Model_Loader

This project is a model loader created in C++ using the OpenGL framework. It aims to provide the ability to load simple .obj files and render them within an OpenGL window.

Getting Started

You can either:

- Open the x64 folder
    - Open the Build folder
    - Find Model_Loader.exe and run it
- Open Model_Loader.sln
    - Set the Model Loader as the current working project
    - Run the local debugger
- From here you'll be greeted with a simple Windows GUI. Simply navigate to file > open > then double click your file.
- To control your model, W A S D manipulated the position of the model, with W moving it closer, A moving it further away, D moving it to the right, A moving it to the left.
- With the arrow keys, you can rotate the object in the direction of each arrow key respectively. ***With the keys 1,2,3,4 you can scale the object by that amount.*** With the scroll wheel, you can zoom in and out of your model.

*Multiple models can be loaded at once, however due to the lack of texture, it may only be visible in wireframe mode. To enable this, click options -> Fill -> Wireframe.

Requirements

FreeGLUT, GLFW and GLEW are used within this project. To easily download these, in Visual Studio:

- Tool
- Nuget Package Manage
- PM Console
- Type Install-Package nupengl.core
- If a large number of standard headers appear to be missing, you'll need to retarget the project.

- To do this, right click the solution within Visual Studio and click, 'Retarget Solution'.

Developed using

Developed using Visual Studio on the Windows SDK Version 10.0.17763.0 with the platform toolset v141.

How does it work?

GUI (GUI.cpp)

Definitions

Definitions have been added for all button click ID's, to easily identify which button click is assigned to which event. For example, `#define FILE_MENU_NEW 1` clearly defines the new button in the file menu at ID 1.

LRESULT CALLBACK WindowProcedure (CallWindowProc)

A standard input manager for the Windows API. See here for more details: https://docs.microsoft.com/en-gb/previous-versions/windows/desktop/legacy/ms633573(v=vs.85 (https://docs.microsoft.com/en-gb/previous-versions/windows/desktop/legacy/ms633573(v=vs.85))

Parameters: HWND hwnd- The window you're processing messages for. UINT msg - The message you're responding to - See system defined messages here: https://docs.microsoft.com/en-gb/windows/win32/winmsg/about-messages-and-message-queues?redirectedfrom=MSDN (https://docs.microsoft.com/en-gb/windows/win32/winmsg/about-messages-and-message-queues?redirectedfrom=MSDN) WPARAM wp - Additional information, here we pass in our previously defined IDS. LPARAM lp - Currently unused, though could have additional information assigned to it.

This takes these parameters and when one of the system defined messages are sent, for example, `WM_COMMAND` activates a switch statement taking in the WPARAM, and activating a function based on its value. E.G, for `FILE_MENU_OPEN` which is ID 2, it opens a file open dialog to take in a file path.

void AddMenus

Parameters: HWND hwnd - The window you're targeting. This creates a menu system for the specified window. Currently creates a File,Options,Help menu and fills it with content.

void OpenFileDialog

Parameters: HWND hwnd - The window you're targeting.

Opens an OpenFileDialog to allow the user to specify a file to load. Defaults to .obj and only allows the user to select appropriate file types.

ColorDialog

Parameters: HWND hWnd - The window you're targeting.

Creates a Colour dialog box when called and allows the user to set the background colour.

Utilities

SplitString

Parameters: Const &string, char delimiter Const &string - the string you're splitting Char delimiter - the delimiter you're splitting by

struct Colour

A struct denoting a colour in floats between 0 and 1, used in the RGBtoFloat to easily change a colour value.

Colour RGBtoFloat(float r, float g, float b, float a);

Parameters: float r: The r value of an rgb vector. float g: the g value of an rgb vector. float b: the b value of an rgb vector.

Converts these values and returns values between 0 and 1, for suitable use within OpenGL.

struct Mesh

vector vertices; - The vertices of the mesh. vector texture_coords; - The texture coordinates of the mesh. vector normals; - The normals of the mesh. vector faces; - The faces of the mesh. string texturePath; - The path to a specified texture for the mesh. GLuint vertexBuffer; - A vertex buffer for a mesh. GLuint uvBuffer; - a uv buffer for a mesh. GLuint texture1; - A texture for a mesh. GLuint textureID; - the ID for the mesh texture.

methods

void SetupMesh(vector vertices, vector uvs, vector normals, vector faces, GLuint shader)

Takes in mesh parameters and binds the data to a buffer.

void DrawMesh(vector vertices, vector uvs, vector normals, vector faces , int counter) Draws the mesh on the currently selected OpenGLContext using the vertices and a previously setup buffer.

Splits a string by a delimiter and returns a vector string.

ParseDAE

Not implemented yet

OpenModel

Parameters: string path String path - The path of the object to load

Takes in a path as a console input and splits it at the last . Everything after and including the dot is taken as a file extension Which is then used to decide which method to run, I.E if the file extension is OBJ, it runs ParseOBJ. If this fails to open a file, it asks for further user input.

CheckForKeyboardInput

Parameters: GLFWwindow *window, int key, int scancode, int action, int mods GLFWwindow - The window you wish to set key callback for. int key - the keycode for user input The rest of the parameters are unused, however required for the method to be used with glfwSetKeyCallback

Checks for user input within the OpenGL window and manipulates any objects within the scene based on this.

CheckForMouseInput

Parameters: GLFWwindow *window, int button, int action, int mods Not currently implemented

CheckForMouseWheel

Parameters: GLFWwindow* window, double xOffset, double yOffset Zooms the character based on the yOffset of the mouse. I.E, if the mouse wheel is scrolling upwards, it moves closer, otherwise assume the action taken was scrolling downwards, and zoom out.

Display

Clears the background colour to a predefined colour, then renders any models within the objectsToRender vector by calling their DrawMesh() function

init

Sets the shader that's in use.

GLFWInit

Initialises GLFW and glew, creates a window and allows for user input. Contains a while loop that functions as an update loop, until the application is closed.

Main

The main function of the CPP, takes an input as a string and sends that as a parameter to OpenFile, then initialises GLFW.

void SetBackgroundColour(float r, float g, float b, float a)

Not currently in use, but sets the background colour of the OpenGL context.

Obj_Loader.cpp

ParseObj

Parameters: Ifstream file, string path Ifstream file - the file stream that should already be open String path - the path of the object to load

Read through an obj file line by line and checks for the first word of each line. Depending on what the word is, it'll split the following values and add them to a vector. For example, if the first word is V, we know the following values are the vertices of the model using SplitString, we split the line into a string vector and ignore the first value since it's the firstWord, i.e V The following values are then converted to floats and added to a GLfloat vector. This is then added to a vector of GLfloat vectors vertices.

Authors Notes

Where did you get the idea from? What did you start with? How did you make yours

unique? Did you start with a given project?

This program was developed as an extension to the model loader developed as part of CW1. As console input was finicky and not user friendly, it seemed appropriate to transfer this to a GUI based system. By looking at programs such as Blender, I was able to grasp what would be intuitive to a user and work from there. In the original project, the program would load the varying components, such as vertices and indices of a .obj, however be unable to render anything from there. Knowing how difficult my code was to extend, due to lack of planning, I reworked many elements of the program and split it into files that were logical to keep together. Unfortunately, this ended up with my Utilities.h class being somewhat of a 'god class' with most includes packed in there. Furthermore, I tried to implement structs wherever I could, to pack data into logically similar structures that could be referenced as a single object. This allowed me to import multiple obj files into one scene, easily.

Originally, I had planned to use Librocket to give me some easily implementable GUI within the OpenGL window, however through further research it appears to have been depreciated since 2014. As my original specification only stated WinAPI or Librocket, I was hesitant to use any additional libraries, however I feel the additions I made, while may not be as technically sound as those I would have implemented through additional libraries, make the project sufficiently unique.

In conclusion, while this program is not likely to be useful in any real world situation, it provided keen insight into my weaknesses within C++ and OpenGL while allowing me to further my understanding of both languages.

Youtube walkthrough:

https://youtu.be/kh9hThXslf4 (https://youtu.be/kh9hThXslf4)

Link to Github: https://github.com/10203040506070809/Model_Loader (https://github.com/10203040506070809/Model_Loader)