
Universität Ulm

**Hauptseminar: Technik der Spieleprogrammierung**

# Physikalische Simulation von Objekten

Axel Gerstmair

25. April 2002

**Zusammenfassung**

In den heutigen Spielen sind in der Regel alle Interaktionsmöglichkeiten mit der virtuellen Welt eigens programmiert oder basieren auf speziell dafür erstellten Regeln. Daher sind die meisten Gegenstände, die für die Handlung des Spieles nicht essentiell sind, nicht benutzbar und nur als Dekoration gedacht. Dies schränkt das Eintauchen in die virtuelle Welt ein und vermittelt einen künstlichen Eindruck. Dieser Bericht beschreibt zum einen wie einfache Physik, d.h. hauptsächlich die Newtonsche Mechanik, bei der Schaffung realistisch wirkender Welten helfen kann und zum anderen wie Interaktionsmöglichkeiten möglichst einfach modelliert werden können, so dass sehr schnell ziemlich komplexe Interaktionen geschaffen werden können.

# Inhaltsverzeichnis

# 1 Einleitung

*„Ich nahm einen Stein und warf ihn auf das Regal mit dem Porzellangeschirr. Durch die heftige Wucht zersprangen die Teller und Tassen entweder sofort oder fielen auf den Boden und gingen dort zu Bruch. Das schepperte entsetzlich laut. Der metallene Topf blieb heil, aber gab beim Aufprall auf dem Boden ein schweres, dumpfes Dröhnen von sich.“* Etwas in dieser Art könnte einem in einem Computerspiel der nächsten Generationen passieren.

Computerspiele, in deren virtuelle Welt man völlig eintauchen kann, sind zur Zeit jedoch noch rar. Nur allzu häufig wird man daran erinnert, dass es sich nur um ein Spiel handelt. Sei es ein Telefon an der Wand, das man nicht bedienen kann, oder ein Karton, der nicht zu brennen anfängt, obwohl man ihn gerade in ein Lagerfeuer geworfen hat. Zwar gibt es einige Spiele, die sich große Mühe geben, solche Situationen realistisch nachzubilden, jedoch bleibt man als Spieler immer auf einen kleineren Handlungsraum beschränkt. Man findet sich oft in der Situation wieder, durch einen virtuellen Raum zu gehen und zu versuchen, mit allen möglichen Objekten zu interagieren. Das Problem dabei ist, dass in der Regel nur diejenigen Objekte benutzbar sind, die etwas mit der Handlung der Geschichte zu tun haben, während die übrigen nur als Dekoration dienen. Diese Erfahrung kann einen ziemlich schnell frustrieren. Deshalb geht die Entwicklung neuer Spiele sicherlich in die Richtung realistisch simulierter Welten und vielfacher Interaktionsmöglichkeiten, um den Spieler völlig abtauchen zu lassen.

## 1.1 Kleiner geschichtlicher Abriss

Die ersten Spiele in der Ich-Perspektive brachten nur wenige Möglichkeiten mit sich, mit der Umwelt zu interagieren oder sie zu ändern. In der Regel beschränkten sich diese auf das Betätigen von Schaltern, das Öffnen von Türen und das Entriegeln von Schlössern (z.B. Wolfenstein 3D von Apogee/id Software).

*Duke Nukem 3D* (von 3D Realms) war eines der ersten Spiele, in denen man in größerem Maße mit der Umwelt interagieren konnte. Da gab es Lichtschalter, Überwachungskameras, Aufzüge, Toiletten (die man benutzen konnte!) usw. Außerdem konnte sich die Umgebung verändern: meistens wurden durch Explosionen neue Bereiche zugänglich und man konnte selbst bestimmte Wände wegsprengen.


Abbildung 1: Interaktion mit einem Automaten (Screenshot aus Duke Nukem Forever [1]).

Etwas jünger und ähnlich innovativ ist das Spiel *Deus Ex* (von Ion Storm). Besonders neuartig sind hier die Handlungsmöglichkeiten, die man als Spieler in den verschiedensten Spielsituationen hat. Man kann seine Spielfigur weiterentwickeln und mit besonderen Fähigkeiten ausstatten, z.B.

hilft eine ausgeprägte Sprungkraft hohe Hürden zu überwinden, eine ausgesprochene Geschicklichkeit Schlösser zu öffnen in bestimmte Räume zu gelangen und der eher aggressivere Spielertyp kann die Kampfkraft verbessern. Auf diese Weise gibt es immer mehrere Möglichkeiten, derselben Situation zu begegnen, je nach Geschmack und Vorliebe des Spielers.

Einen neuen, größeren Freiheitsgrad versprechen das sich in der Entwicklung befindliche *Duke Nukem Forever* [1] und *Deus Ex 2* [2], die ihre Vorgänger nur als Ausgangspunkt nehmen und hoffentlich neue Maßstäbe für interaktive Computerspiele setzen. Man darf gespannt sein, was uns dort erwarten wird.

Natürlich fehlt in dieser Beschreibung eine lange Reihe weiterer Spiele, die ebenso neuartige Funktionalität und Ideen mit sich brachten. Dies geht jedoch über das Thema dieses Artikels hinaus und die obigen Titel seien nur exemplarisch genannt.

## 1.2 Überblick

Was macht nun eine realistisch wirkende, virtuelle Umgebung aus? Sicherlich ist ein sehr wichtiger Aspekt die Darstellung und Detailliertheit der Welt und Objekte in dieser Welt, z.B. kann man alleine mit Licht- und Schatteneffekten sehr viel zur Atmosphäre beitragen. Hierfür ist eine sogenannte *3D-Engine* zuständig.

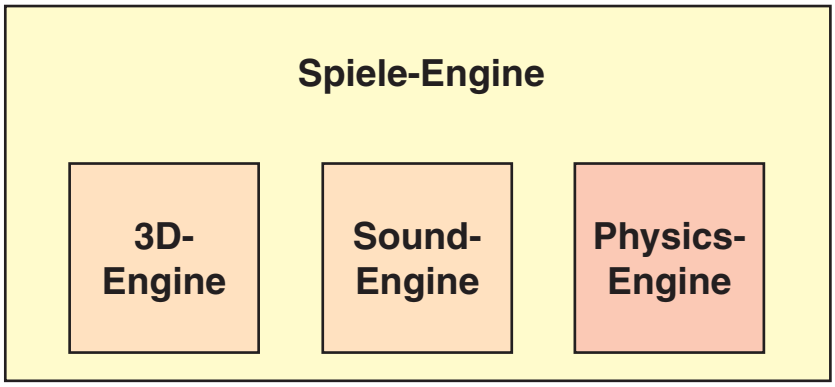


Abbildung 2: Aufbau einer Spiele-Engine.

Eine weitere entscheidende Rolle spielt die Untermalung mit Musik, Soundeffekten und teilweise Sprachausgabe. Es ist doch viel interessanter, wenn ein spannendes Musikstück im Hintergrund spielt, wenn sich die Geräusche der Schritte des Spielers ändern, je nachdem ob er auf einem Teppich, einem Holzboden oder einer Wiese geht, wenn Spielfiguren mit ihm sprechen und sich gegenseitig zurufen. Um diese Effekte kümmert sich eine *Sound-Engine*, die sehr komplex werden kann, wenn sie entsprechende physikalische Gesetze umsetzen will (Wellenlehre, Doppler-Effekt usw.).

Letztendlich sind aber noch zwei weitere Gesichtspunkte für eine fesselnde virtuelle Welt entscheidend. Zum einen sollten sich Gegenstände und Personen gemäß der physikalischen Naturgesetze bewegen und verhalten. Und zum anderen sollte man mit möglichst vielen Gegenständen in der Umgebung etwas anfangen können. Für ersteres ist eine sogenannte *Physics-Engine* verantwortlich. Dieser Begriff ist etwas weit gefasst und könnte vielleicht mit *Motion-Engine* präzisiert werden, jedoch ist der Oberbegriff gebräuchlich. Die nächsten zwei Kapitel werden sich der physikalischen Simulation und der Interaktion von Objekten ausführlicher widmen.

# 2 Physikalische Simulation

Die Bewegung von Figuren und Gegenständen ist einer der wichtigsten Aspekte in einer virtuellen Welt. Damit Bewegungen glaubhaft wirken, muss man eine Schwerkraft einführen, die Objekte auf dem Boden hält oder fallen lässt. Man muss Position, Geschwindigkeit und Beschleunigung von Objekten berücksichtigen und – was insbesondere wichtig ist – man muss entdecken, wann und wo Objekte zusammenstoßen, und entsprechend darauf reagieren. Führt man dann noch die Reibung ein, die z.B. Objekte auch auf einer Schräge noch ruhen lässt, dann hat man bereits die wichtigsten physikalischen Gesetze untergebracht und kann ein vernünftiges Verhalten von Objekten simulieren [3].

## 2.1 Das Spektrum der Größenordnungen

Bei der physikalischen Simulation betrachtet man in der Regel immer Einheiten in einer bestimmten Größenordnung. Beispielsweise betrachtet man in der Medikamentenforschung die Zusammenwirkung von Molekülen. Hier gelten eventuell andere Gesetze oder zumindest wirken sich bestimmte Kräfte stärker bzw. schwächer aus als bei Problemen im täglichen Leben (z.B. bei Aufpralltests von Autos). Das *Spektrum der Größenordnungen* reicht von der subatomaren Ebene bis hin zu astronomischen Maßstäben, wobei jeweils unterschiedliche Aspekte zu beachten sind. Während man z.B. die Schwerkraft bei der Betrachtung von Molekülen und Atomen praktisch vernachlässigen kann, ist sie bei der Simulation von Planetensystemen unverzichtbar. Da Computerspiele normalerweise die wirkliche Welt nachbilden, kommen hier Maße aus dem Alltag, wie Meter, Kilogramm und Sekunde (MKS-System), und entsprechende Kräfte und Gesetze zum Tragen (die sogenannte Newtonsche Mechanik, s. Abschnitt 2.3).

## 2.2 Nachbildung von Chaos

In der realen Welt gibt es sehr viele Vorgänge, die niemals genau identisch ablaufen. Keine zwei Blätter fallen gleich vom Baum, zwei Autos verformen sich unterschiedlich, wenn sie auf eine Wand aufprallen, eine Person fällt jedesmal etwas anders auf den Boden. Auch wenn hinter diesen Abläufen gewisse Muster stecken, so haben sie doch jedesmal eine andere Ausprägung (s.a. Abbildung 3).

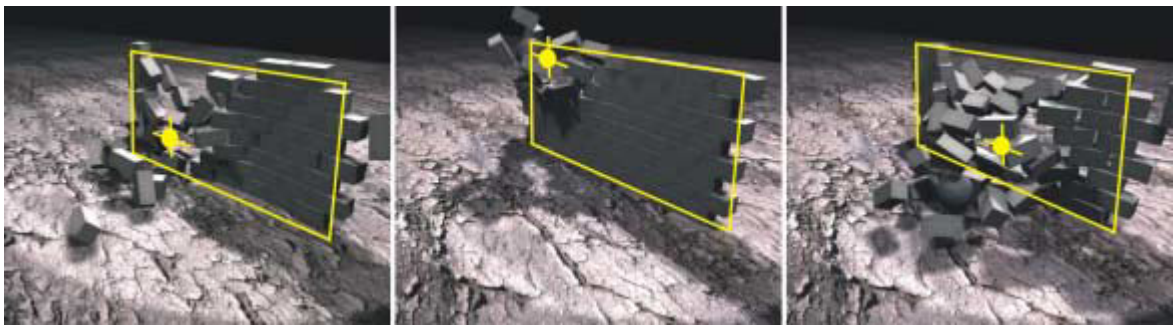

Abbildung 3: Chaotisches Verhalten bei einem Einschlag an einer Wand an unterschiedlichen Stellen (aus [3]).

Da wir dies aus dem Alltag gewohnt sind, erwarten wir dieses Verhalten auch in einer virtuellen Welt. Wendet man hier Tricks an, wie z.B. dieselbe Animation für jedes fallendes Blatt anstatt eines jeweils neu ermittelten Pfades, so werden wir getäuscht und empfinden das als unecht. Diese kleinen „Fehler“ sind uns oftmals gar nicht bewußt, aber wir merken, dass etwas nicht stimmt.

Eine Physik-Engine kann dieses chaotische Verhalten simulieren und so zu einem realistischeren Bild beitragen.

## 2.3 Newtonsche Mechanik

Für die Bewegung von Objekten sind im Prinzip nur drei mechanische Gesetze verantwortlich. Tabelle 1 führt sie auf und erklärt sie kurz. Dabei ist $m$ die Masse eines Objektes, $\vec{a}$ dessen Beschleunigung in eine bestimmte Richtung und $\vec{v}_0$ dessen Anfangsgeschwindigkeit.

| Bezeichnung | Gleichung | Maßeinheit | Beschreibung |
|---|---|---|---|
| Weg | $\vec{s}(t) = \vec{v}_0 t + \frac{1}{2}\vec{a}t^2$ | $m$ | Der von einem Objekt zurückgelegte Weg zu einer bestimmten Zeit. |
| Geschwindigkeit | $\vec{v}(t) = \vec{a}t$ | $\frac{m}{s}$ | Die Geschwindigkeit eines Objektes zu einer bestimmten Zeit. |
| Kraft | $\vec{F} = m\vec{a}$ | $N(= \frac{kgm}{s^2})$ | Die Kraft, die auf ein Objekt wirkt. |

Tabelle 1: Mechanische Grundgesetze.

Die Beschleunigung $\vec{g}$ der Schwerkraft ist eine wichtige Größe, da die Schwerkraft nahezu immer zum Tragen kommt. Der Wert dieser sogenannten Fallbeschleunigung beträgt $9,81\ \frac{m}{s^2}$, aber man kann ihn auch mit $10\ \frac{m}{s^2}$ annähern, um einfacher rechnen zu können.

▷ **Beispiel 1** Ein Würfel mit 1 $m$ Kantenlänge werde aus großer Höhe fallengelassen. Seine Anfangsgeschwindigkeit sei $v_0 = 0\ \frac{m}{s}$ und der Wert von $g$ sei der Einfachheit halber $10\ \frac{m}{s^2}$. Nach einer Sekunde ($t = 1\ s$) hat der Würfel einen Weg $s = 0\ m + \frac{1}{2} \times 10\ m = 5\ m$ zurückgelegt und hat eine Geschwindigkeit $v = 10\ \frac{m}{s^2}$. Nach 4 Sekunden ist der Würfel 80 $m$ gefallen und ist $40\ \frac{m}{s}$ schnell.

In Wirklichkeit kann der Würfel nicht beliebig schnell werden, da der Luftwiderstand der Schwerkraft entgegenwirkt und mit dem Quadrat der Geschwindigkeit zunimmt. Eine einfache Möglichkeit dies umzusetzen ist, einen Maximalwert für die Geschwindigkeit einzuführen, die ein Objekt erreichen kann. Das ist zwar nicht mehr ganz realistisch, aber die Frage ist, merkt der Betrachter den Unterschied überhaupt und kann man die zusätzlich notwendige Rechenleistung entbehren. Dieses Dilemma tritt fortwährend auf und man wird immer versuchen, einen glaubhaften Eindruck zu machen, auch wenn er nicht hundertprozentig den Naturgesetzen entspricht.

### 2.3.1 Implementierungsaspekte

Ein starrer Körper besitzt mehrere für die Simulation wichtige Eigenschaften [5]. Die folgenden können sich im Verlauf der Zeit ändern:

- Positionsvektor eines Referenzpunktes
- lineare Geschwindigkeit des Referenzpunktes (ein Vektor)
- Ausrichtung des Körpers (z.B. eine $3 \times 3$ Rotationsmatrix)
- Vektor der Winkelgeschwindigkeit, der beschreibt, wie sich die Ausrichtung des Körpers über die Zeit hinweg ändert

Folgende Eigenschaften bleiben konstant:

- Masse des Körpers

- Position des Schwerpunkts (in Bezug zum Referenzpunkt)
- Trägheitsmatrix, die die Massenverteilung des Körpers um den Schwerpunkt beschreibt (die Trägheitsmomente um jede der drei Achsen)

Bevor ein Körper bewegt wird, werden alle Kräfte aufsummiert, die auf ihn wirken (bspw. die Schwerkraft, die Kraft beim Abschuss einer Rakete und der Wind). Auf diese Weise erhält man eine resultierende Kraft, mittels der dann eine Beschleunigung ermittelt (s. Tabelle 1) und schließlich der Körper bewegt werden kann.

Eine Physik-Engine bietet in der Regel die Möglichkeit, Objekte zu deaktivieren (und später wieder zu aktivieren). Ein deaktiviertes Objekt wird in der weiteren Simulation nicht weiter berücksichtigt. Das ist insofern sinnvoll, da die Simulation aller Objekte in einer Umgebung sehr rechenintensiv werden kann und sich viele Objekte gar nicht auf das Simulationsergebnis auswirken. Vor allem ruhende Körper bieten sich für diese Art der Optimierung an.

### 2.3.2 Verbindungsstücke

Ein Körper kann mit einem anderen Körper durch ein *Verbindungsstück* („joint“ oder auch „constraint“) verbunden sein. Abbildung 4 zeigt drei unterschiedliche Arten von Verbindungsstücken.

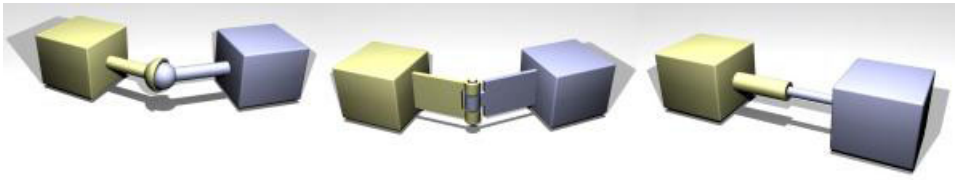

Abbildung 4: Unterschiedliche Verbindungen zwischen Objekten (ein Kugelgelenk, ein Scharnier und ein Kolben mit Gelenkpfanne).

Durch die Verbindungen sind die Objekte in bestimmter Weise voneinander abhängig. So können sich Kräfte übertragen, aufheben oder ändern je nachdem wieviele Freiheitsgrade das Verbindungsstück bietet. Mehrere Parameter, wie z.B. der Verbindungspunkt oder die beweglichen Achsen, beschreiben jede Verbindung. Abbildung 5 zeigt zwei Anwendungsbeispiele für Verbindungsstücke.

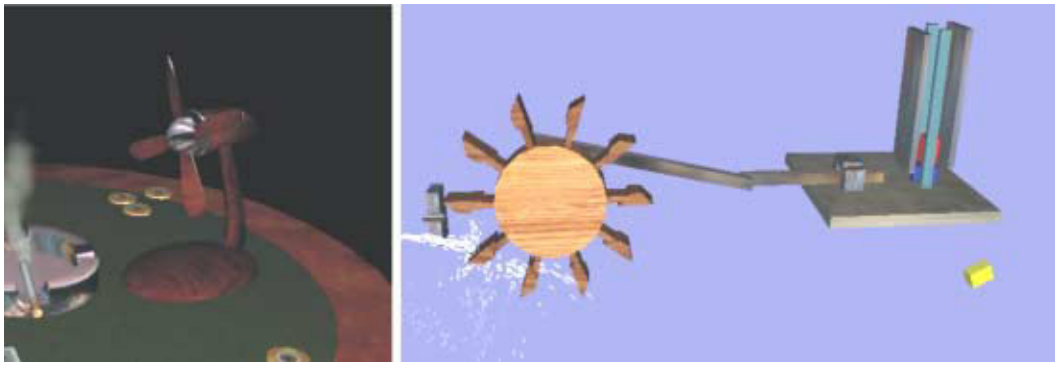

Abbildung 5: *Linkes Bild:* die Propellorblätter sind mit einer Drehachse mit dem Ventilator verbunden. *Rechtes Bild:* Wasser treibt über eine Reihe von Verbindungen einen Block-Schiebe-Mechanismus an.

## 2.4 Kollisionen

Wenn zwei Körper in der Simulation zusammenstoßen, nennt man das eine *Kollision*. Kollisionen treten fortwährend auf, z.B. wenn ein Spieler gegen eine Wand läuft oder von einer Mauer herunterhüpft und auf dem Boden landet. Ihre Erkennung ist deshalb ein sehr wichtiger Bestandteil von Physik-Engines.

In der Simulation werden jeweils immer zwei Objekte auf eine Kollision hin untersucht (sogenannte Kollisionspaare). Dabei werden Teile der Drahtgittermodelle der betreffenden Objekte Stück für Stück verglichen. Je feiner die Objekte strukturiert sind (d.h. je höher die Anzahl ihrer Polygone ist), desto aufwändiger wird ein Kollisionstest. Deshalb verwendet man für diese Tests normalerweise nicht die ursprünglichen 3D-Modelle, sondern einen stellvertretenden Umriss (*bounding box*, s. Abbildung 6).

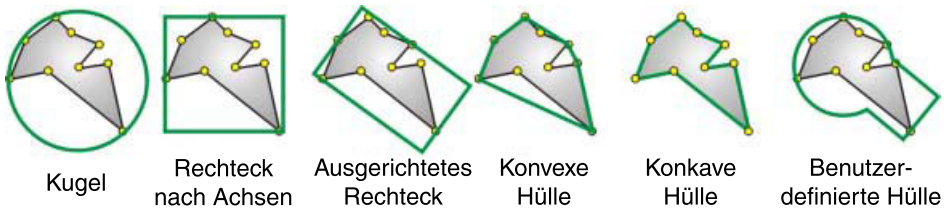


Abbildung 6: Unterschiedliche Objektumrisse.

Der Aufwand für Kollisionstests ist sehr hoch: ist $n$ die Anzahl der zu untersuchenden Objekte, so beträgt der Aufwand $\frac{n(n-1)}{2}$ und fällt in die Klasse $O(n^2)$. Das Problem ist, dass bereits die einzelnen Tests sehr aufwändig sind und der quadratische Aufwand dann schnell spürbar wird.

Etwas Abhilfe können mehrstufige Tests mit unterschiedlicher Genauigkeit schaffen. Wählt man als erstes einen groben Umriss (z.B. einen Würfel) und überprüft dann, ob zwei Objekte kollidieren, so kann man sich weitere aufwändigere Berechnungen sparen, wenn dieser Test bereits fehlschlägt und keine Kollision anzeigt. Ansonsten muss man mit genaueren Umrissen fortfahren, bis man Gewissheit hat.

Bisher sind wir von starren Körpern ausgegangen. Viel schwieriger gestalten sich Kollisionen bei verformbaren Körpern, denn ihre Gestalt kann sich sehr schnell über die Zeit hinweg ändern, was bestimmte Optimierungen zunichte macht, und sie können mit sich selbst zusammenstoßen.

### 2.4.1 Durchdringungen von Objekten

Eine Sonderform von Kollisionen sind Durchdringungen von Objekten (*interpenetration*). Beispielsweise kann eine Kugel, die mit sehr hoher Geschwindigkeit fliegt, eine dünne Wand durchdringen, egal aus welchem Material sie besteht. Das Problem ist, dass die Position eines Objektes (und damit auch mögliche Kollisionen) nur zu diskreten Zeitpunkten und nicht kontinuierlich bestimmt werden kann. So kann es nun passieren, dass ein Objekt zwischen zwei Simulationsschritten in ein anderes Objekte eindringt oder es völlig durchstößt noch vor der nächsten Kollisionsbestimmung.

Hat ein Objekt ein anderes erst einmal durchdrungen, ist es bereits zu spät und man kann nicht mehr viel machen. Man kann solche Fehler ab und zu in Spielen beobachten, z.B. wie eine Türe durch die Hälfte einer Kiste hindurch zugeht. Hier gilt es also, Vorkehrungen zu treffen, um diese Situationen möglichst zu vermeiden. Eine Möglichkeit ist, die Zeitschritte der Simulation zu verkleinern, oder man verzichtet auf sich schnell bewegende Objekte und dünne Wände.

## 2.5 Die Simulationsschleife

Nun können wir den Aufbau der physikalischen Simulation beschreiben. Die Simulation besteht im Grunde aus einer Schleife, die fortwährend wiederholt wird. Nachdem der Anfangszustand zum Zeitpunkt $t = 0$ bestimmt worden ist, wiederholen sich folgende Aktionen:

1. Kollisionen bestimmen
2. Kräfte aktualisieren
3. Neue Objektzustände ermitteln
4. Zeit erhöhen

Kollisionen bringen neue Kräfte mit sich (z.B. Reibung) und wirken sich auf den zweiten Punkt aus. Im zweiten Schritt werden alle Kräfte aktualisiert, die auf Objekte wirken. An dieser Stelle werden auch Benutzereingaben und die Aktionen anderer Wesen (entities) im Spiel berücksichtigt.

Nun werden alle Kräfte, die auf ein Objekt wirken, aufsummiert und das Ergebnis verwendet, um seinen neuen Zustand zu bestimmen. Mit dieser Information wird dann auch die 3D-Ansicht (und andere Teile des Spiels) aktualisiert. Als letztes wird die Zeit um ein bestimmtes Zeitintervall erhöht.

## 2.6 Genauigkeit der Simulation

Wie bereits im Abschnitt 2.4.1 angedeutet, wirkt sich die Länge der Zeitintervalle zwischen den einzelnen Simulationsschritten auf das Simulationsergebnis aus. Abbildung 7 zeigt ein weiteres Beispiel.

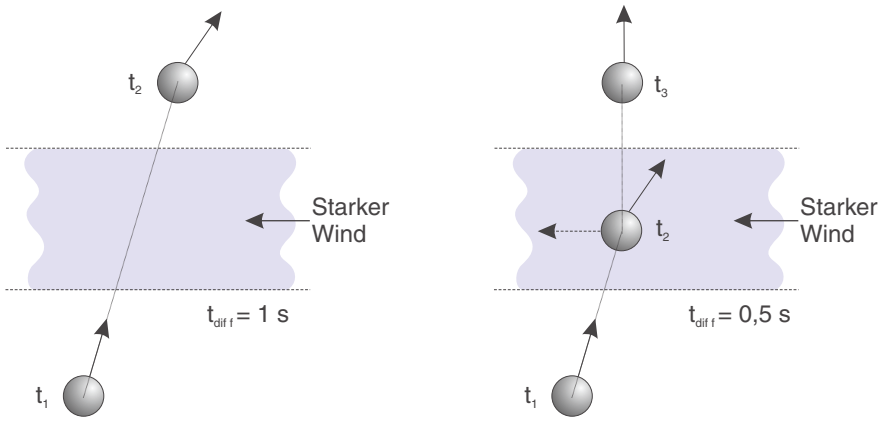


Abbildung 7: Auswirkung unterschiedlicher Zeitintervalle auf das Simulationergebnis.

▷ **Beispiel 2** Eine Kugel werde auf einer parabolischen Kurve in die Luft geworfen (der Luftwiderstand sei dabei zu vernachlässigen). Entgegen der Flugrichtung der Kugel wehe in einem sehr schmalen Abschnitt ein starker Wind.

Einmal werde der Wurf mit einem Zeitintervall von $t_{diff} = 1\ s$ simuliert (linke Seite der Abbildung). Die Kugel bewegt sich hierbei auf ihrer normalen Flugbahn unbeeinträchtigt von dem Wind, da sie sich im ersten Simulationsschritt $t_1$ *unter* dem Windkanal und im nächsten Simulationsschritt $t_2$ *über* dem Windkanal befindet. Zu keinem Zeitpunkt wird also die Kraft des Windes mitberücksichtigt.

Werden die Zeitintervalle dagegen auf $t_{diff} = \frac{1}{2}\ s$ halbiert (rechte Seite), so kann im zweiten Simulationsschritt der Wind berücksichtigt und die Flugbahn der Kugel verändert werden, so dass sie zunächst einmal senkrecht nach oben weiterfliegt.

Wie man hier sieht, sollte beim Entwurf einer virtuellen Umgebung eine bestimmte Zeitdauer eingestellt und dann festgehalten werden. Im Gegensatz zur Grafik können keine Details eingespart werden, um Prozessorarbeit zu reduzieren, da dadurch die Simulationsergebnisse verfälscht würden.

### 2.6.1 Differentialgleichungen

Der bisherige Ansatz zur physikalischen Simulation ging von diskreten Zeitschritten aus, zu denen alle notwendigen Objektzustände neu berechnet werden. Bei dieser Methode entsteht in jedem Zeitschritt ein geringer Rechenfehler, da diese Art der Simulation nur eine mathematische Annäherung ist. Dieser Rechenfehler kann sich im Laufe der Zeit aufsummieren und schließlich derart anwachsen kann, dass er zu falschen Ergebnissen führt.

Will man mit genauen Werten rechnen, so muss man unweigerlich auf Differentialgleichungen zurückgreifen. Dabei gibt man eine Rechengenauigkeit vor und rechnet solange, bis diese erreicht ist. Bei der genauen Rechnung verfolgt man außerdem einen anderen Ansatz: man ermittelt neue Kräfte und Beschleunigungen immer zu dem Zeitpunkt, zu dem sie auftreten, und erhält dann genaue Werte für den berechneten Zeitraum. Bei der visuellen Darstellung (die wiederum zu diskreten Zeitpunkten stattfindet) greift man dann auf die vorausberechneten, genauen Werte zurück.

So schön dieser Ansatz klingt, so wenig Sinn macht es, ihn in heutigen Spielen umzusetzen. Differentialgleichungen haben den gravierenden Nachteil, dass sie enorm aufwändig zu lösen sind. Der Rechenaufwand ist so hoch, dass mit heutigen Systemen keine Echtzeitsimulation möglich wäre. Allerdings ist hier wiederum der praktische Nutzen für ein Spiel fraglich, denn die Entwickler werden mit allerlei Tricks sicherlich genauso glaubhafte Simulationen schaffen können.

## 2.7 Physik-Engines

So wie Spiele-Entwickler heute auf fertige 3D- und Sound-Engines zurückgreifen können, können sie mittlerweile auch komplette Physik-Engines lizensieren und verwenden. Dies kann sich insofern lohnen, da der Entwicklungsaufwand für eine Spiele-Engine so hoch sein kann wie die Entwicklung des Spiels selbst. Außerdem sind die lizensierbaren Engines oftmals ausgereifter und besser dokumentiert als eine Eigenentwicklung. Auf der anderen Seite kann die Integration einer fremden Komponente in ein eigenes Projekt Schwierigkeiten bereiten, weil im Normalfall nicht alles nach den eigenen Vorstellungen funktioniert. Dennoch wird es für viele Projekte unverzichtbar werden, eine bestehende Physik-Engine oder gar komplette Spiele-Engine zu verwenden, da einfach das für eine Neuentwicklung notwendige Geld und die Zeit fehlen.

Nachfolgend seien einige Physik-Engines genannt:

- *Havok Physics Engine.* Eine kommerzielle Engine, mit vielen Besonderheiten, lizensiert von Valve, Ion Storm (für Deus Ex 2, Thief 3), Blizzard (für Cinematics) und vielen mehr [3].
- *MathEngine Karma.* Eine kommerzielle Engine, die erst kürzlich in die bekannte Unreal Engine integriert wurde [4].
- *Open Dynamics Engine (ODE).* Eine freie Engine, die der Doktorand Russell Smith entwickelt [5].

- *Newtonian Object Oriented Physics Engine (NOOPE).* Eine freie Engine in Java [6].

# 3 Interaktion von Objekten

Die Interaktion mit der Umwelt ist eines der wichtigsten Elemente in einem Computerspiel. Sie macht letztendlich auch den Unterschied zu einem Film aus. Mal abgesehen von der Bewegung, die in der Regel eine Grundvoraussetzung für ein Spiel ist, hat der Spieler unterschiedliche Möglichkeiten, seine Umwelt zu beeinflussen und eine Rückkopplung zu erhalten. Je mehr Gegenstände der Spieler handhaben kann, je mehr Objekte ihr natürliches Verhalten aufweisen, je mehr Wesen vorkommen, mit denen der Spieler sich unterhalten kann, desto echter wirkt die virtuelle Welt und umso leichter fällt es dem Spieler, in die künstliche Welt einzutauchen.

## 3.1 Formen von Interaktion

Man kann verschiedene Formen der Interaktion unterscheiden. Sowohl der Akteur als auch das betreffende Objekt kann eine Person oder ein Gegenstand sein:

- *Person zu Person (Kommunikation).* Diese Form der Interaktion kann ein Gespräch sein, ein Geräusch, ein Winken oder anderes Zeichen. Die Kommunikation ist ein Thema für sich und hat ihre eigene Problematik (bis hin zur Spracherkennung/-synthese).
- *Person zu Gegenstand (und umgekehrt).* Der Spieler isst einen Müsli-Riegel, ein Nachtwächter hört beim Betreten des Foyers die Alarmsirene oder ein Fass rollt über eine Kakerlake.
- *Gegenstand zu Gegenstand.* Eine Billiardkugel, die auf eine andere auftrifft, ein Karton, der in ein Feuer fällt, oder eine Pistolenkugel, die in eine Glasscheibe einschlägt.

Gegenstände und Personen unterscheiden sich darin, dass Personen Ziele, Absichten, Motivationen, ein Gedächtnis und möglicherweise eine Persönlichkeit (d.h. bestimmte Charaktermerkmale) haben. Dementsprechend reagieren Personen anders auf ein Ereignis als ein (lebloser) Gegenstand. Personen können situationsbezogen handeln, während Gegenstände ein vorhersehbares Verhalten aufweisen. Beispielsweise kann ein Krieger, der verwundet wurde, seinen Kameraden mutig beistehen; er flieht jedoch, wenn er alleine ist und es niemanden zu beschützen gibt.

## 3.2 Herkömmliche Verfahren

Bisher werden Interaktionen in Spielen entweder fest einprogrammiert oder durch Skripte[1] umgesetzt [2]. Je nach Ausdrucksmöglichkeit der Anwendung müssen Interaktionen ganz konkret implementiert werden oder können allgemeiner gefasst werden.

▷ **Beispiel 3**

```
Wenn der Spieler mit einer Axt in diese Glasscheibe schlägt,
dann zerbricht sie und wird entfernt.
```

Oder eine etwas allgemeinere Variante:

```
Wenn der Spieler dieser Glasscheibe 25 oder mehr Schadenspunkte zufügt,
dann zerbricht sie und wird entfernt.
```

[1] Eine Skriptsprache ist eine vereinfachte Programmiersprache, die normalerweise begrenzten Zugriff auf ganz bestimmte Komponenten einer Anwendung erlaubt und einfache, aber sehr mächtige Werkzeuge, d.h. Befehle und Programmkonstrukte, zur Verfügung stellt.

Bei diesem Lösungsansatz wird jeweils eine direkte Beziehung zwischen Akteur und betreffendem Objekt eingeführt. Er hat den Vorteil, dass er relativ einfach umzusetzen ist und die Leveldesigner ganz kontrolliert Interaktionen einbauen können. Es können praktisch keine unerwünschten Seiteneffekte entstehen, da jede Interaktion geplant und gezielt implementiert wird. Aber genau darin besteht auch der große Nachteil dieser Methode: der Aufwand, eine Umgebung mit vielen Handlungsmöglichkeiten zu schaffen, ist sehr hoch. Hinzu kommt, dass der Designer möglichst viele Interaktionsmöglichkeiten vorhersehen muss, damit er sie einbauen kann und sie dem Spieler letztendlich auch zur Verfügung stehen. Das ist sicherlich auch ein Grund, warum die meisten Spiele bis heute nur einen begrenzten Handlungsraum zur Verfügung stellen.

## 3.3 Ein anderer Ansatz

Es gibt einen Ansatz, der eine Vereinfachung für die Leveldesigner verspricht: die Entkopplung von Quelle und Ziel einer Aktion [2]. Das bedeutet, dass man zwischen einzelnen Objekten keine direkte Beziehung mehr herstellt, sondern zwischen ihnen vermittelt. Im obigen Beispiel könnte man folgende Bedingungen einführen:

```
Eine Axt verursacht 25 Schadenspunkte.

Wenn diese Glasscheibe 25 oder mehr Schadenspunkte erleidet,
dann zerbricht sie und wird entfernt.
```

Der Unterschied besteht darin, dass die Aktion (der Spieler schwingt die Axt) nicht das Ziel (die Glasscheibe) und die Wirkung kennen muss (sie zerbricht). Vielmehr geht von jeder Interaktion ein Impuls aus, auf den die Umwelt reagieren kann (aber nicht unbedingt reagieren muss). Die Glasscheibe „wartet“ also mehr oder weniger darauf, dass sie durch irgendeinen Impuls *25 oder mehr Schadenspunkte erleidet*, und reagiert dann entsprechend darauf.

Diese Methode hat den entscheidenden Vorteil, dass sie sehr flexibel ist und Interaktionen sehr leicht verallgemeinert werden können. Bspw. zerbreche ein Porzellantopf ab 15 Schadenspunkten; dann würde dieser Topf zu Bruch gehen, wenn der Spieler mit seiner Axt darauf einschlägt. Man müsste keine weitere Beziehung zwischen dem Topf und der Axt einführen. Der Leveldesigner kann also die Akteure unabhängig von den betreffenden Objekten erweitern und umgekehrt. So wäre es ein leichtes, in obigem Beispiel noch eine Explosion hinzuzufügen, die 30 Schadenspunkte verursacht und aufgrund dessen sowohl die Glasscheibe als auch den Topf zerstören kann.

### 3.3.1 Weitere Verbesserungen

Der nächste Schritt besteht darin, die Verhaltensweisen und Impulse weiter zu verallgemeinern. Eine Möglichkeit wäre, Attribute einzuführen und diese den Gegenständen zuzuweisen („zerbricht ab x Schadenspunkten“, „entflammbar“ oder „flieht bei lautem Geräusch“), d.h. man definiert Verhaltensmuster und weist sie den Objekten zu.

Auf der anderen Seite führt man parametrisierte Impulse ein und legt fest, wodurch sie ausgelöst werden, z.B. könnte man sagen, eine Axt feuert den Impuls *Schaden* mit den Parametern *Schadenspunkte = 25* und *Schadensform = Klinge*.

Auf diese Weise erreicht man einen hohen Grad der Wiederverwendbarkeit und kann ziemlich schnell eine interaktive Welt schaffen, sobald die grundlegenden Verhaltensmuster definiert sind. Dieser Ansatz hat jedoch auch einen Nachteil. Durch die Entkopplung von Akteuren und Objekten und eine große Zahl von Impulsen und Verhaltensweisen können die Interaktionsmöglichkeiten unübersichtlich werden und es wird schwieriger, einen bestimmten Spielablauf zu garantieren. Es wird immer einen findigen Spieler geben, der eine neue, unvorhergesehene Möglichkeit entdeckt, eine Situation zu umgehen.

### 3.3.2 Implementierungsaspekte

Die Entkopplung von Akteuren und betreffenden Objekten kann man relativ einfach umsetzen. Dieser Ansatz entspricht einem Ereignismodell, wie z.B. dem Event-Listener-Modell von Java, und basiert auf dem *Observer*-Muster [7]. Der Gegenstand, mit dem etwas geschehen soll, ist dabei der Beobachter und wartet auf ein bestimmtes Ereignis. Die Ereignisse (d.h. Impulse) werden von unterschiedlicher Stelle aus erzeugt, z.B. direkt vom Spieler (betätigt Schalter), von der Physik-Engine (Kollision) oder als Folge eines anderen Ereignisses (Splitter verursachen Schaden beim Zerplatzen einer Scheibe).

Man kann das Modell noch insofern erweitern, dass man zwischen Sender und Empfänger von Ereignissen einen Vermittler einführt (*Mediator*-Muster). Das hat den Vorteil, dass sich die Empfänger von Ereignissen nur bei diesem einen Vermittler anmelden müssen und ihm mitteilen, auf welche Ereignisse sie reagieren wollen. Die Impulse von Aktionen werden dann an den Vermittler geschickt, der wiederum die Botschaft an alle angemeldeten Objekte weiterleitet, die sich für sie interessieren (s. Abbildung 8).

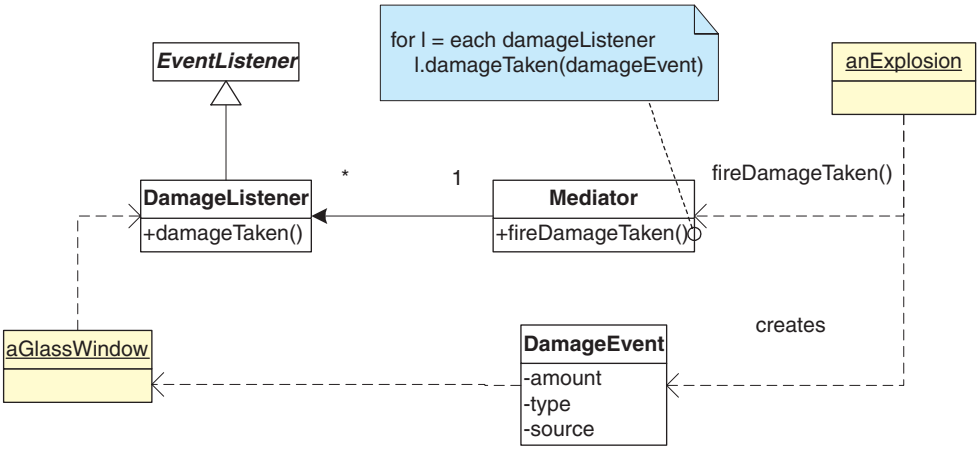


Abbildung 8: Beispiel für ein Ereignismodell.

# 4 Ausblick

Nahezu alle aktuellen Spiele unterstützen heutzutage eine grundlegende Physik. Die Erkennung von Kollisionen ist unabdingbar und eine Form von Schwerkraft und Reibung gibt es normalerweise auch. Neuartig sind dagegen die Verbindungsstücke und Constraints, die sich auf ganze Gruppen von Objekten auswirken. Sie müssen sich erst noch durchsetzen und verbreiten.

Eine andere Einschränkung sind momentan noch die starren Körper. Praktisch nirgends verändern Körper ihre Form oder sieht man gar Stoffe oder Flüssigkeiten. Da solche dynamischen Objekte sehr rechenaufwändig sind, wird es wohl noch einige Zeit dauern, bis sie Verbreitung unter den Computerspielen finden. Allerdings lassen die Fortschritte in der Filmindustrie darauf hoffen.

Auch die Interaktivität mit der virtuellen Welt dürfte mit den nächsten Spiele-Generationen stetig zunehmen. Der Trend geht hier zu immer komplexeren Umgebungen, jedoch besteht hier noch großer Aufholbedarf.

Generell stecken heutige Spiele noch in den Kinderschuhen, was die physikalische Simulation und die Interaktivität angeht. Das liegt sicherlich zum einen daran, dass bisher sehr viel mehr Wert auf gute Grafik und guten Sound gelegt wurde. Zum anderen wird erst langsam Rechenleistung

für andere Zwecke frei, weil der zentrale Prozessor noch genug mit der Darstellung zu tun hat und erst allmählich von Grafik- und Soundchips entlastet wird. Ein Aspekt ist außerdem, dass viele Hersteller auf fertige Spiele-Engines zurückgreifen, die selbst noch nicht diese Neuerungen bieten und den Einbau eigener Erweiterungen erschweren.

Alles in allem jedoch werden die Spiele von Generation zu Generation immer detaillierter und realistischer. Was heute in Filmen wie *Ice Age* oder *Shrek* zu sehen ist, kann man vielleicht schon in den Spielen von morgen selbst erleben.

## Literatur

[1] 3D Realms on the web: *Duke Nukem Forever*. http://www.3drealms.com/duke4/.

[2] Smith, H. on the web: *The Future of Game Design: Moving Beyond Deus Ex and Other Dated Paradigms*. http://www.igda.org/Endeavors/Articles/hsmith_printable.htm.

[3] Havok physics engine on the web: *Havok Physics Primer*. http://www.havok.com.

[4] MathEngine on the web: *Karma and Unreal combine*. http://www.mathengine.com/_mathengine_corp/_news/releases/unreal.html.

[5] ODE on the web: *Open Dynamics Engine*. http://www.q12.org/ode/ode.html

[6] Noope on the web: *Newtonian Object Oriented Physics Engine*. http://www.srcf.ucam.org/nope/

[7] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1994: *Design Patterns: Elements of reusable object-oriented software*. Addison Wesley.