## MODEL LOADER PLUS

This is all about the `MODEL LOADER PLUS`

Model Loader Plus is based on the original previous project of `MODEL LOADER`, all of which commits' and previous work has been cloned over into this repository. As said the starting point of this project was to improve the original `MODEL LOADER` by implementing more advanced features such as Advanced Lighting, Bloom and Shadows as well as a GUI for editing aspects of the shaders. All of the mentioned is implemented within this project. See Below for a More under the hood investigation how all of this works.

## How to Run

[![](http://img.youtube.com/vi/P8T7DuKGDec/0.jpg)](http://www.youtube.com/watch?v=P8T7DuKGDec "How to Run Software in VS 2017")

- Click on the video above to view a walkthrough of running this project.
- Run in release mode in VS 2017, with toolset v141 and windows SDK 10.0.17763.0.
- Must be run in release for Regex to run at an appropriate speed (very long delays otherwise).
- When opening the solution the solution should just run as nuget packages are used for other header files and packages and ImGui is installed via the media Directory, if ImGui proves a problem include this directory via the project properties and VC++ dependencies.
- When upgrading to VS 2019 as well as upgrading the SDK version problems can occur with running.

## Current Functionality

- .obj and dae files can loaded and rendered
- Textures applied (UPDATED)
- Colours added from files
- Lighting applied using UVS/Normals (UPDATED)
- Multiple objects in render space possible
- Objects can be deleted from render space
- Textures can be swapped

- Appropriate memory management applied (UPDATED)
- Larger files can be rendered
- Objects can be Chose to be rendered
- wireframes can be viewed (UPDATED)
- light source is moveable (UPDATED)
- Blinn-Phong Lighting Implemented (NEW)
- GUI using ImGui implemented (NEW)
- Shadows are now implemented (NEW)
- Bloom is Implemented (NEW)

## On Program Run (What to do once program has started to be run)

- The console will open and print out what is being done while the window is being prepared
- Once everything is initialised a window will open and you will be greeted with a world consisting of a flat plane grid and 7 objects, 6 cubes and a boat object.
- The `Model Loader` window will now run and the controls below will control the software.
- Within the program, effects such as Bloom, Shadows and lighting will be demonstrated. By using the GUI windows effects can be turned on or off as well as tweaked to look how the user would like them. Controls below as well as the video explaining the program functionality further down below explains these gui windows further.

## CONTROLS

#### Camera

- Moving the mouse around will move around the x and y axis of whare the camera is pointing
- To move towards where the camera is pointing/ forwards, press <kbd>w</kbd>
- To move away from where the camera is pointing/ backwards, press <kbd>s</kbd>
- To move the camera left, press <kbd>a</kbd>
- To move the camera right, press <kbd>d</kbd>
- To move camera up on z axis (increase height), press <kbd>space</kbd>

- To move camera down on z axis (decrease height), press <kbd>LeftShift</kbd>

#### Enable Features

- Toggle mouse visibility on/off, press <kbd>1</kbd>

- Toggle wire-frame visibility for objects on/off, press <kbd>2</kbd>

- Toggle whether the light follows the camera or stays still on/off, press <kbd>3</kbd>

- Remove object from being rendered (Will remove objects in reverse loaded order down to a minimum of 3 objects), press <kbd>4</kbd>

- Swap textures on all objects <kbd>5</kbd>

- To close the program press <kbd>esc</kbd>

#### IMGUI Windows

All windows will require mouse control to be visible, press <kbd>1</kbd> in order to toggle the mouse on or off.

##### MAIN WINDOW

  - Here in the main window settings the ambient light level can be adjusted with the slider.

  - To change the light colour either click on the light colour box to pick a new colour or use the R,G,B sliders to fine tune the RGB value of the light.

  - Pressing the `Teleport light to me` button will do as said, it teleports the light to just above the camera position.

  - By toggling `Show Camera Settings` the Camera Settings Window will open.

  - By toggling `Show Feature Buttons` the Feature Buttons window will open.

  - Framerate is displayed at the bottom of this window.

 ##### CAMERA SETTINGS WINDOW

  - Here you can change camera settings such as camera speed and mouse sensitivity.

  - By moving the sliders up or down you can increase their retrospective setting.

 ##### FEATURE BUTTONS WINDOW

  - Here you can toggle bloom on or off, as well as access the toggle features mentioned above which are toggled with the keyboard keys.

  - By pressing the buttons these act as if you were pressing the relating keyboard key.

  - The Checkbox for bloom will toggle the bloom effect on or off

To close any new window, press the `close me` button.

For a more in depth investigation into how these windows work view the video below on the use of this program.

## How the Code Runs (What does what)

#### Start of program flow

Everything in the software starts from main, all includes at the top are related to packages and custom classes (header files) needed in order to run this software. The Packages manually installed are ImGui and the others through NuGet are, GLM, NupenGL and GLFW.

Classes along with their headers were created on a Object Oriented basis, so shaders, Meshes(objects), Loaders, Materials and Cameras all have their own constructors in order to be able to create a standard for each item. This way code can be contained as well as variables like VAO's, VBOS and such in the case of an object, as each object requires its own set of variablesto render. Much of this code was reused for the previous project. This also meant Main.Cpp didn’t end up too complex to understand as many of the classes code would be repeated many times or be a large section to implement into one file.

On program run, once all variables needed are initialised, main is called and the set up of the window and GLFW and GLEW is initialised along with all object file paths that will be needed. Shaders and objects are created and loaded into memory ready to be used when rendering the scene. Then initial shader uniforms are set for the objects.

Bloom and shadows are initialised by creating the necessary frame buffers, textures, depth maps and render buffer objects. These framebuffers and their textures are set to the correct shader uniforms ready to render the overlays and set the depth map needed for the shadows.

This is where the main render loop starts, think of this as each frame of the software. ImGUI window frame is created just before a check for input is made, these input checks work alongside call back functions for checking mouse and keyboard input. Then the first render of objects is run using the shadow shaders, this is used to then calculate the depth map for the shadows. After a second pass of the render is run but this time using the shadow depth map in order to calculate where the shadows will be using the object shaders.

After the Post FX frame buffer is bound ready for the bloom, a depth map is used to calculate the bright areas and then this is blurred in a horizontal and vertical direction and rendered to the screen as an overlay. After Bloom is rendered to the scene is rebound via the framebuffer and rendered using the post fx screen shaders.

At the end of the loop the GUI is finally rendered, first the variables that the GUI changes is passed to the relating shaders. Then each of the windows are created and are rendered (if the bool states they are visible) by using ImGui.

On the closing of the software, the rendered loop is jumped out of and all buffers, arrays and other memory items are cleared out of memory for a correct and clean shutdown.

#### Explanations to the main features

Shadows are created by first creating a depth map to work out what objects are in front of other objects from the perspective of the light source. By comparing the depth map of the scene from the cameras view and the light sources view, we can compare the depth value to see if there is a difference; if the difference of the fragments from the lights view is of a lower value than the cameras view then the scenes fragment must be in a shadow. From this Shadows can be calculated and rendered using its own frame buffer object.

Bloom, in the simplest way is created by creating a colour buffer texture and extracting all fragments that exceed a certain brightness level and render them to this texture. This texture is blurred and combined with a colour buffer texture of the original scenes view to add in the bloom/glow effect

ImGui/GUI works by using the package ImGui, each window is created and rendered to frame that is overlaid everything else in the scene. With each window relating static variables, these can be changed using a mixture of sliders, colour pickers, checkboxes etc. ImGui provides a way for a user to edit variables within the code without changing the code itself.

Lighting is implemented using the Blinn-phong lighting model, with this the angles between the camera and light source can be used to calculate the ambient light level, the diffuse and the specular highlights of the object. The specular highlights apply the ideas of the blinn-phong model in order to allow a reflection of greater the 90 degrees from the camera. All of these calculations are combined in order to achieve a somewhat realistic lighting model.

All code from the previous project remains within the project even if it is not used. Only main in main.cpp has removed code.

## Using the Software (A Video Walkthrough)

[![](http://img.youtube.com/vi/ZjbctL4lVps/0.jpg)](https://youtu.be/ZjbctL4lVps "Model Loader Plus")

Click on the above image to play a video explaining how the project works as well as a brief rundown of what's discussed below.

## What Makes This Project

#### What this project started with, as well as the start of development

In the start of this project the previous model loader was used, to see the code for this look back to the commit on 29th November 2019 called 'Merge branch 'master' of https://github.com/SOFT356/S356-bhaggar'. This was the last commit of a previous project that loaded .obj files and .dae files and rendered them with very basic shaders. By using this previous project, edits were made to use the creeper.obj objects as a cube to input into a base world purely to show off the aspects of this project. the texture of the creeper was changed as well as a new manually coded obj and mtl files for a grid floor ready to display shadows once they were implemented.

The parts used from the previous project was the creation of an openGL window, the overall structure of the project as well as the main shaders used in order to render and display objects. The camera, camera controls and keyboard input was also used from the previous project. Some aspects were modified and up in the current functionality section you can see what was updated from the original code.

By using this previous section, a quick start could be made so the real implementations could begin, these implementations were advanced lighting updates, Bloom, Shadows and a GUI.

#### What's unique

When comparing this project to other projects out there, such as the ones taught on LearnOpenGL.com, this project not only implements advanced features but also combines them together, such as using bloom and shadows. This wasn't exactly simple as with openGL understanding what needed to be created and bound to buffers in what order was important to get everything rendered to a scene. Furthermore combining separate shaders into one shader meant that things like post processing effects could all be placed in one code file, although this could make the readability of the code a little harder, it made the running of shaders in the render loop a little simpler.

The use of the GUI also is unique to tutorial and projects online, as aspects of lighting can be changed by the user, and not by editing the code itself, this also made the debugging simpler as finding the right standard values could be found by editing values for the shaders with sliders in the GUI.

#### What works well

What's good about this project is the lighting as a whole, the shadows, bloom, blinn-phong lighting and how all this works together. With all these together you get a higher realism of lighting closer to AAA games and an understanding of how further development could get closer to a AAA game.

The shadows work well, mostly, as multiple shadows can be cast onto the grid and adds a strong aspect of lighting to the rendered scene. Although these could be considered a small addition by looking at them in the rendered scene and although the coding of these are very complex, it is worth it. By looking at the boat in the rendered scene the way that shadows cast on itself makes the lighting look much more realistic, rather than just having a dark side and a light side.

Bloom is very effective at showing where the light is hitting objects the most, although with some light settings a bit too effective as it is almost a bit too bright. However, as for making the light emitters look like light emitters this is great, the blur/glow also decreases depending on how bright the light source is which is a realistic as it gets, this also means that when it isn't a bright light being used the blur is almost invisible making it seem like the light is not on in a way.

The overall lighting is fantastic when it is all working together, with the shadows and bloom working with the upgraded blinn-phong lighting, the scene looks near finished and complete when it comes to rendering objects. If these shaders were used to create a world for the game the graphics would look realistic enough to really create immersion for a player of a game.

One last noting feature is the grid where all the objects hover above, this was made by manually coding a obj and mtl file, this was a very satisfying element as it really meant that I understood how obj and mtl files are constructed.

#### What could be added/ improved

Currently shadows look slightly off in the rendered scene, they don't seem to be in a 100% accurate position, this means the overall effect is spoilt from such a small error. A possible cause for this could either be due to peter panning of the shadows or the fact the light source needs to be further away, this is because the angle used to calculate where the light is coming from isn't coming from the light object itself but rather from the general direction of the light source. Shadows also seem a little too dark, especially on the sides of objects away from the light making them almost completely black, by editing shader values and by how much shadows decrease the frag colour this fix this issue.

As mentioned above bloom sometimes is seen as a bit too much, obstructing the scene by covering objects with light. Perhaps by messing around with the exposure and gamma of the shader, small changes could correct this. Furthermore, when researching how to implement bloom, it was mentioned that HDR and bloom work very well together, maybe this could be an area to look into for a fix. Bloom also seems to implement major framerate issues; this is believed to be due to when the blur is rendered to the scene multiple times with the renderQuad function. Other frame rate issues are most likely due to the loops used in the main render loop.

The main of the program could also be tidied by creating further classes and headers to abstract some of the logic of the program, one main class that could be made is a framebuffer class to reduce the re-writing of code, however as understanding how all these new implementations work was only truly achieved once everything was developed it is only now that this could be achieved.

As for additions, in addition to the above mentions downfalls, an area I would like to approach is the actual material of an object being implemented for the objects. Also, normal mapping along with different textures would make the objects themselves look much better with a 3d effect as opposed to just flat faced edges.