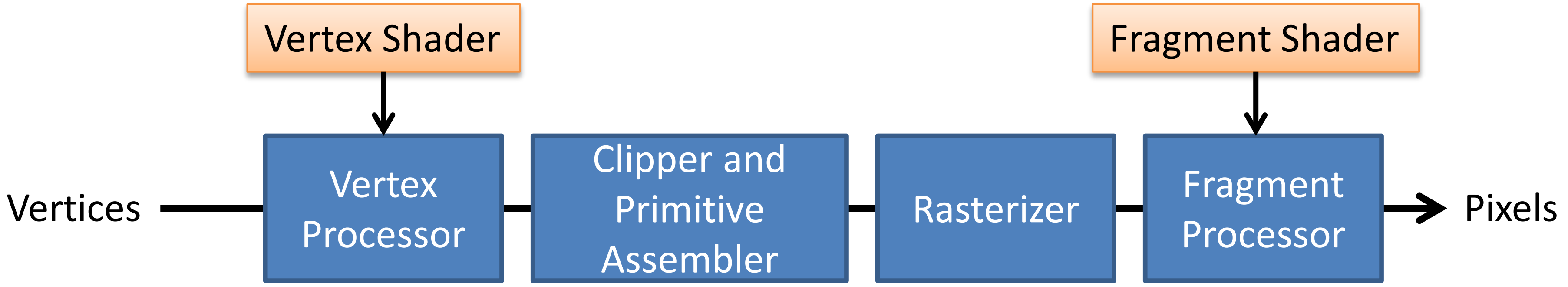
Computergrafik 1

Blatt 9

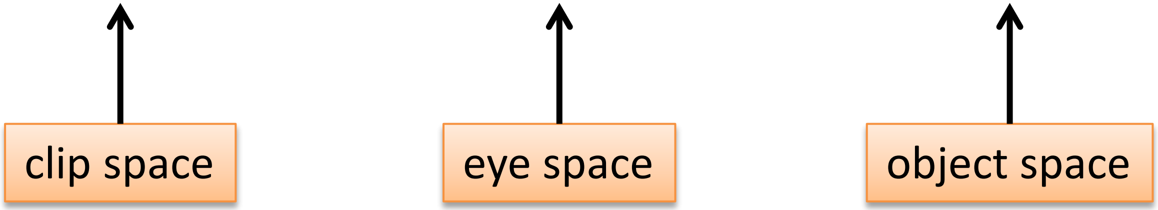
Introduction to GLSL

- OpenGL Shading Language
- Goal: accessible method for programming the GPU
- Allows to manipulate the rendering pipeline at vertex and fragment level



A very simple vertex shader

```
#ifdef GL_ES
precision highp float;
#endif

void main()
{
gl_Position = projectionMatrix * modelViewMatrix * vec4(position,1.0);
}
```



clip space

eye space

object space

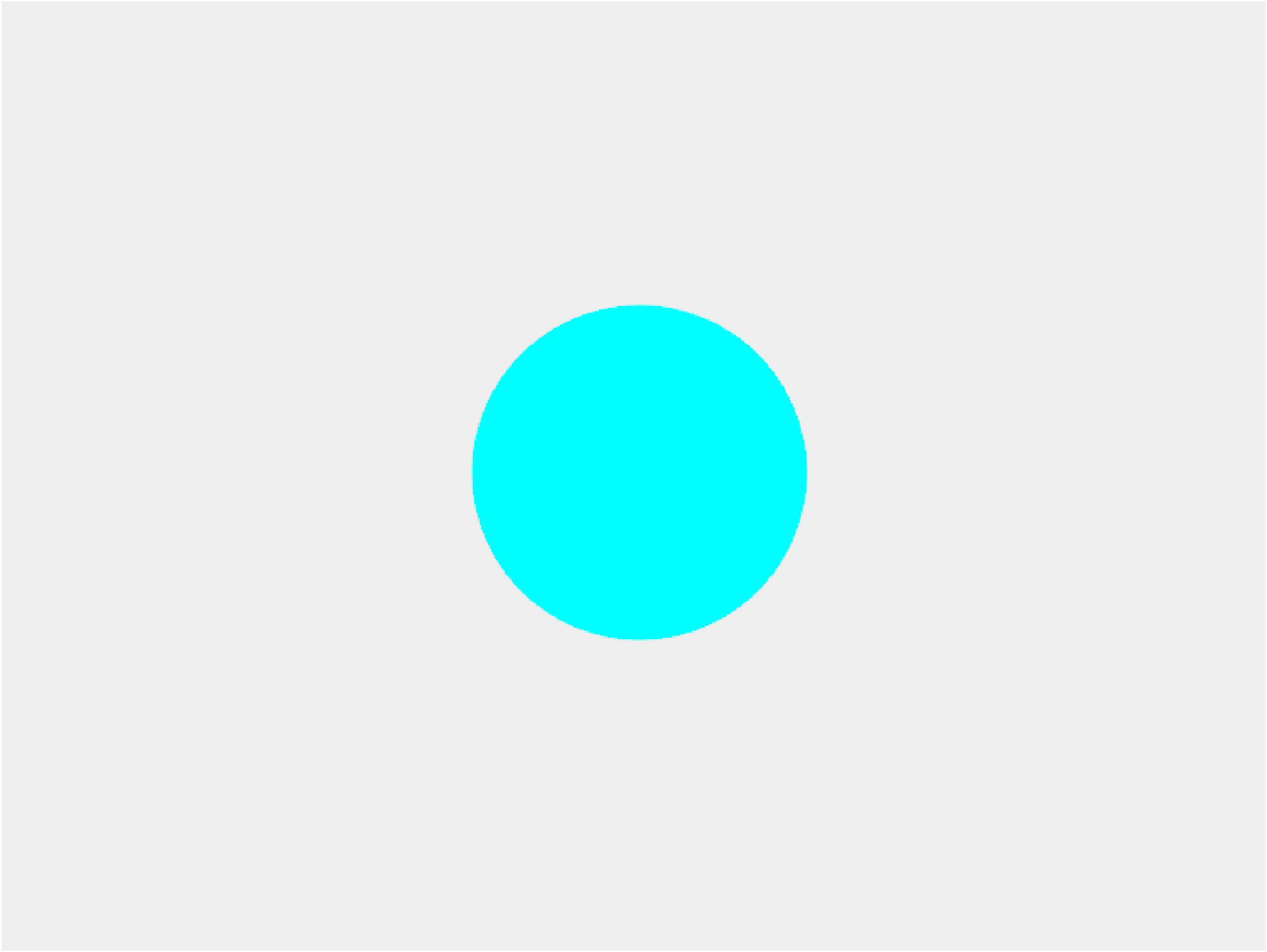
➡ This reproduces the standard rendering pipeline functionality

A very simple fragment shader

```
#ifdef GL_ES
precision highp float;
#endif

void main()
{
gl_FragColor = vec4(0.0,1.0,1.0,1.0);
}
```

➡ Each pixel is colored with the same color

Result



Important

- Shaders always replace the fixed functionality of the rendering pipeline
  - You cannot perform some tasks (e.g. texturing) using shaders and leave other tasks (e.g. additional coloring) for the fixed functionality

How can shaders be used?

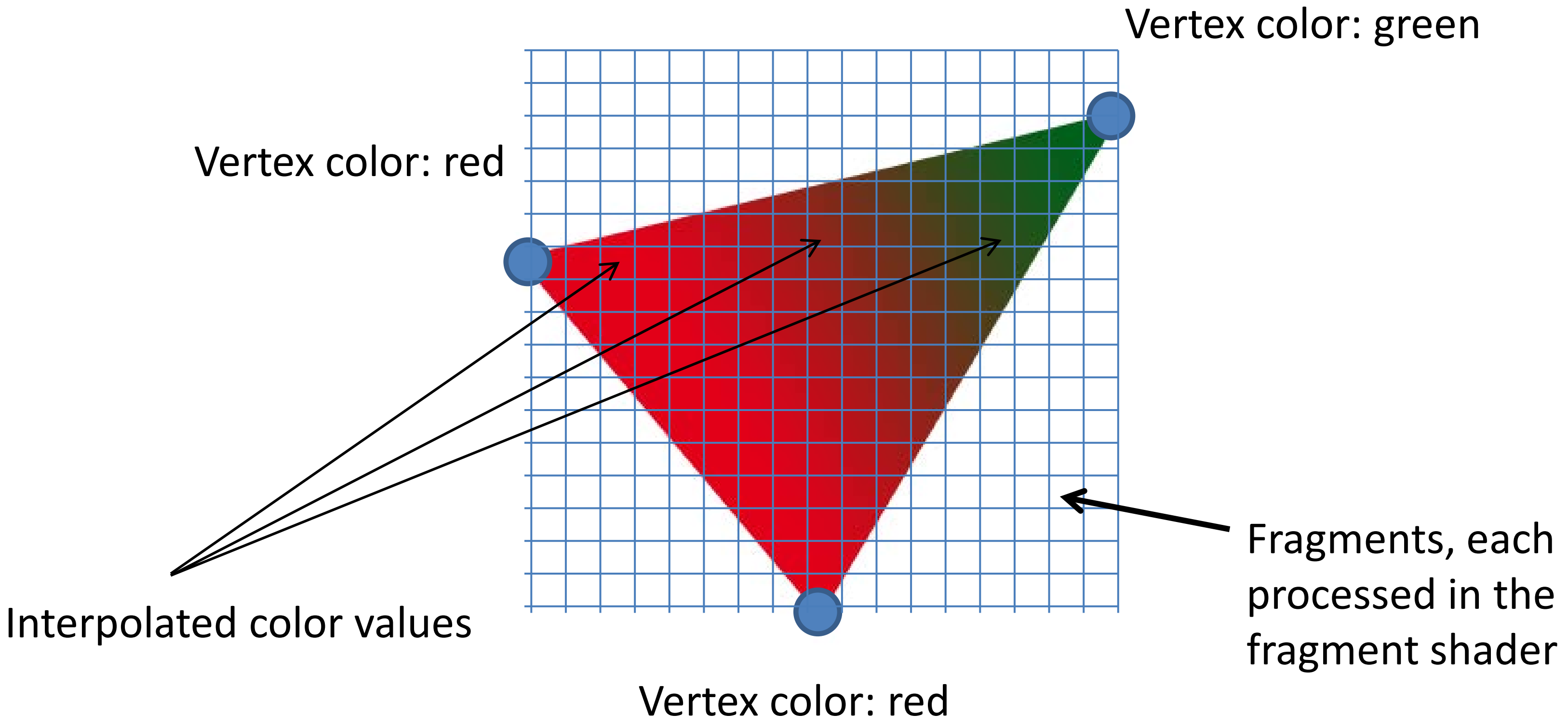
- Create a pair of vertex and fragment shaders and link them to your OpenGL-program
- Think of it as a kind of material
  - You can create multiple shaders
  - You can switch between fixed functionality and shaders (e.g. render one object using fixed functionality and another one using a shader)

Communication with shaders

- One-way: OpenGL -> Shaders
- Shaders have access to parts of the OpenGL state (e.g. light parameters)
- User defined variables
  - Uniforms
    - Constant for each frame -> not suitable for vertex attributes
    - Can be read (read-only) in vertex and fragment shader
  - Attributes
    - Can be updated anytime
    - Can be read only in vertex shader. Why?
  - Varyings
    - Allow access to interpolated vertex data (e.g. color)
    - Defined in the vertex shader and read in the fragment shader

Varying

In the vertex shader: `varying vec4 color = gl_Color;`



Vertex color: green

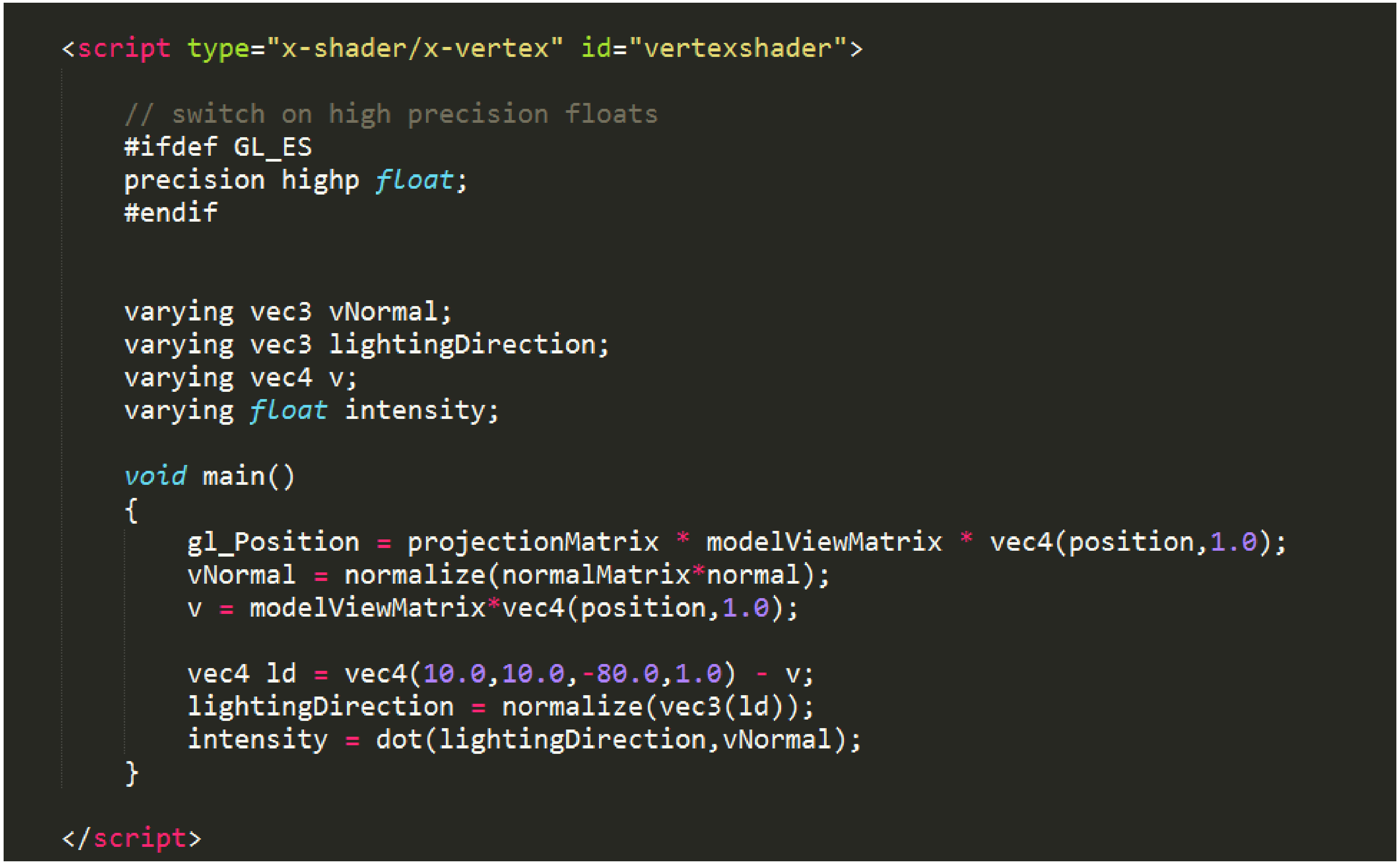
Vertex color: red

Fragments, each processed in the fragment shader

Interpolated color values

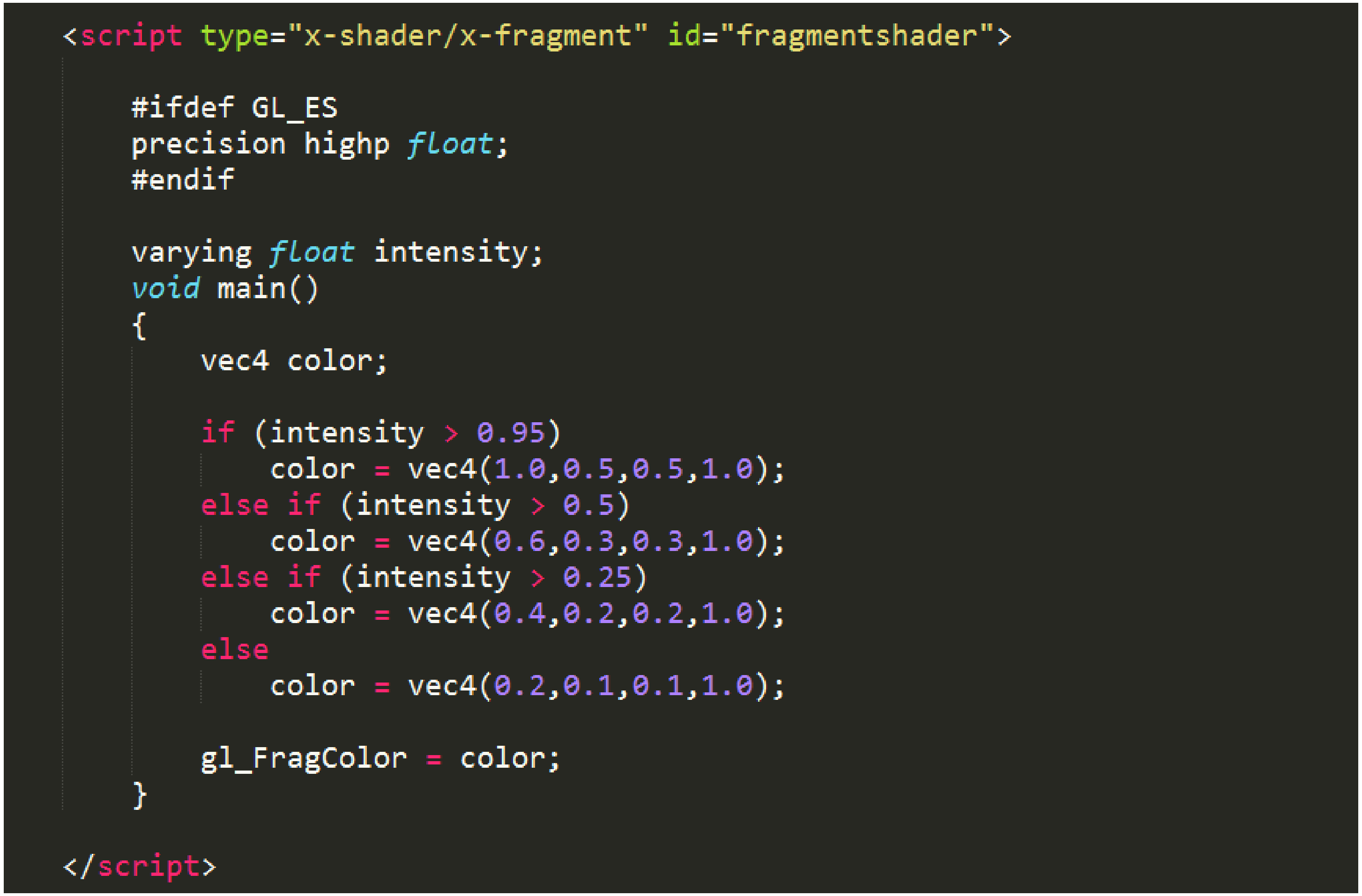
Vertex color: red

Toon Shading (Vertex Shader)



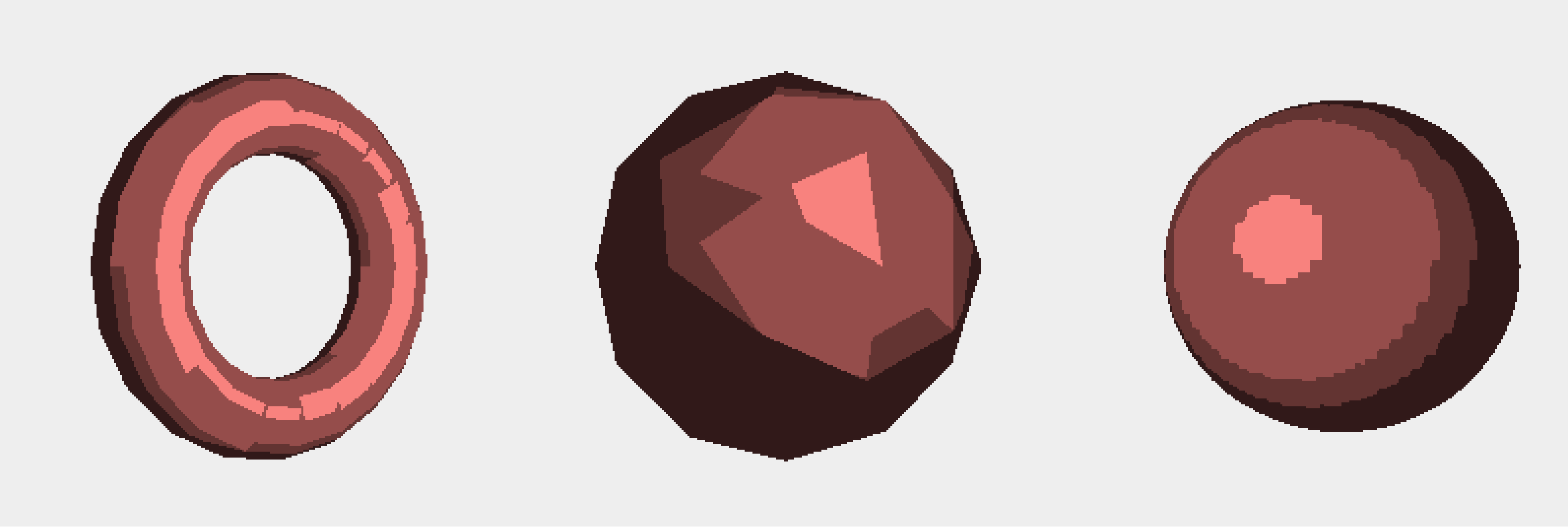
```
<script type="x-shader/x-vertex" id="vertexshader">

    // switch on high precision floats
    #ifdef GL_ES
    precision highp float;
    #endif


    varying vec3 vNormal;
    varying vec3 lightingDirection;
    varying vec4 v;
    varying float intensity;

    void main()
    {
        gl_Position = projectionMatrix * modelViewMatrix * vec4(position,1.0);
        vNormal = normalize(normalMatrix*normal);
        v = modelViewMatrix*vec4(position,1.0);

        vec4 ld = vec4(10.0,10.0,-80.0,1.0) - v;
        lightingDirection = normalize(vec3(ld));
        intensity = dot(lightingDirection,vNormal);
    }

</script>
```

Toon Shading (Fragment Shader)



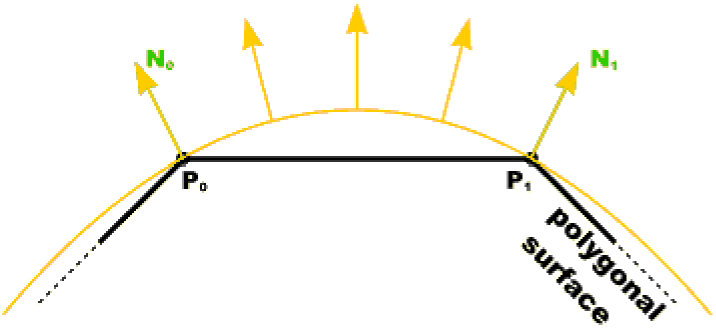
```
<script type="x-shader/x-fragment" id="fragmentshader">

    #ifdef GL_ES
    precision highp float;
    #endif

    varying float intensity;
    void main()
    {
        vec4 color;

        if (intensity > 0.95)
            color = vec4(1.0,0.5,0.5,1.0);
        else if (intensity > 0.5)
            color = vec4(0.6,0.3,0.3,1.0);
        else if (intensity > 0.25)
            color = vec4(0.4,0.2,0.2,1.0);
        else
            color = vec4(0.2,0.1,0.1,1.0);

        gl_FragColor = color;
    }

</script>
```

Result



Phong Shading

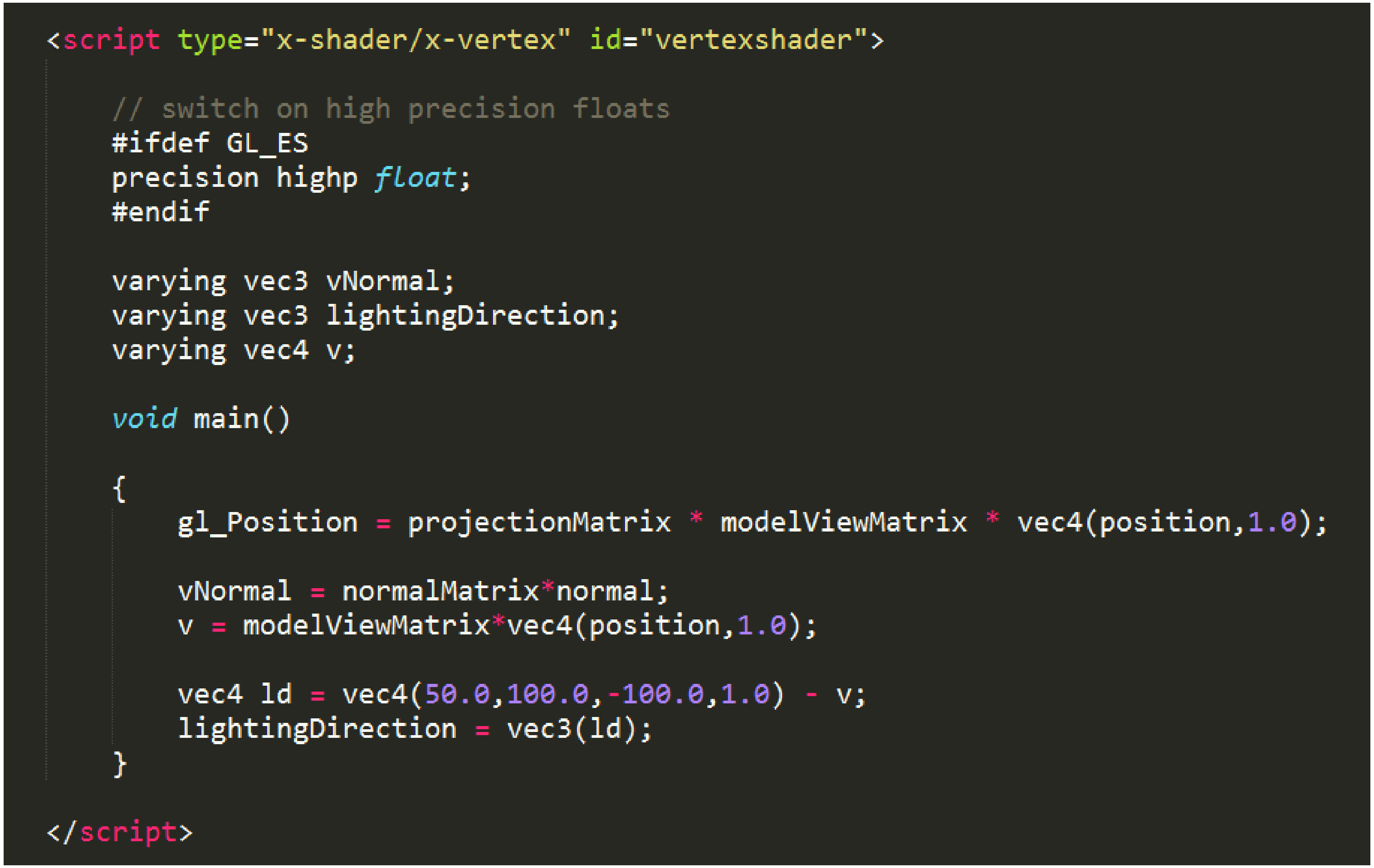
- Do not confuse with the Phong illumination model Idea:
- Compute color at each fragment, using
  - e.g. the Phong illumination model
  - interpolated vertex normals, camera and light position



```
varying vec3 normal;
```

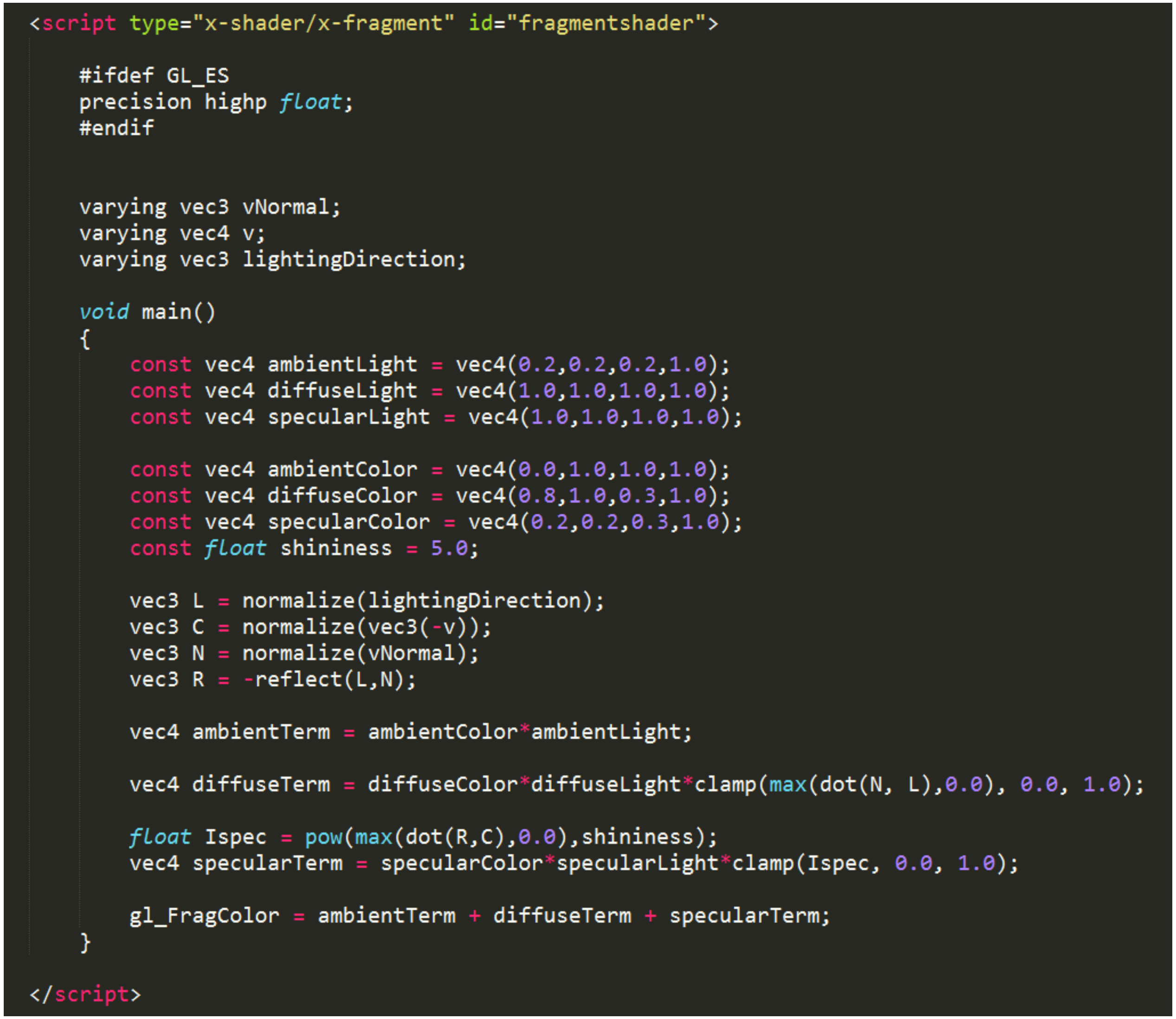
http://www.csee.umbc.edu/~rheingan/435/pages/res/illum/illum11b.gif

Phong Shading (Vertex Shader)



```
<script type="x-shader/x-vertex" id="vertexshader">

    // switch on high precision floats
    #ifdef GL_ES
    precision highp float;
    #endif

    varying vec3 vNormal;
    varying vec3 lightingDirection;
    varying vec4 v;

    void main()

    {
        gl_Position = projectionMatrix * modelViewMatrix * vec4(position,1.0);

        vNormal = normalMatrix*normal;
        v = modelViewMatrix*vec4(position,1.0);

        vec4 ld = vec4(50.0,100.0,-100.0,1.0) - v;
        lightingDirection = vec3(ld);
    }

</script>
```

Phong Shading (Fragment Shader)



```
<script type="x-shader/x-fragment" id="fragmentshader">

    #ifdef GL_ES
    precision highp float;
    #endif


    varying vec3 vNormal;
    varying vec4 v;
    varying vec3 lightingDirection;

    void main()
    {
        const vec4 ambientLight = vec4(0.2,0.2,0.2,1.0);
        const vec4 diffuseLight = vec4(1.0,1.0,1.0,1.0);
        const vec4 specularLight = vec4(1.0,1.0,1.0,1.0);

        const vec4 ambientColor = vec4(0.0,1.0,1.0,1.0);
        const vec4 diffuseColor = vec4(0.8,1.0,0.3,1.0);
        const vec4 specularColor = vec4(0.2,0.2,0.3,1.0);
        const float shininess = 5.0;

        vec3 L = normalize(lightingDirection);
        vec3 C = normalize(vec3(-v));
        vec3 N = normalize(vNormal);
        vec3 R = -reflect(L,N);

        vec4 ambientTerm = ambientColor*ambientLight;

        vec4 diffuseTerm = diffuseColor*diffuseLight*clamp(max(dot(N, L),0.0), 0.0, 1.0);

        float Ispec = pow(max(dot(R,C),0.0),shininess);
        vec4 specularTerm = specularColor*specularLight*clamp(Ispec, 0.0, 1.0);

        gl_FragColor = ambientTerm + diffuseTerm + specularTerm;
    }

</script>
```

Result