

Physikengines

State of the art, Konzepte

Bertolt Schmidt, Martin Bader

11. November 2002



Zusammenfassung

In diesem Dokument wird der “State of the art” bei Physikengines beschrieben. Dabei wird auf die Möglichkeiten der einzelnen Physikengines (sowohl kommerzielle als auch Open Source Engines) eingegangen und auf Punkte die bei der Integration von Physikengines in eine Gameengine beachtet werden müssen, bzw. welche Möglichkeiten sich dadurch bieten. Schließlich werden einzelne Techniken, die in Physikengines verwendet werden, beschrieben.

Inhaltsverzeichnis

1	Überblick über Physikengines/-simulationen	3
1.1	Wissenschaftliche Physiksimulationen	3
1.2	Kommerzielle Spielephysikengines	3
1.2.1	Havok	3
1.2.2	Mathengine/Karma	4
1.2.3	CM-Labs: Vortex	5
1.3	Open Source Spielephysikengines	5
1.3.1	ODE: Open Dynamics Engine	5
1.3.2	DynaMechs (Dynamics of Mechanisms)	6
2	Integration einer kommerziellen Physikengine	6
3	Angewandte Techniken	8
3.1	Kollisionserkennung	8
3.1.1	Bounding Spheres	8
3.1.2	Bounding Boxes	8
3.1.3	Binary Space Partitioning	9
3.1.4	Kollisionserkennung auf konvexen und konkaven Objekten	9
3.2	Rigid body dynamics	9
3.2.1	“Penalty-method” / Feder- und Dämpfersysteme	9
3.2.2	Lagrange multiplier	10
3.2.3	Featherstone Methode	10
3.2.4	Impulsmethoden	10

1 Überblick über Physikengines/-simulationen

1.1 Wissenschaftliche Physiksimulationen

Für wissenschaftliche physikalische Simulationen existiert Spezialsoftware, die diese speziellen Modelle unter Nebenbedingungen mit bestimmter Genauigkeit berechnen können. Als Beispiel sind hier auch Klimasimulationen (z.B. der Earth Simulator in Japan) oder Wettersimulationen zur Wettervorhersage zu nennen. Diese Anwendungen benötigen im Allgemeinen keine Echtzeitberechnung bzw. haben viel zu hohe Hardwareanforderungen um in Spielen verwendet werden zu können. Im Folgenden werden wir deshalb nicht auf diese Art von Physikengines eingehen.

1.2 Kommerzielle Spielephysikengines

Derzeit gibt es auf dem Markt einige kommerzielle Spielephysikengines, die man in seinen eigenen Spieleprojekten integrieren kann. Dadurch läßt sich die Entwicklungszeit einer Gameengine verkürzen, da man sich nicht mehr mit den Berechnungsmodellen für eine Physik und deren Optimierungen beschäftigen muss. Diese Physikengines sind meistens so allgemein gehalten, dass sie in diverse Arten von Spielen integriert werden können. So ist es möglich eine Physikengine z.B. sowohl in einen Flugsimulator als auch in einem Rollenspiel einzusetzen, obwohl die Anforderungen an die Spielphysik grundverschieden sind.

1.2.1 Havok

Die Firma Havok wurde 1998 gegründet und brachte 1999 die erste Version ihrer Physikengine auf den Markt. Havok bietet heute eine ganze Reihe von Produkten an, die auf ihrer Physikengine (oder Teilen davon) basieren. Hier eine Auswahl:

Havok Game Dynamics SDK [4]: Ist für Spieleentwickler gedacht. Stellt die komplette Physikengine dar:

- **Kollisionserkennung:** Unterstützt die Erkennung von Kollisionen von verschiedensten Objekttypen, z.B. Kollision zwischen konkaven Objekten
- **Rigid body response:** Korrektes Verhalten bei Kollisionen von Körpern mit fester Polygonhülle, mit gegebenen physikalischen Eigenschaften (wie Masse, Reibung, ...)
- **Stoff-/Kleideranimation**
- **Auto-/Fahrzeugsimulation:** Über 100 Fahrzeugparameter: Aerodynamik, Fahrverhalten, Beschleunigung, ...
- **Constraints:** Die Engine kennt diverse Arten von Verbindungen zwischen Objekten wie z.B. "Punkt-zu-Punkt"-Verbindungen, Federsysteme, Achsen, ...
- **Sensors, events, callback:** "Triggersystem" zum Erkennen, Auslösen und Behandeln von Ereignissen

- **Rapid prototyping layer:** Schnelle Entwicklung eines Prototypens mit eingeschränkter Funktionalität und Geschwindigkeit
- **Debugging und profiling support:** Diverse Werkzeuge zur Fehlersuche und für Optimierungen.
- **Exporter** für 3D Studio max, Maya,...

Reactor ist die “Physikerweiterung” von 3D Studio max

Havok Xtra ist die Standard Physikengine für Macromedia’s Shockwave 3D.

Die Physikengine ist auf folgenden Plattformen verfügbar:

- PlayStation 2
- GameCube
- Xbox
- PC

Sie wird in folgenden Produkten verwendet bzw. soll verwendet werden [3]:

- “Deus Ex 2” und “Thief 3” von Ion Storm (action adventure)
- “Freelancer” von Microsoft Games (space combat)
- “Starcraft: Ghost” von Nihilistic (action adventure)
- “Asylum” von Dark Black (first person horror game)
- jede Menge Rennspiele: z.B. “Megarace 3” von Dreamcatcher

1.2.2 Mathengine/Karma

Karma von Mathengine [6] ist eine weitere kommerzielle Spielphysikengine.

Features:

- Für viele Plattformen mit jeweiligen Optimierungen und Lösungen verfügbar
- Constraints: vielfältige Möglichkeiten zu Verbindungen und Nebenbedingungen
- Schnelle und flexible Kollisionserkennung zwischen einzelnen Objekten und mit dem Terrain
- Integriertes “physics authoring tool”

Karma wird in folgenden Spielen eingesetzt:

- **Unreal Engine/ UT2003** von Epic
- **Black & White** von Lionhead in der Konsolenversion
- **PlanetSide** von Sony Online: Ein Massively multi-player First Person Shooter

1.2.3 CM-Labs: Vortex

Vortex von CM-labs [5] basiert ursprünglich auf Mathengine/Karma, wurde dann ausgliedert und eigenständig weiterentwickelt. Vortex ist eigentlich für die Simulation von physikbasierten Bewegungen gedacht.

Von CM-Labs ist sie für folgende Anwendungsgebiete gedacht:

- Simulation von Fahrzeugen auf unebenem Gelände
- force feedback Mechanismen
- Design von Robotersystemen
- Aufbau von virtuellen Welten

Vortex ist nicht unbedingt für die Integration in eine Gameengine gedacht. Im Gegensatz zu Karma, das auf Geschwindigkeit innerhalb von Spielen auch auf “schwächeren” Rechnern gedacht ist, ist Vortex eher für den Ingenieursbereich mit genauer Simulation optimiert.

Sie bietet folgende Features:

- Rigid body dynamics
- Kollisionserkennung
- Constraints: Gelenke, Achsen, ...
- Plattformübergreifend: Microsoft Windows; SGI IRIX; Linux
- schöne Demos (z.B. vom Mars Rover, ...)

Vortex wird u.a. von folgenden Firmen benutzt:

- EADS (European Aeronautic Defence and Space Company)
- NASA: Simulation von Robotern
- Simlog: Simulation eines “log forwarder” (spezielles Forstfahrzeug)
- Uni Bielefeld: Neuroinformatik: Robotersimulation

1.3 Open Source Spielephysikengines

1.3.1 ODE: Open Dynamics Engine

Die “Open Dynamics Engine” (ODE) [12] ist eine freie Softwarebibliothek für die Simulation von “Rigid Body Dynamics”, d.h. die Bewegungssimulation von festen Körpern/Objekten. Sie wurde von Russell Smith geschrieben, der auch den Kern der Mathengine implementiert hat. Konzipiert wurde sie, genauso wie die Vortex-Engine, eher für die Echtzeitsimulation von Robotern als für den Einsatz in Spielen.

Verfügbare Features:

- verschiedene Gelenktypen: Kugelgelenke, Scharniere, feste Verbindungen, Achsen, ...
- Kollisionserkennung zwischen Kugeln, Quadern, Zylindern und Flächen
- Berührungs- und Reibungsmodell
- plattformspezifische Optimierungen
- dynamische Eingriffe in die Simulation möglich

Als weitere Features sind u.a. geplant:

- Erweiterung der Kollisionserkennung
- Statisch inaktive Gruppen bei der Simulation ignorieren
- weitere Gelenktypen

1.3.2 DynaMechs (Dynamics of Mechanisms)

Auch DynaMechs wurde für die Simulation von Robotern entwickelt. Allgemein ermöglicht diese Softwarebibliothek die Simulation von mechanischen Systemen (z.B. Roboter mit verschiedenen Armen ...). Außerdem lassen sich diese Systeme auch “unter Wasser” simulieren. [8]

DynaMechs wurde in folgenden Projekten verwendet:

- Aquarobot: Ein gehender Unterwasserroboter, der vom “Port and Harbour Research Institute” in Yokosuka, Japan entwickelt und gebaut wurde.
- Tiburon: Ein remote gesteuerter Unterwasserroboter, der am Monterey Bay Aquarium Research Institute entwickelt wurde
- Genetic Programming System: Das Sigel Projekt vom “Lehrstuhl für Systemanalyse” der Universität Dortmund benutzte DynaMechs, um die Bewegungen von virtuellen Robotern zu simulieren. In diesem System wurden genetische Algorithmen eingesetzt, um diese Roboter zu einer Vorwärtsbewegung hin anzusteuern. [11]

2 Integration einer kommerziellen Physikengine

Falls man keine eigene Gameengine schreiben will, kann man auch auf ein komplettes “Game Development Environment” zurückgreifen. Zum Beispiel “RenderWare” von Criterion Software [13]. RenderWare enthält eine Grafikengine, eine Audioengine und eine Physikengine (entweder eine “kleine” von Criterion Software oder Mathengine/Karma). Damit muss man sich nicht mehr um das Zusammenspiel von Grafik, Physik und Sound

kümmern, sondern kann sich auf den Content und das “Gameplay” konzentrieren. Man erhält “automatisch” eine Engine, die auf mehreren Plattformen benutzbar ist (also für PC, Konsolen).

Um eine kommerzielle Physikengine in seine eigene Gameengine zu integrieren muss man viele Gesichtspunkte beachten [7].

- **Geometriedaten:** Die Physikengines können nur mit bestimmten Geometriedaten umgehen:
 - primitive Körper: Kugel, Zylinder, Quader, Flächen
 - konvexe Polygone
 - beliebige Polygone (“polygon soups”)

In der gleichen Reihenfolge steigt auch die Komplexität für deren physikalischen Berechnungen und der Aufwand Kollisionen festzustellen. Zum Teil müssen dann die komplizierten Objekte in primitive Körper, sogenannte “proxies”, eingesetzt werden, um damit vernünftig umgehen zu können (Boundingboxes/Boundingsheres). Dies geschieht teilweise implizit durch die Gameengine, teilweise explizit durch den Designer des Objektes in entsprechenden Tools.

- **Zeitmanagement:** Es wird zwischen 3 verschiedenen “Zeiten” unterschieden:
 - **Spiel-Zeit:** real-time-clock
 - **Frame-Zeit:** Die aktuelle Framerate, d.h. die Zeit zwischen 2 Bildern. Diese Framerate variiert je nach Aufwand zum Rendern eines Bildes.
 - **Simulations-Zeit:** Die aktuelle Zeit in der Physiksimulation
- **Objekte in Bewegung setzen:** Man kann Objekte auf verschiedene Art und Weise in Bewegung setzen:
 - eine Kraft an das Objekt anlegen
 - einen Impuls auf das Objekt wirken lassen
 - seine Geschwindigkeit direkt setzen

Das Problem mit “Kraft an Objekt anlegen” ist die Zeit, die die Kraft am Objekt wirken muss. Das bedeutet, es kommt die Aktion “wende diese Kraft an diesem Objekt für diese Zeit an”. Dies wirkt über mehrere Physik-Zeit-Schritte, weshalb es aufwendig ist für alle Objekte auf denen eine Kraft wirkt für jeden Physikschrift zu berechnen, wie lange die Kraft bereits wirkt und wie lange sie noch wirken muss (da die Physikschrift unterschiedlich lang dauern).

Bei Impulsen hat man den Vorteil, dass man die Wirkung in einem Zeitschritt berechnen kann. Der Nachteil ist, dass man ein Objekt damit nicht langsam beschleunigen kann.

- **“Räumliche Anfragen”**: Physikengine benutzen normalerweise “räumliche” Datenstrukturen, d.h. die Datenstruktur bildet den Raum mit seinen Objekten ab. Diese Datenstruktur lässt sich für viele Anfragetypen nutzen:
 - “trigger volumes”: z.B. löse ein Ereignis aus wenn Spieler Region betritt
 - “line-of-sight”: kann ich den Turm von hier aus sehen?
 - “ray casts” für KI: kann eine KI-Person einen Spieler sehen?
 - “ray casts” für User Interface: z.B. Spieler will Schalter auslösen
 - Bereichsanfragen für KI: z.B. wendet sich eine KI-Person dem Spieler zu, wenn er in seiner Nähe ist
 - Auswertung von theoretischen Bewegungen: z.B. kann eine Türe öffnen ohne etwas anderes zu berühren
 - Anfragen zur Wegfindung

3 Angewandte Techniken

Bei den derzeit verfügbaren Physikengines stösst man auf verschiedenartige Simulationstechniken, was sich auf Geschwindigkeit, Genauigkeit und Stabilität der Simulationen auswirkt. Da keine der Techniken die anderen auf allen Gebieten dominiert, hängt die Auswahl der Methode stark vom erwünschten Resultat ab.

3.1 Kollisionserkennung

Für das schnelle Detektieren von Kollisionen gibt es verschiedene Ansätze, die jedoch alle auf hierarchischen Modellen arbeiten. Auch die Genauigkeit der Kollisionserkennung variiert stark.

3.1.1 Bounding Spheres

Bei dieser Technik wird für jedes Objekt ein hierarchisches Modell von Kugeln aufgebaut, welche das Objekt einschliessen. Dieser Baum kann bei der Suche nach Kollisionen durchlaufen werden, bis auf Blattebene der tatsächliche Kollisionenpunkt gefunden wird. Dieser Ansatz ist bei beliebigen Objektstrukturen relativ effizient. [1]

3.1.2 Bounding Boxes

Dieser Ansatz ist ähnlich dem der Bounding Spheres, mit dem Unterschied, dass anstatt von Kugeln Quader verwendet werden. Dies entspricht gerade bei Spielen mehr der eigentlichen Form der Objekte, daher ist es hier auch der häufiger verwendete Ansatz. Man unterscheidet hierbei zwischen Axis-aligned Bounding Boxes (AABB) und Orientated Bounding Boxes (OBB). Bei AABB werden die Bounding Boxes am Koordinatensystem ausgerichtet,

wodurch die Kollision zweier Bounding Boxes ziemlich schnell berechnet werden kann [10]. Allerdings ändert sich ihre Grösse bei rotierenden Objekten permanent, und sie umschliessen i.A. relativ viel Raum, der nicht zum entsprechenden Objekt gehört. OBB rotieren mit dem Objekt mit und sind somit besser der Form des Objektes angepasst. Dagegen wird die Berechnung von Überlappungen aufwendiger. [1]

3.1.3 Binary Space Partitioning

Binary Space Partitioning (BSP) unterteilt die gesamte zu berechnende Welt durch eine Ebene in zwei Hälften, und wiederholt diesen Schritt rekursiv für beide Hälften. Dadurch wird die Welt in eine Art Baum gegliedert, den BSP-Tree. Da es genügt, eine Ebene zu finden, die zwischen zwei Objekten liegt, um nachzuweisen, dass keine Kollision vorliegt, braucht man nur diesen Baum zu durchlaufen, bis man eine passende Ebene gefunden hat. Um den genauen Kollisionspunkt zu finden, kann man wiederum für die einzelnen Polygone testen, auf welcher Seite der Flächen sie sich befinden. Binary Space Partitioning (BSP) ist eine in der Spieleindustrie weit verbreitete Technik und wurde erstmals in Doom verwendet. Allerdings ist ihre Genauigkeit durch die Feinheit des Baumes begrenzt. [1]

3.1.4 Kollisionserkennung auf konvexen und konkaven Objekten

Hat man eine Kollision der Bounding Boxes zweier Objekte detektiert, gibt es ausser dem hierarchischen Bounding Box Modell einen weiteren Ansatz zum Ermitteln des genauen Kontaktpunktes: Man wählt auf einer konvexen Hülle jedes Objektes einen Punkt und versucht nun, auf diesen Hüllen die Punkte so weit wie möglich anzunähern. Dieser Algorithmus lässt sich sogar auf konkave Objekte ausweiten, wenn man nach dem Erkennen einer Kollision der Hüllen die darunterliegenden Bereiche genauer untersucht. Dadurch kann man den Kollisionspunkt mit beliebiger Genauigkeit ermitteln. Allerdings steigt auch der Rechenaufwand, wenn man die Kollisionserkennung auf konkave Bereiche ausweitet. [10]

3.2 Rigid body dynamics

Die nun angesprochenen Techniken der Kollisionsantwort basieren alle auf “Rigid Body Dynamics”, d.h. es wird das Verhalten von Körper mit fixem Polygonmuster simuliert.

3.2.1 “Penalty-method” / Feder- und Dämpfersysteme

Hier werden Kollisionen dadurch behandelt, dass bei Objektüberlappungen Federkräfte zwischen den Objekten simuliert werden, die diese wieder auseinanderdrücken.

Diese Technik ist sehr schnell für jede Iteration, benötigt aber kleine Zeitschritte, da sie für Stabilitätsprobleme anfällig ist. Dadurch ist sie insgesamt eher langsam. Ein weiterer Nachteil ist die fehlende Unterstützung für feste Verbindungen und für Gelenke. Diese Methode wird u.a. in der Aero Engine eingesetzt. [12]

3.2.2 Lagrange multiplier

Bei dieser Methode wird das Verhalten der Objekte als ein Optimierungsproblem modelliert. Die Constraints, also die Beziehungen der Objekte untereinander, werden als Nebenbedingungen interpretiert, dann wird das ganze Problem mit Hilfe von Lagrange multiplern gelöst.

Lagrange multiplier erlauben flexible Behandlung von festen Verbindungen und Gelenken, können jedoch bei grossen Gelenksystemen langsam sein. Diese Technik wird von Havok und der Mathengine verwendet, wobei in der Mathengine durch reduzierte Genauigkeit die Geschwindigkeit verbessert wurde.[12]

3.2.3 Featherstone Methode

Roy Featherstone entwickelte in den letzten Jahren eine Methode, mit der er die Berechnung der Constraints in Subräume projizieren konnte [2]. Dadurch konnte die Komplexität des Problems stark reduziert werden, allerdings auf Kosten einer relativ hohen Konstante k (von $O(n^3)$ auf $O(kn)$).

Diese Methode ist sehr schnell, benötigt jedoch ein baumartig strukturiertes System und kann keine festen Verbindungen simulieren. Sie wird u.a in DynaMechs eingesetzt.[12]

3.2.4 Impulsmethoden

Der Grundgedanke bei impulsbasierenden Methoden ist der Verzicht auf Constraints, welche Beziehungen zwischen zwei Objekten angeben (wie "liegt auf"-Beziehungen oder auch Gelenke). Stattdessen werden alle Kollisionspunkte zwischen den Objekten berechnet und mittels Energie- und Impulserhaltungssatz die Reaktion der Objekte ermittelt.

Impulsmethoden eignen sich bei einer hohen Anzahl von Kollisionen und bei Systemen mit schnell veränderlichen Objektbeziehungen, wie z.B ein Ball, der auf einem Tisch hüpfet. Allerdings haben sie starke Probleme mit fixen Verbindungen und Gelenken. [9]

Literatur

- [1] Nick Bobic. Advanced Collision Detection Techniques. http://www.gamasutra.com/features/20000330/bobic_01.htm, 2000. 8, 9
- [2] Roy Featherstone. Modelling and Control of Contact between Constrained Rigid Bodies. <http://www.anu.edu.au/~roy/traXXmodel.pdf>, 2002. 10
- [3] Havok. Havok : Dynamik Gameplay. <http://www.havok.com>, 2002. 4
- [4] Havok. Havok Game Dynamics SDK, a produkt overview of the most comprehensive, optimized, real-time physics solution for game development. http://www.havok.com/products/downloads/Havok_1.7_Product_Overview.pdf, 2002. 3
- [5] Critical Mass Labs. Vortex. <http://www.cm-labs.com>, 2002. 5
- [6] Mathengine. Karma. <http://www.mathengine.com>, 2002. 4
- [7] Matt McLaurin. Outsourcing Reality: Integrating a Commercial Physics Engine. http://www.gamasutra.com/features/20020816/maclaurin_01, 2002. 7
- [8] Scott McMillan. DynaMechs (Dynamics of Mechanisms): A Multibody Dynamic Simulation Library. <http://dynamechs.sourceforge.net/>, 2001. 6
- [9] Brian Mirtich and John Canny. Impulse-based Simulation of Rigid Bodies. <http://www.cs-unc.edu/~dm/UNC/PHYSICS/Papers/thesis.ps.gz>, 1995. 10
- [10] Madhav K. Ponamgi, Dinesh Manocha, and Ming C. Lin. Incremental algorithms for collision detection between solid models, 1995. 9
- [11] PG 368 SIGEL. PG 368 - SIGEL. http://ls11-www.informatik.uni-dortmund.de/~sigel/seiten/einleitung_de.html, 2001. 6
- [12] Russell Smith. Open Dynamics Engine. <http://www.q12.org/ode/ode.html>, 2002. 5, 9, 10
- [13] Criterion Software. RenderWare Platform Homepage. <http://www.renderware.com>, 2002. 6