CSC 471: Program I - Software rasterization of triangle meshes

This is an individual project

Due: Sunday, January 19th 11:55pm

Goal: Create a program, which reads-in and renders an indexed face set meshes (of type .obj) to an image via software rasterization. You may use existing resources to load in the mesh and to write out an image. You must write your own rasterizer. In general the required steps for the program are:

- Read in triangles
- Compute colors per vertex
- Convert triangles to window coordinates
- Rasterize each triangle using barycentric coordinates for linear interpolations and in-triangle test
- Write interpolated color values per pixel using a z-buffer test to resolve depth

Each task is described in more detail here:

1. Download the base code, which has a mesh loader and an image writer from polylearn.

   For your reference, the mesh loader is an obj loader from (https://github.com/syoyo/tinyobjloader) and the image writer is the same as the one used in prior labs.

   Example mesh files are included in the base code and you can create your own to represent a single triangle, etc. In addition, there are numerous OBJ meshes on the web. For grading purposes, your program will be run using the Stanford bunny and Utah Teapot.

   Ultimately you will want each triangle to be represented in a C/C++ structure or class, with 3 vertices and a color per vertex. In addition, your triangle data should include a 2D bounding box, which will represent the triangle's extents in window coordinates.

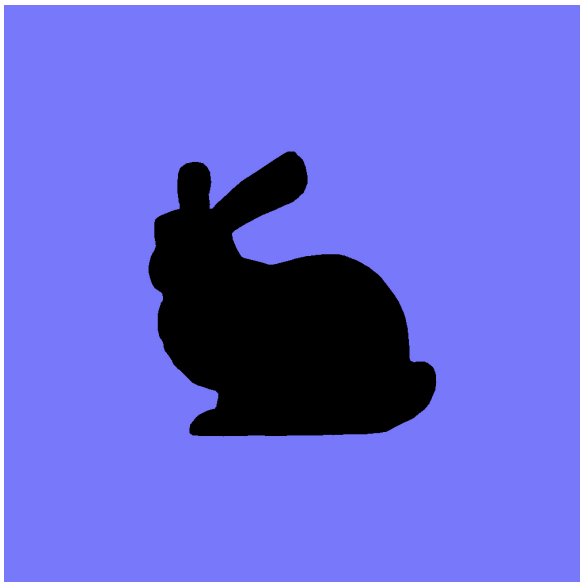
2. Add a command line argument to accept the following command line arguments.
   -Input filename of the .obj file to rasterize
   -Output image filename
   -Image width
   -Image height
   -Coloring mode (see Task 5)
   For example, your program should be able to be run as follows:

   ```
   ./raster ../resources/bunny.obj out.png 512 512 1
   ```

   Add error checking to specify the required command line arguments if an

incorrect number are given – and include an example correct command line run. Your program should not dump core if no input file is specified, nor fail without an error message! Follow the golden rule, treat your user/grader/instructor the way you'd like to be treated as a user/grader/instructor.

3. Write code to convert each 3D coordinate into 2D window coordinates. Assume the camera is at the origin looking down negative z. Choose a reasonable window size (for example 400x400 or 600x800). Consider testing first with a solid color, which might look something like this:



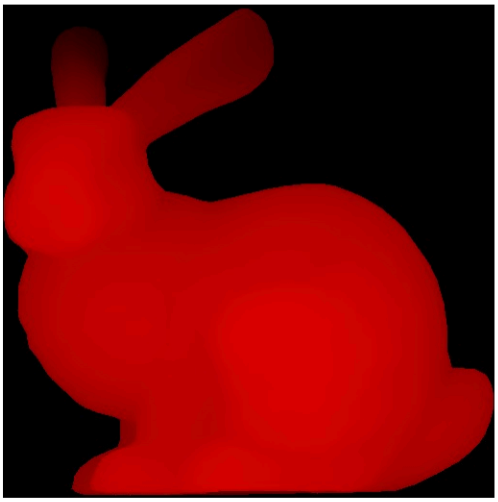
Make sure you test non-uniform window sizes (for example see the teapot below where the aspect ratio of the teapot is preserved).

4. Create a data structure to support z-buffer tests. (Your z-buffer should be the same size as your window/image). When converting coordinates (step 3), leave z in world coordinates for z-buffer testing.

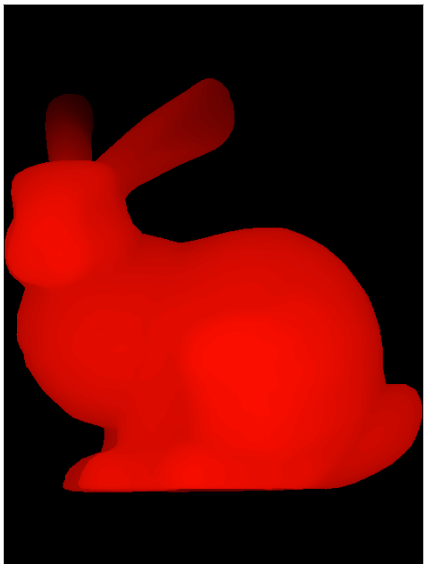
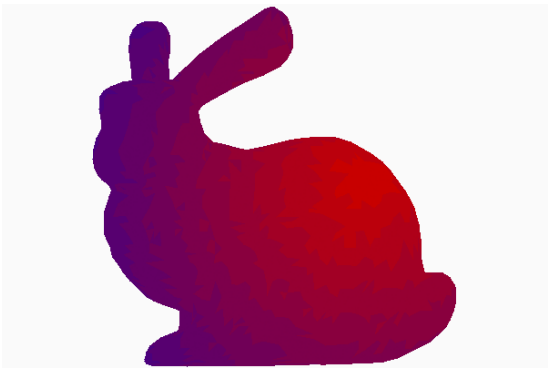
Incorporate your lab code to rasterize a triangle with a color defined per vertex. Any point within the triangle should be drawn with colors interpolated via the barycentric coordinates. Likewise, depth should also be interpolated using the

barycentric coordinates and written to the z-buffer.

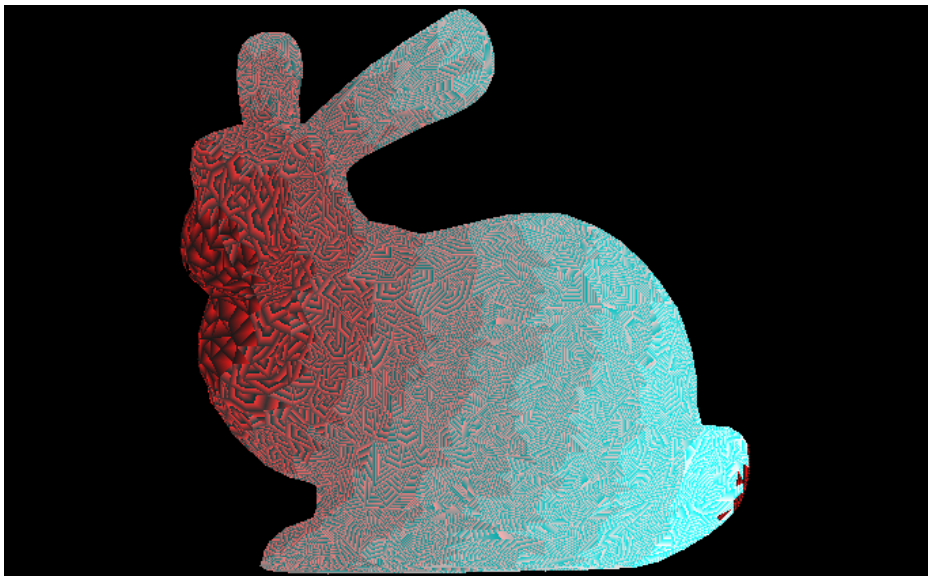
Use the z value of the vertices as the color. (You can choose any color, not just red.) To do this, you have to map the z-value to the range 0 to 255. If your z-buffer test is not working, you'll see some strange results, since some pixels that are farther from the camera may be drawn on top of closer pixels.

	
No depth testing	With the z-buffer!

5. Next, compute colors per vertex using two different coloring algorithms, that can be chosen on the command line via selecting (1) or (2):

(1) color the vertices based on its depth (ie closer vertices are brighter and vertices that are further away are darker) - you may choose to scale a particular r, g, or b by the depth, for example a red depth bunny would look as below:	(2) ****To be announced some kind of specialized color drawing mode**.** For example last year we did coloring the vertices based on binned distance to a point in window space - A binned distance, will create a banded look as shown below. Be sure your color is blending between two different colors based on the distance. ****I will announce this years special coloring scheme by week 2****
	

Important Note: Make sure to pass your std::vector by reference rather than by value. (e.g., `void foo(std::vector<float> &bar)`) Otherwise, your program may become too slow.



And a note about graphics “bugs” - cut and paste errors are SO easy to introduce to your code, with three coordinates and three colors, etc. Be sure to re-read your code. The up side in “bugs” sometimes look cool. (See image)

Point break-down:

20 points for window coordinate transforms for square images.
10 points for window coordinate transforms for non-square images.
20 points for correct rasterization of triangles.
15 points for correct z-buffer implementation.
10 points for mode 1 color interpolation.
10 points for mode 2 color interpolation.
15 points for coding style and general execution. For example, do not put everything in main() .
Total: 100 points

What to hand in:

Failing to follow these points may decrease your "general execution" score. Make sure that your code compiles and runs by typing, for example:

```
>mkdir build
> cd build
> cmake ..
> make
> ./raster  <ARGUMENTS>
```

Make sure the arguments are exactly as specified. Include a README file that includes:

- Your name
- Information about the distance coloring point
- Citations for any downloaded code (e.g., barycentric)
- Plus anything else of note

Make sure you don't get any compiler warnings. Remove unnecessary debug printouts. Remove unnecessary debug code that has been commented out.

Hand in src/ , CMakeLists.txt , and your readme file. Do not hand in the build directory, the executable, input obj files, output image files, old save files (*.~) ,or object files (*.o).

Create a single zip file of all the required files. The filename of this zip file should be CALPOLY_USERNAME.zip (e.g., zwood.zip). The zip file should extract everything into a folder named CALPOLY_USERNAME (e.g. zwood).