Menger Sponge

Grant Martin (glm2367)

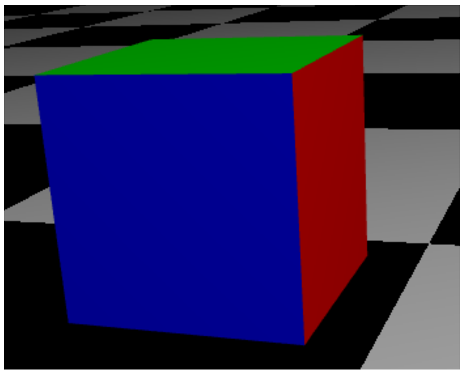
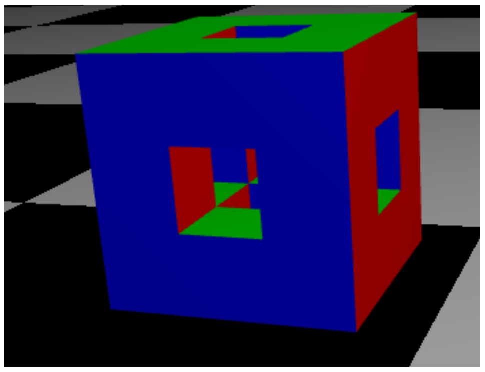
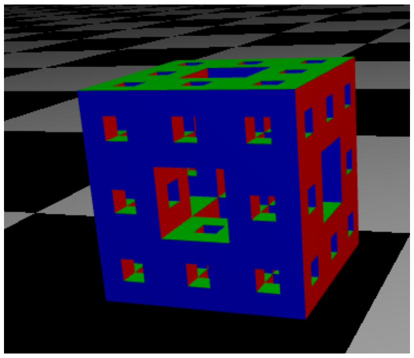
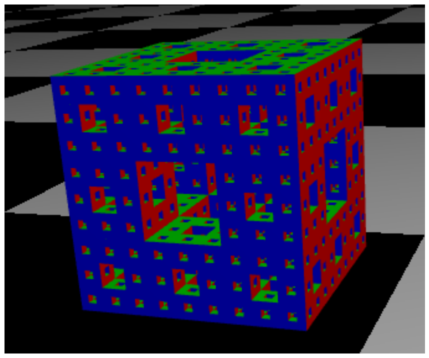
Menger Sponge

- Positions:
  For the positions, I'm using a recursive method that splits up the cube into 27 sub cubes, but it ignores the inner 9 cubes. It keeps recursing until its level is 1, where it then makes the sub cube's positions. The positions are always calculated the same way and are made in drawing order that way implementing indices is easier.

- Indices:
  Indices are made after all positions are made. It's just an array from 0 to positions length-1.
  This works because the order that I constructed the positions array is consistent with the order that it is drawn (vertices of drawn triangles are grouped together).

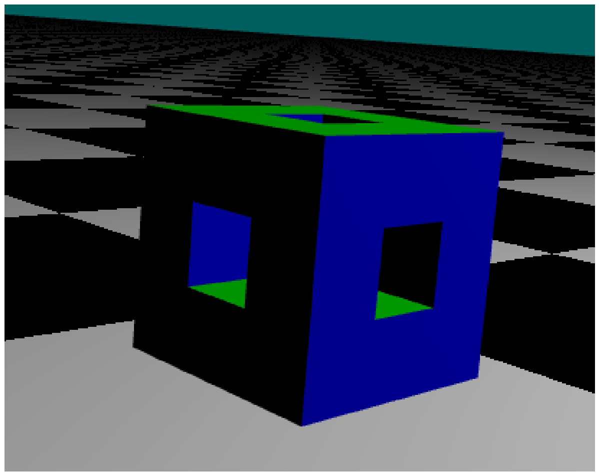
- Normals:
  The normals are made after all positions are made. For every triangle, it calculates the cross product between two edges and normalizes it.

Results:
Levels 1-4:

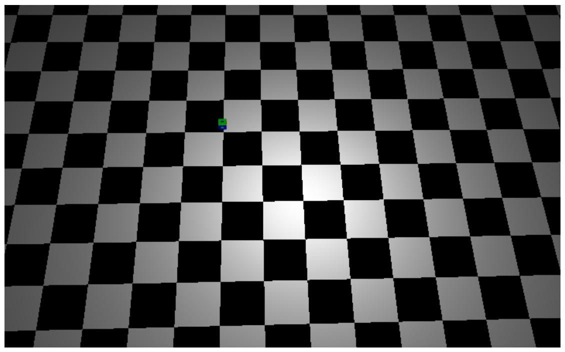





Shaders

There are 4 shaders used in total. I implemented three of them:

- Cube fragment shader:
  In order to add light attenuation, I normalize the absolute value of the normal passed in (because we don't want negatives in our RGB values). Then, we multiply the normals by the dot product of the normal and the light direction, which is related to the angle between the two. This way, the more directly it hits the surface, the higher, and therefore the brighter, the RGB values become.
  
- Floor vertex shader:
  This is copy-paste from the provided cube vertex shader, except I pass in the vertex position to a variable pos to be used in the floor fragment shader.

- Floor fragment shader:
  For the floor coloring, I alternate between white and black squares. Once I choose the color, I apply light attenuation in the same way as the cube fragment shader.



I also made a Floor.ts file in the same format as MergerSponge.ts to represent the floor. I then made the proper initializations of the buffers and attributes, which was nearly a copy and paste from the logic already done in Merger as well.

Camera Controls

All controls were successfully implemented. I set the default mode to fps so my output more closely matches the test suite reference output. Some controls I implemented include:

- M1 dragging:
  M1 dragging was implemented by rotating the camera on the axis perpendicular to both the mouse and camera directions. It rotates around the target if in orbital mode, else just about the axis.

- M2 dragging:
  M2 dragging was implemented by offsetting the distance between the camera and the target.

- Left/Right keys
  The left and right keys were implemented by rolling the camera.

- W and S (For orbital)
  W and S for orbital were implemented the same way as M2 dragging

All other movement keys were implemented using some variation of offsetEye and offset target, both of which I implemented in Camera.ts. OffsetEye offsets the eye of the camera by some scalar of a direction vector passed in. Offset target does the same thing, except for the camera's target. The logic was pretty much given to us on the assignment page, so it wasn't hard. I just got the direction I wanted to offset each per functionality and then offset them in that direction scaled by the relevant GUI setting. The functions implemented this way include the following:

- W (for FPS)
  Offsets eye and target by zoomSpeed in the backwards direction
- A (for FPS)
  Offsets eye by panSpeed in the left direction
- A (for orbital)
  Offsets target by panSpeed in the left direction
- S (for fps)
  Offsets both eye and target by zoom speed in the forward direction
- D (for fps)
  Offsets eye by panpeed in the rightdirection
- D (for orbital)
  Offsets eye by panspeed in the rightdirection
- ArrowUp (for fps)
  Offsets eye by panspeed in the upwards direction
- ArrowUp (for orbital)
  Offsets target by panspeed in the upwards direction
- ArrowDown (for fps)
  Offsets eye by panspeed in the downwards direction

- ArrowDown (for orbital)
  Offsets target by panspeed in the downwards direction

Other Features

The following keypresses were also implemented:

- 1-4
  Creates a menger sponge with the corresponding level
- C
  Toggles fps mode
- R
  Sets fps mode to true and reinitializes the camera.

Final Thoughts

Everything works exactly as expected except for some extremely minor pixel differences. You have to zoom in like 250% and compare the reference image side by side really fast to see it. In the middle of some edges, the coloring is a single-pixel too far one way or the other. This could be due to mathematical precision differences when calculating normals or in the fragment shader somewhere. It’s not necessarily incorrect, just different.

There were a ton of bugs I ran into when developing this assignment, such as a messed-up sponge due to improper ordering of indices or incorrectly calculated normals messing up colors and the shapes themselves. 97% of the development time was spent debugging countless silly errors :(