Collision Detection in Large Particle Systems

System Entwicklungsprojekt am Lehrstuhl für
Computer Graphics and Visualization der TU München

Sebastian Stratbücker

1 Aufgabenstellung

Aufgabe dieses Projekts war es, einen geeigneten Ansatz zur Kollisionserkennung innerhalb eines großen Partikelsystems[1] zu finden. Die Partikel sollten dabei kugelförmig sein und von beliebigem Radius, wobei es aber sinnvoll ist, einen maximalen Radius vorzugeben. Für dieses *schwierige* Problem gibt es sofort eine naive Lösung: *Teste jeden der **N** Partikel mit jedem anderen, ob ihr Abstand kleiner ist als die Summe ihrer Radii.* Daraus folgt aber direkt die Anzahl der Vergleiche für einen Zeitschritt des Partikelsystems, nämlich:

$$\frac{N(N-1)}{2} = O(N^2)$$

Diese Komplexität ist z.B. für Echtzeit 3D-Anwendungen nicht wünschenswert[3]. Deshalb wird oft versucht, durch Raumpartitionierung[2] die *Broadphase-Suche*[3] zu verbessern, d.h. räumlich weit entfernte Objekt werden nicht auf Kollisionen untersucht. Hierbei haben sich verschiede Verfahren als sehr effizient und robust bewährt, wie z.B *BSP-Bäume*[3] oder *Oct- bzw. Quadtrees*. Eine sehr schnelle Methode zur Erkennung von Kollisionen unter beliebigartigen 3D-Körpern stellt *I-COLLIDE*[5] zur Verfügung. Die Anforderungen an eine Kollisionserkennung unter verschieden großen Kugelkörpern sind allerdings weit geringer. Trotzdem ist es nützlich, dafür einige gute Verfahren parat zu haben, da oft komplexere Körper oder Flüssigkeiten durch Kugeln approximiert [6] oder durch hierarchische Modelle vereinfacht werden[3].

2 Ansatz

Bei der folgenden Methode handelt es sich um ein einfach zu implementierendes Verfahren, dass in einem Partikelsystem auf einem beschränkten 3 dimensionalen Gebiet mit einer maximalen und nicht beliebig kleinen Partikelgröße **R**, kollidierende[1] Partikel während eines Zustands des Systems erkennt. Die gefundenen Kugelpaare werden dann durch eine Rückstellkraft so bewegt, dass Überschneidungen vermieden werden.

[1] Kollisionen werden hier gleichgesetzt mit Überschneidung eines Kugelpaares

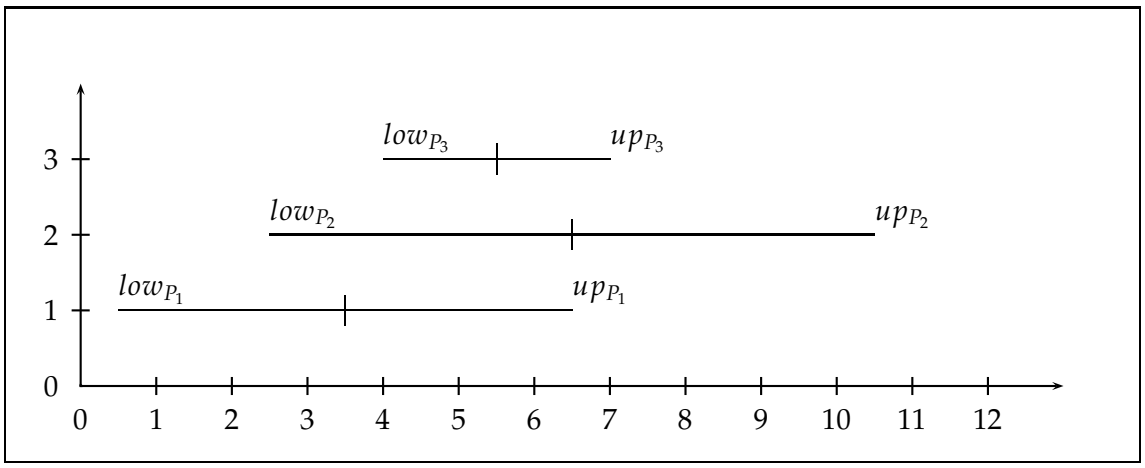

Abbildung 1: Überschneidung der Partikel auf einer Raumachse

Das ist in diesem Fall von großer Bedeutung, da einige Annahmen sehr wohl auf Partikelradius und Verteilung beruhen.

Die Partikel $P_i \quad i \in N$ werden zunächst entlang einer Raumachse $d \in D = \{x, y, z\}$ *absteigend* sortiert. Dabei verwendet man den unteren Wert der Partikelgrenzen auf dieser Achse, also $pos^d_{P_i} - rad_{P_i} = low_{P_i}$. Entsprechend ist die Obergrenze definiert durch $pos^d_{P_i} + rad_{P_i} = up_{P_i}$.
Auf den verbleibenden zwei Raumachsen $e, f \in D \quad e \neq f \neq d$ wird ein Gitter $G_{U,V}$ aufgespannt. Es besteht aus $U \times V$ quadratischen Zellen $G_{u,v} \in \mathbb{R} \times \mathbb{R} \quad u \in [1..U], v \in [1..V]$ und vorgegebener Seitenlänge s_G. Diese Länge wird aus dem maximalen Radius rad_{max} bestimmt und ist idealerweise $2rad_{max} = s_G$.

Nun werden die P_i jeweils einer Zelle $G_{u,v}$ zugeordnet,d.h. $(pos^e_{P_i}, pos^f_{P_i}) \in G_{u,v}$. Für alle Partikel $P_i^{u,v}$ innerhalb der Zellen $G_{u,v}$ gilt nun $low_{P_i^{u,v}} <= low_{P_j^{u,v}} \quad i < j$.

Will man nun feststellen, mit welchen Partikeln P_i eventuell kollidiert, testet man ob $up_{P_i^{u,v}} > low_{P_j^{u,v}} \quad j > i$. Das würde bedeuten, die Projektionen (vgl. [5]) der Partikelvolumina auf die Raumachse d überschneiden sich (Abb.1).

Aufgrund der Gitterstruktur $G_{U,V}$ kann man nicht davon ausgehen, dass alle möglichen Kollisionspartner somit gefunden werden. Da ein Partikel nicht vollständig innerhalb eines Gebietes $G_{u,v}$ liegen muss, überschneidet er gegebenenfalls auch dessen Nachbarschaft $\tilde{G}_{u,v} \subset \biguplus G_{u+e,v+e} \quad e \in \{-1, 0, 1\}$. Deshalb ist es notwendig, diese Nachbarzellen ebenfalls zu testen. In Abbildung 2 sind einige dieser Nachbarschaften hervorgehoben. Auf ihr ist leicht zu erkennen, dass höchstens 3 benachbarte Zellen zusätzlich belegt werden können - unter der gegebenen Voraussetzung, dass $s_G > rad_{max}$. Diese einfache Behauptung bleibt deshalb ohne Beweis.

Mit diesen Annahmen ist es nun möglich, eine Kollisionserkennung anzugeben, die un-

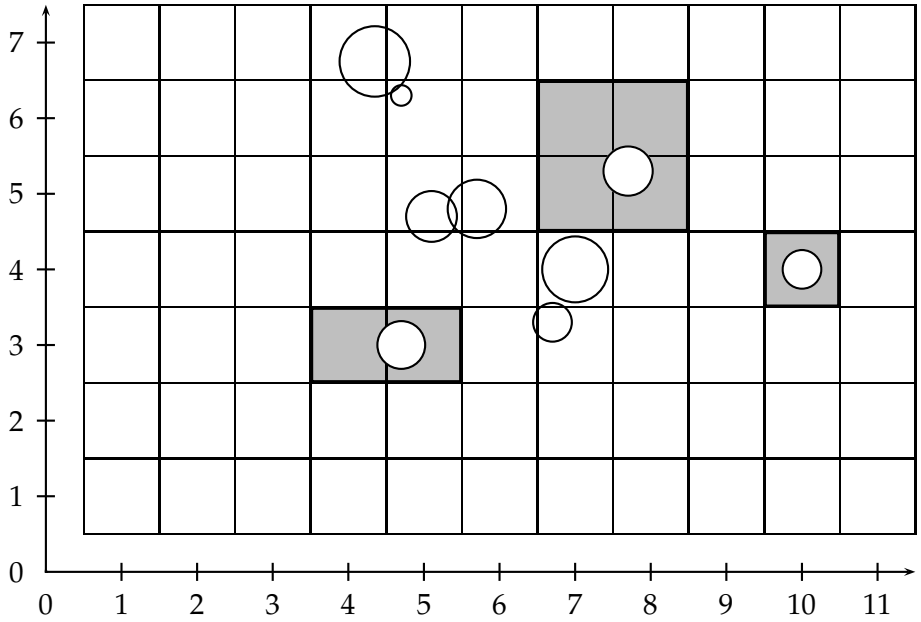

Abbildung 2: Projektion der Partikel auf die Gitterebene

ter bestimmten Voraussetzungen von der Komplexität $O(N)$ ist. Diese sind vor allem, dass die Dichte und Energie des Partikelsystems gewissen oberen Schranken unterliegen. Die Dichte sollte deshalb stets auf einem Niveau bleiben, in dem sich *keine* Partikel über mehrere Simulationsschritte hinweg überschneiden. Kollisionen *müssen* also behoben werden. Die Energie des Systems drückt sich unter anderem durch die mittlere Geschwindigkeit, aber auch den maximalen Geschwindigkeitsvektor der Partikel aus. Diese maximale Geschwindigkeit ist mitunter sehr entscheidend für die Betrachtung der Zeitkomplexität des Algorithmus.

3 Datenstrukturen

Bei dem dem verwendeten Partikelsystem handelt es sich um eine *mehrfach* verkettete Liste von *Partikelstrukturen*. Ein Partikel P_i besitzt neben festem Radius und Masse, jeweils einen Vektor für Position,Geschwindigkeit und wirkende Kraft. Ebenso werden low_{P_i} und up_{P_i} Werte benötigt, um die Sortierung und die anschließenden Vergleiche durchführen zu können. Die 5 verwendeten Listenzeiger auf Partikelstrukturen dienen dazu :

- eine globale, sortierte Liste über alle Partikel zu verwalten
- lokale, sortierte Listen für jede Gitterzelle zu verketten
- jeweils einen Verweis auf die drei möglichen *Nachbarlisten* zu halten

Das Gitter $G_{U,V}$ wird durch $U \times V$ *Partikelzeiger* realisiert, die über ein zweidimensionales Array indiziert werden.

4 Algorithmus

4.1 Liste sortieren

Zentraler Punkt der Kollisionserkennung ist die sortierte, globale Partikelliste $P_N(t)$. Sie hält alle Partikel des Systems und bewahrt eine Ordnung $\geq^{low}$ für jeden Simulationsschritt $t \in \mathbb{N}_0$. Diese Ordnung bleibt offensichtlich von Schritt t zu Schritt $t+1$ größtenteils erhalten. Man kann sagen $P_N(t)$ ist vorsortiert, falls $t > 0$ und $vel^d_{P_i} < vel_{max}$. Es sei also die Geschwindigkeit der Partikel entlang der Raumachse d beschränkt, d.h. die *Energie* des Systems ist so gering, dass kein Partikel große Teile des Gesamtvolumen des Partikelsystems in einem Zeitschritt durchlaufen kann. [2] Wenn man nun annimmt, dass $P_N(t) \quad t > 0$ vorsortiert ist, liegt es nahe, ein Sortierverfahren anzuwenden das dies berücksichtigt. Das hier verwendete *tryMergeSort* auf Listen erfüllt die Anforderungen. Ähnlich wie das bekannte *mergeSort*, ist auch dieser Algorithmus von der Komplexität $O(N \log N)$[4]. Allerdings zeigt sich ein hervorragendes *best-case* Ergebnis bei vorsortierten Listen, wie in diesem Fall. Zuerst testet *tryMergeSort*, ob die Liste bereits sortiert *ist*. Das geschieht in $O(N)$ Vergleichen. Falls die Ordnung allerdings verletzt ist, wird die Liste mittels eines *Reißverschlussverfahrens* in $O(N)$ Schritten in P_{gerade} und $P_{ungerade}$ getrennt. Danach werden wie gewöhnlich die zwei neuen Listen rekursiv übergeben und sortiert. Das übliche *mergen* soll hier nicht erläutert werden. Der Platzbedarf bei dieser Methode ist $O(N)$. Wie man unschwer erkennt, hängt die Rekursionstiefe direkt vom Grad der Unordnung ab. Sind z.B. die einzigen Verletzungen der Ordnung von der Art $low_{P_i} \leq low_{P_{i+1}} \quad i \in N$, so sind die beiden Listen nach der Trennung bereits sortiert, und die Rekursion bricht ab.

4.2 Gitter besetzen

Der nächste Schritt ist dann, aus dem globalen $P_N(t)$ ein $P^{U,V}(t)$ zu erstellen, indem man alle P_i einem $G_{u,v}$ zuordnet.

$$P^{U,V}(t) = \{P^{u,v} | \quad u \in U, v \in V, (pos^e_{P^{u,v}_k}, pos^f_{P^{u,v}_k}) \in G_{u,v}, k \in L_{u,v}\}$$

Wobei $L_{u,v} \subseteq N$ der Anzahl von Partikeln in $P^{u,v}$ entspricht.

[2] Sinnvoll ist es eher, das Geschwindigkeitsmaximum etwa bei rad_{max} anzusetzen, damit alle Kollisionen erkannt werden können[3].

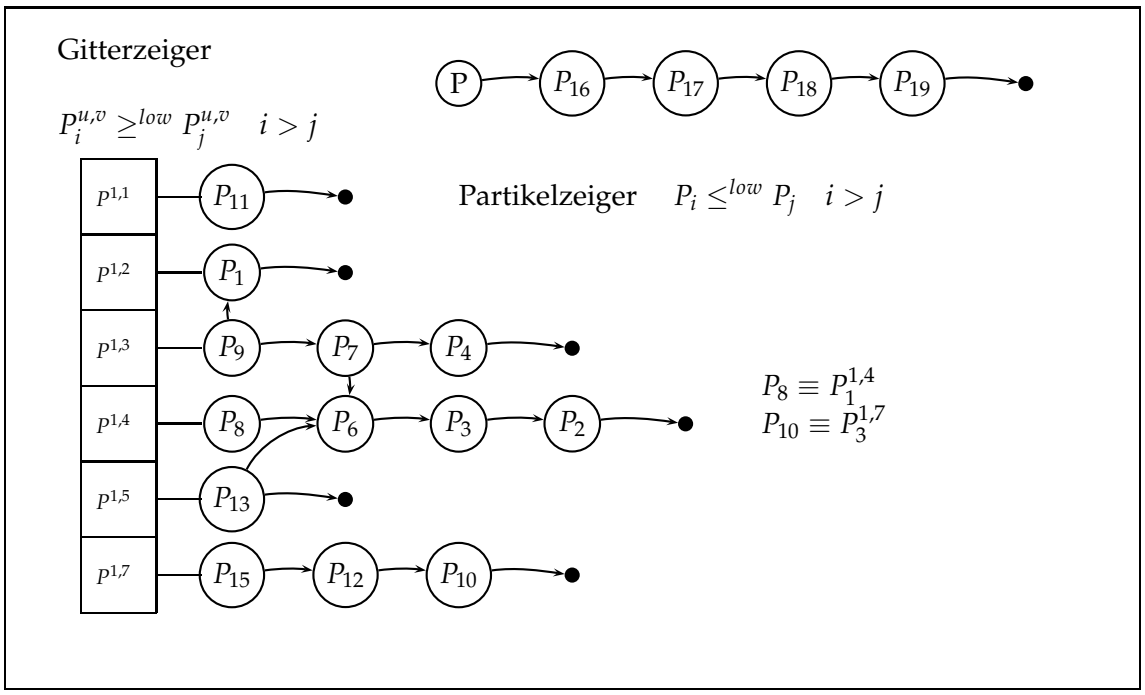

Abbildung 3: Einsortieren der Liste in das Gitter in O(N).

Für die so gebildeten $P_k^{u,v}$ gilt nun aber:

$$low_{P_k^{u,v}} \geq low_{P_l^{u,v}} \iff k > l \quad k, l \in L_{u,v}$$

Auf dieser neu gewonnenen Ordnung [3] basiert nun die *broad-phase* der Kollisionserkennung. Will man nun ebenfalls die Partikel in den umgebenden Gitterzellen berücksichtigen, ist es günstig, die Nachbarn bereits beim Einsortieren des Gitters zu wählen. So erhält man die relevanten Partikel direkt aus dem aktuellen Gitterzustand $P_k^{U,V}$, da alle $low_{P_1^{u,v}} >= low_{P_k}$. Wobei P_k der gerade einzuordnende Partikel ist. In Abbildung 3 ist ein Gitterzustand $P_{16}^{U,V}$ dargestellt. Neben den eigentlichen $P_k^{u,v}$ sind dort auch einige Verweise auf eventuelle Kollisionspartner in den Nachbarzellen eingezeichnet. Das erzeugen einer $P^{U,V}(t)$ Struktur aus den sortierten $P_N(t)$ geschieht mit einem Aufwand $O(N)$.

[3] Bedingt durch die einfach verketteten Partikellisten, kehrt sich die Ordnung beim Einfügen in das Gitter um.

4.3 Kollisionen suchen

Wie bereits anfangs erwähnt, untersucht man nun innerhalb $P^{U,V}(t)$, ob sich Partikel auf der Achse d überschneiden. Wenn das der Fall ist, gilt :

$$up_{P_k^{u,v}} > low_{P_l^{u,v}} \quad k < l \quad k,l \in L_{u,v}$$

Zusätzlich ergibt sich aus der Dreiecksungleichung für einen gerade betrachteten Partikel $P_k^{u,v}$ aus der gegebenen Ordnung $\leq^{low}$:

$$up_{P_k^{u,v}} < low_{P_l^{u,v}} \Rightarrow up_{P_k^{u,v}} < low_{P_m^{u,v}} \quad k < l < m \quad k,l,m \in L_{u,v}$$

Das bedeutet aber für den Algorithmus, dass die Abbruchbedingung klar gegeben ist:

> Wenn für einen Partikel $P_k^{u,v}$ keine Überschneidung mit dem Partikel $P_{k+i}^{u,v}$ stattfindet, wobei $k < l$ und $i = 1, 2, 3..$, dann gibt es auch keine Überschneidungen mit einem Partikel $P_{k+j}^{u,v} \quad j > i$.

Wie wir aber angenommen haben, ist die Dichte des Partikelsystems endlich und im besten Fall kleiner als die der Partikel, also gibt es wenige - oder besser - *keine* Überschneidungen. Dann gibt es auch ein $i_{max} \ll N$, so dass $up_{P_k^{u,v}} < low_{P_{k+i_{max}}^{u,v}}$. Allerdings wird dieses i_{max} auch von rad_{min} bestimmt, welches letztenendes wiederum für die $L_{u,v}$ mitverantwortlich ist. Abschließend kann man davon ausgehen, dass i_{max} bei gleichbleibender Dichte des Systems um einen festen Wert schwankt, und nicht von N abhängig ist. Somit ergibt sich für diese Phase des Verfahrens ein Aufwand von $O(N)$ und ein zu vernachlässigender zusätzlicher Platzbedarf von O(UV).

Insgesamt wird die Komplexität des Verfahrens maßgeblich durch die Rekursionstiefe R_{sort} von *tryMergeSort* bestimmt und diese ist wiederum abhängig von der Gesamtenergie des Systems. Bei einem statischen System ohne jegliche Bewegungsenergie ergibt sich ein optimales $O(N)$, bei gleichbleibender Dichte des Partikelsystems. Kann man allerdings Aussagen über die *Bewegungsenergie* E des Systems machen, d.h. wie stark ändern sich die Positionen der Partikel bei einem Simulationsschritt $t \rightarrow t + 1$, so läßt sich auch ein Zusammenhang zu $R_{sort}(E) \in [1..\log_2 N]$ herstellen, der bisher allerdings nur statistisch in Versuchen gezeigt werden konnte. Ohne diese Kenntnis aber, muss man sich auf eine Gesamtkomplexität von $O(N \log N)$ beschränken, obwohl sie in der Praxis vermutlich bei einem $O(NR_{sort}(E))$ liegt.

5 Probleme

Derzeit ist es noch nicht möglich, alle Kollisionen während eines Simulationsschrittes zu erfassen. Einige Partikelkonstellationen können nicht durch das aufgesetzte Gitter

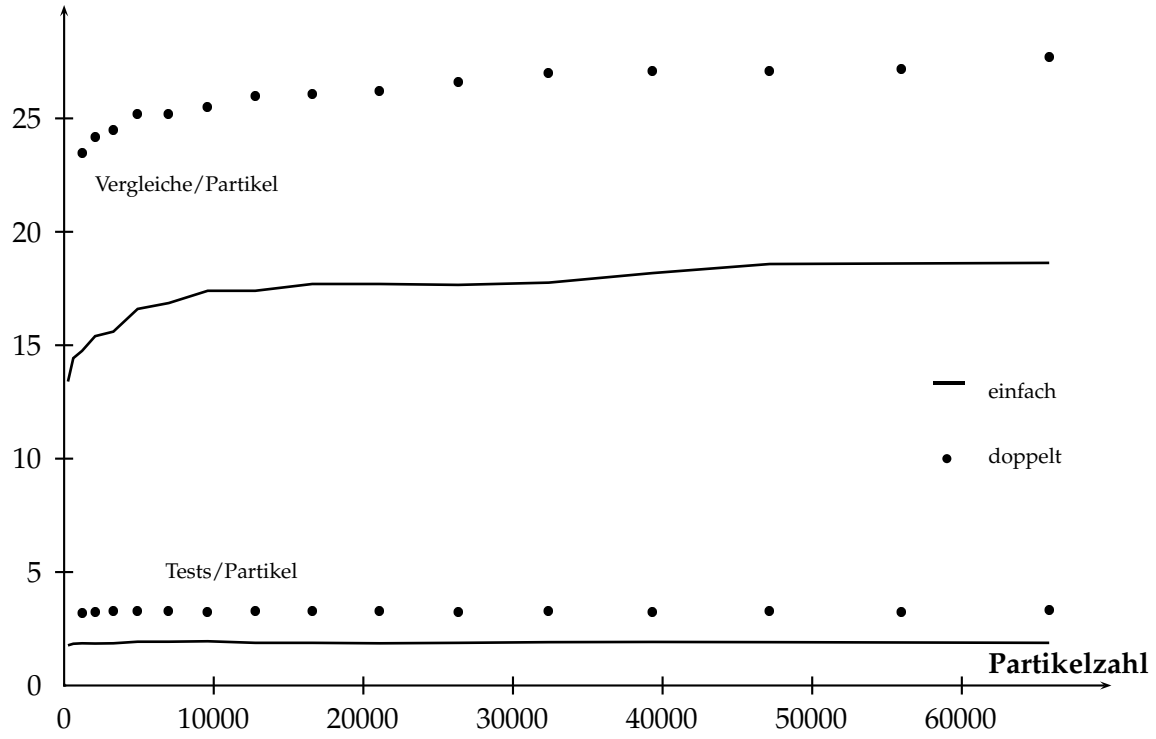

Abbildung 4: Kollisionstests und Gesamtvergleiche pro Partikel

mittels dem vorgestellten Verfahren detektiert werden. Überschneidungen mit Partikeln aus anderen Zellen können nämlich nur erkannt werden, wenn der Nachbarpartikel in $\leq^{low}$ größer ist, da nur in aufsteigender Richtung nach Überlappungen gesucht wird. So kann es passieren, dass ein Partikel $P_k^{u,v}$ nur eine Zelle belegt, aber tatsächlich kollidiert er mit einem $P_l^{q,r}$ mit $low_{P_k^{u,v}} \leq low_{P_l^{q,r}}$. Dieses Problem wird dadurch behoben, dass das Gitternetz bei jeder zweiten Einsortierung um den Vektor $(\frac{s_G}{2}, \frac{s_G}{2})$ auf den Achsen e, f verschoben wird. Dadurch werden alle *inaktiven* Nachbarn zu *aktiven* Partikeln, die in der Umgebung auf Überlappung testen. Diese Translation kann sowohl bei jedem t_{gerade} durchgeführt werden, was bedeutet, dass erst bei jedem zweiten Schritt alle Kollisionen entdeckt wurden. Oder man führt die Einsortierung in das Gitter und die anschließenden Überlappungstests pro Simulationsschritt zweimal durch. Dadurch ergibt sich natürlich eine zusätzliche lineare Komponente bei der Komplexität.

6 Ergebnisse

Die Messungen aus einigen Simulationen mit konstanter Dichte des Partikelsystems zeigen deutlich, dass die Zahl der Überlappungstests pro Partikel nicht mit N steigt. Die Zahl der Gesamtvergleiche, also unter anderem während der Sortierphase steigt zwar anfangs logarithmisch, scheint sich allerdings bei hohen Partikelzahlen auf einen

festen Wert einzupendeln. Auf dem Plot von Abbildung 4 sind die mittleren Werte für die einfache, als auch für die doppelte Gittersortierung dargestellt.

Die grafische Darstellung des Partikelsystems wurde mit der *OpenGL* Schnittstelle realisiert. Implementiert wurde das System auf *Visual C++*. Bei Simulationen auf 2.8GHz Rechnern mit der neuesten *Graphic Hardware* Generation[4], ließen sich bei mehreren Zehntausend Partikeln sehr wohl *Frameraten* von ruckelfreien 25-30Hz erreichen.

Literatur

[1] Andrew Witkin. Particle System Dynamics, 1997

[2] Dong Jin Kim, Leonidas J. Guibas, Sung Yong Shin. Fast Collision Detection among Multiple Moving Spheres ,1997

[3] R. Westermann. Game Programming (Vorlesung am VIS.3D Lehrstuhl)

[4] Jurg Nievergelt, Klaus Hinrichs. Algorithms & Data Structures - with applications to graphics and geometry, 1993

[5] Jonathan D. Cohen, Ming C. Lin ,Dinesh Manocha, Madhav K. Ponamgi.
I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments

[6] Nikhil Gagvani, Deborah Silver. Shape-based Volumetric Collision Detection

[4] Die Tests fanden im September 2003 statt