Physikalisch-basierte Simulation in der Computergraphik

Schriftliche Ausarbeitung

Echtzeit SPH Wasserkanal

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

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 simulierten. 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

$$\vec{f} = \vec{f}^{pressure} + \vec{f}^{viscosity} + \vec{f}^{external} \tag{1}$$

$$\rho_i = \sum_i m_j W_{ij} \tag{2}$$

$$p_i = K(\rho - \rho_0) \tag{3}$$

$$\vec{f}_i^{\text{pressure}} = -\sum_j m_j (\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) \bigtriangledown W(\vec{r}_i - \vec{r}_j, h) \tag{4}$$

$$\vec{f}_i^{\text{external}} = \vec{g} \tag{5}$$

$$\vec{f}_i^{\text{viscosity}} = \mu \sum_j m_j \frac{\vec{v}_i - \vec{v}_j}{\rho_j} \bigtriangledown^2 W(\vec{r}_i - \vec{r}_j, h) \tag{6}$$

2.2.1 Smoothing Kernels

$$W_{poly6}(\vec{r},h) = \frac{315}{64\pi h^9}\begin{cases}(h^2-|\vec{r}|^2)^3, & \text{falls } 0 \le |\vec{r}| \le h\\ 0, & \text{sonst}\end{cases} \tag{7}$$

$$\bigtriangledown W_{poly6}(\vec{r},h) = \frac{945}{32\pi h^9}\begin{cases}\vec{r}(h^2-|\vec{r}|^2)^2, & \text{falls } 0 \le |\vec{r}| \le h\\ 0, & \text{sonst}\end{cases} \tag{8}$$

$$\bigtriangledown W_{spiky}(\vec{r},h) = -\frac{45}{\pi h^6}(h^2-r^2)^3\begin{cases}\frac{\vec{r}}{|\vec{r}|}(h-|r|)^2, & \text{falls } 0 \le |\vec{r}| \le h\\ 0, & \text{sonst}\end{cases} \tag{9}$$

$$\bigtriangledown^2 W_{viscosity}(\vec{r},h) = \frac{45}{\pi h^9}\begin{cases}(h-|r|), & \text{falls0} \le |\vec{r}| \le h\\ 0, & \text{sonst}\end{cases} \tag{10}$$

2.2.2 Oberflächenspannung

Um akkurates Wasserverhalten in Wasserspritzern und an Grenzen zu Luft zu erzeugen wird zusätzlich die molekulare Attraktion und Repulsion zwischen Wassermolekülen modelliert.

Eine neue Kraft für Oberflächenspannung wird wie folgt in die bisherige Kraftberechnung eingefügt:

$$\vec{f} = \vec{f}^{pressure} + \vec{f}^{viscosity} + \vec{f}^{external} + \vec{f}^{surface} \tag{11}$$

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

Die Spline-Funktion (12) sorgt ab einem gewählten Radius (h) für die zunächst anziehende Wirkung, sowie den ansteigenden abstoßenden Effekt, sollten zwei Partikel zu nahe beieinander sein.

$$C(r) = \frac{32}{64\pi h^9}\begin{cases}(h-r)^3r^3, & \text{falls } 2r > h \wedge r \le h\\ 2(h-r)^3r^3, & \text{falls } r0 \wedge 2r \le h\\ 0, & \text{sonst}\end{cases} \tag{12}$$

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

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 (15) hinzu. Diese sorgt dafür, dass Partikel-Mengen unabhängig von Ausgangsposition versuchen eine Kugelform einzunehmen und dadurch die Oberfläche ihres Tropfens minimieren.

Zunächst wird für jedes Partikel ein Normal-Vektor berechnet (14). 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 ein geglättetes Feld optimale Ergebnisse erreichen.

$$\vec{n}_i = h\sum_j \frac{m_j}{\rho_j} \bigtriangledown W(\vec{r}_i - \vec{r}_j, h) \tag{14}$$

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

Um den Zusammenhalt von wenigen Partikel an den von Gruppen mit mehreren Anzugleichen, wird des weiteren ein Korrekturfaktor beim Zusammensetzen Oberflächenspannung eingeführt(16).

$$\vec{f}_{i\leftarrow j}^{\text{surface}} = \frac{2\rho_0}{\rho_i + \rho_j}(\vec{f}_{i\leftarrow j}^{\text{cohesion}} + \vec{f}_{i\leftarrow j}^{\text{curvature}}) \tag{16}$$

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 Daempfungsvariable 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 durchgefuehrt. Der eigentliche SPH-Algorithmus wird auf sieben Shader aufgeteilt, welche von der GPU ausgefuehrt werden. Es ist je ein Shader dafuer zustaendig, 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 durchzufuehren.

Die Positionen in x-,y- und z-Richtung werden gehasht, um daraus den Zellindex zu bestimmen. Um die Anzahl der Hash-Kollisionen moeglichst gering zu halten, werden 8-stellige Primzahlen und eine grosse Anzahl an Partikeln - und damit auch Zellen - benutzt.

Fuer das Sortieren der Partikel anhand ihrer zugewiesenen Zelle wird eine fertige Implementierung des Algorithmus Bitonisches Sortieren verwendet. Dieser Sortieralgorithmus kann parallel auf der GPU ausgefuehrt werden.

Das Einbeziehen der Nachbarpartikel bei der Berechnung der Dichte und im Anschluss der Force wird im Dichte- und im Force-Shader auf aehnliche Weise implementiert. Hierbei muss ueber die 27 moeglichen Nachbarzellen iteriert werden. Mithilfe des zuvor gespeicherten Abstands kann dann effizient auf alle Partikel einer bestimmten Nachbarzelle zugegriffen werden. Zusaetzlich wird ueberprueft, 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 unnoetigen Overhead, eigenstaendige GameObjects pro Partikel zu erstellen. In diesem Schritt werden die Partikel anhand ihrer Dichte in einen blassen bis kraeftigen Blauton gefaerbt.

Neben dem GameObject, welches die Simulation behandelt, werden auch weitere Objekte benoetigt. Der Raum, in welchem sich die Partikel bewegen koennen, 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 Oberflaechen benoetigt, um Parameter einerseits vor dem Beginn der Simulation wie in Bild 1 und andererseits waehrend der laufenden Simulation wie in Bild 2 einstellen und anpassen zu koennen. Dazu werden fertige UI-Elemente von Unity verwendet. Beim Starten der Simulation findet ein Szenenwechsel statt. Dadurch werden nicht mehr benoetigte Objekte entfernt und neue Objekte eingeblendet.

4 Ergebnisse und Evaluierung

Bild von Test mit Boxspawn Bild von Explosion 2 Bilder von Finalem 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.

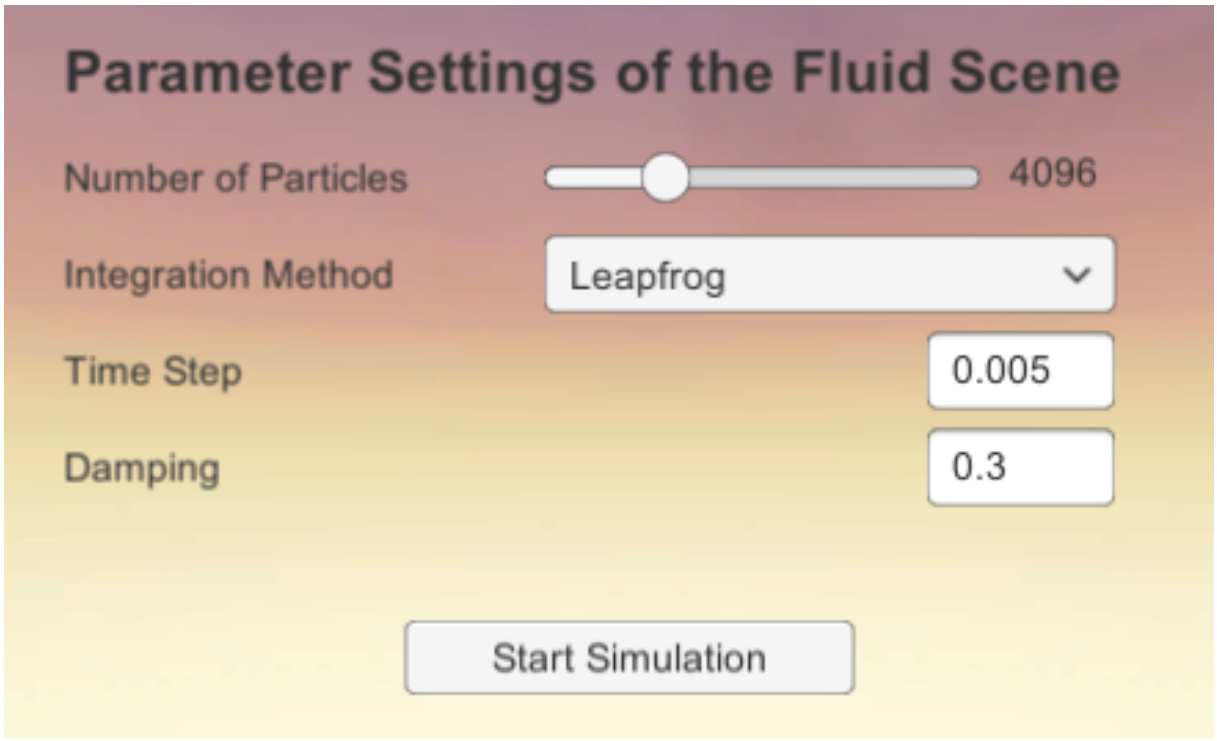

Fig. 1: Oeberflaeche vor dem Starten der Simulation, um die entsprechenden Parameter der Simulation einzustellen.

Peter Wichert	Kirill Menke	Linda Stadter
• 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	• 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	• 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

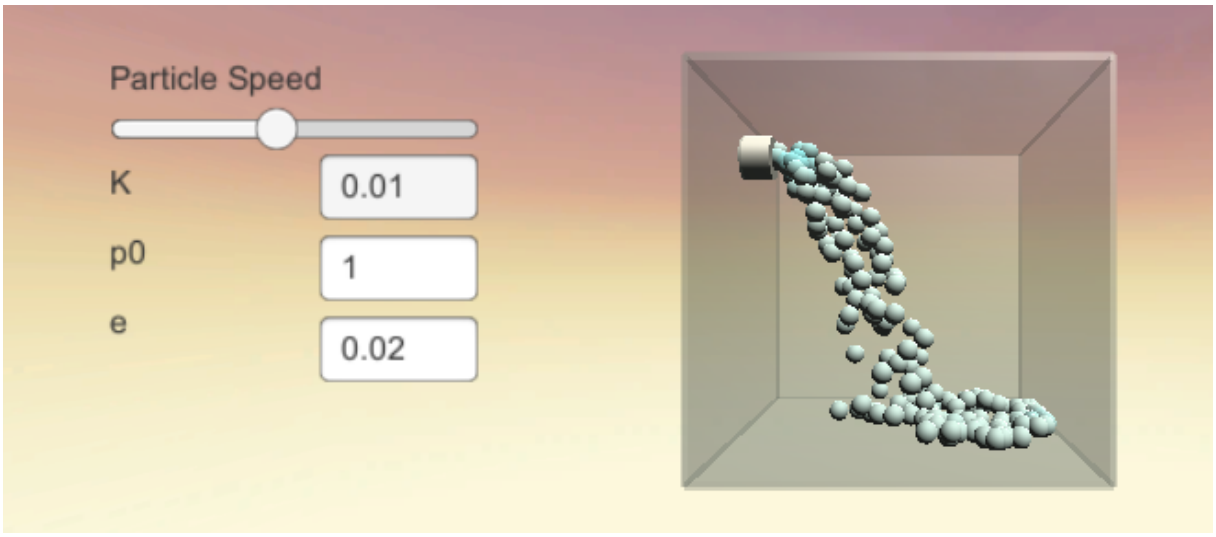

Fig. 2: Oeberflaeche waehrend der Simulation, um die Parameter des SPH-Algorithmus anzupassen.

6 Diskussion

Durch das Verwendung von Spatial Hashing und der Sortierung nach Zellen laesst sich die Laufzeit der aufwaendigen Suche der Nachbarpartikel von $O(n^2)$ auf $O(n)$ reduzieren. In Kombination mit der parallelen Ausfuehrung 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.