Tadhg Riordan
Computer Graphics - CS4052
Lab 6 - Interactive Game Project

Game Title: **Rings**

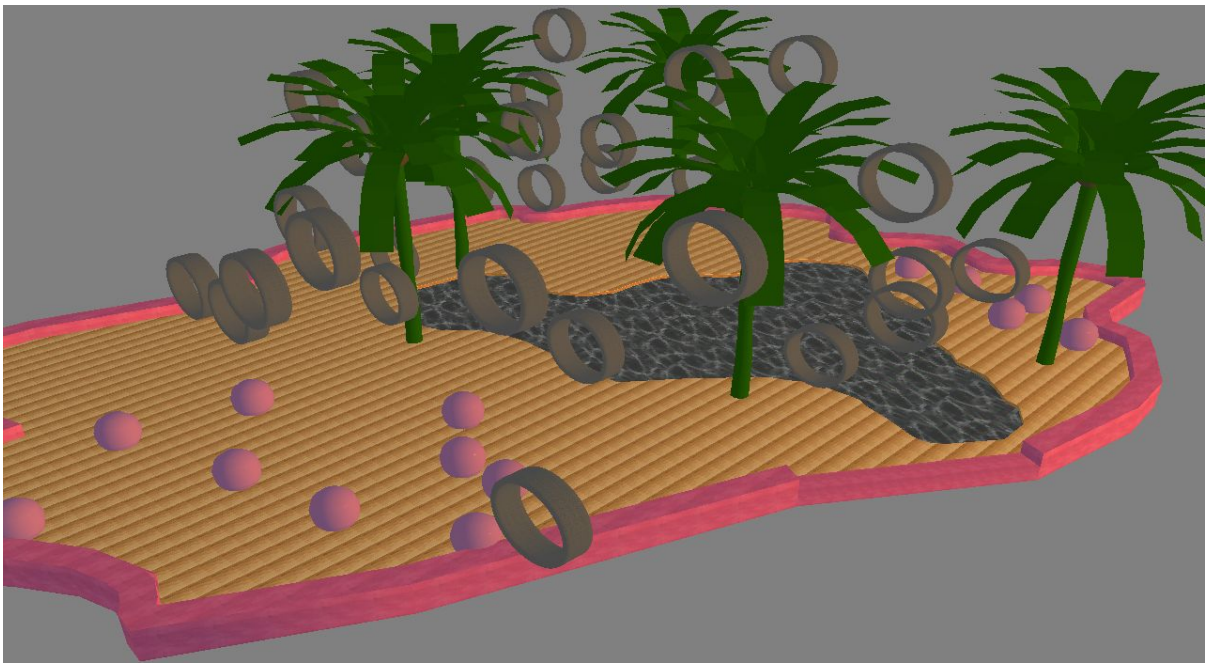

fig.1 - full scene view

Rings is a game which allows a player to roam around a beach scene, moving through hoops as they go to score points. Each single ring traversed earns the player a single point; the player is given 30 seconds to score as many points as they can.

Standard Features

user interaction and camera-control

The game is viewed from a first person position, ie. the player is the camera object as it traverses the scene. I used the Camera class from learnopengl.com for this purpose. I set up my callbacks to this class when I wanted to change the position of the camera. The mouse position on the screen was used to control where the player was facing. I set this sensitivity of this to be quite high as I wanted to avoid a situation where the player would move the mouse to the farthest edges of the screen and not be able to move in that direction anymore. The callback for the mouse was put in glutPassiveMotionFunc(). Forward, back, left and right were controlled with i,k,j and l respectively. These were also set high which gave the game a fast pace.

scoring and winning/losing

As mentioned, the player scores points as they move through the rings, where the player must try and score as many points as possible within a 30-second period. When a ring pass-through is detected, a global score integer is incremented. This score is then printed out in the console. I

wrote a timer function which keeps track of the elapsed time once the game has started. Once this reaches the 30 second mark, the user loses control of the game and the game ends, being printed in the console. I simply stopped execution of the user control callbacks when this event occurred.

3-dimensional objects and views

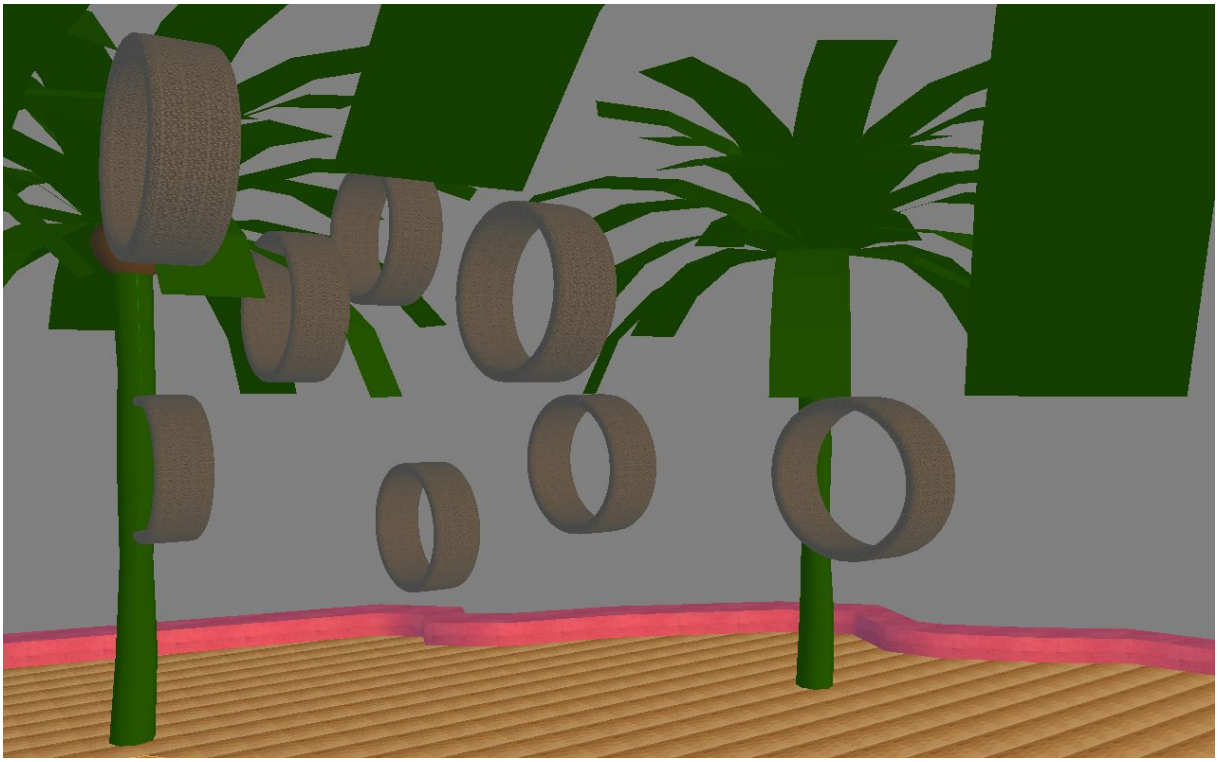

fig.2 - inner scene view

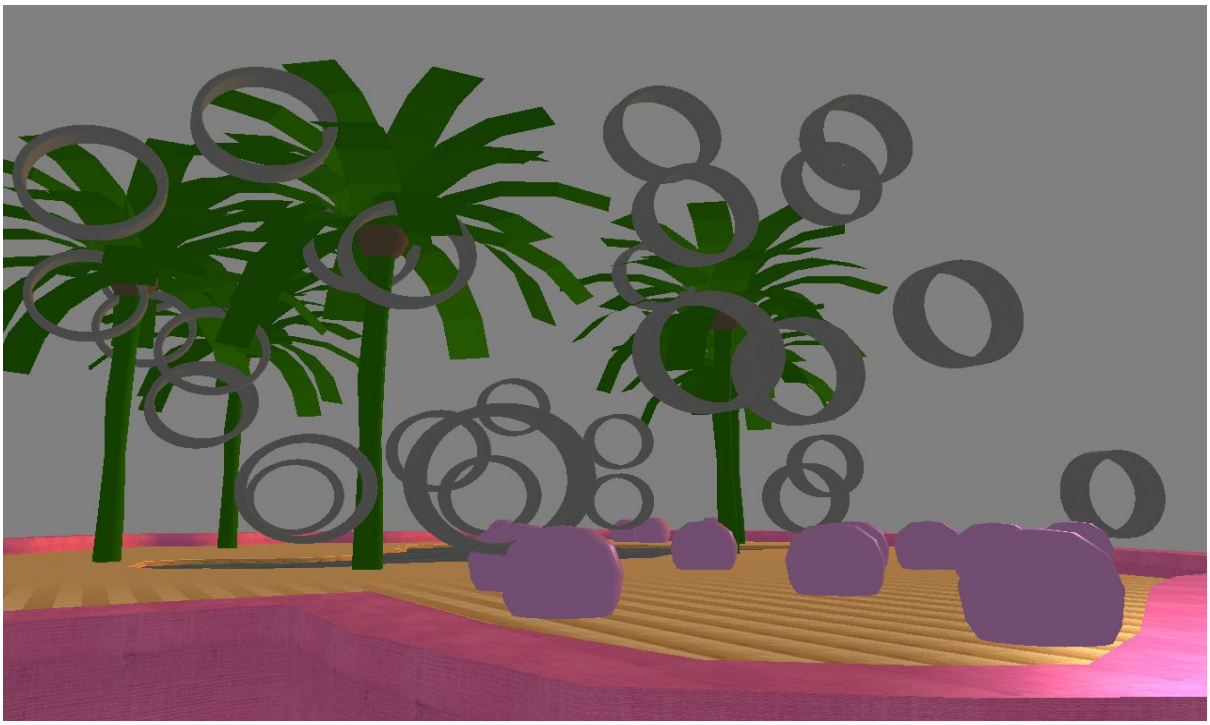

fig.3 - front view

As evidenced from my screenshots, my game contains a variety of 3D objects and views. The scene was imported from an online source and the ring object was designed by myself in SketchUp using an online tutorial. As mentioned, the view may be altered by the position of the camera.
From the source code provided, I was having trouble understanding how an object with multiple meshes (such as the desert in my game) could be imported and set up correctly. I realised that the default code was designed to work with an imported object that had just a single mesh, and this was going to be a problem in my case, as I ultimately wanted to work with each mesh in the scene individually, for texturing and collision detection. I altered the code to load an object at a time from a file, and then to load each mesh individually from that file. I stored the point counts of each mesh individually, as well as creating separate VAOs and VBOs for each. The vertex points of each ring imported were also saved as these would be needed later for collision detection.

Use OpenGL shaders for Phong lighting

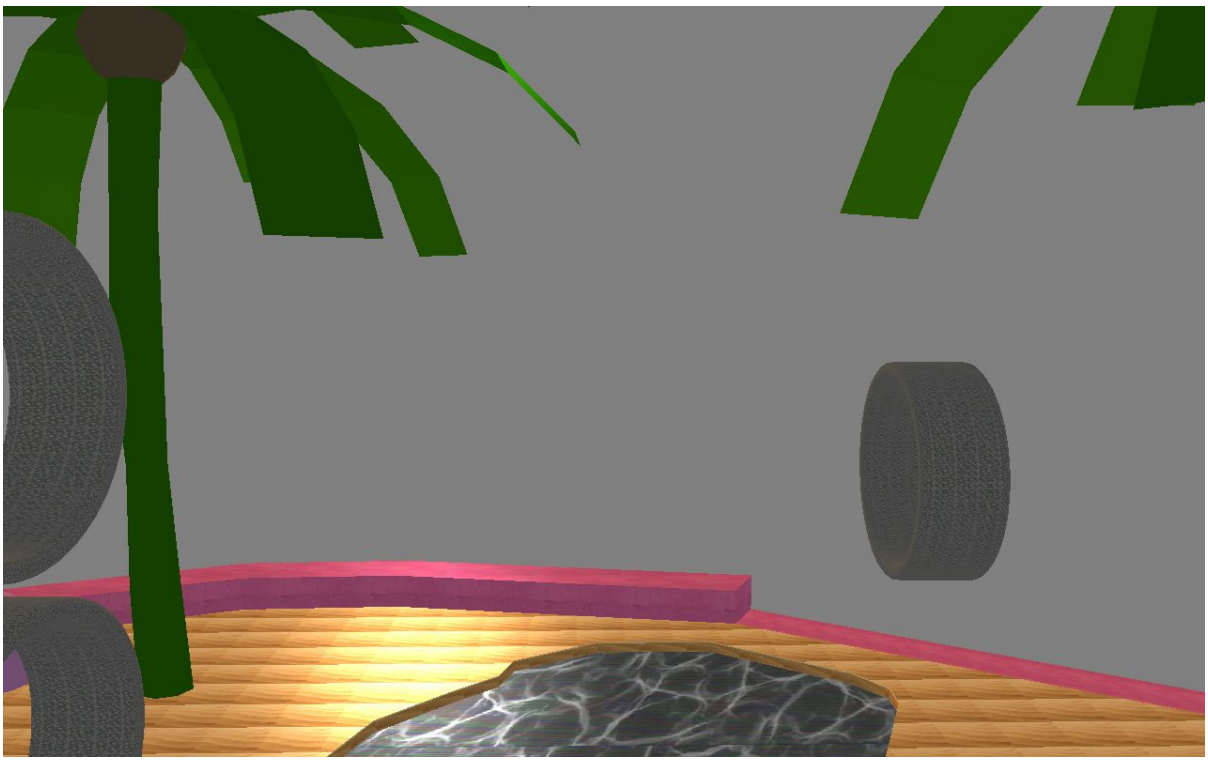

fig.4 - side/lighting view

The default Phong lighting code from the lecture notes was set up in the shaders, exactly how it was from my previous lab submission.

At least 1 texture-mapped object

Anton gave us some great tips on the discussion board on implementing a texture loading function, which I also made use of. This function makes use of the popular stb_image.h file. It loads the texture, binds it with some parameters, and then frees it. I also added a function to load the texture file names to a textures array. Each position of this array had a corresponding mesh with the same index. In the loop in my draw function each texture was bound before the shader and drawArrays calls. Finally, GLSL code had to be added to the shaders; code was added in both the fragment and vertex shaders, where it essentially added another multiplication operand for the output.

I made the effort to texture every object in the scene. The scene I downloaded already had texture coordinates in place, while the ring mesh I designed had to be imported into Blender to add these coordinates. When complete, i simply had to find suitable files online to use as textures for the meshes. I think the resulting textured objects turned out looking very well.

Extra Features

Sound

In C++ on Windows, there exists a function called PlaySound, which allows for simple, short sounds to be played. I completed this assignment on a Mac however and this was not possible. After trying some other methods, I found a cross-platform library called IrrKlang. This library was very easy to build (requiring one library file to be stored at usr/local/lib, avoiding the use of manually creating a makefile) and required just two more header files to be usable which came with the downloadable. Once this was implemented, I created a sound engine globally, and called a function to play whatever sound I wanted when I wanted it in code. As you can hear from the video, I have a background .wav file of a beach type scene, and when a ring is traversed and a point is scored, a short sound is also played.

Pseudo-Randomly Placed Scene Objects

The ring file was loaded a number of times equal to the NUM_RINGS global value and there vertex positions stored as mentioned. The ring positions were established in load_ring_translations(). This function uses a pseudo-random algorithm (using srand to get a seed based on the current clock time) to get points for the rings. Each ring mesh was then be translated to its respective position on each draw call.

Collision Detection

object boundary collision detection

I attempted to get object boundary collision detection working in the following way:

- I compared the current position of the camera to every position of every mesh in the scene.
- I stored the last position of the camera in a global variable.
- If the camera position was equal to any mesh vertex position, within a small region, then move the camera back to it's last vertex position, preventing the camera becoming unusable.

I originally had semi-decent results with this method, but it certainly did not work in all cases. On top of this, it worked in far less cases when I tripled the speed of the camera. What this was really doing was moving the camera at triple the vertex positions per draw call, and I think as a result my collision detection method has less of a chance of interpreting the current value, and it has a greater chance of moving through an object. Also when a collision is detected, the camera can "jitter" for a short period. This can be seen in the video.

Ring pass-through collision detection

I had a another type of collision detection which worked very well, and that was in the detection of the camera as it passed through a ring. My idea of this was as follows:

- On the first draw call the vertex points of each ring are translated, and the results stored back into the same array.

- following this I calculated the centroid (mean, or central point) of the vertex points of each ring.
- It then calculates all the points from the centroid to each point, at a step of .1 from the centroid.
- finally, a function is called which checks whether the camera position is within a distance of one of those points. any point within .5 on the y and z axes and .125 on the x axis is considered. The reason for this is to only get the points within the central “cut” of the ring, as when we pass through it we only want to score when we are at least half way through.
- When a collision is registered, a score is registered and the object ceases to be drawn.

This collision detection works in all cases.

Text rendering

Despite my best efforts I could not seem to get text rendering working. I tried multiple libraries and tutorials online, as well as attempting to implement Anton’s text rendering library, all to no avail. I asked some of my fellow students about how they implemented it but for some reason mine just wouldn’t show. I can only assume that I have some extra library conflict which is not allowing the text to be rendered, or alternatively, as I am doing it on a Mac, it may have something to do with the native version of OpenGL or GLEW.