

Physikalisch-basierte Simulation in der Computergraphik

Schriftliche Ausarbeitung

Echtzeit SPH Wasserkanal

Peter Wichert, Kirill Menke, Linda Stadter



1 Einleitung

1.1 Hintergrund

1.2 Ziele

Das Projekt soll einen realistischen Wasserfluss in einer begrenzten 3D-Umgebung in Echtzeit simulieren. Eine finale Simulation soll das Wasser in Bewegung in einer Art Kanal/Fluss sehen. Zur Visualisierung sollen zunächst kugelförmige Objekte dienen und später Screen Space Rendering oder Marching Cubes benutzt werden. Für ausreichend detailliertes Verhalten der Simulation werden 30.000 Partikel simuliert. Als Kriterium für Echtzeitanforderungen sollen 30 Bilder pro Sekunde auf Hardware, die zum Zeitpunkt des Projekts in einem normalen PC zu finden sein kann, erreichbar sein, ohne die Anzahl der Partikel zu verringern.

1.3 Methoden

SPH Tait Surface

2 Theorie

2.1 Related Work

2.2 SPH

2.2.1 Oberflächenspannung

2.2.2 Smoothing Kernels

Bei der Geglätteten Teilchen-Hydrodynamik (SPH) wird die zu simulierende Flüssigkeit durch eine endliche Anzahl an Partikeln diskretisiert. Die Partikel interagieren innerhalb eines bestimmten Radius h miteinander und besitzen eigene physikalische Eigenschaften wie Dichte, Masse oder Druck.

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 durch Spatial Hashing, damit eine endliche Anzahl an Zellen für einen unendlich grossen Raum ausreicht. 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

Die Dichte wird dabei wie in der Gleichung 14 mithilfe eines Poly6 Smoothing Kernels W_{ij} aus Gleichung 15 berechnet.

$$\rho_i = \sum_j m_j W_{ij} \quad (1)$$

$$W_{ij} = \frac{315}{64\pi h^9} (h^2 - r^2)^3 \quad (2)$$

Der Druck p_i wird mit der Gaskonstanten K und dem Referenzdruck ρ_0 in Gleichung 3 berechnet.

$$p_i = K(\rho - \rho_0) \quad (3)$$

Die Force f aus Gleichung 16 kann als Addition der drei einzelnen Forces pressure, viscosity und external berechnet werden.

$$f = f^{pressure} + f^{viscosity} + f^{external} \quad (4)$$

$$W_{poly6}(\vec{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - |\vec{r}|^2)^3, & \text{falls } 0 \leq |\vec{r}| \leq h \\ 0, & \text{sonst} \end{cases} \quad (5)$$

$$\nabla W_{poly6}(\vec{r}, h) = \frac{945}{32\pi h^9} \begin{cases} \vec{r}(h^2 - |\vec{r}|^2)^2, & \text{falls } 0 \leq |\vec{r}| \leq h \\ 0, & \text{sonst} \end{cases} \quad (6)$$

$$\nabla W_{spiky}(\vec{r}, h) = -\frac{45}{\pi h^6} (h^2 - r^2)^3 \begin{cases} \frac{\vec{r}}{|\vec{r}|} (h - |r|)^2, & \text{falls } 0 \leq |\vec{r}| \leq h \\ 0, & \text{sonst} \end{cases} \quad (7)$$

$$\nabla^2 W_{viscosity}(\vec{r}, h) = \frac{45}{\pi h^9} \begin{cases} (h - |r|), & \text{falls } 0 \leq |\vec{r}| \leq h \\ 0, & \text{sonst} \end{cases} \quad (8)$$

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

$$\vec{f}_{i \leftarrow j}^{surface} = \frac{2\rho_0}{\rho_i + \rho_j} (\vec{f}_{i \leftarrow j}^{cohesion} + \vec{f}_{i \leftarrow j}^{curvature}) \quad (10)$$

$$\vec{f}_{i \leftarrow j}^{curvature} = -\gamma m_i (\vec{n}_i - \vec{n}_j) \quad (11)$$

$$\vec{n}_i = h \sum_j \frac{m_j}{\rho_j} \nabla W(\vec{r}_i - \vec{r}_j, h) \quad (12)$$

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

$$\rho_i = \sum_j m_j W_{ij} \quad (14)$$

$$W_{ij} = \frac{315}{64\pi h^9} (h^2 - r^2)^3 \quad (15)$$

$$\vec{f} = \vec{f}^{pressure} + \vec{f}^{viscosity} + \vec{f}^{external} \quad (16)$$

$$\vec{f} = \vec{f}^{pressure} + \vec{f}^{viscosity} + \vec{f}^{external} + \vec{f}^{surface} \quad (17)$$

$$\vec{f}_i^{pressure} = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W(\vec{r}_i - \vec{r}_j, h) \quad (18)$$

$$\vec{f}_i^{external} = \vec{g} \quad (19)$$

$$\vec{f}_i^{viscosity} = \mu \sum_j m_j \frac{\vec{v}_i - \vec{v}_j}{\rho_j} \nabla^2 W(\vec{r}_i - \vec{r}_j, h) \quad (20)$$

2.3 Integration

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 ueberschreiten. Ausserdem werden die Geschwindigkeiten der entsprechenden Richtungen umgekehrt. Eine Daempfungvariable wird eingefuehrt, die die Geschwindigkeit dabei zusaetzlich abmildern kann.

3 Implementierung

3.1 Übersicht

3.2 Unity

3.3 Compute Shader

3.4 Rendering

3.5 Performance

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.

Die Positionen in x-,y- und z-Richtung werden gehasht, um daraus den Zellindex zu bestimmen. Um die Anzahl der Hash-Kollisionen möglichst gering zu halten, werden 8-stellige Primzahlen 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. Dieser Sortieralgorithmus kann parallel auf der GPU ausgeführt werden.

Das Einbeziehen der Nachbarpartikel bei der Berechnung der Dichte und im Anschluss der Force wird im Dichte- und im Force-Shader auf ähnliche Weise implementiert. Hierbei muss über die 27 möglichen Nachbarzellen iteriert werden. 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 Integrations-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 Blautönen 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 Laufzeit erzeugt wird. Die Box wird dabei durch ein eigenes Skript kontrolliert, welches die Positionen der Seitenplatten anhand einer vorgegebenen Bodenplatte berechnet. Ausserdem werden zwei Oberflächen 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

Bild von Test mit Boxspawn Bild von Explosion 2 Bilder von Finale mit Rohr (1 leer 1 voll)
+ ein paar simulations zahlen

5 Beiträge

Verteilung der Aufgaben erfolgte meisten spontan nachdem in einem Meeting die nächsten Schritte besprochen wurden.

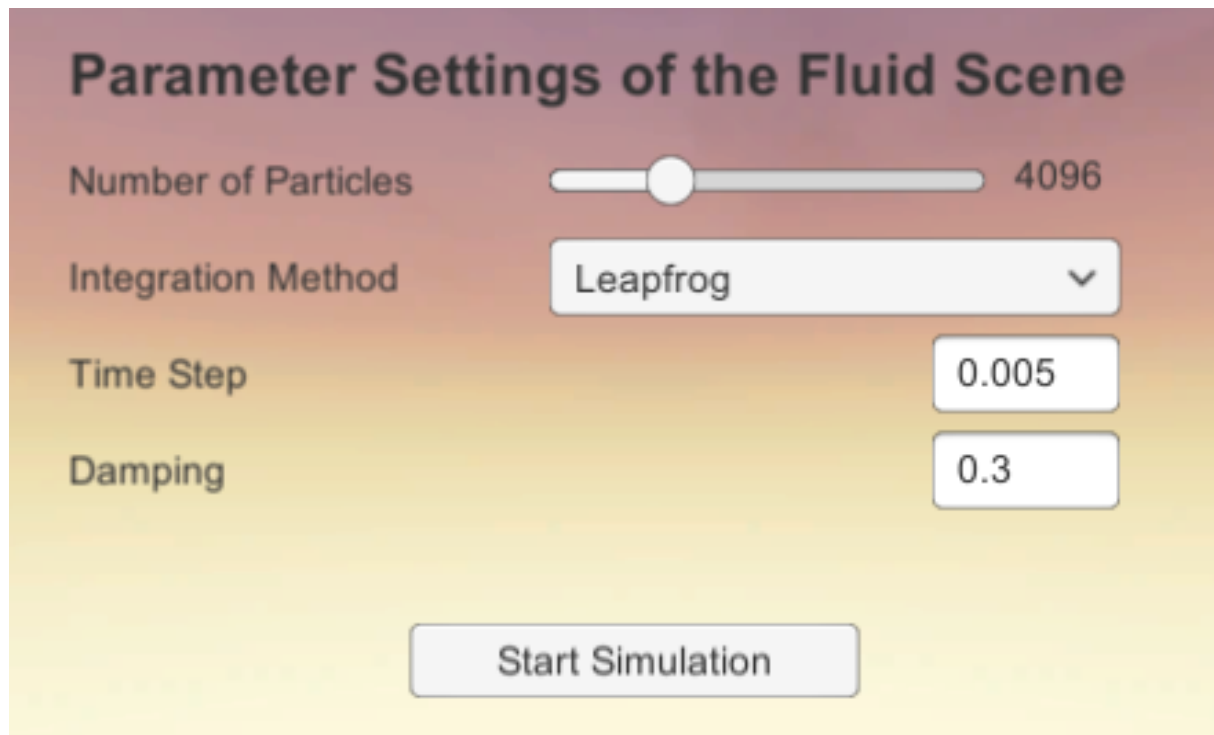


Fig. 1: Oberfläche vor dem Starten der Simulation, um die entsprechenden Parameter der Simulation einzustellen.

| Peter Wichert | Kirill Menke | Linda Stadter |
|--|--|---|
| <ul style="list-style-type: none"> • Erstellen der Projektplan Präsentation • Erstellen der Milestone Präsentation • Erstellen und Vortragen der finalen Präsentation • Erstellen des finalen Berichts • Recherche für eigene und gruppenrelevante Themen • Leapfrog Integration • Surface Tension • Particle Spawn • Parameter Berechnung • Ergänzen des Debug Modus • Debugging eigener und anderer Implementierungen | <ul style="list-style-type: none"> • Erstellen und Vortragen der Projektplan Präsentation • Erstellen und Vortragen der Milestone Präsentation • Erstellen der finalen Präsentation • Recherche für eigene und gruppenrelevante Themen • C# und Compute Shader Grundgerüst • Unity Grundstruktur • Spatial Hashing und Nearest Neighbour Search • Rendering • Implementierung des Debug Modus • Stabilitäts Tests • Debugging eigener und anderer Implementierungen | <ul style="list-style-type: none"> • Erstellen der Projektplan Präsentation • Erstellen der Milestone Präsentation • Erstellen der finalen Präsentation • Erstellen des finalen Berichts • Recherche für eigene und gruppenrelevante Themen • Unity Grundstruktur • Spatial Hashing und Nearest Neighbour Search • Simulationsumgebung und Kollisionen • Finaler kontinuierlicher Partikel Spawn • Interface • Unity Szenen • Farbcodierung der Partikel • Debugging eigener und anderer Implementierungen |

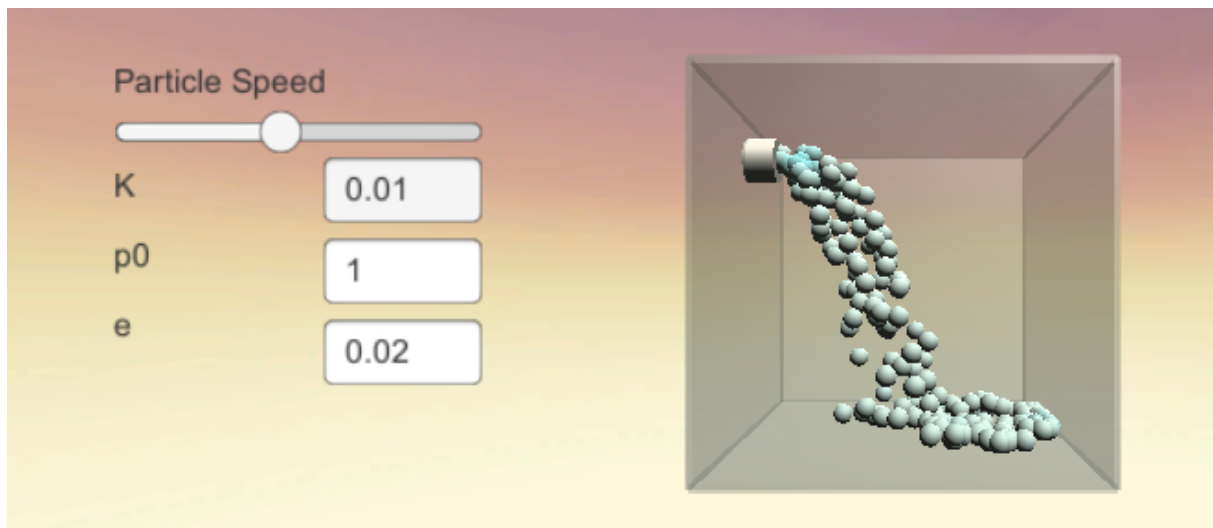


Fig. 2: Oberfläche während der Simulation, um die Parameter des SPH-Algorithmus anzupassen.

6 Diskussion

Durch das Verwenden 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 grosse Anzahlen an Partikeln in Echtzeit simulieren.

Merge Sort kann nur 2-er Potenzen von Partikel. Echtzeitperformance würde etwas schlechter werden mit dem Extra Render Schritten. Echtzeitperformance würde vermutlich ein Stück schlechter werden mit Festkörper Interaktion.

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

7 Zusammenfassung

Das Projekt hat letztendlich die meisten gesetzten, sowie einige optionale, Ziele erreicht. Realistisches Wasserverhalten durch SPH mit zusätzlicher Oberflächenspannung//TODO ä ist gegeben. 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, Interaktion mit Objekten und Komplexität des Wasserstroms gemacht werden.

Literatur

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