Chapter 10

Advanced Techniques

This chapter includes a "grab-bag" of interesting techniques that you should find useful for creating your WebGL applications. The techniques are mostly stand-alone, so you can select and read any section that interests you. Where there are dependencies, they are clearly identified. The explanations in this chapter are terse in order to include as many techniques as possible. However, the sample programs on the website include comprehensive comments, so please refer to them as well.

Rotate an Object with the Mouse

When creating WebGL applications, sometimes you want users to be able to control 3D objects with the mouse. In this section, you construct a sample program `RotateObject`, which allows users to rotate a cube by dragging it with the mouse. To make the program simple, it uses a cube, but the basic method is applicable to any object. Figure 10.1 shows a screen shot of the cube that has a texture image mapped onto it.

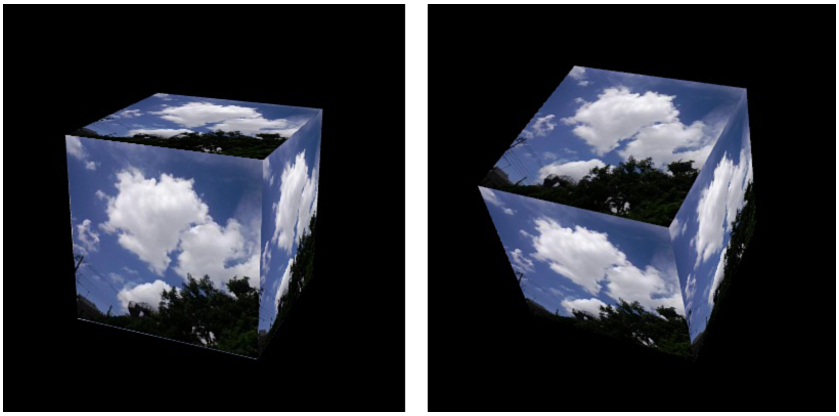

Figure 10.1 A screen shot of RotateObject

How to Implement Object Rotation

Rotating a 3D object is simply the application of a technique you've already studied for 2D objects—transforming the vertex coordinates by using the model view projection matrix. The process requires you to create a rotation matrix based on the mouse movement, change the model view projection matrix, and then transform the coordinates by using the matrix.

You can obtain the amount of mouse movement by simply recording the position where the mouse is initially clicked and then subtracting that position from the new position as the mouse moves. Clearly, an event handler will be needed to calculate the mouse movement, and then this will be converted into an angle that will rotate the object. Let's take a look at the sample program.

Sample Program (RotateObject.js)

Listing 10.1 shows the sample program. As you can see, the shaders do not do anything special. Line 9 in the vertex shader transforms the vertex coordinates by using the model view projection matrix, and line 10 maps the texture image onto the cube.

Listing 10.1 RotateObject.js

```
  1  // RotateObject.js
  2  // Vertex shader program
  3  var VSHADER_SOURCE =
       ...
  8    'void main() {\n' +
  9    '  gl_Position = u_MvpMatrix * a_Position;\n' +
 10    '  v_TexCoord = a_TexCoord;\n' +
 11    '}\n';
       ...
 24  function main() {
       ...
 42    var n = initVertexBuffers(gl);
       ...
 61    viewProjMatrix.setPerspective(30.0, canvas.width / canvas.height,
                                                                  ➥1.0, 100.0);
 62    viewProjMatrix.lookAt(3.0, 3.0, 7.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
 63
 64    // Register the event handler
 65    var currentAngle = [0.0, 0.0]; // [x-axis, y-axis] degrees
 66    initEventHandlers(canvas, currentAngle);
       ...
 74    var tick = function() {             // Start drawing
 75      draw(gl, n, viewProjMatrix, u_MvpMatrix, currentAngle);
 76      requestAnimationFrame(tick, canvas);
 77    };
```

```
  tick();
}
  ...
function initEventHandlers(canvas, currentAngle) {
  var dragging = false;             // Dragging or not
  var lastX = -1, lastY = -1;       // Last position of the mouse

  canvas.onmousedown = function(ev) {  // Mouse is pressed
    var x = ev.clientX, y = ev.clientY;
    // Start dragging if a mouse is in <canvas>
    var rect = ev.target.getBoundingClientRect();
    if (rect.left <= x && x < rect.right && rect.top <= y && y < rect.bottom) {
      lastX = x; lastY = y;
      dragging = true;
    }
  };
   // Mouse is released
  canvas.onmouseup = function(ev) { dragging = false; };

  canvas.onmousemove = function(ev) {     // Mouse is moved
    var x = ev.clientX, y = ev.clientY;
    if (dragging) {
      var factor = 100/canvas.height; // The rotation ratio
      var dx = factor * (x - lastX);
      var dy = factor * (y - lastY);
      // Limit x-axis rotation angle to -90 to 90 degrees
      currentAngle[0] = Math.max(Math.min(currentAngle[0] + dy, 90.0), -90.0);
      currentAngle[1] = currentAngle[1] + dx;
    }
    lastX = x, lastY = y;
  };
}

var g_MvpMatrix = new Matrix4(); // The model view projection matrix
function draw(gl, n, viewProjMatrix, u_MvpMatrix, currentAngle) {
  // Calculate the model view projection matrix
  g_MvpMatrix.set(viewProjMatrix);
  g_MvpMatrix.rotate(currentAngle[0], 1.0, 0.0, 0.0); // x-axis
  g_MvpMatrix.rotate(currentAngle[1], 0.0, 1.0, 0.0); // y-axis
  gl.uniformMatrix4fv(u_MvpMatrix, false, g_MvpMatrix.elements);

  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED_BYTE, 0);
}
```

At lines 61 and 62 of `main()` in JavaScript, the view projection matrix is calculated in advance. You will have to change the model matrix on-the-fly according to the amount of mouse movement.

The code from line 65 registers the event handlers, a key part of this sample program. The variable `currentAngle` is initialized at line 65 and used to hold the current rotation angle. Here, it is an array because it needs to handle two rotation angles around the x-axis and y-axis. The actual registration of the event handlers is done inside `initEventHandlers()`, called at line 66. It draws the cube using the function `tick()` that is defined from line 74.

`initEventHandlers()` is defined at line 138. The code from line 142 handles mouse down, the code from line 152 handles mouse up, and the code from line 154 handles the mouse movement.

The processing when the mouse button is first pushed at line 142 is simple. Line 146 checks whether the mouse has been pressed inside the `<canvas>` element. If it is inside the `<canvas>`, line 147 saves that position in `lastX` and `lastY`. Then the variable `dragging`, which indicates dragging has begun, is set to `true` at line 148.

The processing of the mouse button release at line 152 is simple. Because this indicates that dragging is done, the code simply sets the variable `dragging` back to `false`.

The processing from line 154 is the critical part and tracks the movement of the mouse. Line 156 checks whether dragging is taking place and, if it is, lines 158 and 159 calculate how long it has moved, storing the results to `dx` and `dy`. These values are scaled, using `factor`, which is a function of the canvas size. Once the distance dragged has been calculated, it can be used to determine the new angle by directly adding to the current angles at line 161 and 162. The code limits rotation from –90 to +90 degrees simply to show the technique; you are free to remove this. Because the mouse has moved, its position is saved in `lastX` and `lastY`.

Once you have successfully transformed the movement of the mouse into a rotation angle, you can let the rotation matrix handle the updates and draw the results using `tick()`. These operations are done at lines 172 and 173.

This quick review of a technique to calculate the rotation angle is only one approach. Others, such as placing virtual track balls around the object, are described in detail in the book *3D User Interfaces*.

Select an Object

When your application requires users to be able to control 3D objects interactively, you will need a technique to allow users to select objects. There are many uses of this technique, such as selecting a 3D button created by a 3D model instead of the conventional 2D GUI button, or selecting a photo among multiple photos in a 3D scene.