CO1301: Games Concepts
Lab 7 - Vectors Continued

1. Introduction

1. You are going to carry on working with vectors in this lab. Before you get started on the lab sheet, take this quiz on vectors to refamiliarise yourself with the concepts covered last week's lecture.

2. If you found the quiz to be difficult, you may find the videos of solved examples in the Week 6 folder on Blackboard helpful.

3. Now that you have refreshed your knowledge on vectors, let's get on to the lab. The project you will be creating this week will make use of direction vectors and the length of vectors.

2. Setting Up

1. Create a TL-Engine project called Lab7_Drones_Project.

2. Download the associated media files for the project, save them to a folder called media inside your project folder. Now use the AddMediaFolder() method to add the folder's path to your folder.

3. Use the "ground.x" mesh to create a floor model at the origin. Then scale the floor model so that it is twice its initial size.

```
floor->Scale(2)
```

4. Use the "drone.x" mesh to create two drone models named "playerDrone" and "enemyDrone".

    - Position playerDrone at (0.0f, 60.0f, 0.0f),
    - Position enemyDrone at (0.0f, 60.0f, 400.0f)
    - Set the skin of enemyDrone to "Red_Drone.jpg".

5. Create a kManual camera and position it at (0.0f, 600.0f, -250.0f) and rotate it 45 units along its local X-axis so that it is tilted down.

6. Declare an IMesh* variable named "bulletMesh" and load the "bullet.x" mesh. Then declare and IModel* variable named "bullet", but don't initialise it yet (it will be used later).

7. Create a constant float called kGameSpeed and initialise it to 0.5f.

8. In your game loop add code that responds to holding the W key to move playerDrone forward along local Z-axis, and responds to holding A and D keys by rotating playerDrone left and right along local Y-axis. Use kGameSpeed to determine the speed of these movements e.g. you can move playerDrone forward at a speed equivalent to kGameSpeed and rotate it at a speed equivalent to kGameSpeed/2.

9. LookAt() is a method for models (or cameras) that points the local Z-axis towards another model. This effectively makes the first model/camera look at the second:

```
shooter->LookAt( target );
```

   Use the new 'LookAt' method to make enemyDrone always point at playerDrone. Think carefully whether to put this in the initialisation section or game loop. Move playerDrone around to test that enemyDrone continues to point at it.

3. Getting the Vector between Two Models

1. Create a new integer variable (in the initialisation section) called 'shoot' and initialise it to –
   1. You will use this variable to track a bullet fired by enemyDrone:
      - When shoot is –1 there is no bullet and one needs to be created
      - When a bullet is created we set shoot to a large value, say 500, that will count down as the bullet travels – this is the life-time of the bullet
      - When the value of shoot reaches 0 the bullet will be destroyed - then we will reset the value of shoot to -1 to indicate there is no bullet again
      - This will allow enemyDrone to fire one bullet at a time, and the bullet will have a limited range depending on its life-time
2. In order for enemyDrone to fire a bullet at playerDrone, the direction of movement between playerDrone and enemyDrone must be known. The direction of movement between two points is described by a vector. The vector between the two points that the two models are positioned at can be calculated by subtracting the position of one point from the position of the second point. An example of how this can be done is shown below:

```
float vectorX;
float vectorY;
float vectorZ;
vectorX = Model2->GetX() – Model1->GetX();
vectorY = Model2->GetY() – Model1->GetY();
vectorZ = Model2->GetZ() – Model1->GetZ();
```

   Note: the order in which you perform the subtraction above will affect the direction of your vector.

3. In the game loop, write code to calculate the vector from enemyDrone to playerDrone. The x, y and z coordinates of enemyDrone and playerDrone can be found using GetX(), GetY() and GetZ() methods. The result of the calculation each component of the vector should be assigned to vectorX, vectorY and vectorZ as shown in the code snippet above.

4. Calculating the Length of a Vector or Distance Between Two Points

1. The distance between two points is the length of the vector between the two points. The length of the vector is given by:

$$\|V\| = \sqrt{x^2 + y^2 + z^2}$$

2. Declare three new floats named distanceX, distanceY and distanceZ for storing the squares of your vector componentes you calculated in the last section i.e. vectorX, vectorY and vectorZ. Use simple multiplication to derive the square of any two values e.g.

```
distanceX = vectorX * vectorX; // distanceX squared
```

3. Sum the squares of the components and then calculate the square root of the result to get the length of the vector. There is a function within C++ to calculate square roots called sqrt(). To use it, you need to add the following two lines of code to the top of your program.

```
#include &ltiostream>
using namespace std;
```

   This will give you access to the C++ maths library and to the sqrt() function. The function takes just one parameter e.g.

```
float root;
root = sqrt( 9*9 ); // the square root of 9 squared
```

Now use the sqrt() method to calculate the distance between enemyDrone and playerDrone (the length of the vector). Use a new float variable called "distance" to store the distance calculated.

5. The Direction Vector

1. You have derived the vector between the enemyDrone and playerDrone. You have also calculated the length of the vector. In order to use these values to move a bullet fired by enemyDrone towards playerDrone you are going to have to calculate the direction vector between the two drones, therefore you have to normalise the vector you have derived.

2. The direction vector is independent of the distance between two points. For example, it allows you to move a model in a particular direction disregarding the speed of the movement. If the vector between the two positions is V(X, Y, Z), and the length of this vector is L, then the direction vector is VDir = (X/L, Y/L, Z/L)

3. To calculate the direction vector from enemyDrone to playerDrone, first create three new float variables: directionX, directionY and directionZ. Then use the variables to store the components of your direction vector as shown below:

```
float distance = distanceX + distanceY + distanceZ; //sum of the squares of the vector's compo
distance = sqrt( distance ); // the square root of distance

float directionX = vectorX / distance;
float directionY = vectorY / distance;
float directionZ = vectorZ / distance;
```

4. Check that you are dividing each vector component by the distance and not, for example, distanceX by distance.

5. After the calculations from the previous step, use an ‘if’ statement to test:

    - If the distance between enemyDrone and playerDrone is less than 250 (use your float variable "distance")
    - and the shoot variable is equal to –1

    If both these conditions are true then:

    - Using the IModel* variable named bullet you declared in the initialisation section, create a bullet model using bulletMesh. Create it in the same position as enemyDrone. Note that you are creating the model inside the game loop.
    - Initialise the bullet's direction vector variables using directionX, directionY and directionZ from above (you'll have to first declare bulletDirX, bulletDirY, and bulletDirZ in the initialisation section, then in the if-block assign to them the values of directionX, directionY and directionZ respectively).
    - Set the shoot variable to 500 – this will be the life-time of the bullet.

6. Now add another ‘if’ statement to test if the ‘shoot’ variable is greater than 0. If so, then move the bullet model by the bullet direction vector.

```
bullet->MoveX( directionX );
bullet->MoveY( directionY );
bullet->MoveZ( directionZ );
```

    Also, decrement the shoot variable by 1 here so that the bullet life decreases as it travels.

7. Add a final ‘if’ statement that tests if the ‘shoot’ variable is equal to 0. If so destroy the bullet model. You need a statement like:

```
bulletMesh->RemoveModel( bullet );
```

Then, set the shoot variable to –1 as there is no bullet now

8. You should now have an enemy drone that tracks your drone and shoots at it when it comes near. Read the code you have written paying special attention to the logic that is used to create and destroy the bullets.

6. Practice Exercise - Continued Vectors

1. If you complete the tasks, extend your program by either:
    - Test if the bullets have hit playerDrone (using vector distances).
    - Having enemyDrone shoot more than one bullet at a time.
    - Adding more enemy drones.