Computer Graphics

Student names and ids: Andreas Jensenius (s093199) & Jannick (s09)

Title : Final project

Work distribution is equal. We worked together on all exercises, project and report and each contributed 50%.

1 Exercise 1

1.1 Part 1

1.1.1 A

The function `display` is the display function used by openGL, as defined by `glutDisplayFunc(display)`, and is called continuously. It defined which shaders to use, binds the VAO, draws the triangle and handles the swapping of buffers.

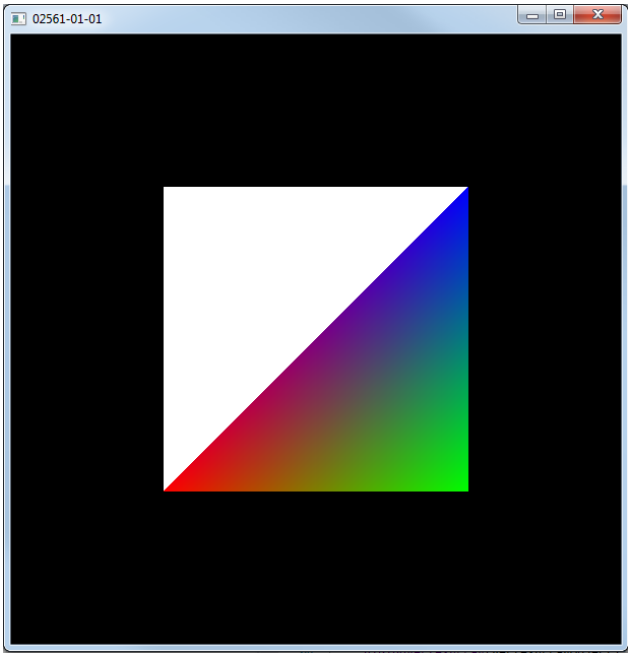
The function `reshape` handles the resizing of the window.

The function `loadBufferData` sets up the buffers and attaches the VAO.

The function `loadShader` sets up the shader and sets up the transfer of data to the shader.

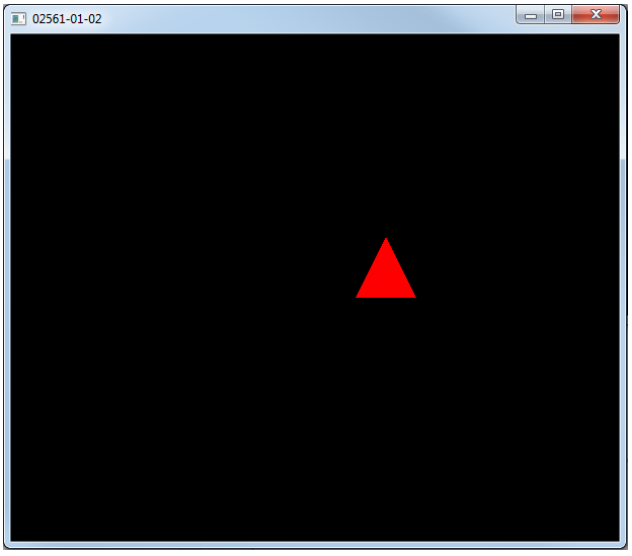
The function `main` is the one which is run when executing the program. It sets up openGL/glut.

1.1.2 B

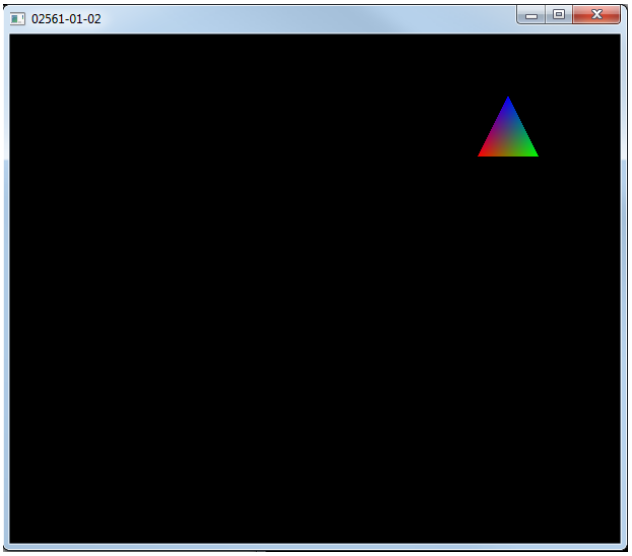


1.2 Part 2

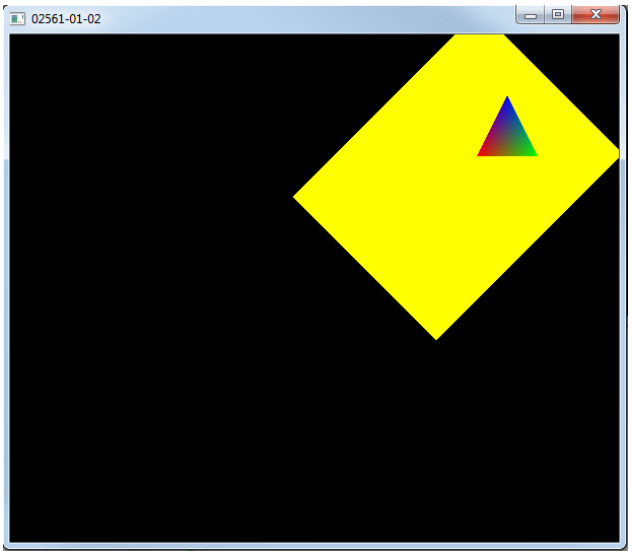
1.2.1 B



1.2.2 C



1.2.3 D



1.3 Part 3

1.3.1 A

glDrawElements uses the given indices within to specify in which order the points should be used, instead of using them from 0 to end as glDrawArrays does.

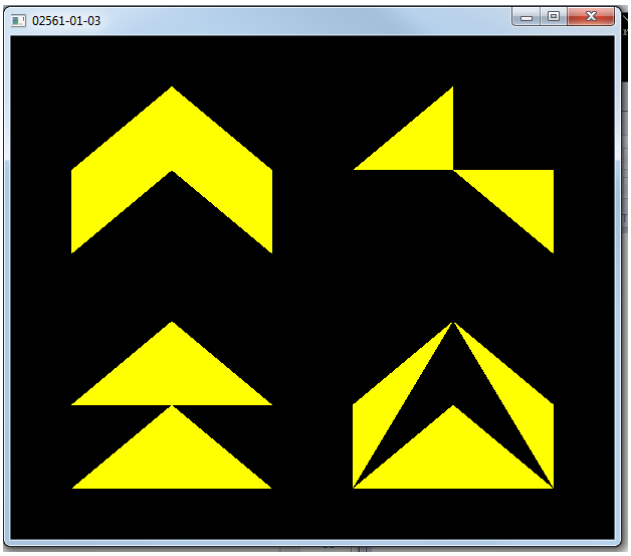
1.3.2 B

$GL_TRIANGLES$ simply forms triangles from three points. Given 6 points it would create 2 triangles.

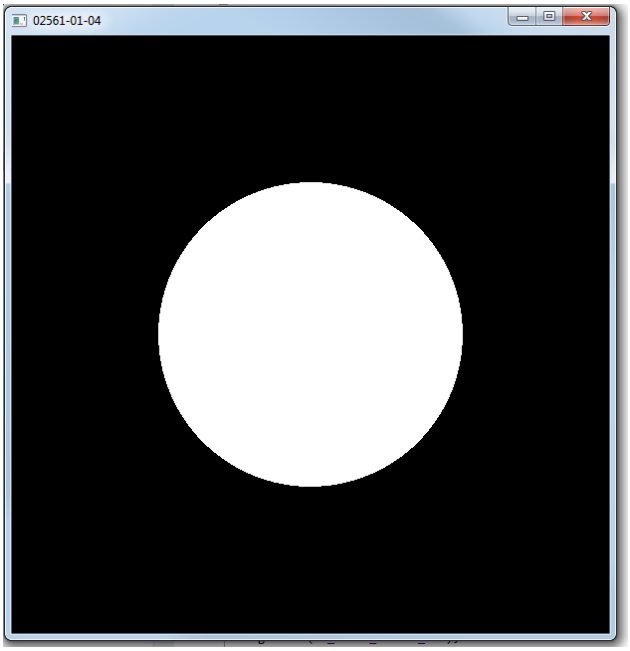
$GL_TRIANGLE_STRIP$ creates connected triangles, in which the two last points of the previous triangle are used as the two first point of the next triangle.

$GL_TRIANGLE_FAN$ uses the first point given for every triangle it draws. Given 3 points it draws a triangle. For each new point given it creates adds a new connected triangle, using the newly added point, the first point given, and the previously added point.

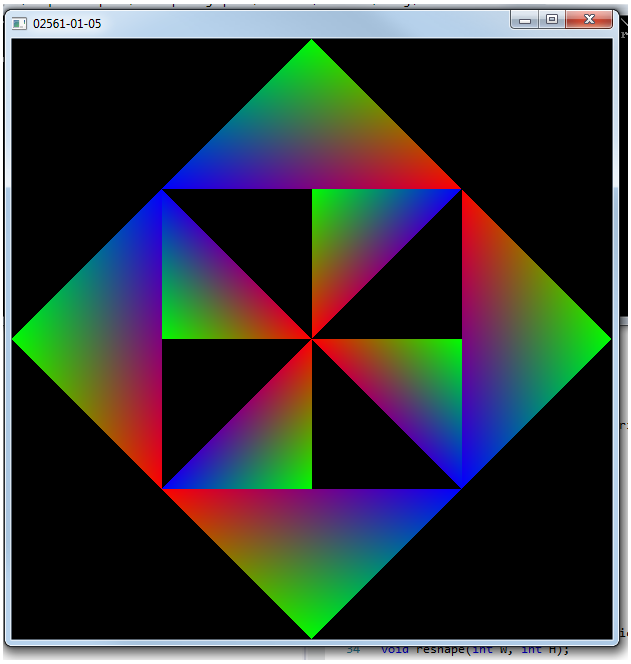
1.3.3 C



1.4 Part 4



1.5 Part 5



2 Exercise 2

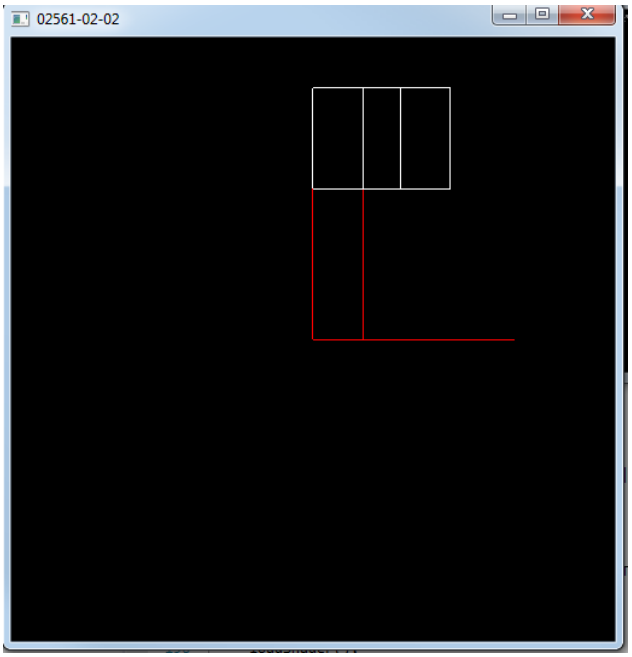
2.1 Part 1

The main difference between the two `display` functions is that this one defines "the camera" by creating the modelview.

The difference in the `vertex` struct is that the old struct handles both color and 2D position, whereas the one in exercise 2 handles position in 3D and not color.

2.2 Part 2

2.2.1 A



2.2.2 B

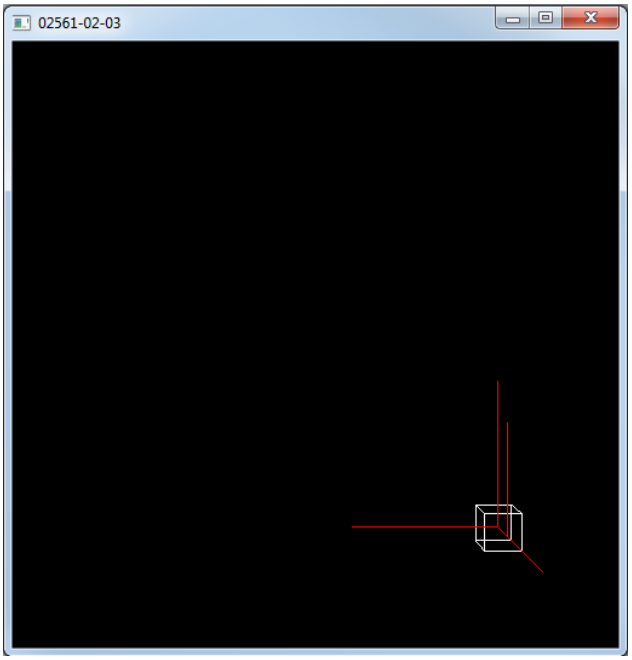
Translate(0,3,0), Scale(2) and RotateY(30) was used.

2.2.3 C

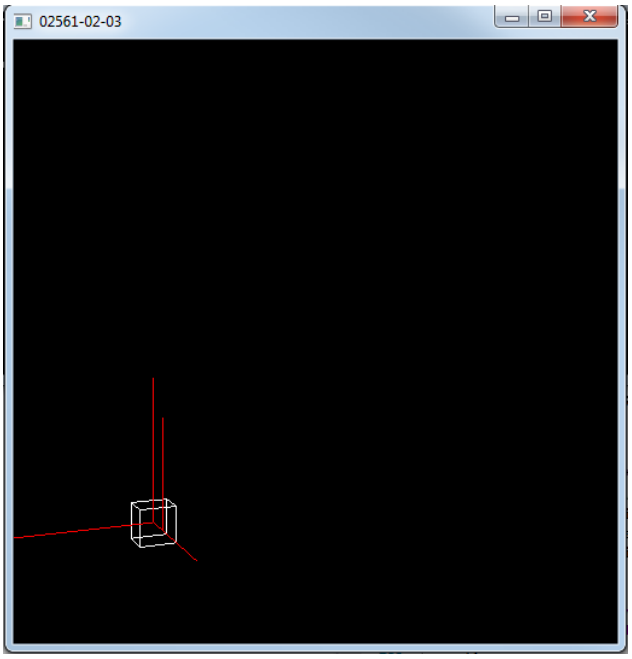
First Translate, then Scale and finally RotateY.

2.3 Part 3

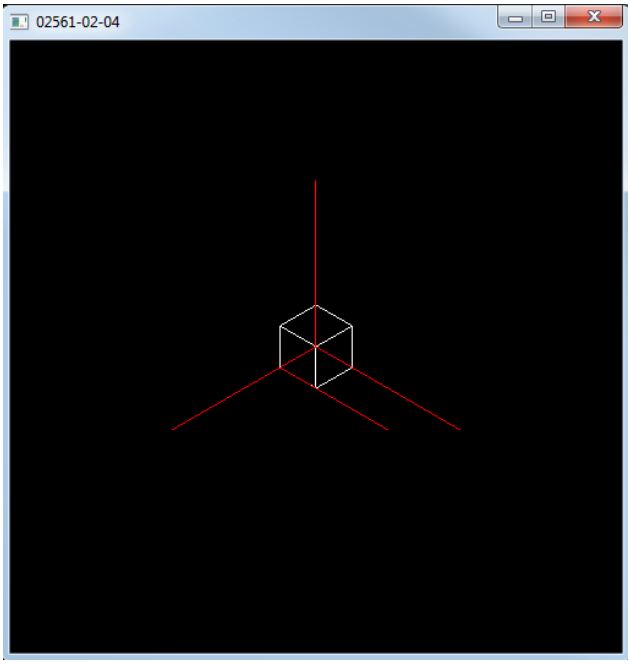
2.3.1 A



2.3.2 B



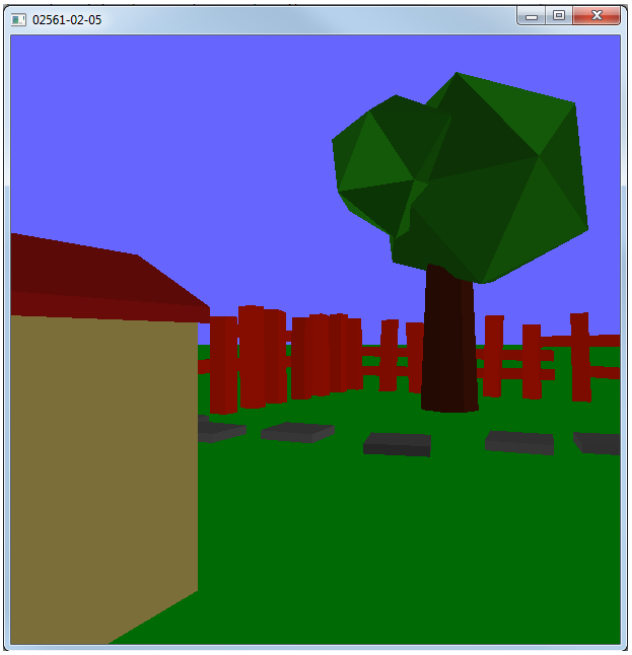
2.4 Part 4



2.5 Part 5

2.5.1 key 2

2.5.2 key 4

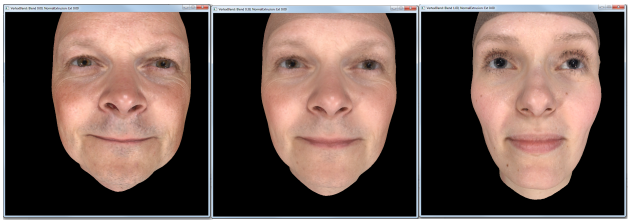


2.5.3 key 6

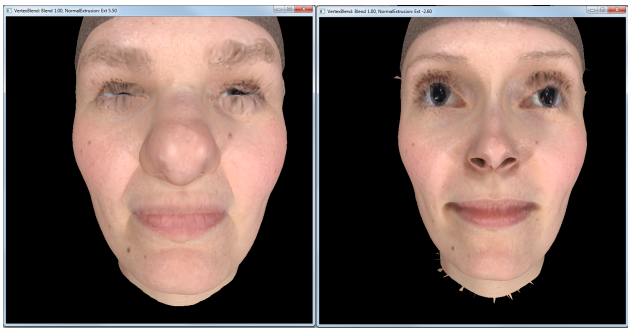
2.6 Part 6

3 Exercise 3

3.1 Part 1



3.2 Part 2

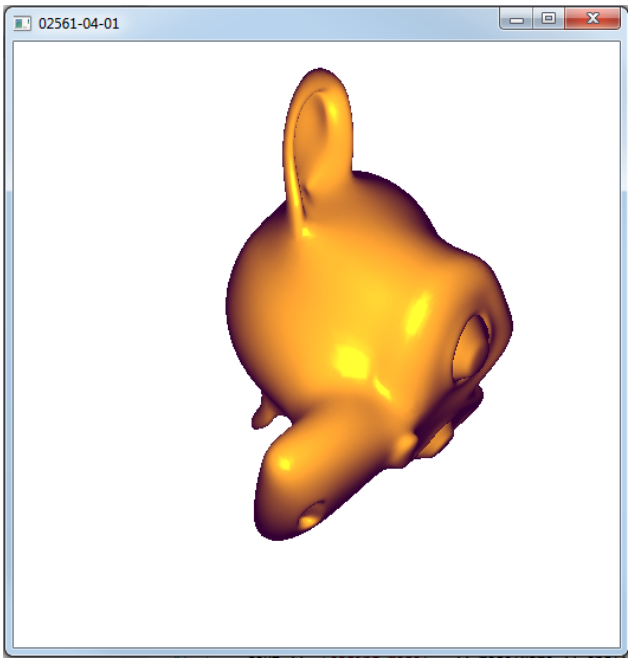


3.3 Part 3

- An attribute is unique for each vertex while an uniform variable is global for all vertexes.
- Vertex shader program is executed on each vertex.
- The fragment shader program is executed on each pixel in each vertex.
- The vertex shader calculates data for the vertex and sends it to the fragment shader.
- Yes and it is a good idea as it makes it less edged.

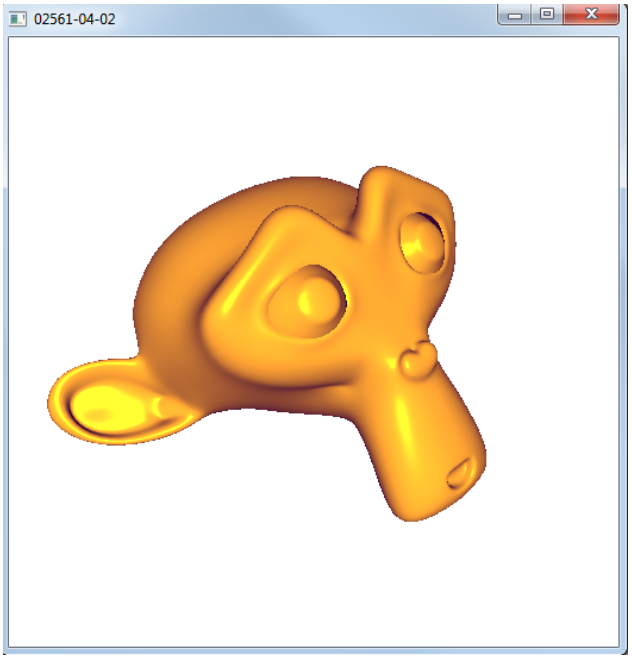
4 Exercise 4

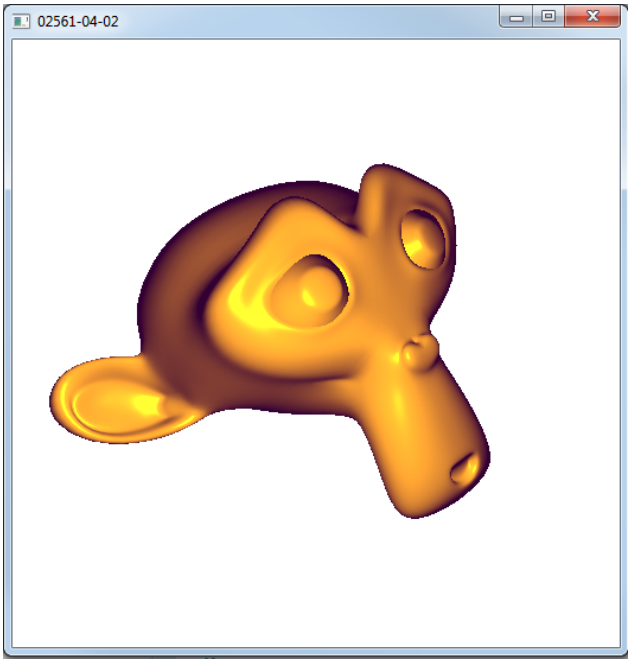
4.1 Part 1



4.2 Part 2

First is point light, second is directional.





4.3 Part 3

4.3.1 A

- Phong lightning uses Ambience, specular and diffuse inorder to give a virtual representation of light given off from a source.
- Phong shading is changing the colors of individual fragments based on the angle and distance from a light source.

4.3.2 B

Gouraud shading is applied on a vertex level, while Phong shading is applied on a per-fragment level. Gourad is cheaper but makes the model look more rough, and phong is more expensive but makes it look more smooth.

4.3.3 C

Using point light, the light originates from a single point with a given direction, while directional lighting does not have a specific origin. In directional light all the light rays are parallel, while using point light the light rays starts from the same origin and spread out.

4.3.4 D

Yes.

4.3.5 E

It simply removes the specular component.

4.3.6 F

It makes it look more shiny, like polished metal, by increasing the intensity of specular lighting.

4.3.7 G

We made no simplifications.

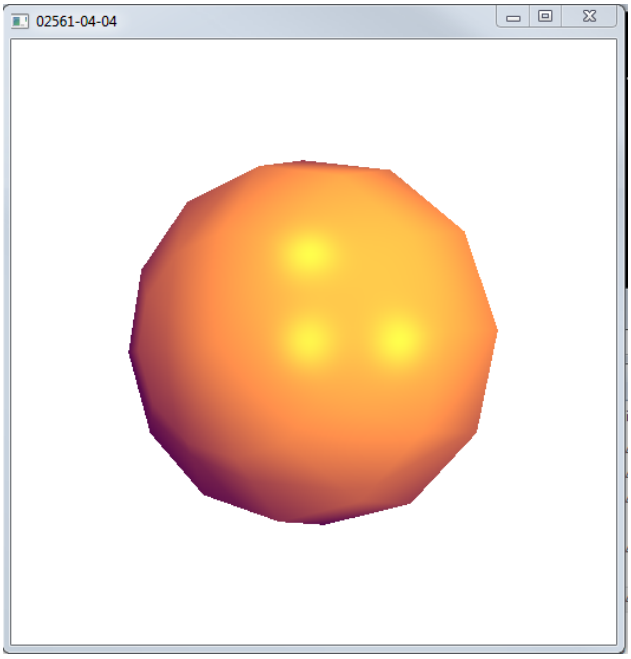
4.3.8 H

It is used to convert to normal vector into eye space. You obtain the normal matrix by taking the transpose, inverse of the modelview matrix.

4.3.9 I

Eye-space.

4.4 Part 4



5 5

5.1 Part 1

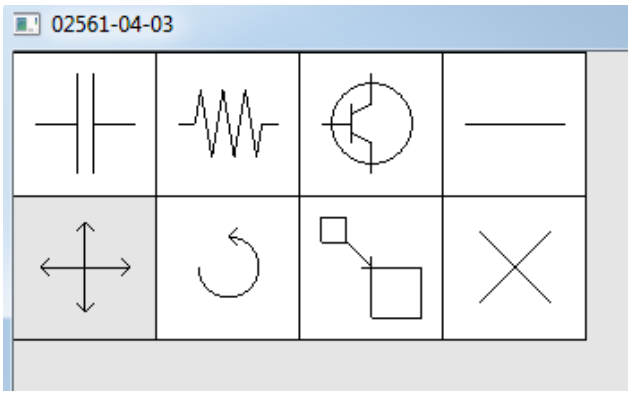
Pick select which object to draw once you click outside the colored boxes.

5.2 Part 2

When the user clicks on a square the uniform variable colorUniform is set and the frameBufferObject i bound. renderScene(true) generates the color and stores, and we then retrieve it in getId decoded from a color. The value of the selected index on the board is update to one of the three possible colors, and we ask to have the display function run again, so that renderScene(false) can draw our changes

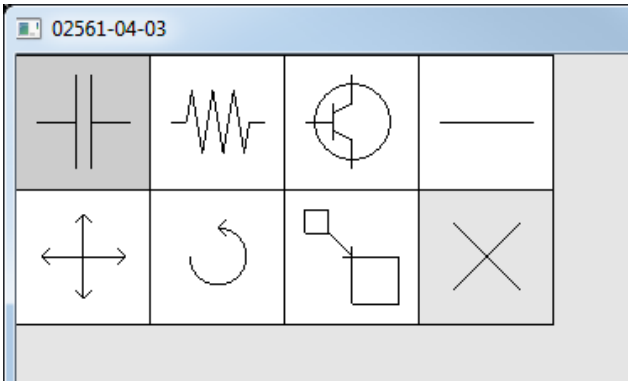
5.3 Part 3

5.3.1 A

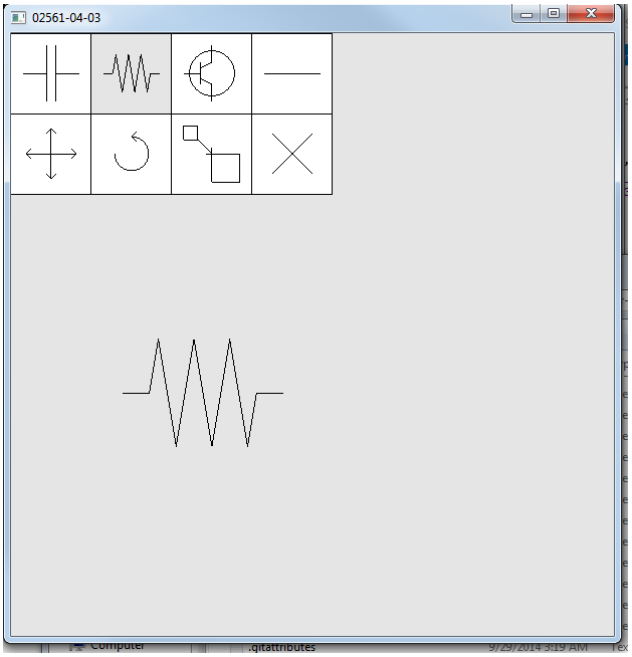


5.3.2 B

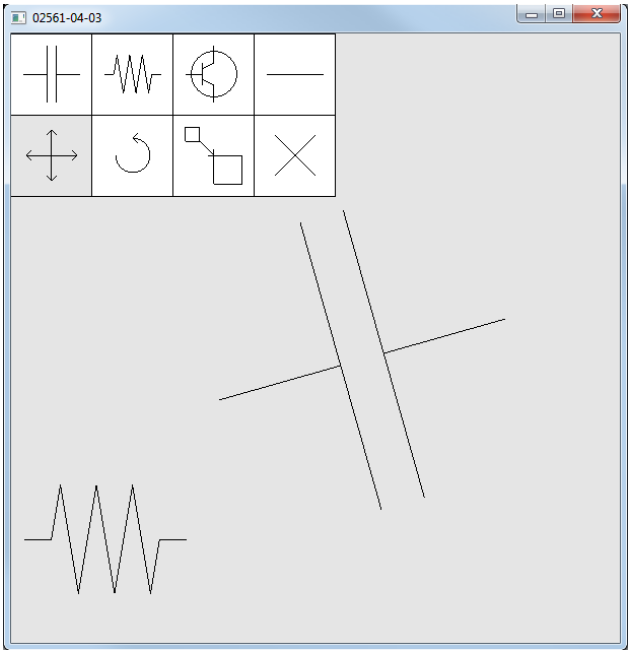
When doing screenshots, the cursor is not captured in the image so it is not shown on the image below.



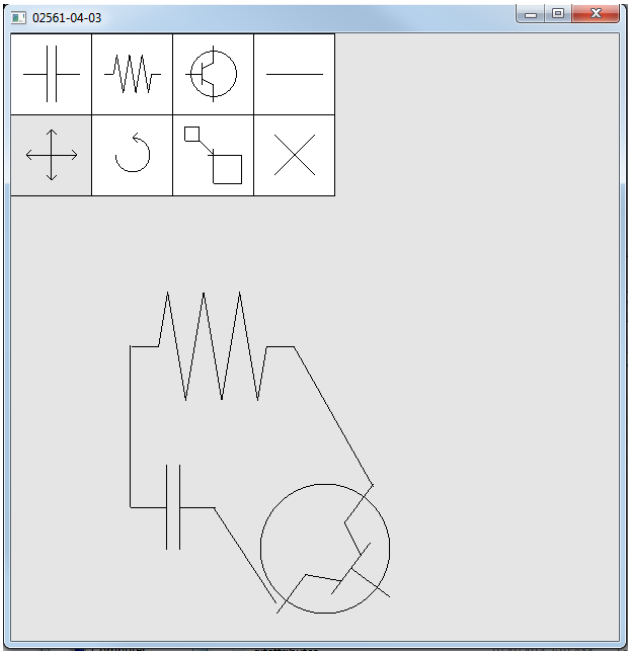
5.3.3 C



5.3.4 D



5.3.5 E

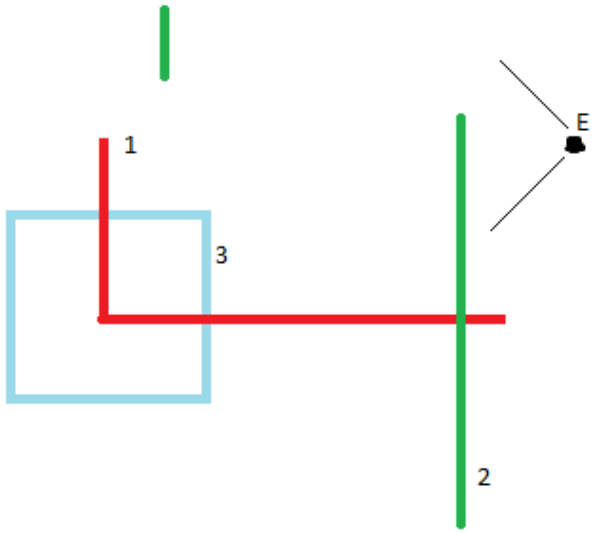


6 Exercise 6

6.1 Part 1

6.1.1 A

Hidden surface removal removes objects which behind other objects. The reason the blue object is visible at first (without HSR) is that it is drawn last, which is indicated by the numbers on the drawing.



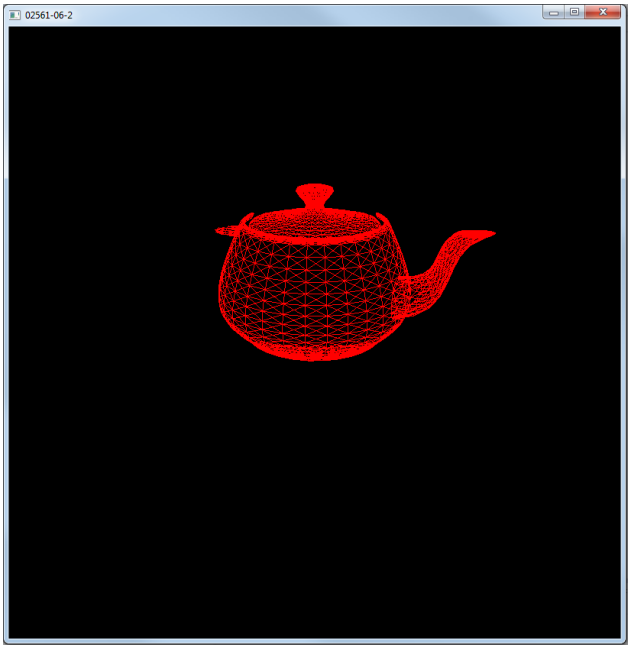
6.1.2 B

The small green object is rotated along the y-axis which makes it face away from the eye point. Face culling then removes it by looking at the orientation.

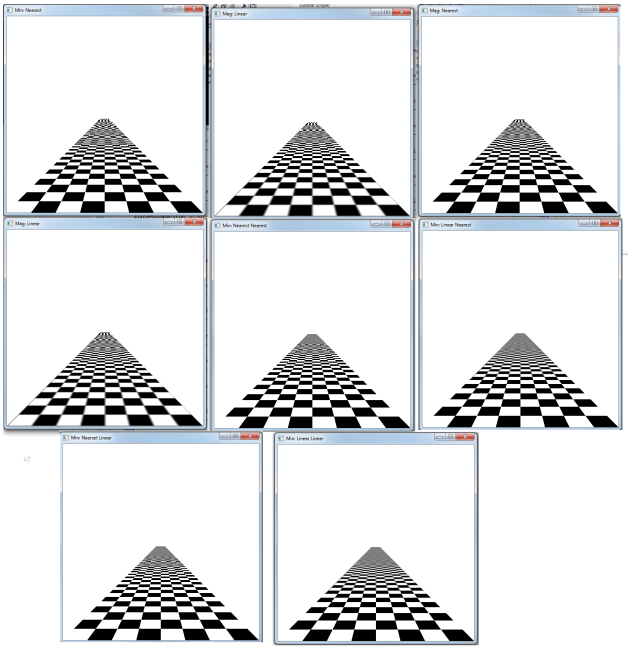
6.1.3 C

OpenGL decides what the orientation is, by looking at the direction of a surface's normal vector.

6.2 Part 2

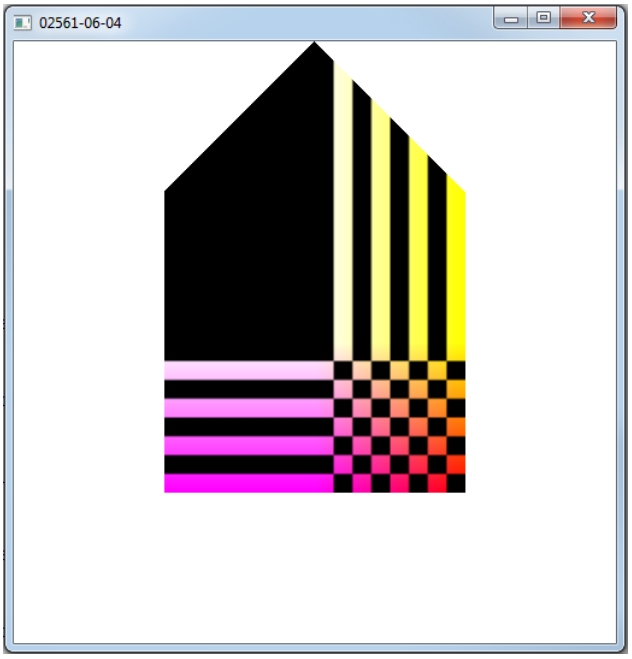


6.3 Part 3

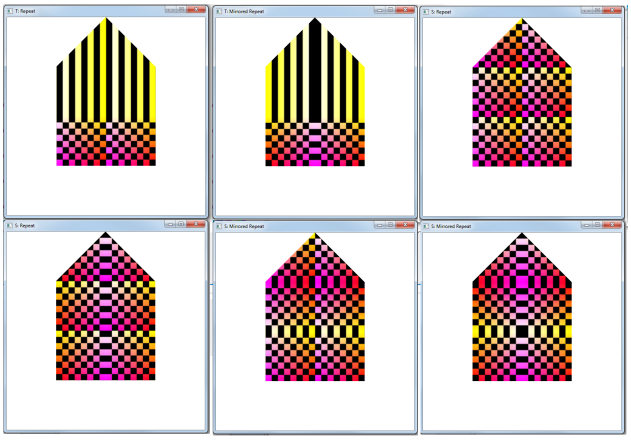


6.4 Part 4

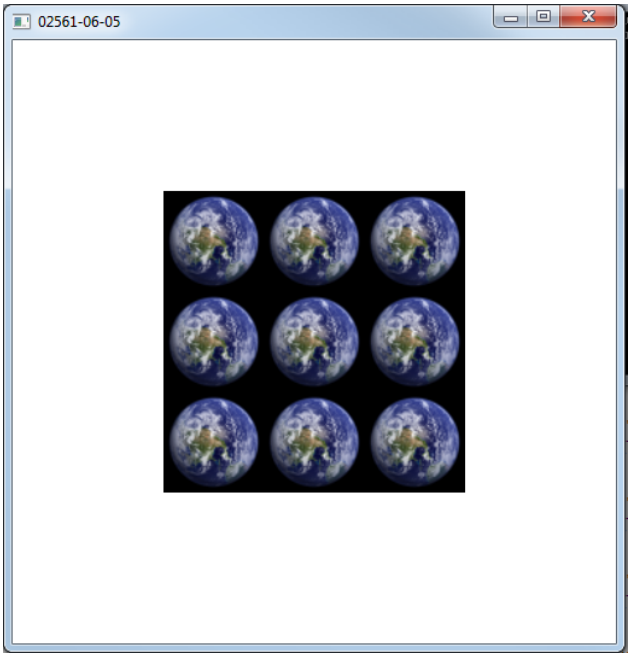
6.4.1 A



6.4.2 B



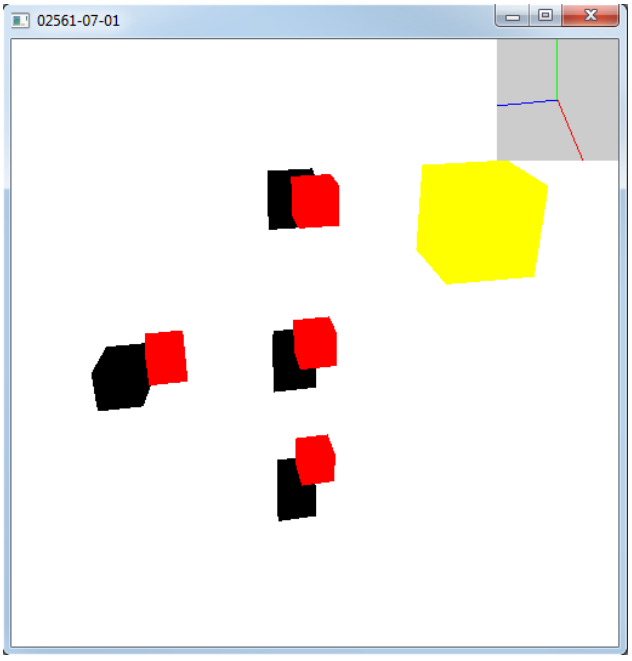
6.5 Part 5



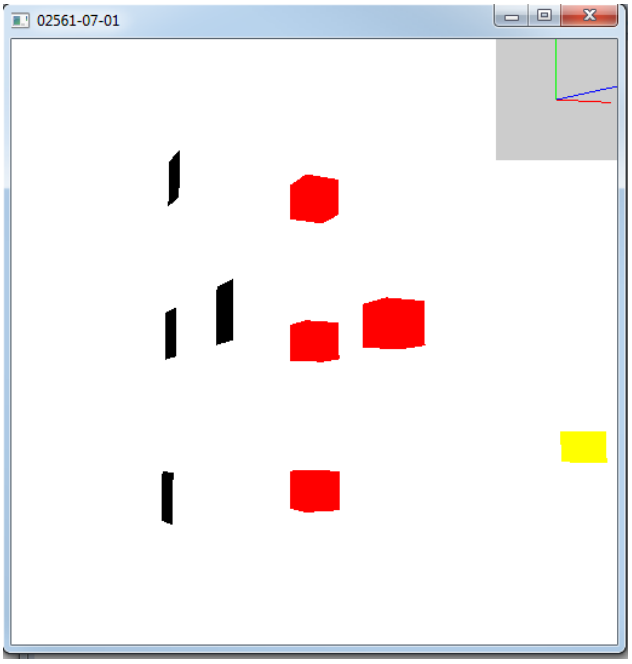
7 Exercise 7

7.1 Part 1

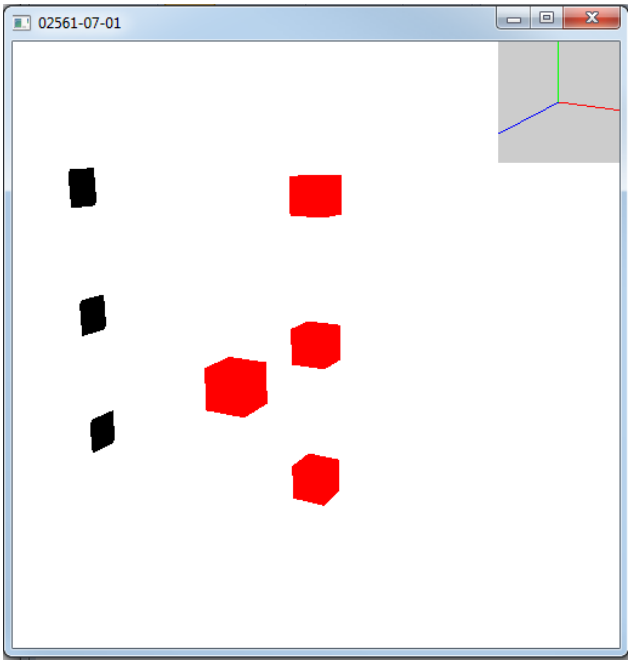
7.1.1 A



7.1.2 B



7.1.3 C

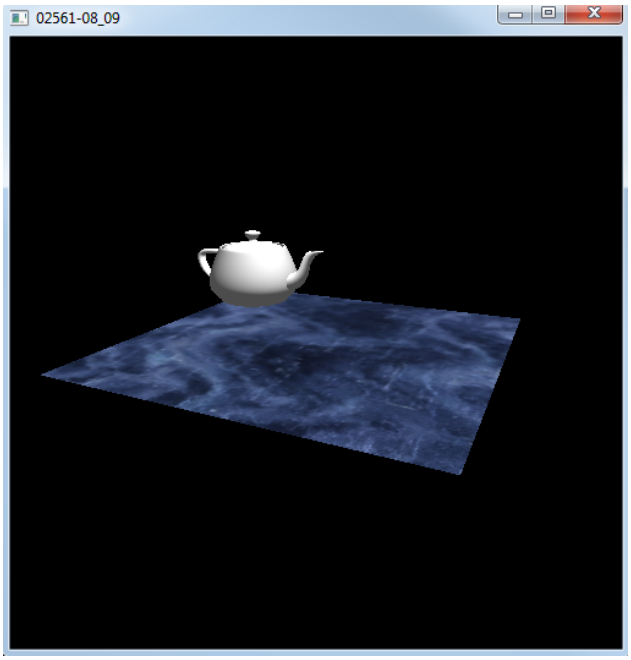


7.1.4 D

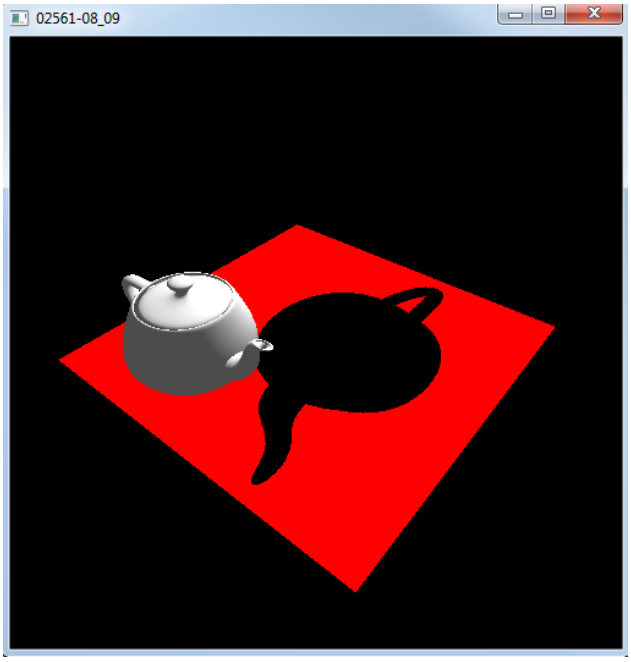
The 4 boxes are visible in the above screenshots.

8 Exercise 8

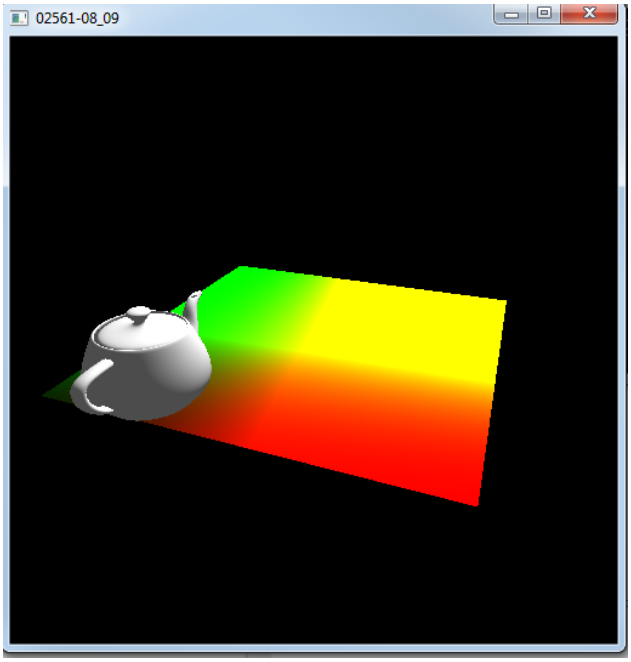
8.1 Part 1



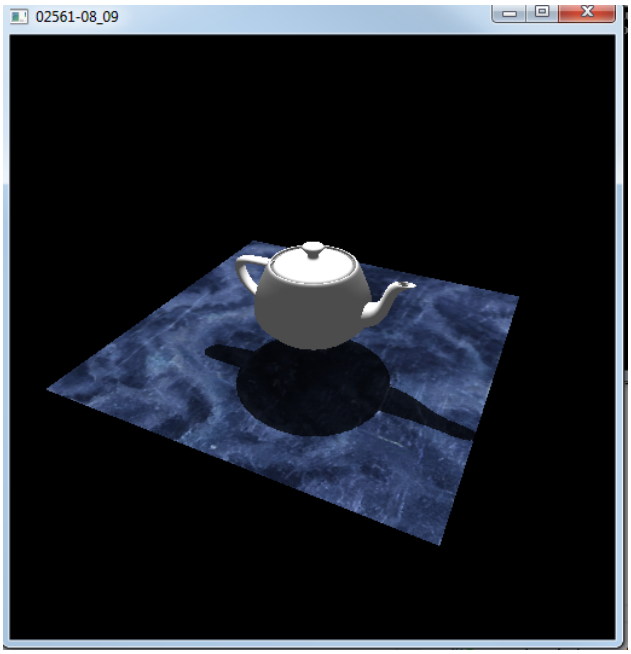
8.2 Part 2



8.3 Part 3



8.4 Part 4



9 Exercise 9

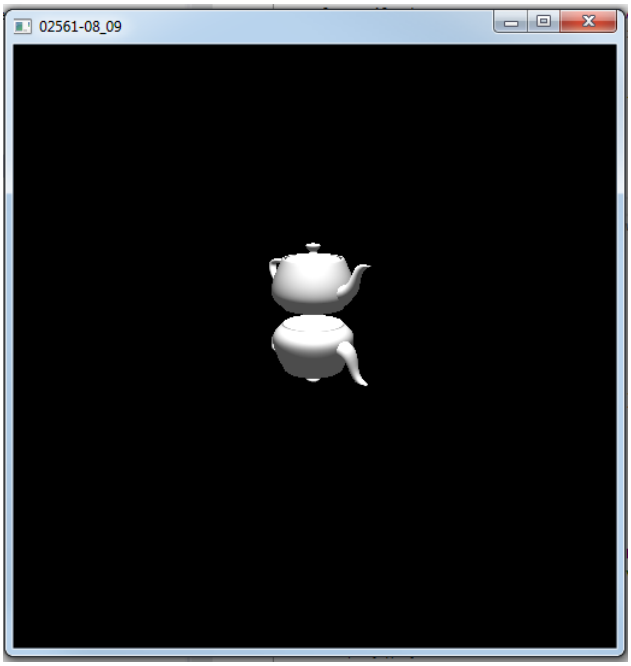
9.1 Part 1

In the picture below we have pasted the code the setup stencil function including our comments to explain what it does.

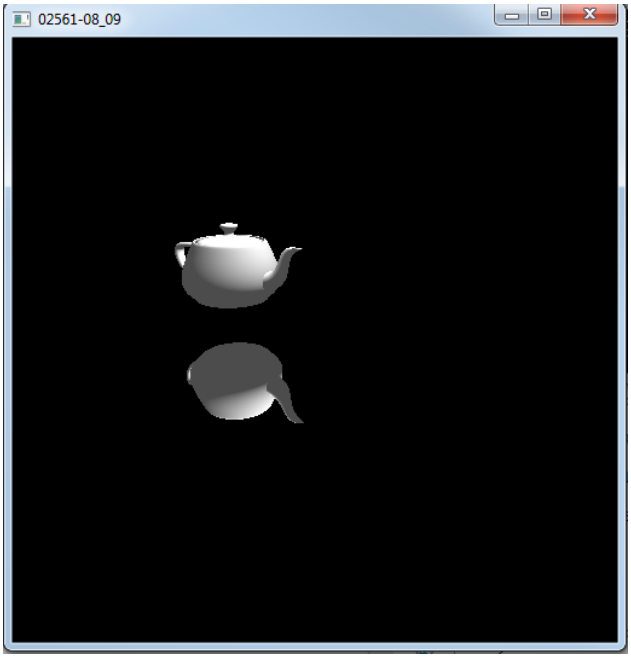
```
52 void setupStencil(mat4& projection, mat4& modelView){
53     glClear(GL_STENCIL_BUFFER_BIT);          // Clear stencil buffer (set values to 0)
54     glStencilFunc(GL_ALWAYS, 1, 1);          //Test always success, value written 1
55     glColorMask(false, false, false, false); //Disable writting in color buffer
56     glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE); //Stencil & Depth test passes => replace existing value in stencil buffer
57
58     for (int i=0;i<meshes.size();i++){
59         if (drawStencil[i]){
60             meshes[i].drawMesh(projection, modelView); // render object to stencil
61         }
62     }
63
64     glColorMask(true, true, true, true);    //Enable writting in color buffer
65     glStencilFunc(GL_EQUAL, 1, 1);          //Draw only to color buffer where stencil buffer is 1
66     glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP); //Stencil buffer read only
67 }
```

9.2 Part 2

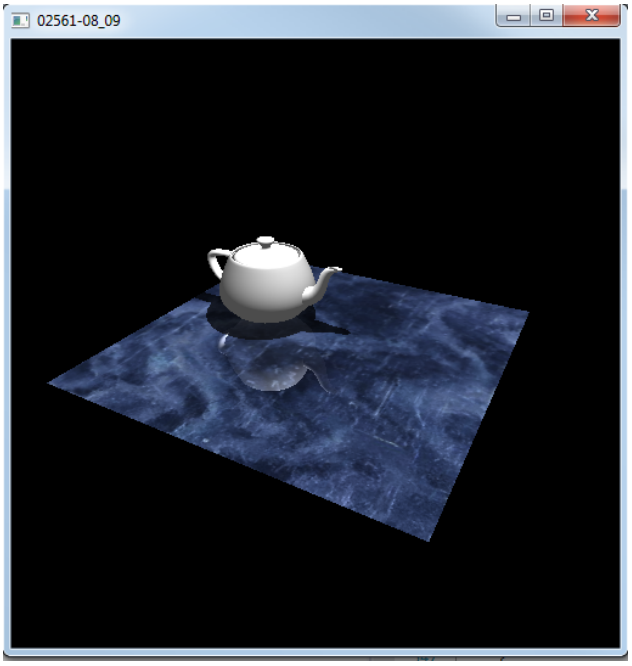
9.3 Part 3



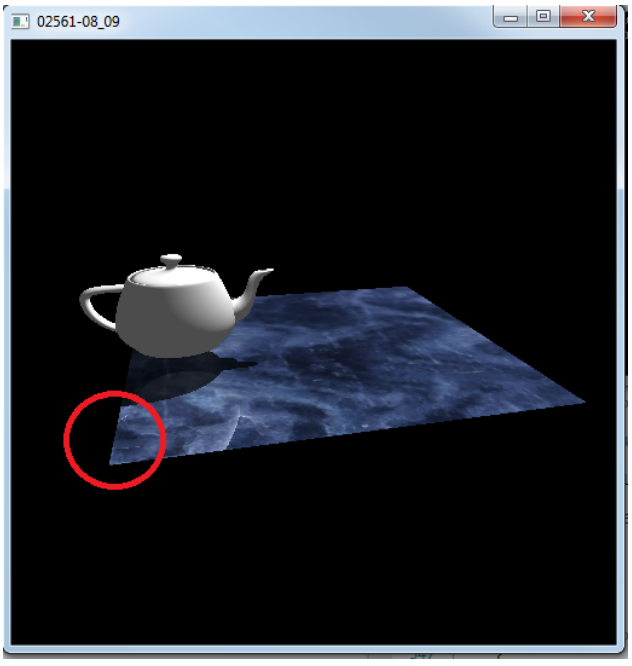
9.4 Part 4



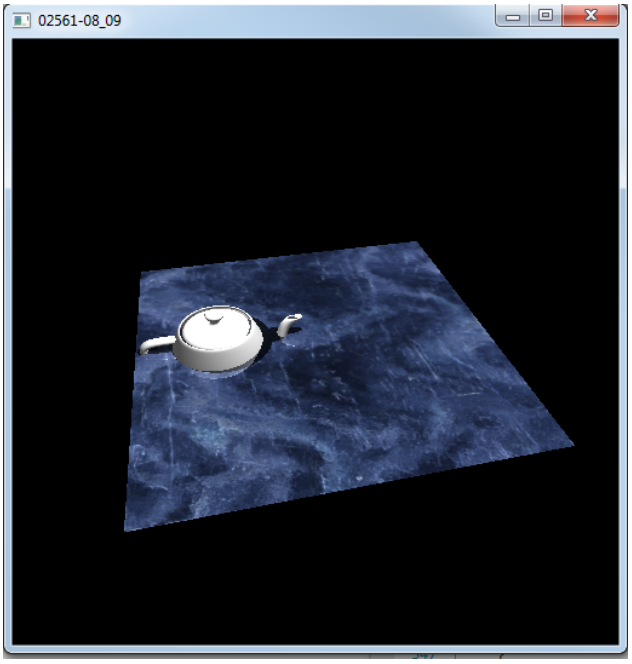
9.5 Part 5



9.6 Part 6



9.7 Part 7

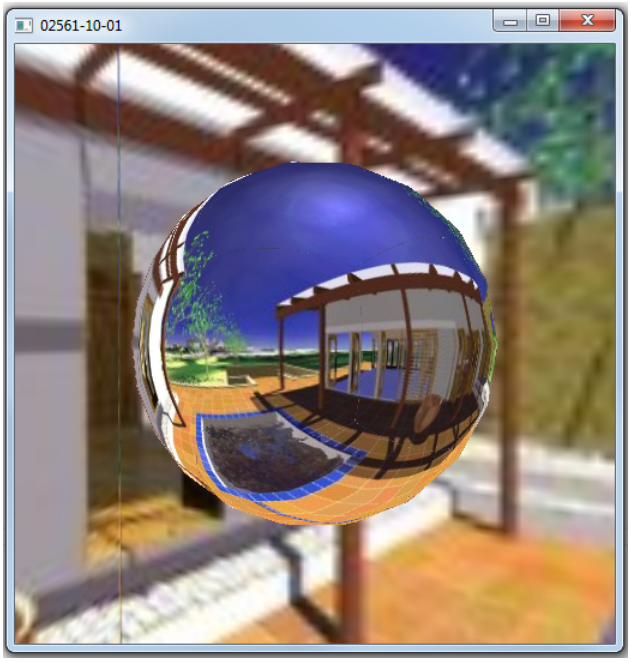


10 Exercise 10

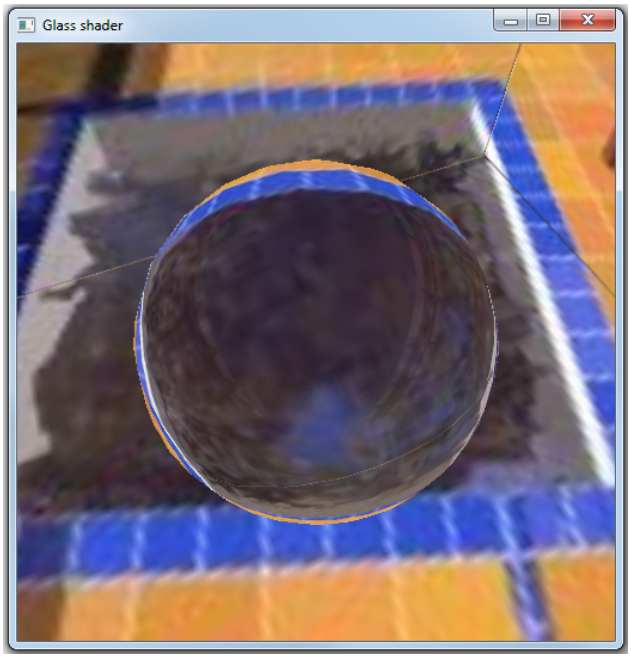
10.1 Part 1

The skybox is visible in the following screenshots.

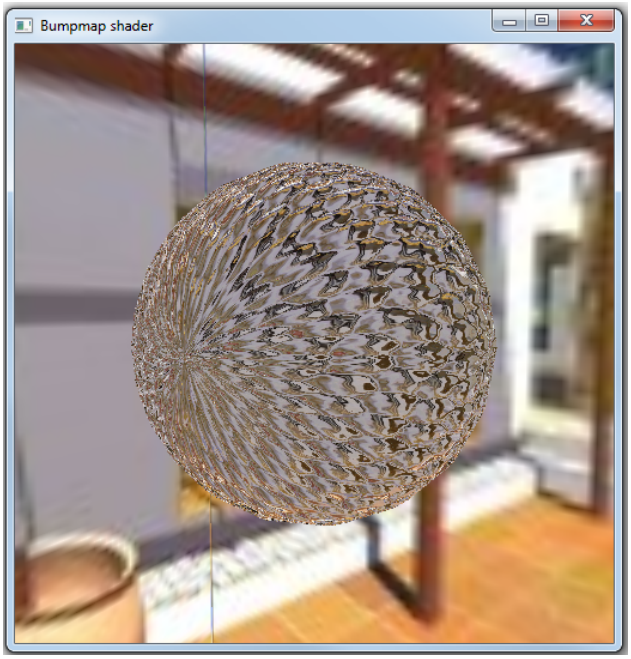
10.2 Part 2



10.3 Part 3

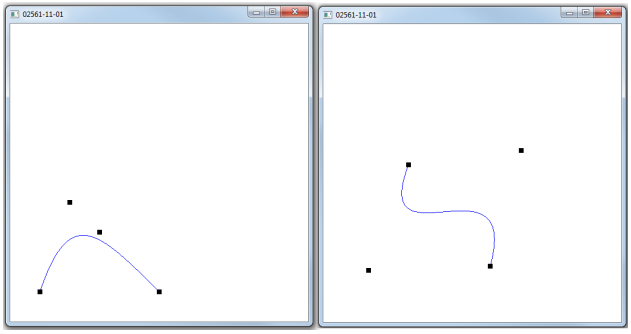


10.4 Part 4

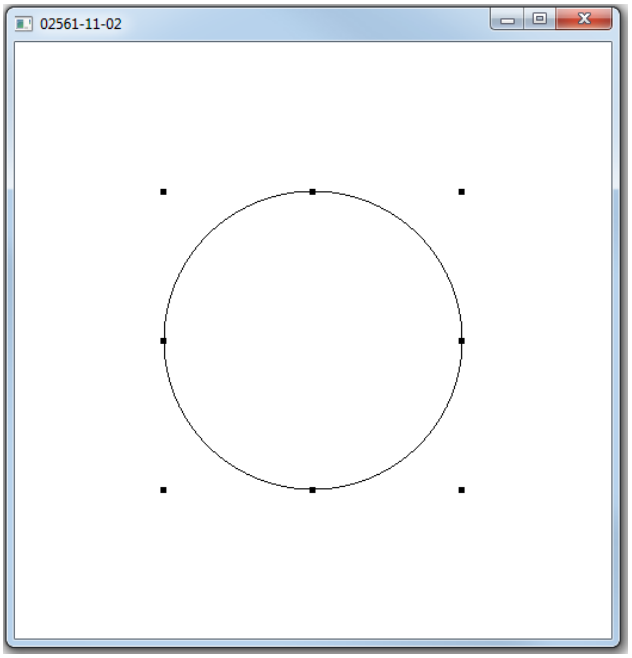


11 Exercise 11

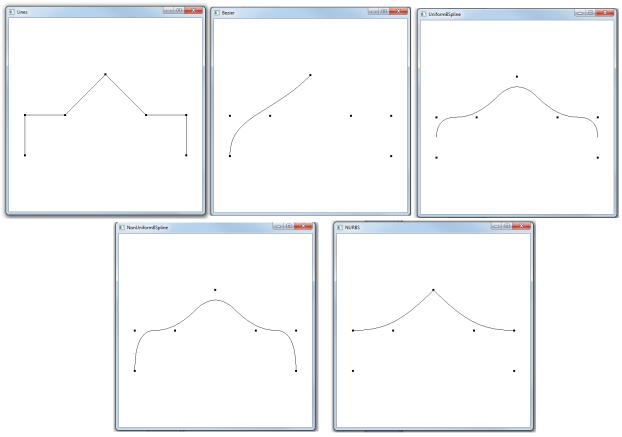
11.1 Part 1



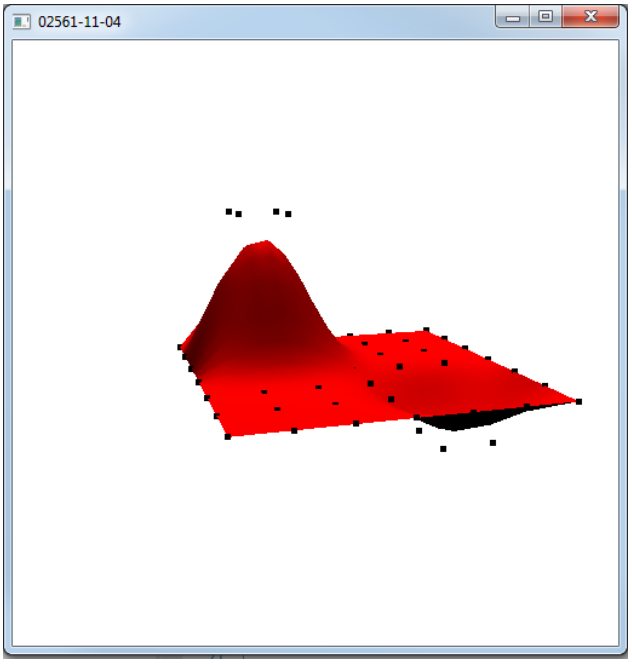
11.2 Part 2



11.3 Part 3



11.4 Part 4

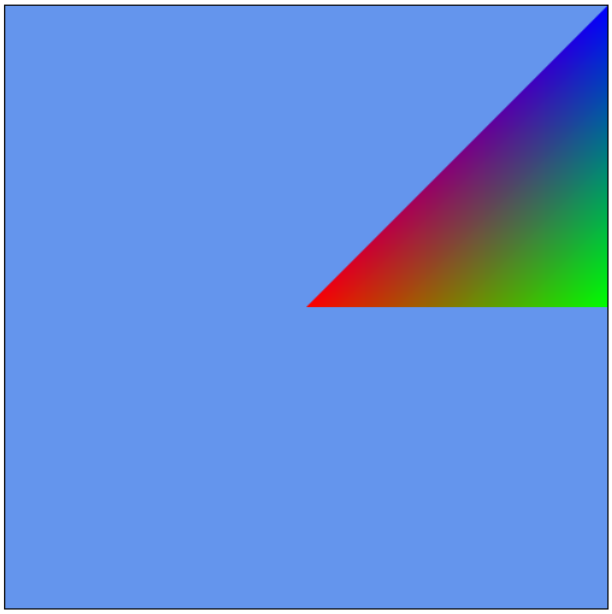


12 Exercise 12

12.1 Part 1

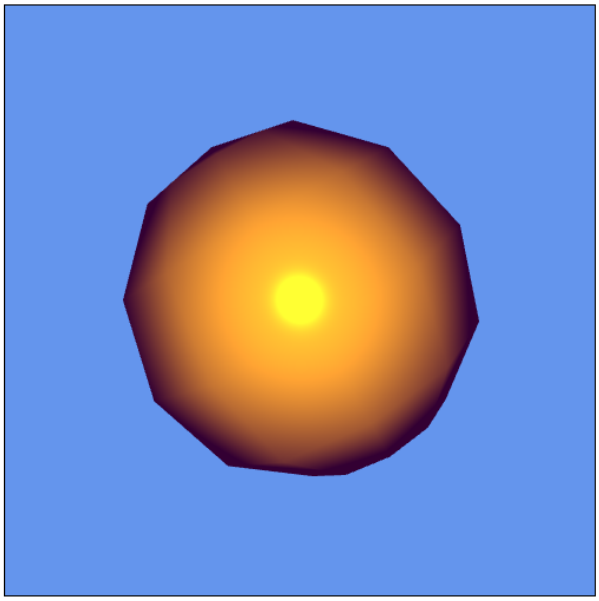
12.2 Part 2

12.3 Part 3

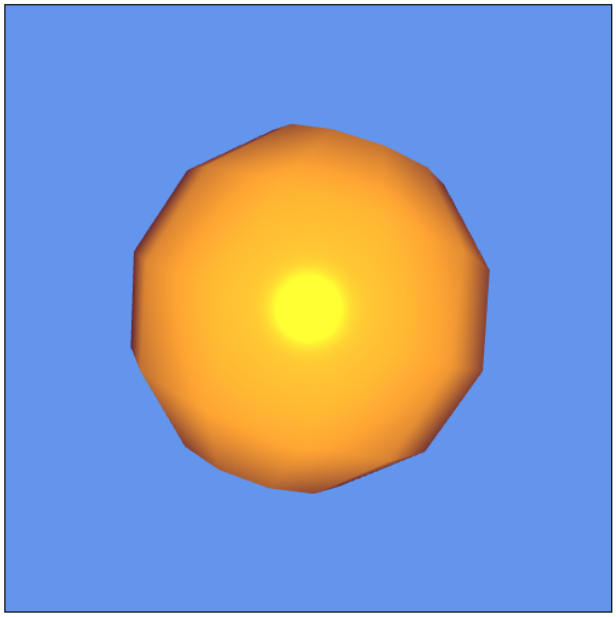


12.4 Part 4

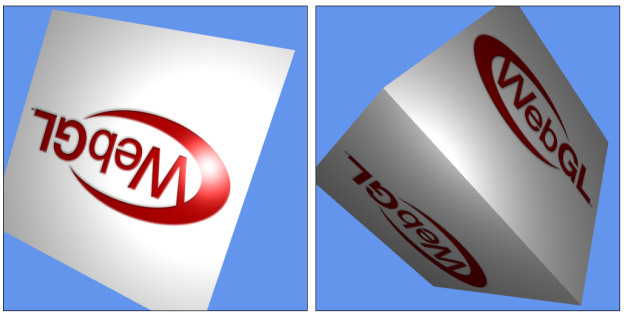
12.4.1 A



12.4.2 B

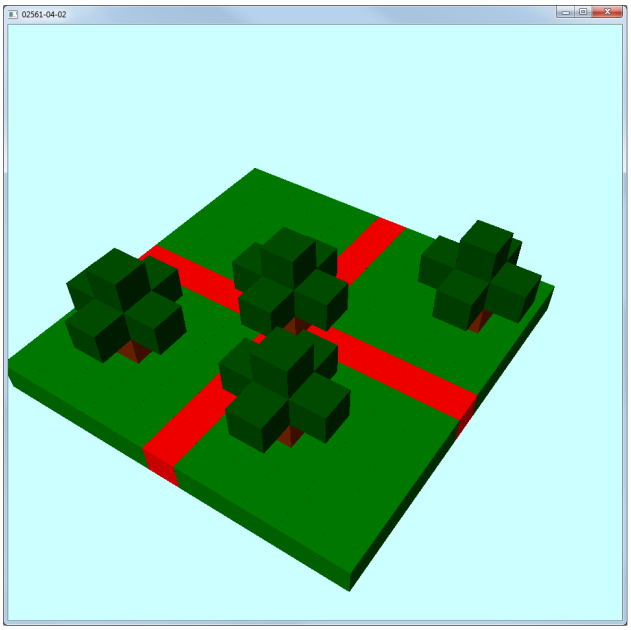


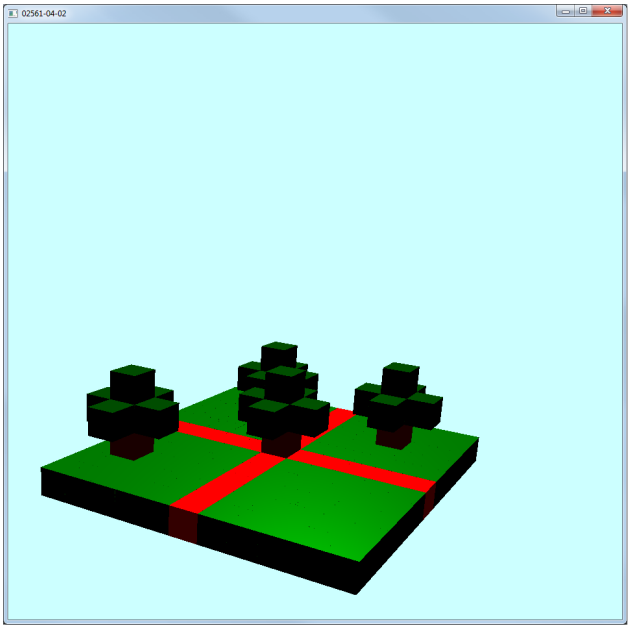
12.5 Part 5

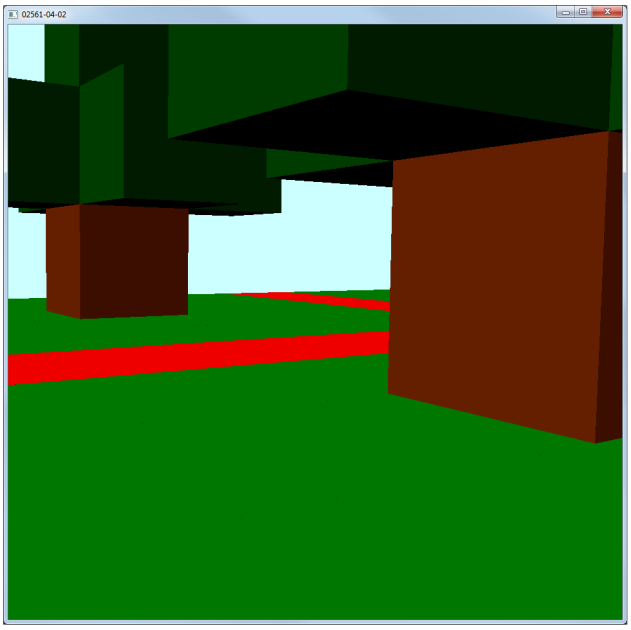


13 Project

We load our voxels by file.




02561-04-02


02561-04-02