CO1301: Games Concepts
Lab 10 - Trigonometry: Waves and Points

1. Introduction

1. In this lab you will be recreating the trigonometric function waves shown in the lecture.
2. This lab sheet will also provide practice in the use of arrays and methods.

2. Setting Up

1. Create a new TL-Engine project called "Lab10_Waves_Project".
2. We will be setting up a program that moves lots of models along wavy paths. To ensure that we are able to see our models clearly we won't be using any backdrops.
3. Create a kManual camera and place it at location (13, 0, -17).
4. Load the sphere mesh and create a sphere model at location (0, 0, 0), then set the skin of the sphere to "RedBall.jpg".
5. Scale the sphere model down by a factor of 100.

```
sphere->Scale( 0.01f );
```

6. In the game-loop, add code that moves the sphere across the screen from left to right at a speed of 0.01, i.e. along the x-axis (use MoveX not MoveLocalX()).
7. Define a constant called kPi for π, assign it a value of 3.1415926f. Next define a constant called kEdge for the edge of movement i.e. the end if the wave. It needs to have a value of 8 * π.

```
const float kPi = 3.1415926f;
const float kEdge = 8 * kPi;
```

8. Update your program so that when the x coordinate of the sphere is greater than or equal to kEdge, its x-position is reset to 0. You should now have a sphere moving continuously from left to right.

3. An Array of Spheres

1. In this section you will update your program to use an array of sphere models in place of a single sphere model.
2. Declare an integer constant called kSpheres and assign it a value of 126.
3. Remember arrays? In the setup area of your program, remove the code that creates the single sphere model, then declare an array of IModels named "spheres", use kSpheres to specify the size of the array.

```
IModel* spheres[kSpheres];
```

4. Use the for-loop below to create 126 models using the sphere mesh and assign them to the IModels in the spheres array.

```
for( int i = 0; i < kSpheres; i++ )
{
    spheres[i] = sphereMesh->CreateModel( i * 0.2f, 0, 0 );
    spheres[i]->Scale( 0.01f );
```

```
        spheres[i]->SetSkin( "RedBall.jpg" );
    }
```

Note the way that the loop index is being used to access each individual member of the array, place sphere models in a line along the x axis, scale the models and set their textures.

5. Update the code in your game loop to move each sphere in the spheres array along the x-axis by a small amount (e.g. 0.01f), and reposition it to 0 along it x-axis if its x position is greater than or equal to kEdge. You will need to use a for-loop.

4. Sine and Cosine Waves

1. The next step is to change the path of movement of the spheres to a sine wave.
2. At the moment as the spheres move across the screen only their x coordinates are being changed, which is why they move in a straight line. The y coordinate of the spheres need to be changed in order to produce a sine wave. Therefore, you need to set the y coordinate of each sphere to sin(x), where x is simply the current x coordinate of the sphere. You will need to:
    1. Get the current x coordinate of the sphere
    2. Apply sin() in order to obtain the new y value
    3. Set the y coordinate of the sphere to this new value

    Your spheres should now be moving along the path of a sine wave.
3. To shows both sine and cosine waves using the spheres in the array, the two for-loops that create and move the sphere models need to be updated (the modulus operator will be useful for the tasks below).
    - Update the for-loop that creates the sphere models so that every other model created uses the "BlackBall.jpg" skin. Therefore, half the spheres in the array will use the RedBall texture and the other half will use the BlackBall texture.
    - Update the for-loop that moves the sphere models so that every other model's y position is set to its cos(x). Therefore you will now have 63 red spheres moving in sine wave and 63 black spehres moving in a cosine wave.
4. What does this say about the relationship between sine and cosine?
5. The height of the waves can be adjusted by scaling the y value, try scaling it up and down after you have calculated the value of y from sin(x) and cos(x). Changing the value of y in this fashion is adjusting the amplitude of the wave.
6. You can also change the frequency of the waves by changing the speed of the spheres, adjusting the value of x before calculating sin(x), and increaing the number of spheres.

5. Points of a Circle

1. This section will guide you through a creating a project that moves a sphere so that it traces the path of circle. This will be done by moving the sphere along points on the circumference of the circle.
2. Create a new TL-Engine project called "Lab10_Circle_Project".
3. Create a kManual camera and place it at location (0, 0, -250).
4. Define a floating point constant called kRadius with a value of 100.0f. This will be the radius of the circle which is going to be created.

5. Load the sphere mesh and create a sphere model at location (kRadius, 0, 0), then set the skin of the sphere to "RedBall.jpg".

6. Define a constant called kPi for π and assign it a value of 3.1415926f.

7. In the setup area, declare a floating point variable called "degrees" to hold the current angle for which the x and y points of the circle are being derived and initialise this variable to 0. Then in your game loop:

    - Increment degrees by a small amount (e.g 0.1f) each pass of the game loop
    - Reset the value degrees back to 0 each time it reaches 360.

8. However, the trigonometric functions sin(), cos() and tan() in C++ use radians and not degrees. Therefore, you will need to convert the angle stored in the variable "degrees" to radians.

```
float radians = kPi/180 * degrees
```

9. Now that you have the angle in radians,the x and y xordinates of the point associated with this angle can be derived by:

```
float x = kRadius * cos ( radians )
float y = kRadius * sin ( radians )
```

10. Use the derived x and y cordinates to position your sphere (the z cordinate should be set to 0)

11. The sphere should trace out the path of a circle with radius kRadius.

6. Multiple Points Using Arrays

1. Declare a constant integer kSpheres and assign it a value of 8.

2. Declare an array of IModels and name it spheres. The array should be of size kSize, i.e. 8, and will be used to store sphere models.

3. Declare another array of floating point numbers also of size kSpheres and name it "degrees", then initialise it as below:

```
float degrees[kSpheres] = { 0, 45, 90, 135, 180, 225, 270, 315 };
```

   This array will be used to store the initial angle to be used for each of the spheres in the previous array.

4. Use a for-loop to create the sphere models for the "spheres" array. Each sphere needs to be re-textured to "RedBall.jpg" and placed at its correct location based on the angle in its equivalent index in the "degrees" array (i.e the potition of spheres[i] should be derived using degrees[i]). Use the sine and cosine formula given 5.9 above to derive the x and y coordinates for each sphere.

5. Just as was done in section 5, increment the angle of each sphere by a small amount and reset it to 0 when it reaches 360. This will make the spheres move round the circles path. You can do this in the for-loop from the previous step or in a separate for-loop.

7. Utilising Functions

1. A function is a block of code that performs a specific task. To define a method you have to specify a name for it, its return type (void if does not return a value), its paramters (type and order of its required input(s)), and the functions body i.e. its code.

```
return_type function_name( parameter list ) {
    body of the function
}
```

You can also declare a method dirst by proving its return type, name, and paramters, and then define its code later. The function's name is used to call the function's code anytime in other parts of your code.

2. Let us create a function called rotateSpheres() to perform all of the operations required to rotate the array of spheres. The function is a void function, i.e. it does not return any values.

3. You will need to move all of the code you've been using to rotate the spheres into the function's body.

4. The rotateSpheres() function will need to take the array of spheres and the array of degrees as parameters. When the identifier (the name) of an array is used as a parameter, the whole array is passed over. In order to pass an array over to a function, the receiving function needs:

    - the data type of the array (int, char, float, etc.)
    - the size of the array

   Therefore your function definition should be similar to the code below:

```
void rotateSpheres( IModel* rotationArray[], float degreeArray[], int sizeOfArrays )
{
//function body goes here
}
```

5. In order to invoke the above array you would call it by name and place the name of the array of spheres inside the round brackets, e.g.

```
rotateSpheres( spheres, degrees, kSpheres );
```

6. Modify your code to include the rotateSpheres() declaration and definition, then invoke the function in your game-loop with the eapprorpiate parameters.

8. Advanced

1. Create a brand new array of spheres. Set the spheres to yellow rather than red. Place the spheres in a different location to the original circle. Make both arrays rotate using the rotateSphere() function.

2. Apply the sine function to the spheres as they move around the circle. You should be able to create a wriggle in the circle.

3. Change kRadius from a constant into a normal variable and rename it so that it no uses the k prefix. Set up two keys so that you are ablt to increase and decrease the size of the radius whilst the program is running.