# 1 Scene Description

The scene consists of a four-walled, symmetrical, equilateral castle, where each corner is joined by a tower, and each wall has an opening. Each wall is translated 40 units away from the origin of the scene. Outside the castle, at the position $(0, 0, 70)$, there is a cannon facing away from the castle. The cannon can fire a single cannonball. Since there's no requirement that the cannon be reloaded, the cannon can only fire once. At $(-30, 20, 70)$, there is a large donut-shaped UFO, which has white light being projected down towards the ground from its centre. A rocket lies at the scene's origin, which can be launched from the ground. A single robot patrols around the castle, always exactly 10 units away from the walls. Inside the castle, another robot patrols in a circular fashion around the rocket, and at each gate, moves outside the castle walls for a brief period. A third robot is bouncing on a trampoline positioned at $(30, 0, 30)$.

The textures for the walls, tower, and tower roof are from textures.com [1] [2] [3]. The texture from the skybox is from COSC363 Lab 5 [4].

# 2 Screenshots

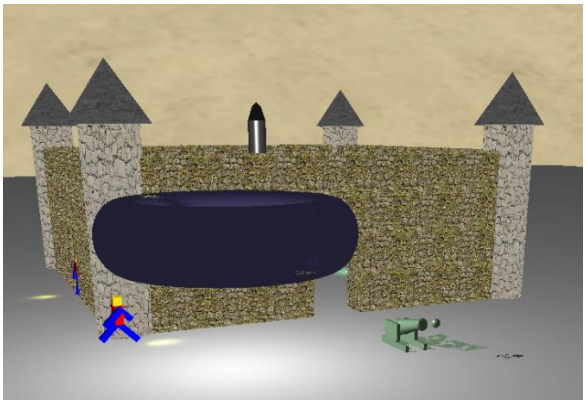


*Figure 1: Rocket launching, cannon firing, shadows from cannon and cannonball, robots patrolling with lights attached, donut UFO*

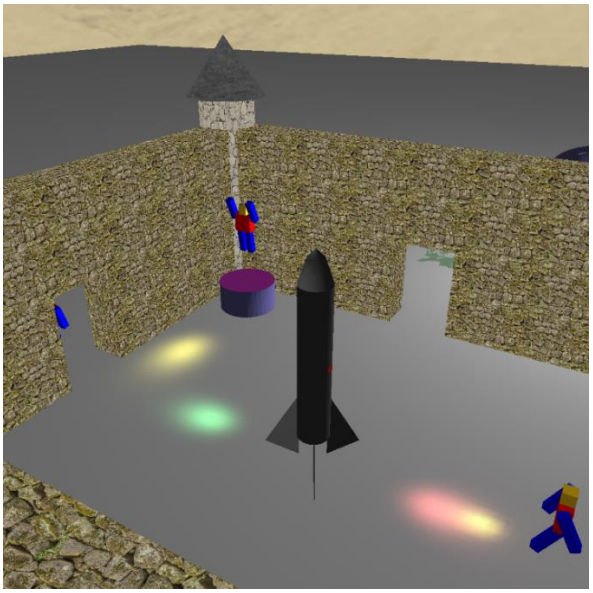


*Figure 2: Robots patrolling, robot bouncing on trampoline, lights attached to rocket*

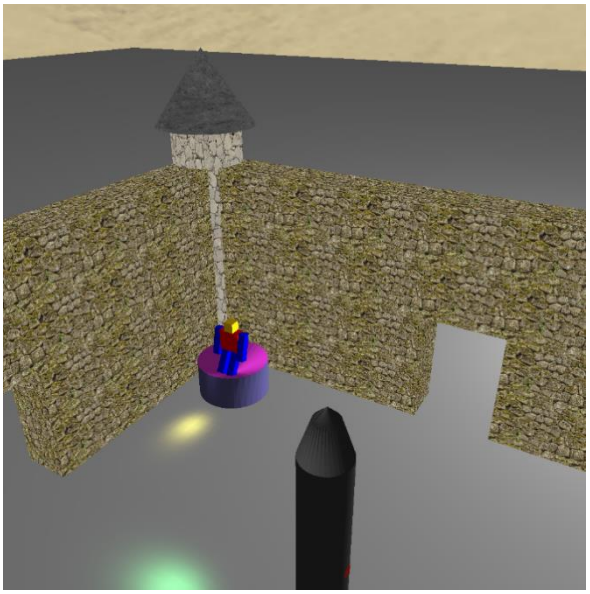


*Figure 3: Robot bouncing on trampoline, showing the downwards curve of the trampoline's fabric*

*Figure 4: Cannonball rising after bouncing*

# 3 Extra Features

## 3.1 Planar Shadows

Planar shadows are cast by the cannon and the cannonball. The shadow of the cannonball moves appropriately as the cannonball is flying through the air. In Figure 1, both shadows can be seen.

## 3.2 Spotlights

Spotlights are attached to each robot, which all move. In Figure 1, Figure 2, and Figure 3, the yellow spotlights can be seen in front of the robots. The spotlights are positioned at the robots' head, and face down towards $(0, -10, 10)$.

## 3.3 Additional Animated System

A robot can be seen in Figure 1 and in Figure 2, bouncing up and down on a trampoline. Both the purple base and the pink fabric of the trampoline are sweep surfaces.

The period of the trampoline is $15 \times 2 \times 50ms = 1500ms$, where each timer event is $50ms$ apart, and the trampoline takes $30$ timer events to completely fall and rise.

The curve of the trampoline is determined by the formula

$$f(x, time) = \frac{x^2 + ((time + 5)^2 - 5^2)}{(time + 5)^2}$$

where $time$ modulates between $0$ and $15$, inclusively. The composition $f(x, time)$ of ensures that all of the curves generated will go through $(x, y) = (\pm 5, 1)$, as seen below:

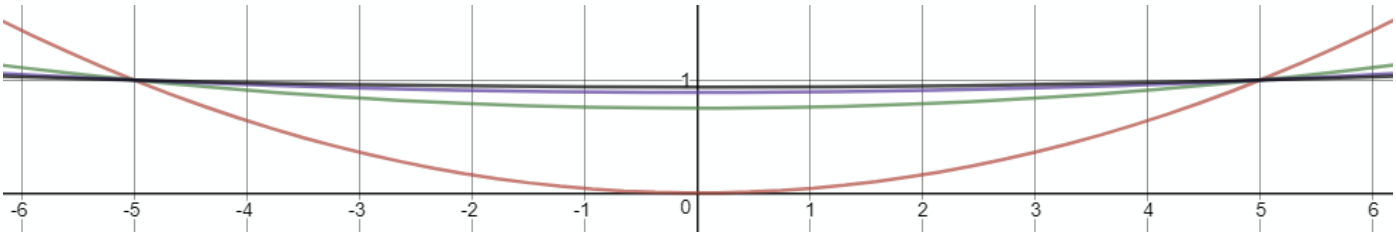



*Figure 5: The range for time is [0, 15]*

Key for Figure 5:

| Colour | Formula |
|---|---|
| | $f(x, time = 0) = \frac{x^2 + (0 + 5)^2 - 5^2}{(0 + 5)^2}$ |
| | $g(x, time = 5) = \frac{x^2 + (5 + 5)^2 - 5^2}{(5 + 5)^2}$ |
| | $h(x, time = 10) = \frac{x^2 + (10 + 5)^2 - 5^2}{(10 + 5)^2}$ |
| | $f(x, time = 15) = \frac{x^2 + (15 + 5)^2 - 5^2}{(15 + 5)^2}$ |

The trampoline is scaled $5$ times in the $y$-direction.

As expected, the robot bouncing on the trampoline accelerates away from the trampoline's fabric. Since half of the period is $15$, the change in $y$-axis for each unit is given by the formula

$$f(time - 7.5) = -\left(\frac{(time - 7.5) - 7.5}{d}\right)\left(\frac{(time - 7.5) + 7.5}{d}\right)$$

where $d$ is a constant value which determines the steepness of the parabola. $f(time)$ will always intercept the $x$-axis at $\pm 7.5$. If the trampoline is rising, then the robot's $y$-value will increase by the result of $f(time)$, and likewise the $y$-value will decrease by $f(time)$ if the trampoline is falling. However, if the $y$-value minus the result of $f(time)$ is less than the $y$-value of the trampoline fabric's lowest point, the fabric's lowest $y$-value is assigned to the robot.

## 3.4 Camera Modes

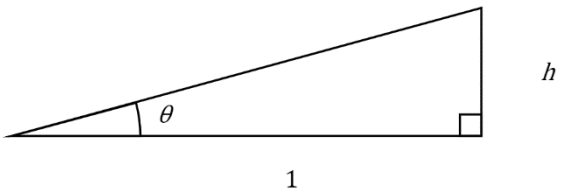
When the rocket is stationary, on the ground, the second camera mode is fixed to $(0, 18, fuselage\ radius)$, and looks out towards $(0, 0, 45)$. When the rocket starts launching, the camera remains stationary until the spaceship has an altitude greater than the $y$-axis value for the camera. Only once that threshold is reached does the camera start rising with the rocket. This was done in order to move the camera during the launch to nearer the bottom of the rocket. As the rocket increases in height, it continues to look at $(0, 0, 45)$, and thus shows a view of the castle beneath.

## 3.5 Physics Models

In addition to the physics model for the trampoline's animated system, a separate physics model was developed for the cannonball's flight.

Given $\theta$, the angle of the cannon in relation to the ground, the gradient $m$ of the cannon can be found:

$$m = \tan(\theta) \times 1$$




The linear line, given by the cannon in relation to the origin is:

$$g(z) = mz + z_1$$

where $z_1$ is a constant value.

To find a curve $f(z)$ which $g(z)$ forms a tangent to at $(z, y)$:

1. $y = -cx^2 + az + b$
2. $\frac{dy}{dz} = m = -2cz + a$

Substituting the two simultaneous equations into each other gives:

$$f(z) = y = -cz^2 + (m + 2cz_1)z + y_1 + cz^2 - (m + 2cz_1)z_1$$

where $c$ is a constant representing the power of the cannon. $f(z)$ represents the parabolic arc of a single bounce for the cannonball.

As the ball moves along the $z$-axis, its velocity changes. This is done by decrementing $z$ by a value $\Delta z$, which is constant. However, to avoid the possibility that the ball will stop in the middle of the air as $z = 0$, but $f(z) = y \neq 0$, once $\Delta z$ becomes less than a certain value, $z$ is no longer decremented until the ball has reached the ground. At that point, the cannonball rolls, and $z$ is decremented by $\Delta z$ until $z = 0$.

## 3.6 Sweep Surface

The donut/UFO is a sweep surface. A ring is generated from a sweep surface of triangle strips, with a radius of $5$, and height $1$. The ring is then translated by $(0, 10, 10)$, and rotated by an angle $i$, for $0 \leq i < 360$. This forms the donut/UFO.

## 3.7 Surface Shape

The nose cone of the rocket is a paraboloid. A triangle strip is generated by:

$$\{ f(x) = -x^2 + 4 \mid 0 \leq x \leq 2 \}$$

This is then rotated around the origin to form the paraboloid, and thus the nose cone.

### 3.8 Collision Detection

The camera is prevented from going outside or looking outside the skybox. This is done by ensuring that the camera always leaves a gap of 6 units with the skybox.

### 3.9 Skybox

A desert skybox consisting of the front, back, left, right, and top was placed into the scene $(\pm 500, \pm 500)$.

## 4 Control Functions

| Action | Control | Symbol |
|---|---|---|
| **Move forward** | Up arrow | ↑ |
| **Move backward** | Down arrow | ↓ |
| **Rotate left by 5°** | Left arrow | ← |
| **Rotate right by 5°** | Right arrow | → |
| **Move up** | Page Up | PgUp |
| **Move down** | Page Down | PgDn |
| **Launch the rocket** | s key | s |
| **Switch camera mode** | Home | Home |
| **Fire the cannon** | c key | c |

## 5 Discussion

### 5.1 Expanding Trampolines

One irritating problem was that the fabric of the trampoline would gradually increase in size over time, and the robot would correspondingly be translated higher and higher. Initially I thought it was a memory leak, involving passing pointers to arrays into the `sweepSurface` function, or the initialization of arrays whose size was determined by a parameter. So, I switched to passing in pre-initialized arrays to the function. However, this did not solve the problem. Using the Visual Studio Code debugging tools, I went through every piece of source code which affected the height of the trampoline and robot, including experimenting with various methods of casting and or assigning floats. None of that fixed anything. The debugging tools indicated that all the values were correct. Finally, in an act of desperation, I changed from doing:

```
static float fabric_x[fabric_n] = {0};
…
 for (int x = 0; x < fabric_n; x++) {
    fabric_x[x] = float(x);
    fabric_y[x] = (x * x + h) / base;
  }
```

to initializing `fabric_x` like:

```
static float fabric_x[fabric_n] = {0, 1, 2, 3, 4, 5};
```

and not assigning it inside the for loop. I'm still not sure why that fixed the problem, but it did.

### 5.2 Robots functionality

The functionality for creating robots was based upon the lab material. In order to streamline the definition of the robots, their limb movements, and their physical movements, I created a pseudo-object-oriented architecture. The source code for displaying the robots was largely similar to the lab. However, limb

movements, translation, and the rotation of the robot is entirely new. Each robot has a `void (*movement)(robot_props *robot);` method which defines its movement over time.

# 6 References

[1] textures.com, “BrickJapanese0117,” 7 April 2019. [Online]. Available: https://www.textures.com/download/brickjapanese0018/113852?q=brick&filter=seamless.

[2] textures.com, “BrickJapanese0018,” 7 April 2019. [Online]. Available: https://www.textures.com/download/brickjapanese0018/113852?q=brick&filter=seamless.

[3] textures.com, “Rock Cliff Volcanic 3 - PBR0190,” 7 April 2019. [Online]. Available: https://www.textures.com/download/pbr0232/133288?q=stone&filter=seamless.

[4] R. Mukundan, “COSC363 Lab 1 to Lab 5,” April 2019. [Online].

[5] The Khronos Group, Inc., “OpenGL 2.1 Reference Pages,” April 2019. [Online]. Available: https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/.

[6] Desmos, Inc, “Graphing Calculator,” April 2019. [Online]. Available: https://www.desmos.com/calculator.

[7] Wolfram, April 2019. [Online]. Available: https://www.wolframalpha.com/.

[8] R. Mukundan, “COSC363 Lecture Notes,” April 2019. [Online].