

CSE 167 Assignment 1: Rendering Point Clouds

- DUE Feb. 2nd Fri 11:59 a.m.

In this project the goal is to read a number of 3D point positions from a file and render them onto the screen.

Besides becoming familiar with the linear algebra involved in rendering 3D scenes, this project will get you familiar with the tools and libraries that will be your friends throughout the quarter. These will include **GLFW** and **GLEW**, as well as **GLM**.

1. Getting Your Development Environment Ready

You should start this assignment by modifying the starter code from Assignment 0. This part should be done in assignment 0 already. If not, please catch up.

2. Reading 3D Points from Files (10 pts)

A [point cloud](#) is a 3D collection of points, representing a 3D object. These point clouds are often acquired by laser scanning, but can also be acquired with the Microsoft Kinect and special software, or by processing a large number of photographs of an object and using Structure from Motion techniques (see [Microsoft Photosynth](#) or [Autodesk 123D Catc](#)).

In this project we're going to render the points defined in OBJ files. Note that OBJ files are normally used to define polygonal models, but for now we're ignoring that and use the vertex definitions to render points, ignoring all connectivity data. OBJ files are 3D model files which store the shape of an object with a number of vertices, associated vertex normals, and connectivity information to form triangles. [Wikipedia](#) has excellent information on the OBJ file format. The file format is an ASCII text format, which means that the files can be viewed and edited with a any text editor, such as Notepad.

Write a parser for the vertices and normals defined in OBJ files. It should be a simple 'for' loop in which you read each line from the file, for instance with

the `fscanf` command. Your parser does not need to do any error handling - you can assume that the files do not contain errors. Add the parser to the starter code.

Use your parser to load the vertices from the following OBJ files and treat them as point clouds:

- [Bunny](#)
- [Dragon](#)

The files provided in this project only use the following three data types (other OBJ files may support more): `v` for vertex, `vn` for vertex normal, `f` for face.

The general format for vertex lines is:

```
v v_x v_y v_z r g b
```

Where `v_x`, `v_y`, `v_z` are the vertex `x`, `y`, and `z` coordinates and are strictly floats.

The values `r`, `g`, `b` define the color of the vertex, and are optional (i.e. they will be missing from most files). Like the vertex coordinates, they are strictly floats, and can only take on values between 0.0 and 1.0.

All values are delimited by a single whitespace character.

The general format for normal is the same as for vertices, minus the color info.

In summary:

- `v`: 'vertex': followed by six floating point numbers. The first three are for the vertex position (`x`, `y`, `z` coordinate), the next three are for the vertex color (red, green, blue) ranging from 0 to 1. Example:

```
v 0.145852 0.104651 0.517576 0.2 0.8 0.4
```

- `vn`: 'vertex normal': three floating point numbers, separated by white space. The numbers are for a vector which is used as the normal for a triangle. Example:

```
vn -0.380694 3.839313 4.956321
```

Lines starting with a `#` sign are comments and should be ignored.

In this homework assignment, you only need to parse for vertices and vertex normals, which are those lines of the file starting with a `v` and `vn`.

Write your parser so that it goes through the OBJ file, line by line. It should read in the first character of each line and based on it decide how to proceed, i.e., ignore all lines which do not start with a 'v' or 'vn'. The vertex definitions can be read with the `fscanf` command. Here is an example:

```
1 FILE* fp; // file pointer
2 float x,y,z; // vertex coordinates
3 float r,g,b; // vertex color
4 int c1,c2; // characters read from file
5
6 fp = fopen("bunny.obj","rb"); // make the file name configurable so you can load other files
7 if (fp==NULL) { cerr << "error loading file" << endl; exit(-1); } // just in case the file can't be found or is corrupt
8
9 c1 = fgetc(fp);
10 c2 = fgetc(fp);
11 if (c1=='v') && (c2==' ')
12 {
13     fscanf(fp, "%f %f %f %f %f %f", &x, &y, &z, &r, &g, &b);
14 }
15
16 // read normal data accordingly
17
18 fclose(fp); // make sure you don't forget to close the file when done
```

Load all two models in at startup.

3. Rendering the Points with OpenGL (20 pts)

To display the vertices you loaded, use the provided starter code. It contains hooks for rendering points with OpenGL, which are ready for you to use. Change the



Display callback and replace the call to render the cube to render the OBJ object - don't call `cube.draw()` but call the respective OBJ object's draw command.

Please render two meshes on the same screen. (As shown above)

Just like the cube spins during display, make your OBJ models spin as well. **THE BUNNY AND DRAGON SHOULD ROTATE IN DIFFERENT DIRECTION (i.e. one along y-axis, the other along z-axis)**

The points should be colored with normal coloring: use the vertex normals you parsed from the file, and map them into the range of zero to one. The x coordinate of the normal should map to red, y to green, z to blue.

Use the `glPointSize()` command to allow the user to adjust the size of the points with the 'p' and 'P' keys (for smaller and larger points, respectively).

4. Manipulating the Points(20 pts)

Once a model has been loaded, support the following keyboard commands to manipulate it:

- 'p'/P": scale down/up the size of each point as mentioned
- 'a'/d": move left/right (along the x axis) by a small amount
- 'w'/s": move down/up (along the y axis) by a small amount
- 'z'/Z": move into/out of the screen (along the z axis) by a small amount
- 'c'/C": scale down/up (about **the model's** center, not the center of the screen)
- 'r': reset position (move objects back to center of screen, without changing orientation or scale factor)
- 'R': reset orientation and scale factor, but leave the objects where they are

Tweak the values for changes in position and size so that you can manually center each OBJ model in the window and scale it to the size of the window during grading. All translations must be in world space, not objects space.

Note: keep the objects spinning at all times during manipulation (Note this rotation is in object's local space). Also, all translations should be relative to the viewer, not

to the objects' local space. In other words, if an object has spun 180 degrees, then pressing the key to move the objects left should still make the projected image of it be shifted to the left, not to the right. Also note that the operations have to be cumulative, which means that every time you move the objects they move from where it was moved to before.

5. Rendering the Points through Rasterization (50 pts)

In this part of the assignment you need to **write code to display the point cloud using your own rasterization code**. This is important: in this part of the project we're going to render into a block of memory. We provide **starter code** (This is just a reference, you should understand the provided code and merge them into yours) with the functionality to make this memory show up on the screen as pixels.

This is going to require that you evaluate the complete vertex transformation: $p' = D * P * C^{-1} * M * p$

As part of the starter code comes a file called rasterizer.cpp which contains code to render a 2D RGB array into your window. Use this code and modify it to render your point models. The models have to be rendered in the same place as when you render them with OpenGL. **Add support for the 'm' key to switch between the two rendering modes.**

Use normal coloring again for the points, just like in OpenGL mode. Also, just like in OpenGL mode, support the 'p'/'P' keys to change the point size. For bigger points, draw squares of integer widths, such as 2x2, 3x3, 4x4,...,nxn.

YOU ONLY NEED TO RENDER ONE OF THE OBJs IN YOUR RASTERIZER.

EXAMPLE



6. Submission

Upload your code in zip file of your project on Gradescope by Feb. 2 11:59 a.m.

ZIP file should include all the h/cpp files. **DO NOT ZIP THE OBJ FILE!**

Grading session: Feb 2nd 1:00 p.m. — 3:00 p.m.