

# Physikalisch-basierte Simulation in der Computergraphik

Schriftliche Ausarbeitung

Titel des Projektes

Liste der Studierenden



## 1 Einleitung

Detaillierte Instruktionen sind in der Englischen Vorlage zu finden.

## 2 Theorie

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 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 1 mithilfe eines Poly6 Smoothing Kernels  $W_{ij}$  aus Gleichung 2 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  $\rho_0$  in Gleichung 3 berechnet.

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

Die Force  $f$  aus Gleichung 4 kann als Addition der drei einzelnen Forces pressure, viscosity und external berechnet werden.

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

TODO Force

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 durch die Begrenzungen ueberschreiten. Ausserdem wird ihre Geschwindigkeit in der entsprechenden Richtung umgekehrt. Eine Daempfungvariable wird eingefuehrt, die die Geschwindigkeit dabei abmildern kann.

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

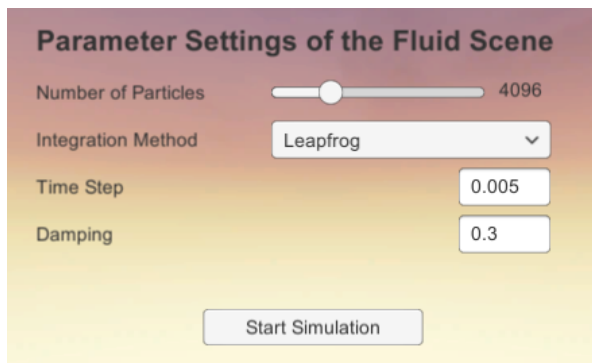


Fig. 1: Oberfläche vor dem Starten der Simulation, um die entsprechenden Parameter der Simulation einzustellen.

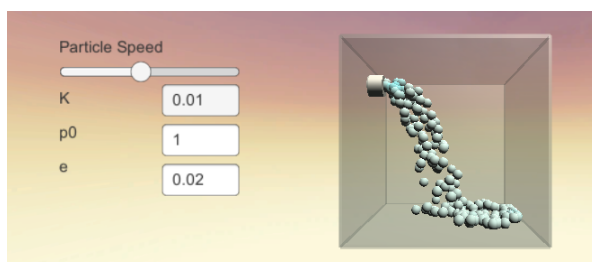


Fig. 2: Oberfläche während der Simulation, um die Parameter des SPH-Algorithmus anzupassen.

Das Einbeziehen der Nachbarpartikel bei der Berechnung der Dichte und im Anschluss der Force wird 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.

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

## 5 Beiträge

## 6 Diskussion

## 7 Zusammenfassung

## Literatur