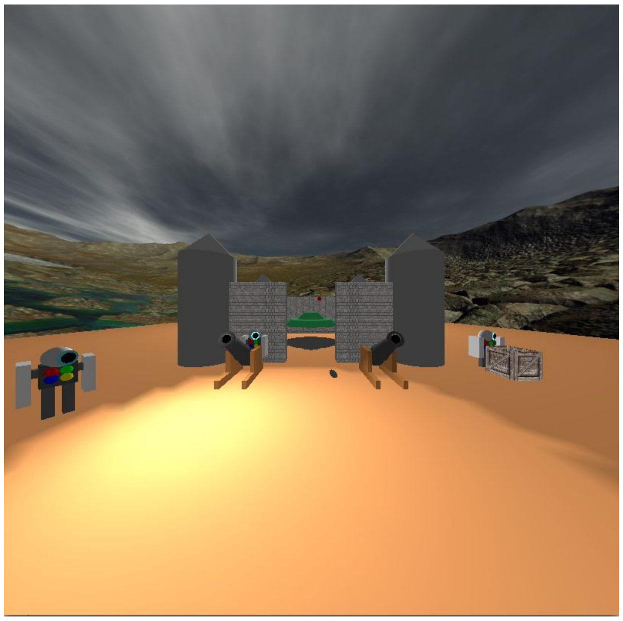
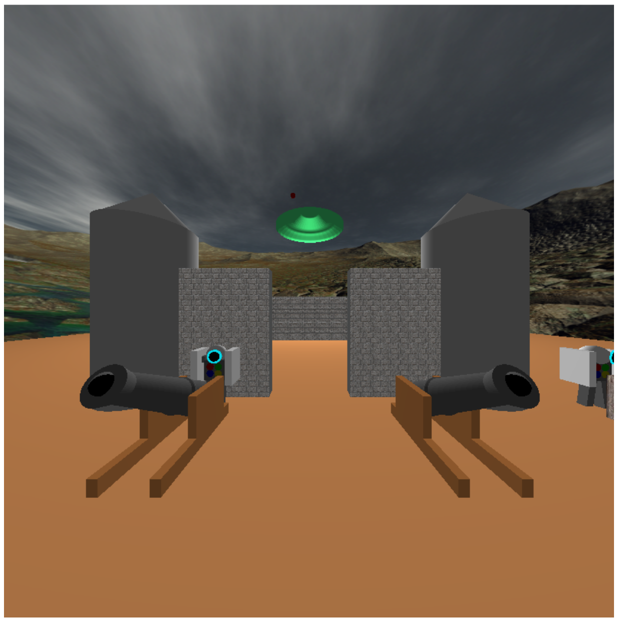
Assignment 1: Alien Invasion!

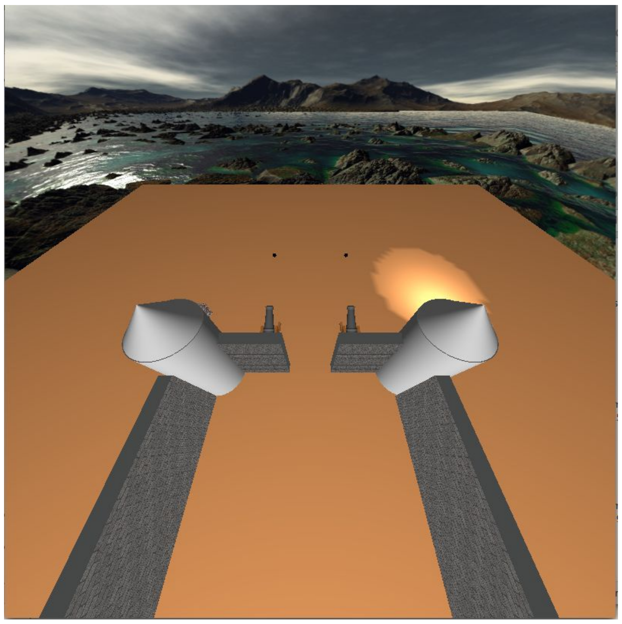
Chuan Law (81677469)

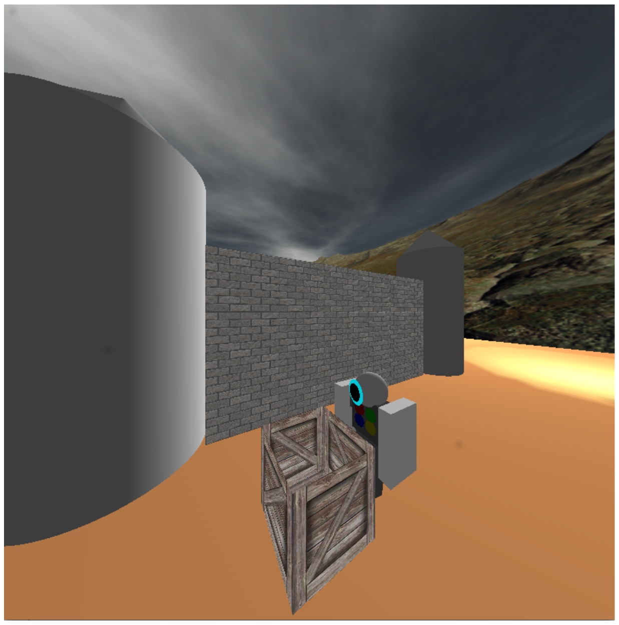
The Scene

The initial view shows a large gray brick fortress with each corner fortified by a pillar. The fort was built facing a lake and a rocky shore. Stone hills surround the fortress and a small light source is present just behind and above the back walls. In the center of the fortress hovers a a large green spinning UFO ready to take off at any time. A blinking red light orbits the spacecraft. There are two functioning cannons by the entrance and a robot stands ready to fire them. There is a robot to the right of the castle moving a crate and supplies around. Another robot has been deployed to patrol around the fortress parameter and to identify any intruders using its spotlight.









Additional & Extra Features

Planar shadows - Spaceship

A planar shadow was cast on the spaceship inside the castle. The primary light source for the scene is placed at {0., 300., 950., 1.0} which is near the edge of the scene (on the z axis) and above the height of the castle walls (on the y axis). This casts the shadow of the spaceship towards the entrance of the fortress and our initial view. The shadow matrix is defined as: { 300,0,0,0, -0,0,-950,-1, 0,0,300,0, 0,0,0,300 }.

Spot Light - Patrolling robot

The spotlight is defined with grey and white values respectively: {0.2, 0.2, 0.2, 1.0}, {1.0, 1.0, 1.0, 1.0}. It is placed at a point on a patrolling robots' head as the origin. It's direction is specified as {-0.5, -1, 0} due to wanting the spotlight to appear like an oval when tilted into the ground. The light is also rotated along with the robot head and moved along a path around the castle. This makes it look like the robot is actively scanning the area.

Two camera modes - default & spaceship

The default camera sets the scene at (0, 100, -250) which is just outside the fortress walls. The y is set to 100 to imitate eye level and the z is negative so the view captures the majority of the scene as objects are mostly placed on the positive half of floor. This mode is capable of moving around forward and backwards through the scene and rotating the current view by 5 degrees in either direction.

The second camera mode is activated by pressing the HOME button. This places the current view at a point of view on the spaceship. This switch is activated by a simple boolean value to check what the current view is and what gluLookAt function to load. When 's' is pressed, the spaceship takes off and the y value in the view is increased with a timer call. This enables you to see the castle beneath you as the ship rises. You can toggle between these two modes at any time.

Physics models - cannonball trajectory

A very simplified model for projectile trajectory was used to calculate the launch arc and distance of the cannonball when fired from the canon.

Assume 1 tick of the timer traverses a distance of 10 units. The x position is incremented 4 times per tick, so the cannonball is fired at an initial velocity of 40 units per tick.

Also assume gravity is regarded as 1 tick as 9.81 is rounded up to 10 units.

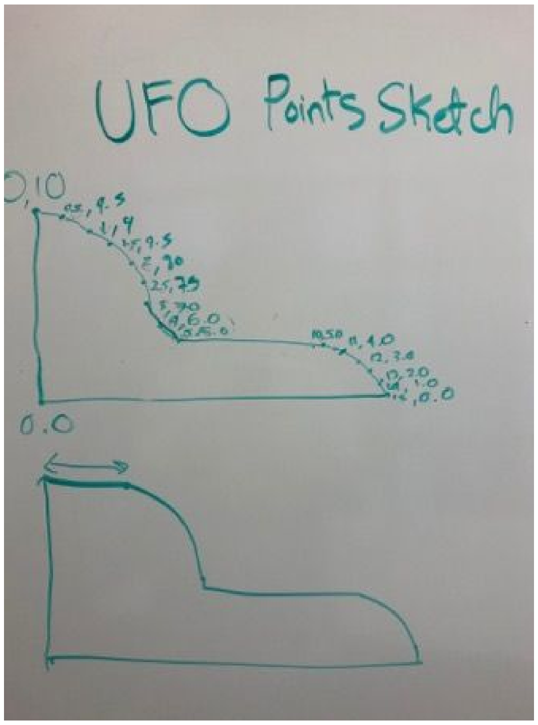
The only equation used was the maximum height equation. $(\frac{v^2 * sin(\theta)^2}{2 * 10})$ which defines initial velocity squared multiplied by sin squared multiplied by the angle (in degrees).

Putting in our variables gets $(\frac{40^2 * sin(30)^2}{2 * 10})$ = 20. This is then multiplied by 4 to match our assumption of 4 increments per tick. This gave the max height of 80 units in the program. This is not including the starting y position of 64 units of the cannonball as it rests in the cannon.

It is calculated that for each tick, the y position is incremented once. This is done to imitate the real world gravitational force of 9.81 m/s (10 units in the program) acting on an airborne object. This is applied both on the way up to the maximum height and decremented thereafter.

The assumptions used mean that the calculation of the entire trajectory is not completely accurate, however it does successfully produce a parabolic path as shown in the program.

Sweep surface - spaceship computation
A sweep surface was implemented to create the spaceship UFO model. A base polygon was constructed using 17 vertices with an array of x and y values. The figure shows early plot points being defined.
Using quad strips, each set of 4 points is used to specify a quad shape of the defined model.
After the first surface has been created, the points are then updated to the next in the arraylist of x and y values. Repeating this process 72 times in 5 degree steps (sweep) around the y axis completes the 3D UFO model using a surface of revolution.



Skybox
A set of 6 tga image files are loaded into the program and textured onto the 6 surfaces of the scene. Top, left, right, front and back are visible, but down is located below where the floor is drawn. This was done using glTranslatef(0, -200, 0) on the entire skybox to allow lights to be visible on the floor and to avoid the texture clipping. Each image has been textured once per surface (no repeats).

Special Challenges

Skybox
Initially when I first mapped the textures to the walls there would be large black gap that covered the bottom of each skybox corner. It only occurs when the camera angle was low enough and the view was being rotated either left or right. This issue was fixed by increasing the frustum value to 3000 in the gluPerspective function. This increased the camera view volume and allowed the full skybox image to be rendered regardless of the angle changes.

Creating and mapping textures
When creating my own surfaces for texturing, I would make 1 base surface that lay flat on the ground and just rotate each of them to form a box / wall. This meant that the x, y and z values were very inaccurate and I had to do a lot of trial and error to make them connect properly. This also caused many issues with the texture mapping as the texture would be mapped correctly but it would look rotated and incorrect due to the nature of the surface. To solve this I had to redo the surfaces again and created them on their right axis' respectively.

Special key usage and cannon fire
Only when and the 'c' key is pressed, a cannonball should be launched from the canon. However after 'c' was pressed, pressing any key afterwards would control the cannon fire. This was due to my misunderstanding that the keyboard function was only for the cannon and I left some commands without a valid button press check. Now each action is correctly checked with the proper if statement and ASCII key before it is processed.

Correct 5 degree angles on camera turn
This was difficult to understand as the conversion to the turning step was -100 * sin(angle) with angle being incremented or decremented in 0.1 amounts. This was solved by realising that under the conditions specified earlier, it would take 62 right arrow button presses to complete a full 360 degree revolution. For 5 degrees each turn, it would take 72 button presses. 72 / 62 means that 72 is 1.16129032 times larger than 62. Using simple ratios, 0.1 /1.16129032 gives the angle value of 0.08611111 to enable a turn of 5 degrees for each right / left button press.

List of Controls

Action	Key
Turn Left	← Left Arrow
Turn Right	→ Right Arrow
Move Forward	↑ Up Arrow
Move Back	↓ Down Arrow
Shoot Cannon	C
Initiate spaceship lift-off	S
Toggle camera modes (default/spaceship view)	Home

Resources and References

Skybox Texture (Boulder Bay) - http://www.custommapmakers.org/skyboxes.php by Mighty Pete
Crate - https://www.turbosquid.com/Search/Texture-Maps/free/wood
Caste Brick - https://filterforge.com/filters/2897.jpg