Edinburgh Napier UNIVERSITY

Coursework Report

Marco Moroni
402138730@napier.ac.uk
Edinburgh Napier University - Computer Graphics (SET08116)

Abstract

This computer graphics project aims to show a scene that replicates the art style used in the game *Monument Valley* by using OpenGL. The main characteristic of the game – and this project – is the clever use of the clever use of a camera with an orthographical projection that allows to create impossible geometries.

Keywords – OpenGL, Computer Graphics, GLSL, Monument Valley, Games Development

1 Introduction

Monument Valley (figure 1) is a game where the player guides a character through architectures made by impossible geometries. The goal of this scene (figure 2) is to recreate this same illusion with a very similar art style. The illusions are made possible through the use of an orthographical projection that eliminates the sense of perspective from the user (or the player).
The core techniques to archive such effects are explained in one development videos made by the creators of the game [1], and are here in part reproduced with OpenGL.

Last thing to note is that the aim of the project is not to build a foundation for a game. This means that the visual design of the project is not limited by the practical considerations of implementing gameplay.

2 Related work

This is a coursework made for a Computer Graphics module, which provides a workbook that covers most of the techniques used here and slides explaining the mathematical principles.

2.1 Maths

Embedding Maths is Latex's bread and butter

$$J = \left[\frac{\delta e}{\delta \theta_0}\frac{\delta e}{\delta \theta_1}\frac{\delta e}{\delta \theta_2}\right] = e_{current} - e_{target}$$

2.2 Code Listing

You can load segments of code from a file, or embed them directly.

Listing 1: Hello World! in c++

```
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
    std::cin.get();
    return 0;
}
```

Listing 2: Hello World! in python script

```
print "Hello World!"
```

2.3 PseudoCode

for $i = 0$ **to** 100 **do**
  print_number = true;
  if *i is divisible by 3* **then**
    print "Fizz";
    print_number = false;
  end
  if *i is divisible by 5* **then**
    print "Buzz";
    print_number = false;
  end
  if *print_number* **then**
    print i;
  end
  print a newline;
end

Algorithm 1: FizzBuzz

3 Implementation

For simplicity, this report will use the terms left block, central block and right block to refer to the 3 distinct part of the scene (figure 2).

3.1 Camera

Probably the most important aspect of this project is the orthographical projection, which makes the impossible geometries possible (see figure 2: it has a perspective projection).
The orthographical projection is a parallel planar projection. Because of the position of the camera (at (20.0, 20.0, -20.0) pointing at the origin (0.0, 0.0, 0.0)) the projection is also isometric, which means that the 3 axis *x*, *y* and *z* have the same length.
The projection of a point $P = (p_x, p_y, p_z)^T$ onto a projection plane is

$$\tilde{P} = (p_x, p_y, 0)^T$$

Figure 1: **Monument Valley** - A screenshot of the game

which means that the equivalent transformation matrix A is the following:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

therefore

$$\tilde{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

From the matrix is evident that the perspective has no part in the calculation.
The code for the projection `P` is the following:

```
1 P = glm::ortho(−screen_width, screen_width, −screen_height, ↩
      screen_height, near, far);
```

There is a second free camera, that can be used to navigate around the scene.

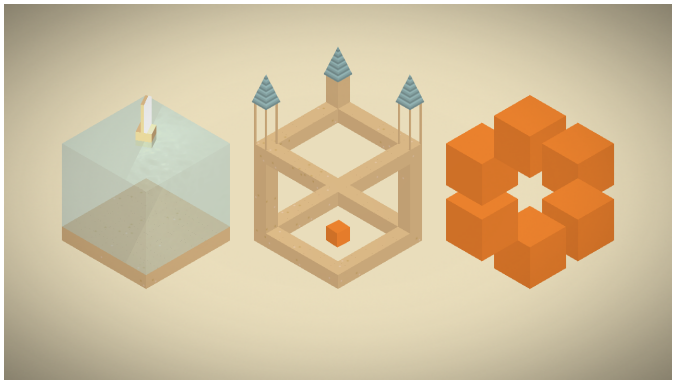

Figure 2: **The entire scene**

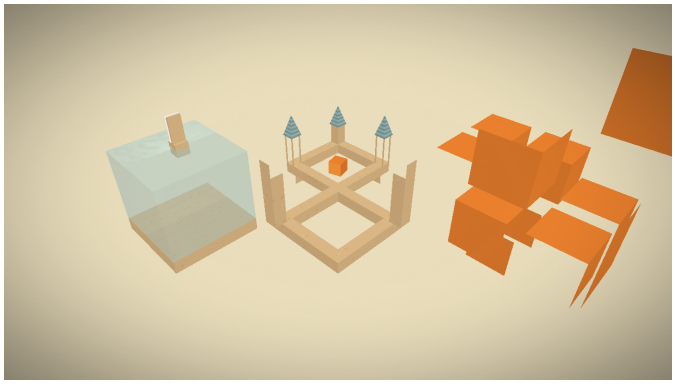

Figure 3: **The scene with a perspective projection**

Because of the differences between the orthogonal and perspective projections, some of the control had to be tweaked. For example to move left from it original position the camera would move (20.0 + left, 20.0, 20.0 + left). It is also possible to use a zoom by changing the code for `P`:

```
1 P = glm::ortho(−screen_width / zoom, screen_width / zoom, −↩
      screen_height / zoom, screen_height / zoom, near, far);
```

3.2 Lights
3.3 Phong shading and materials
3.4 Textures
3.5 Movement of meshes
3.6 Post-processing
3.7 Optimization

4 Future work

5 Conclusion

References

[1] “ustwo at nordic game 2014: Making of monument valley in unity.”