Physikalisch-basierte Simulation in der Computergraphik

Schriftliche Ausarbeitung

Echtzeit SPH Wasserkanal

Peter Wichert, Kirill Menke, Linda Stadter

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

1 Einleitung

Das Projekt soll einen realistischen Wasserfluss in einer begrenzten 3D-Umgebung in Echtzeit simulieren. Als Inspiration dient das Kinderspielzeug 'Aquaplay', bei dem Kinder über Schleusen, Kurbeln, Pumpen oder Schranken den Wasserfluss verändern können.

Zur Visualisierung des Wassers sollen zunächst kugelförmige Objekte dienen, welche durch Screen Space Rendering oder Marching Cubes zu realistisch aussehendem Wasser erweitert werden könnten. Für ein ausreichend detailliertes Verhalten der Simulation werden etwa 30.000 Partikel simuliert. Als Kriterium für Echtzeitanforderungen sollen 30 Bilder pro Sekunde erreicht werden. Dies soll auf handelsüblicher Hardware wie der NVIDIA GTX 1070 möglich sein, ohne die Anzahl der Partikel zu verringern.

Als Entwicklungsumgebung dient die Spiel-Engine Unity, welche hauptsächlich für das Rendering eingesetzt werden soll. Echtzeit-Interaktionen lassen sich mit dieser Laufzeitumgebung einfach umsetzen, was neben des Renderings einen weiteren Vorteil darstellt. Außerdem bietet Unity eine übersichtliche Nutzeroberfläche.

2 Theorie

Unsere Echtzeit Wassersimulation basiert hauptsächlich auf der Methode der Geglätteten Teilchen-Hydrodynamik (SPH), welche durch durch eine verbesserte Suche der Nachbarpartikel beschleunigt werden soll. Zur numerischen Integration wird sowohl das Leapfrog-Verfahren als auch das explizierte Euler-Verfahren eingesetzt.

2.1 Wissenschaftlicher Hintergrund

Die Methode von SPH wurde erstmalig von Gingold und Monaghan [Gingold and Monaghan, 1977] und Lucy [Lucy, 1977] vorgestellt. Ursprünglicherweise wurde sie dabei verwendet, um astrophysikalische Probleme wie die Dynamiken von Sternen darzustellen.

Müller et al. [Müller et al., 2003a] wandte zuerst SPH für Computergrafik Simulationen in Echtzeit an. Es wurde eine Methode gefunden, die schnell genug war, bis zu 5.000 Partikel in interaktiven Systemen abzubilden.

2.2 SPH

In Smooth Particle Hydrodynamic werden Eigenschaften von kontinuierlichen Flüssigkeitsmengen approximiert, indem man eine diskretisierte Repräsentation durch Partikel für die Navier–Stokes Gleichungen benutzt. Die Partikel interagieren innerhalb eines bestimmten Radius h miteinander und besitzen eigene physikalische Eigenschaften wie Dichte, Masse oder Druck.

Smoothing Kernels Um die diskreten Werte der Partikel zur Berechnung verschiedener kontinuierlicher Felder zu glätten werden die Kernelfunktionen aus Eq. (1), Eq. (2), Eq. (3) und Eq. (14) verwendet. Für den Gradienten eines Feldes wird entsprechend der Gradient einer Kernelfunktion benutzt. Diese Fuktionen sind normalisiert und gewichten symmetrisch den Einfluss von Partikel aufeinander abhängig von ihrer Distanz zueinander. Ein Partikel wird nur von anderen innerhalb des Radius h beeinflusst.

$$W_{poly6}(\vec{r},h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - |\vec{r}|^2)^3, \\ \text{falls } 0 \leq |\vec{r}| \leq h \\ \\ 0, \\ \text{sonst} \end{cases} \tag{1}$$

$$\bigtriangledown W_{poly6}(\vec{r},h) = \frac{945}{32\pi h^9} \begin{cases} \vec{r}(h^2 - |\vec{r}|^2)^2, \\ \text{falls } 0 \leq |\vec{r}| \leq h \\ \\ 0, \\ \text{sonst} \end{cases} \tag{2}$$

$$\nabla W_{spiky}(\vec{r},h) = -\frac{45}{\pi h^6}(h^2-r^2)^3 \begin{cases} \frac{\vec{r}}{|\vec{r}|}(h-|r|)^2, \\ \text{falls } 0 \leq |\vec{r}| \leq h \\ \\ 0, \\ \text{sonst} \end{cases} \tag{3}$$

$$\nabla^2 W_{viscosity}(\vec{r},h) = \frac{45}{\pi h^9} \begin{cases} (h-|r|), \\ \text{falls } 0 \leq |\vec{r}| \leq h \\ \\ 0, \\ \text{sonst} \end{cases} \tag{4}$$

Dichte und Druck Mit dem für SPH typischen Smoothing Kernel aus Eq. (1) wird die Dichte aus Eq. (5) an den Partikeln interpoliert. Die aus der idealen Gaszustandsgleichung abgeleiteten Formel aus Eq. (6) mit p_0 als Restdichte nach Müller et al. [Müller et al., 2003a] sorgt abhängig von ihrem Parameter k für zu hohe Kompressibilität oder Federverhalten zwischen Partikeln.

$$\rho_i = \sum_i m_j W(\vec{r}i - \vec{r}j, h) \tag{5}$$

$$p_i = k(\rho - \rho_0) \tag{6}$$

Daher wird stattdessen die Tait-Gleichung aus Eq. (7) genutzt.

$$p_i = k((\frac{\rho}{\rho_0})^7 - 1) \tag{7}$$

Kräfte Für jeden Partikel kann nun die auf ihn wirkende Kraft bestimmt werden. Die Kraft setzt sich dabei wie in Eq. (8) mit den entsprechenden Formeln Eq. (9), Eq. (10), Eq. (11) zusammen. Dabei ist Eq. (9) eine leicht veränderte Version der normalen interpolierten Kraft für Druck, die sich symmetrisch zwischen Partikeln verhält. Die Variable μ wird eingeführt um die Viskosität der Flüssigkeit zu kontrollieren.

$$\vec{f} = \vec{f}^{pressure} + \vec{f}^{viscosity} + \vec{f}^{external} \tag{8}$$

$$\vec{f}^{pressure}i = -\sum jm_j(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) \nabla W(\vec{r}i - \vec{r}j, h) \tag{9}$$

$$\vec{f}^{viscosity}i = \mu \sum jm_j \frac{\vec{v}i - \vec{v}j}{\rho_j} \nabla^2 W(\vec{r}i - \vec{r}j, h) \tag{10}$$

$$\vec{f}_i^{external} = \vec{g} \tag{11}$$

2.3 Oberflächenspannung

Um ein überzeugendes Verhalten bei Wasserspritzern und an der Wasseroberfläche zu erzeugen, wird zusätzlich die molekulare Attraktion und Repulsion zwischen Wassermolekülen modelliert. Das gewählte Modell folgt der Arbeit von Akinci et al.[Akinci et al., 2013].

Eine neue Kraft für Oberflächenspannung wird wie in Eq. (12) in die bisherige Kraftberechnung aus Eq. (8) eingefügt.

$$\vec{f} = \vec{f}^{pressure} + \vec{f}^{viscosity} + \vec{f}^{external} + \vec{f}^{surface} \tag{12}$$

Das molekulare Verhalten wird durch Eq. (13) ausgedrückt. γ sei hier ein von uns wählbarer Koeffizient.

$$\vec{f}_{i \leftarrow j}^{\text{cohesion}} = \gamma m_i m_j C(|\vec{x}_i - \vec{x}_j|) \frac{\vec{x}_i - \vec{x}_j}{|\vec{x}_i - \vec{x}_j|} \tag{13}$$

Die Spline-Funktion aus Eq. (14) sorgt ab dem Radius h für eine zunächst anziehende Wirkung, sowie einen ansteigenden abstoßenden Effekt, sollten zwei Partikel zu nahe beieinander sein.

$$C(r) = \frac{32}{64\pi h^9} \begin{cases} (h-r)^3 r^3, \\ \text{falls } 2r > h \wedge r \leq h \\ \\ 2(h-r)^3 r^3 - \frac{h^6}{64}, \\ \text{falls } r > 0 \wedge 2r \leq h \\ \\ 0, \\ \text{sonst} \end{cases} \tag{14}$$

Um zu verhindern, dass beim Formen eines Tropfens durch Oberflächenspannung, die Ausgangsposition der betroffenen Partikel maßgeblich die finale Form beeinflusst, fügen wir eine weitere Kraft aus Eq. (15) hinzu. Diese sorgt dafür, dass Partikel-Mengen unabhängig von Ausgangsposition versuchen eine Kugelform einzunehmen und dadurch die Oberfläche ihres Tropfens minimieren.

$$\vec{f}_{i \leftarrow j}^{\text{curvature}} = -\gamma m_i (\vec{n}_i - \vec{n}_j) \tag{15}$$

Zunächst wird für jedes Partikel ein Normal-Vektor wie in Eq. (16) berechnet. Hier sollen sich Partikel an der Oberfläche von denen innerhalb eines Wasserkörpers dadurch unterscheiden, dass letztere einen Normal-Vektor von $\sim \vec{0}$ besitzen. Hier würde das Benutzen eines geglätteten Feldes optimale Ergebnisse erreichen.

$$\vec{n}_i = h \sum_j \frac{m_j}{\rho_j} \nabla W(\vec{r}_i - \vec{r}_j, h) \tag{16}$$

Um den Zusammenhalt von wenigen Partikeln an den von größeren Gruppen anzugleichen, wird in Eq. (17) außerdem ein Korrekturfaktor eingeführt.

$$\vec{f}_{i\leftarrow j}^{\text{surface}} = \frac{2\rho_0}{\rho_i + \rho_j}(\vec{f}_{i\leftarrow j}^{\text{cohesion}} + \vec{f}_{i\leftarrow j}^{\text{curvature}}) \tag{17}$$

2.4 Integration

Die Kraft jedes Partikels wird verwendet, um dessen Beschleunigung via $F = ma \Leftrightarrow a = F/m$ zu ermitteln. Die Beschleunigung wird anschließend benutzt, um Position und Geschwindigkeit für den nächsten Zeitschritt zu erhalten. Die Genauigkeit der Integration und damit Stabilität verbessern wir, indem wir explizite Euler durch Leapfrog Integration ersetzen.

Leapfrog Integration Die Leapfrog Integration berechnet standardmäßig abwechselnd Geschwindigkeiten v und Positionen r an verschiedenen Zeitpunkten. Die Größe des Fehlers wird so von $\mathcal{O}(n)$ auf $\mathcal{O}(n^2)$ reduziert. Die energie-erhaltende Eigenschaft der Integration für das System trägt ebenfalls zur Stabilität bei. Für die Implementation von Leapfrog wird eine von Cossins [Cossins, 2010] vorgestellte Variation benutzt. Die Variante berechnet an einem halben Zeitschritt die Formeln Eq. (18) und Eq. (19). Diese werden benutzt, um die Beschleunigung $a_{i+1/2}$ für Eq. (20) und Eq. (21) im nächsten ganzen Zeitschritt zu erhalten.

$$v_{i+1/2} = v_i + \frac{\Delta t}{2} a_i \tag{18}$$

$$r_{i+1/2} = r_i + \frac{\Delta t}{2} v_i \tag{19}$$

$$v_{i+1} = v_i + \Delta a_{i+1/2} \tag{20}$$

$$r_{i+1} = r_i + \frac{\Delta t}{2}(v_i + v_{i+1}) \tag{21}$$

Boxkollision Der Raum der Partikel wird durch eine quadratische Box begrenzt. Kollisionen mit den Seiten der Box werden durch einfache Positionsabfragen behandelt. Diese verhindern, dass die Partikel die Begrenzungen überschreiten. Außerdem werden die Geschwindigkeiten der entsprechenden Richtungen umgekehrt, um die Partikel von der Box abprallen zu lassen. Eine Dämpfungsvariable wird eingeführt, die die Geschwindigkeit dabei zusätzlich abmildern kann.

2.5 Effiziente Nachbarsuche

Die Nachbarpartikel eines Partikels sind die Menge der Partikel, welche sich innerhalb von h befinden und somit Einfluss auf den Partikel besitzen. Die Suche dieser Nachbarpartikel stellt einen sehr aufwändigen Schritt des SPH-Algorithmus dar. Um diese effizient zu finden, teilen wir den Raum in ein uniformes Gitter auf. Eine Zelle besitzt die Größe des Radius h, sodass die Nachbarpartikel nur in den anliegenden 26 Zellen und in der Zelle des Partikels gesucht werden müssen. Durch ihre Positionen im Raum werden die Partikel je einer Zelle zugewiesen. Dies erfolgt zunächst durch eine Diskretisierung jeder Partikelposition (x, y, z) in Eq. (22).

$$(i, j, k) = (\left\lfloor \frac{x}{h} \right\rfloor, \left\lfloor \frac{y}{h} \right\rfloor, \left\lfloor \frac{z}{h} \right\rfloor) \tag{22}$$

Anschließend kann der diskretisierte Zellenwert in Kombination mit der Hash-Funktion aus Eq. (23) eingesetzt werden. Diese bildet eine 3D-Position eines Partikels auf einen flachen 1D-Wert ab.

$$hash(i, j, k) = (i\ p_1\ \textbf{\textit{xor}}\ j\ p_2\ \textbf{\textit{xor}}\ k\ p_3)\ \textbf{\textit{mod}}\ n \tag{23}$$

Die Variable n stellt die Anzahl der Zellen beziehungsweise der Partikel dar und p_1, p_2 und p_3 sind große Primzahlen. Dies wird Spatial Hashing genannt und wird eingesetzt, damit eine endliche Anzahl an Zellen für einen unendlich großen Raum ausreicht. [Müller et al., 2003b]

Dann können die Partikel anhand ihrer zugewiesenen Zelle sortiert werden. Für jede Zelle wird der Abstand zu dem ersten Partikel der Zelle gespeichert. Diese Idee wurde 2008 von Nvidia vorgestellt. [Nvi]

3 Implementierung

Das Projekt wird in der Laufzeit und Entwicklungsumgebung Unity implementiert. Dazu wird ein sogenanntes GameObject erstellt, welches ein eigens geschriebenes C#-Skript zugewiesen wird. Dieses Skript wird verwendet, um die Simulation zu initialisieren und aufzurufen. Ein Zeitschritt wird mithilfe der Update-Funktion von Unity einmal pro Frame durchgeführt. Der eigentliche SPH-Algorithmus wird auf sieben Shader aufgeteilt, welche von der GPU ausgeführt werden. Es ist je ein Shader dafür zuständig, die Buffer zu initialisieren, die Partikel in Zellen einzuteilen, die Partikel anhand ihrer Zelle zu sortieren, die Dichte und im Anschluss die Force eines Partikels zu berechnen und zum Schluss den Integrationsschritt durchzuführen. Der Ablauf dieses Skripts ist in Pseudocode im Algorithmus 1 gezeigt.

Die Positionen in x-,y- und z-Richtung eines Partikels werden diskretisiert und anschliessend gehasht, um daraus den flachen Zellindex zu bestimmen. Um die Anzahl der Hash-Kollisionen möglichst gering zu halten, werden die 8-stellige Primzahlen 73856093, 19349663 und 83492791 eingesetzt und eine große Anzahl an Partikeln - und damit auch Zellen - benutzt.

Für das Sortieren der Partikel anhand ihrer zugewiesenen Zelle wird eine fertige Implementierung des Algorithmus Bitonisches Sortieren verwendet. Die-

Algorithm 1 C# Skript

```
1: Berechne Parameter
2: Initialisiere Partikel
3: Initialisiere Buffer
4: Initialisiere Shader
5: while true do                          ▷ Unitys Update Schleife
6:     for Zeitschritt i + 1/2 und i + 1 do
7:         Initialization Shader
8:             Intialisiere Buffer
9:         Partition Shader
10:            Weise jedem Partikel den Zellindex wie in 22 und 23 zu
11:        Bitonic Sort
12:            Sortiere nach Zellindex
13:        Offset Shader
14:            Speichere Abstand zu dem erstem Partikel jeder Zelle
15:        Density Shader
16:            Berechne mit 5
17:        Normals Shader
18:            Berechne mit 16
19:        Force Shader
20:            Berechne mit 7, 9, 10, 11, 12, 13, 14, 15, 17
21:        Integration Shader
22:            Berechne mit 18, 19 oder 20, 21
23:    end for
24: end while
```

ser Sortieralgorithmus kann parallel auf der GPU ausgeführt werden.

Das Einbeziehen der Nachbarpartikel bei der Berechnung der Dichte und im Anschluss der Kraft wird im Dichte- und im Force-Shader auf ähnliche Weise implementiert. Hierbei muss über 27 Zellen iteriert werden, welche Nachbarpartikel enthalten können. Mithilfe des zuvor gespeicherten Abstands kann dann effizient auf alle Partikel einer bestimmten Nachbarzelle zugegriffen werden. Zusätzlich wird überprüft, ob der Abstand zwischen den Partikeln kleiner als der Radius h ist.

Im Integration-Shader wird neben der Berechnung der Position und der Geschwindigkeit auch die Kollision mit der Box behandelt.

Die Partikel werden in der Update-Funktion in jedem Frame gerendert. Dazu wird eine von Unity vorgegebene Zeichen-Funktion aufgerufen, welche das gleiche Kugel-Mesh parallel auf der GPU zeichnet. Dadurch verhindern wir den unnötigen Overhead, eigenständige GameObjects pro Partikel zu erstellen. In diesem Schritt werden die Partikel anhand ihrer Dichte in einen blassen bis kräftigen Blauton gefärbt.

Neben dem GameObject, welches die Simulation behandelt, werden auch weitere Objekte benötigt. Der Raum, in welchem sich die Partikel bewegen können, wird durch eine Box begrenzt, welche zur

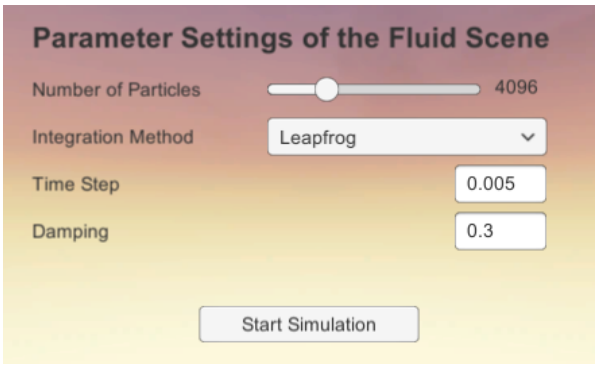

Fig. 1: Oberfläche vor dem Start, um die entsprechenden Parameter der Simulation einzustellen.

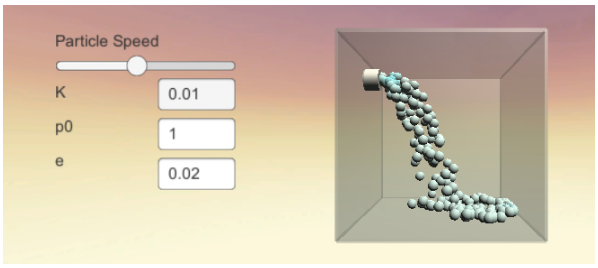

Fig. 2: Oberfläche während der Simulation, um die Parameter des SPH-Algorithmus anzupassen.

Laufzeit erzeugt wird. Die Box wird dabei durch ein eigenes Skript kontrolliert, das die Positionen der Seitenplatten anhand einer vorgegebenen Bodenplatte berechnet. Außerdem werden zwei Varianten implementiert, wie die Partikel spawnen können. Zum einen können alle Partikel auf einmal in quadratischen Form erzeugt werden. Zum anderen lassen sich die Partikel auch Frame für Frame aus einer Röhre erschaffen, um einen Wasserstrahl zu imitieren.

Zusätzlich werden zwei Nutzeroberflächen benötigt, um Parameter einerseits vor dem Beginn der Simulation wie in Bild 1 und andererseits während der laufenden Simulation wie in Bild 2 einstellen und anpassen zu können. Dazu werden fertige UI-Elemente von Unity verwendet. Beim Starten der Simulation findet ein Szenenwechsel statt. Dadurch werden nicht mehr benötigte Objekte entfernt und neue Objekte eingeblendet.

4 Ergebnisse und Evaluierung

Um die Stabilität und Performance zu Testen wurden mehrere Iterationen der Simulation mit variierendem Zeitschritt und gleichen Parametern ausgeführt und für ca. 30 Minuten laufen gelassen. Getestet wurden Simulationen sowohl mit kontinuierlicher Partikel-Erzeugung durch das Rohr, als auch mit Beginn aller Partikel in Boxform. Simulationen, die Leapfrog Integration mit einem Zeitschritt

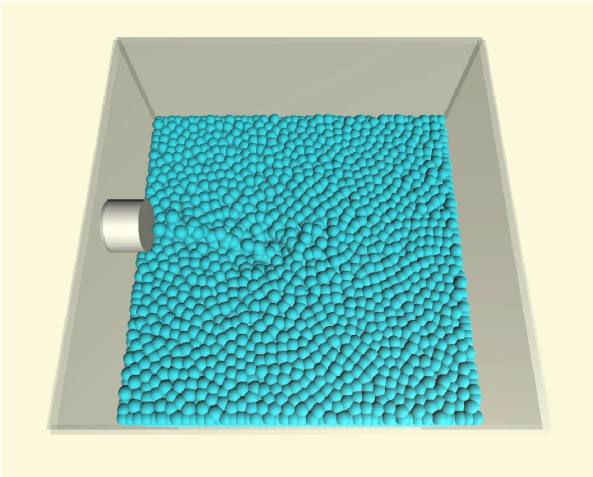

Fig. 3: Simulation mit 32.768 Partikeln und $\geq$ 60 FPS.

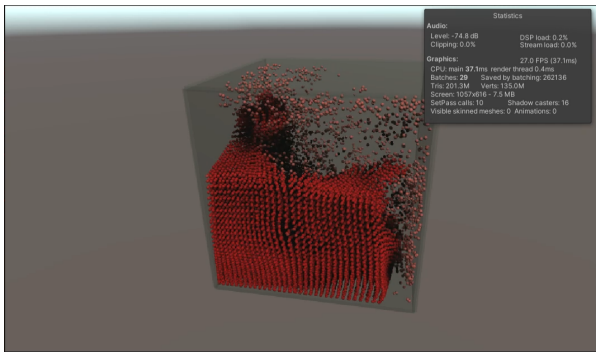

Fig. 4: Testsimulation wird instabil.

von ≤ 0.006 Sekunden benutzen, blieben dabei stabil.

Eine Simulation wie in Bild 3 mit $\Delta t = 0.005$, dem kontinuierlichen Erscheinen der Partikel, 32.768 Partikeln und den angegebenen Parametern aus Tabelle 1 beginnt mit ∼400 Frames pro Sekunde. Im Verlauf der Simulation sinken mit steigender Partikelzahl die FPS, bleiben aber immer über 60. Sobald der Spawnvorgang abgeschlossen ist, also keine neuen Partikel mehr hinzukommen und das Wasser zur Ruhe kommt, pendelt sich die Simulation bei ∼140 FPS ein. Eine ähnliche Simulation mit den gleichen Parametern aber sofortigem Partikelspawn als Box bleibt konstant über 100 FPS.

Test Hardware
AMD Ryzen 7 3700X
NVIDIA GeForce GTX 1070
32GB RAM

Parameter	Wert
μ	0.02
k	1
$p0$	1
γ	1
$damping$	0.5

Tab. 1: Parameterwahl für Stabilitätstests.

5 Beiträge

Kirill Menke hat über den SPH-Algorithmus recherchiert und die Methode gefunden, wie effizient auf die Nachbarpartikel zugegriffen werden kann. Er hat das Grundgerüst der Simulation in Unity aufgesetzt, sodass die einzelnen Schritte des SPH-Algorithmus auf der GPU ausgeführt werden können. Zusätzlich hat er das Rendering der Partikel implementiert, bei der Umsetzung des SPH-Algorithmus mitgewirkt und einen Modus zum vereinfachten Debuggen eingeführt. Außerdem hat er die Milestonepräsentation erstellt und vorgetragen.

Linda Stadter hat ebenfalls bei der Umsetzung des SPH-Algorithmus mitgewirkt. Zusätzlich hat sie sich um das Erzeugen der Box und die Kollisionen der Kugeln mit den Seiten gekümmert. Sie hat die beiden Nutzeroberflächen umgesetzt, die Farbcodierung der Partikel eingeführt und ein kontinuierliches Erzeugen der Partikel umgesetzt. Außerdem hat sie die Projektplanpräsentation erstellt und vorgetragen, bei der Milestonepräsentation mitgewirkt und den finalen Bericht geschrieben.

Peter Wichert hat über die Leapfrog Integration und Oberflächenspannung recherchiert und diese implementiert. Zusätzlich hat er eine automatische Berechnung der Parameter in Kombination mit dem Erzeugen der Partikel als Box eingeführt, den Debug-Modus ergänzt und Stabilitäts-Tests durchgeführt. Außerdem hat er die Milestone Präsentation erstellt und vorgetragen und am finalen Bericht mitgewirkt.

6 Diskussion

Da in unserem Modell die Geschwindigkeit v abhängig von der Beschleunigung a ist, müssen wir für die Leapfrog Integration zwei Berechnungen der Kräfte ausführen. Dieser zusätzliche Rechenaufwand wird jedoch für eine erhöhte Stabilität und dadurch größere Zeitschritte in Kauf genommen.

Hash-Collisions bei der Nachbarsuche lassen sich leider nicht vermeiden und fallen bei einer sehr geringen Anzahl an Partikeln auf. Daher werden diese geringen Partikelanzahlen von der Simulation ausgeschlossen.

Das Implementieren der Oberflächenspannung machte unsere Simulation deutlich realistischer. Allerdings könnte sie noch durch Glätten des Feldes für

die Normalen Berechnung etwas verbessert werden. Für die physikalische Simulation des Wassers war die Verbesserung der Optik des Wassers durch beispielsweise Space Rendering oder Marching Cubes nicht unbedingt nötig. Es ist ungewiss, ob diese Erweiterung die aktuelle Echtzeit-Performance signifikant negativ beeinflussen.

Die Interaktion mit dem Wasser durch Festkörper konnte aus zeitlichen Gründen nicht umgesetzt werden. Stattdessen lassen sich immerhin Parameter wie zum Beispiel die Geschwindigkeit beim Spawnen aus der Röhre zur Laufzeit anpassen. Dies macht den Eindruck, als könne der Wasserdruck oder Durchmesser des Rohres angepasst werden.

Durch die Verwendung von Spatial Hashing und der Sortierung nach Zellen lässt sich die Laufzeit der aufwändigen Suche der Nachbarpartikel von $O(n^2)$ auf $O(n)$ reduzieren. In Kombination mit der parallelen Ausführung des SPH-Algorithmus auf der GPU lassen sich erfolgreich große Anzahlen an Partikeln realistisch in Echtzeit simulieren.

7 Zusammenfassung

Das Projekt hat letztendlich die meisten gesetzten, sowie einige optionale Ziele erreicht. Ein realistisches Wasserverhalten durch SPH mit zusätzlicher Oberflächenspannung wurde erfolgreich umgesetzt. Die zunächst optionale GPU Implementierung und Spatial Hashing sorgten dafür, dass die gewünschte Menge an Partikeln mit stabiler Frameanzahl simuliert werden konnten. Abstriche mussten allerdings in Sachen Visualisierung gemacht werden, da in dem Projekt der Fokus hauptsächlich auf die physikalische Simulation gelegt wurde. Weitere Zusätze wie die Interaktion mit Objekten und die Komplexität des Wasserstroms wurden aus zeitlichen Gründen nicht implementiert, eigenen sich aber gut, um sie im Rahmen eines weiteren Projekts umzusetzen.

Literatur

Particle-based fluid simulation. `http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_ParticleFluids.pdf`. Accessed: 2021-02-24.

Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for sph fluids. *ACM Transactions on Graphics*, 32(6):1–8, 10 2013. doi: 10.1145/2508363.2508395.

Peter J Cossins. Smoothed particle hydrodynamics. *arXiv preprint arXiv:1007.1245*, pages 38–40, 07 2010.

Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

Leon B Lucy. A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024, 1977.

Matthias Müller, David Charypar, and Markus H Gross. Particle-based fluid simulation for interactive applications. In *Symposium on Computer animation*, pages 154–159, 2003a.

Matthias Teschner Bruno Heidelberger Matthias Müller, Danat Pomeranets, and Markus Gross. Optimized spatial hashing for collision detection of deformable objects. *Vision, Modeling, and Visualization 2003: Proceedings, November 19-21, 2003, München, Germany*, page 47, 2003b.