SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.017 Graphics and Visualization

Assignment 1 – OpenGL Mesh Viewer

Handout date: 2024.02.01

Submission deadline: 2024.02.12, 11:59 pm

Late submissions are not accepted

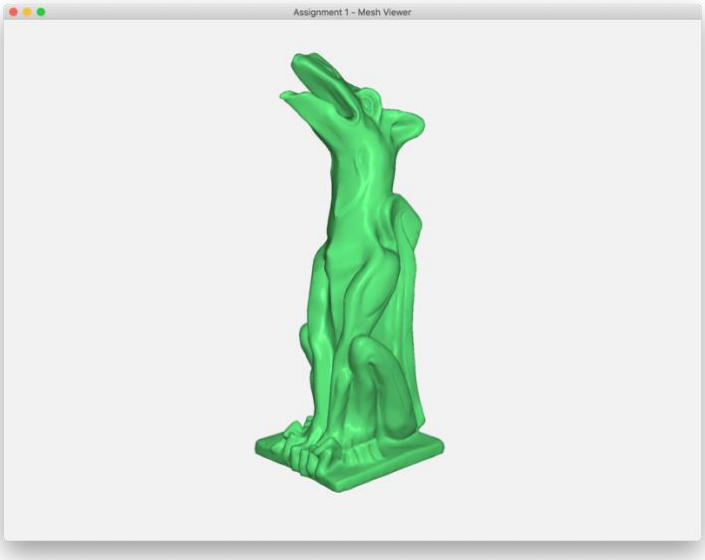

Figure 1. Expected program output by loading the garg.obj model.

In this assignment, you will load, render, and interact with a 3D mesh model saved as an OBJ file. The framework code for this assignment extends the one from last week. “TODO” comments have been inserted in main.cpp to indicate where you need to add your implementations.

Overview

This assignment consists of 4 parts:

1. Mesh Loading: Load a mesh model saved as an OBJ file
2. Mesh Display: Render the mesh model with OpenGL
3. Mesh Coloring: Change rendering color of the mesh model
4. Mesh Transformation: Rotate, translate, and scale the mesh model

Three test datasets (garg.obj, mickey.obj, and sphere.obj) are provided in the code framework (data folder). Make sure that you're able to load, view, and interact with the three provided mesh files without crashing; these are the only three files we'll test your program on.

Mesh Loading

In the code framework, we have provided several 3D meshes in OBJ format. It is your job to write the code to load and display these files. OBJ files are a fairly standard format that can describe all sorts of shapes, and you'll be handling a subset of their functionality.

Let's look at sphere.obj. It's a big file, but it can be summarized as follows:

```
#This file uses ...
...
v 0.148778 -0.987688 -0.048341
v 0.126558 -0.987688 -0.091950
...
vn 0.252280 -0.951063 -0.178420
vn 0.295068 -0.951063 -0.091728
...
f 22/23/1  21/22/2  2/2/3
f 1/1/4  2/2/3  21/22/2
...
```

Each line of this file starts with a token followed by some arguments. The lines that start with v define vertices, the lines that start with vn define normals, and the lines that start with f define faces. There are other types of lines, and your code should ignore these.

Your first task is to read in all of the vertices ("v") and all the normals ("vn") into an array (`verList`). The elements should be stored in the following order in the array:

```
v1.x, v1.y, v1.z, vn1.x, vn1.y, vn1.z,
v2.x, v2.y, v2.z, vn2.x, vn2.y, vn2.z,
...
```

where vn1 is the normal of the first vertex v1, vn2 is the normal of the second vertex v2, and so on. The correspondence between the vertices and normals is defined in the faces ("f").

Understanding the faces ("f") is a little more difficult. Each face is defined using nine numbers in the following format: *a/b/c d/e/f g/h/i*. This defines a face with three vertices with indices *a*, *d*, *g*, and respective normal *c*, *f*, and *i* (you can ignore *b*, *e*, and *h* for this assignment). Hence, the correspondence between the vertices and normals is:

- vertex with index *a* corresponds to normal with index *c*;
- vertex with index *d* corresponds to normal with index *f*; and
- vertex with index *g* corresponds to normal with index *i*.

The general OBJ format allows faces with an arbitrary number of vertices; you'll just have to handle triangles in this assignment. Note that you'll need to fill another array to store the faces (`triList`).

In main.cpp, `verList` is defined as an STL vector of float. An STL vector is simply a list of arbitrary objects. In this case, it is a list of float numbers.

```
vector< float > verList;
```

To add a new entry to this array, use push back:

```
verList.push_back( 0.1f );
```

There are several ways to iterate over an STL vector. Here's an example of using indices:

```
for (unsigned int i=0; i < verList.size(); i++)
{
     float v = verList [i];

     // do something with v
}
```

Your task is to fill in the function

- `int LoadInput(vector<float> &verList, vector<unsigned> &triList)`

to achieve the above task of loading OBJ models; i.e., read the data in an OBJ file and save them in the vectors (`verList` and `triList`). Note that the index of vertices, normals, and faces in OBJ files starts from 1 while the index of an STL vector starts from 0. This difference should be taken into consideration when storing data in `verList` and `triList`.

Mesh Rendering

We will use shader to draw the mesh with data stored in `verList` and `triList`. Note that this part has been implemented for you in main.cpp and shaderSource.h so you do not need to write code for this part. You are suggested to read and understand the code about mesh rendering. See Figure 1 for an example rendering result.

Mesh Coloring

Add the ability to change the color of the displayed model. Your task is to fill in function

- `void SetMeshColor(int &colorID)`

such that pressing the c key will toggle through four colors stored in the array `colorTable`.

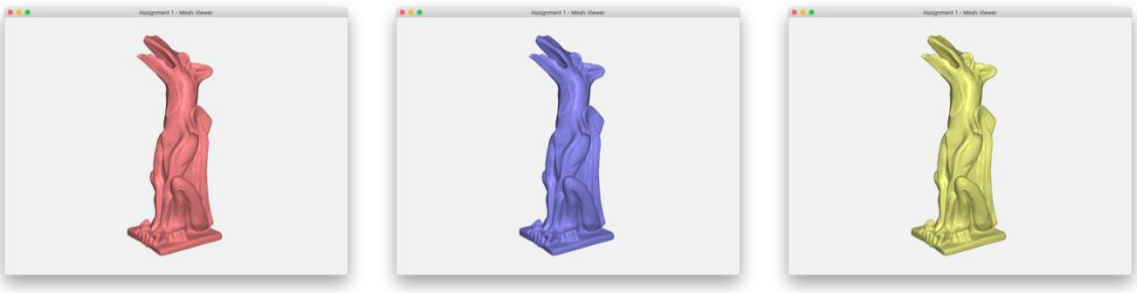

Figure 2. Expected program output by rendering the garg.obj model in three different colors.

Mesh Transformation

Add the ability to transform the displayed model, i.e., rotating, scaling, or translating the model, by using the mouse. Your task is to fill in the following three functions:

- `void RotateModel(float angle, glm::vec3 axis)` to rotate the model;
- `void TranslateModel(glm::vec3 transVec)` to translate the model;
- `void ScaleModel(float scale)` to scale the model.

For each transformation (i.e., rotation, scaling, and translation), you should first construct a transformation matrix based on the function's parameter(s) (e.g., translation vector `transVec`), and then multiply the transformation matrix with `modelMatrix` defined in the code framework.

You can test your code by using mouse, which has been provided in the code framework.

- Left mouse drag will rotate the object around a mapped axis based on the mouse motion (this technique is called "ArcBall Rotation").
- Shift + left mouse drag will translate the object in the screen space based on the mouse motion.
- Scrolling the mouse will scale the object to make it either smaller or bigger depending on the mouse scrolling direction.

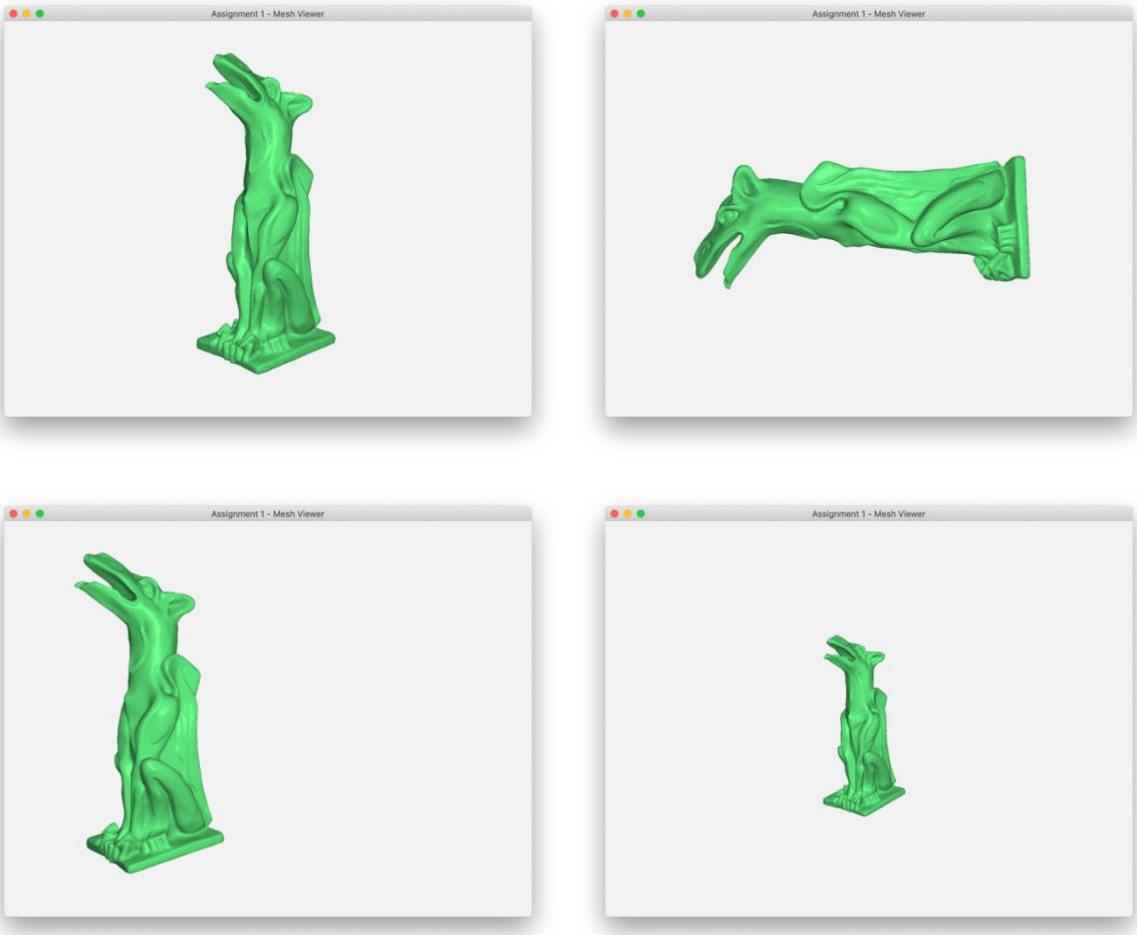

Figure 3. Given a model with an initial pose (top left), expected program output by rotating (top right), translating (bottom left), or scaling (bottom right) it, respectively.

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Grading

Each part of this assignment is weighted as follows:

- Mesh loading: 50%
- Mesh coloring: 20%
- Mesh transformation: 30%

Note: in this assignment, you can complete all above tasks by filling in these five functions:

```
int LoadInput(vector<float> &verList, vector<unsigned> &triList)
void SetMeshColor(int &colorID)
void RotateModel(float angle, glm::vec3 axis)
void TranslateModel(glm::vec3 transVec)
void ScaleModel(float scale)
```

Assessment of this assignment will be based on your code in these functions.

Submission

A .zip compressed file renamed to AssignmentN_Name_I.zip, where N is the number of the current assignment, Name is your first name, and I is the number of your student ID. It should contain only:

- The **source code** project folder (the entire thing).
- A **readme.txt file** containing a description of how you solved each part of the assignment (use the same numbers and titles) and whatever problems you encountered. If you know there are bugs in your code, please provide a list of them, and describe what do you think caused it if possible. This is very important as we can give you partial credit if you help us understand your code better.
- A couple of **screenshots** clearly showing that you can display the three mesh models and that you can interact with them (change color and views) by using the mouse.

Upload your zipped assignment to e-dimension. Late submissions receive 0 points!