Project 2 - Texture Mapping and Polygon File Format (PLY)

CS 1566 — Introduction to Computer Graphics

Check the Due Date on the Canvas

The purpose of this project is for you write a program to display 3D objects with texture mapping. The main control and display of this project should come from the Project 1. For this project, there are two types of objects:

1. computer generated sphere, and
2. objects stored in a graphics file.

Again, make sure your project 1 works properly.

Computer Generated Sphere (30 Points)

For this part, you need to use the 8th ball texture (from Lab 5) and put it on your computer generated sphere. Note that a billard ball has two identical side. So, you can use the given texture for both front and back side or top and bottom. Note that the green strap may not align correctly (not dead center in the texture). Try to find tune it as best as you can.

Object and Texture Coordinate (or Color) from PLY (70 Points)

In thi spart, you have to write a program to read a simple graphic file format. The file format that we are going to use is Polygon File Format (PLY). This file can contain information about vertices and their attribute such as their locations, colors, or texture coordinate. The file can be in ASCII format or binary format. For this project, we will only focus on PLY in binary format that contains either texture coordinate or colors attributes.

The header of a binary ply format will be ASCII character. It starts with the magic number `ply` in one line. For this project, a ply file will contain vertex locations with either texture coordinates or colors. The following is the header of a given file named `bus_ply.ply` which contains texture coordinates:

```
ply
format binary_little_endian 1.0
comment File exported by Artec Group 3D Scanning Solutions
comment www.artec-group.com
element vertex 343049
property float x
property float y
property float z
```

```
element face 686393
property list uchar int vertex_indices
element multi_texture_vertex 374855
property uchar tx
property float u
property float v
element multi_texture_face 686393
property uchar tx
property uint tn
property list uchar int texture_vertex_indices
end_header
```

The above header indicates that it is a binary file using the little endian format. Any lines start with the keyword `comment` are comments.

The `element` keyword introduces a description of how some particular data element is stored and how many of them there are. The first `element` of our file is as follows:

```
element vertex 343049
property float x
property float y
property float z
```

The above `element` says that there are 343049 vertices. Each vertex consists of three `float` values for `x`, `y`, and `z`. Note that these are just a list of locations. They are not the actual sequence of vertices to be used to draw yet. So, just store them in another array of `vec4` such as `vertex_refs` or something.

The next `element` of our file is as follows:

```
element face 686393
property list uchar int vertex_indices
```

This `element` indicates that there are 686393 faces. Each face is associate with a list. The `list` indicates that the data is a list of values. In this case, the first byte (`unsigned char`) indicates the number of elements in the list and each element is four bytes (`int`). Each `int` element is the index to the array of vertices from the previous `element vertex`. In general, each face does not have to be a triangle. Luckily, every face in the provided files is a triangle.

To help understand how `element vertex` and `element face` work, consider the following header:

```
ply
format binary_little_endian 1.0
element vertex 4
property float x
property float y
property float z
element face 2
property list uchar int vertex_indeces
end_header
```

The binary data right after `end_header\n` where the first byte right after `\n` is offset 0 is as follows:

Offset(s) in Decimal	Data
0 - 3	0.5
4 - 7	0.5
8 - 11	0.0
12 - 15	-0.5
16 - 19	0.5
20 - 23	0.0
24 - 27	-0.5
28 - 31	-0.5
32 - 35	0.0
36 - 39	0.5
40 - 43	-0.5
44 - 47	0.0

The above data tell you that the locations of those four vertices starting at index 0 are (0.5, 0.5, 0.0), (-0.5, 0.5, 0.0), (-0.5, -0.5, 0.0), and (0.5, -0.5, 0.0). Again, these are not sequence of vertices to draw. The data starting at offset 48 are as follows:

Offset(s) in Decimal	Data
48	3
49 - 52	0
53 - 56	1
57 - 60	2
61	3
62 - 65	0
66 - 69	2
70 - 73	3

The above data tell you that the first face consists of three vertices, indices 0, 1, and 2 (in that order). The second face consists of three vertices, indices 0, 2, and 3 (in that order). In other words, we should end up with an array of vertices of size 6 with the following data:

- `vertices[0]` is (0.5, 0.5, 0.0, 1.0)
- `vertices[1]` is (-0.5, 0.5, 0.0, 1.0)
- `vertices[2]` is (-0.5, -0.5, 0.0, 1.0)
- `vertices[3]` is (0.5, 0.5, 0.0, 1.0)
- `vertices[4]` is (-0.5, -0.5, 0.0, 1.0)
- `vertices[5]` is (0.5, -0.5, 0.0, 1.0)

Note that you have to hard code that the fourth element of each vertex is 1 (homogeneous coordinate).

The next part of the header is as follows:

```
element multi_texture_vertex 374855
property uchar tx
property float u
property float v
element multi_texture_face 686393
property uchar tx
property uint tn
property list uchar int texture_vertex_indices
```

It says that there are 374855 texture coordinate each of which consists of there items:

- one byte (`uchar`) indicating the texture number (should always be 0 in our case)
- four bytes (`float`) `u` indicating the s coordinate
- four bytes (`float`) `v` indicating the t coordinate

Again, these are just (s, t) reference coordinates. They are not coordinates for vertices yet. The next element says that there are 686393 faces of texture coordinate each of which consists of three items:

- one byte (`uchar`) indicating the texture number (should always be 0 in our case)
- four byte (`uint`) unused in our case
- a list of indices similar to `element face` discussed earlier.

The following is an example of a header of a PLY file with colors:

```
ply
format binary_little_endian 1.0
comment File exported by Artec Group 3D Scanning Solutions
comment www.artec-group.com
element vertex 749754
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
element face 1499999
property list uchar int vertex_indices
end_header
```

As you may guest, there are 749754 vertex references each of which consists of a 12-byte vertex location and 3-byte vertex color. The element face are pretty much the same.

Since your program needs to use either texture coordinates or colors, your fragment shader needs to know what to do. Here is an example of a fragment shader:

```
#version 120

varying vec4 color;
varying vec2 texCoord;

uniform sampler2D texture;
uniform int use_color;

void main()
{
        if(use_color == 1)
                gl_FragColor = color;
        else
                gl_FragColor = texture2D(texture, texCoord);
}
```

You need to set the value of `use_color` before the first draw. Use the following code snippet in the `init()` function assuming that the variable `has_colors` is 1 if the PLY file contain color information. Otherwise, it will be 0.

```
    :
    if(has_colors)
        glUniform1i(glGetUniformLocation(program, "use_color"), 1);
    else
        glUniform1i(glGetUniformLocation(program, "use_color"), 0);
    :
```

For this part, your program should ask a user to enter a file name (just name without extension). If the PLY file contains texture coordinate, there will be two files, a `ply` file and a data file (texture). For example, `chair.ply` and `chair.data` or `full_body.ply` and `full_body.data`. If the PLY file contains colors, there will be only one files such as `nike.ply`. So, your program should open the PLY file and see whether it contains texture coordinates or colors and perform an appropriate action. Again, use the transformation engine from Project 1 to display the result.

The rubric for this part is as follows:

- 40 points if you can correctly display PLY file with texture
- 30 points if you can correctly display PLY file with colors

Submission

The due date of this project is stated on the Canvas. Late submissions will not be accepted. Zip all files related to this project into the file named `project2.zip` and submit it to the Canvas. Do not forget that you must demo this project to your TA on the due date during your recitation.