CSP2307 Introduction to 3D Game Programming

Workshop 6: I can't believe he did that with trigonometry.

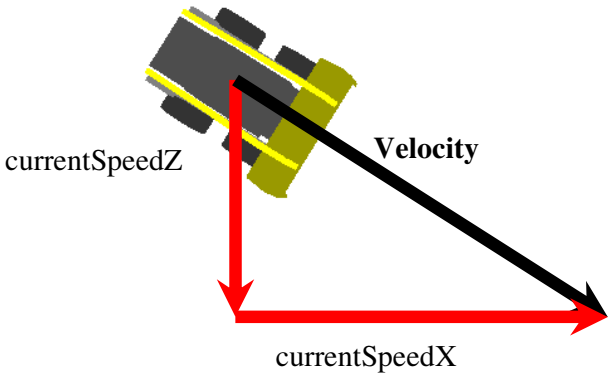
Value: 3 Marks
Due: Next workshop

Background:

Animation of the camera during game play can be used to enhance the player's experience. For example, in a top-down car driving game the camera 'zooms out' when the player is going at speed to give them a wider field of view. We will implement this in the workshop, along with basic boundary collision detection and response.

Procedure:

In the Excavator class, the velocity of the object is represented using **currentSpeedX** (the speed along the x direction) and **currentSpeedZ** (speed along the z direction). Together, you can think of these as forming a 2-dimensional velocity vector.



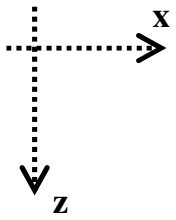
$$Velocity = (currentSpeedX, currentSpeedZ)$$

The speed of the excavator is the magnitude of the velocity vector. This can we worked out using Pythagoras as:

$$Speed = \sqrt{currentSpeedX^2 + currentSpeedZ^2}$$

Task 1 – Collision with Axis-Aligned Boundaries (1 Mark)

When collisions occur with obstacles aligned to the coordinate system axes the velocity component normal to the axis is reversed.



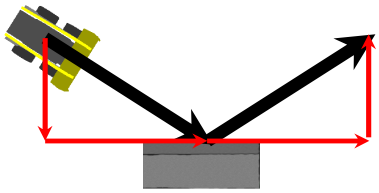


Figure 1

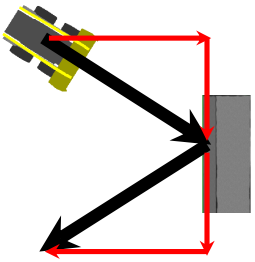


Figure 2

For example, if initial velocity before the collision was (5, 3), after the collision in Figure 1, the new velocity would be (5, -3). The same initial velocity (5, 3) would become (-5, 3) after the collision in Figure 2.

Step 1:
Modify the 3rd person view transform so that cameraDistance is 80 units from the excavator and cameraHeight is 50 units, and that the cameras x-position follows the excavator.

Step 2:
Implement some boundaries for the play area, so that if the excavators location on the x or z axis exceeds the boundaries it bounces off them. Eg. **z** between -60 and 60, **x** between 200 and -200.

Task 2 – Camera Animation (1 Mark)

Step 1:
Add a method to the excavator class, getSpeed(), which returns the speed of the excavator, determined from the currentSpeedX and currentSpeedZ as given on the first page. To do this you will have to look up the c++ functions for square root and power.

Back in the view transform in the render method; use the excavator speed to change cameraHeight. Tune your use of this to make the camera movement smooth and the game playable. A generic format is:

```
cameraHeight = (A*pow(speed,B)) + C;
```

Where A, B, and C are constants you can tune.

Task 3 – Make it look nicer (1 Mark)

Step 1:
The current render method has the following code to draw the 'ground' as a series of triangle strips

```
glColor3f(1,0,0);
int planeSize = 100;
glNormal3f(0,1,0);
```

```
for (int zCoord = -planeSize; zCoord<planeSize; zCoord++)
{
      glBegin(GL_TRIANGLE_STRIP);
        for (int xCoord = -planeSize; xCoord<planeSize; xCoord++)
        {
              glVertex3i(xCoord,0,zCoord);
              glVertex3i(xCoord,0,zCoord+1);
        }
    glEnd();
}
```

Replace this with a Geometry object, stored in the same C++ vector object along with the houses, excavator, and police cars (ie. Constructed in the `generateMap()` method). Also, make it lower resolution (it currently gets drawn with 1 vertex per unit, 1 vertex every 20 units should be enough). Ignore the normal, but assign texture coordinates.

Since Geometry objects are drawn as individual triangles, rather than a triangle strip you will have to think this through.

Step 2:
Make the ground the same size as the play area.

Step 3:
Add objects along the boundary – the following boundary object is included in this week's workshop files on blackboard.