

Masana Ikeshima
BSGS3_1
Mathematics – MATM346
Coursework 3
Matric Number 200601037

Contents

1.	Terms of Reference	3
2.	Abstract	3
3.	Introduction	3
a)	Overview.....	3
b)	Structure of Report.....	3
4.	Overall Design	4
5.	Time Evolution Procedure	5
a)	Cannon Ball	5
i.	Initial Position.....	5
ii.	Initial Velocity.....	5
iii.	Acceleration	5
iv.	Euler's Method	5
b)	Cube	6
6.	Event Modelling.....	7
8.	Conclusion.....	9
9.	Appendix I.....	10
10.	Appendix II	12
a)	Vertex Matrix.....	12
b)	Adjacency Matrix.....	12
c)	Face Matrix.....	12

1. Terms of Reference

This coursework was undertaken by Masana Ikeshima (mikesh10@caledonian.ac.uk) as part of the Mathematics for 3D Games (MATM346) Development course at Glasgow Caledonian University.

2. Abstract

This report describes the use of a differential equation model, developed using Newton's second law to determine where the ball will hit a moving cube. The ball is projected at the cube using angles and a velocity specified by the user. Euler's method will be used to approximate the movement of both the ball and the cube.

3. Introduction

a) Overview

This document details the mathematics behind creating the trajectory of a ball using Newton's second law. The ball has a fixed mass and is subject to the force of gravity during its transformation from the cannon to the target – a rotating cube. The initial launch velocity and angles are determined by the user using an input system.

The cube is created using multiple matrices – vertex, adjacency and face, the cube will be moving along a given trajectory specified in the brief.

To calculate if the ball has collided with the cube Euler's method will be implemented. The position of the ball and the cube can then be calculated at regular time intervals. Depending on whether the ball collides with the cube before it completes its trajectory, the rotation of the cube will slow down. If the ball fails to collide with the ball the rotation is sped up.

b) Structure of Report

This report shows how the differential equations used to model the ball and cube trajectories were derived. Thereafter it will describe how collision detection is calculated and how the collision affects the cubes rotation.

An annotated appendix is also included, which provides pseudo-code to implement the game and a mathematical model appendix explaining the derivation of the model and its accuracy.

4. Overall Design

Assumptions

The overall design of the game involves a simple ball being fired from a cannon to see if it will collide with a rotating cube. The specification, however fails to specify certain criteria with regards to the balls weight, the gravitational pull and the balls dimensions, thus the following assumptions have been made; the weight has been assumed to be 1 Kg, the gravitational pull is set to 10N and the diameter is set to 0.5m. There is no air resistance acting upon it.

Similarly, there is no definition of where the cannon is set, so it is assumed that the cannon is in line with the x axis (see Figure 1) and the thickness of the cannon wall has been assumed that it is of negligible thickness.

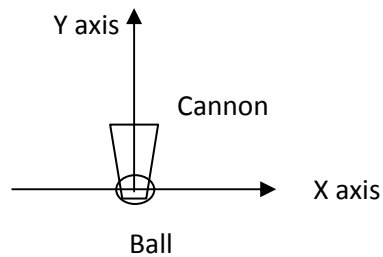


Figure 1 – Position of the ball and cannon

With regards to the cube the only assumption made is that there is no restitution acting on the ball once a collision occurs. The cube's trajectory and rotation speed has already been specified.

5. Time Evolution Procedure

Euler's Method & Cube Movement

The time evolution procedure is used to calculate how the ball and the cube translate from one position to another taking into account its acceleration, current velocity and its position.

a) Cannon Ball

In order to obtain the path of the ball, we must use Euler's method. In order to use Euler's method we must first obtain the ball's initial position, then using the information assumed and provided, we can create a model for its acceleration, and its velocity.

i. Initial Position

The balls original position has been determined to be:

$$\begin{aligned}x(0) &= 0 \\y(0) &= 0 \\z(0) &= 0.5\end{aligned}$$

The reason that the ball is 0.5 above the z axis is due to the balls radius.

ii. Initial Velocity

There are two angles acting upon the direction of the ball. The first is XY direction, which is calculated using θ_h . The other angle is in the YZ direction, using θ_v .

$$\begin{aligned}\dot{x}(0) &= V \sin \theta_h \\ \dot{y}(0) &= V(\cos \theta_h \times \cos \theta_v) \\ \dot{z}(0) &= V \sin \theta_v\end{aligned}$$

iii. Acceleration

The forces acting on the ball using Newton's second law are:

$$m \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix} + \begin{pmatrix} -4 \\ 0 \\ 0 \end{pmatrix}$$

Since the z axis is acting in the up-down direction weight is pulling the object downwards. The -4N acting on the x axis is the wind.

iv. Euler's Method

In order to use Euler's method, we must first introduce new variables u, v and w.

$$\begin{array}{lll}u = \dot{x} & u = V \sin \theta_h & \dot{u} = -4 \\v = \dot{y} & v = V(\cos \theta_h \times \cos \theta_v) & \dot{v} = 0 \\w = \dot{z} & w = V \sin \theta_v & \dot{w} = -10\end{array}$$

$$\begin{aligned}
 x_{n+1} &= x_n + \tau(u_n) & u_{n+1} &= u_n + \tau(\dot{u}_n) \\
 y_{n+1} &= y_n + \tau(v_n) & v_{n+1} &= v_n + \tau(\dot{v}_n) \\
 z_{n+1} &= z_n + \tau(w_n) & w_{n+1} &= w_n + \tau(\dot{w}_n)
 \end{aligned}$$

b) Cube

The cube has been specified to move along a given trajectory of:

$$\begin{pmatrix} -20 \\ 20 \\ 20 - \frac{(\tau - 20)^2}{20} \end{pmatrix}$$

The cube's movement will be based around an iterative system similar to Euler's method. The trajectory matrix is added to the position matrix to achieve the new position of the cube. Thereafter the rotation cube is multiplied by the new position matrix, with t_2 being the rotations per second.

The cube has also been specified to rotate parallel to the xy plane. This is given by:

$$\tau_2 \begin{pmatrix} \cos \theta_c & \sin \theta_c & 0 & 0 \\ -\sin \theta_c & \cos \theta_c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The rotational matrix has been defined as above, as is multiplied by a timestamp that is different to the translation matrix due to the fact that the value will change depending on the aftermath of the collision between ball and cube.

6. Event Modelling

Firing Projectile

In order to start the game application, the user will be required to specify the angle and speed of the ball, then fire the ball by pressing the spacebar.

Collision Detection

The event taking place within the game will be the collision of the ball hitting the cube. It has been stated that if the ball hits the cube on the specified face the game will be over, otherwise if the ball hits the cube on any other face the rotation will slow down by 10 degrees per second and if the ball does not hit the cube it will increase by 5 degrees per second.

In order to basic collision detection, we must take the current position of the ball and the cubes vertex position and compare them. However, as the ball is going to be measured from the centre, we must also take into account its radius. We will use $x+0.25^1$.

Aftermath

Depending on whether the ball hit the cube will result in one of three actions:

- If the ball hits the cube on the specified target the game will be over and the player has won.
- If the ball hits the cube but not on the current face, the cube's rotation speed will decrease by 10 degrees per second.
- If the ball doesn't hit the cube at all, the cube's rotation will speed up by 5 degrees per second.

¹ $\frac{1}{2}$ diameter

7. Overview of Implementation

Design Code

The code will have to be broken down into several modules in order to be implemented in an organised structure. The following are the modules that would have to be created:

- Start program
- Initial value prompt
- Begin program
- Euler's method and cube movement
- Update position
- Collision detection
- Aftermath of collision
- End program

Start program

This will be a standard function that will bring in all necessary initial variables and their values, as well as a visual representation if there will be one.

Initial value prompt

Prompt the user to provide the program with the ball's initial velocity as well as its horizontal and vertical angles.

Begin Program

Await the user to fire the ball by pressing the spacebar.

Euler's method and cube movement

Euler's method will be used to obtain the movement of the ball, its velocity and acceleration. The cube movement has already been specified to move along a given trajectory.

Update position

Once the new positions of the ball and cube have been determined, they will be updated simultaneously.

Collision detection

In order to determine whether the ball has collided with the cube, the code will compare the position of the ball with each of the cubes vertices. If there are no collisions, then it will take the balls position and add the ball's radius to it and compare again.

Aftermath of collision

Depending on if the ball has not hit the cube, it will rotate 5 degrees per second faster, otherwise it will rotate 10 degrees per second slower.

End program

If the ball has hit the cube on the specified surface, then it will be game over as the player has won. The game will also be over if the cube has completed its trajectory.

8. Conclusion

Design Issues

In conclusion the detailed documentation provides the programmer with more than adequate detail to allow him to understand and implement the game into any platform. The only problems that arose were due to time-constraints as it was not feasible to get a fully and adequate collision detection functioning for the particular face on the cube. However, the ball will be able to determine if it has collided with the cube by checking its position against the cubes.

Furthermore, the movement of the ball isn't as accurate as could be. The reason for this is that the ball would have to move in a fixed linear movement when being fired from the back of the cannon to the exit, which is not taken into account in this physics model. As there is no air resistance acting on the ball as well, it won't be as realistically accurate.

9. Appendix I

Pseudo-Code

```
//Main program
1.1 Start Program
1.1.1 Create ball
1.1.2 Create cube

//Prompt the user for the necessary variable values
2.1 Prompt user for ball variables
2.2.1 Ask for ball velocity
2.2.2 Ask for ball horizontal angle
2.2.3 Ask for ball vertical angle

//If the ball is already in play then alert the user, otherwise start the game
2.3.1 IF space has been pressed AND ball is dead then
2.3.2 Start movement
2.3.3 Set ball to not dead
2.3.4 Call Time Evolution
2.3.5 Call Event Modelling
2.3.6 End IF
2.3.7 Else IF space has been pressed AND ball is not dead then
2.3.8 Alert user
2.3.9 END IF

//Function - Time Evolution
3.1 Euler's Method

//While the ball is still in play, obtain the x, y and z coordinate, velocity and
acceleration of the ball
3.1.1 WHILE ball is alive
3.1.2 Obtain balls current position (x,y,z)
3.1.3 Obtain balls current velocity (x,y,z)
3.1.4 Obtain balls current acceleration (x,y,z)

//Obtain the new position and velocity, by adding the new acceleration or velocity
and multiply by the timestamp
3.1.5 Get new x position
3.1.5.1 Old x position + timestamp (Old x velocity)
3.1.6 Get new y position
3.1.6.1 Old y position + timestamp (Old y velocity)
3.1.7 Get new z position
3.1.7.1 Old z position + timestamp (Old z velocity)
3.1.8 Get new x velocity
3.1.8.1 Old velocity + timestamp (x acceleration)
3.1.9 Get new y velocity
3.1.9.1 Old velocity + timestamp (y acceleration)
3.1.10 Get new z velocity
3.1.10.1 Old velocity + timestamp (z acceleration)

//Function - Update position
4.1 Update position
4.1.1 Update position of ball and cube
```

```

//Check for overlapping – i.e. if ball overlaps the cube
5.1 Check for collision detection

//Compare values using the ball’s centre
//If the ball has collided with the cube then set it to dead
5.1.1    FOR number of cube vertices
5.1.1.1    Compare ball position to cube vertex
5.1.1.2    IF ball is further than cube vertex THEN
5.1.1.2.1    Set hit as true
5.1.1.2.2    END IF
5.1.1.3    END FOR

//Compare values using the furthest point of the ball
5.1.4    Ball position + radius
5.1.5    FOR number of cube vertices
5.1.5.1    Compare ball position to cube vertex
5.1.5.2    IF ball is further than cube vertex THEN
5.1.5.2.1    Set hit as true
5.1.5.2.2    END IF
5.1.5.3    END FOR

//Checks to see if the ball has fallen to the ground
5.1.6    IF balls z position is equal to 0.25 AND x is > 0 THEN
5.1.6.1    Set hit as false
5.1.6.2    END IF

//Aftermath of collision
6.1 Aftermath

//Ball hits cube
6.1.1    IF hit is true THEN
6.1.2    Cube rotation speed = cube rotation speed – 10
6.1.3    END IF

//Ball fails to hit cube
6.2.1    IF hit is false THEN
6.2.2    Cube rotation speed = cube rotation speed + 5
6.2.3    END IF

//End of Program
7.1 End
7.1.1    IF Cube position > 20 THEN
7.1.2    Alert user – GAME OVER
7.1.3    END IF

```

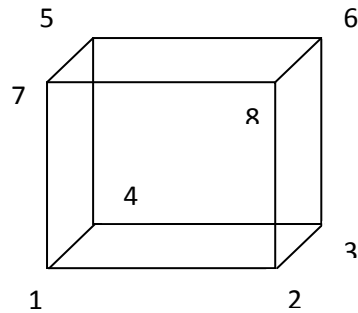
10. Appendix II

Derivation of the Model

In order to create the cube we must create three matrices:

<u>Matrix Type</u>	<u>Purpose</u>
Vertex	The position of each vertex
Adjacency	How each vertex connects to other vertices
Face	What vertices are used to form a face

In the creation of the cube, we are using the following vertex numbering system:



a) Vertex Matrix

$$\begin{pmatrix} -10 & 19 & 0 & 1 \\ -9 & 19 & 0 & 1 \\ -9 & 20 & 0 & 1 \\ -10 & 20 & 0 & 1 \\ -10 & 20 & 1 & 1 \\ -9 & 20 & 1 & 1 \\ -10 & 19 & 1 & 1 \\ -9 & 19 & 1 & 1 \end{pmatrix}$$

b) Adjacency Matrix

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

c) Face Matrix

$$\begin{pmatrix} 1 & 2 & 7 & 8 \\ 2 & 3 & 8 & 6 \\ 4 & 3 & 6 & 5 \\ 1 & 4 & 5 & 7 \\ 7 & 8 & 6 & 5 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$