Chapter 6
Fragment shader
The inputs to the fragment shader

- **Varyings:** The per-vertex output variables produced by the vertex shader are interpolated to determine the per-fragment ones.
- **Uniforms** to be used by the fragment shader.
- **Samplers** = Textures.
- The fragment color stored in `gl_FragColor` is passed to the output merger.
A simplest fragment shader

```glsl
uniform sampler2D s_tex0;
varying vec2 v_texCoord;

void main() {
    gl_FragColor = texture2D(s_tex0, v_texCoord);
}
```

- The above fragment shader declares a uniform variable, `s_tex0`, of type `sampler2D`, which represents a texture.
- Recall that our vertex shader simply output 2D texture coordinates without modification. The per-vertex texture coordinates were interpolated so that the per-fragment texture coordinates, `v_texCoord`, are now passed to the fragment shader and used for fetching the texture, `s_tex0`.
- The fragment shader invokes the built-in function `texture2D`, which returns a color. It is output to the built-in variable `gl_FragColor`. 
Phong Lighting Model

- Illumination or lighting refers to the techniques handling the interaction between light sources and objects.
- In real-time applications, the most popular illumination method is based on the Phong model. Even though it is not physically correct, it is widely adopted in commercial games and lays foundations of various advanced lighting techniques.
Among various light sources (including point, area, spot, and directional light sources), for now let’s take the simplest, the directional light source, where the light vector \( l \) connecting a surface point and the faraway light source is constant for the entire scene.

The Phong model is composed of diffuse, specular, ambient, and emissive terms. For the sake of simplicity, let us consider the diffuse term only.

The diffuse term is based on Lambert’s law. Reflections from ideally diffuse surfaces (Lambertian surfaces) are scattered with equal intensity in all directions. The amount of reflection perceived by the eye is just proportional to the amount of incoming light, which is inversely proportional to the angle between \( l \) and the surface normal \( n \). It is described as \( n \cdot l \). In order to avoid negative reflection, it is modified into \( \max(n \cdot l, 0) \).
Suppose a white light $(1,1,1)$. If an object lit by the light appears yellow, it means that the object reflects R and G and absorbs B. We can easily implement this kind of filtering through material parameter, i.e., if it is $(1,1,0)$, then $(1,1,1) \otimes (1,1,0) = (1,1,0)$ where $\otimes$ is component-wise multiplication.

The diffuse term combines the intensity and color: $\max(n \cdot l, 0) \ s_d \otimes m_d$ where $s_d$ is the light source color and $m_d$ is the object material color.

The texture provides $m_d$ whereas $s_d$ is given by the user.
Let us show how to implement Phong lighting (with the diffuse term only) and texturing. The vertex shader shown below computes the clip-space vertex position, as usual, and also the ‘world-space’ vertex position and normal of each vertex. In addition, it bypasses the texture coordinates. The output variables \( v\text{\_}position, v\text{\_}normal, \) and \( v\text{\_}texCoord \) will be interpolated for fragments and they will be provided for the fragment shader.

```glsl
uniform mat4 viewProjMat;
uniform mat4 worldMat;

attribute vec3 position;
attribute vec3 normal;
attribute vec2 texCoord;

varying vec3 v_position;
varying vec3 v_normal;
varying vec2 v_texCoord;

void main() {
    gl_Position = viewProjMat * worldMat * vec4(position, 1.0);
    v_position = (worldMat * vec4(position, 1.0)).xyz;
    v_normal = mat3(worldMat) * normal;
    v_texCoord = texCoord;
}
```
The fragment shader shown below obtains the object material color \((m_d)\) by filtering the texture and then asks the user-defined function `phongDiffuse` to compute the diffuse term. It uses `uniform vec3 lightPos` for calculating the light vector \(\mathbf{l} (\text{lightDirection})\) and simply implements \(\max(n \cdot l, 0) \, s_d \otimes m_d\).