit abgeschwächt werden, aber da die Szene relativ klein ist, ist dies noch nicht lohnenswert. Das was hierbei am meisten Rechenleistung verbraucht, ist das Zusammenstellen der Liste der Partikel von den Nachbar Rastern.

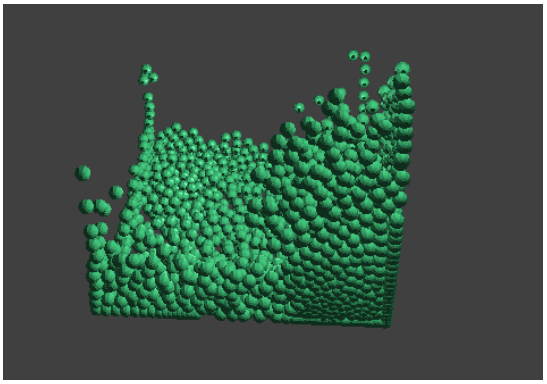


(a) Verlauf 1

(b) Verlauf 2



(c) Verlauf 3



(d) Verlauf 4

Es gibt bei SPH eine große Auswahl. Zum einen gibt es mehrere smoothing kernel [6] [10] und zum anderen eben so mehrere Randbedingungs möglichkeiten für Behälter und andere Objekte. Wenn die kernel für den gleichen Zweck sind, wie z.B. Dichte, Druck usw., sind sie sich änlich, weil sie das gleiche Ziel haben. Bei den Randbedingungen ist das allerdings anders. Die Randbedingungen können großen Einfluss auf die Rechenleistung haben. In dieser Arbeit habe ich mich für etwas eigenes entschieden, eine reflektierende Eigenschaft mit raytracing. Das Gute daran ist, dass die Rechenzeit von den Wänden des Behälters abhängig ist und somit geringer als bei anderen Methoden bei denen z.B. statische Partikel benutzt werden. Das Schlechte daran ist, dass es viele Spezialfälle gibt. Um die Partikel am flüchten zu hindern, kommt in seltenen Fällen immernoch der standard Fall zum Einsatz und die Bewegung des Partikels wird auf 0 gesetzt. Die erforderlichen Frames für live Berechnungen wurden dennoch nicht erreicht, also hätte man auch die kostspieligere Methode nehmen können und dafür ohne die Spezialfälle arbeiten.

6 Zusammenfassung und Fazit

Wir wissen jetzt wie SPH funktioniert, wie dies umgesetzt wurde und wie die Randbedingungen aussehen, sowie umgesetzt wurden. Wir kennen das Ergebnis und wo die Probleme liegen deswegen schauen wir uns jetzt an, was noch verbessert werden kann und was zusätzlich eingeführt werden könnte.

6.1 Verbesserungen

Bei der Behälter Kollision hätte ich im nachhinein lieber eine andere Methode gewählt. Zum Beispiel statische Partikel die den Behälter bilden. Dies ist physikalisch etwas korrekter und es müssen keine weiteren Berechnungen implementiert werden, da diese einfach in die normalen Partikel Berechnungen mit eingebunden werden. Nur der Behälter mösste dann per Hand erstellt werden. Um die Arbeit weiter zu führen, gäbe es auch noch einiges zu Tun. Es gibt eine Formel zur Feststellung der Oberfläche [2]. Damit kann eine saubere Oberfläche erstellt werden, so dass keine Kugeln mehr zu sehen sind, sondern nur die Oberflächen gezeichnet werden, die auch gesehen werden. Die Funktion funktioniert durch die Dichte, da die Dichte innerhalb der Flüßigkeit höher ist, als die Dichte der Partikel an der Oberfläche. Die Dichte ist die Masse aller umliegenden Partikel summiert mit einer Gewichtung mal genommen. Deswegen ist die Dichte an der Oberfläche mit weniger Partikeln kleiner. Eine Interaktion mit anderen Strukturen, wie zum Beispiel einem Boot oder einer Holzkiste, die auf dem Wasser schwimmt, ist auch noch möglich. Zu realisieren wäre das z.B. durch weitere Partikel Clouds, diese sind sozusagen weitere verformbare Körper [6]. Sollte die Viscosität hoch genug eingestellt werden, wird die Flüssigkeit immer fester. Somit könnten auch andere Stoffe als Flüssigkeiten simuliert werden. Es sind sicherlich noch Performanceverbesserungen möglich. Z.B. durch andere Datenstruckturen die kleiner sind, womit die Hardware besser rechnen kann oder Umgestaltung der erstellten. Bei den RasterUnits werden die Nachbarn momentan noch als RasterUnit gespeichert. Es könnte aber die Berechnungen beschleunigen, wenn die RasterUnit gleich alle Partikel der Nachbarraster bereits als Liste kennen würde.

6.2 Zukunft

In der Gegenwart und sicherlich auch in der Zukunft gibt es Professoren und Forscher, die an SPH arbeiten. Zum einen um die Berechnungen schneller zu machen, dabei geht es nicht um die mathematischen Berechnungen von SPH, weil man dort zur Zeit noch nicht an dem n hoch zwei vorbei kommt. Es geht eher um das drumherum, wie bei der Nachbarnsuche, was die Berechnungszeit bei größeren Szenen erheblich verbessert. Außerdem wird bei der

Oberflächenfindung noch an Verbesserungen getestet, im Moment durch einen Marching Cubes Algorithmus. Durch Marching Cubes kann die Zeichnung beschleunigt werden, indem die Dreiecke vergrößert werden. Dadurch sieht die Oberfläche kantiger aus, es wird bei der Darstellung gespart für mehr Frames per second. Aber auch um es genauer zu berechnen, zum Beispiel mit mehr Variablen wie Temperatur und weiterer Einfluss wie Wind. Dafür muss nur eine neue SPH Formel erstellt werden und vieleicht ein neuer kernel.

7 Anhang

7.1 Glossar

SPH	Smoothed Particle Hydrodynamics
Partikel	Ein Partikel ist ein kleiner Berechnungsbereich, der zur darstellung dient.

Literaturverzeichnis

[1] A. Chorin and J.-E. Marsden. A mathematical introduction to fluid mechanics. *Springer Verlag*, 2000.

[2] N. Foster and R. Fedkiw. Practical animation of liquids. *Stanford University and PDI/DreamWorks*, 2001.

[3] R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 1977.

[4] C. Höfler. Smoothed particle hydrodynamics (sph) codes zur numerischen vorhersage des primärzerfalls an brennstoffeinspritzdüsen. *Fakultät für Maschinenbau Karlsruher Institut für Technologie*, 2013.

[5] M. Kelager. Lagrangian fluid dynamics using smoothed particle hydrodynamics. *Department of Computer Science University of Copenhagen*, 2006.

[6] G.R. Liu, M.B. Liu, and K.Y. Lam. Adaptive smoothed particle hydrodynamics for high strain hydrodynamics with material strength. *Springer-Verlag*, 2006.

[7] L.B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 1977.

[8] J.J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 1992.

[9] M. Muller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. *Als Fortschritt der Fachtagung Computer Animation von Eurographics/SIGGRAPH*, 2003.

[10] J.M. Owen, J.V. Villumsen, P.R. Shapiro, and H. Martel. Adaptive smoothed particle hydrodynamics: Methodology ii. *The Astrophysical Journal Supplement Series*, 1998.

[11] A.P. Singh and R.D. Borker. Smoothed particle hydrodynamics. *Department of Aerospace Engineering Indian Institute of Technology*, 2010.

[12] R.F. Stellingwerf and C.A. Wingate. Impact modelling with sph. *Memorie della Societa Astronomia Italiana*, 1994.

[13] M. Vesterlund. Simulation and rendering of a viscous fluid using smoothed particle hydrodynamics. *Umea University*, 2004.

Videoquellen:
https://www.youtube.com/watch?v=SQPCXzqH610
Ein Video der Firma AMD, der Vortrag gehalten von Alan Heirich, 2010.

https://www.youtube.com/watch?v=f5jKknVBflM
Aufgenommene Vorlesung von NPTEL (National Programme on Technology Enhanced Learning), gehalten von Dr. Arup Kumar Das, 2016.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. September 2017 Maximilian Reinhardt