Mesh Destruction
Final Report

Students Name *
Edinburgh Napier University
Physics-Based Animation (SET09119)

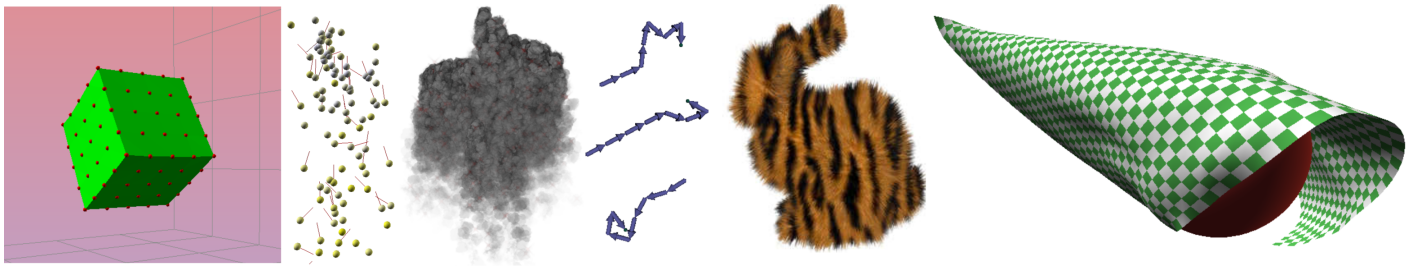

Figure 1: *Place a teaser image at the top of your report to show key examples of your work (e.g., multiple screenshots of the different test situations) - Every figure should have a caption and a description. For example, each figure is labelled and explained: (a) soft bodies, (b) particles, (c) inverse kinematics, (e) fur shells, and (f) position-based dynamics for cloth effects.*

Abstract

In most physics based games, any destructible objects are normally loaded in with their fractured components on runtime. What this physics simulation is hoping to look at is ways to slice meshes in real time to generate new fragments.

The approach looked at the in the design report suggested a technique called voronoi shattering in order to break the objects up into random fragments. However this was not implemented in the final simulation due to difficulty and time constraints. Therefore a simpler approach was used where the object is split is recursively using a series of random planes which intersect through the model mesh.

Keywords: Rigid body, collision detection, bounding volumes, mesh destruction, OpenGL, real-time, physics, computer graphics

1 Introduction

In most video games, destructible objects are pre-computed models which are the equivalent of the model fragments. The purpose of this simulation is to create proper mesh fragments in real-time just using the original meshes information. A key part of this simulation is optimisation as creating new objects is an expensive process.

2 Related Work

Currently there are several plugins that use either the same idea or has a similar solution. However these are mostly implemented into modelling software where optimisation is not an issue as it is not required in a real-time environment. The most similar idea is the plugin by [Esteve 2011]. The tutorial provided gave a insight on how to split the meshes by using planes.

3 Simulation

Overview

Detailed description

Implementation The implementation of this project can be broken into three major segments. These are: rigid bodies, collision detection and meshes.

Collision detection

There are three types of colliders that are used in this simulation. These are the plane collider, the sphere collider and the orientated bounding box.

Collider

This is the basic collider, which every other collider will inherit from. As it is the most general form it only contains a position.

```
Class Collider
{
        Vector3 Position;
}
```

Plane collider

The plane collider is used to simulate the floor. However as the floor is not meant to have true rigid body properties. i.e it should not 'fall' as it will never move, therefore there is a plane collider and a graphic drawn for it rather than it being a true 'Model' object in the scene.

Sphere collider

This is a simple sphere which encompasses the entire model object. It only requires a position in the world, which is updated whenever the rigid body is updated, and a radius. The radius is calculated by finding the distance from the centre of the model to its furthest point.

*e-mail:40167111@live.napier.ac.uk

Orientated bounding box

This is similar to an axis aligned box however it can handle rotations. Detecting collisions for an OBB however is more cost as there are more potential ways two OBB can collide. Therefore, in order to minimise calculations, the OBB checks only occur if both object pass the sphere collider check.

This is where i am going to show pseudocode for sphere to sphere and OBB to OBB.

Rigid bodies

In order to have a simulation which simulates proper physics, rigid bodies need to be attached to the objects. An rigid body consists of the following: position, previous position, mass, inverse mass, orientation, forces being applied to the object.

Meshes

Update

To create an realistic simulation a concept of a 'physics tick' is required. This is in place in order to decouple the physics from the frame-rate. Thus the physics will run the same if the frame rate is at 10 FPS or 60 FPS.

There are three parts to the physics update. There are summerised below:

Colliding object The first stage is detecting if any objects are currently colliding with each other. If there is a collision between two objects then a struct called 'CollisionInfo' is created which contains the following data:

- Pointer to object A.
- Pointer to object B.
- Depth of overlap in objects.
- Direction to be pushed towards.
- Position of collision.

This struct is then pushed back onto a vector; which contains all the collisions that have happened between the models in this current update.

Resolve collisions

For each of the collisions that have occurred a resolve stage needs to occur. This calculates what should happen to each object in terms of direction it is now going; the velocity of the objects and if the proper conditions have been met, one of the objects shatter.

Integrate rigid bodies

After collisions have been sorted, each model need to update there current position, previous position, orientation.

4 Testing and evaluation

5 Conclusion and Future work

To conclude, I am done.

References

DAY, R., AND GASTEL, B. 2012. *How to write and publish a scientific paper*. Cambridge University Press.

SAKO, Y., AND FUJIMURA, K. 2000. Shape similarity by homotropic deformation. *The Visual Computer 16*, 1, 47–61.