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

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

2.2.1 Oberflächenspannung

2.2.2 Smoothing Kernels

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

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

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

$$\nabla^2 W_{viscosity}(\vec{r},h) = \frac{45}{\pi h^9}\begin{cases}(h-|r|), & \text{falls0} \le |\vec{r}| \le h\\ 0, & \text{sonst}\end{cases} \tag{4}$$

$$C(r) = \frac{32}{64\pi h^9}\begin{cases}(h-r)^3 r^3, & \text{falls } 2r > h \wedge r \le h\\ 2(h-r)^3 r^3, & \text{falls } r0 \wedge 2r \le h\\ 0, & \text{sonst}\end{cases} \tag{5}$$

$$\vec{f}^{\text{surface}}_{i\leftarrow j} = \frac{2\rho_0}{\rho_i+\rho_j}(\vec{f}^{\text{cohesion}}_{i\leftarrow j} + \vec{f}^{\text{curvature}}_{i\leftarrow j}) \tag{6}$$

$$\vec{f}_{i\leftarrow j}^{\text{curvature}} = -\gamma m_i(\vec{n}_i - \vec{n}_j) \tag{7}$$

$$\vec{n}_i = h\sum_j \frac{m_j}{\rho_j} \bigtriangledown W(\vec{r}_i - \vec{r}_j, h) \tag{8}$$

$$\vec{f}_{i\leftarrow j}^{\text{cohesion}} = \gamma m_i m_j C(|\vec{x}_i - \vec{x}_j|)\frac{\vec{x}_i - \vec{x}_j}{|\vec{x}_i - \vec{x}_j|} \tag{9}$$

2.3 Integration

Die Suche der Nachbarpartikel stellt einen sehr aufwaendigen Schritt des SPH-Algorithmus dar. Die Nachbarpartikel eines Partikels sind die Menge der Partikel, welche sich innerhalb eines bestimmten Radius h befinden und somit Einfluss auf den Partikel besitzen. Um diese effizient zu finden, teilen wir den Raum in ein uniformes Gitter auf. Eine Zelle besitzt die Groesse des Radius h, sodass die Nachbarpartikel nur in den anliegenden 26 Zellen und in der Zelle des Partikels gesucht werden muessen. Durch ihre Positionen im Raum werden die Partikel je einer Zelle zugewiesen. Dies erfolgt durch Spatial Hashing, damit eine endliche Anzahl an Zellen fuer einen unendlich grossen Raum ausreicht. Dann koennen die Partikel anhand ihrer zugewiesenen Zelle sortiert werden. Fuer jede Zelle wird der Abstand zu dem ersten Partikel der Zelle gespeichert. Diese Idee wurde 2008 von Nvidia vorgestellt. **??**

Die Dichte wird dabei wie in der Gleichung 10 mithilfe eines Poly6 Smoothing Kernels W_{ij} aus Gleichung 11 berechnet.

$$\rho_i = \sum_i m_j W_{ij} \tag{10}$$

$$W_{ij} = \frac{315}{64\pi h^9}(h^2 - r^2)^3 \tag{11}$$

Der Druck p_i wird mit der Gaskonstanten K und dem Referenzdruck ρ_0 in Gleichung 12 berechnet.

$$p_i = k(\rho - \rho_0) \tag{12}$$

$$p_i = k((\frac{\rho}{\rho_0})^\gamma - 1) \tag{13}$$

Die Force f aus Gleichung 14 kann als Addition der drei einzelnen Forces pressure, viscosity und external berechnet werden.

$$\vec{f} = \vec{f}^{pressure} + \vec{f}^{viscosity} + \vec{f}^{external} \tag{14}$$

$$\vec{f} = \vec{f}^{pressure} + \vec{f}^{viscosity} + \vec{f}^{external} + \vec{f}^{surface} \tag{15}$$

$$\vec{f}_i^{\text{pressure}} = -\sum_j m_j(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) \bigtriangledown W(\vec{r}_i - \vec{r}_j, h) \tag{16}$$

$$\vec{f}_i^{\text{external}} = \vec{g} \tag{17}$$

$$\vec{f}_i^{\text{viscosity}} = \mu\sum_j m_j \frac{\vec{v}_i - \vec{v}_j}{\rho_j} \bigtriangledown^2 W(\vec{r}_i - \vec{r}_j, h) \tag{18}$$

3 Implementierung

3.1 Übersicht

3.2 Unity

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

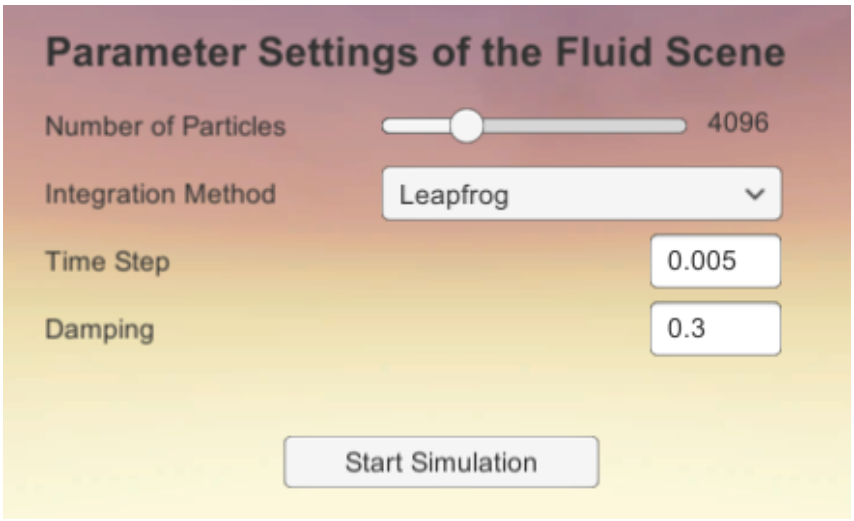

Fig. 1: Oeberflaeche vor dem Starten der Simulation, um die entsprechenden Parameter der Simulation einzustellen.

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.

3.3 Spatial Hashing

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.

3.4 Compute Shader

3.5 Rendering

3.6 Performance

Das Einbeziehen der Nachbarpartikel bei der Berechnung der Dichte und im Anschluss der Force wird auf aehnliche Weise implementiert. Hierbei muss nur ueber die 27 moeglichen Nachbarzellen iteriert werden. Mithilfe des zuvor gespeicherten Abstands kann dann effizient auf alle Partikel einer bestimmten Nachbarzelle zugegriffen werden. Zusaetlich muss noch ueberprueft werden, ob der Abstand zu dem Partikel kleiner als der Radius h ist.

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 benoetigte 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

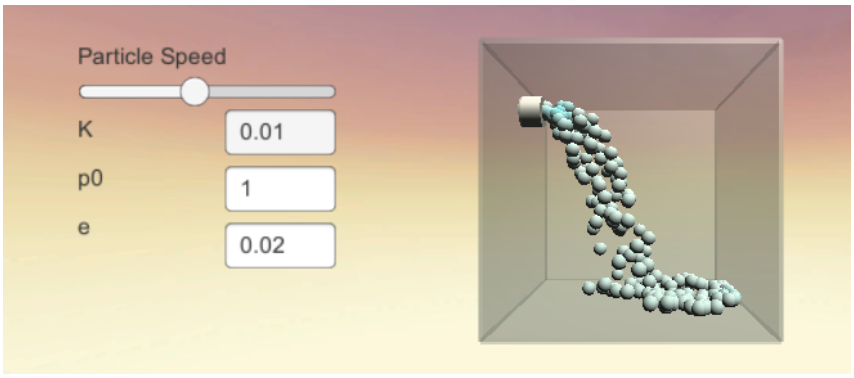

Fig. 2: Oeberflaeche waehrend der Simulation, um die Parameter des SPH-Algorithmus anzupassen.

5 Diskussion

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.

6 Beiträge

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

Auflistung was wer so gemacht hat bei den ??? bin ich nicht sicher fügt hinzu was fehlt (ist eigentlich relativ gemischt idk ob wir das überhaupt auflisten wollen)

Kirill hält ersten 2 Präsentationen Peter hält die letzte Präsentation Erstellen der Präsentationen größtenteils vom Vortragenden mit Hilfe der anderen. Alle 3 Recherche Linda und Kirill spatial hashing Peter autom. Parameterberechnung Peter Leapfrog ??? Kirill Debug Mode (später Peter kleine änderungen für neuen Code/Debug) Linda Interface und Unity Szenen Linda und Peter Report

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