

Improved Model Loader

SOFT356 - Programming for Entertainment Systems

Coursework 2 - 10518778

Function

This program is an updated version of a model loader I previously worked on. As before, it loads an .obj* file and renders it in a window.

Addition features in this version:

- Added more shader types and option to select a shader to load with
- Can choose between a few object types
- Full camera movement implemented
- Can spawn an object up to 10 times
- Hold 'P' to view wireframes of objects

Developed with Microsoft Visual Studio Professional 2019 using OpenGL Version 4.0.

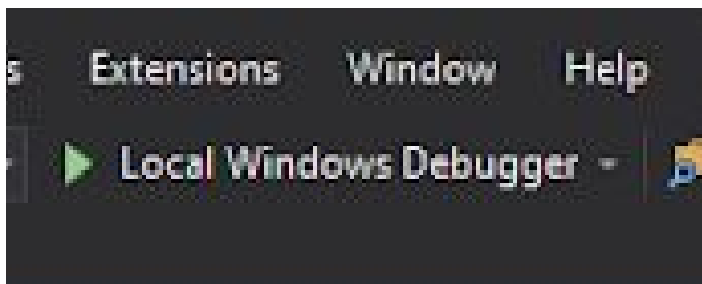
*https://en.wikipedia.org/wiki/Wavefront_.obj_file

User Guide

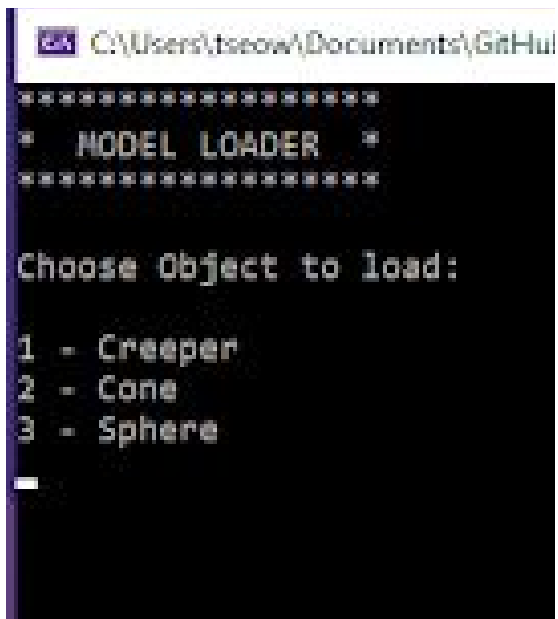
How to Load Object

Step 1: Locate the "SampleProjects.sln" file and open it (Note: you must have Visual Studio installed with nupengl.core and glm imported).

Step 2: Click the green "Local Windows Debugger" button to start the program.



Step 3: Choose which object you who like to load by entering the corresponding number and press enter.

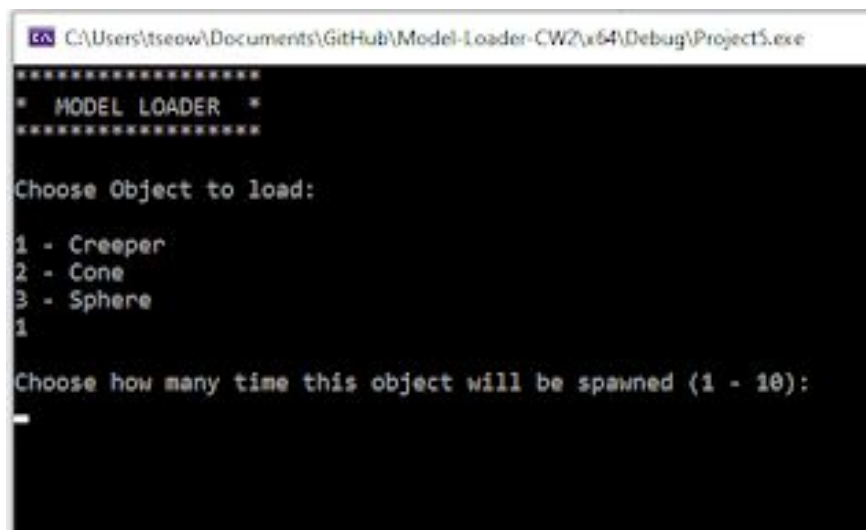


```
C:\Users\tseow\Documents\GitHub\
*****
*  MODEL LOADER  *
*****

Choose Object to load:

1 - Creeper
2 - Cone
3 - Sphere
_
```

Step 4: Choose how times you what to spawn this object then enter the amount (from 1 to 10) and press enter.



```
C:\Users\tseow\Documents\GitHub\Model-Loader-CW2\x64\Debug\Project5.exe
*****
*  MODEL LOADER  *
*****

Choose Object to load:

1 - Creeper
2 - Cone
3 - Sphere
1

Choose how many time this object will be spawned (1 - 10):
_
```

Step 5: Now choose which shader you would like to use by entering the corresponding number and press enter, a window should pop up.

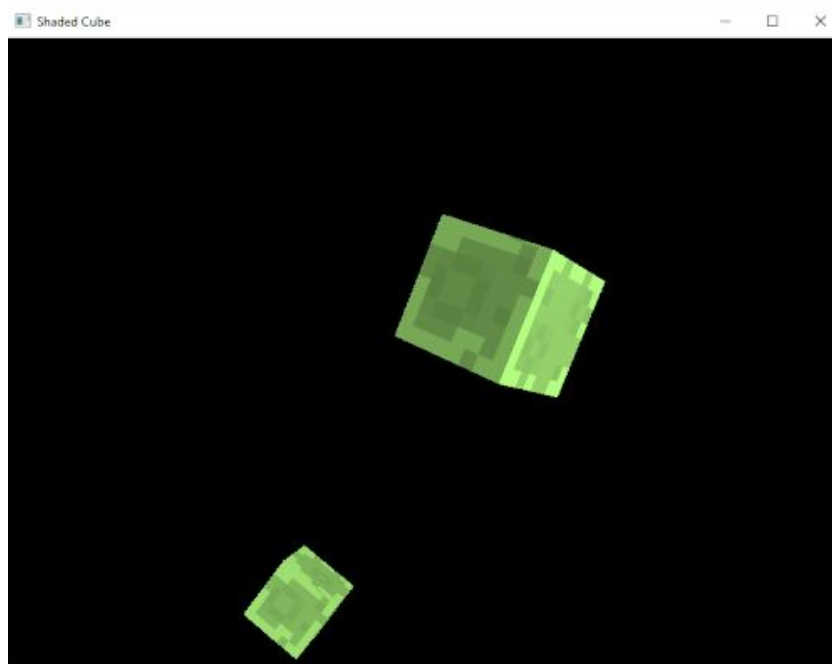
```
CAUsers\tseow\Documents\GitHub\Model-Loader-CW2\x64\Debug\Project5.exe
*****
*  MODEL  LOADER  *
*****

Choose Object to load:
1 - Creeper
2 - Cone
3 - Sphere
1

Choose how many time this object will be spawned (1 - 10):
2

Choose shader type:
1 - Textured - Normal
2 - Textured - Flat
3 - No Texture - Normal
4 - No Texture - Flat
5 - Cel
```

You should see a window like this:



Controls in Render Window

Camera:

W - Forward

A - Left

S - Back

D - Right

Mouse - Look around

Scroll Wheel - Zoom in/out

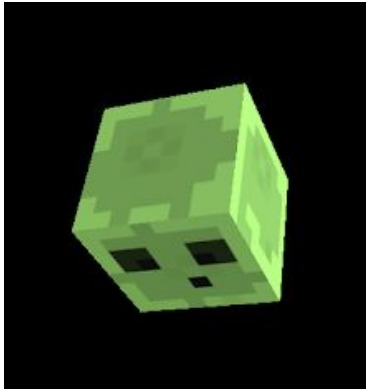
Other:

P - Hold to view wireframe

ESC - Close render window

Shader Types

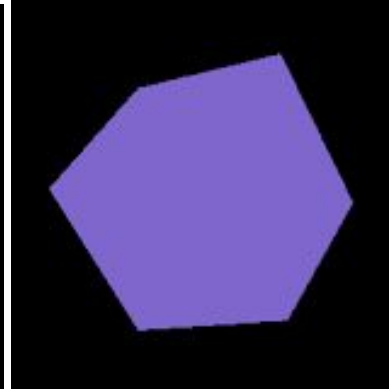
Textured - Normal Shader



Textured - Flat Shader

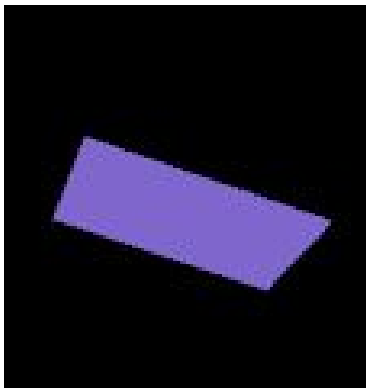


No Texture - Normal Shader

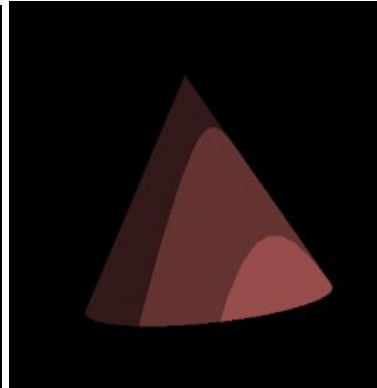


No

No Texture - Flat

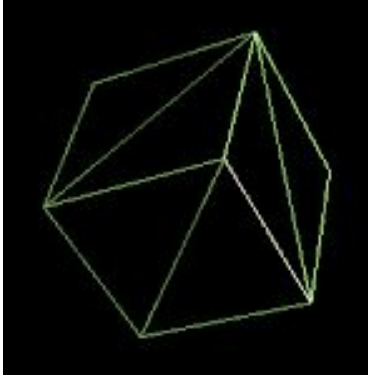


Cel Shader

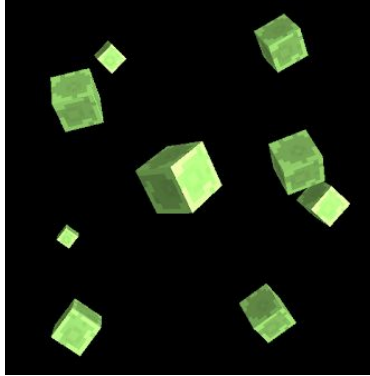


Other Features

Wireframe



Multiple Spawns



Project Starting Point

As implied by the title, this project is a continuation of a model loader I have made, however this is less focused on the “loader” part and focuses on adding more features/interactivity to the program, therefore no extra file types have had any added support.

The original model loader was only able to read an obj file and an mtl file and render it with a texture. There was no lighting, no movement, and only a single default shader.

As a note, this program is based on code provided by Swen Gaudl (<https://swen.fairrats.eu>)

What's new?

Lighting

The first thing was to implement lighting. This was done by implementing what I had already done into a different project provided by Swen Gaudl that had basic lighting implemented. This meant being able to load obj and mtl files and pass through the data into the rendering pipeline.

Camera

This was implemented with the help of a tutorial from learnopengl.com. It allows the user to move around with the WASD keys, look around with the mouse and zoom in and out with the scroll wheel. Additionally the object rotates on its own axis.

Spawning More Than One Object

This basically redraws the same object set in memory based on the input by the user. The object is transformed into a different set position.

Multiple Shaders

It is now possible to switch between different shaders (Normal/Flat/Cel). The user can select one before loading/rendering the object.

As an extra the user can view the wireframes of objects by holding 'P' down.

Code Structure

The loader and render are both in a single file.

There are six main functions:

```
loadTexture()  
init()  
display()  
loadFile()  
loadMTL()  
main()
```

Overview

In main() the user is asked to choose an object type, spawn amount and shader type. The corresponding obj and mtl files are loaded with loadFile() and loadMTL() which extracts all the data from the files. The data is then passed into init() to create and populate buffers. After that, in a while loop, display() is called that displays the model in a window.

How data is read from an obj file

Using fstream the file is opened. Then in a while loop each line is read individually and checked for what piece of data it holds e.g. a line the begins with "vt" holds data about the texture coordinates.

Example:

```
If (line.substr(0, 2) == "v ")           //this line checks for vertices using .substr()  
{  
    glm::vec3 vertex;                     //a temp variable  
    istringstream s(line.substr(2));      //removes the first two char i.e. "v "  
    s >> vertex.x; s >> vertex.y; s >> vertex.z; //extract data and add into temp vec3  
    temp_vertices.push_back(vertex);      //pushes back into vector  
}
```

Cel Shader

The bulk of this shader happens in the fragment shader so the vertex shader only need to pass through the normal and the light direction, using these two it calculates the intensity of the light and renders that pixel based on the intensity. So, the intensity is split into bands and a specific colour is rendered based on where the intensity falls in.

Vertex Shader

```
void main()
{
    // view-space coordinate
    vec4 P = mv_matrix * vec4(vPosition,1.0);

    vec3 L = lightPos - P.xyz;
    // calc the view vector

    //Normalise
    L = normalize(L);

    gl_Position = p_matrix * P;

    lightDir = L;

    normal = vec3(mv_matrix * vec4(vNormal,0.0));

    gl_Position = p_matrix * P;
}
```

Fragment Shader

```
void main()
{
    float intensity;
    vec4 color;
    vec3 n = normalize(normal);
    intensity = dot(vec3(lightDir),n);

    if (intensity > 0.95)
        color = vec4(1.0,0.5,0.5,1.0);
    else if (intensity > 0.5)
        color = vec4(0.6,0.3,0.3,1.0);
    else if (intensity > 0.25)
        color = vec4(0.4,0.2,0.2,1.0);
}
```



```

        else
            color = vec4(0.2,0.1,0.1,1.0);
        gl_FragColor = color;
    }

```

Flat Shader

There are two parts to this shader. The frag shader sets the colour of the object to a single solid colour while the vert shader it makes the object “2D” by just setting the z value of the vertex to zero.

Vertex Shader

```

void main()
{
    void main(void)
    {
        // view-space coordinate

        vec4 t = vec4(vPosition,1.0);
        t.z = 0.0; //Flatten

        ... (rest of code has been omitted as not needed to explain how the effect works)
    }
}

```

Fragment Shader

```

void main()
{
    fColor = vec4(0.5, 0.4, 0.8, 1.0);
}

```