CE1051A Coursework - 3D Rigid Body Dynamics

Jake Morey
BSc(Hons) Computer Games Technology,
School of Arts, Media and Computer Games
University of Abertay Dundee
100424@live.abertay.ac.uk

January 12, 2014

Contents

1 Introduction

The purpose of this project is to create an application that involves 3D rigid body dynamics more specifically an application that simulates rigid body general motion when an impulse force is added to the object at a particular location. This application will take the users mouse input and draw a ray to the object on screen. Once the user clicks the mouse a force will be applied to the object at the location of the mouse. This force will be instant and will cause the cube to move in the game world. Before talking more about the actual mechanics of the application the next sections will talk about so of the background equations that are needed in general motion of rigid bodies.

2 Physics

2.1 Rigid Body:

A rigid body can be described as a system of particles in which each particle remains the same distance from every other particle in that system. This means that the shape retains its shape and size whilst moving or that if there are any changes it is so small it is negligible or can be safely neglected. Also because the particles don't move within the system the objects distribution of mass is kept (Bourg and Bywalec, 2013; MacTaggart, 2013).

2.2 Centre Of Mass:

The centre of mass or the centre of gravity is the point in an object when cut in two will result in two objects with the same weight (Millington, 2007). Millington (2007) also describes how to calculate the centre of mass. The object must be split into particles so that the average position can be calculated using equation 2.1.

$$G = \frac{1}{M} \sum_{n} p_i m_i \tag{2.1}$$

Where G is the position of the centre of mass, M is the mass of the entire object, p_i is the position of the particle and m_i is the mass of each particle.

The centre of mass for any 3D object with uniform density can be found by performing a triple integral on the each separate coordinate component and then dividing by the mass of the entire object. The triple integrals for the xyz components can be seen in equations 2.2 - 2.4. Equation 2.5 shows the position vector of the centre of mass compiled from the results of equations 2.2 - 2.4.

$$G_x = \frac{1}{M} \int\int\int x \; dxdydz \tag{2.2}$$

$$G_y = \frac{1}{M} \int\int\int y \; dxdydz \tag{2.3}$$

$$G_z = \frac{1}{M} \int\int\int z \; dxdydz \tag{2.4}$$

$$\mathbf{G} = (G_x, G_y, G_z) \tag{2.5}$$

For certain shapes such as cuboids and spheres with uniform density the centre of mass is the centre of the shape (Millington, 2007). Millington (2007) describes the reason why the centre of mass is so important to rigid body physics is because "By selecting the center of mass as our origin position we can completely separate the calculations for the linear motion of the object (which is the same as for particles) and its angular motion." In the application there is only one object and because it is a cube the centre of mass is always going to the geometric centre of the object. This means the previous calculation for working out the centre of mass is won't be needed. However if other objects were added the centre of mass may need to be calculated.

2.3 Inertia Tensor:

The inertia tensor, which is sometimes called the mass matrix, is a 3x3 matrix which hold the characteristics of a rigid body. This including the moment of inertia for each of its axes stored along the diagonal of the matrix and the products of

inertia (Millington, 2007). Millington (2007) describes the products of inertia as the "tendency of an object to rotate in a direction in which torque is being applied." An example of the phenomena is given as a child's top that starts spinning in one direction but will then suddenly jump upside and spin in another direction (Millington, 2007).

Equation 2.6 is the inertia tensor matrix with the previously mention diagonal moments of inertia and the remaining values representing the products of inertia. A, B, C or equation 2.7 are the moments of inertia, while F, G, H are the products of inertia or equation 2.8.

$$\begin{bmatrix} A & -H & -G \\ -H & B & -F \\ -G & -F & C \end{bmatrix} \tag{2.6}$$

$$A = \sum \rho \left(y^2 + z^2\right) \qquad B = \sum \rho \left(x^2 + z^2\right) \qquad C = \sum \rho \left(x^2 + y^2\right) \tag{2.7}$$

$$F = \sum \rho \left(yz\right) \qquad G = \sum \rho \left(xz\right) \qquad H = \sum \rho \left(xy\right) \tag{2.8}$$

Equation 2.9 - 2.11 show the triple integrals needed to calculate the inertia tensor for any object that has uniform density.

$$A = \iiint_V \rho(y^2 + z^2)\ dxdydz \qquad B = \iiint_V \rho(x^2 + z^2)\ dxdydz \tag{2.9}$$

$$C = \iiint_V \rho(x^2 + y^2)\ dxdydz \qquad F = \iiint_V \rho(yz)\ dxdydz \tag{2.10}$$

$$G = \iiint_V \rho(xz)\ dxdydz \qquad H = \iiint_V \rho(xy)\ dxdydz \tag{2.11}$$

The inertia tensor and angular velocity are used in The Euler Equations to calculate the amount of torque needed to have an object rotate about a pivoted part of itself (**?**). The equations to achieve this are equations 2.12 - 2.14 with A, B and C representing the principal moments of inertia, ω_1, ω_2, ω_3 being the xyz components of the angular velocity $\boldsymbol{\omega}$, and Γ_1, Γ_2, Γ_3 are the components of the

total moments of all the forces Γ.

$$A\dot{\omega_1} - (B - C)\omega_2\omega_3 = \Gamma_1 \tag{2.12}$$

$$B\dot{\omega_2} - (C - A)\omega_3\omega_1 = \Gamma_2 \tag{2.13}$$

$$C\dot{\omega_3} - (A - B)\omega_1\omega_2 = \Gamma_3 \tag{2.14}$$

These equations are not used in the application and would only be used if functionality were to be added which would allow the user to set a point on the object which is fixed so that it would pivot around it.

2.4 General Motion of a Rigid Body:

General motion of a rigid body is what the application for this project aims to show. It uses general motion of a rigid body to move the object once an impulse force has been applied. As mentioned briefly in section 2.2, motion can be thought of as two parts. The linear motion of the centre of mass in an inertial frame which in the case of the application would be movement in the world co-ordinates. The next part of general motion would be rotational motion of the object about an axis that passes through the centre of mass.

Below there are three equations (2.15 - 2.17) that represent how to calculate the position, velocity, and acceleration of any point of an object at any moment in time. To start the axes of the body must be set so that it aligns with the principle axes at the centre of mass, and ω must not change no matter what origin is used. Also $\tilde{\mathbf{p}}$ which is the position vector of the point relative to the centre of mass has to be expressed by the inertial frame co-ordinates which in the application are known as world co-ordinates.

$$\mathbf{p} = \mathbf{g} + t\mathbf{V} + R\tilde{\mathbf{p}_0} \tag{2.15}$$

$$\mathbf{v} = \mathbf{V} + \boldsymbol{\omega} \times R\tilde{\mathbf{p}_0} \tag{2.16}$$

$$\mathbf{a} = \mathbf{A} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times R\tilde{\mathbf{p}_0}) \tag{2.17}$$

Equation 2.15 is the equation for calculating a position of a point at any time. g represents the centre of mass for the object, while t is the time taken, and $\mathbf{V}$ is the velocity of the centre of mass. $\tilde{\mathbf{p}_0}$ is the initial value of $\tilde{\mathbf{p}}$ which as mentioned before is the position vector between the point and the centre of mass. Finally R is a standard rotation matrix by amount θ about an unit vector axis $\hat{\mathbf{v}} = \alpha\hat{\mathbf{i}} + \beta\hat{\mathbf{j}} + \gamma\hat{\mathbf{k}}$ passing through the origin. θ is determined by ωt with the ω being equal to $|\boldsymbol{\omega}|$. The C in the matrix is a shorthand for $1 - \cos\theta$.

$$R = \begin{bmatrix} \alpha^2(C) + \cos\theta & \alpha\beta(C) - \gamma\sin\theta & \alpha\gamma(C) + \beta\sin\theta \\ \alpha\beta(C) + \gamma\sin\theta & \beta^2(C) + \cos\theta & \beta\gamma(C) - \alpha\sin\theta \\ \alpha\gamma(C) - \beta\sin\theta & \beta\gamma(C) + \alpha\sin\theta & \gamma^2(C) + \cos\theta \end{bmatrix} \tag{2.18}$$

The only new symbol in equation 2.16 or the velocity equation compared with equation 2.15 is $\boldsymbol{\omega}$ which is the angular velocity vector. In the final equation (2.17) the A is the acceleration of the centre of mass.

Like simple 2D acceleration, velocity, and position there is a relationship between acceleration, velocity, and position of rigid bodies. This relationship can result in the equations 2.15 - 2.17 being written slightly differently. Because acceleration is a derivative of velocity over time, the acceleration equation (2.17) can be integrated with respect to time to create the equation 2.19 which differers only slightly from the previous velocity equation (2.16).

$$\mathbf{v} = \mathbf{A}t + \boldsymbol{\omega} \times R\tilde{\mathbf{p}_0} \tag{2.19}$$

Also because velocity is a derivative of position with respect to time, the previous velocity equation (2.19) can be integrated again with respect to time to get equation 2.20, with $\mathbf{g}$ being the centre of mass.

$$\mathbf{p} = \frac{1}{2}\mathbf{A}t^2 + R\tilde{\mathbf{p}_0} + \mathbf{g} \tag{2.20}$$

The equation 2.20 will be used when checking the accuracy of the application be calculating the position of a point on the object after a force is applied.

3 Simulation

3.1 Application:

As briefly described in the introduction this application was created to show what happens when a force is applied to a 3D Rigid Body. In the case of this application this body is a cube with a mass of 1 kg, sides of length 5, and a centre of mass at $(0, 2.5, 0)$. To do this the application was made using Unity Technologies (2013) Unity3D and it's built in physics engine.

The application allows the user to move around using the WASD keys and the mouse. Moving the mouse moves the direction of the ray coming from the camera hitting the box. This ray is the position the force will be applied to on the object. To apply the force the user clicks the left mouse button while the looking at the cube.

As seen in figure 3.1 the application outputs the location of the centre of mass at the top right of the screen as well as the position of the 8 vertices just below this. At the bottom left hand side of the screen the user can see the vector for the position at which the force is applied to on the object. While on the other side of the screen at the top right the angular velocity is outputted once it is calculated by Unity3D. At the middle of the right hand side of the screen are three variables needed for calculating the motion of rigid bodies; time, force, and mass. Time here is displayed in seconds, mass in kilograms, and force is displayed in newtons. The bottom right hand of the screen displays the direction the force is applied in.

3.2 Code:

In figure 3.2 there is a snippet of the code used to create the application. The code shows the process described earlier in which the user presses the left mouse

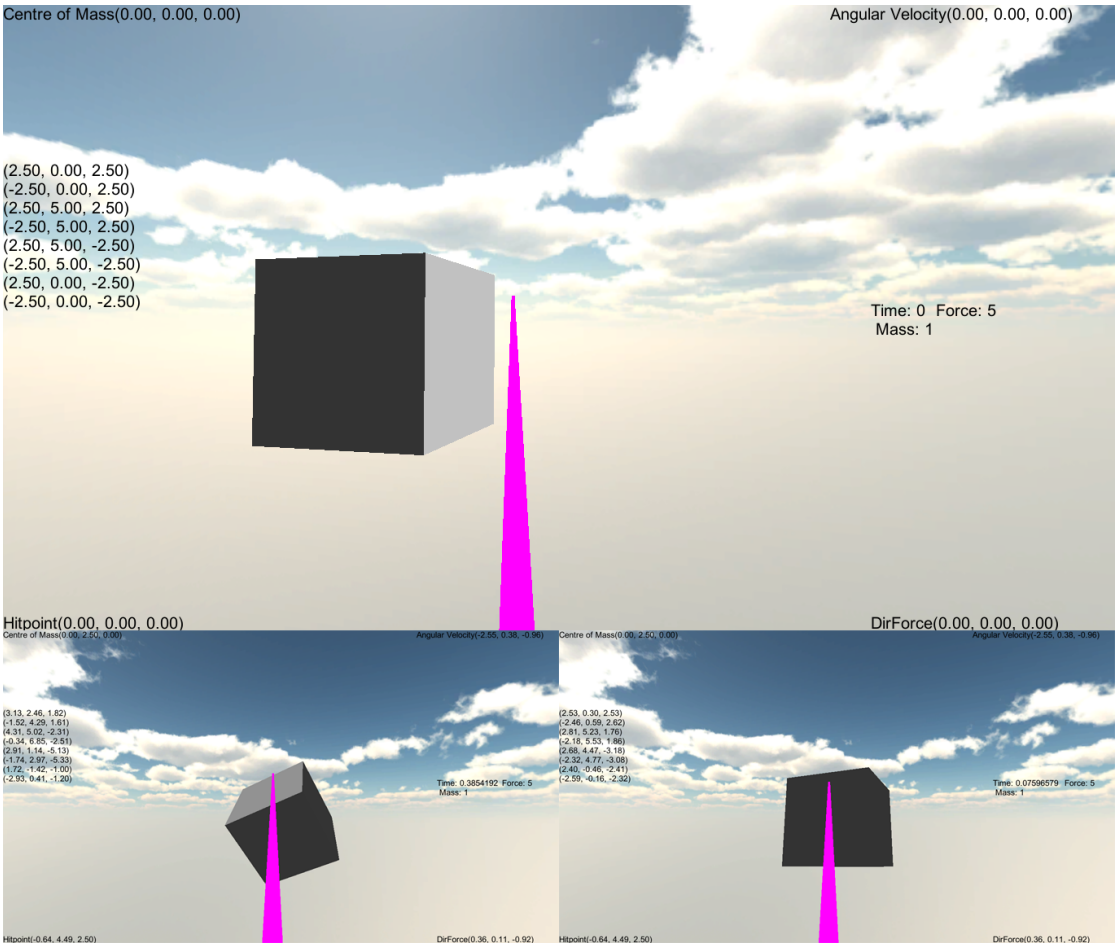

Figure 3.1: Simulation Screenshots

button to start the movement of the object. Parts of the code have been removed because it is not relevant to the motion of the object. These lines of code follow and comments which start with /* and ends with */.

```
if (Input.GetMouseButton (0) && isPressed == false){
        //Cast a Ray forwards and check to see if it hits the object.
        Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
        RaycastHit hit;
        //If the ray hits and object initalise the variables.
        if (Physics.Raycast (ray, out hit, distance)){
                /*Start the timer and disable pressing the mouse button again.*/
                //Calculate the force * direction of force.
                Vector3 forceVector = (force* ray.direction);
                //Add force to force at a particular location on object. Use
                    Impulse so its an instantaneous force.
                rigidbody.AddForceAtPosition(forceVector, hit.point,
                    ForceMode.Impulse);
                /*Collect direction of force and the position the force hits.*/
                /*Output a screenshot.*/
        }
}
```

Figure 3.2: Code Snippet

The relevant part of the code starts where the force vector is calculated. The force vector is calculated by multiplying the force variable by the vector containing the direction of the ray that has been cast from the camera to the object. This vector is always normalized so this calculation doesn't need to be performed. This force vector is then applied to the rigid body at the position the user is looking at using a function provided by Unity3D. This function also takes a third variable which tells the function that the type of force added is an impulse force and should happen instantaneously.

4 Review

4.1 Comparison:

To see how well the application is at simulating general motion of rigid bodies, this section will look to compare the position the application has after a moment of time has occurred to the position calculated by hand at the same moment. Figure

4.1 shows the cube after 0.526 seconds. This time will be used when calculating

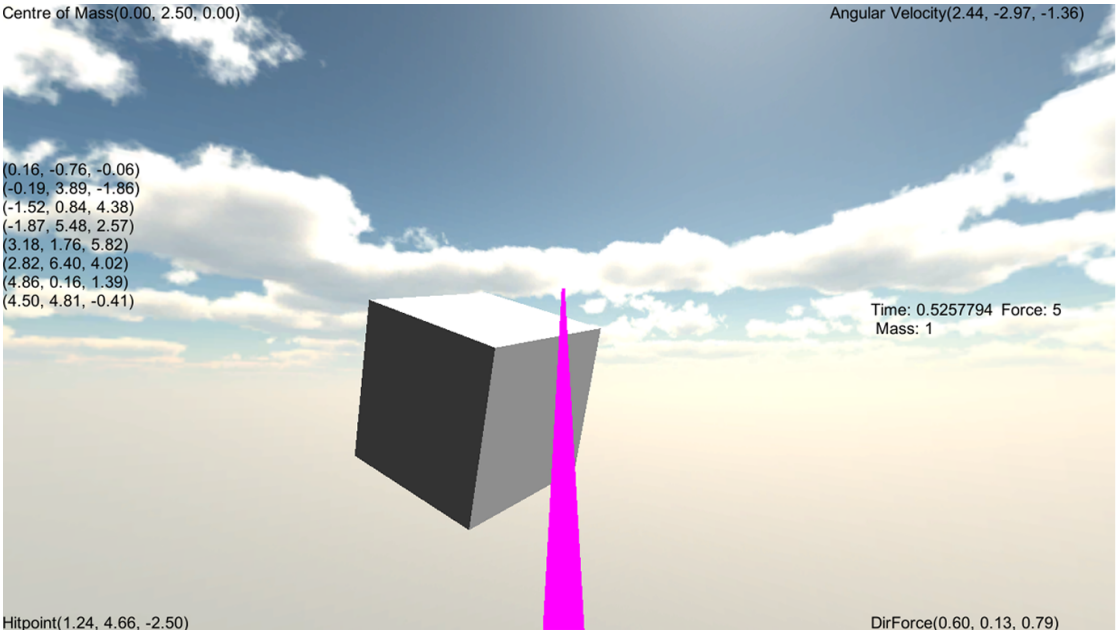

Figure 4.1: Screenshot of Cube at $t = 0.526s$

position by hand using Equation 4.1 because in this scenario force is available meaning the acceleration can be worked out using Newton's Second Law $F = MA$.

$$\mathbf{p} = \frac{1}{2}\mathbf{A}t^2 + R\tilde{\mathbf{p}}_0 + \mathbf{g} \tag{4.1}$$

For the hand calculation the angular velocity will be represented by $\boldsymbol{\omega}$ and will be taken from the application. The force direction vector will also be taken form the application and will be represented by F. Also shown in 4.2 is $\mathbf{P}$ and this represents the initial position, and $\mathbf{g}$ which is the centre of mass. Equations 4.3 show three of the variables used M beings the mass of the object, f is the force applied to the object, and t is the time since the initial force is applied.

$$\boldsymbol{\omega} = \begin{bmatrix} 2.44 \\ -2.97 \\ -1.36 \end{bmatrix} \qquad \mathbf{F} = f \begin{bmatrix} 0.6 \\ 0.13 \\ 0.79 \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} 2.5 \\ 0 \\ 2.5 \end{bmatrix} \qquad \mathbf{g} = \begin{bmatrix} 0 \\ 2.5 \\ 0 \end{bmatrix} \tag{4.2}$$

$$f = 5N \qquad t = 0.526s \qquad M = 1kg \tag{4.3}$$

$$\omega = |\boldsymbol{\omega}| = 4.077 \qquad \theta = \omega t = 2.144 \tag{4.4}$$

To start first acceleration will have to be calculated this will be represented by **A**. It will be calculated as mentioned before using Newton's Second Law $F = MA$. This means $\mathbf{A} = \mathbf{F}/M$ and results in Equation 4.5.

$$\mathbf{A} = \frac{f}{M}\begin{bmatrix} 0.6 \\ 0.13 \\ 0.79 \end{bmatrix} = 5\begin{bmatrix} 0.6 \\ 0.13 \\ 0.79 \end{bmatrix} = \begin{bmatrix} 3 \\ 0.65 \\ 3.95 \end{bmatrix} \tag{4.5}$$

Next needs to be calculated, this is done by calculating the position vector of the point relative to the centre of mass, **g**, this can be shown in Equation 4.6.

$$\tilde{\mathbf{p}_0} = \mathbf{P} - \mathbf{g} = \begin{bmatrix} 2.5 \\ 0 \\ 2.5 \end{bmatrix} - \begin{bmatrix} 0 \\ 2.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2.5 \\ -2.5 \\ 2.5 \end{bmatrix} \tag{4.6}$$

The next step is to calculate the matrix R, which is a standard rotational matrix about any axis. Equation 4.7 is the initial matrix that will be used. C is shorthand for $(1 - \cos\theta)$. Below the rotational matrix are the values for α, β, γ in Equation 4.8 and in Equation 4.9 are the values of $\cos\theta, \sin\theta, (1 - \cos\theta)$.

$$R = \begin{bmatrix} \alpha^2(C) + \cos\theta & \alpha\beta(C) - \gamma\sin\theta & \alpha\gamma(C) + \beta\sin\theta \\ \alpha\beta(C) + \gamma\sin\theta & \beta^2(C) + \cos\theta & \beta\gamma(C) - \alpha\sin\theta \\ \alpha\gamma(C) - \beta\sin\theta & \beta\gamma(C) + \alpha\sin\theta & \gamma^2(C) + \cos\theta \end{bmatrix} \tag{4.7}$$

$$\alpha = \frac{\boldsymbol{\omega}_x}{\omega} = 0.5984 \qquad \beta = \frac{\boldsymbol{\omega}_y}{\omega} = -0.7284 \qquad \gamma = \frac{\boldsymbol{\omega}_z}{\omega} = -0.3336 \tag{4.8}$$

$$\cos\theta = -0.5421 \qquad \sin\theta = 0.8403 \qquad 1 - \cos\theta = 1.5421 \tag{4.9}$$

After substituting $\alpha, \beta, \gamma, \cos\theta, \sin\theta$, and $(1-\cos\theta)$ into the matrix the result can be seen in Matrix 4.10. Now multiply R and $\tilde{\mathbf{p}_0}$ together to get Vector 4.11.

$$R = \begin{bmatrix} 0.0102 & -0.3919 & -0.9199 \\ -0.9525 & 0.2761 & -0.1282 \\ 0.3043 & 0.8776 & -0.3705 \end{bmatrix} \tag{4.10}$$

$$R\tilde{\mathbf{p}_0} = \begin{bmatrix} 0.0102 & -0.3919 & -0.9199 \\ -0.9525 & 0.2761 & -0.1282 \\ 0.3043 & 0.8776 & -0.3705 \end{bmatrix} \begin{bmatrix} 2.5 \\ -2.5 \\ 2.5 \end{bmatrix} = \begin{bmatrix} -1.2945 \\ -3.3921 \\ -2.3596 \end{bmatrix} \tag{4.11}$$

$R\tilde{\mathbf{p}_0}$, $\mathbf{A}$, and t^2 can be substituted back into Equation 4.1 to find the final position of the chosen point.

$$\mathbf{p} = \frac{1}{2} \begin{bmatrix} 3 \\ 0.65 \\ 3.95 \end{bmatrix} 0.526^2 + \begin{bmatrix} -1.2945 \\ -3.3921 \\ -2.3596 \end{bmatrix} + \begin{bmatrix} 0 \\ 2.5 \\ 0 \end{bmatrix} \tag{4.12}$$

$$\mathbf{p} = \begin{bmatrix} 0.4145 \\ 0.0898 \\ 0.5158 \end{bmatrix} + \begin{bmatrix} -1.2945 \\ -3.3921 \\ -2.3596 \end{bmatrix} + \begin{bmatrix} 0 \\ 2.5 \\ 0 \end{bmatrix} \tag{4.13}$$

$$\mathbf{p} = \begin{bmatrix} -0.8801 \\ -0.8024 \\ -1.8138 \end{bmatrix} \tag{4.14}$$

The final result for $\mathbf{p}$ is $(-0.88, -0.8, -1.81)$ when compared to the application vector of $(0.16, -0.76, -0.06)$ an error vector of $(1.04, 0.04, 1.75)$ can be calculated. This vector is calculated by subtracting the calculated position vector from the applications position vector to show the difference between the two. Using screenshots taking during the application, which can be seen in figure 4.2, more calculations can be done using all the same variables except for the time variable

to see if the application is constantly different from the calculated result.

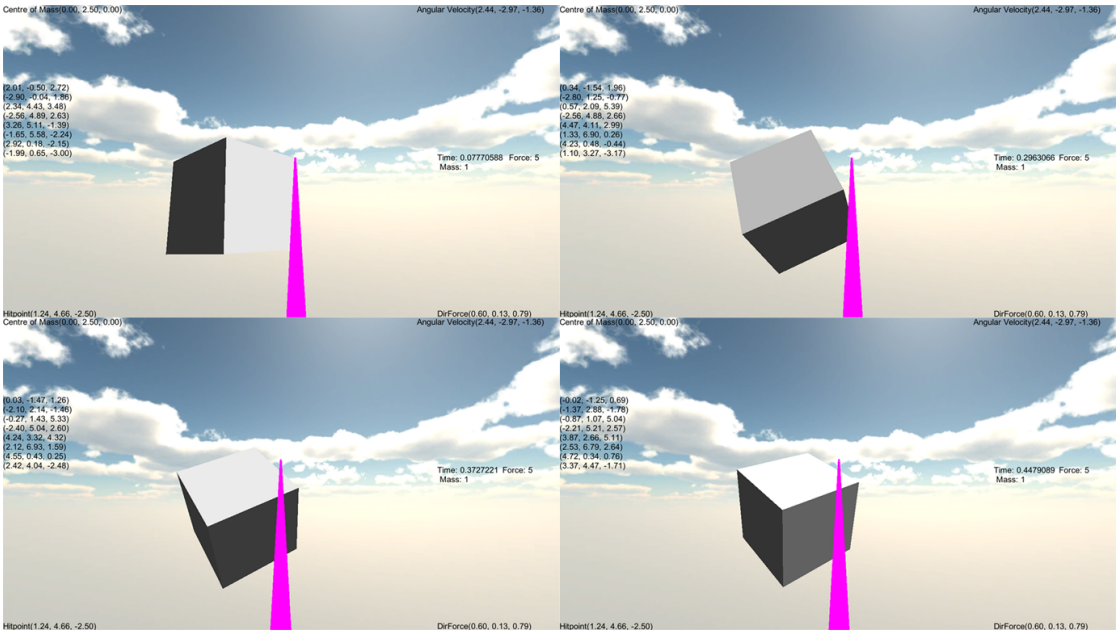

Figure 4.2: Screenshot of Cube at $t = 0.078s$, $t = 0.296s$, $t = 0.373s$, $t = 0.498s$

The Table 4.1 shows the results from performing the previous calculations on different time steps both by hand and by the application. Looking at the error vectors it can be seen that as the application continues the error between the application and the calculated results increase.

Table 4.1: Position Comparison

t (seconds)	Calculated Position	Application Position	Error Vector
0.078	$(1.63, -0.69, 2.45)$	$(2.01, -0.50, 2.72)$	$(0.38, 0.19, 0.27)$
0.296	$(-0.50, -1.70, 0.83)$	$(0.34, -1.54, 1.96)$	$(0.84, 0.16, 1.13)$
0.373	$(-0.91, -1.62, -0.06)$	$(0.03, -1.47, 1.26)$	$(0.94, 0.16, 1.32)$
0.448	$(-1.04, -1.32, -0.96)$	$(-0.02, -1.25, 0.69)$	$(1.02, 0.07, 1.65)$
0.526	$(-0.88, -0.80, -1.81)$	$(0.16, -0.76, -0.06)$	$(1.04, 0.04, 1.75)$

There could be a number of different reasons which accumulated to give this error. For example it could be because of the way the physics engine in Unity3D handles rigid bodies but due to Unity3D being proprietary software it would be impossible to know exactly how the physics engine solves the problem or if any sacrifices where made to make performance fast and looking right compare to being actually physically realistic. Also some of the errors could have come from

rounding errors that follow through the equations, or from the possibility that the is timer not completely in sync with the physics engine in the application.

4.2 Conclusion:

Looking at the accuracy of the application compared to the hand calculated results the Unity3D game engine might not have been the best way to create the simulation. A better way could have been to create the application for scratch for a number of reasons. The first being that it could have given the user better control over physics of the application for example allowing the user to manually set the angular velocity, or the time steps to take screenshots at. Another reason would be because it would allow for better debugging because output information code be provided for every stage of linear and angular motion.

As well as possibly changing how the application was developed if the project was under taken again a few other improvements could have been made to the application. A major one would be add the functionality to change the object the force is being applied to or by adding multiple objects. Instead of just changing what object is being affected the user could be allowed to draw and create objects in real time and see what effects the force being applied has on it. This would mean needing to work out the centre of mass of the object once it is created and the equations for working out the centre of mass from section 2.2 would be helpful in doing so.

Overall the project was fairly successful, there were places were improvements could be made such as in the accuracy of the final results. However the application meet the specification of showcasing general motion of a rigid body when a force is applied at a particular point on the object.

References

Bourg, D. M. and Bywalec, B. 2013. *Physics for Game Developers: Science, Math, and Code for Realistic Effects*. O'Reilly Media, Inc. 2.1

MacTaggart, D. 2013. *Applications of Integration & Basic 3D Motion*. UAD - CE11051A. 2.1

Millington, I. 2007. *Game physics engine development*. Taylor & Francis US. 2.2, 2.2, 2.3

Unity Technologies. 2013. *Unity3D* [software] Version no 4.3. Available from: http://unity3d.com/. 3.1