CHAPTER 10

Effects, Tips, and Tricks

In this chapter, we will introduce a variety of WebGL effects, tips, and tricks such as these:

- Basic image processing
- Image processing using convolution filters
- Antialiasing
- Nonphotorealistic shaders
- Framebuffers and renderbuffers
- Picking objects from the canvas
- Shadow map implementation

Effects

A wide variety of effects can be achieved through image-processing and convolution filters such as sharpening, blurring, grayscale, sepia tone, color adjustments, and edge detection.

To apply these effects, we will start by loading a texture image. Then we will alter the raw color values at each pixel in the texture within the fragment shader. For these examples, the setup is similar to some of the Chapter 6 examples in which algorithms were used to create images purely within the fragment shader. This time around, we have a starting texture image to alter. In practice, the texture image could be from a HTMLVideoElement object, so we could alter streaming video on the fly using these same techniques. We will concentrate on static image processing.

Basic Image Manipulation

Our first example of image manipulation will show grayscale, inverted color values and a green tinted image next to the original texture image. We do this by first setting some effect constants and a variable to store a uniform value that will inform our shader which effect to use:

```javascript
var NO_EFFECT = 0,
GRAYSCALE_EFFECT = 1,
NEGATIVE_EFFECT = 2,
GREEN_TINT_EFFECT = 3;
var effectUniform = null;
```

When we render to the canvas, we will actually draw our scene four times, using a quarter of the viewport and changing the effect each time. The rendering is shown in Figure 10-1. Unfortunately, it is hard
to see any difference in black and white print, so please visit the site http://www.beginningwebgl.com/ for a full colour version.

![Image of four variations of a child in a sled: original, grayscale, inverted colors, and tinted more green.](image)

**Figure 10-1.** Top left: original image; top right: grayscale; bottom right: inverted colors; bottom left: tinted more green

Adjusting the viewport and rerendering allows us to easily view several variations at once and is an application of using multiple viewports in the same scene as discussed in Chapter 1. The code for the viewport setup is shown in Listing 10-1. In it, we draw four times to different areas of the viewport and inform the fragment shader which effect to apply each time by changing the uniform value.

### Listing 10-1. Code for the viewport setup

```javascript
//top left
gl.uniform1i(effectUniform, NO_EFFECT);
gl.viewport(0, canvas.height/2.0, canvas.height/2.0, canvas.height/2.0);
drawScene();

//bottom left
gl.uniform1i(effectUniform, GREEN_TINT_EFFECT);
gl.viewport(0, 0, canvas.height/2.0, canvas.height/2.0);
drawScene();
```