T-111.4310 Vuorovaikutteinen tietokonegrafiikka
Programming Assignment 0: OpenGL Mesh Viewer
Due Sunday, September 30 by 23:59

1 Getting Started

First open the project by double clicking the file 'assignment.sln'. Let's start off by looking at `App.hpp`, and `App.cpp`. These files, and the framework library, contain a fully functional application that displays a teapot. Other than that, it's not very interesting. To compile this application, navigate to 'Build → Build Solution'.

Once you've successfully built the executable, run it by pressing F5. It should display a teapot. Yes, that's all it does. It's now your job to make this application a bit more interesting by modifying the code.

2 Requirements

2.1 Color Changes

Add the ability to change the color of the displayed model. Right now, the color is set to [0.5, 0.5,0.9] (RGB), which is a boring light blue. Your task is to wire the `c` key to toggle through several other colors (feel free to choose which colors you want). How do you handle keyboard events? Notice that, when you press keys while the application is running, the console says something like this:

```
Unhandled key press U+0043.
Unhandled key press Up.
Unhandled key press Down.
Unhandled key press Left.
Unhandled key press Right.
```

The code that prints these messages is in the `App::handleKeyDown` function. Modify the code to handle the `c` key appropriately. A reasonable way to do this might be to have the `c` key increment some sort of class member counter variable and then use that variable to select a color in the `renderFrame` function. Note that you don't have to call a function to tell OpenGL to swap buffers. This is handled elsewhere.

2.2 Light Position Changes

Add the ability to change the position of the light. In the code, the light is placed at [1.0,1.0,5.0]. Wire the arrow keys to change the position of the light. More specifically, the left/right arrow keys should increment/decrement the first value of the position by 0.5, and the up/down arrow keys should do the same for the second value. This can be done quite similarly to the suggested method for the previous requirement.

2.3 Mesh Loading and Display

Once you have completed the above requirements, we can move on to the tough part: loading new objects. In the sample code, 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") into an array (`m_vecv`) (or any other data structure that allows you to quickly reference the ith element). Then, do the same for the normals ("`vn`), loading them into another array (`m_vecn`). We have suggested such class member variables in `App.hpp`.

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 normals c, f, and i (you can ignore b, e, and h for this assignment). The general OBJ format allows faces with an arbitrary number of vertices; you'll just have to handle triangles.

So let's say you have the vertices and normals stored in `m_vecv` and `m_vecn`. Then you'd draw the aforementioned triangle using the following code:

```
glBegin(GL_TRIANGLES);}
glNormal3d(m_vecn[c-1][0], m_vecn[c-1][1], m_vecn[c-1][2]);
glVertex3d(m_vecv[a-1][0], m_vecv[a-1][1], m_vecv[a-1][2]);
glNormal3d(m_vecn[f-1][0], m_vecn[f-1][1], m_vecn[f-1][2]);
glVertex3d(m_vecv[d-1][0], m_vecv[d-1][1], m_vecv[d-1][2]);
glNormal3d(m_vecn[i-1][0], m_vecn[i-1][1], m_vecn[i-1][2]);
glVertex3d(m_vecv[g-1][0], m_vecv[g-1][1], m_vecv[g-1][2]);
glEnd();
```

You may be wondering why there are all those minus-ones. It's because the faces index vertices and normals from 1, and C/C++ indexes from 0. If you have this implemented, the rest is fairly straightforward: you just have to loop over all the faces to draw the complete mesh.

In `main.hpp`, m_vecv is defined as an STL vector of Vecfs. An STL vector is simply a list of arbitrary objects. In this case, it is a list of Vec3f objects.
`vector<Vec3f> m_vecv;`

To add a new entry to this array, use push_back:
`m_vecv.push_back(Vec3f(0,0,0));`

There are several ways to iterate over an STL vector. Here's an example of using indices (if you're interested in learning more about STL, check out the documentation at http://www.sgi.com/tech/stl/):

```
for(unsigned int i=0; i < m_vecv.size(); i++) {
   Vec3f &v = m_vecv[i];
   //do something with v[0], v[1], v[2]
}
```

Please also keep in mind that you'll need another array to store the faces (perhaps `vecf`). It may be tempting to try to draw them as they are read from the OBJ, but OpenGL requires you to redraw the model whenever the window is obstructed or resized (and also when you change the color or lightning).

Your final application should ask the user for an object file when the key 'L' is pressed. This behavior is already implemented in the function `handleKeyDown` . The class Window has a member function `showFileDialog` which opens a normal window to specify a file. You only need to pass the returned name to the `loadMesh` function you implemented.

Open a file stream with the fileName

```
ifstream in (fileName.getPtr(), ios::in);
```

For example, to read a single line of data from the stream (all characters up to the next newline):

```
char buffer[MAX_BUFFER_SIZE];}
in.getline(buffer, MAX_BUFFER_SIZE);}
```

in.getline will return zero at the end of the file. You can use this fact to step through each line in the file. Once you have an array of characters (the text from a single line of the file), you can parse it using a stringstream object. Create a stringstream object from an array of chacters (buffer) as follows:

```
stringstream ss(buffer);
```

Remember that faces are stored in the following format in an OBJ file:

```
f 22/23/1 21/22/2 2/2/3
```

so you have to replace the '/' characters before using the ">>" operator.

```
for (char *c = &buffer[0]; *c != '\0'; c++)
        {
            if (*c == '/')
                *c = ' ';
        }
```

Now that you have a stringstream object, you can read tokens (separated by spaces) from the buffer in order by using the ">>" operator. For example, given the input string "v 1.0 1.1 1.2", in the following code:

```
Vector3f v;
string s;
ss >> s;
ss >> v[0] >> v[1] >> v[2];
```

will put the value "v" into s, and load the values 1.0, 1.1, and 1.2 into v[0], v[1], and v[2]. Note that you can compare the string objects to constant strings using the regular "==" operator.

```
if (s== "v") {
      //do something
}
```

Make sure that you're able to load and view the three provided files without crashing; these are the only three files we'll test your program on.

You may want to run the provided sample solution `example.exe` to get an idea of how your application should work.

3 Extra Credit

Here are some ideas (sorted roughly by increasing level of difficulty) that might spice up your project. The amount of extra credit given will depend on the difficultly of the task and the quality of your implementation. In addition, feel free to suggest your own extra credit ideas! Just because it's not on this list doesn't mean we won't give you some extra points (although if it's a big addition, make sure you run it by the course staff first just to make sure).

Easy

- The sample solution lets you hit `r` to spin the model. Implement this functionality in your code. See `Timer` class in framework/base and note that `App::handleEvent` is called periodically.
- Display the model using OpenGL display lists or vertex buffer objects for higher performance rendering.
- Modify the code so that the `c` key smoothly transitions between different colors (rather than just toggling it).

Medium

- Implement a mouse-based camera control to allow the user to rotate and zoom in on the object. Credit will vary depending on the quality of the implementation. See `App::handleEvent`, `Window::EventType`, and `Window::Event`.

Hard

- Large meshes are quite difficult to draw and process. For interactive applications, such as video games, it's often desirable to simplify meshes as much as possible without sacrificing too much quality. Implement a mesh simplification method, such as the one described in Surface Simplification Using Quadric Error Metrics (Garland and Heckbert, SIGGRAPH 97).

4 Submission

As a final step, write a `README.txt` that answers the following questions:

- Did you collaborate with anyone in the class? If so, let us know who you talked to and what sort of help you gave or received.
- Were there any references (books, papers, websites, etc.) that you found particularly helpful for completing your assignment? Please provide a list.
- Are there any known problems with your code? If so, please provide a list and, if possible, describe what you think the cause is and how you might fix them if you had more time or motivation. *This is very important, as we're much more likely to assign partial credit if you help us understand what's going on.*
- Did you do any extra credit? If so, let us know how to use the additional features. If there was a substantial amount of work involved, describe what and how you did it.
- Got any comments about this assignment that you'd like to share? Was it too long? Too hard? were the requirements unclear? Did you have fun, or did you hate it? Did you learn something, or was it a total waste of your time? Feel free to be brutally honest; we promise we won't take it personally.

Submit the following in a .zip archive on Optima folder "Assignment 0"

- The whole Visual Studio project with your additions.
- The aforementioned `README.txt` file.
- Any additional files necessary to run your program.