Shader programming

Learning Goals:
1. To understand the difference between using traditional rendering pipeline and using vertex and fragment shaders.
2. To understand what kind of effects a vertex and a fragment shader can achieve.
3. To be able to write a simple shader program.

Assignment:
You are given several examples of shaders.

Your task is to:
- Change the sphere in toon.frag to make it look like an eye. Change the colors to white, blue/brown and black on half of the sphere. Also try to modify the intensity values to make it look more like an eye.
- Play around with the light-parameters of the Blinn-phong shader and be able to explain what you have done.

Older graphics cards provide only a fixed-function pipeline in which lighting, texturing, and transformation is hardwired. The range of effects that can be achieved on such cards is limited. Newer cards (GeForce3/Radeon and later) have a programmable pipeline that allows C-like code to be downloaded onto the card and executed for each vertex and pixel. The latest cards are programmable-only and have no dedicated circuits for the old fixed-function routines. Instead they emulate the fixed-function pipeline (which is slowly disappearing from OpenGL and DirectX altogether!). The programs that execute on the graphics card are called Vertex and Fragment (or pixel) shaders.

There are several shader languages, all rather similar to traditional ‘c’-code;
- Cg (Specific for Nvidia-cards)
- HLSL - High Level Shading Language (Microsoft)
- GLSL - OpenGL Shading Language (Used in VRT)
Some examples of features that can be achieved through shaders in real-time rendering applications:

- Realistic materials
- Lightning effects
- Fire, smoke, water, clouds
- Painting-effects
- Texture effects using normals, etc.
- Non-static texturing

Vertex processing involves operations that occur at each vertex, such as lightning and transformation. This kind of processing is performed before the 3D-data is transformed into the screen-space data that will be shown as a frame in the scene (rasterization). Fragment processing, on the other hand, requires a rasterized image and involves operations on pixel-level, such as texturing.
### Vertex shader

**Vertex processors**

- Transform
- Animation
- Per-vertex lighting
- Displacement
- Mostly: Parameter setup for fragments
- Cannot modify topology
- No access to other vertices
The following piece of code (toon.vert) is an example of a simple vertex shader. The code is executed for each vertex of the geometry to which the shader is applied.

```glsl
varying vec3 normal, lightDir;

void main()
{
    lightDir = normalize(vec3(gl_LightSource[0].position));
    normal = normalize(gl_NormalMatrix * gl_Normal);

    gl_Position = ftransform();
}
```

**Fragment shader**

- Per-pixel lighting
- Texturing
- Compositing
- Filtering
- Fog
- Cannot read or write pixels
- No access to other fragments

The following piece of code (toon.frag) is an example of a fragment shader. The code is executed for each fragment (~pixel) of the rasterized scene.

```glsl
varying vec3 normal, lightDir;

void main()
{
    float intensity;
    vec3 n;
    vec4 color;

    n = normalize(normal);
    intensity = max(dot(lightDir,n),0.0);

    if (intensity > 0.98)
        color = vec4(0.8,0.8,0.8,1.0);
    else if (intensity > 0.5)
        color = vec4(0.4,0.4,0.8,1.0);
    else if (intensity > 0.25)
        color = vec4(0.2,0.9,0.4,1.0);
    else
        color = vec4(0.1,0.1,0.1,1.0);

    gl_FragColor = color;
}
```
**Shaders in VRT**

VRT uses the OpenGL-based GLSL-language.  
[http://www.opengl.org/documentation/glsl/](http://www.opengl.org/documentation/glsl/)

There are currently three VRT-functions related to shader-programming. First you need to initialize shader functionality using:

```c
void VRT_InitShader()
```

A shader can then be applied to a node in the scene-graph. You need to supply a VRT_Node, the filename of a vertex shader (.vert), and a filename of a fragment shader (.frag):

```c
int VRT_NodeSetShader(VRT_Node *node,  
                      char *vertex_file,  
                      char *fragment_file)
```

Shaders can also be removed with this function:

```c
int VRT_NodeRemoveShader(VRT_Node *node)
```