Computersimulationen für die Physik -

Welche Anforderungen müssen Physiksimulationen erfüllen und wie werden die mit Programmiertechniken umgesetzt?

Gliederung:

1. Vorwort

Physik-Simulationen, oder auch Physik-Engines werden immer wichtiger. Anwendungsgebiete sind digitale Crashtests, aber auch Computerspiele. Doch liegt bei Computerspielen das Hauptaugenmerk auf der Echtzeit-Simulation, schließlich möchte man nicht in einem sich in Zeitlupe bewegenden Auto sitzen. In anderen Gebieten ist aber wichtig, dass die Simulation möglichst realistisch abläuft. Aber auch moderne Computerspiele haben den Anspruch realitätsgetreu zu sein. Dafür werden diverse Tricks angewandt z.B. die Verletintegration, auf die wir später zurückkommen werden. Sie vereinfachen die Berechnungen für den Computer sehr, währen aber z.B. für Crashtests ungeeignet. Jetzt weren wir Ihnen unsere Physikengine zum ersten Mal präsentieren, müssen jedoch anmerken, dass wir uns auf die Mechanik beschränken.

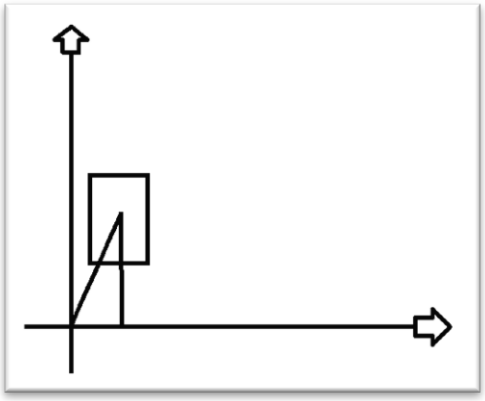
#Physik-Engine - Take 1

2. Kurze Einführung in BlitzBasic 3D

BlitzBasic ist eine von Mark Sibly entwickelte Programmiersprache. Wie der Name schon sagt, orientiret sich BlitzBasic am Syntax der von Microsoft entwickelten Sprache Basic, erweitert diese jedoch um eine ganze Reihe an 2D (und 3D) Befehlen.
Da der von BlitzBasic mitgelieferte Editor an einigen Stellen zu Wünschen übrig lässt, benutzen wir den Editor IDEal, welcher das Erstellen von Projekten mit verschiedenen Dateien ermöglicht.

3. Grundgerüst der Physik-Engine

Wir betrachten Ort, Geschwindigkeit und Beschleunigung als Vektoren. Die Berechnung der Vektoren funktioniert nach folgender Syntax: „add/multiply/subtractVector(Vektor1, Vektor2, Ergebnis)“.
(siehe Help.bb)

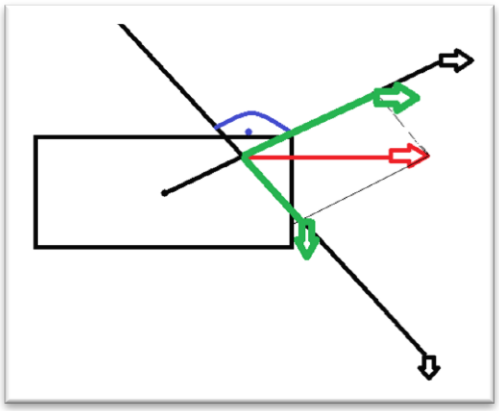


Zudem haben wir die Funktion DoTick, welche wir Ihnen jetzt demonstrieren werden.
#Physik-Engine - Take 2 – DoTick

Wir bemerken, dass wir mit der jetzigen Implementierung keine Möglichkeit haben Rotation, Geschwindigkeit und Beschleunigung zu verändern (mittels Kräften).

4. Einfache Bewegungen und Kräfte

Wenn eine Kraft senkrecht a.d. Radius anliegt löst sie Rotation aus, die mit F*r (Drehmoment) berechnet wird das ist die Hebelwirkung.
Kräftezerlegung/Kräfteparallelogramm:



Eine Kraft kann an einem beliebigen Punkt innerhalb des Objekts wirken. Ist sie direkt zum bzw. direkt vom Masseschwerpunkt weg gerichtet, so entsteht außschließlich eine **Geschwindigkeit**. Ist die Kraft senkrecht a.d. Radius gerichtet so, entsteht außschließlich eine **Rotationsgeschwindigkeit**. Wirkt eine Kraft müssen also die Anteile der Kraft in Richtung des Radiusses und die Anteile senkrecht zum Radius mittels des Kräfteparalelogramms berechnet werden. Als Ergebnis erhält man zwei Kräfte, wobei die "Bewegungskraft" einfach mit der Formel *F = m*a* in eine Beschleunigung umgerechnet werden kann. Die Kraft senkrecht zum Radius muss zuerst in einen Drehmoment(M) umgerechnet werden: *M = r*F* wobei *r* die Länge der Strecke vom Masseschwerpunkt (Rotationsachse) um Ansatzpunkt der Kraft ist (Radius). Das Verhältniss von Drehmoment und Rotationsbeschleunigung ist gleich dem Verhältnis von Kraft und Beschleunigug, wobei als Faktor nicht die Masse sondern das Trägheitsmoment verwendet wird. Das Trägheitsmoment ist ein Tensor (mehrdimensionale Matrix*), da der Trägheitsmoment von der Rotationsachse abhängt. Da es im 2-Dimensionalen aber nur **eine** Rotationsachse gibt vereinfacht sich das Trägheitsmoment auf einen Skalar (nur eine Zahl). Die Berechnug dieser wird in unserer Physikengine durch die Funktion *Ph_CalculateMomentOfInertia#* mittels der Objekt-Form (Shape) und der Masse übernommen.

*Matrix=zweidimensionaler Vektor

5. Kollison

5.1 Reibung

6. Was fehlt?
In unserer Physik-Engine gibt es allerdings auch ein paar Dinge, die wir (noch) nicht implementiert haben. Darunter Federn, diese sind aber sehr einfach mittels einer Kraft in Richtung der Ruhelage nachzubilden. Kraftfelder sind etwas schwieriger, da genau auf alle Bereiche (hier Punkte) des Objekts die in dem Kraftfeld sind eine Kraft wirken muss, aber nicht auf die Bereihe außerhalb. Jedoch können wir immer nur auf einen bestimmten Punkt eine Kraft wirken lassen. Ebenfalls haben wir den Luftwiederstand außeracht gelassen, da dieser ebenfalls sehr schwer zu berechnen ist (…).

7. Die Verletintegration
Mit der Verletintegration wird die Geschwindigkeit anhand der Differenz der aktuellen und der vorherigen Geschwindigkeit definiert. Verändert sich also die aktuelle Position, sei es auf Grund einer Kollison oder einer anderen Krafteinwirkung, so ändert sich automatisch auch die Geschwindigkeit. Ausserdem wird bei der Veletintergration jeder Punkt einzeln berechnet, wobei die einzelnen Punkte mit unendlich starken Federn (man kann sich diese als Eisenstangen vorstellen) verbunden sind. Dadurch entstehen automatisch realistisch aussehende Resultate, ohne dass alle physikalischen Gesetze implementiert werden müssen. Allerdings gibt es auch Probleme wie „einknickende“ Objekte, und zusätzlich benötigte Federn um die Objekte stabil zu halten (sh. Bild).
#ÄNDERN

8. Fazit
Die Anforderungen an Physik-Engines können sehr verschieden sein, je nachdem welchen Zweck die Physik-Simulation hat. Eine Physiksimulation in Computerspielen muss auf aktuellen Prozessoren eine Echtzeitberechnung schaffen, und trotzdem einen Grad der Realitätsähnlichkeit erreichen, der zwar keiner präzise Berechnung entspricht, jedoch die Anforderungen an das Spiel erfüllt. Um dies zu erreichen greift man oft auf Vereinfachungen zurück.

In anderen Anwendungsbereichen geht Qualität vor Effizienz. Virtuelle Crashtests zum Beispiel müssen nicht in Echtzeit berechnet werden, sondern sollten lieber sehr nahe an die Realität herankommen. Damit spart man viel Geld, General Motors will durch Umstellung auf virtuelle Crashtests Kosten in Höhe von ca. 420.000€ pro Crashtest sparen. Auch wenn viele EU-Staaten auf reale Bedingungen bestehen, könnte es in Zukunft vermehrt virtuelle „Vortests“ geben.

Auch Pilotenschulen setzen vermehrt auf virtuelle Rundflüge. Auch wenn in Deutschland ein echter Rundflug immer noch Voraussetzung für den Pilotenschein ist. Dieser Anwendungsbereich ist für Programmierer, sowohl als auch für die Hardware-Entwickler eine Herausforderung. Denn hier ist eine sehr realistische und genaue, aber auch Echtzeit-Berechnung notwendig.

Unsere Physik-Engine hat wieder einen anderen Zweck sie soll einem Nahebringen, wie so etwas funktionieren könnte. Aus zeitlichen Gründen und der Einfachheit halber ist sie auch nur auf 2-Dimensionen beschränkt und würde somit für viele Computerspiele und erst Recht nicht für Sicherheitssimulationen nicht in Frage kommen. Ihren Zweck hat sie jedoch (hoffentlich) erfüllt.

Quellen:
Physikunterricht 9./10. Klasse von Hr.Dr.Ebert

Literatur:
Maturarbeit "Pixel im freien Fall" - Programmierung einer Physiksimulation mithilfe der Verletintegration von Penedikt Bitterli, 3BS, 2009 - zu finden auf

http://www.noobody.org/Maturarbeit/Maturarbeit.pdf (zuletzt aufgerufen 23.4.2013 - 15:18 Uhr)

1. Kapitel Metzler Physik - 3. Auflage (2004) - herausgegeben von Joachim Grehn und Joachim Krause ; Schroedel Verlag GmbH, Hanofer

http://de.wikipedia.org/wiki/Drehmoment (zuletzt aufgerufen 23.4.2013 15:25 Uhr)

###Einige Vektorfunktionen aus verschiedenen Seiten im Internet, da diese in BB nicht vorhanden sind

Spieleprogrammierung mit DirectX 11 und C++; Susanne Wiqard; mitp - 1. Auflage (2010) - Kapitel 4

Quellcodeverzeichnis:

Help.bb - Funktion "LineLine": http://www.blitzbasic.com/codearcs/codearcs.php?code=1202 (zuletzt aufgerufen 26.4.2013 19:49 Uhr) - (2. Post) leicht verändert

Help.bb - Funktion "RotateVector": http://stackoverflow.com/questions/620745/c-rotating-a-vector-around-a-certain-point (zuletzt aufgerufen 26.5.2013 20:02 Uhr) - umgeschrieben in BlitzBasic

Ph_CollisonBox.bb - Funktion "Ph_CalculateMomentOfInertia": http://stackoverflow.com/questions/3329383/calculating-the-moment-of-inertia-for-a-concave-2d-polygon-relative-to-its-orgin (zuletzt aufgerufen 26.4.2013 19:51 Uhr) - umgeschrieben in BlitzBasic

Lizenzen:

Als Editor haben wir den auf BltzBasic (3D) angepassten Editor IDEal verwendet, zu finden auf http://www.fungamesfactory.com/ (zuletzt aufgerufen am 23.4.2013)

Sämtlicher Code ist in der Programmiersprache Blitz-Basic geschrieben,
kann aber ähnlich (bis auf die Anzeige) auch in anderen Programmiersprachen
geschrieben werden.

Für die Teamarbeit haben wir den Code auf GitHub (https://github.com) hochgeladen. Der Code ist somit auch zu finden auf (https://github.com/CCxBlacksmith/Physic-MSA-2013/
(evt. aktuellerer Stand als in der Präsentation verwendeter Code)