Documentation for OpenGL Assignment 4

Project - 3D remake of the Pac-man

Xinglong (Luke) Chang
ID: 06227663
Institute of Information
& Mathematical Sciences
Massey University at Albany,
Auckland, New Zealand

Use OpenGL 4 to real time render a 3D scene, where the user can interact with the character in the scene

Keyword: OpenGL, GLSL, Phong lighting, real time rendering, game engine

1 Overview

This project aims to develop a 3D real time rendering game. The game is a 3D remake of the classic arcade game – Pac-man which developed by Namco in 1980. With the game engine I developed, the view angle is able to move around in the scene, the game can pause and restart in any time, and the character and map can be replaced easily.

2 Environment

- GLEW version: 1.12.0
- GLFE version: 3.1
- GLM version: 0.9.6.3 - OpenGL 3rd party math library
- SOIL version: July 7, 2008 - Simple OpenGL image library
- Assimp version: 3.0 (released July 2012) - mesh file loader
- The program was compiled and tested on Visual Studio 2013(x86) on a Windows 8 (x64) machine with Nvidia GeForce 970M graphic card.
- The minimal GLSL version is 4.1.

2.2 Environment Setup Notes:

- An order version of Assimp is used in this project due to it provides a binary prebuild for Windows x86. The mesh files in this project are Wavefront .obj files. The Assimp is only used to read basic data from the .obj file, such as vertices, normal, colour and indices, thus the Assimp version shouldn't be an issue for compiling the project.
- When testing on Mac OS, I noticed the program may running slow on Macbook pro with retina display screen, due to the antialiasing effect and the mediocre performance of the Intel integrated graphic card. However, use an external monitor to display the GLFW window can solve the problem.

3 Features

3.1 The camera position can move around in the scene along the X, Y and Z axes.
3.2 The game provides a restart function, where the character, stars and camera relocate to the game starting position.
3.3 The game can be paused and resumed at any time by pressing the space key. Unlike the restart key, there is no information loss during the pause. For example, the character, stars and camera position will stay where they were after resume the game.

3.4 The GLSL subroutine is used in fragment shader. It allows the colour of the model switches between the material colour and the texture. The model which only applies material colour will not bind with vertex buffer object (VBO) for texture UV coordinates. On the other hand, there is no colour RGB value saved in VBO when the model already has texture UV. With this technic, the data in VBO are allocated more efficiently.

3.5 The scene has an entering animation and most of the objects in the scene have moving animations, such as the stars are rotating along y-axes and the character has animation when turning and moving around.

3.6 The GLFW window is resizable, and it always keeps the correct aspect ratio.

3.7 The models are dynamically loaded from Wavefront .obj files. There is no need to define how many meshes in a model. Thus the map and character can be easily replaced with other models. For instance, change the *meshCharFileName* from "*pacman_ver2_flipNormal.obj*" to "*monkey.obj*" in main.cpp line 64, the character will change to the Blender monkey instantly.

3.8 The game can load with different map easily. I made 2 map for the game so far, a simple testing maze "*simpleMaze.obj*" and a larger maze "*maze3.obj*". To switch between maps, all it requires are a position text file, a map model .obj file and define some stars in the map. The position text file is used to indicate the valid position in the 2D grid world. The model files provides the 3D model for each object in the scene. The star can be added in any location by calling the *AddStar* method from the *World* class. All these change are done inside the main.cpp file.

3.9 The Phong lighting model is used for illumination. I combine the flat and smooth shading to provide a more realistic look.

4 Key control

The program shows the key control table at beginning. The table below shows the key control.

KEY	COMMAND
W	Move forward
A	Turn left
D	Turn right
S	Turn back
R	Game reset
SPACE	Game pause
[	Camera Y position ++
]	Camera Y position --
LEFT	Camera X position ++
RIGHT	Camera X position --
UP	Camera Z position ++
DOWN	Camera Z position ++
ESC	Exit game

Table 1 Key control table

The game is using quaternion for rotation. Because of the camera may move around, control the character rotation with global reference could be confuse. I use local object as reference, thus turning left means rotating to the character's left by 90 degree.

5 3D Model Drawing and Load Meshes

5.1 3D Model in Blender

The 3D models in this project are my original work drawn by Blender 2.74. There are 7 exported models which include 2 characters and 2 maps. The main character – Pac-man has 4 meshes and 5 materials. I use smooth shading on the eye and tongue areas for the realistic look, and flat shading on the body parts for the classic game feeling.

I tried to reduce the unnecessary vertices by removing the vertices and faces where inside the body. After reduction, the Pac-man has more total 13,320 vertices.

5.2 Face Culling Error with Blender Exported Files

In OpenGL, the front and back faces are determined by the winding order. However, the Blender cannot control the winding order directly. Thus the faces may display incorrectly after enable the back face culling in OpenGL. This problem solved by correcting the normal for each face of the object in Blender. To do this, select the model in edit mode, then select "Display face normals as lines" option in "Mesh Display" area from the right side menu. If the face normal is not pointing outside, then flip the target face normal. After the face normal correction, use GL_CCW to draw the vertices in counter clock wise order, then the face culling can be enabled, and the objects in OpenGL window should display correctly now.

5.2 Load Model with Assimp

The Assimp 3.0 cannot load the Blender output file ".blend" directly. I export the Blender model with Wavefront .obj format, since it has better support. When export a Blender file as .obj format, it produces 2 files, the .obj file and .mtl file. There is no need to link the .mtl in Assimp. The program can find the file in the same directory automatically. If the .mtl file does not exist in the directory, Assimp cannot find materials. When this happens, the complete model shows in grey colour.

In Wavefront .obj format, each mesh only has a single set of colour, the diffuse colour and specular colour. These data are saved in a separate .mtl file. Moreover, if a model uses 2 or more diffuse colours, it will generate 2 or more meshes with 1 material colour for each mesh.

The UV coordinates are optional depend on the texture option in Blender, but the normals are essential for the lighting. With this data structure, there is no need to store the RGB values for individual vertices. I only update the vec3 uniform variable for the RGB value once before draw the mesh. I use GLSL subroutine to switch between texture UV and material colour.

6 Code Structure

The ***global.h*** file holds the included libraries and Model class. The Model class provides an easy access to the shader program, vao, vbo, ebo, indices size, and the diffuse colour for each mesh.

The ***shader*** file holds the ***Shader*** class. It is used for loading the shader files, and retrieving the shader program index.

The ***render*** file provides the essential methods for rendering the scene, and keyboard call back functions.

The ***gl_utils*** file provides some utility methods such as read GLSL from a text file, and error logging.

The ***imageLoader*** file provides the texture loading methods by using the SOIL library.

The program is only using the GLM core file. The matrix transformations are implemented in the ***math_matrix*** file.

The ***mesh_utils*** file provides the Assimp supports for loading mesh files.

The ***world*** file defines the ***world*** class. It provides the access to the data of the character and map. The character status and 2D grid map also can be accessed from this class.

7 Game Engine

The entire data for the game are saved in the world class. It contains models for map, character and stars, the character's current location and direction, the location for each star, and the map in 2D grid world. Each model may have multiple meshes. When destructor is called, the program clean up the VAO, VBO, and EBO for each meshes automatically. There is an initial copy for resetting the game. It is saved when the class constructor has been called. The 2D grid world map is loaded from a text file, thus it becomes easy to update the map. There is no limited for the number of stars and the star's location. However the character can only move within the valid position inside the grid world.