Effective Collision Detection Techniques and Rigid Bodies

Jordan Stephano Gray*
40087220, Edinburgh Napier University
BSc Games Development Year 3

Figure 1: *From 60 objects to 1000 objects with implemented collision detection*

Abstract

Collision detection in games can be one of the most expensive processes if not done correctly. This report will be focusing on using techniques to reduce the cost in correlation with using rigid bodies. There will be a large focus on performance and how an increase in objects affect the running of the program using different methods. For example using narrow phase collision detection alone and then implementing broad phase collision detection. Some of the first games ever created used collision detection, asteroids, Pac-Man and pong etc. So this report will explain the importance of this in modern games.

Keywords: collision detection, rigid body, collision response, octree

1 Introduction

In almost every modern 2D and 3D game there will be some form of collision detection implemented, if done in an inefficient manner this process will bottleneck the games performance. There are two main stages to collision detection in games, the narrow phase and the broad phase detection. Generally speaking the narrow phase is more expensive to work out computationally especially with the addition of more and more objects in a scene. The purpose of the broad phase is to divide up a scene in a way that only specific objects are tested for collision with other objects nearby to it. In games these two main methods are both implemented to create more realistic looking scenes with object interaction, within these two methods there are several techniques however to how the collisions are computed and results calculated.

*e-mail:graybostephano@gmail.com

2 Related Work

In the early days of research into collision detection and rigid bodies there was a desire to produce realistic object collision in animation. This is researched and discussed in [Moore and Wilhelms] where the main focus is on detecting collisions between objects and their collision response using springs and rigid bodies. It's mentioned here as it was early on in the development of collision techniques that animators would have to manually deal with collisions themselves beforehand. The main idea discussed here was based on detecting when two objects are close and inserting a temporary spring at the point of contact which would apply equal and opposite forces to the objects based on their elasticity. Another discussed solution for working out the collision response is a solution called the Analytical Solution which depends upon the conservation of momentum during collisions which results in new linear and angular velocities.

Another method using impulses in conjunction with collision detection is also discussed by [Mirtich and Canny] early on. Here it's discussed that the dynamic simulation of rigid bodies and how they interact with other objects can be done using impulse-based methods. It is said that impulse-based methods produce more realistic results in a 3D scene/animation. The main idea behind this concept is the addition of an impulse force between two objects when colliding rather than using a penalty-based method which just applies a negative force rather than applying a whole new force all together based on the results of the collision.

In the paper [Robert Bridson and Anderson] the main topic of discussion is the robust treatment of collision detection and how they handle it in relation to cloth animation. There is a large focus on the handling of collision detection using techniques and data structures to improve performance as they believe collision detection causes bottle-necking when it comes to performance.

Rigid bodies have a close relationship with collision detection and the paper [Eran Guendelman] explain the importance of this whilst trying to simulate the stacking of non-convex rigid bodies on each other and other objects. This paper explores a variety of problems and how to solve them since the geometry of the objects are more complex than a simple convex rigid body. One of the techniques discussed is Interference Detection along with Time Integration and Collision all to solve the issue of optimising a scene full of colliding rigid bodies.

3 Overview

The simulation is made up using a combination of graphics with the help of OpenGL and the physics with the help of some basic maths headers which will help with the physics calculations etc. One of the main goals of the simulate is to demonstrate a scene where moving objects react in a realistic way when colliding with other moving objects and other static objects. The other main goal is to demonstrate with the use of data structures and techniques that many of these objects can be simulated at one without being impacted by the collision detection bottle-necking. The expected comparison between using just narrow phase techniques compared with the implementation of a broad phase technique is desired here to prove how effective broad phase methods can be.

The simulation keeps the use of complicated graphics to a minimum to allow for maximum computational power to be allocated to the physics side of it so the use of basic geometry is apparent. When the simulation is started there is an initial empty room which rotates and is made up of four walls which are counted as planes, the user can then press the space key in order to generate sphere objects in random locations, at random sizes, coloured randomly and having a random initial velocity. This makes for a simulation which varies each time it is run and makes the results more interesting. Testing if there is a collision between the spheres is done in two optional ways which can we switched back and forth during runtime, just using brute the brute force method which involves constantly testing each sphere against each other sphere object and using the implemented octree which should improve performance when there is a lot of objects in the scene.

The main collision techniques used are sphere sphere which detects when two spheres collides and produces a collision response and sphere plane when the sphere collides with one of the walls to produce a realistic collision response.

4 Collision Detection

Collision detection is the process of detecting when two objects in a computer generated scene collide. It isn't enough just to detect when a collision happens but it is also necessary to find out which objects collide, how they collide (contact data) and the response of both objects due to the collision based on a variety of factors. In [Moore and Wilhelms] it's mentioned that without collision detection in a scene that objects would sail through each other rather than producing the desired realism. This realism is required in such things as modern day games and without collision detection they would appear more like a film rather than an interactive game. As mentioned in [Moore and Wilhelms] collision detection is a very expensive process especially when using just brute force methods which would be a very unrealistic way of doing things in today's games/simulations.

There are many different techniques when it comes to collision detection each with their own advantages and disadvantages. For example working out sphere-sphere collision is very easy to implement and cheap and also returns accurate contact data given the time step is small enough but its disadvantage lies when bounding a sphere around a shape that graphically isn't a sphere itself. On the other hand there is the AABB technique for example which is harder to implement and is more suited to convex objects that aren't spherical but can become inaccurate when the object is rotated and expensive if accuracy is desired by redrawing the bounding box several times.

5 Narrow Phase Technique

The goal behind the narrow phase is to work out the exact point in a game or simulation that two objects make contact and using all the contact information and object information to work out the results.

5.1 Bounding Spheres

This is the simplest way of detecting collisions between objects. In games and simulations like this one the graphics engine and physics engines are kept separate. This means a sphere can be placed around an object which only requires two parameters, centre position of sphere (typically centre of graphics object) and a radius. All that's needed here to test for collisions is to test the radius of the bounding spheres of two objects in relation to each objects centre position. If these radii overlap then there is a collision. If distance between two objects is less than or equal to the radius, collision has occurred. See Figure 2.

Intersection if: $d^2 < (r1 + r2)^2$
Where d is the distance between the centres on both objects and r is each objects radius.

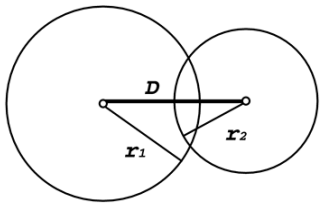

Figure 2: *Example of collision with bounding spheres Image taken from* `http://flylib.com/books/en/3.188.1.23/1/`

5.2 Sphere Plane Collision

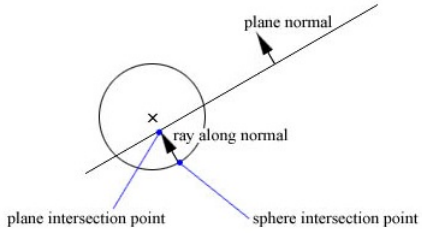

Figure 3: *Example of collision with sphere and plane. Image taken from* `http://www.peroxide.dk/download/tutorials/tut10/pxdtut10.html`

This is the detection of the collision between a basic sphere and a plane in a scene. An example of this here would be the sphere objects colliding with walls, the floor or the ceiling in a room. When a collision is detected in the broad phase stage a narrow phase calculation must be carried out to see if the sphere is actually penetrating the plane (As planes aren't usually infinite). A contact point must be established between the sphere and the plane, then depending on penetration depth a valid correction force must be applied in the opposite direction towards the surface normal of the plane. This is done with the help of the plane equation.

Intersection if:
$N \cdot S + d < r$

Calculation:
$p = r - (N \cdot S + d)$

$P = S - N(r - p)$

Where N is the normal of the plane, d is the distance from the centre of the sphere to the plane and r is the radius of the sphere.

5.3 Collision Response

After calculating if collision have occurred it isn't enough just to calculate that there was a collision. In order to create a realistic looking physics-based scene the objects must react how they would in the real world based on many variables. Normally the linear calculations are worked out first which is the calculation of the linear velocity of the object and overall force. The corrective forces take this into account and apply forces back onto the object to give the look of how an object would react in the real world to collisions. The main variables to take into account are force = F, velocity = v, initial velocity = u, mass = m, acceleration = a, distance = s, time = t and gravity acts as acceleration in this case accelerating all objects downwards on the Y-axis usually at 9.82ms. Some very vital equations are needed for working out the linear components of the objects which must constantly be updated. These calculations include:

$s = t(u + v)/2$
$s = ut + (0.5)at^2$
$v = u + at$
$v^2 = u^2 + 2as$

When working with rigid bodies there is also the angular calculations to take into consideration such as Torque and Inertia etc, for now lets keep it simple.

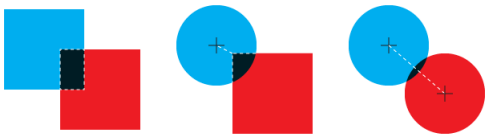

Figure 4: *Basic collision response. Image taken from* `https://github.com/jeffThompson/CollisionDetectionFunctionsForProcessing`

When the two objects collide three main variables are taken into account when working out corrective force. The contact point, the contact normal and the penetration depth. With this information and perhaps the velocity of the object(s) in question a reasonable corrective force can be applied in the correct direction with the correct amount of force.
The order in which collisions and collision response is as follows:

Step 1 Move Objects:
-Apply any forces.
-Apply Newtonian Laws.
-Integrate.
-Apply linear forces (and angular in the case of rigid bodies.)

Step 2 Detect Collisions:
-Broad phase.
-Narrow Phase.

Step 3 Resolve Collisions/Collision Response:
-No collision.
-Resolve collision in realistic manner.

5.4 Axis Aligned Bounding Box

This is another narrow phase technique used in collision detection. This technique was not used in this project for a few reasons that did not make it suitable. Firstly the technique itself is inefficient especially when compared to the easy to calculate sphere-sphere method. Secondly it's difficult to use when taking rotation into account, you could either keep rebuilding the bounding box per rotation which is extremely inefficient or settle for inaccuracy. This is not optimal when using rigid bodies as each of the eight points of the box must be rotated along with the object.

The benefits to using AABB include it being easy to calculate normals on the box and you can use simple point-inside tests to test for object penetration using plane equation. Two points are specified in this method as seen in Figure 5. The (Xmin, Ymin, Zmin) and (Xmax, Ymax, Zmax). Working out the point inside test is as simple as seeing if the point is in between these two points:
$Xmin <= X <= Xmax$
$Ymin <= Y <= Ymax$
$Zmin <= Z <= Zmax$

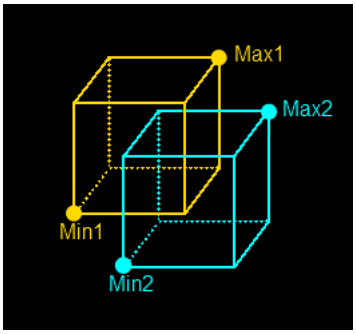

Figure 5: *Example of AABB to AABB collision detection. Image taken from* `http://www.miguelcasillas.com/?p=30`

6 Broad Phase Technique

The purpose of the broad phase is to reduce an exponential problem N^2 complexity when adding more objects to the scene by eliminating collision checks that are very unlikely to happen. For example there is no need to check collisions between two objects that are at opposite end of a large scene as it's highly unlikely that in the next frame they will be colliding (Only chance is if both object are moving at an extremely high velocity towards each other). The goal here is to reduce the number of efficiencies and optimize the simulation as much as possible. Most collision detection methods use approximation to save on computational power this is where broad phase comes in and will speed up the simulation.

There are many ways of achieving this using spatial data structures, including:
-Uniform grids.
-Hierarchical grids.
-Quad tree.
-Octree.
-Binary space partitioning.
-3D K-D tree.
-Axis sorted list.

6.1 Octree

An octree is used in 3-Dimensional space. What it does is split up a scene into smaller cube shapes called nodes dividing up by

eight nodes each time. Only objects in specific nodes are tested for collision against other objects in the same node. If an object spans over several nodes that object is tested against each object in each node it spans over.
The main drawback of using this method is if it is implemented into a scene with many moving objects it can be fairly expensive as each object is moving from node to node every few frames but this is still less expensive than just using narrow phase techniques (N^2 Complexity). This broad phase technique was implemented in this project as it was fairly simple to grasp and use although it is not the most optimal solution here.

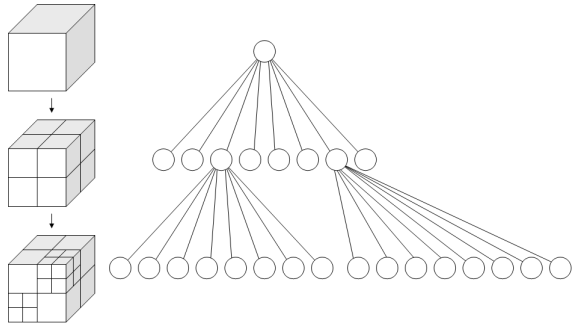

Figure 6: *Example of Octree and its tree structure. Image taken from* `https://en.wikipedia.org/wiki/Octree`

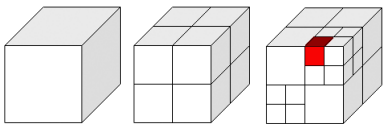

Figure 7: *Example of Octree splitting up a scene into nodes. Image taken from* `http://procworld.blogspot.co.uk/2010/11/space-warps-and-octrees.html`

6.2 Sweep and Prune

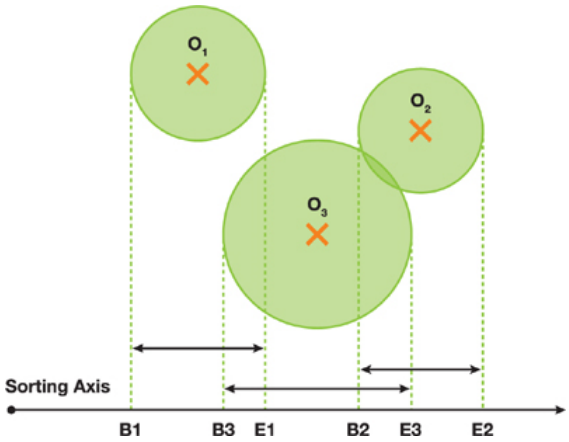

Figure 8: *Example of Sweep and Prune method. Image taken from* `http://http.developer.nvidia.com/GPUGems3/gpugems3_ch32.html`

In hindsight this would perhaps have been a better broad phase method to use. This is because this method is better at dealing with dynamic scenes with many moving objects. This method requires the representation of objects via AABB's (Axis Aligned Bounding Boxes) and works along a sorting axis. The objects are lined up with this axis and it is simple to work out any overlap/collisions between objects. For this algorithm to work a sorted list of min and max edges for each object must be kept, this will also keep a list of currently overlapping objects due to penetrating mins and maxes.

Once the list is sorted there should be a group of overlapping objects which is done using AABB tests. Then you go along the sorting axis and cull any collision checks that are unnecessary. There are many ways to sort these objects using different sorting algorithms.

7 Rigid Bodies

It was desired to include rigid body objects in this project. Rigid bodies are objects that do not change shape/form no matter what force is exerted upon them. The distance between each point in a rigid body does not ever change.

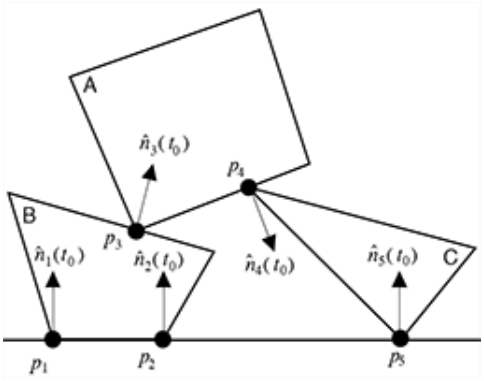

Figure 9: *Example of stacked rigid bodies. Image taken from* `http://www.cs.cornell.edu/courses/cs5643/2010sp/`

With rigid bodies, as well as having to calculate linear forces there is also the need to calculate angular forces. This includes the calculation of both torque = τ and inertia = I. Rather than the usual linear components that make up the linear calculations of forces there is a new set of variables including, angular velocity = ω, angular acceleration = α and the angle = θ. These are 3-Dimensional vectors rotating around all three axes.

Newtons second law is: $F = ma$ where F is force, m is mass and a is acceleration. To find the rotation force (torque) we use:

$$\tau = I\alpha$$

Where τ is the torque, I is inertia and α is angular acceleration. In order to calculate the torque itself we use:

$$\tau = F \times d$$

Where d is the distance from the centre of mass of the object to the point of contact in order to cause rotation as seen in Figure 10.

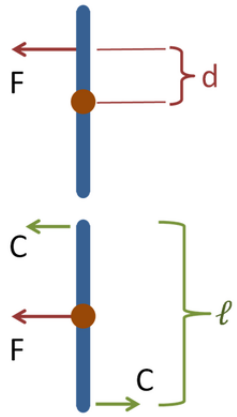

Figure 10: *Acting rotational force on rigid body. Image taken from* `http://www.quickiwiki.com/sh/Spreg_sila`

In order to find the angular acceleration you rearrange $\tau = I\alpha$ for α. The result is: $\alpha = I^{-1}\tau$.

When working in 3-Dimensions a 3x3 inertia matrix is required. Here the matrix for symmetrical objects is used:

The inertia matrix for symmetrical object in 3D space is given by:

$$\begin{pmatrix} Ixx & 0 & 0 \\ 0 & Iyy & 0 \\ 0 & 0 & Izz \end{pmatrix}$$

Rigid bodies have the following properties in addition to regular properties:
-Orientation matrix.
-Angular velocity vector.
-Torque vector.
-Inverse inertia matrix.

When using rigid bodies in a scene it is better to use impulse-based methods for collision response. This is discussed in [Mirtich and Canny] where they solely focus on this aspect and is said here that this method is simpler and faster than constraint-based simulation. The basic calculation for calculating an impulse which is the change in momentum of an object is:
$J = Ft$

Where J is the impulse, F is the force and t is the time. To work out the actual momentum of the object the calculation:
$p = mv$

is used where p is the momentum, m is mass of the object and v is the velocity of the object.

Basic attributes of a rigid body:

```
//This is the main rigid body class.
//This holds information about rigid body objects.
class TestRigidBody
{
public:
  //Basic rigid body variables.
  /*
  LINEAR
  */
  Vector3 mPosition;
  float mInvMass;
  Vector3 mLinearVelocity;
  Vector3 mForces;

  /*
  Angular
  */
  Quaternion mOrientation;
  Matrix4 mLocalInvInertia;
  Vector3 mAngleVelocity;
  Vector3 mTorques;

  //Custom optimisation parameters.
  Matrix4  mMatWorld;
  Matrix4  mWorldInvInertia;
  float  mRadius;

  //Cube variables.
  Vector3  mLocalxyz[3]; // Local x-y-z Axes
  Vector3  mHalfExtends; // Positive halfwidths along
       each axis

  //--------------------------------------------
  //--------------------------------------------
```

8 Experimental Results

Method	Number of Objects	FPS
Brute Force	0	35
Brute Force	100	32
Brute Force	200	28
Brute Force	300	26
Brute Force	400	24
Brute Force	500	20
Brute Force	600	20
Brute Force	700	18
Brute Force	800	15
Brute Force	900	12
Brute Force	1000	10
Brute Force Average	——	21.8
Broad Phase	0	35
Broad Phase	100	35
Broad Phase	200	34
Broad Phase	300	32
Broad Phase	400	31
Broad Phase	500	28
Broad Phase	600	28
Broad Phase	700	27
Broad Phase	800	26
Broad Phase	900	25
Broad Phase	1000	23
Broad Phase Average	——	29.6

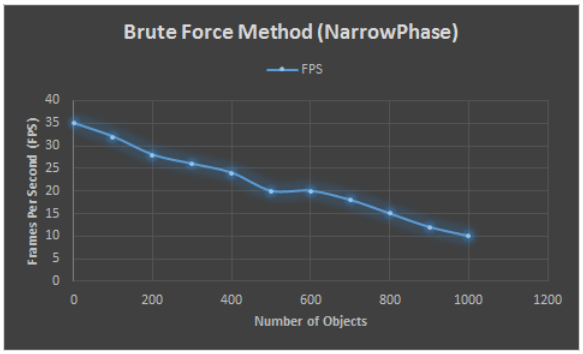

Figure 11: *Results from just using narrow phase technique.*

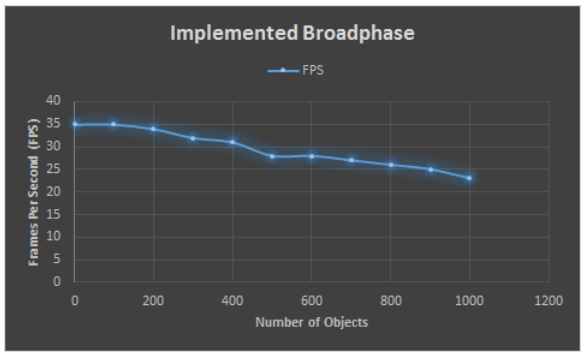

Figure 12: *Results from using broadphase technique.*

This data shows that with the implementation of the broad phase instead of just using brute force to calculate collisions there is an increase in the simulations performance. A time step of 0.01 was used in this simulation.

Acknowledgements

To Ben Kenwright for the initial learning material.

Blog

`http://graybostephano.wix.com/gamedevprogress`

References

ERAN GUENDELMAN, ROBERT BRIDSON, R. F. Nonconvex rigid bodies with stacking. `http://dl.acm.org/citation.cfm?id=882358`.

MIRTICH, B., AND CANNY, J. Impulse-based simulation on rigid bodies. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.6207&rep=rep1&type=pdf`.

MOORE, M., AND WILHELMS, J. Collision detection and collision response for computer animation. `http://www.cs.princeton.edu/courses/archive/spring01/cs598b/papers/moore88.pdf`.

ROBERT BRIDSON, R. F., AND ANDERSON, J. Robust treatment of collisions. `http://dl.acm.org/citation.cfm?id=566623`.