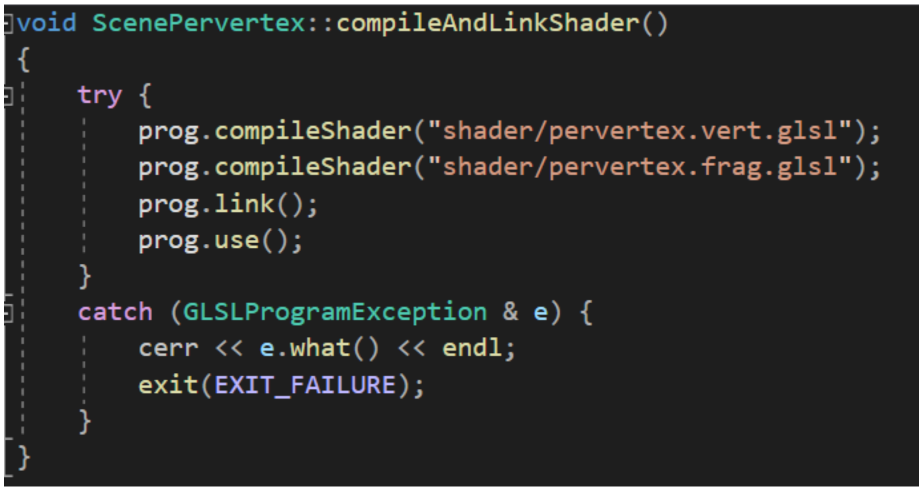
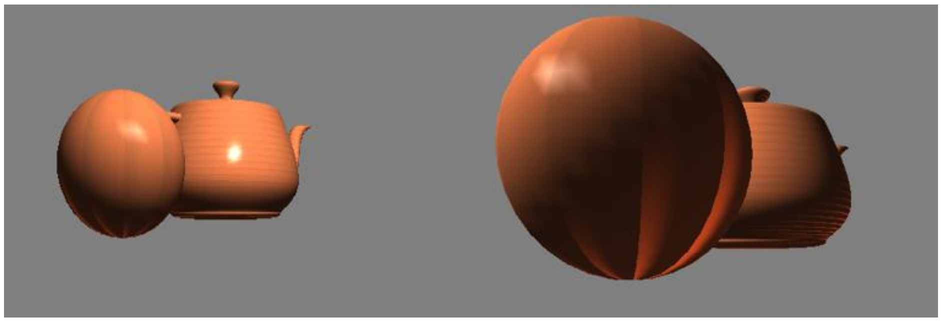
1. Setup a glsl program: create files for the shaders (at least vertex and fragment shaders) and load them at runtime of the program.

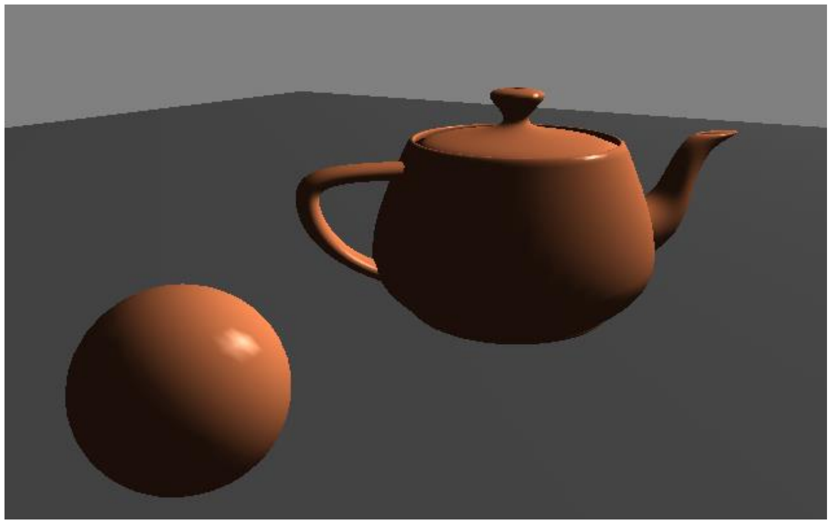


```
void ScenePervertex::compileAndLinkShader()
{
    try {
        prog.compileShader("shader/pervertex.vert.glsl");
        prog.compileShader("shader/pervertex.frag.glsl");
        prog.link();
        prog.use();
    }
    catch (GLSLProgramException & e) {
        cerr << e.what() << endl;
        exit(EXIT_FAILURE);
    }
}
```

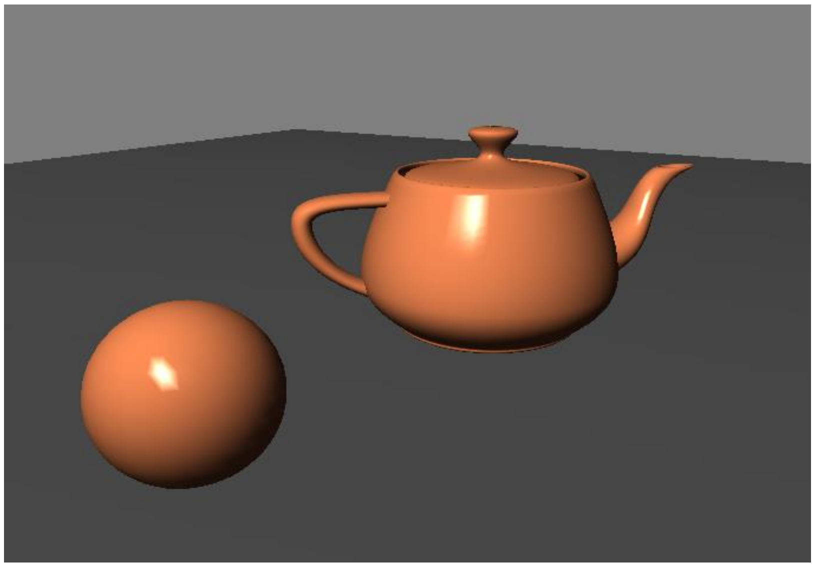
2. Set up camera and object transformations (at least rotation, and zoom for camera and translation for objects) that can be manipulated using the mouse and keyboard



3. Implement Phong lighting+Gouraud shading [1] (the Phong lighting model is evaluated in the vertex shader)

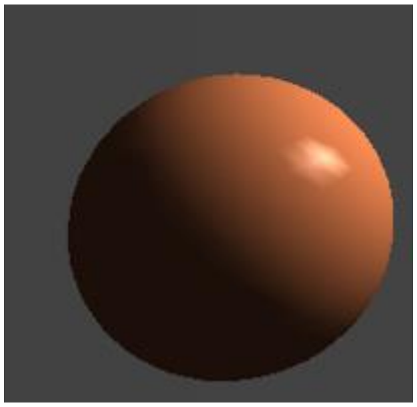


4. Implement Phong lighting+Phong shading [2] (the Phong lighting model is evaluated in the fragment shader, not in the vertex shader as for Gouraud shading).

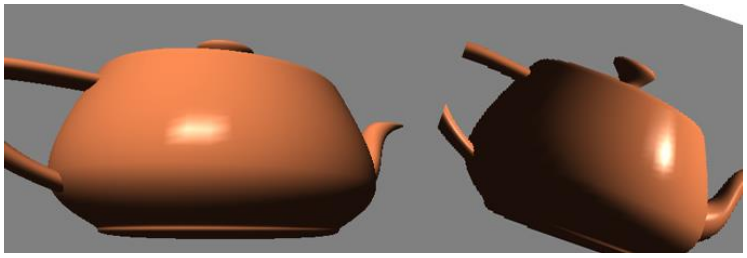


5. Implement a class that generates and stores the mesh geometry for a) a cylinder and b) a sphere. Have a look at how the geometry of the cube (VBOCube class) is specified and used.

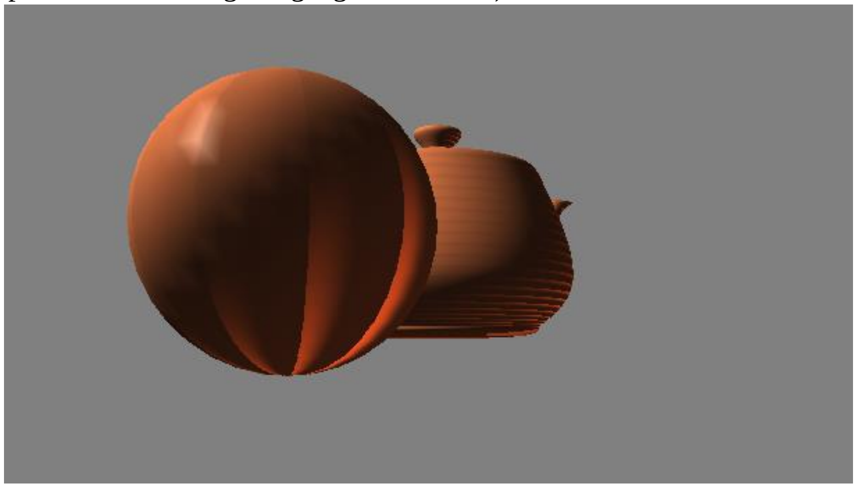
Teapot

Sphere:


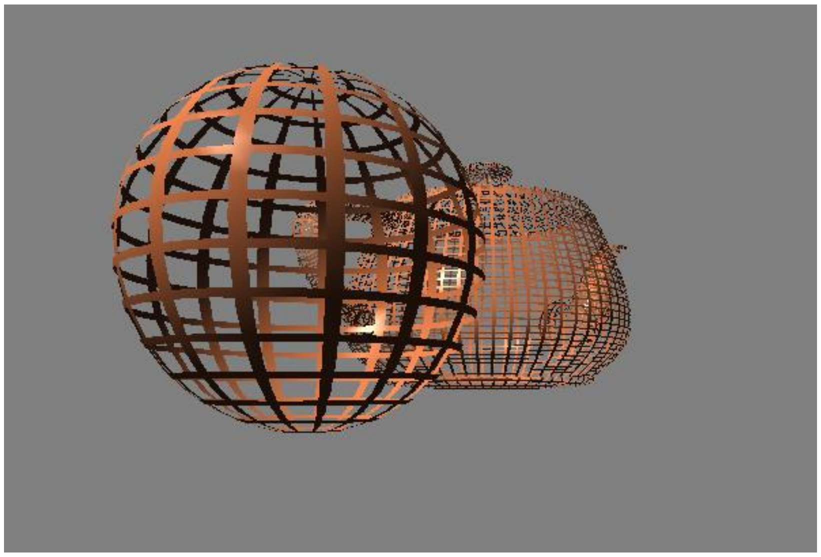
6. Render multiple instances of an object within one scene. Render the same object multiple times, applying different transformations to each instance. To achieve this you can set a different transformation matrix for each instance as a uniform variable in the vertex shader.



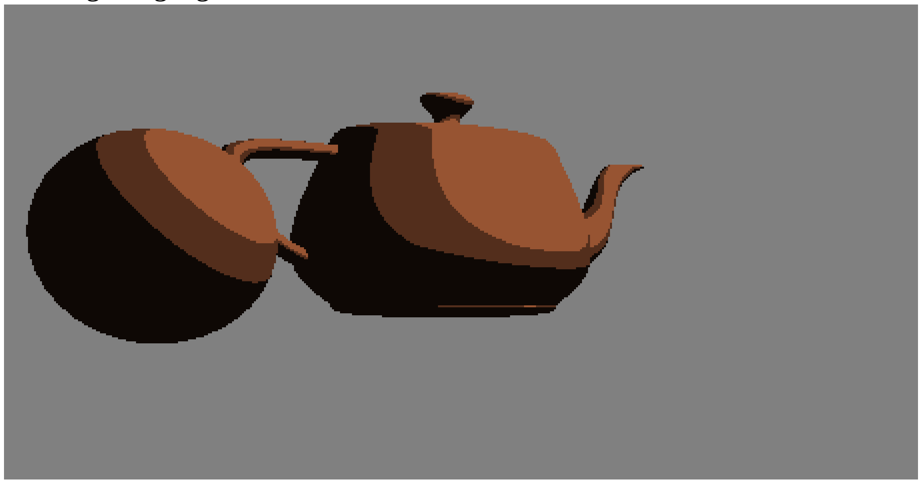
7. Perform different kinds of procedural shading (in the fragment shader): Implement the following procedural shaders
   - Stripes described in chapter 11.1 of the 'OpenGL Shading Language' book (this is not the 'OpenGL 4.0 Shading Language Cookbook')



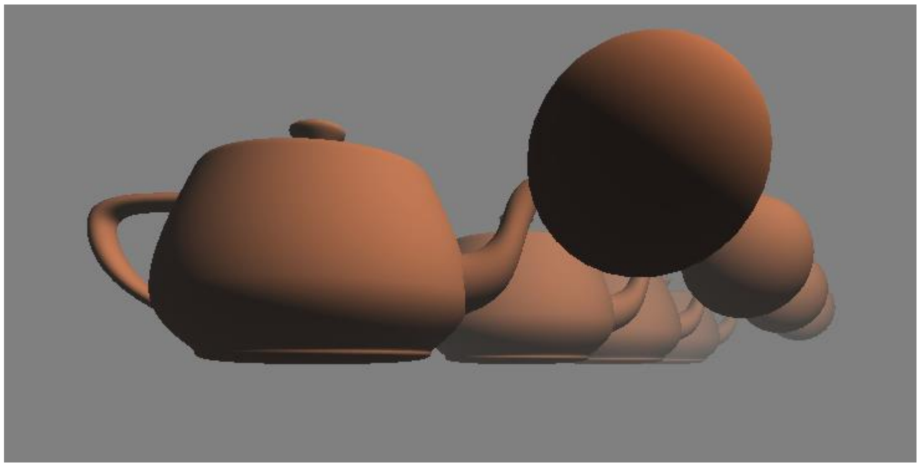
   - Lattice described in chapter 11.3 of the 'OpenGL Shading Language' book (this is not the 'OpenGL 4.0 Shading Language Cookbook')



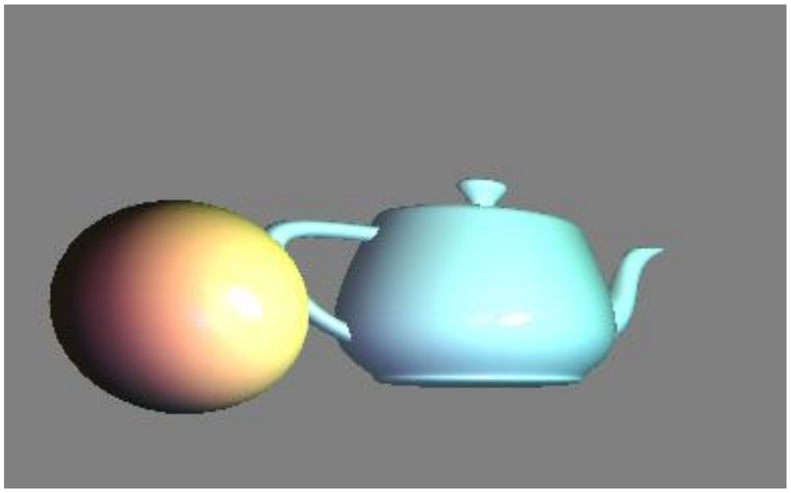
- Toon shading described in chapter 3 section 'Creating a cartoon shading effect' of the 'OpenGL 4.0 Shading Language Cookbook'

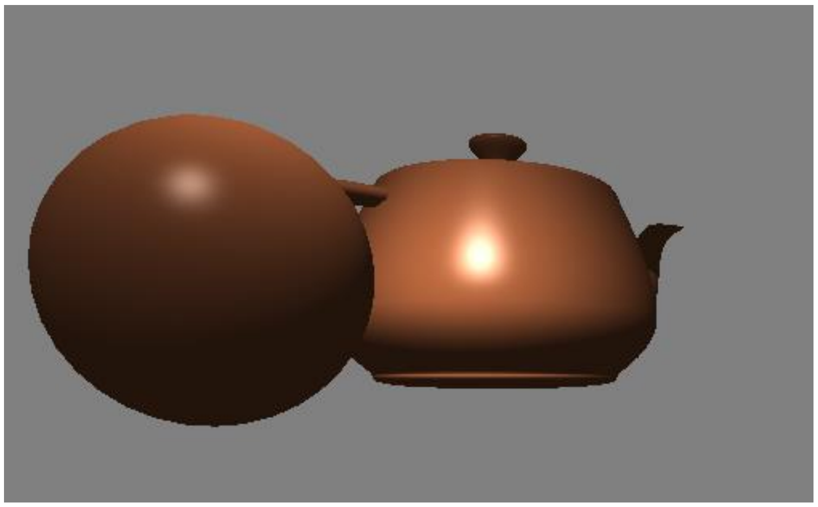


- Fog described in chapter 3 section 'Simulating fog' of the 'OpenGL 4.0 Shading Language Cookbook'



8. Provide key mappings to allow the user to switch between different kinds of shading methods and to set parameters for the lighting models.

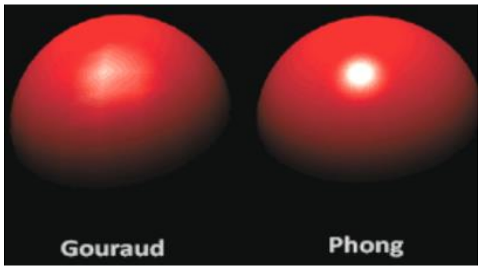
Multi-light: 

Spot-light: 

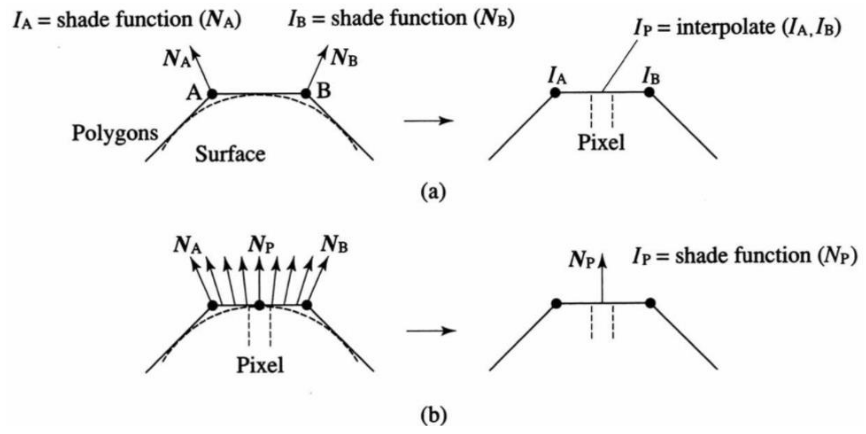
9. Submit your program and a report including the comparison of Phong and Gouraud shading and results of the different procedural shading methods.

Per-vertex versus per-fragment

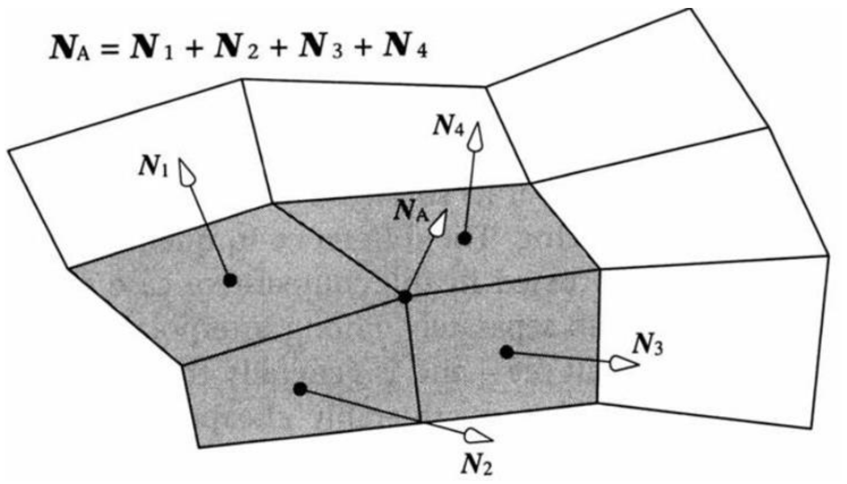
Since the shading equation is computed within the vertex shader, we refer to this as per-vertex shading. One of the disadvantages of this is that specular highlights can be warped or lost, due to the fact that the shading equation is not evaluated at each point across the face. For example, a specular highlight that should appear in the middle of a polygon might not appear at all when per-vertex shading is used, because of the fact that the shading equation is only computed at the vertices where the specular component is near zero. In the Using per-fragment shading for improved realism recipe Lighting, Shading, and Optimization, we'll look at the changes needed to move the shading computation into the fragment shader, producing more realistic results.



In the earlier days of lighting shaders, developers used to implement the Phong lighting model in the vertex shader. The advantage of doing lighting in the vertex shader is that it is a lot more efficient since there are generally a lot less vertices compared to fragments, so the (expensive) lighting calculations are done less frequently. However, the resulting color value in the vertex shader is the resulting lighting color of that vertex only and the color values of the surrounding fragments are then the result of interpolated lighting colors. The result was that the lighting was not very realistic unless large amounts of vertices were used: When the Phong lighting model is implemented in the vertex shader it is called Gouraud shading instead of Phong shading. Note that due to the interpolation the lighting looks somewhat off. The Phong shading gives much smoother lighting results. (a) Gouraud, (b) Phong shading.


I_A = shade function (N_A)
I_B = shade function (N_B)
N_A
N_B
A
B
Polygons
Surface
I_P = interpolate (I_A, I_B)
I_A
I_B
Pixel
(a)
N_A
N_P
N_B
Pixel
N_P
I_P = shade function (N_P)
(b)

1. Both are shading technology
2. Both are firstly use the normals of the polygons to calculate the normals of their common vertices


$\mathbf{N}_A = \mathbf{N}_1 + \mathbf{N}_2 + \mathbf{N}_3 + \mathbf{N}_4$
N_1
N_2
N_3
N_4
N_A

Bonus：

1. Implement (procedural) bump mapping (normal mapping) as in chapter 11.4 of the 'OpenGL Shading Language' book.