Infomationen Physic Engine

Bestandteile

1. Mass Aggregation
2. Collision Detection
3. Collision Resolver

Koordinatensystem

1. Linkshändig -> DirectX
2. Rechtshändig -> OpenGL

Geometrie

Meistens werden 3 Typen von Geometrie berücksichtigt

- Primitive Objekte (Kugel, Quader, Kapsel, Zylinder, Plane)
- Konvex Polygonale Geometrie (Tetraeder)
  -> Konvexe Objekte sind einfach aufzuspüren (Quickhull Algorithmus)
- Polygon Soups (Komplexe Objekte für zB einen Canyon)
  -> kritisch und schwer robust zu implementieren

Jedes Objekt braucht eine zusätzliche Proxy Geometrie die genutzt wird um Berechnungen auszuführen (Collider). Dieser Proxy braucht eine ständige Verbindung zum Objekt.

Implizite Proxys -> automatisch generierte minimalste primitive Objekte
Explizite Proxys -> selbst aufgebaute Geometrie

Transformationen

Jeder Geometrie sowie der dazugehörige Proxy müssen dieselben Transformationen durchlaufen und werden dadurch immer parallel bewegt.
Die Vertices der Proxys sollten in die Raumkoordinaten der jeweiligen Geometrie übertragen werden, außer man will einen Proxy für unterschiedliche Geometrien benutzen (Effizient durch Instantiierung aber wenig intuitiv).
Skalierung wird bei Rigid Body Transformationen nicht berücksichtigt.

Zeitmanagement

Game Time:
Realzeit in Sekunden, einfache und übertragbare Zeit -> schafft Konsistenz bei Kommunikation zwischen Subsystemen
Sollte skalierbar angelegt sein

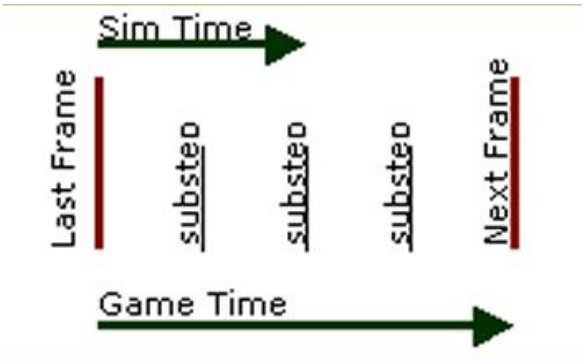


Figure 3: The simulation takes smaller steps than the game clock. They meet at frame boundaries.

Frame Time:
Moment des gerenderten Bildes, Berechnungen werden zwischen Frames durchgeführt

Simulation Time:
Momentane Zeit in der Physik-Engine, In jedem Frame werden Zwischenschritte gestartet bis die Simulation Time der Frame Time entspricht.

Simulationen werden inkonsistent dargestellt wenn die Zwischenschritt Rate verändert wird, was dazu führt, dass Kollisionen nicht richtig dargestellt werden bzw nicht richtig berechnet werden. Kleinere Schritte erzeugen bei schnellen und komplexen Bewegungen genauere Berechnungen.
Man muss die Simulation Time an die Game Time anpassen was aber dazu führt, dass drops in der Framerate die Zwischenschritte verkleinern und somit den CPU anspruch erhöhen (was wiederum zu FPS Drops führt usw.). Deshalb muss ein Weg eingeführt werden um die CPU zu entlasten (zB durch komplettes aussetzen der Physik).

Gleiche Kollisionen werden in verschiedenen Frames erkannt und deshalb müssen redundante Kontakte ausgefiltert werden.

<u>Krafteinwirkungen</u>
Bewegungen können auf verschiedene Weisen dargestellt werden

Krafteinwirkung (Force):
Jeder Zwischenschritt erzeugt eine Kraft bis die Geometrie an ihrem Ziel angekommen ist. Dabei muss beachtet und gespeichert werden wie lange welche Kraft angewendet werden soll. Komplexität wird dadurch erhöht und die Genauigkeit leidet.

Impulse:
Ein Impuls ist einer Krafteinwirkung mit der benötigten Dauer im vorhinein multipliziert und wird dann direkt auf die Geometrie angewendet. Das Objekt erreicht die Höchstgeschwindigkeit sofort und wird nicht bis dorthin beschleunigt.

Geschwindigkeit:
Mittelweg zwischen den anderen Krafteinwirkungen. Besseres Handling als mit Forces und bessere Kontrolle als bei Impulsen. Berechnungen um die geschwindigkeit zu ermitteln die benötigt wird um in gewollter Zeit das Ziel zu erreichen.
Nachteil: Andere Kräfte werden nicht automatisch berücksichtigt sondern müssen einzeln berechnet werden.

Beschleunigung:
Wird oft von der Positionsberechnung ausgeschlossen da minimale Beschleunigung keine großen Auswirkungen hat aber bei großen schlagartigen Beschleunigungen wird die berechnung der Positionsveränderung schwierig. (Millington, 56)

Mehrere Kräfte die gleichzeitig auf einen Körper wirken können gleichzeitig dargestellt werden durch D'Alembert's Principle einer Alternative für die 2. Regel von Newtons Laws of Motion. Alle Kräfte werden dabei per Vektoraddition zusammengerechnet mit der Formel

$$f = \sum_i f_i$$

Springs

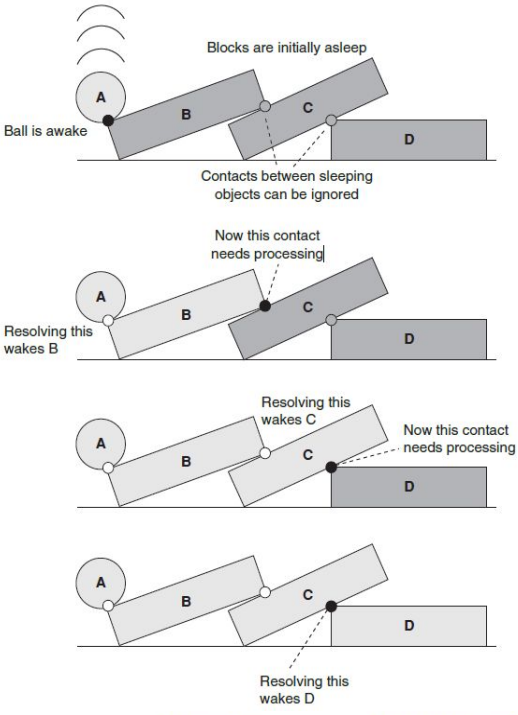
Die Kraft die eine Feder (Spring) ausübt bezieht sich nur auf die Distanz der Ausdehnung/Kompression wie von Hooke festgelegt. (zB doppelte Ausdehnung = doppelte Kraft)
Die Kraft wird an beiden Enden spürbar, allerdings gibt es auch Springs die nur in eine Richtung funktionieren.

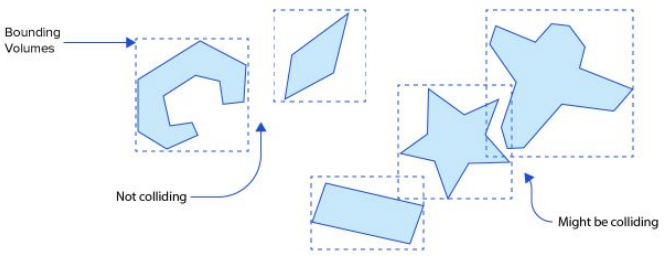
Auftrieb (Buoyancy)

Auftriebskraft ist abhängig von der Masse des Objekts. Der Auftrieb kann gemessen werden indem Objekt ins Wasser gelegt wird, das Gewicht des verdrängten Wasser entspricht dabei der Auftriebskraft.
f = mg
g = Gravitation(veränderbar)
m = Masse Objekt(bleibt gleich nachdem festgelegt)

Collisions

1. Let the start time be the current simulation time, and the end time be the end of the current update request.
2. Perform a complete update for the entire time interval.
3. Run the collision detector and collect a list of collisions.
4. If there are no collisions, we are done: exit the algorithm.
5. For each collision, work out the exact time of the first collision.
6. Choose the first collision to have occurred.
7. If the first collision occurs after the end time, then we're done: exit the algorithm.
8. Remove the effects of the Step 2 update, and perform a new update from the
start time to the first collision time.
9. Process the collision, applying the appropriate impulses (no interpenetration
resolution is needed, because at the instant of collision the objects are only just
touching).
10. Set the start time to be the first collision time, keep the end time unchanged,
and return to Step 1.
(Millington, 135f)





Player Character

Üblicherweise wird die Physik-Engine an das gewollte Spielprinzip angepasst und hierbei wird besonderer Wert auf den Spieler und seinen Proxy gelegt. Dieser wird stark angepasst, so dass dort möglichst wenige Probleme entstehen.
Da wir einen flexiblen Collider für unsere später selbstgebauten Charakter haben, wird das wohl auf ein minimum reduziert (oft wird hier anfangs eine Kapsel benutzt aber da wir Collider hinzufügen können sollten, können wir das auch für den Charakter tun)

Benötigte Elemente/Schnittstelle

- Daten der Objekte in der Szene
- Collider
- Rigid-Body
- Physics Material (standardmäßig im Rigid-Body enthalten aber sollte anpassbar sein)
- Zeit