A GPU-Based Particle Engine

Seminar Computergrafik: Procedural Content Generation

Sommersemester 2010

Sebastian Rockel
8rockel@informatik.uni-hamburg.de

Universität Hamburg

Zusammenfassung Dieser Bericht gibt eine Einführung in das Thema GPU-basierte Partikelsysteme. Es wird schrittweise versucht von allgemeinen Partikelsystemgrundlagen und Beispielen die Entwicklung hin zu einem komplett auf der GPU ausgeführten System nachzuvollziehen. Dazu wurde die Recherche hauptsächlich auf Basis des Artikels von [KSW04] und dem darin vorgestellten Partikelsystem *UberFlow* gemacht. Darüber hinaus werden Vergleiche mit anderen aktuellen GPU-basierten Systemen angeregt, sowie Vor- und Nachteile des vorgestellten Systems in einer abschließenden Betrachtung diskutiert.

1 Einführung

1.1 Partikelsystem

Partikelsysteme stellen Funktionen zur Verfügung, die eine große Anzahl von Objekten animieren können. Damit ist es möglich, zum Beispiel im Bereich der Computergrafik, physikalische Vorgänge (wie Rauch) darzustellen. Dies wäre mit konventionellen Methoden sehr komplex. Mit einem Partikelsystem lässt sich der Gesamtprozess aber aufgrund eines relativ einfachen Bewegungsmodells von einzelnen Objekten, eben den Partikeln, modellieren. Abbildung 1 zeigt eine Rauchdarstellung, die auf dem Bewegungsmodell einzelner Partikel beruht.

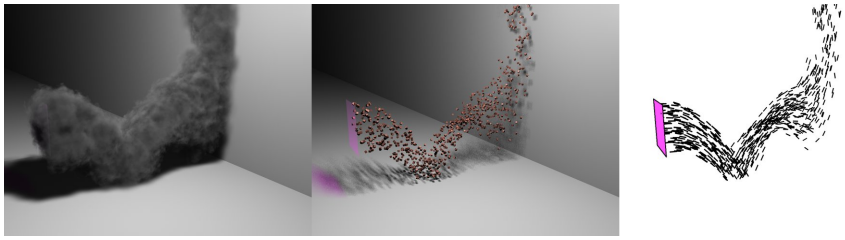

Abbildung 1. Rauch dargestellt mit Partikeln [Wikipedia]

1.2 Details

Das Bewegungsmodell von einzelnen Partikeln ergibt ein Modell für das gesamte zusammengesetzte Objekt. Darauf basierend eignet sich ein solches System für so genannte volumetrische Effekte. Gerade Feuer, Explosionen, Rauch und Flüssigkeiten werden in Animationen und auch Computerspielen zunehmend gefordert. Dabei kann das sich darunter befindliche physikalische Modell mit relativ einfachen Gesetzmäßigkeiten begnügen.

Für anspruchsvollere Dynamik-Simulationen können an ein Partikelsystem noch weitere Anforderungen gestellt werden. Zum Einen sollen eventuell auch komplexere physikalische Modelle simuliert werden (z.B. Fliehkraft oder sogar mehrere Gravitationszentren) und zum Anderen sollen Partikel auch unter einander, und nicht nur mit ihrer Umgebung, interagieren.

Ein Beispiel eines einfachen aber optisch sehr überzeugenden Partikelsystems ist ein in POV-Ray Implementiertes [Joh02]. Abbildungen 2 und 3 zeigen Animationsausschnitte fast fotorealistischer Effekte. In diesem System gibt es keine Partikel-zu-Partikel-Interaktion. Natürlich hat es auch den Nachteil fehlender Interaktion (keine Echtzeit) aufgrund der Implementierung als Ray-Tracer.

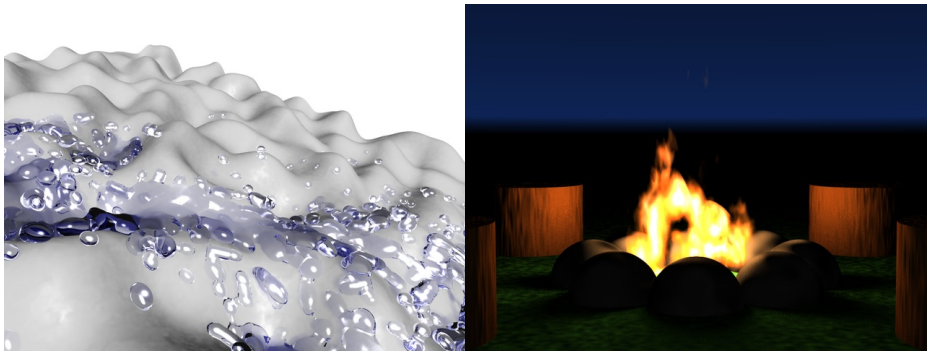

Abbildung 2. Wasserfluss und Feuer [Joh02]

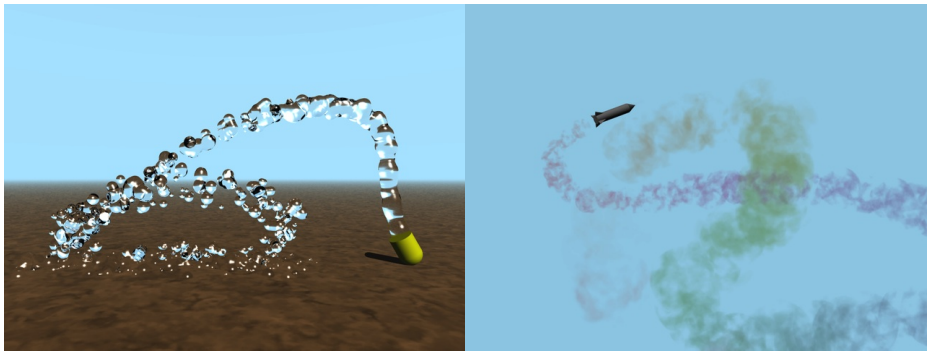

Abbildung 3. Wasserfontäne und Rauch [Joh02]

Partikelparameter Die Bewegung der Gesamtheit aller Partikel hängt von der Bewegung eines Einzelnen ab. Dazu sind eine Reihe von (einfachen) Partikelparametern zu definieren.

Emission Die „Lebensphase“ eines Partikels beginnt mit seiner Platzierung am Emitter. Dabei ist eine zunächst konstante Anfangsgeschwindigkeit festgelegt. Dieser wirkt in der Regel eine Dämpfung entgegen um die Partikelbewegung über die Zeit zu verringern. Eine Lebensdauer legt das Zeitfenster fest, in der sich ein Partikel in der Animation befindet. Sie wird meist in *Clockcycles* angegeben. Um die Animation in gewissen Grenzen zu skalieren, ist darüber hinaus eine maximale Anzahl von Partikeln festgelegt.

Bewegung Das Bewegungsmodell selber hängt von der gewünschten Anwendung des Partikelsystems ab und kann nahezu beliebige (auch dynamische) Systeme implementieren. In [KSW04] wird zum Beispiel das Euler-Verfahren zur numerischen Integration des newtonschen Grundgesetzes der Dynamik verwendet (siehe Gleichung 1 und 2).

$$v(r, t + dt) = v(r, t) + v_{ext}(r, t) + \frac{F}{m} dt \tag{1}$$

$$r(t + dt) = r(t) + \frac{1}{2}\Big(v(r, t) + v(r, t + dt)\Big) dt \tag{2}$$

Diese zwei Gleichungen verwenden die aktuelle Geschwindigkeit $v(r, t)$ und Position $r(t)$ abhängig von der Zeit t um die neuen Partikelparameter zu berechnen. v_{ext} ist ein externes Windfeld, F ist eine externe Kraft (z.B. Gravitation) und m ist die Partikelmasse. Externe Kräfte werden auch in Abschnitt *Externe Kräfte* auf dieser Seite erklärt. dt bezeichnet das Zeitintervall für eine Berechnung (der Partikelauslenkung).

Kollision Die Kollision eines Partikels wird in zweierlei Hinsicht getrennt betrachtet, und zwar in Abhängigkeit mit der Umgebung und mit anderen Partikeln. Manche Partikelsysteme verzichten komplett auf komplizierte Partikel-zu-Partikel-Interaktionen (siehe Runes Partikelsystem auf Seite 2).

Kollisionsreaktion Diese legt fest wie ein Partikel nach der Kollisionserkennung mit einem Objekt reagieren soll. Beispielsweise soll ein Wasser- anders reagieren als ein Rauch-Partikel oder ein festes Objekt.

Externe Kräfte Je nach Anwendung sind externe Kräfte wie Gravitation oder Wind notwendig. Dabei ist jede Art von (berechenbarem) Kräftefeld (oder auch mehrere sich beeinflussende) möglich. Die Rechenzeit für die Partikel-Dynamik setzt hier eine praktische Grenze.

1.3 GPU-Partikelsysteme

Für Simulationen oder Computerspiele kommt es in der Regel auf die Echtzeitfähigkeit an. Regen oder Rauch muss sich realistisch in die Animationssequenz

einfügen. Einfache Physikmodelle halten die Berechnungskosten im Rahmen. Doch um volumetrische Effekte einigermaßen sinnvoll einzusetzten bedarf es in der Regel einer sehr großen Anzahl von Partikeln von einigen 1000 bis dem Tausendfachen, je nach Anwendung. Solche Massen an Berechnungen können von der CPU nicht mehr schnell genug berechnet werden (und nebenbei müssen noch andere Funktionen bearbeitet werden).

Grafikarten (GPUs) bieten heute durch ihre massive Anzahl von parallelen Recheneinheiten die Möglichkeit viele Berechnungen gleichzeitig auszuführen. Es folgt eine kurze Liste, mit Überblickscharakter, möglicher (populärer) GPU-Schnittstellen:

- CUDA (NVIDIA)
- OpenGL
- OpenCL
- DirectX (Microsoft Windows)

Neben proprietären GPU-Schnittstellen wie CUDA und DirectX, sind OpenGL und zunehmend auch OpenCL als hardware- und betriebssystemunabhängige Standards verbreitet.

NVIDIA bietet mit dem CUDA-Software-Development-Kit [NVI10] eine Reihe von Demo-Applikationen an, die auch einige Partikelsysteme mit GPU-Berechnung umfassen (Abbildung 4). Darunter ist eine Art zweidimensionaler Wassersimulation *fluidsGL*, eine Galaxie-Simulation mit komplexerem Gravitationsmodell *nbody* und eine allgemeine Partikelsimulation *particles*. Letztere lässt die Partikelparameter (siehe Abschnitt 1.2) frei einstellen und (interaktiv) deren Einfluss erkennen.

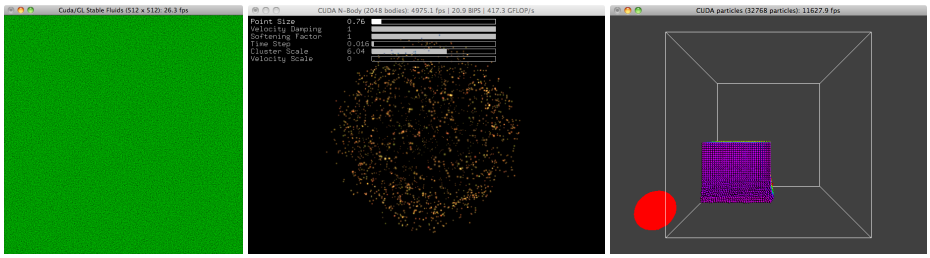

Abbildung 4. Von links nach rechts: 2D-Wassersimulation, 3D-Galaxysimulation und 3D-Partikelsimulation [NVI10]

Als Entwicklungsschritt hin zu einem auf der GPU berechneten Partikelsystem, hat dieses Demo die Animation relative vieler Partikel (> 30.000) zu bieten. Im Gegensatz zu *Runes Partikelsystem* (auf Seite 2) ist es darüber hinaus echtzeitfähig. Das Rendern der Partikel geschieht auf der GPU, wenn auch die Partikel-(Positions-) Berechnung auf der CPU stattfindet. Bedarf es einer größeren Anzahl an Partikeln (> 100.000) müssen sehr viele Partikeldaten über den Bus von CPU zu GPU transportiert werden. Die Durchsatzrate bei der Animation ist dann busbegrenzt.

Die Anforderung an ein Partikelsystem mit einer solchen Anzahl von Partikeln, ist die Berechnung auch der Partikelpositionen, und somit der kompletten Animation, auf der GPU, um (fast) jeglichen Bus-Transfer zwischen CPU und GPU zu vermeiden.

2 UberFlow-Partikelsystem

2.1 Überblick

Das UberFlow-Partikelsystem [KSW04] wurde mit dem Ziel entwickelt eine sehr große Anzahl von Partikeln ($>$100.000) in Echtzeit zu berechnen. Die Realisierung erfolgte in einer reinen GPU-Implementierung von allen daran beteiligten Algorithmen, um jegliche Bus-Belastung durch Daten-Transfers zu vermeiden.

Als GPU-Schnittstelle wurde OpenGL verwendet. In dessen aktuellen Standard bietet es das so genannte *Pixelshader 3.0* -Modell an. Dabei werden den Shadern (parallele GPU-Programme) Speicherobjekte zu beliebigem Einsatz (also nicht nur zum Rendern) angeboten. Das ermöglicht eine flexible Programmierung nach dem Vorbild von CPU-(Host)-Programmen.

Priorität liegt auf der korrekten grafischen Darstellung der Partikel und der Kollisionserkennung. Die Berechnungen sind optimiert für Darstellungen mit Heightmaps.

2.2 Anwendung

Die Kombination aus dem verwendeten Bewegungsmodell und der riesigen Anzahl an darstellbaren Partikeln eignet sich sehr gut für Strömungssimulationen, wie in Abbildungen 5 und 6 auf der nächsten Seite gezeigt. Dabei können sämtliche Simulationsparameter in Echtzeit geändert und deren Einfluss gezeigt werden. Unterschiedliche Darstellungen der Partikel sind möglich (Abbildungen 5 und 6).

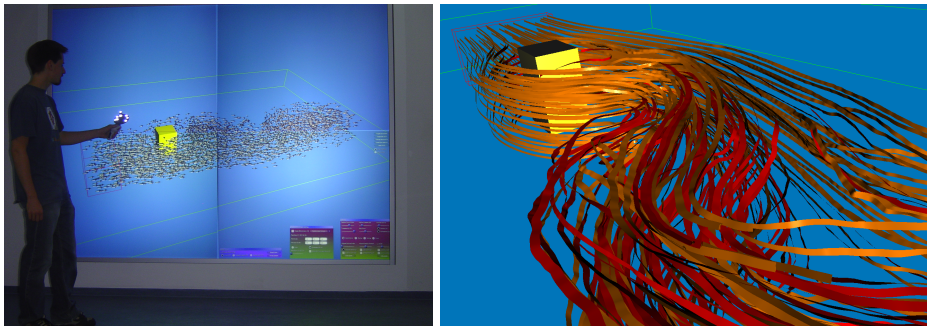

Abbildung 5. Strömungssimulation, rechts Darstellung der Partikel als *Ribbons* [KSW04]

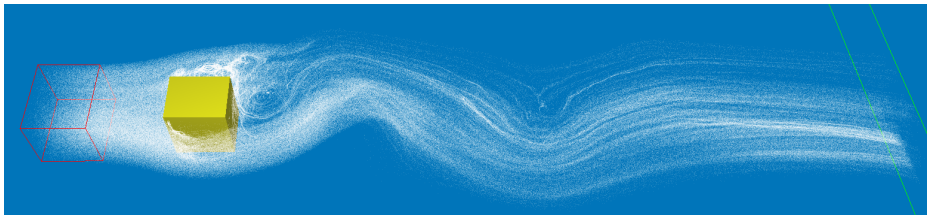

Abbildung 6. Strömungssimulation 2: Partikeldarstellung mit einzelnen Punkten [KSW04]

2.3 Alles mit Texturen

Um das Potential von modernen GPUs auszunutzen reicht es nicht aus, existierende CPU-Algorithmen zu portieren. Die Shadereinheiten auf GPUs sind optimiert um Daten zu verarbeiten, die als Textur vorliegen.

Im Uberflow-Partikelsystem werden alle Partikelpositionen in einer zweidimensionalen RGBA-Textur gespeichert. Die Anfangsposition $r(t)$ kann dabei zufällig oder geordnet festgelegt werden. Jede Koordinate der n x n-Dimensions-Textur enthält einen vierdimensionalen Vektor um die räumlichen Koordinaten und den Zeitstempel eines Partikels zu speichern (Gleichung 3). Der Zeitstempel wird während der Animation benutzt um zu überprüfen, ob das Partikel schon sichtbar sein soll bzw. seine Lebensdauer schon überschritten hat.

$$\begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix}^{n\,x\,n} = r(t)^{n\,x\,n} = \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix}^{n\,x\,n} \tag{3}$$

Das in Abschnitt 2.1 erwähnte Pixelshader 3.0-Modell ermöglicht es Speicherbereiche (so genannte *Memory Objects, MO*) für beliebige Daten anzufordern. Shaderprogramme können so Eingabedaten als MO bekommen um neue Daten in einem MO zu speichern, dass wieder als Eingabe für ein anderes Programm benutzt wird. Diese Speicherbereiche können in „unsichtbaren“ Regionen des Grafikspeichers gehalten werden, wenn sie nicht zum Rendern vorgesehen sind. Abbildung 7 stellt das schematisch dar.

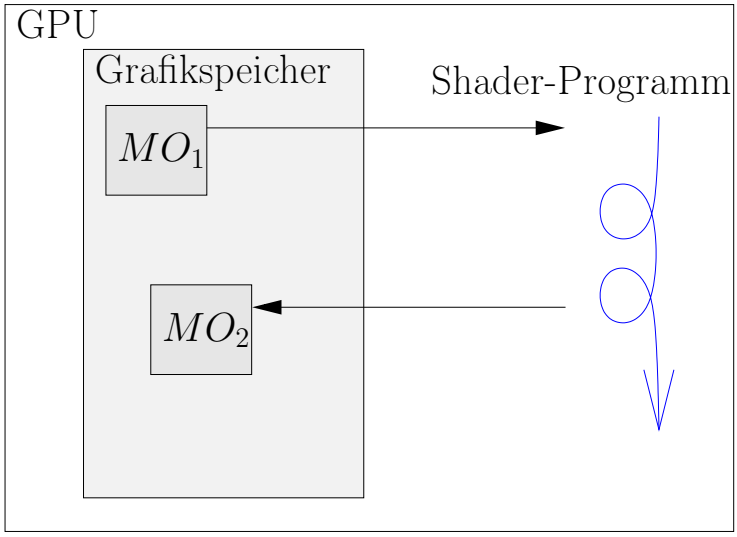

Abbildung 7. GPU-Shader-Programme: Speicheranforderung mittels OpenGL Memory Objects

Sortierung Die Sortierung der Partikel ist das Schlüsselelement für die korrekte Darstellung der Partikel in der Rendering-Phase. Zum Einen müssen Partikel nach ihrem Abstand zueinander sortiert werden um Kollisionen zu erkennen (1),

zum Anderen muss, für eine (halb-) transparente Darstellung von Partikeln, die Entfernung dieser zum Betrachter berücksichtigt werden (2).

Diese zwei Eigenschaften des UberFlow-Systems sind voneinander unabhängig und können während der Animation zu- oder abgeschaltet werden.

- Für Methode 1 wird der dreidimensionale Raum in ein quadratisches Gitter mit der Kantenlänge g_0 unterteilt. Der Sortierschlüssel für jedes Partikel berechnet sich nach $x/g_0^2 + y/g_0 + z$.
- Methode 2 speichert als Sortierschlüssel den Abstand des Partikels zum Betrachter.
- Beide Methoden teilen jedem Partikel zusätzlich einen eindeutigen Index zu.

Partikel werden nun in einer zweidimensionalen (RG)-Textur mit ihrem Index und Schlüssel gespeichert. Die Textur wird reihenweise sortiert und gibt die neue Ordnung für die erwähnten Partikel-Positions- und Geschwindigkeits-Texturen vor, die entsprechend neu arrangiert werden. Abbildung 8 stellt die zwei verschiedenen Methoden schematisch dar.

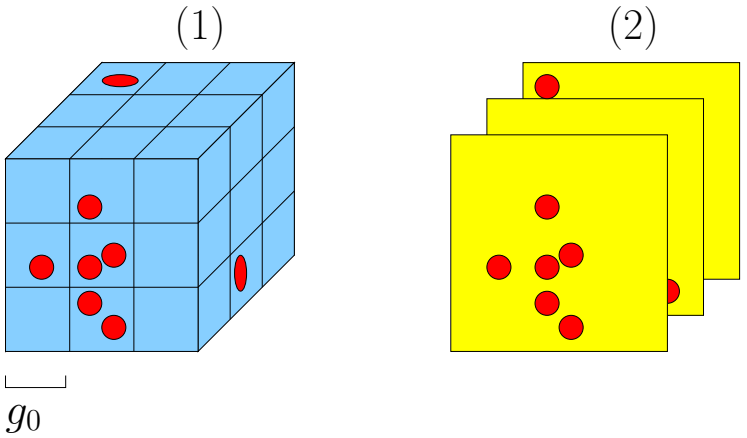

Abbildung 8. Partikelsortierung: links Gitter für Kollisionserkennung, rechts Abstand zum Betrachter für Transparenz-Effekte

In [KSW04] wurde viel Aufwand in den Sortieralgorithmus gesteckt, da er ein wichtiger Teil des Partikelsystems ist und relativ viel Rechenzeit benötigt. In dem Artikel wurde ein verbesserter *Bitonic*-Sortieralgorithmus verwendet. Anpassungen und Neuimplementierungen betrafen vor allem die Minimierung der auszuführenden Instruktionen per Shader und da insbesondere die Textur-Operationen.

Der verwendete Basisalgorithmus ist jedoch schon für die massiv parallele Ausführung auf GPUs ausgelegt.

Kollisionserkennung Die sortierte zweidimensionale (RG)-Textur enthält die Nachbarbeziehung(en) zwischen den Partikeln. Rechts und links nah bei einander Liegende haben auch im dreidimensionalen Raum eine geringe Entfernung. Zur Laufzeit-Optimierung wird jedoch nur der nächste (Kollisions-) Partner pro Partikel gesucht.

Multiple Kollisionen bleiben so unentdeckt. Des Weiteren hat die Zellbreite g_0 direkten Einfluss auf die mögliche Anzahl von Partikeln in einer Zelle. Es sind unter Umständen mehr als von einem Shaderprogramm überprüft werden kann. Die eigentliche Anordung erfolgt entsprechend dem Zellindex, sodass es passieren kann, dass die engsten Kollisionspartner in einer Reihe am weitesten auseinander liegen und somit auch nicht erkannt werden. Partikelkollisionen in adjazenten Zellen können aufgrund der Enumeration der Gitterzellen nicht ohne weiteres erkannt werden. Ein weiterer Nachteil betrifft die zeitliche Diskretisierung dt. Aufgrund dieser und abhängig von der Partikelgeschwindigkeit können Kollisionen, die zwischen aufeinander folgenden Bilder passieren, nicht detektiert werden.

Als (Teil-) Abhilfe sollte so die Zellenbreite g_0 immer klein genug gewählt sein, um die Partikelanzahl in einer Zelle nicht zu groß werden zu lassen. Um Zellkollisionen in adjazenten Zellen zu entdecken werden alle Partikel nach einem anderen Sortierschlüssel in einer zweiten Textur geordnet. So können Kollisionspaare erkannt werden, die wenigstens nach einer Sortierung in einer gemeinsamen Zelle enthalten sind (Abbildung 9). Obwohl insgesamt so in einigen Fällen keine Kollision erkannt wird, ist dies je nach Anwendung akzeptabel, da bei > 100.000 Partikel einige einfach nicht auffallen (Abbildung 10).

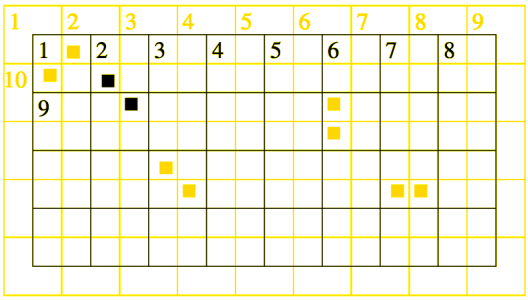

Abbildung 9. 2-Schrittpartikelsortierung: Kollisionserkennung manchmal erst im 2. Schritt (schwarzes Raster) [KSW04]

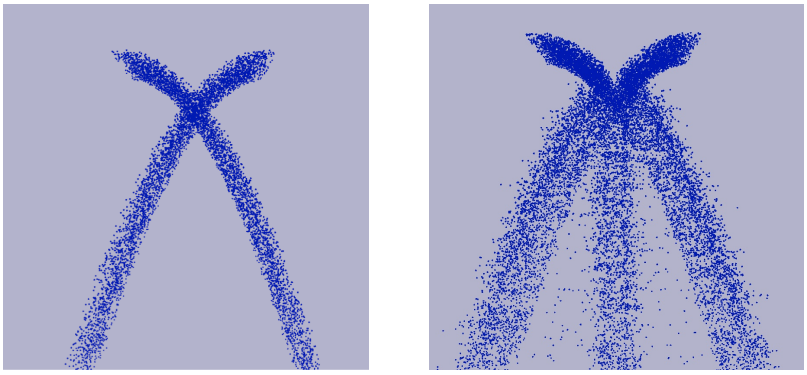

Abbildung 10. Ohne (links) und mit (rechts) Kollisionserkennung [KSW04]

3 Diskussion

Der Schritt von reinen CPU-Partikelsystemen hin zu GPU-Systemen ermöglicht, neben den bekannten Möglichkeiten einer Physiksimulation, die Interaktivität bei einer sehr großen Anzahl von Partikeln. Das in [KSW04] vorgestellte Partikelsystem nimmt sich vor, die Performance existierender GPU-Systeme durch die intelligente Implementierung von Algorithmen zur Partikel-Erzeugung, Manipulation und Darstellung auf der GPU wesentlich zu steigern. Als Grafikschnittstelle wurde das plattformübergreifende OpenGL, jedoch mit einer *ATI* spezifischen Erweiterung zur Laufzeitoptimierung, verwendet. Laut den Autoren ist es das erste Partikelsystem, dass zur Vermeidung des CPU-GPU-Busses, zur

Laufzeit komplett auf der GPU ausgeführt wird. Partikel-Partikel-Kollisionen werden erkannt, genauso wie solche mit der Oberfläche von Heightmaps. Eine für GPUs optimierte Sortierung wurde auch zur Sichtbarkeitssortierung implementiert, lässt sich darüber hinaus auch als generische Methode für nahezu alle Bereiche verwenden, in der Aufgaben es erfordern große Datenmengen zu ordnen. Damit ist es möglich den aktuellen Trend zur Ausnutzung der massiven parallelen Rechenkraft aktueller Grafikkarten auszubauen. Weiterhin lassen sich auch andere physikalische Modelle, je nach Anwendung, in dieses Partikelsystem integrieren.

Die präsentierten Resultate im Artikel, wie auch im Demovideo [TUM07], sind beeindruckend. Laut den Ergebnissen funktioniert die implementierte Variante bereits sehr gut. Man kann die Anzahl der dargestellten Partikel (unter dem Versuch, die Randbedingungen vergleichbar zu halten) mit dem in dieser Arbeit erwähnten Partikelsystem von NVIDIA, *particles*, vergleichen. Letzteres ermöglicht die GPU-beschleunigte Berechnung und Darstellung von ca. 32 tausend Partikeln inklusive Partikel-Partikel- und Umgebungs-Kollision. Dabei wird eine moderate Echtzeit von ca. 20 bis 24 Bilder pro Sekunde auf dem Laptop des Autors erreicht. Die Ergebnisse des Artikels zeigen eine ähnlich Bildrate (31) bei vergleichbaren Bedingungen aber mit ca. 262 tausend (512^2) Partikeln. Bei 1 Mio. Partikeln (1024^2) sind es immernoch 7 und bei deaktivierter Partikel-Partikel-Kollisionserkennung sogar 42 Bilder pro Sekunde.

Partikelsysteme sind heute bereits weit verbreitet und erfreuen sich großer Anwendung in Computerspielen, Animationen und Simulationen. Die Grundlagen dazu sind erkannt und publiziert (beispielsweise [Joh02, KSW04]). Das verwendete Physikmodell kann aber noch Gegenstand aktueller Forschung sein und in das vorgestellte Partikelsystem implementiert, damit getestet und angewandt werden.

Der Artikel diskutiert objektiv aktuelle Methoden für Sortierung und Partikelsysteme im Allgemeinen, hinsichtlich GPU-Beschleunigung. Die verwendete Grafikschnittstelle von OpenGL wird nur im Ansatz erwähnt. Des Weiteren wird auf die Grundlagen von Partikelsystemen nicht detailliert eingegangen. Dagegen ausführlich werden die Algorithmen zur Sortierung und Kollisionserkennung und ihre Nachteile dargelegt und diverse Anwendungen am Ende angedeutet.

In zukünftigen Arbeiten kann man die momentanen Nachteile des vorgestellten Systems beseitigen. Die nicht entdeckten Kollisionen sollten weiter minimiert und Algorithmen implementiert werden, die Mehrfachkollisionen erkennen. Auch

die proprietären Erweiterungen eines Grafikkartenherstellers sollten einer geräteunabhängigen Implementierung weichen, um die potentielle Hardwareunterstützung zu erweitern. Die Implementierung anderer Physikmodelle kann neue interessante Anwendungen ermöglichen.

Partikelsysteme sind ein Paradebeispiel um komplexe Effekte mittels prozeduraler Inhaltsgenerierung in der Computergrafik handhabbar zu machen und geeignet darzustellen. Der Artikel geht nur auf die notwendigen Details des vorgestellten Partikelsystems ein, aber auf diese richtig. Das Thema ist interessant und optisch ansprechend und bietet Potential für zukünftige Arbeiten.

Literaturverzeichnis

[Joh02] Johansen, Rune S.: *rune/vision - Rune's Particle System (POV-Ray)*. http://runevision.com/3d/include/particles/. Version: 2002, Abruf: Donnerstag, 10. Juni 2010

[KSW04] Kipfer, Peter ; Segal, Mark ; Westermann, Rüdiger: UberFlow: a GPU-based particle engine. Version: 2004. http://dx.doi.org/10.1145/1058129.1058146. In: *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. New York, NY, USA : ACM Press, 2004. – DOI 10.1145/1058129.1058146. – ISBN 3–905673–15–0, S. 115–122

[LR10] Lau, Oliver ; Rütten, Christiane: Farbrausch. In: *c't magazin für computer technik* 14 (2010)

[NVI10] NVIDIA: *CUDA Release Archive*. http://developer.nvidia.com/object/cuda_archive.html. Version: 2010, Abruf: Donnerstag, 10. Juni 2010

[TUM07] *tum.3D - Computer Graphics and Visualization*. http://wwwcg.in.tum.de/Research/Publications/UberFLOW_GH. Version: September 2007, Abruf: Dienstag, 29. Juni 2010