In this chapter we will cover two topics that are fundamental to producing realistic scenes, texture and lighting. Specifically, we will

- discuss what textures are and how to apply them
- show what texture options are available and how to configure these
- use multiple textures in a shader
- present a basic lighting model
- create a directional light shader

By the end of the chapter, we will produce the textured and lit mesh on the right of Figure 3-1.

Figure 3-1. Left - No texture or lighting; Right - both texture and lighting

The left image in Figure 3-1 is a concrete example of why we need to use texture and lighting. In the last example of Chapter 1, a triangle mesh was visible as a 3D figure. The reason it appeared three-dimensional was only because the vertex colors were distinct and interpolated by our fragment shader. This provided depth cues for us. As you can see, when all the vertex points have the same color, and no lighting or texture is applied, the image looks like a flat two-dimensional polygon. It is actually still 3D; the reason that it appears flat is that there are no context clues to let us know that this is in fact a solid figure.
When we look at an image, we depend on clues such as variance on the faces of a solid in terms of lighting: darkness/illumination, reflection, shadows, and directional pattern changes from textures to inform us where one face ends and another begins. In the image on the right of Figure 3-1, we have added texture and lighting clues, and you can clearly tell that this is a solid.

Textures

Textures are images that are applied to surfaces within our program. Images used as textures may be bitmapped in origin or generated procedurally. Textures must be applied (mapped) to our image and in doing so are usually stretched, scaled, distorted, and/or repeated.

The width and height of a texture are usually the same and a power of 2, $2^n$, such as 64, 128, 256, and 512. Each basic element of a texture is known as a texel which stands for texture element or texture pixel.

Texture Coordinates

In two dimensions, texture coordinates are referred to in $(s, t)$ pairs instead of $(x, y)$ pairs like vertex positions. Normally, texture coordinates are also limited to the range $(0, 0)$ to $(1, 1)$. For a texture size of 128x128 pixels, all points will be divided by 128 in order to lie within this range. The texture coordinate $(0.5, 0.25)$ for a 128x128 texture would refer to the texel $(64, 32)$.

Figure 3-2 shows the coordinates of a source image on the left and the equivalent texture coordinates on the right.

![Figure 3-2. Left - a square 128x128 pixel image with vertex coordinates; Right - the equivalent texture coordinates](image)

Texture coordinates are usually sent to the shader program as vertex attribute values, but (as we saw in the previous chapter) we can also manipulate them within our shader program.

Texture Objects

In WebGL, a texture is stored within a WebGLTexture object. To create and bind a WebGLTexture object, the API functions used are:

```javascript
WebGLTexture createTexture();
void bindTexture(GLenum target, WebGLTexture texture);
```

The target for 2D textures will be TEXTURE_2D. Other target types are listed in Appendix C.

The code to create and bind a WebGLTexture will look like this:

```javascript
var texture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, texture);
```

To check that a certain texture has loaded properly, you can use the API call: