CS324 Coursework Assignment

University id: 1420684

January 16, 2017

1 Compiling, Running and Using the Program

1.1 Compiling

In order to compile the program on Linux, simply navigate to the main directory and use the makefile by running:

```
make
```

1.2 Running

Once the program has been appropriately compiled using the makefile, it can be run using:

```
./maze
```

1.3 Using the Program

To turn the camera left and right, use the 'a' and 'd' keys respectively. To move forward and backward, use the 'w' and 's' keys.

The player starts in the middle of the maze. The objective is to exit the maze without crossing through any walls (collision detection has not been implemented).

2 Design

2.1 Tree Structure and Recursive Functions

A tree-like system of objects of the class `game_object` has been used to represent the graphical elements of the maze. Each instance of the `game_object` class has a pointer to its parent, and an `std::vector⟨game_object⟩` containing its children, as well as physical information such as position, rotation and scale.

Whenever some amount of time passes and objects need to be updated in regards to their position, velocity and other such qualities, the `update()` method can be called on the root `game_object`. This will then recursively update each of its children, and the same will happen for other methods such as `display()`.

2.2 Graphical Component Inheritance

Each instance of `game_object` also has a `game_component` member field. This class is inherited by multiple others, namely `graphics_object`, `textured_graphics_object` and `light_object`. The nature of the inheritance means that each `game_object` within the tree can have a light, 3D model or the camera attached to it.

In the future, the implementation could be extended to include a camera_object, which would allow for the camera to move together with other objects and graphical components such as 3D models and lights. In addition, `game_object` could have its `game_component` member changed to a `std::vector⟨game_component⟩` for ease of adding multiple components to one object.

3 Features of the Program

A number of graphical features have been implemented. Objects can be easily moved in 3D space because of the tree-like implementation, however no physics features besides this have been implemented, such as collision detection.

3.1 Graphical Features

The maze has been constructed using a number of wall objects placed at different positions. These are instances of the `game_object` class, and their graphical components are set to an instance of the `textured_graphics_component` class. The maze also has ceiling and floor objects. These are all textured, and the appropriate textures are loaded into the program with the code used in the lab sessions. The textures are themselves located in the `./images/` directory.

The maze is lit up using a light that moves with the camera; its position is updated to the position of the camera each display cycle. If a `camera_object` was implemented, it would have been possible to make a light follow the camera much more simply; the camera object and a light could both be set as graphics components of a single instance of `game_object`.

A number of 3D torch models have also been placed around the maze. Had the `light_object` class been implemented, lights could easily be attached to these to make for realistic looking light sources.

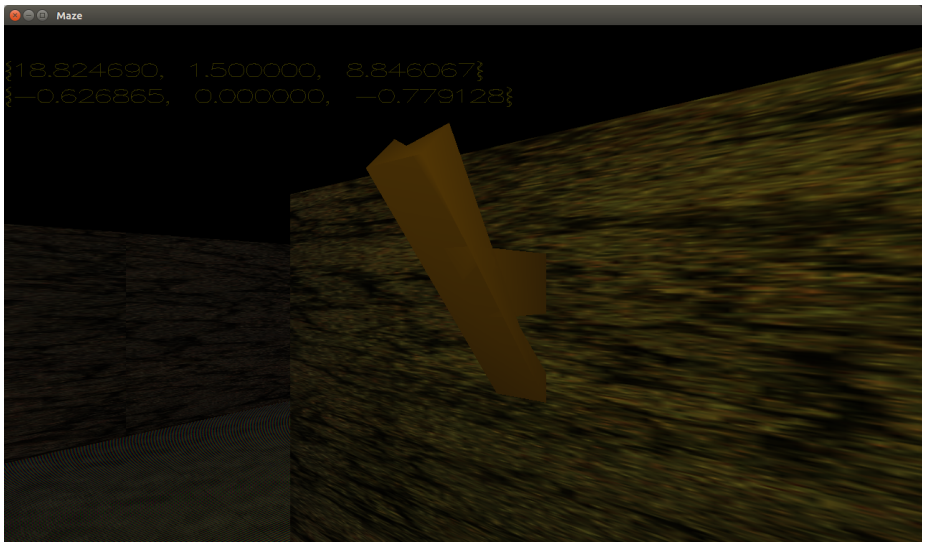

Figure 1: One of the torches inside the maze.

3.2 Main OpenGL Features Used

- Textures

- Solid-Fill
- Lighting (Ambient and Diffuse)
- Perspective Projection