2D-Physik-Engines

Inhaltsverzeichnis:

2D Physikengines

Physikengines simulieren das Verhalten von durch ihrer Form definierten Körpern (Bodies) in einer virtuellen Welt. Ein physikalischer Body besteht dabei aus mathematisch definierten Formen (Shapes), meist konvexen Polygonen. Kreise oder Linienketten sind unter Umständen auch möglich. Diese Shapes können, aber müssen nicht mit der Grafik des Bodies übereinstimmen. Zumeist werden physikalische Bodies durch mehrere konvexe Polygone definiert, manche Engines bieten aber auch bequemere Lösungen. Es wird zwischen festen (Rigid) und weichen (Soft) Bodies unterschieden. Nur Soft Bodies können verformt werden. Die Verkettung von Rigid Bodies kann jedoch einen Soft Body simulieren, wie auch besonders starre (oder zähe) Soft Bodies einen festen Körper simulieren können. Verkettet werden Bodies über verschiedene Joints (z.B. drehbare Achsen, starre jedoch brechbare Verbindungen, etc.) . Grundsätzlich ist es auch möglich, eine dreidimensionale Physikengine zu verwenden, wobei hier meist bei jeden definierten Body eigens Einschränkungen im Bewegungsfreiraum auf zwei Dimensionen festzulegen sind. Neben den erhöhten Programmieraufwand bedeutet dies für die meisten Engines auch einen erhöhten Rechenaufwand, da Rechenkraft auf die dritte Dimension vergeudet wird. Empfehlenswert ist dies daher nur bedingt, sofern die betroffenen Programmierer sich bereits gut in einer bestimmten 3D Engine auskennen.

Einzelne Engines

Es folgen die wichtgsten Merkmale von vier gängigen 2D Physikengines, sowie eine Gegenüberstellung ihrer eventuellen Stärken und Schwächen bezüglich möglicher Anwendungsfälle.

Box2D

Hierbei handelt es sich um die wohl am meisten verbreitete 2D Physik Engine. Die zugrundeliegende Programmiersprache ist C++ Ein C# Port (Farseer) bietet einige Funktionen, die dem Original fehlen, wie die automatische Generierung von mehreren konvexe Polygonen aus einen Konkaven. Die Möglichkeit Shapes aus Kantenreihen zu definieren bietet eine naheliegende Lösung für das Leveldesign, solange statische Landschaftselemente erzeugt werden sollen. Derartige Shapes können nämlich nur mit herkömmlichen Polygonen oder Kreisen kollidieren. Sehr schnelle bewegliche Objekte können dank der wahlweise kontinuierlichen Kollisionsberechnung nicht aus Versehen Objekte zwischen zwei Updatephasen durchdringen. Dieses auf Integration basierte Verfahren ist jedoch relativ rechenaufwendig (daher optional). Stabil funktioniert Box2D nur in einem begrenzten Rahmen von feststehenden physikalischen Einheiten.

URL: http://box2d.org/ Farseer: http://farseerphysics.codeplex.com/

Chipmunk

Chipmunk hat den Ruf, besonders einfach zu sein, und findet vor allem in iOS Spielen starke Verbreitung. Für stark objektorientiert arbeitende Programmierer ist der Programmierstil jedoch eher gewöhnungsbedürftig. C als zugrundeliegende Programmiersprache ermöglicht viele Portierungen und Wrapper, allerdings bietet Box2D da trotzdem eine höhere Bandbreite an unterstützten Sprachen. Die Syntax ist etwas umständlicher, so können Vektoren aufgrund fehlender Operatorenüberladung nich mit einem simplen [Vec1 + Vec2 + Vec3] addiert werden. Stattdessen finden Funktionsaufrufe statt.[VecAdd(Veccadd(Vec1, Vec2), Vec3)] Ein Nachteil von C kann sein, dass kein automatisches Speichermanagement stattfindet. Anstelle eines integrationsbasierten Kollisionsverfahrens, wie bei Box2D, wird bei Chipmunk auf Raycasting als eine Möglichkeit zur Vermeidun ungewollter Durchdringungen verwiesen. Hiermit können zukünftige Kollisionen bei approximierten Kursen detektiert werden können.

URL: http://chipmunk-physics.net/

Jello

Die Besonderheit von Jello ist die Simulation von Soft Bodies. Diese verformen sich entsprechend bei Kollision. Empfehlenswert ist Jello nur für diesen Spezialfall, da keine große Verbreitung existiert, und die Entwicklung nur von einer Person sporadisch betrieben wird. Dies hat auch zur Folge, dass viele nützliche Funktionalitäten, wie eine kontinuierliche Kollisionsberechnung oder diverse Joints fehlen.

URL: http://walaber.com/wordpress/?p=85

Physics2D.Net

Diese Engine ist für C# besonders optimiert worden, und bietet in dieser Umgebung vermutlich die schnellsten Ergebnisse. Hervorzuheben ist neben der Unterstützung von konkaven Polygonen ein Werkzeug, das automatisch Polygone aus Sprites erzeugen kann, und somit die Integration von Assets beschleunigt. Nachteilig ist der letzte Entwicklungsstand von 2008, und die wie bei Jello eher geringe Verbreitung.

URL: https://sites.google.com/site/physics2d/

Gegenüberstellung

Da die Wahl der Physikengine stark projektabhängig ist, ist eine globale Bewertung nicht möglich. Einige allgemeine Aussagen, wie beispielsweise über die Geschwindigkeit, können auch nicht getroffen werden, da jeder Anwendungsfall oder Codestil andere Ergebnisse erzeugen kann. Die Punkte in der nachstehenden Tabelle sind daher für eine Wertung nur dann zu berücksichtigen, falls sie für den konkreten Anwendungsfall von Bedeutung sind. Es wird dringend empfohlen, die Engines auch eigens zu recherchieren, da diverse weitere Eigenschaften, wie die möglichen verwendbaren Joints, keine Berücksichtigung fanden.

	Body	Shapes	Kollision	Programmiersprachen	Support	Entwicklung	Sonstiges	Lizenz
Box 2D	+ Rigid	+ Kreise + Konvexe Polygone o Kantenreihen	+ Continuous	+ C++ + C# Farseer Engine o Objective C o As3 o Java o Diverse Scriptsprachen	+ Community + Dokumentation + Anleitung + Beispiele	+ Konstant Aktiv	- Einheiten Beschränkung auf Meter, Kg, Sekunden zwischen 0.1 bis 10	+ Frei

Chipmunk	+ Rigid	+ Konvexe Polygone	- Non Continuous	+ C + Objective C (Kostenpfl. Pro Version) o As3 o Java o Diverse Scriptsprachen	+ Community + Dokumentation + Beispiele	+ Konstant aktiv	- Kein Speichermanagement o Raycasts als Workarround statt Continous Collision	+ Frei in C - Kostenpflichtig für andere Sprachen
Jello	+ Soft	+ Konvexe Polygone	- Non Continuous	+ C# o C++ o As3	+ Beispiele o Dokumentation	o Sprunghaft		+ Frei
Physics 2D. Net	+ Rigid	+ Konvexe Polygone + Konkave Polygone + Kreise	- Non Continuous	+ C#	o Dokumentation o Community	- Letzte Version von 2008	+ Werkzeug zur Generierung von Shapes aus Sprites	+ Frei

Anwendungshilfe

+ Erfüllt die beschriebene Eigenschaft gut / ist bei Anwendungsrelevanz positiv zu werten

o Erfüllt die beschriebene Eigenschaft nicht genügend / ist bei Anwendungsrelevanz mittelmäßig zu werten

- Ist bei Anwendungsrelevanz negativ zu werten

Stand: 30.11.2012 - Pascal Schroeter