Computer Graphics

Lab exercises 2014-2015

Dimitrios Emmanouil Karavias – s141278

Exercise 1 – Primitives and attributes.

Part 1 – Hello triangle world.

a. Try to understand the purpose of each function of the C++ program main-01-01.cpp. Write what each function in the program does.

Display function.
This function displays the shapes on the screen. It clears the screen to a default color and the color buffer. Then it selects a shader program to use, then it binds buffers with the vertex data and draws the vertices. The result has been drawn to a buffer that is not displayed on the screen. In the last part the function swaps the buffers and displays the result.

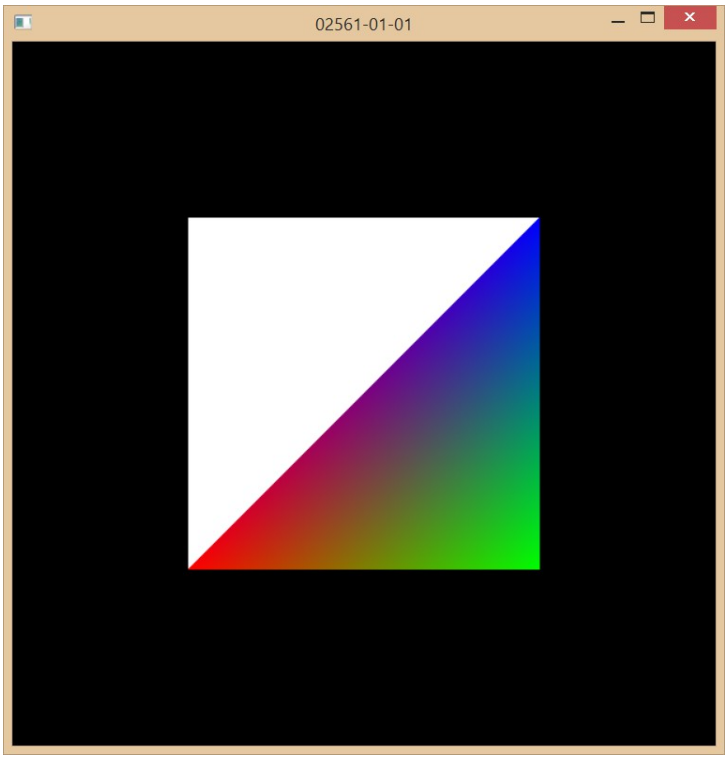
Reshape function.
This function is called when the application window is resized. Here it calls the viewport function of opengl with the new window size.

LoadBufferData function.
This function generates the buffers that contain the vertex data that opengl uses to display the different shapes.

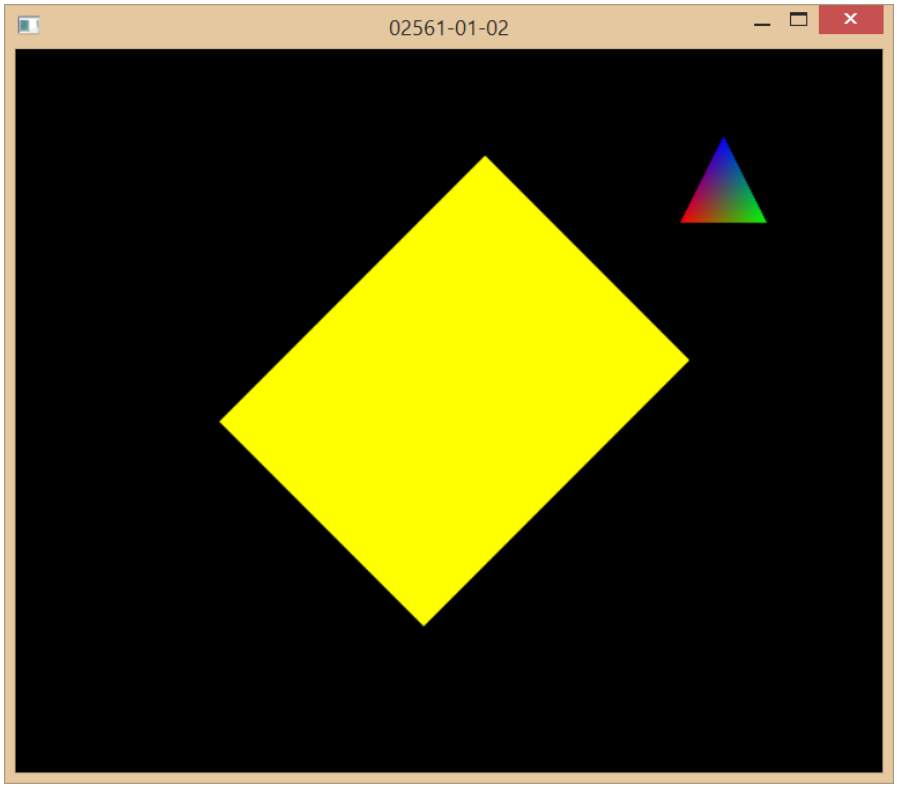
LoadShader function.
This function initializes a shader program and retrieves the shader attribute positions that can be used to assign values.

Main function.
This is the starting point of the program. The glut application gets initialized.

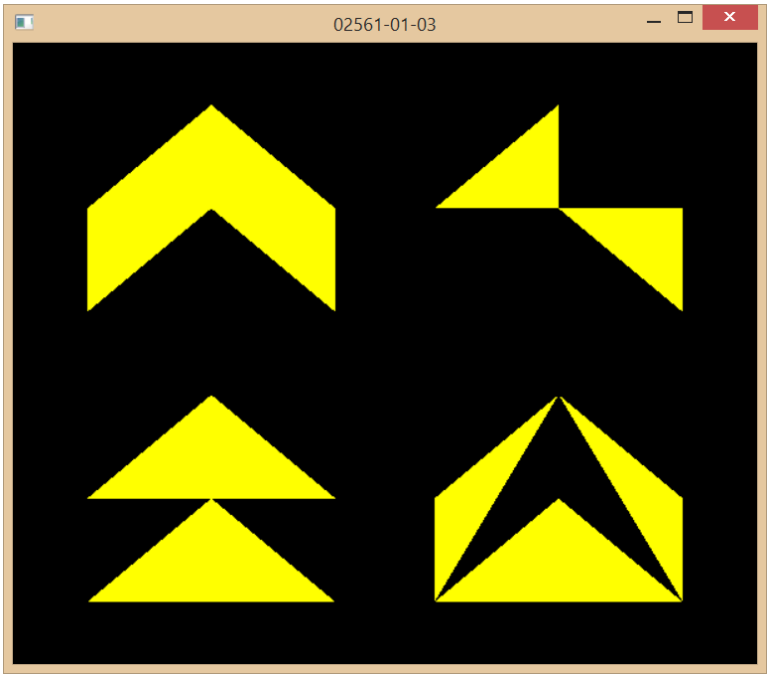
b.



Part 2 – Vertices and transformations

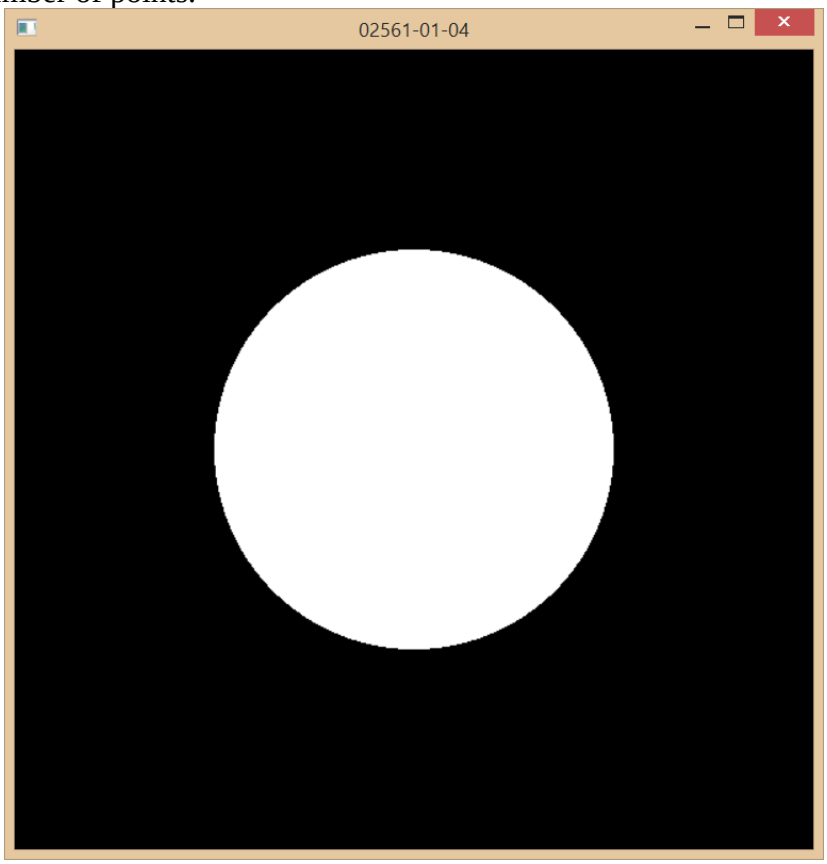


Part 3 – Draw arrays and draw elements using different primitive types

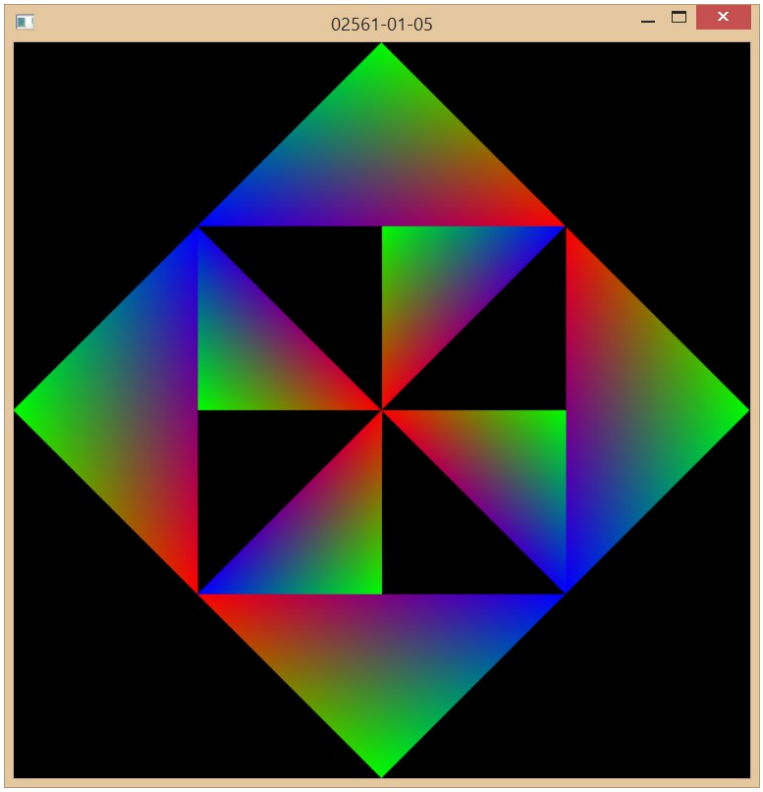


Part 4 – Creating a mesh

For this part I created a function that calculates the vertices using the NUMBER_OF_VERTICES number for the number of points.



Part 5 – Model transformation



Part 6 – Viewport transformations

a)

$$\begin{bmatrix} 1/wx & 0 & wx/2 \\ 0 & 1/wy & wy/2 \\ 0 & 0 & 1 \end{bmatrix}$$

b)

Translation:

$$\begin{bmatrix} 1 & 0 & wx/2 \\ 0 & 1 & wy/2 \\ 0 & 0 & 1 \end{bmatrix}$$

Scale:

$$\begin{bmatrix} 1/wx & 0 & 0 \\ 0 & 1/wy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

c) We have to define the Viewport. OpenGL uses this information to create the window – viewport transformation.

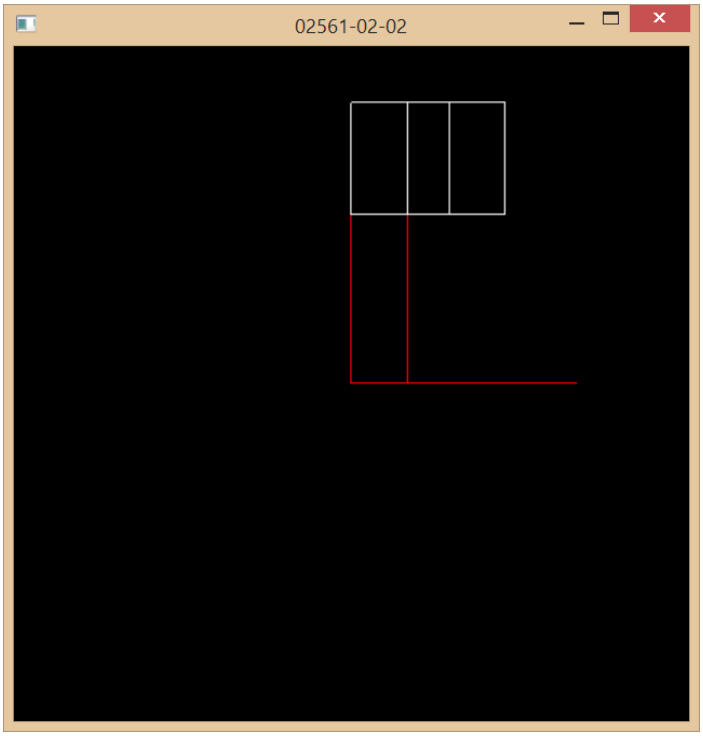
Exercise 2 – Projections – Virtual Camera

Part 1 – Hello 3D

The main difference between the vertex struct in exercise 1 is that the vertex in exercise 1 used vec2 or vec3 for the position attribute, and it has a color attribute as well. In this part of the exercise 2 the vertex struct has vec4 position attribute. The display function differs to the fact that it defines projection and model view matrices.

Part 2 – Model transformations in 3D

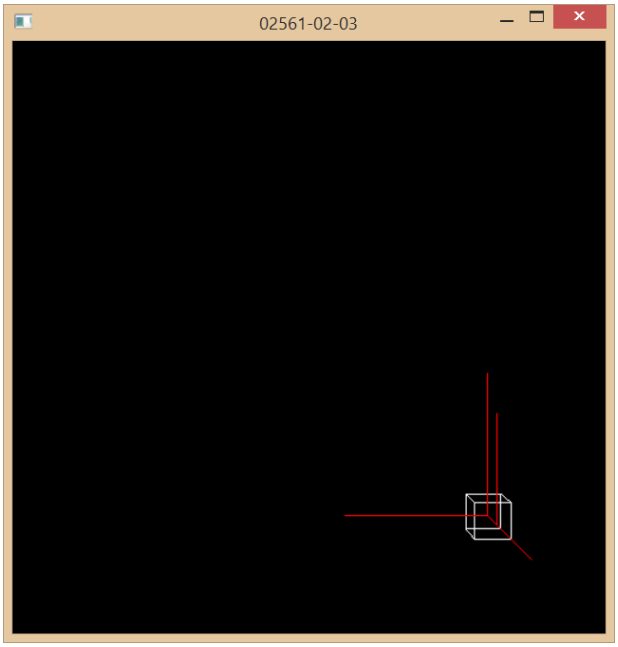
a)



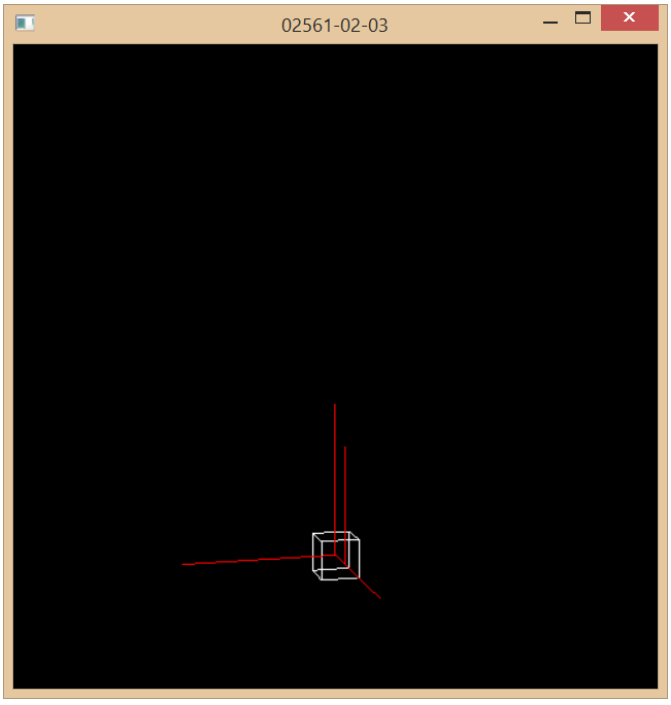
b) The transformations I used are Translate, Scale and Rotate.
c) The order I used them is first Translate, then Scale and then Rotate. By doing the Translate of the position first, we can change the position by 3 units with respect to the original scale and we avoid bad effects of translating the scaled object.

Part 3 – View transformation

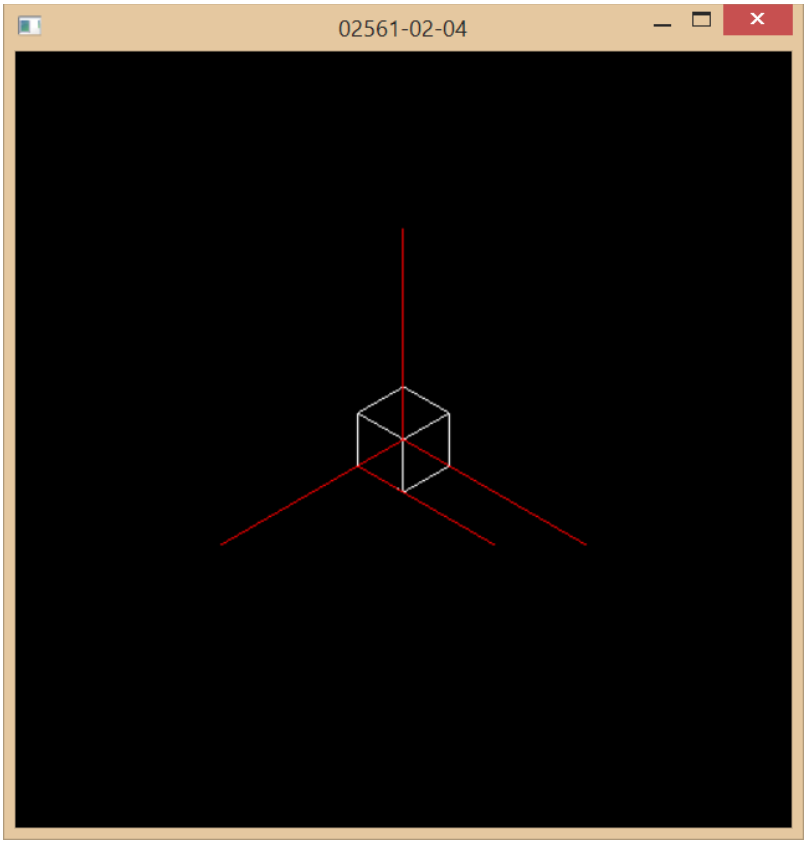
Front perspective:



X perspective:

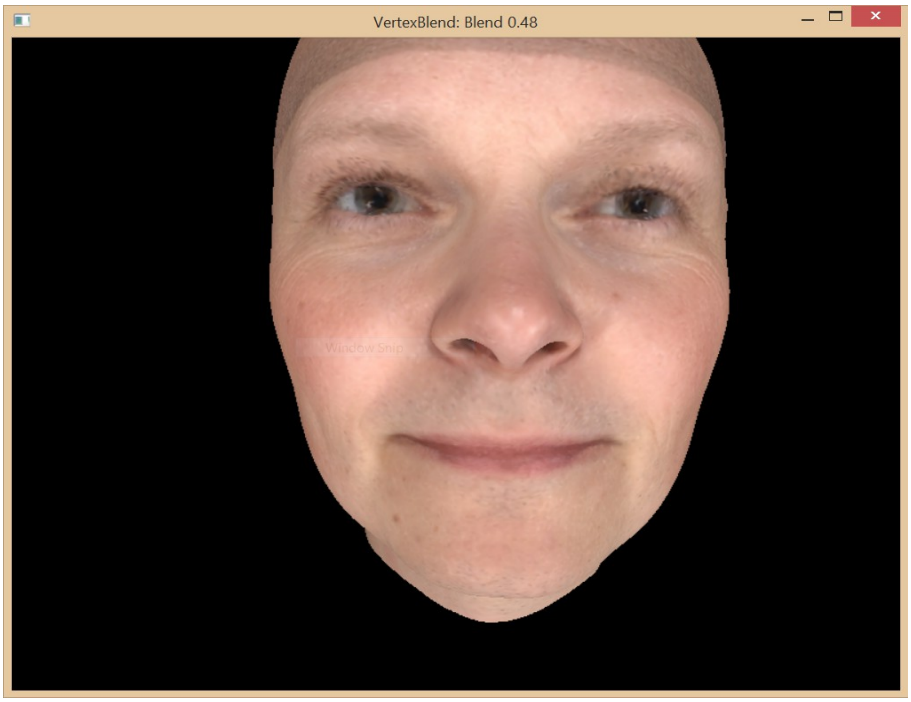


Part 4 – View transformation



Exercise 3 - Introduction to shaders and GLSL

Part 1 – Vertex blend in GLSL



Part 2 – Normal extrusion

Part 3 - Questions

What is the difference between a vertex attribute and a vertex uniform variable used in a shader? When can they be changed?

Vertex attributes and uniforms are used to pass information from the main code to the vertex code run by the GPU. The difference is that vertex attributes receive information that may be different for every vertex of the geometry it displays, vertex uniforms on the other hand have the same value for each vertex, thus they are used for different kind of purposes. For example a vertex attribute can describe the position of the vertex, while a vertex uniform would describe the position of the eye or the light. Technically both can be changed anytime at every frame, but it is a good practice to change attributes as less as possible, while changing uniforms at every frame. That is because it is more expensive to pass data for each vertex than a variable that is the same for all instances of the vertex shader.

Explain in general terms how a vertex shader works.

Vertex shader gives instruction to the GPU to manipulate the way an object is displayed. They receive attributes about the object, like the position, normals and color. They also receive unifroms like a light position or other changing factors. The vertex uses the input to change the position of each vertex of the object and the color of the vertex to achieve the desired visual result.

Explain in general terms how a fragment shader works.

The fragment shader receives input from the Rasterizer. This input could come from the main program or from the vertex shader. The fragment shader can only affect the color of the fragment. In addition the fragment shader may discard a fragment.

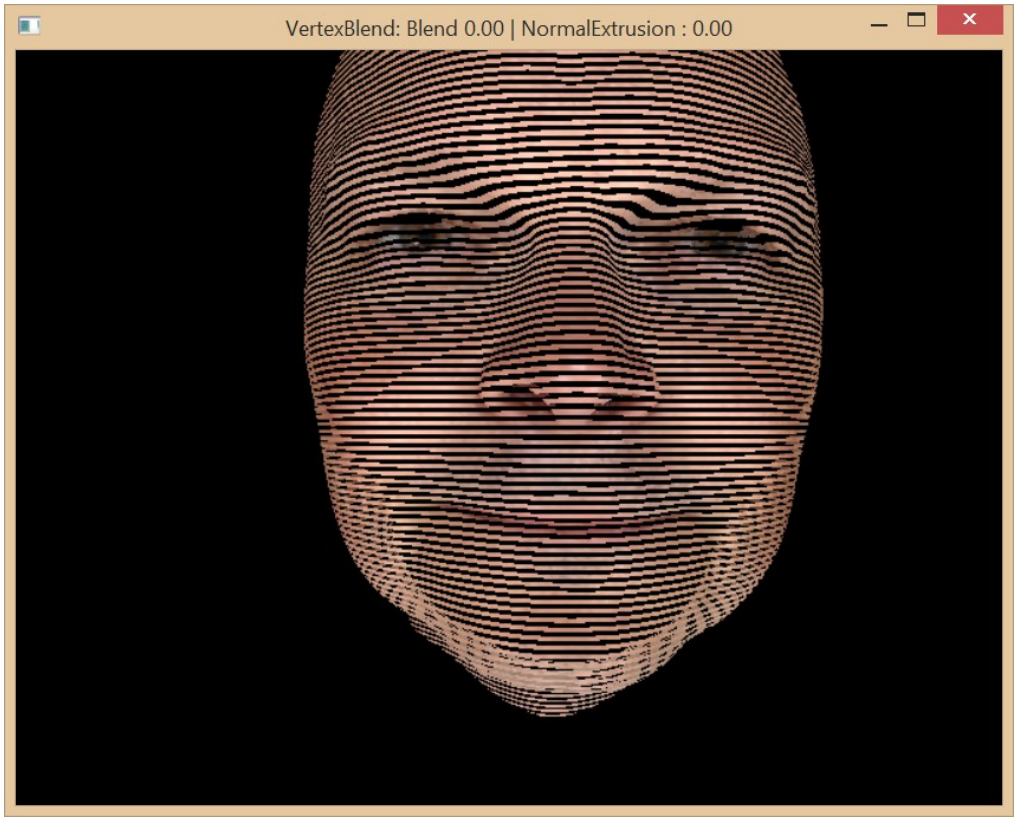
Describe what happens between the vertex shader and the fragment shader.

Between the vertex and fragment shader is where rasterizing is happening. Everything that is not visible is removed.

Is it possible to blend the color in the fragment shader instead of the vertex shader? If so, it it a good idea doing this?

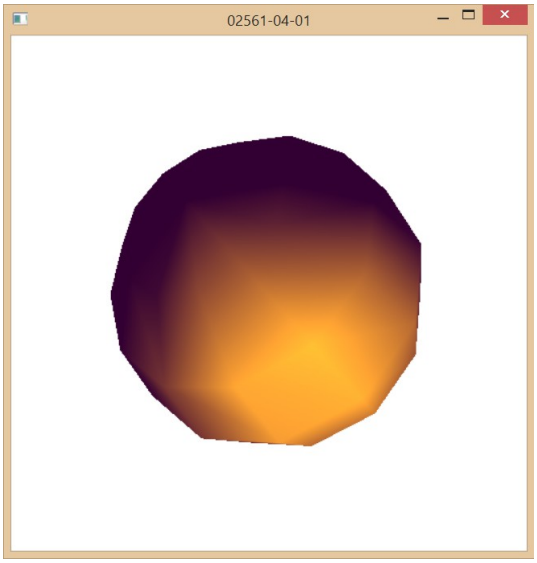
It is possible to blend the color in the fragment shader but that would require more data to be sent from vertex shader to the fragment shader, thus making the same calculation more expensive for the GPU.

Part 4 – Optional – Discarding fragments



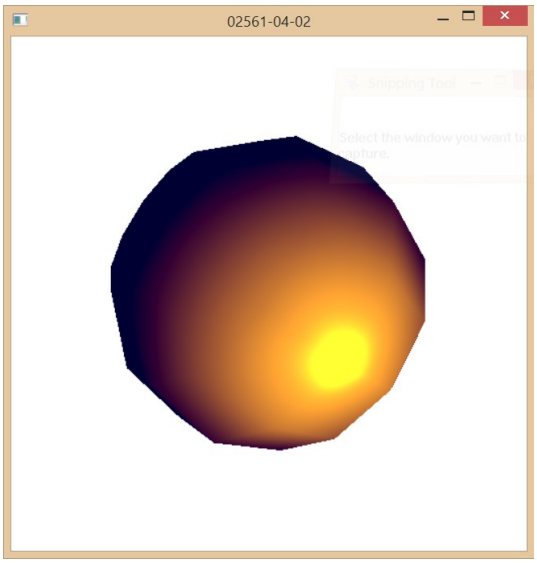
Exercise 4 – Local illumination / Phong lighting and gouraud / Phong shading

Part 1 – Gouraud shading



Part 2 – Phong shading

Point light:



Directional light:

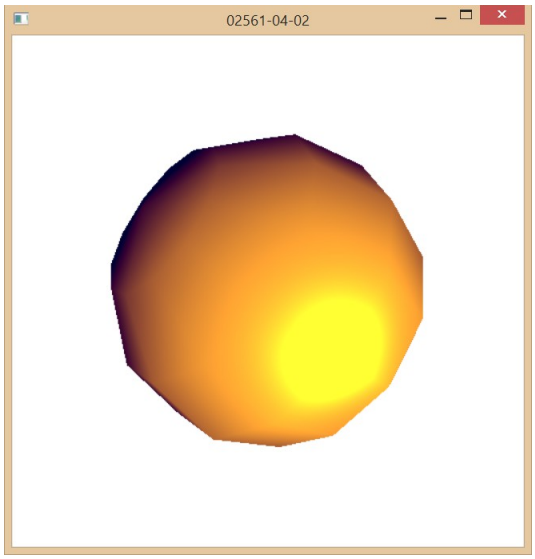


Figure 4-2: Directional light.

Part 3 – Shading and light questions

a) What is the difference between Phong shading and Phong lighting

Phong lighting is an equation that describes how we combine ambient, diffuse and specular color to illuminate the vertices. Phong shading is per pixel lighting. The normal is interpolated from per vertex normals before the per pixel lighting calculation.

b) What is the difference between Gouraud shading and Phong shading. What is the pros and cons of each?

Gouraud shading doesn't have smooth surfaces, while phong shading has. The advantage of the Gouraud shading is in the low complexity that makes it fast, in contrast with the phong shading which is more expensive.

c) What is the difference between point light and directional light?

Directional uses parallel rays from the source to a given direction. This direction is the same for all vertices and the effect of the light to the vertices depends on the corresponding normals. On the other hand for point light the direction of the light ray changes for every vertex and the result is a compination of the direction between each vertex and the light source, and the vertex normal.

d) Has the eye position any influence on the shading of the object?

The eye position doesn't affect the diffuse color since this color only depends on the position of the vertex and the light source. However specular color is the reflection of the light to the eyes, which depends on the position of the eyes.

e) What is the effect of setting the specular term equal to (0,0,0)?

Setting the specular term equal to (0, 0, 0) means that the specular color will not have any effect to the illumination of the polygons and only the diffuse color will affect the result.

f) What is the effect of increasing shininess exponent?

Increasing the shininess exponent makes the reflection spot smaller.

h) Explain the importance of the normal matrix. What is the purpose of the normal matrix and how is it computed.

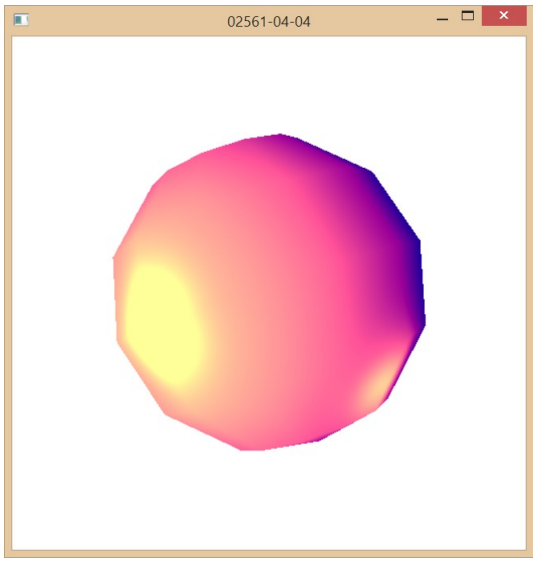
The importance of the normal matrix

I) In what coordinate space are your computing the light?

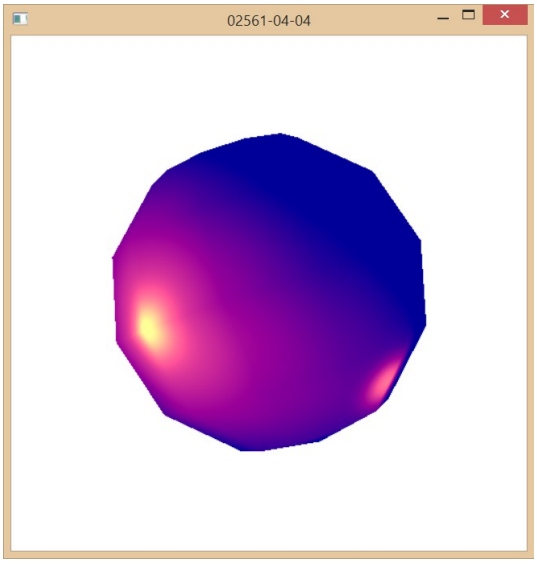
The light is computed in the world-space coordinate system.

Part 4 – multiple light sources

Directional light with two of the three light sources visible:



Point light with two of the three light sources visible:



Exercise 5 – Input – mouse, keyboard, menus, windows, selection, pick

Part 1 – Pick

a) What does the function pick actually do?

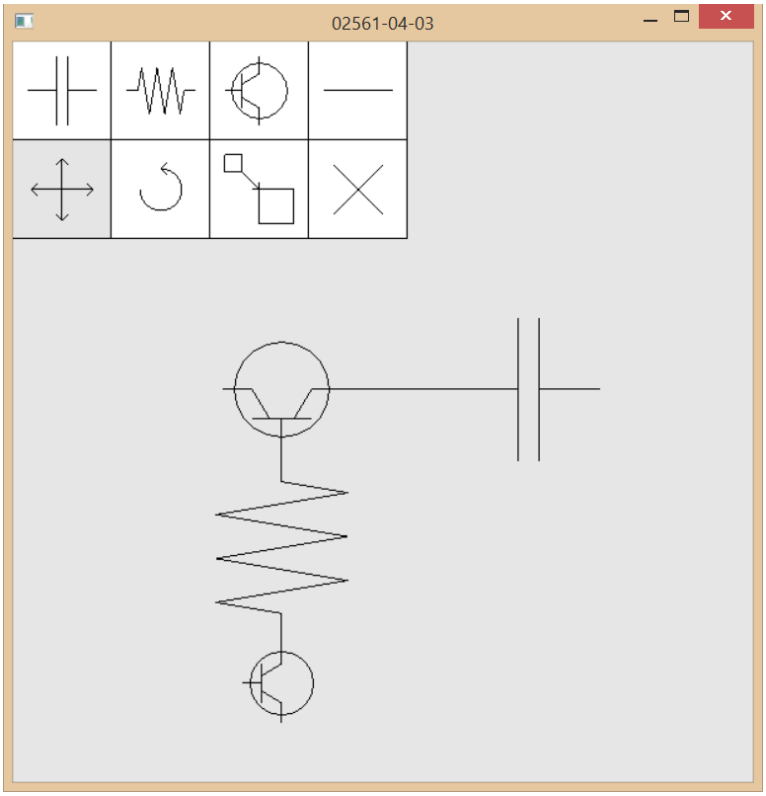
The pick function receives vec2 position attribute. What it does is that it checks if that position corresponds to a button in the window view. If the position corresponds to a button then an identifier to what the button was is returned by the function, otherwise the function returns the value zero. In more details, the function translates the screen position to a world position. It then checks if the y position is in the top tenth of the screen. Finally determines the corresponding button by the x position.

Part 2 – Select

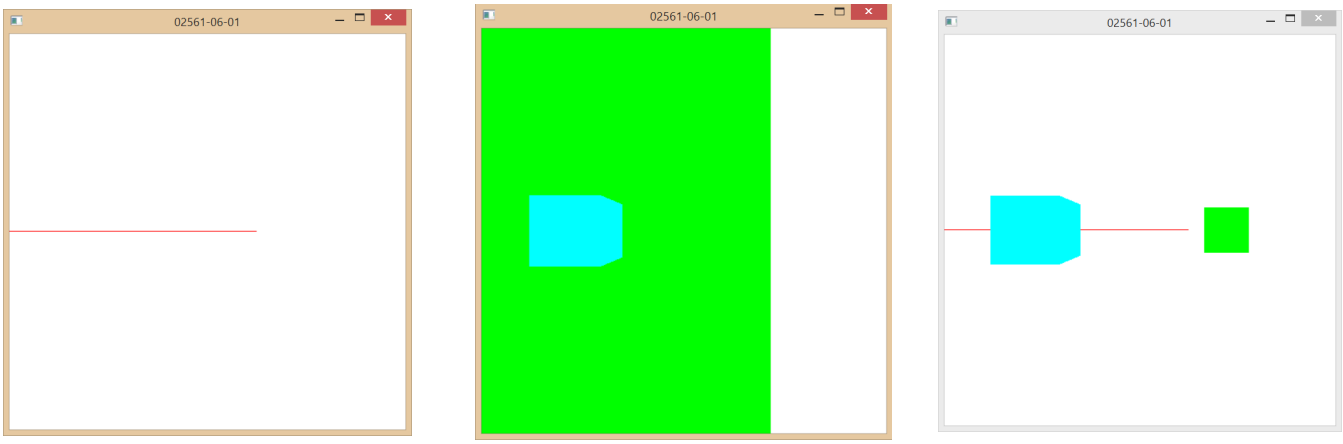
a) Outline what steps are needed to perform a selection.

To perform a selection the program is initializes the glutMouseFunc with function pickSquares. Then everytime a mouse button is clicked, it will execute the function pickSquares. When a click is performed the pick Squares checks that it is a left click. Then the function sets the color uniform to a given color. The render function then sets the ids for the selectBuffer. Finally by using the mouse coordinates we get the id for the given position and thus determining the selection.

Part 3 – Circuit diagram editor



Exercise 6 – Texture Mapping – and more..

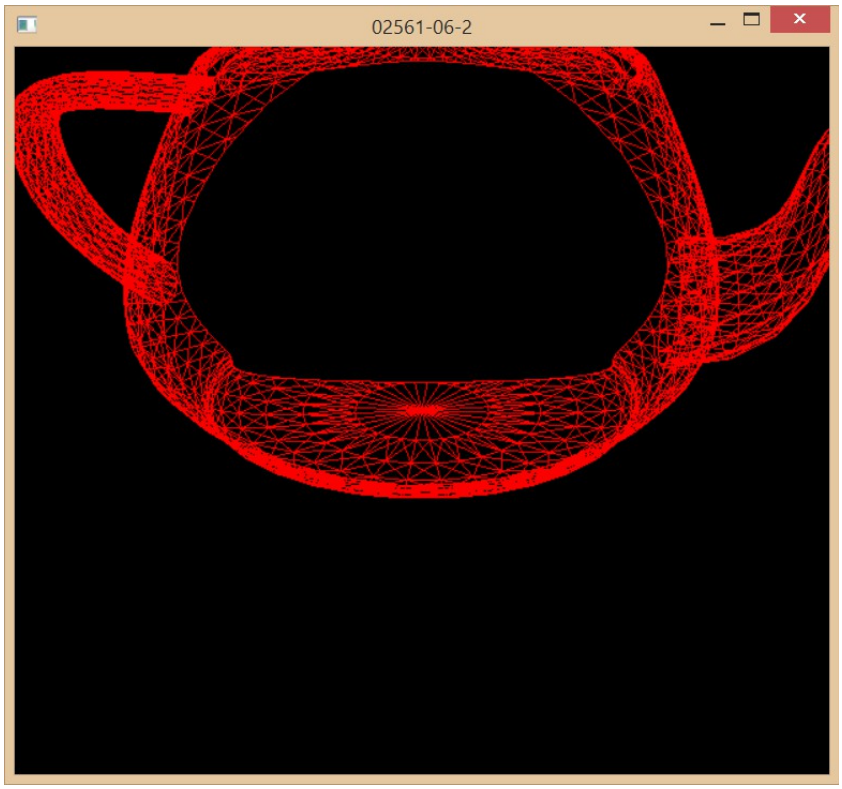


Sketch of the scene with culling
GL_FRONT, GL_BACK and GL_FRONT_AND_BACK enabled.

Part 1 – Hidden surface removal. Face culling.

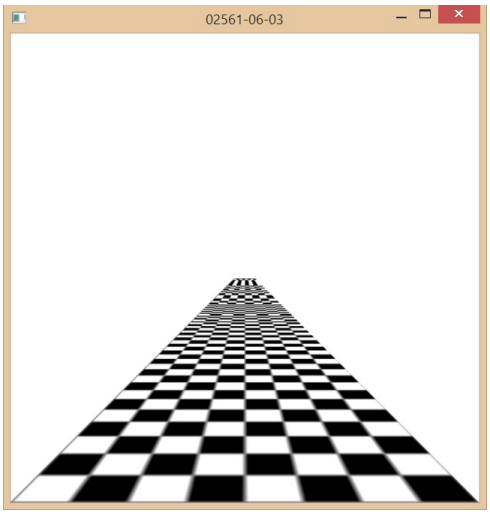
When we enable face culling then some geometries are dissapearing depending on the culling mode we are using. OpenGL determines if an object is front or back facing by the order the vertices are drawn for each polygon.

Part 2 – Viewing frustrum.

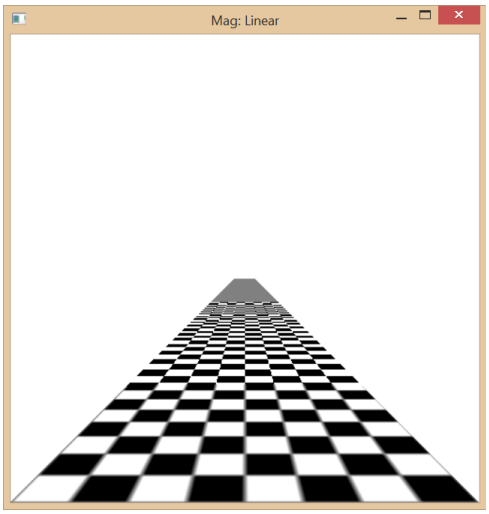


Part 3 – Textures and filter

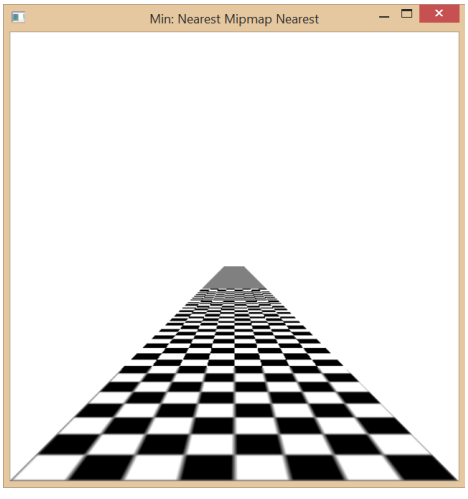
The different filter techniques used in OpenGL are shown in the figures below. They keys that have been used to change the filter are the numbers 1-6.



Mag nearest



Mag linear.



Min nearest mipmap niearest.

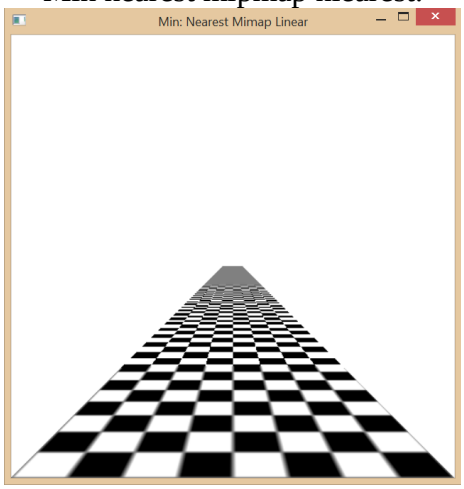
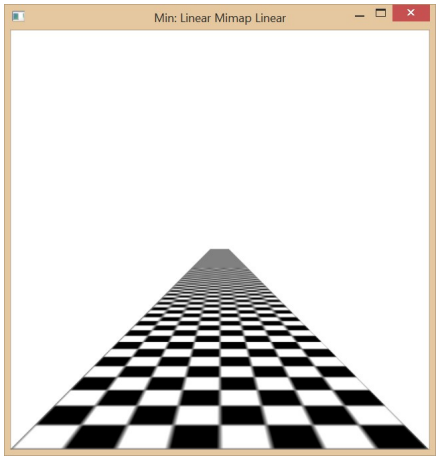
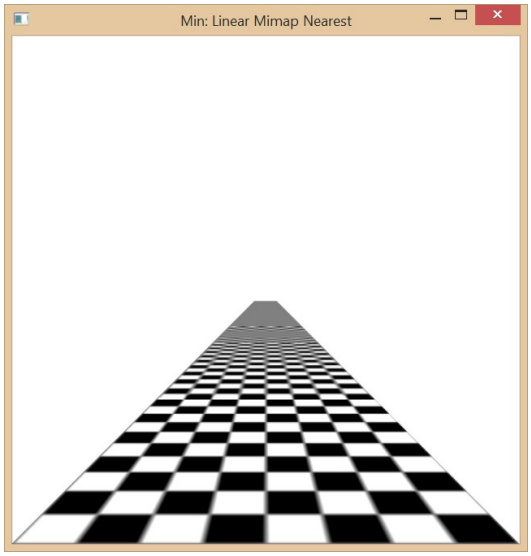


Figure 6-7: Min nearest mipmap linear.



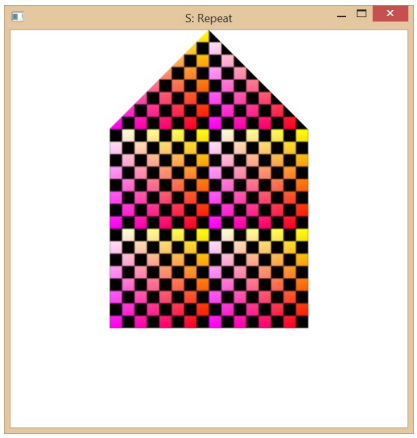
Min linear mipmap linear.



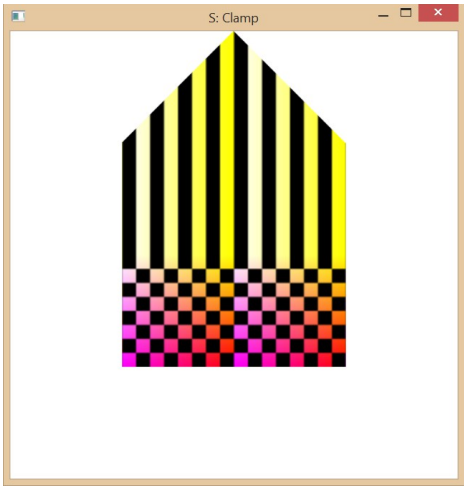
Min linear mipmap nearest.

Part 4 – Textures wrap and repeat

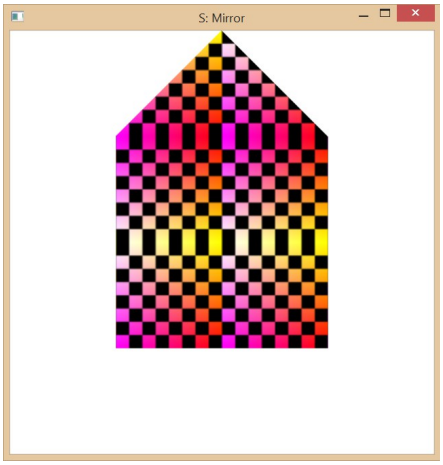
To show the different usages of texture wrapping I implement the keyboard function where I use the keys 1 to 3 to change the GL_TEXTURE_WRAP_T attribute and 4 to 6 to change change the GL_TEXTURE_WRAP_S attribute. Then we can combine the two attributes to get 9 different visual results. The results are shown in the figures below:



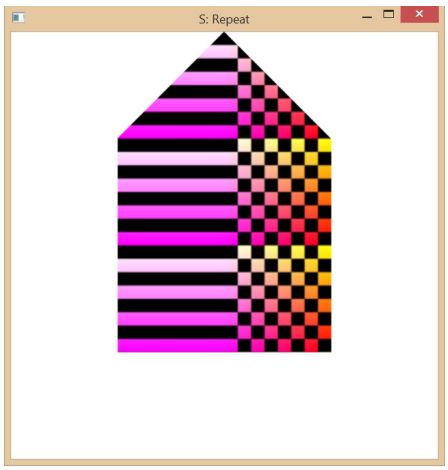
T Repeat, S Repeat.



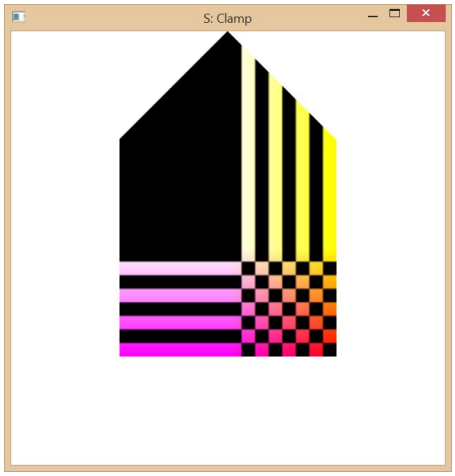
T Repeat, S Clamp.



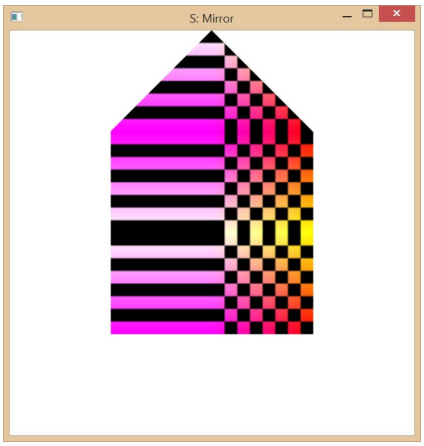
T Repeat, S Mirror.



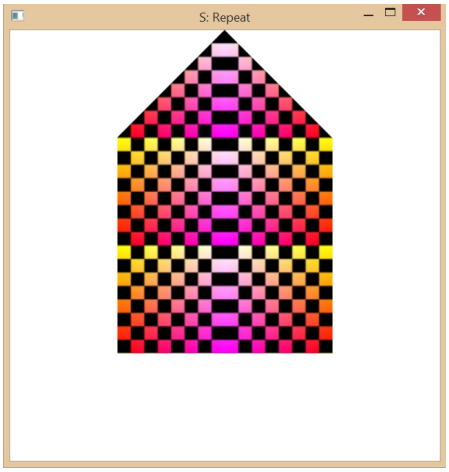
T Clamp, S Repeat.



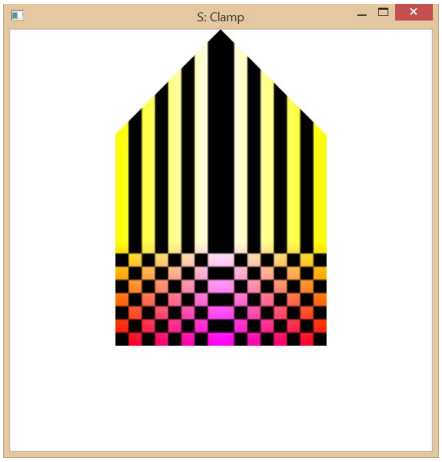
T Clamp, S Clamp.



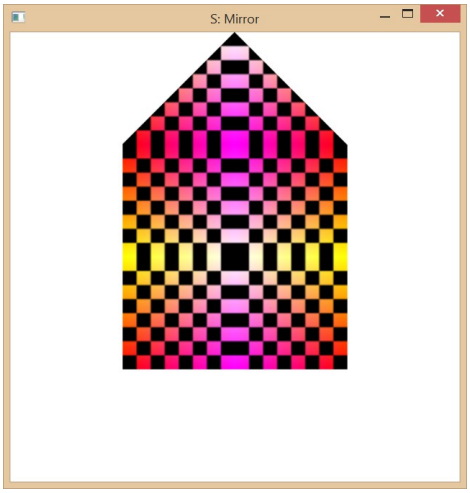
T Clamp, S Mirror.



T Mirror, S Repeat.



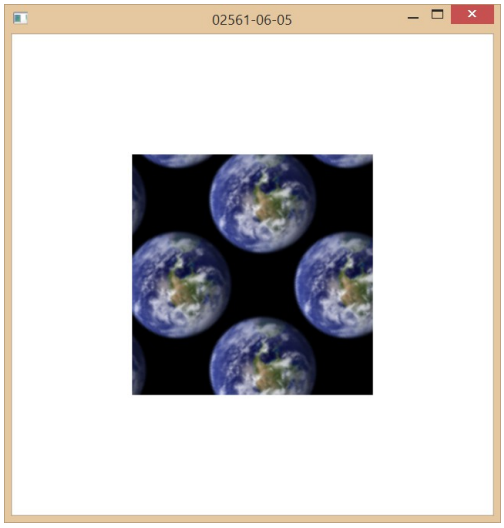
T Mirror, S Clamp.



T Mirror, S Mirror.

Part 5 – Texture coordinates transformation. Loading bmp images.

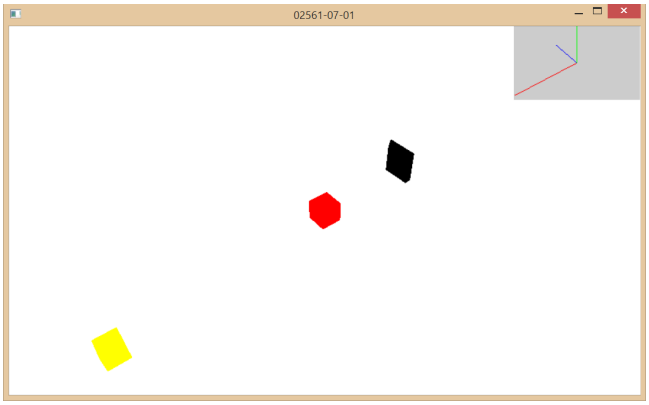
For this part I apply RotateZ, Translate and Scale transformations to the textureTrans matrix. The result is shown below:



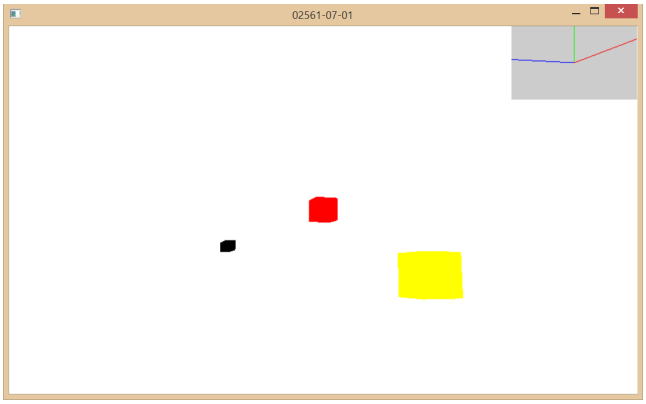
Exercise 7 – Shadows and render pipeline

Part 1 – Shadow projection

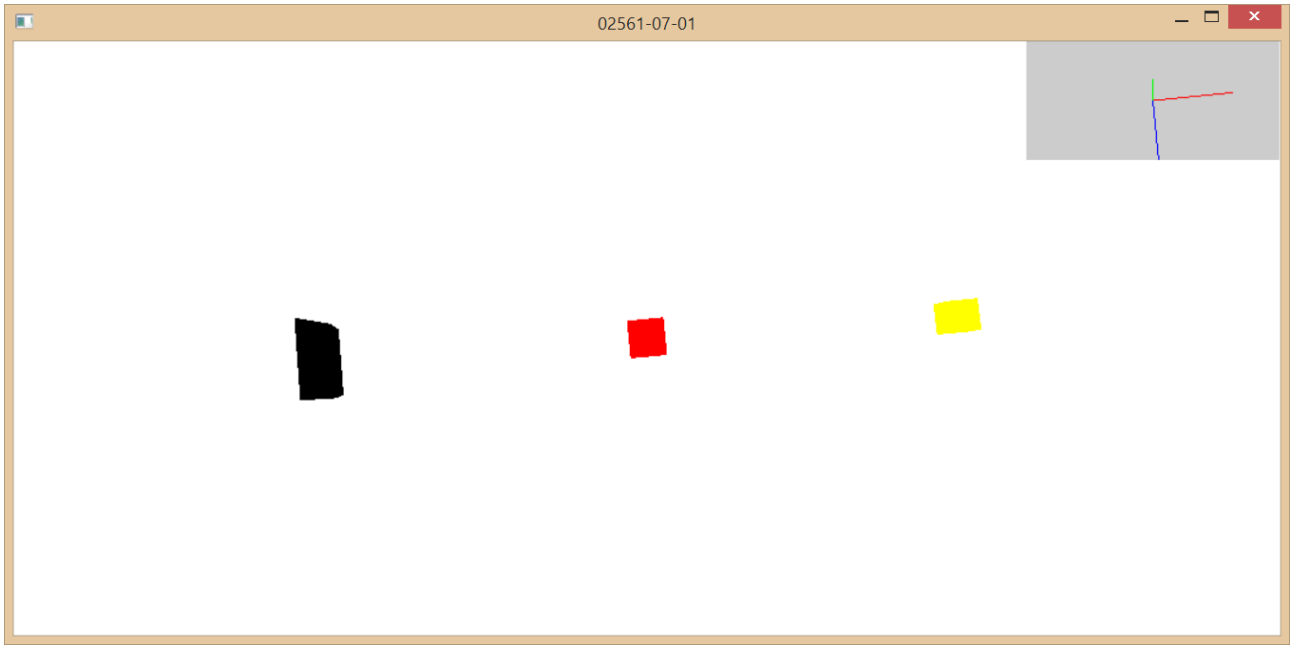
a) Point light.



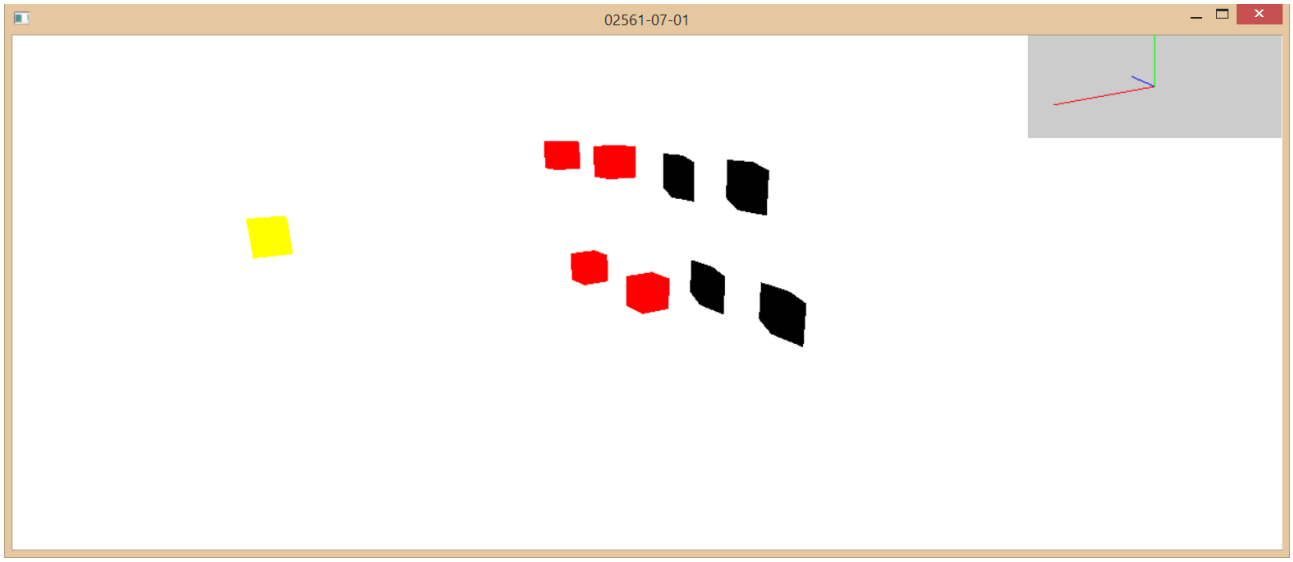
b) Directional light.
I did have trouble figuring this one out. I didn't end up with a correct result.



c) Changing Xw to -8 by pressing “d” and using point light we get a result like the one below:

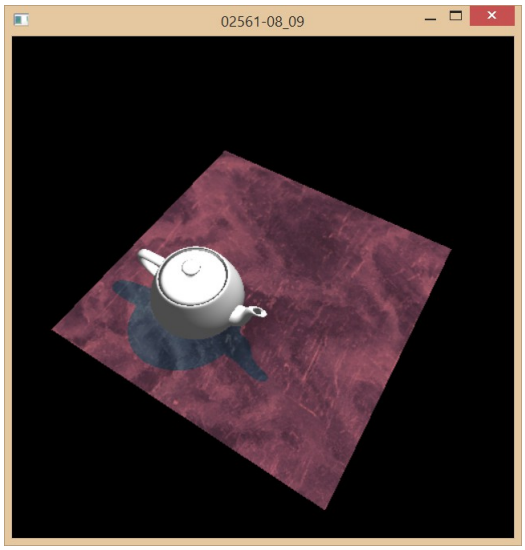


d) Using 4 cubes and point light:



Exercise 8 – Shadow Maps

Part 1,2,3 & 4
Teapot with shadow implementation:



Exercise 9 – Reflections

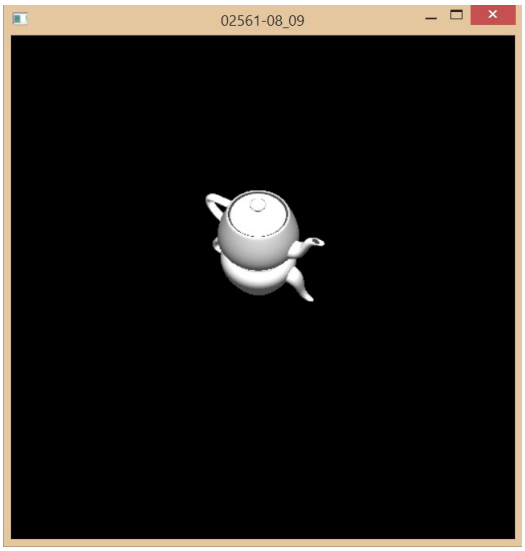
Part 1 – Introduction to stencil buffer

Explain what the setupStencil-function does and how it affects the rendering taking place after the function is called.

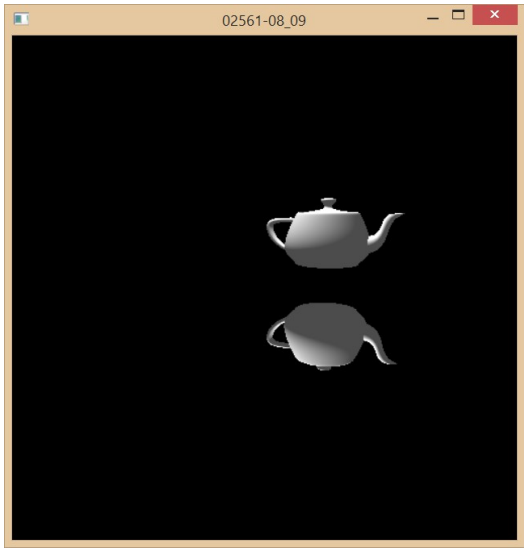
The setupStencil function setups the stencil buffer. The meshes that will be rendered inside this function will be used by the stencil buffer. The meshes tat will be rendered after this function in the display function will be rendered in the stencil buffer. Depending on how this buffer is setup, we might see some shapes on the screen, or parts of the shapes, or nothing.

Part 2 - Setup

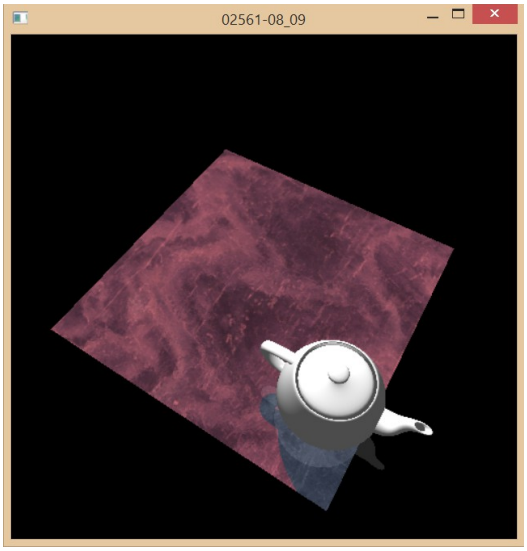
Part 3 – Reflect geometry



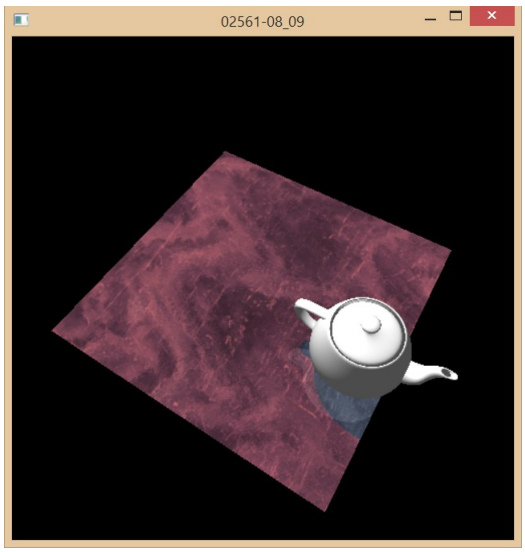
Part 4 – Reflect lighting



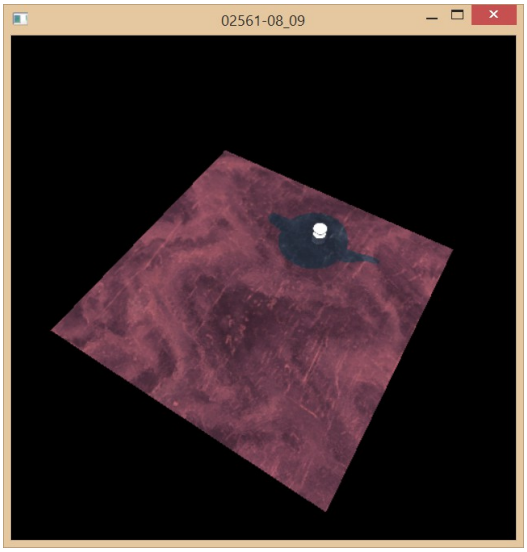
Part 5 - Blending



Part 6 – Stencil buffer

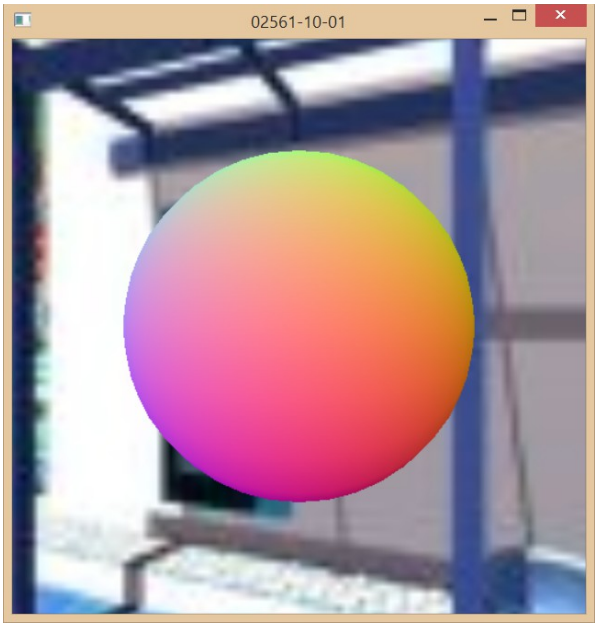


Part 7 – Clipping plane

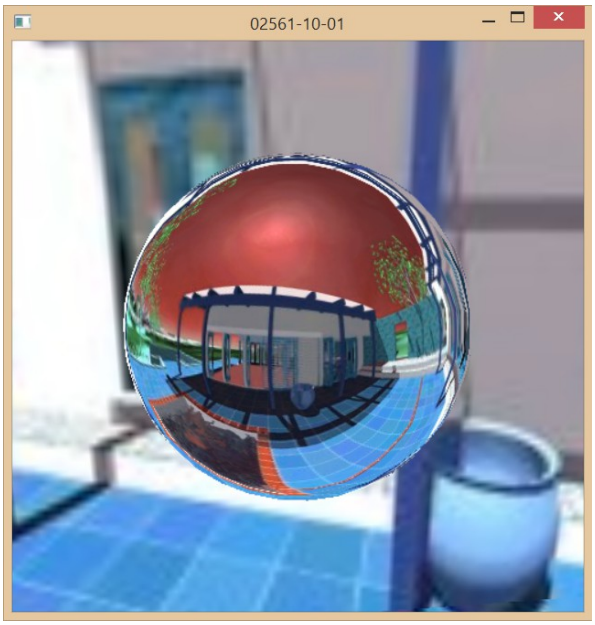


Exercise 10 – Environment Mapping and Bump Mapping

Part 1 – Loading cube map and create skybox shader



Part 2 – Creating a mirror shader.



Part 3 – Creating a glass shader

Part 4 – Creating a bump-map

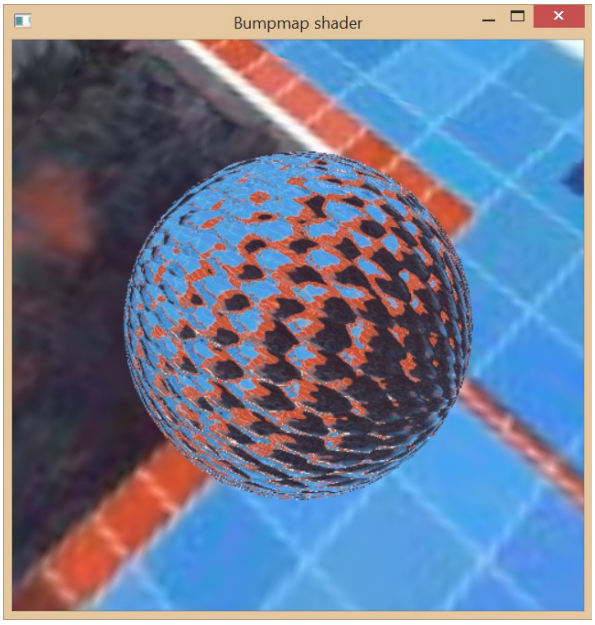
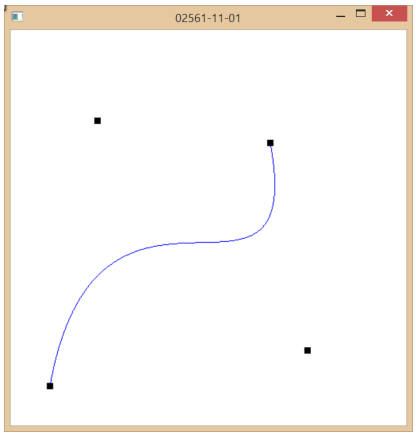


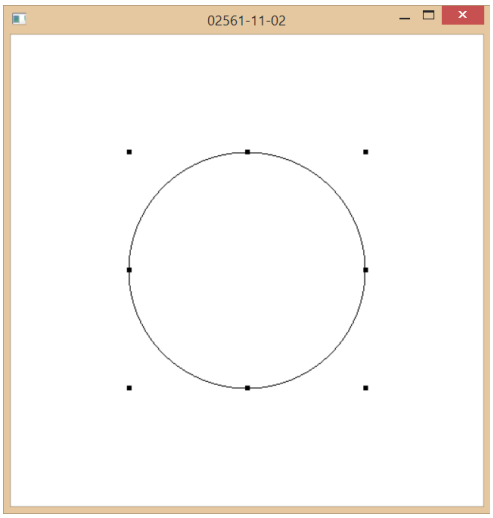
Figure 10-4: bump shader.

Exercise 11 – Geometry Modeling – Parametric Curves and Surfaces

Part 1 – Bezier curves

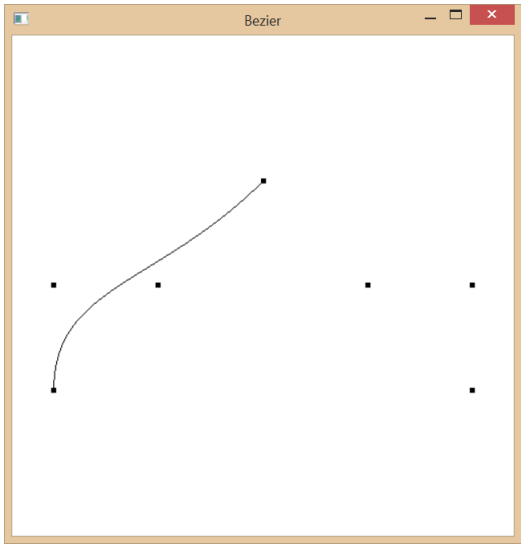


Part 2 – NURBS circle

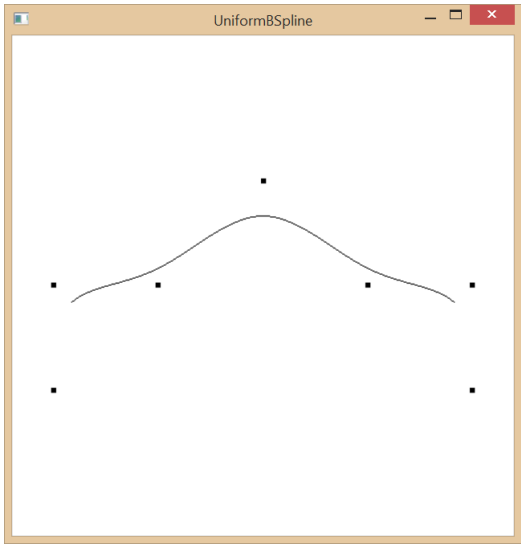


Part 3 – NURBS curves

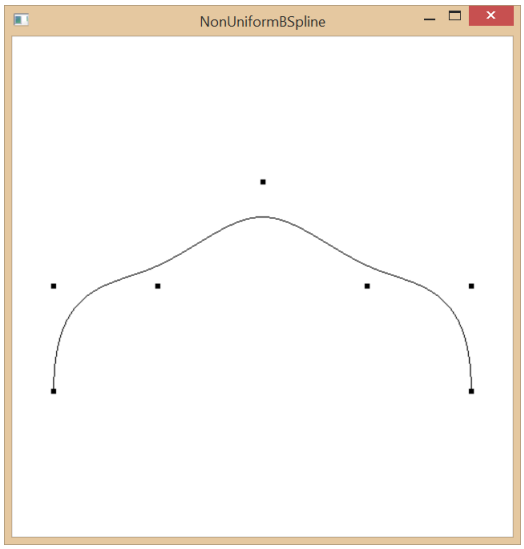
The Bezier curve:



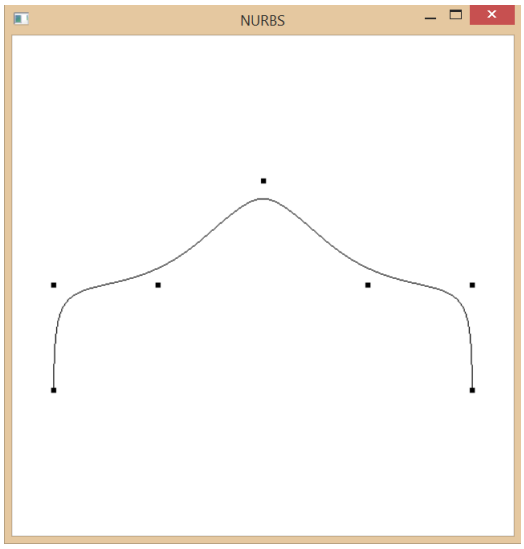
Uniform B-spline curve:



Non Uniform B-Spline:

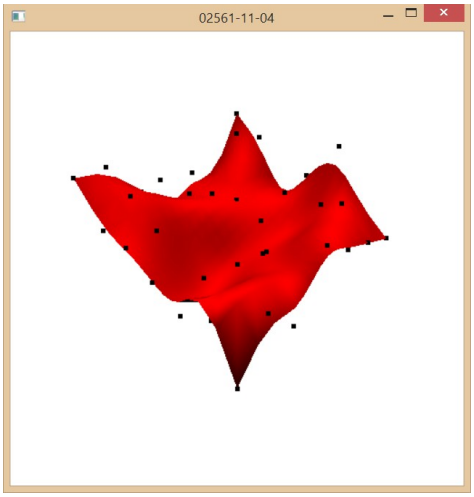


NURBS:



Part 4 – NURES surface patch

For this part I change the number of control points to 7 x 7 and I change the init_surface function to create a surface where the y position of each point is randomized, this way we can generate interesting surfaces in every execution.



Exercise 13 – Volume Visualization using Raycasting

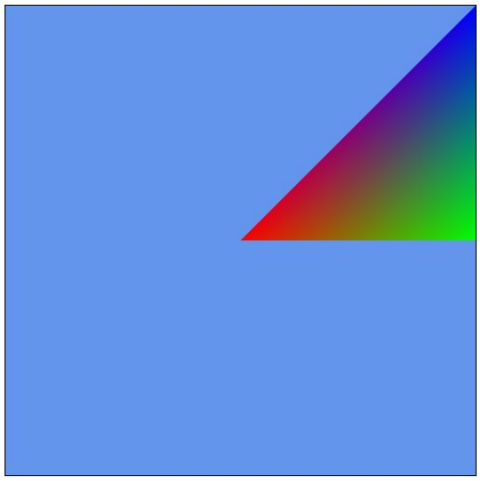
Unfortunately I couldn't render and validate the result, everytime I used the texture3D function in fragment shader the application start crashing without error messages. However I included the code in the handin.

Exercise 12 – WebGL Introduction

Part 1 & 2

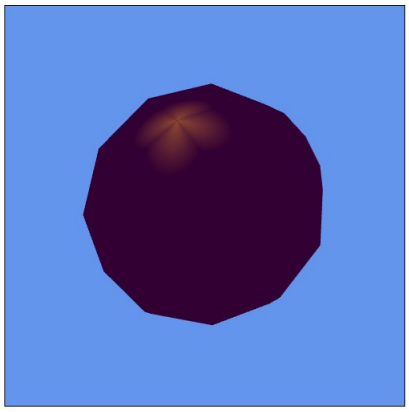
This part does not require any modification. It is intended for study.

Part 3

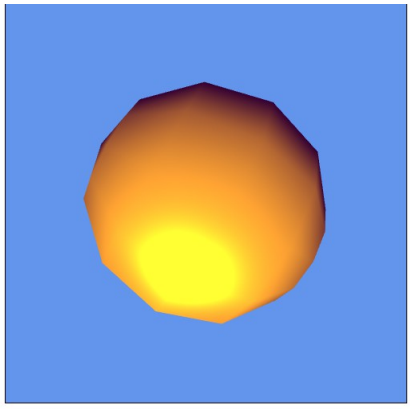


Part 4

Phong shading, pont light:



Phong shading, directional light:



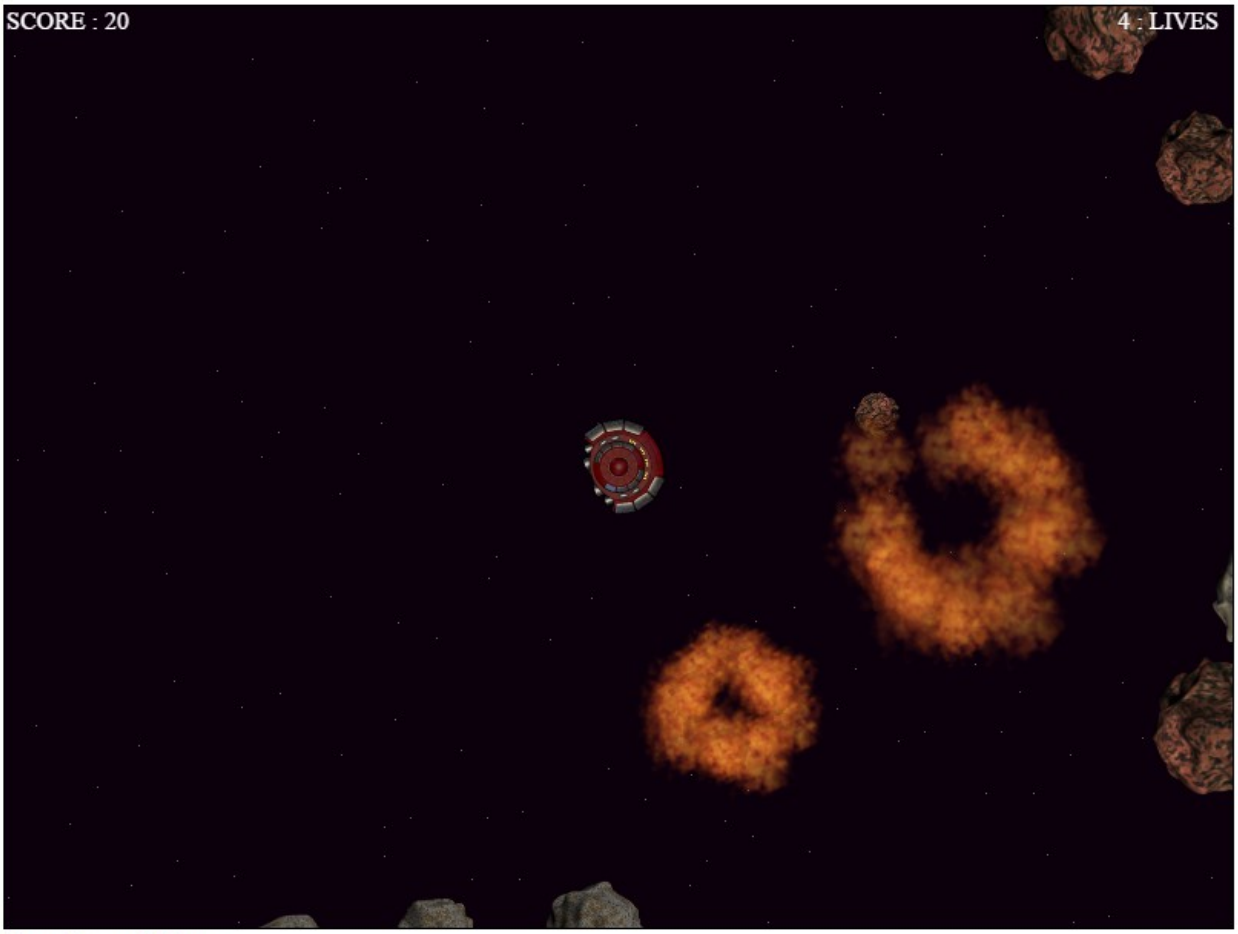
Part 5

I had some trouble creating the correct shading. This is what I ended up with using directional light:

Exercise 13 – WebGL – A simple game

Part 1, 2, 3, 4

Writing this report after I made this exercise, I only include a screen shot of the final result, and not the individual parts. The following screenshot includes something from all parts.



FrameTime: 4.616ms
Asteroid Count: 16
Shot Count: 0
Particle System: 60 particles active, 140 left in pool