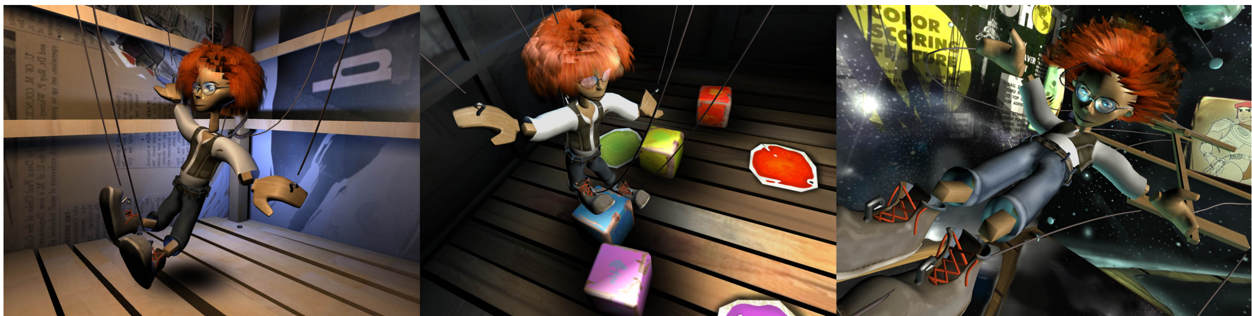
Physikalische Simulation virtueller Charaktere durch Feder-Masse-Systeme am Beispiel einer „Virtuellen Marionette“

Tobias Ritschel, Stefan Müller

AG Computergrafik
Universität Koblenz-Landau
56016 Koblenz
Tel.: +49 (0)261 - 28 72 733
E-Mail: ritschel@uni-koblenz.de



Zusammenfassung: Virtuelle Charaktere stellen eine große Herausforderung für die physikalische Simulation, vor allem in Echtzeit, dar. Diese Arbeit stellt der dazu oft verwendeten Starrkörper-Simulationen ein Feder-Masse-System gegenüber. Es wird gezeigt, dass die meisten für Charaktere relevanten physikalischen Erscheinungen bei der Animation von Kleidung, Haaren, Gliedmaßen, Gelenken mit Freiheitsgraden, usw. vereinheitlicht auf ein Feder-Masse-System zurückgeführt werden können. Solche Systeme verhalten sich zwar nicht immer physikalisch exakt, bieten aber dafür höhere Performanz, bessere Skalierbarkeit, einfachere Implementierung und im Rahmen ihrer Möglichkeiten, größere „empfundene“ Stabilität. Die einfache Form der Verarbeitung macht eine effiziente Implementierung mit SIMD-Instruktionen möglich, die vorgestellt wird. Das System wird erfolgreich verwendet, um eine „Virtuelle Marionette“ mit getrackten Eingabegeräten so zu manipulieren, dass diese in einer virtuellen physikalischen Umgebung einfache Aufgaben (Puzzle, Fußball) lösen kann.

Stichworte: Virtuelle Charaktere, physikalische Simulation, Feder-Masse-Systeme, SIMD

1 Einleitung

Diese Arbeit beschreibt die Verwendung von *Feder-Masse-Systemen* im Vergleich zu Starrkörper-Verfahren bei der physikalischen Simulation virtueller Charaktere am Beispiel einer „*Virtuellen Marionette*“. Bei dieser wird ein reales Spielkreuz optisch getrackt, wobei die Bewegungen auf ein virtuelles Spielkreuz übertragen werden. Der Körper der Marionette wird virtuell an acht Seilen aufgehängt, die mit dem virtuellen Spielkreuz verbunden sind. Die Marionette reagiert mit

Hilfe einer physikalischen Simulation in Echtzeit und sehr realitätsnah auf die Bewegungen des Benutzers. Dazu werden auch Haare und Stoff simuliert und die Marionette reagiert auf ihre Umwelt durch animierte Gesichtsausdrücke. In ansprechend gestalteten und ebenfalls physikalisch simulierten Räumen kann man schließlich versuchen, verschiedene Aufgaben mit Hilfe der Virtuellen Marionette zu lösen, wie z. B. das Zurechtrücken farbiger Kisten auf dazu passende farbige Flächen oder das Marionetten-Fußball. Die Darstellung der Marionette ist von hoher Qualität, wobei sehr detaillierte Geometrien und Texturen mit Hilfe von Shadern auf einer Stereoleinwand in hohen Bildraten visualisiert werden.

Diese Paper ist im Weiteren wie folgt aufgebaut: Im zweiten Abschnitt wird auf die vorherrschenden Techniken zur physikalischen Simulation eingegangen, im dritten Abschnitt werden die Komponenten des verwendeten Feder-Systems beschrieben, zu denen im vierten Teil die genutzten Optimierungen vorgestellt werden. Die Arbeit schließt mit einer Bewertung und einem Ausblick.

2 Stand der Technik

Die gängige Technik zur Simulation physikalischer Prozesse arbeitet meist wie folgt: eine zu simulierende Welt wird aus einer Anzahl von Körpern verschiedener Art konstruiert, zwischen denen Beziehungen bestehen und auf die Kräfte wirken. Die Zustände der Körper werden dazu nach der Systematik der Physik in Größen wie Ort, Geschwindigkeit, Beschleunigung und Trägheit für Ort und Orientierung beschrieben. Die Beziehungen zwischen den Körpern werden durch verschiedene Techniken modelliert, die z. B. Körper an Gelenken verbinden, indem sie Kräfte berechnen, die für einen Zusammenhalt sorgen. Solche Systeme verwenden meist physikalisch exakte Größen, variable oder sogar adaptive Zeitschritte und verschiedene Integratoren, die das Verhalten des Systems im Zeitverlauf berechnen [WB97]. Oft werden die auftretenden Objekte dabei nach Prinzipien der Objektorientierung und aus der Perspektive der Physik strukturiert. Die entstehenden Systeme sind komplex und voller Abhängigkeiten und Sonderfälle. Der Call-Graph dieser Systeme ist heterogen und es wird viel Zeit in vielen verschiedenen Funktionen verbraucht, was eine Optimierung erschwert. Außer Frage steht in jedem Fall die Machbarkeit solcher Ansätze.

Dem gegenüber werden Feder-Masse-Systeme erfolgreich zur Simulation diverser einzelner Erscheinungen im Bereich der deformierbaren Körper, wie Stoff und Haaren, eingesetzt. Jakobsen [Jak01] beschreibt ein System, das auch andere Erscheinungen auf Feder-Masse-Systeme zurückführt. Er weist darauf hin, dass Verlet-Integration [Ver67] und das Erfüllen von Constraints „durch Projektion“ [Ebe04] ein solches System ideal ergänzt und gute Performanz mit einer einfachen Implementierung verbindet.

Einige neuere Ergebnisse [Lat04, MSr05] nutzen die GPU zur physikalischen Simulation von Partikeln. Obwohl das hier beschriebene System keine GPU-Implementierung vorsieht, lassen sich die dort gewonnenen Erkenntnisse auch auf die Umsetzung für die CPU (z. B. SIMD) rückbeziehen.

3 Feder-Masse-Systeme

3.1 Setup

Im hier beschriebenen System wird ein *Character* als eine Menge von Objekten modelliert, zwischen denen Beziehungen mit bestimmten Freiheitsgraden existieren, und von denen einige vom Nutzer beeinflusst werden können. Einige Objekte sind *verformbar* (Haare, Fäden, Stoff), andere sog. *Starkörper*. Als Eingabe dienen die gleichen Quelldaten, die auch die Darstellung verwendet und die um physikalische Attribute erweitert wurden. Dabei handelt es sich um hochaufgelöste und texturierte Geometrie, die durch das Szenegraphen-System OpenSG [RVB02] verwaltet und dargestellt wird.

3.2 Elemente

Die Simulation arbeitet im Wesentlichen auf zwei Elementen: Partikeln (oder Massepunkten) und den sog. „Constraints“ (oder (Un-)Freiheitsgraden).

Die *Massepunkte* sind durch ihren Ort, ihre Geschwindigkeit, ihre Beschleunigung und ihr Gewicht bestimmt. Zur Darstellung des Gewichts wird die inverse Masse $1/m$ verwendet, durch die auch unbewegliche Partikel mit unendlicher Masse ($1/m = 0$) modelliert werden können. Weiter wirken im System sog. *Constraints*. Diese können als Bedingungen verstanden werden, die das System versucht einzuhalten. So sollen z. B. bestimmte Partikel sich nicht in einem *Halbraum* aufhalten, bestimmte Paare aus Partikeln eine bestimmte *Distanz* einhalten, oder 4-Tupel aus Partikeln einen Tetraeder bestimmten *Volumens* bilden. Auch exotischere Bedingungen sind denkbar, z. B. einen bestimmten Abstand zu einer Spiralkurve im Raum einzuhalten. Auch muss die Bedingung sich nicht nur auf den Zustand des Ortes beziehen: es können z. B. Partikel versuchen eine bestimmte Geschwindigkeit zu halten (*Motor*), oder Winkel zu bilden (*Angular Constraints*). Wie all diese Constraints sich realisieren ist dabei unerheblich, es gilt nur: sie müssen das System in den Zustand überführen, in dem sie gelten [Jak01]. Insbesondere wirken sie *nicht* über Kräfte. Ist ein Partikel z. B. im falschen Halbraum, wird er einfach heraus projiziert. Auch kann ein „Auf die Schraubenlinie zu bewegen“ als Projektion im Zustandsraum verstanden werden, es handelt sich bei Constraints also allg. um *Projektionen im Zustandsraum*, im Sonderfall identisch mit der Projektion im metrischen Raum [Ebe04], [WB97]. So lange nur ein Constraint erfüllt werden muss, arbeitet ein solches System exakt. Sind mehrere Constraints zu erfüllen (z. B. eine Distanz einzuhalten und in einem Halbraum zu bleiben), kann es sein, dass diese sich *entgegenwirken*. Dieses Problem wird gelöst, indem man annimmt, dass die Maßnahmen nicht völlig gegensätzlicher Natur sind und durch wiederholtes Anwenden (*Relaxation*) ein Kompromiss herbeigeführt werden kann.

Alle weiteren Elemente dienen nur noch der Vereinfachung von Operationen auf den Massepunkten. Es existieren globale Listen von Kanten, Flächen und Körpern (*Meshes*). Darüber hinaus wirkt noch eine Anzahl von Kräften auf das System. Es wird jedoch nicht jede Wirkung als Kraft realisiert, was einen wichtigen Unterschied zu normalen Starrkörper-Simulationen darstellt.

3.3 Simulation

Der Ablauf der Simulation kann so beschrieben werden:

```
loadFromOpenSG();
forever {
  for i = 0..simulationStepCount {
    applyForces();
    applyConstraints();
    integrate();
    storeToOpenSG();
  }
}
```

Da die Ausgangsdaten nicht als Partikelsystem vorliegen, kommt es in der Funktion `loadFromOpenSG` einmal zu einer *Übersetzung*. Neben dem Szenegraphen zur Darstellung wird ein Partikelsystem *getrennt* gepflegt, was z. B. die Speicherung der Partikel geeignet flexibel macht, um diese sequentiell und einfach verarbeiten zu können. Beim Laden des Szenegraphs, wird jedem Node die Gelegenheit gegeben, sich als physikalisches Simulationsobjekt (*Mesh*) darzustellen und die Verbindung zwischen Simulations-Mesh und OpenSG-Node gespeichert. Die Funktion `integrate` geht über alle Partikel und wendet den Verlet-Integrator an. Die Funktion `applyForces` geht alle Kräfte durch, und gibt diesen Gelegenheit, sich auf Partikel anzuwenden. Die Funktion `applyConstraints` enthält eine weitere Schleife über eine Anzahl von Schritten. Auf jede Integration kommen `realaxationStepCount` Anwendungen der Constraints:

```
for i = 0..realaxationStepCount {
  for j = 0..constraintCount {
    constraint[i]->apply();
  }
}
```

Die Funktion `storeToOpenSG` traversiert abschließend den Szenegraph und ruft für jene Nodes, die physikalisch simuliert werden, eine virtuelle Funktion ihres Simulations-Meshes auf, die das Partikel-System wieder in OpenSG übersetzt. Die Simulation enthält also zwei einstellbare Parameter: die Anzahl der Relaxationen der Constraints (`realaxationStepCount`) und die Anzahl der Zeitschritte pro Bild (`simulationStepCount`). Typischerweise liegen beide zwischen 3 und 7.

3.4 Verlet-Integration

Die verwendete Verlet-Integration arbeitet wie folgt. Zunächst gilt:

$$r' = r + vdt + adt^2$$

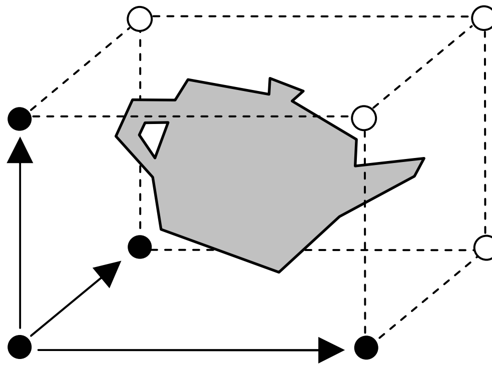


Abbildung 1: Starrkörper

Wobei r' für die neue, r für die alte Position, v für die Geschwindigkeit und a für die Beschleunigung steht. Legt man dt als 1 fest, ergibt sich

$$r' = r + v1 + a1^2 = r + v + a$$

Die Vorraussetzung dt festlegen zu können, kann erfüllt werden, indem man die Simulation immer mit $dt = 1s$ durchführt. Da aber $1s$ ein viel zu großer Schritt ist, müssen alle anderen Einheiten angepasst werden. Es ist zuzugeben, dass man sich hierbei von physikalisch exakten Einheiten zur Laufzeit entfernt. Diese mögen als Eingabe-Daten angemessen sein, zur Laufzeit zerstören sie nur Präzision und machen alles komplizierter. Verwendet man statt der Geschwindigkeit jetzt die Differenz zur letzten bzw. vorletzten Position r_{old}, an Stelle der Geschwindigkeit, ergibt sich:

$$r' = r + (r - r_{old}) + a = 2r - r_{old} + a$$

Diese Operationen sind *einfach* und ergeben *stabilere Schritte*, wenn die Constraints um sich zu realisieren, nur den Ort anpassen müssen. Pro Massepunkt werden bei Verlet-Integration also nicht die Geschwindigkeit, sondern die alte und die neue Position gespeichert.

3.5 Meshes

Die verschiedenen Mesh-Typen sind eine Methode, die Menge der Partikel zu strukturieren. Sie dienen dazu, beim Startup den Pool der Partikel, Kanten und Flächen mit bestimmten Kombinationen zu füllen, werden als Granularität der Kollision verwendet und kapseln die Übersetzung von und nach OpenSG. Die folgenden Abschnitte beschreiben die verschiedenen Mesh-Typen. In den Abbildungen dazu stehen Kreise für Partikel, gestrichelte Linien für Federn, einfache Linien für Geometrie des Szenegraphen und breite Linien für Boundingvolumen.

3.5.1 Starrkörper

Die Simulation von *Starrkörpern* mittels eigentlich flexibler Feder-Masse-Systeme ergibt sich wie folgt: beim Startup wird für jedes Objekt seine achsenparallele Boundingbox (*AABB*) berechnet,

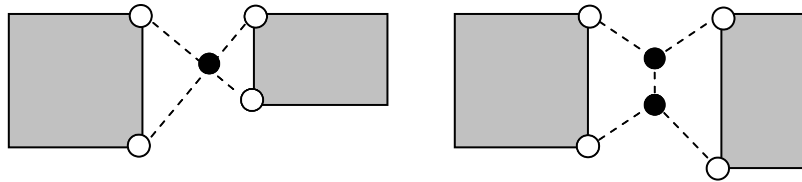


Abbildung 2: Kugel- (links) und Scharnier-Gelenk (rechts)

acht Partikel an ihren Ecken und alle dazwischen möglichen Federn erzeugt. Davon werden vier beliebige Partikel-Positionen p_0, bis p_3 gewählt die nicht in einer Ebene liegen und somit drei Richtungen d_0, bis d_2 angeben:

$$d_0 = p_1 - p_0$$
$$d_1 = p_2 - p_0$$
$$d_2 = p_3 - p_0$$

In Abb. 1 wird die Auswahl von Massepunkten zu einer AABB eines Teapots dargestellt. Diese Richtungen werden normalisiert, in eine Matrix $A_{startup}$ eingetragen und gemerkt. Sowohl die Matrix des OpenSG-Nodes als auch die Partikel bleiben davon unverändert. Zur Laufzeit wird diese Matrix aus den gleichen Punkten wieder bestimmt. Es ergibt sich eine i.A. andere Matrix $A_{runtime}$, da die Partikel ihre Position verändert haben. Der Unterschied von $A_{startup}$ zu $A_{runtime}$ wird berechnet und nur dieser rechts an die Transformation im Szenegraph multipliziert. Dieses Vorgehen koppelt die beliebig komplexe Geometrie eines OpenSG-Nodes an die beliebig einfache Geometrie des Partikelsystems. Diese Technik ermöglicht es, die normale Transformationshierarchie des Szenegraphen beizubehalten. Alle Starrkörper verhalten sich beim Update des Szenegraphen gleich, einziger Unterschied ist die angelegte Partikel-Geometrie, die nicht immer eine AABB sein muss. Es stehen dazu Tetraeder, Box und Geosphere zur Verfügung. So wird z. B. ein Fussball in einem der Spiele durch eine Geosphere angenähert.

3.5.2 Gelenke

Kugel- und *Scharnier*-Gelenke werden modelliert, indem mehrere Federn in einem bzw. zwei Massepunkten zusammenlaufen (Abb. 2). Diese Federn sind erstens sehr hart und zweitens haben sie genau die Länge die sich aus der *Anfangshaltung* des Characters ergibt. Durch diese Kombination versuchen die verbundenen Körper zu einem Punkt oder zu zwei Punkten einen festen Abstand zu halten, was der Rotation um einen Punkt mit zwei DOF oder um zwei Punkte mit einem DOF entspricht.

3.5.3 Seile

Die *Fäden* zur Aufhängung der Marionette werden als eine Aneinanderreihung von Einzel-Federn simuliert. Ende und Anfang dieser Verbindung liegen oft auf Partikeln, die zu anderen Meshes

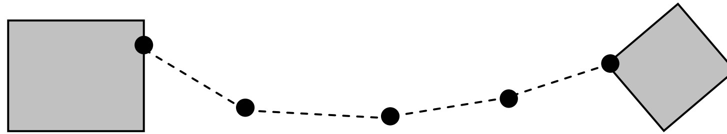


Abbildung 3: Seil

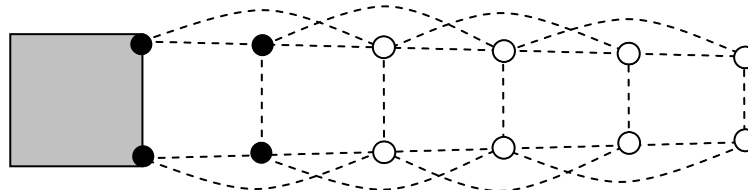


Abbildung 4: Haar-Strähne

gehören, $1/m = 0$-fixiert sind, oder vom Nutzer beeinflusst werden (z. B. das Steuerkreuz). Zur Visualisierung wurde durch die Partikel ein *B-Spline* gelegt, so dass die Auflösung der sichtbaren Geometrie, von der Auflösung der Simulationsgeometrie entkoppelt wird. Dieser Spline wurde weiter als „*Generalised Cylinder*“ visualisiert. Abb. 3 zeigt zwei Körper die mit einem Seil aus vier Segmenten und fünf Partikeln verbunden sind.

3.5.4 Stoff

Zur Darstellung von *Stoff* wird ein Netz aus Federn und Massen verwendet. Dabei werden die üblichen *Sheer*, *Bend* und *Twist*-Federn erzeugt. Einfaches Skinning mit Starrkörpern als Bones erzielte unter unseren Bedingungen jedoch bessere Ergebnisse: So zeigt Skinning zwar keinerlei physikalische Dynamik bei der Animation, es kommt aber auch zu keinen Durchdringungen mit sich selbst oder anderer Geometrie.

3.5.5 Haare

Der Mesh-Typ Hair, realisiert *Haare* die auf beliebigen anderen Körpern wachsen können. Die Haare werden dabei als eine Menge sog. *Strips*, als eine Kette von Rechtecken, dargestellt. Jeder Strip wird für sich erzeugt. Dazu wird ein zufälliger, gleichverteilter Punkt auf einer Oberfläche gewählt von dem der Strip orthogonal wächst. Die spätere Lage der Strips ergibt sich erst durch die Simulation (das Haar *fällt*). Jeder Strip ist N Segmente lang und eine Feder breit. Das erste Segment enthält vier Partikel, die *fixiert* sind und das Haar abstehen lassen. Neben den Federn, die zwischen allen *nächsten* Massepunkte bestehen, existieren vertikal auch Verbindungen zu allen *Übernächsten* (Bögen in Abb 4). Diese Konstruktion verleiht den Strähnen Festigkeit beim Fall und lässt diese (je nach Federkonstante) leicht abstehen. Ausrichtung und Länge der Strähnen sind als „Frisur-Funktion“ über einer Halbkugel modelliert.

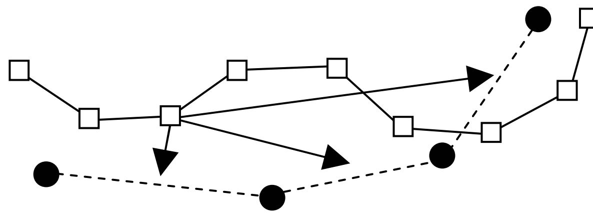


Abbildung 5: Registrierung für ein Proxy mit $N = 3$

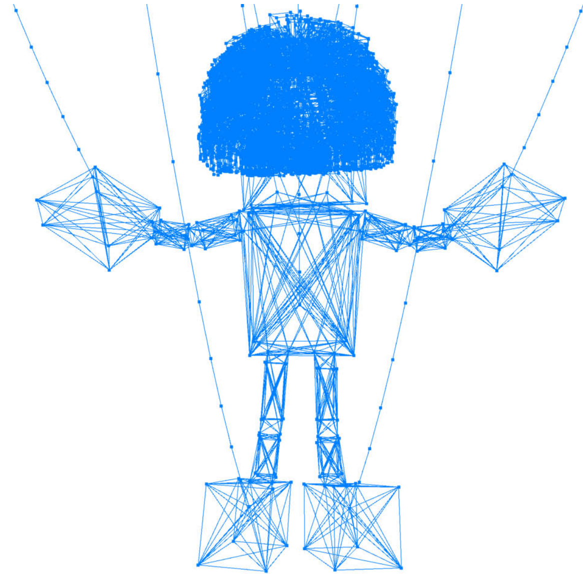


Abbildung 6: Das Feder-Masse-System der Marionette

3.5.6 Deformierbare Objekte / Proxies

Die komplexeste Art Mesh, stellt das *Proxy* dar. Ein Proxy ist ein hoch aufgelöstes Modell (*Highmesh*), dessen Verformung durch ein gering aufgelöstes Partikel-Mesh (*Lowmesh*) bestimmt wird [SK00]. Die Verformung des Highmesh wird dazu als Kombination der Verschiebung der N nächsten Flächen des Lowmesh gebildet. Abb. 5 zeigt die Flächen des High- (Linien) und Lowmeshs (gestrichelte Linie), die Partikel des Lowmesh (Kreise) und einen beispielhaften Vertex des Highmesh (Rechteck). Zu diesem Vertex bestehen N Verbindungen zu Flächen des Lowmesh (Pfeile). Der Einfluss einer Fläche hängt von der mittleren Entfernung des Vertex ab. Zur Anwendung der Verformung wird jeder Vertex beim Startup in die lokalen Koordinatensysteme das die Kanten und die Normale jeder Fläche aufspannen transformiert und zur Laufzeit wieder rücktransformiert. Über die rücktransformierten Vertices wird dann gewichtet summiert. Dieser Vorgang kann als eine Verallgemeinerung des bekannten Skinnings gesehen werden.

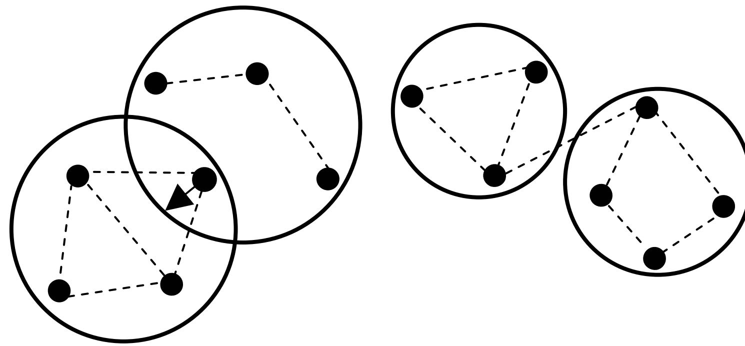


Abbildung 7: Boundingvolumen

3.5.7 Marionette

Abb. 6 zeigt die Marionette und das sie kontrollierende Feder-Masse-System. Die Gliedmassen sind als Boxen dargestellt, zwischen denen die Gelenke als Federn gezogen sind. Es wurden 20 Starrkörper und 15 Gelenke verwendet, die Haare sind in dieser Darstellung durch 500 Strähnen á 4 Segmente modelliert, die Fäden mit 20 Segmenten. Insgesamt ergeben sich 5327 Partikel und 9560 Federn. Auch andere Kombinationen von Meshes für vollkommen andere Puppen mit mehr Gliedmassen, Köpfen, Seilen und Haaren sind hier vorstellbar.

3.6 Constraints

3.6.1 DistanceConstraint

Das *DistanceConstraint* versucht, Punkte in bestimmten Abständen zu halten. Es arbeitet dazu alle Kanten ab, und behandelt diese wie Federn: Ist der Abstand zwischen zwei Punkten die durch eine Kante verbunden sind größer als ein einstellbarer Soll-Abstand, werden die Punkte aufeinander zu bewegt, ist er kleiner, stoßen sich die Punkte ab. Die Stärke dieser Wirkung ist als *Konstante* des *hook'schen Federgesetzes* bekannt und kann ebenfalls pro Feder eingestellt werden. Die Einstellung der Konstante, je nach Verwendung der Feder ist für eine gute Handhabung entscheidend.

3.6.2 Kollision

Auch die *Kollision* ist als Constraint modelliert. Dazu werden bestimmte Teile des Raumes, die analytisch einfach zu beschreiben (*Boundingvolumen*) sind, als „verboten“ für alle Partikel gekennzeichnet. Bewegt sich ein Partikel doch einmal in einen solchen Raumbereich, wird es einfach heraus projiziert. Die Anpassung der Geschwindigkeiten v ergibt sich durch die implizite Form als Differenz der Positionen $v = r - r_{old}$: wenn sich r und r_{old} durch die Projektion nähern geht v gegen 0. Diese Modellierung erlaubt einfache Formen von „*Resting Contact*“, die zwar inexakt sind und nicht für alle Fälle funktionieren, jedoch meistens glaubwürdig erscheinen und z. B. nicht oszillieren. Die beiden implementierten Volumen sind Halbraum und Kugel. Alleine aus der Wirkung der Federn und dem Nicht-Eindringen-Können von Partikel in bestimmte Gebiete resultieren

alle auftretenden Formen von Stoss, Separierung, usw. auch in ihren rotatorischen Komponenten. Für vollkommen starre Körper ergibt diese Technik immer etwas zu weiche Stöße, da ihre Gegenbewegung sich erst durch den ganzen Körper fortsetzen muss. Sind die Zeitschritte allerdings klein (und gerade das erlaubt diese einfache Konstruktion) fällt der Unterschied kaum ins Gewicht.

Das `HalfSpaceCollisionConstraint` kann die Partikel direkt gegen alle Ebenen testen und ggF. projizieren. Es wird z. B. für Wände genutzt. Das `SphereCollisionConstraint` wird für Gliedmassen und Gegenstände in der Welt verwendet. Es geht dazu in jedem Simulationsschritt alle Objekte durch und erzeugt deren Boundingspheres neu, da diese sich i.A. auch verformen können. Danach werden in einer „*Broad Phase*“ alle Boundingspheres gegeneinander getestet. Nur wenn dieser Test positiv ausfällt, werden auch Partikel gegen Kugeln in einer „*Narrow Phase*“ geprüft [Ebe04]. Da die Anzahl der Partikel wesentlich höher als die der Meshes ist, machen diese beiden Phasen Sinn. In einem typischen Fall wird mit 50 Meshes, aber 10000 Partikel gearbeitet. Da die Kollisionserkennung immer noch quadratisch in der Anzahl der Kugeln ist, wird jedes Frame eine rekursive Raumaufteilung aller Kugeln vorgenommen. Dieser Ansatz erreicht $O(nlog(n))$ für die Erkennung der Kugelkollisionen. Andere Ansätze, die temporale Kohärenz ausnützen würden $O(n)$ erreichen, wozu aber noch kein Bedarf bestand [Ebe04]. Abb. 7 zeigt vier verschiedene Meshes und deren Boundingspheres. Es kommt nur zwischen den linken beiden zu einem Kontakt und nur die Partikel dieser Meshes müssen behandelt werden, wobei nur ein Partikel beeinflusst wird (Pfeil).

4 Optimierungen

4.1 SIMD

Die Konzeption des Systems zielt darauf ab, den größten Teil der Rechenzeit in *wenigen, einfachen* Schleifen zu konzentrieren, die dann optimiert werden können. Im unoptimierten Fall liegen über 90 % des Zeit-Aufwands für die Simulation in einigen wenigen Schleifen: dem `DistanceConstraint`, der Integration und der Kollision. Diese wurden dazu mit *SIMD-Operationen* neu implementiert und um Faktoren beschleunigt. Fast alle sind einfach, sequentiell, arbeiten auf $\mathbb{R}^3$-Vektoren und sind damit optimal für eine Umsetzung in z. B. der Intel SIMD-Implementierung *ISSE* geeignet [Int05].

4.2 Beispiel: Spring-Constraint

Eine effiziente Implementierung für Federn wird bei Jacobsen beschrieben [Jak01]. Dort wird eine Annäherung an die hook'schen Federgesetze die ohne Quadratwurzel arbeitet dargestellt. Dabei wird ein weiteres mal die Annahme das die Zeitschritte klein sind entscheidend: bei kleinen Zeitschritten und einem anfänglich ausgeglichenen System ist die Länge der ausgelenkten Federn auch nahe der Ruhelänge. Daraus folgt, dass der benötigte Quotient der beiden auch nahe 1 ist. Und in einer so engen, bekannten Umgebung lässt sich die Quadratwurzel durch eine Taylor-Erweiterung annähern. Die teuerste Operation ist hier nur noch eine Division pro Feder, aber die ISSE-Funktion `rcpss` bildet selbst eine schnelle Annäherung an $1/x$ und zeigte gute Ergebnisse.

Wahrscheinlich ist die Indirektion durch die beiden Indizes noch am teuersten. Eine automatische cache-freundliche Anordnung von Partikeln scheint hier untersuchenswert. Vor allem das matrixähnliche Layout der Cloth-Massepunkte, könnte so verbessert werden[LRW91]. Die mittlere Anzahl der Cycles pro Feder liegt auf einem Intel P4 bei 102.

4.3 Beispiel: Integration

Die Integration besteht wieder nur aus einer einfachen Schleife über alle Partikel. Hier können pro Durchlauf zwei Partikel gleichzeitig bearbeitet werden, um Abhängigkeiten zu reduzieren und die Kapazitäten besser auszunutzen [Int05]. Damit ist Verlet-Integration bei fixem Zeitschritt auch nur Addition, ein Schreiben und dreimal Lesen, und kann für alle Partikel in einer einzelnen Schleife erledigt werden.

4.4 Beispiel: Kollision

Beide Formen von Kollision sind Schleifen über alle Partikel, eine Entscheidung auf welcher Seite sie liegen und ggf. eine Projektion. Für eine Ebene sind pro Partikel ein Skalarprodukt und ein Vergleich nötig. Sollte der Vergleich eine Projektion nötig machen, fallen noch eine Subtraktion und eine Multiplikation an. Für Kugeln sind eine Subtraktion und eine Multiplikation nötig, da die Radien der Kugeln quadriert vorliegen. Eine Projektion ist hier ggf. durch eine Addition, eine Quadratwurzel und eine Division möglich.

5 Bewertung und Ausblick

Diese Arbeit beschrieb ein System zur Simulation virtueller Charaktere und deren Umgebung mittels Feder-Masse-Systemen. Dieses System zeigt im Vergleich zur Verwendung von Starrkörpern bei gleichem Aufwand bessere Ergebnisse und lieferte bei der Anwendung auf eine virtuelle Marionette gute Ergebnisse. Die Perspektive der Objektorientierung spielte dabei für Performanz und Flexibilität eine entscheidende Rolle. Die Nachteile des Systems, vor allem im Bereich der Kollision scheinen weniger auffällig zu sein, was zahlreiche Nutzertest mit vielen verschiedenen Nutzern (z. B. Kindern) gezeigt haben.

Eine Verbesserung der Kollisionserkennung und Behandlung stellt die wichtigste Mögliche Weiterentwicklung dar. Die SIMD-Implementierung könnte durch die Analyse von Cache-Effekten begleitet von systematischem Vermessen der Performanz weiter verbessert werden. Um die Verarbeitungsgeschwindigkeit weiter zu steigern, wäre es denkbar, die Verarbeitung von Partikeln in Pakete aufzuteilen, die mehrere Prozessoren unabhängig abarbeiten, wenn sie schnell genug übetragen werden könnten.

Die Simulation einzelner Erscheinungen mit Hilfe der GPU wurde in der Literatur beschrieben, es ergibt sich jedoch die Frage, wie ein *vereinheitlichtes* System wie das beschriebene, das eine Anzahl verschiedener Erscheinungen auf Feder-Masse-Systeme zurückführt von einer GPU profitieren kann. Dieses System hätte dann auch die Verformung von Proxies, das Verformen der Haar-Geometrie und der Fäden oder das Erzeugen der Matrizen für Starrkörper, neben Integration

und dem Anwenden der Constraints ohne Rückgriff auf die CPU zu verarbeiten. Da in Zukunft mit Architekturen die eine steigende Anzahl von SIMD-Einheiten bieten auszugehen ist [IBM04], wird sich zeigen ob Feder-Masse-Systeme wie zu erwarten gut mit der Anzahl dieser Prozessor-Einheiten skalieren.

6 Danksagungen

Dank gilt allen am Projektpraktikum „Virtuelle Marionette“ beteiligten Studenten der Universität Koblenz-Landau und den betreuenden Mitarbeitern des Labors Computergrafik.

Literatur

[Ebe04] EBERLY, D.H.: *Game Physics*. Morgan Kaufman Publishers, 2004.

[IBM04] IBM CORPORATION: *IBM, Sony, Sony Computer Entertainment Inc. and Toshiba unveil Cell processor*, 2004.

[Int05] INTEL CORPORATION: *IA32 Intel Architecture Software Developers Manual*, 2005.

[Jak01] JAKOBSEN, T.: *Advanced Character Physics*. In: *GDC Proceedings*, 2001.

[Lat04] LATTA, L.: *Building a Million Particle System*. In: *GDC Proceedings*, 2004.

[LRW91] LAM, M. S., E. E. ROTHBERG und M. E. WOLF: *The Cache Performance and Optimizations of Blocked Algorithms*. In: *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 1991.

[MSr05] MOSEGAARD, J. und T. S. SØ RENSEN: *GPU Accelerated Surgical Simulator for Complex Morphology*. In: *Proceedings of the IEEE VR2005*, 2005.

[RVB02] REINERS, D., G. VOSS, und J. BEHR: *OpenSG: Basic concepts*. In: *Proc. 1st OpenSG Symposium*, 2002.

[SK00] SINGH, K. und E. KOKKEVIS: *Skinning Characters using Surface-Oriented Free-Form Deformations*. In: *Proceedings of Graphics Interface 2000*, 2000.

[Ver67] VERLET, L.: *Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules*, 1967.

[WB97] WITKIN, A. und D BARAFF: *Physically Based Modeling: Principles and Practice*. In: *Siggraph 97 course notes*, 1997.