Assignment 2 - Maze

Introduction

This assignment requested implementation of maze creation and rendering using OpenGL. This maze is to be created using standard methods of OpenGL, capable of generating from a given text file, and is to be experienced as if first person. Further aims is gain understanding and experience of computer graphics and develop Australian Computer Society's core body of knowledge for ICT professionals.

Implementation

The implementation procedure can be described as the following aspects of the program as requirement in the assignment specification. Furthermore, any problems that were encountered during the course of programming will also be discussed.

The final implementation used existing code, from Assignment 1, to provide an Object-Oriented approach. This broke the maze into three objects; a floor tile, a wall segment, and the destination marker. These objects were based on a cube shape that simplified the maze generation.

Read the Maze

Reading any provided text file was an initially difficult task, however the implementation was simple. Design choices were the main hindrance to this aspect; whether to use the file for additional class interface, to use the maze as provided, and how to store the maze. The final decision was to import that maze as global, two dimensional array. This allowed the maze to be read within multiple functions without having to be passed as a parameter; for some functions that were implement, the parameter list for function calling began to be extensive.

The maze array describes the matrix using integers for rendering and object instantiation, and hence, iteration through the maze could generate required objects and their positional transforms using the location of the element within the maze. This procedure simplified upon a previous design that generate the entire maze as a single collection of data that had potential issues with scaling and mid array pointers for texture mapping and lighting.

Rendering

The maze reading process appended the objects comprising the maze to a global structure containing further details about the virtual world. This significantly simplified the rendering process as each object had its own drawing method using local attributes.

As the required scope of the assignment was quite shallow, the design used some simplistic methods that would fail earlier than other possible methods of maze instantiation. Many objects were duplicates of previously created objects within the scene, just located at different positions. This method adopted the existing code's rendering paradigm. The author notes that having multiple objects of the same type (containing the same vertex data and shader programs) would require a higher level of hardware capability, however it was a design decision that saved a considerable amount of time. Further modification of code at a later date could significantly reduce the memory requirements of this program.

The final implementation read over the maze, created all the comprising objects with the required attribute data and transformations that could be used by the local draw method.

First Person Viewing

Using the same object hierarchy as previously implemented allowed each object to retain the state of the view transform matrix, though this use of the view matrix had not been encountered by the author before. Only slight issues were encountered due the order of matrix application to the vertex positions.

The majority of the problems that were encountered were the use of the view matrix itself. A considerable amount of time was dedicated to understand this transform as without it, the maze generation could not be verified as functioning. This aspect illustrated issues with object placement that formed the maze structure.

Movement using the view transform required implementation of a virtual “compass” and a localised spawn point to calculate the movement vectors, and the collision and teleport mechanics. As the view point was only to be located in discrete positions, the teleport mechanic was quite simple.

The final implementation passed the view and perspective transforms to the vertex shader to be applied to the object vertices.

Lighting and Shading

Applying Phong lighting was also an extensive process. Several prototype designs were implemented, including a brute force method of apply normals by using the maze design. The final method used programmer calculated normals for each of the three object types, and these were use for reflection handling within the fragment shader.

The final solution employed a “torch” approximation using an illumination profile, coupled with proximity and reflectance calculations. This method employed many empirical values, and the result was reasonable accurate considering its simplistic model.

Texture Mapping

The texture mapping was the most simple aspect of the entire program. All textures were loaded before object creation and each object had its own pointer to the required texture. As the texture handling was hard coded to the object class, using a solid coloured marker colour required a craft solution. Rather than edit the object class, a solid colour texture was loaded and applied to the marker. This technique should not be used in future OpenGL programs; each objects should be capable of handling its own colour.

Conclusion

The final implementation of this assignment was a simplistic approach that highlighted many further modifications to the general object class approach. However, the solution also illustrated the advantages of such a technique, further refining will be applied for future implementations. Assignment 2 successfully demonstrates understanding of ACS core body of knowledges, computer graphics aims. Assignment 2 was successfully completed.