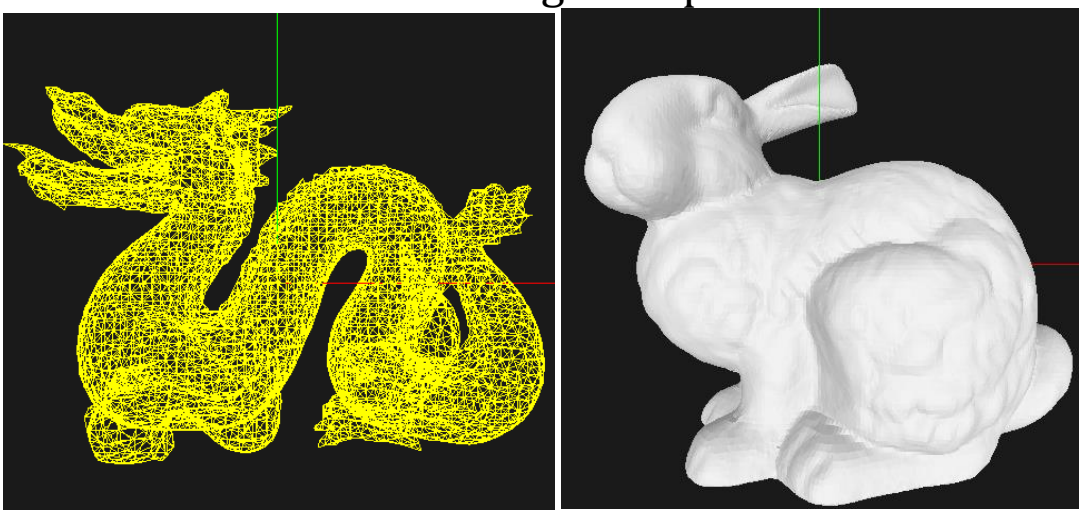
Lab 1: Loading a Shape File



Description:

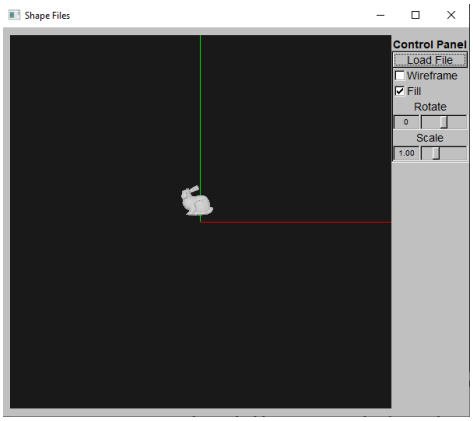
You've learned that we can draw polygons to the screen by plotting them out one at a time, or even using mathematical functions to draw nice curves and geometric patterns. Often times to create more complex shapes, 3D objects will be scanned from the real world by lasers or modeled in 3D software. This data is then output in a 3D file format that can be read.

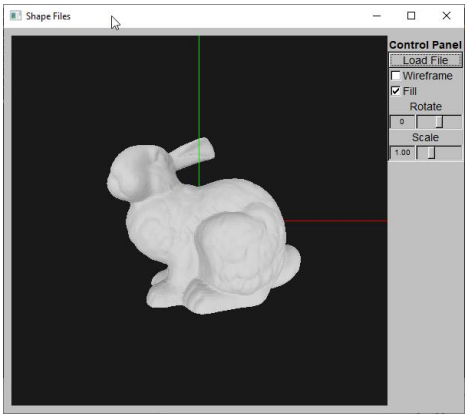
Your Tasks:

- You will be loading a file known as the PLY format. This format consists of a, header, a vertex list, and a face list.
- Start by looking at the headers given to familiarize yourself with the functions provided. We have already provided the parser[1]. You can improve upon it in your own time, but you shouldn't have to edit the parsing component of the support code.
- Instead, your task is to fill out the two functions related to rendering:
  - scaleAndCenter() – In ply.cpp
  - render() – In ply.cpp
- In ply::render(), you need to add code to: (1) render each triangle of the model, and (2) set the normal of the triangle (before rendering it)
- In ply::scaleAndCenter(), the objective is to take the geometry of your 3D model and make sure that the model: (1) is centered at (0, 0, 0), and (2) is bounded by a 1x1x1 cube (i.e., values of all the vertices in the three dimensions (x, y, and z) are between -0.5 and 0.5).
  - Note that for this function, you should modify the position of each vertex (instead of using an OpenGL function).

[1] The provide parser is very simple and will NOT work on all PLY files. The parser is limited to: (1) ascii versions of PLY, (2) PLY files that use triangles only (e.g., no Quads), and (3) models without texture

Hints:





- We suggest that you start with the ply::render() function. If you write your render process correctly, you can load the "bunny.ply" file and see results already. You should get something like the top right image.
    - Note that you can use the "scale" button to zoom in, but the point is that the bunny is: (1) not centered at the origin, and (2) not automatically sized to be between -0.5 to 0.5
- Once you have implemented ply::scaleAndCenter(), you should see the bottom right image.
- For how to approach implementing render(), see the lecture slides on how to draw triangles with OpenGL.
- For scaleAndCenter(), there are two tasks:
    - First, find the "center" of the object: for this assignment, we consider the center to be the average position of all vertices (this is the equivalent to finding the moment of inertia of the object).
    - Once the object has been centered, the second task is to find the "scaling" factor of the object: think about this a little bit. Our goal is to make the object smaller or bigger such that no vertex has an x, y, or z value that's outside of the range of [-0.5, 0.5]. How do you do find that scaling factor?

Files Given:

main.cpp – Main program interface (you do not have to modify this) ply.h – The shape header file with data (start here) ply.cpp – The shape class that loads the PLY model geometry.h – Some data structures to store geometry information (you do not have to modify this). Data folder – These are the files you will be loading. Take a moment to load each in a text editor and familiarize yourself with this text-based 3D file format.

Example PLY File [1]:

```
ply
format ascii 1.0          { ascii/binary, format version number }
comment made by Greg Turk  { comments keyword specified, like all lines }
comment this file is a cube
element vertex 8          { there are 8 vertices in this 3d object }
property float x          { vertex contains float "x" coordinate }
property float y          { y coordinate is also a vertex property }
property float z          { z coordinate, too }
element face 6            { there are 6 faces in the file }
property list uchar int vertex_index { "vertex_indices" is a list of ints }
end_header                { delimits the end of the header }
0 0 0                  { start of vertex list }
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
3 0 1 2        {start of faces. First digit (3) means there are 3 vertices (i.e. a triangle)}
3 7 6 5        {the remaining integers are indexes to the list of vertices above}
3 0 4 5
3 1 5 6
3 2 6 7
3 3 7 4
```

0 1 2 are indices for the list

Going Further:
Did you enjoy this in class assignment? Take a look at some other 3D formats such as .md2 (includes basic animation data). Look at some popular 3D modeling software like Maya, Blender3d (download for free!), and check out their file formats. Think about other functions you can add to the ply.cpp. What interesting manipulations can you do with the file?

Some ideas:

- Highlight faces that have an area above a certain threshold
- Render surface normals from each face of the object (you will do this in A1)
- Color the triangles based on how parallel they are to the viewing plane
- Create a noise function that adjusts the positions of vertices on the mesh.
- Add color to vertices above a certain height.
- Modify the rendering function to use a Display List (relatively simple) or vertex buffer objects (advanced topic)

C++ Refresh -- Helper functions:
(for understanding the parser)

Opening a File

```
1.  #include <fstream>
2.  ifstream myfile ("filepath");
3.  if (myfile.is_open()){
4.      string line;
        while(getline(myfile,line)){
5.          // some code
6.      }
7.  }
```

Parsing a line

```
8.  #include <stdio.h>
9.  char* delimeter_pointer;
10. delimeter_pointer = strok(line," ")
11. while(delimeter_pointer != NULL){
12.     cout << delimeter_pointer;
13.     delimeter_pointer =
    strtok(NULL," ");
14. }
```

References:

[1] http://paulbourke.net/dataformats/ply/