Graphics

Jungchan Cho
Dept. of Software, Gachon University

Many slides from Edward Angel and Dave Shreine
Many examples are from https://webglfundamentals.org/

Review of Uniform Qualifier

Uniforms can be many types. For each type you have to call the corresponding function to set it.

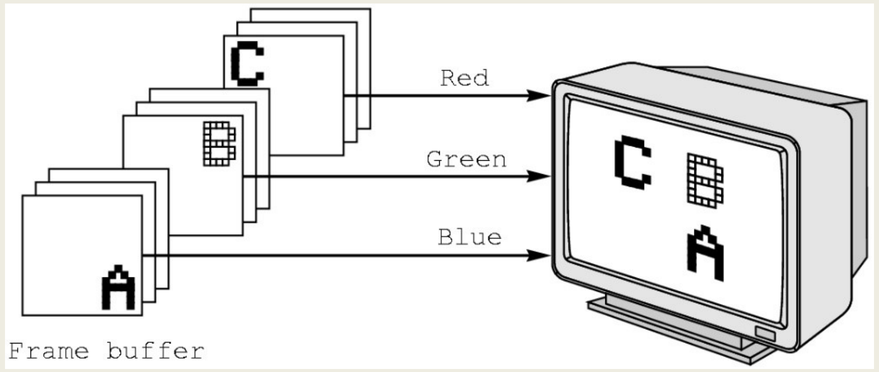
- gl.uniform1f (floatUniformLoc, v); // for float
- gl.uniform1fv(floatUniformLoc, [v]); // for float or float array
- gl.uniform2f (vec2UniformLoc, v0, v1); // for vec2
- gl.uniform2fv(vec2UniformLoc, [v0, v1]); // for vec2 or vec2 array
- gl.uniform3f (vec3UniformLoc, v0, v1, v2); // for vec3
- gl.uniform3fv(vec3UniformLoc, [v0, v1, v2]); // for vec3 or vec3 array
- gl.uniform4f (vec4UniformLoc, v0, v1, v2, v4); // for vec4
- gl.uniform4fv(vec4UniformLoc, [v0, v1, v2, v4]); // for vec4 or vec4 array
-
- gl.uniformMatrix2fv(mat2UniformLoc, false, [4x element array]) // for mat2 or mat2 array
- gl.uniformMatrix3fv(mat3UniformLoc, false, [9x element array]) // for mat3 or mat3 array
- gl.uniformMatrix4fv(mat4UniformLoc, false, [16x element array]) // for mat4 or mat4 array

Uniforms can be many types. For each type you have to call the corresponding function to set it.

- gl.uniform1i (intUniformLoc, v); // for int
- gl.uniform1iv(intUniformLoc, [v]); // for int or int array
- gl.uniform2i (ivec2UniformLoc, v0, v1); // for ivec2
- gl.uniform2iv(ivec2UniformLoc, [v0, v1]); // for ivec2 or ivec2 array
- gl.uniform3i (ivec3UniformLoc, v0, v1, v2); // for ivec3
- gl.uniform3iv(ivec3UniformLoc, [v0, v1, v2]); // for ivec3 or ivec3 array
- gl.uniform4i (ivec4UniformLoc, v0, v1, v2, v4); // for ivec4
- gl.uniform4iv(ivec4UniformLoc, [v0, v1, v2, v4]); // for ivec4 or ivec4 array
-
- gl.uniform1i (sampler2DUniformLoc, v); // for sampler2D (textures)
- gl.uniform1iv(sampler2DUniformLoc, [v]); // for sampler2D or sampler2D array
-
- gl.uniform1i (samplerCubeUniformLoc, v); // for samplerCube (textures)
- gl.uniform1iv(samplerCubeUniformLoc, [v]); // for samplerCube or samplerCube array

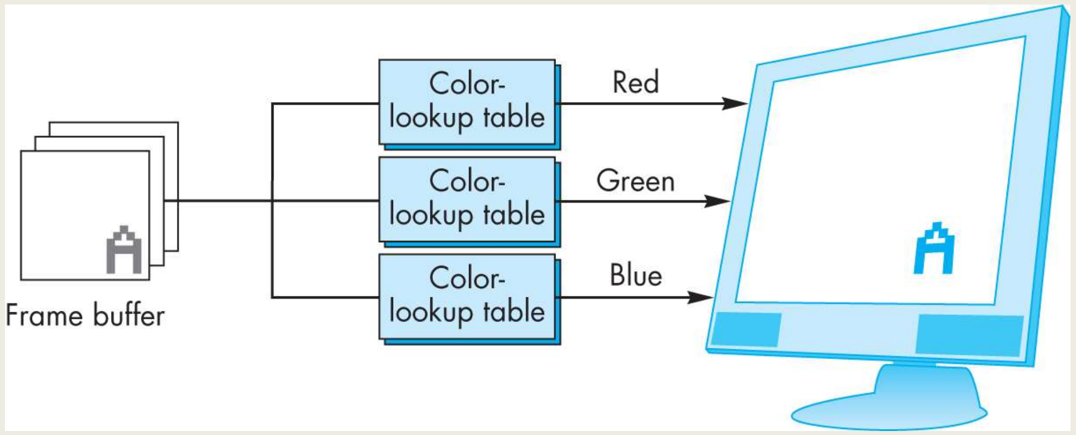
RGB color

- Each color component is stored separately in the frame buffer
- Usually 8 bits per component in buffer
- Color values can range from 0.0 (none) to 1.0 (all) using floats or over the range from 0 to 255 using unsigned bytes



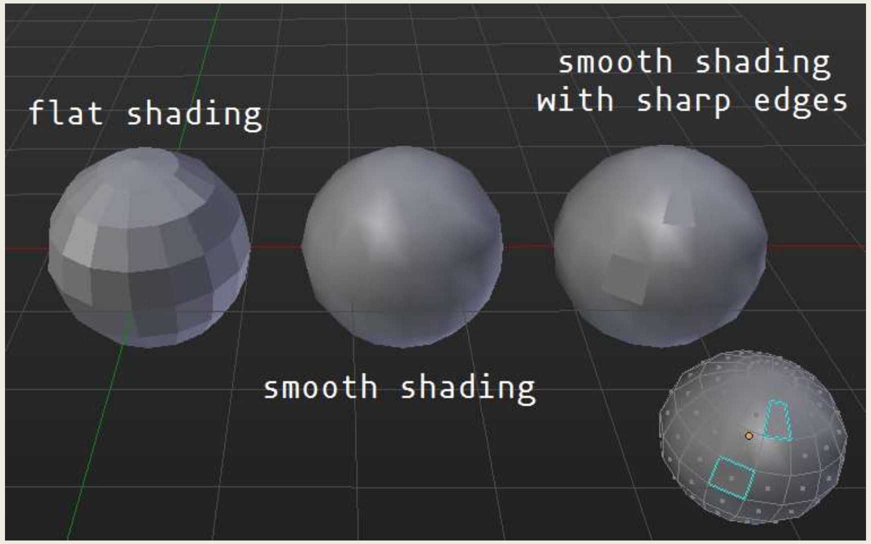
Indexed Color

- Colors are indices into tables of RGB values
- Requires less memory
  - indices usually 8 bits
  - not as important now
    - Memory inexpensive
    - Need more colors for shading

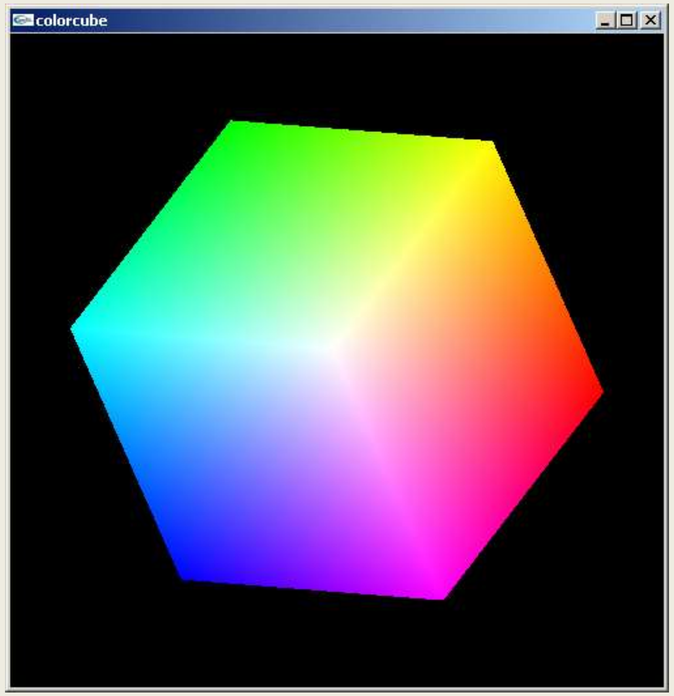


Smooth Color

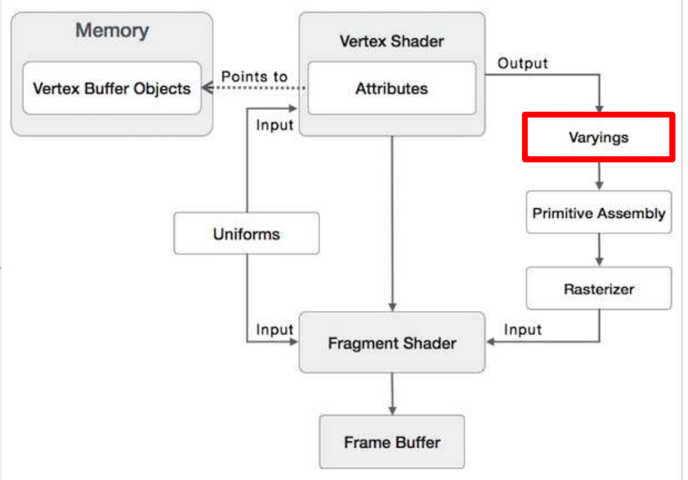
- Default is *smooth* shading
  - Rasterizer interpolates vertex colors across visible polygons
- Alternative is *flat shading*
  - Color of first vertex determines fill color



https://doc.babylonjs.com/resources/blender_tips



Varying Qualified



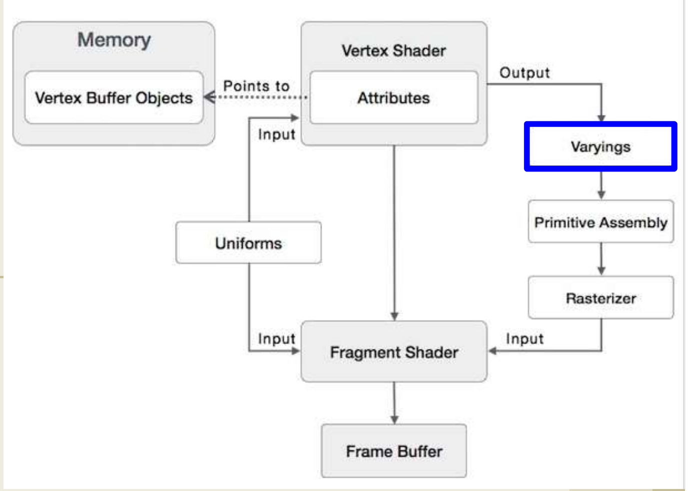
- Variables that are passed **from vertex shader to fragment shader**
- Automatically interpolated by the rasterizer
- With WebGL, GLSL uses the varying qualifier in both shaders

```
varying vec4 color;
```

- More recent versions of WebGL use **out** in vertex shader and **in** in the fragment shader

```
out vec4 color; //vertex shader
in vec4 color;  // fragment shader
```

Our Naming Convention



- Variable variables begin with **f** (fColor) in both shaders
  - **<u>must</u>** have same name

Example: Vertex Shader

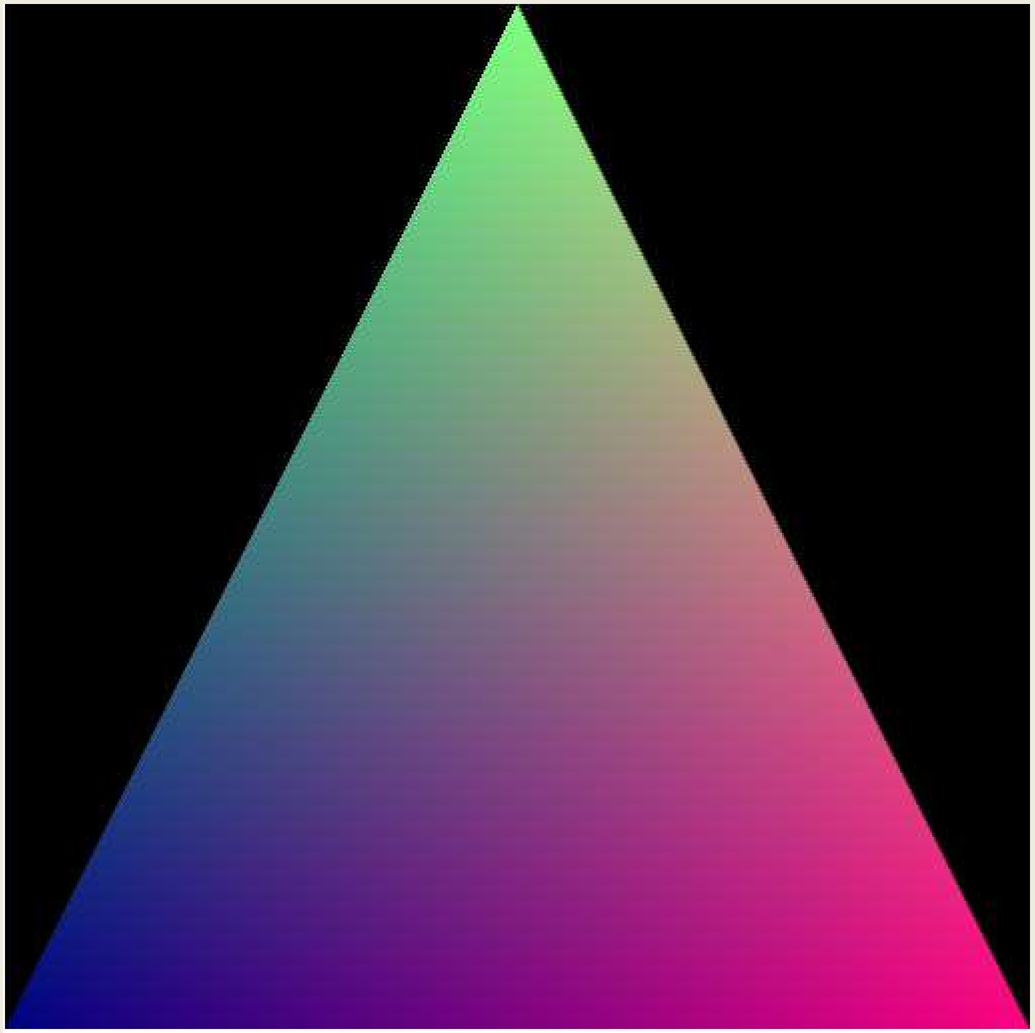
```
attribute vec4 vColor;
varying vec4 fColor;
void main()
{
  gl_Position = vPosition;
  fColor = vColor;
}
```

Corresponding Fragment Shader

```
precision mediump float;

varying vec4 fColor;
void main()
{
  gl_FragColor = fColor;
}
```

Programming with WebGL: Varying Qualifiers

[varying]
triangle_varying.html



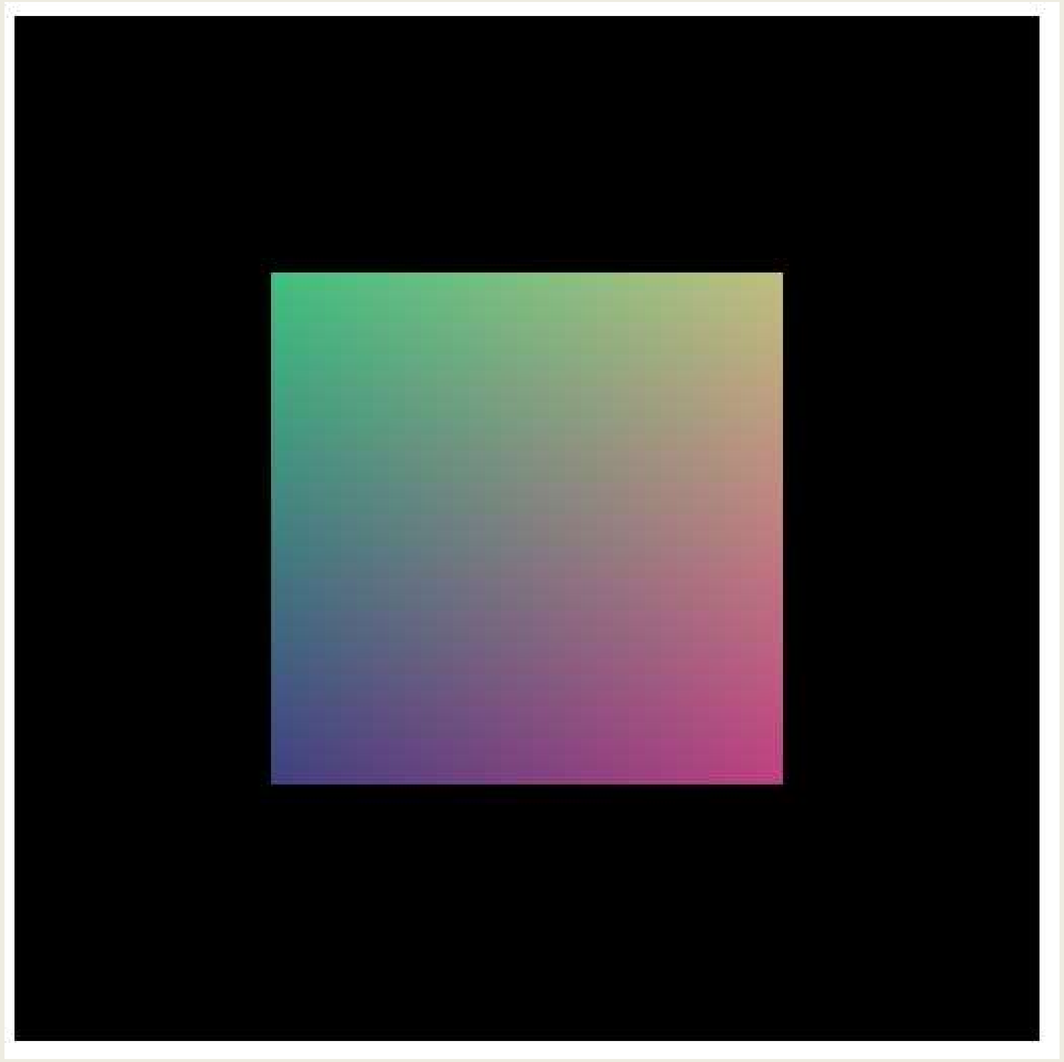
Vertex Shader

```
attribute vec4 vPosition;
// we declare the same varying in the fragment shader.
varying vec4 fColor;

void
main()
{
            // Convert from clipspace to colorspace.
            // Clipspace goes -1.0 to +1.0
            // Colorspace goes from 0.0 to 1.0
           gl_Position = vPosition;
          fColor = gl_Position * 0.5 + 0.5;
}
```

Fragment Shader

```
precision mediump float;

// we declare the same varying in the fragment shader.
// WebGL will connect the varying in the vertex shader to the
//varying of the same name and type in the fragment shader.
varying vec4 fColor;

void
main()
{
		//gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
		gl_FragColor = fColor;
}
```

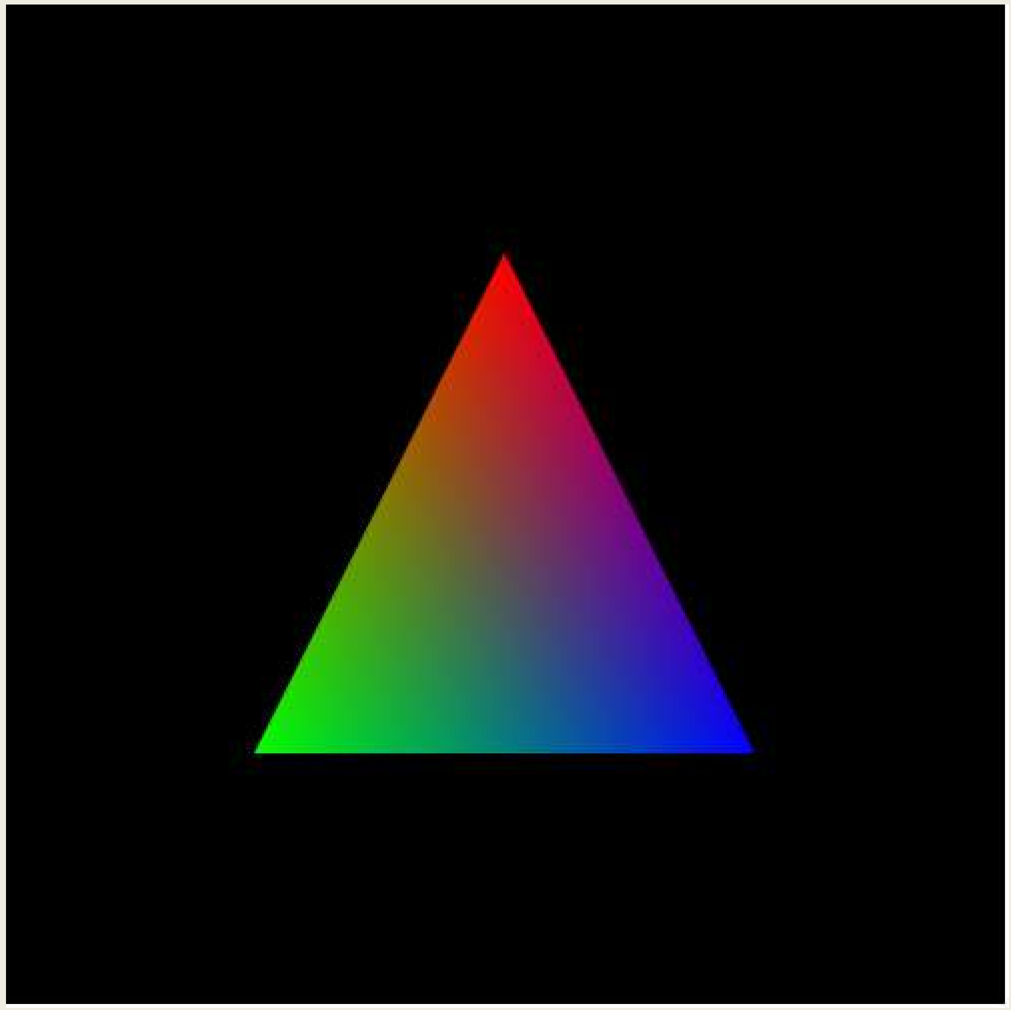
[varying]
square_color.html



square_color.html

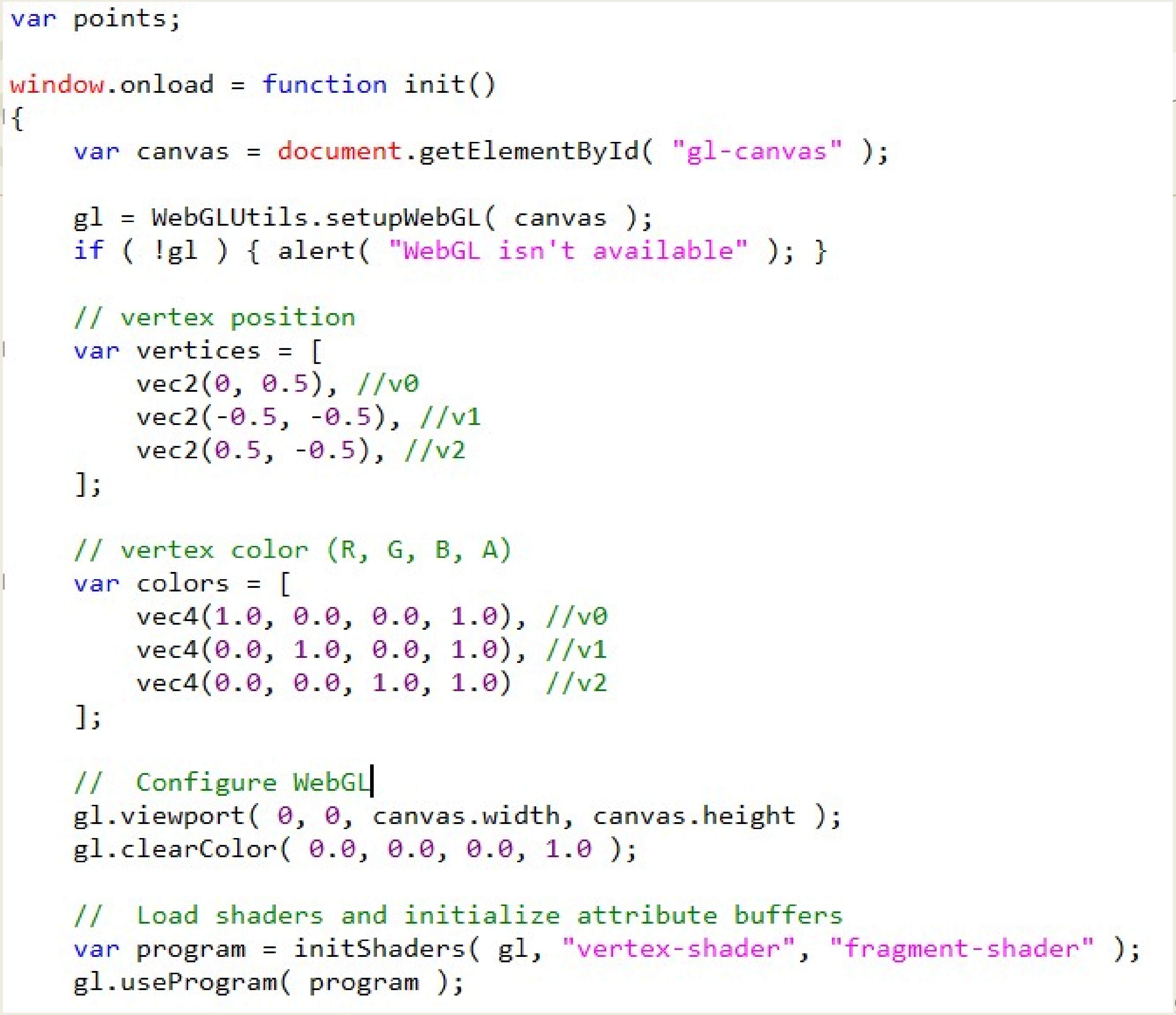
```
<script id="vertex-shader" type="x-shader/x-vertex">
varying vec4 v_color;
attribute vec4 vPosition;
void
main()
{
   gl_Position = vPosition;
   v_color = gl_Position * 0.5 + 0.5;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
varying vec4 v_color;
void
main()
{
   gl_FragColor = v_color;
}
</script>
```

[vertex color] triangle_colors.html

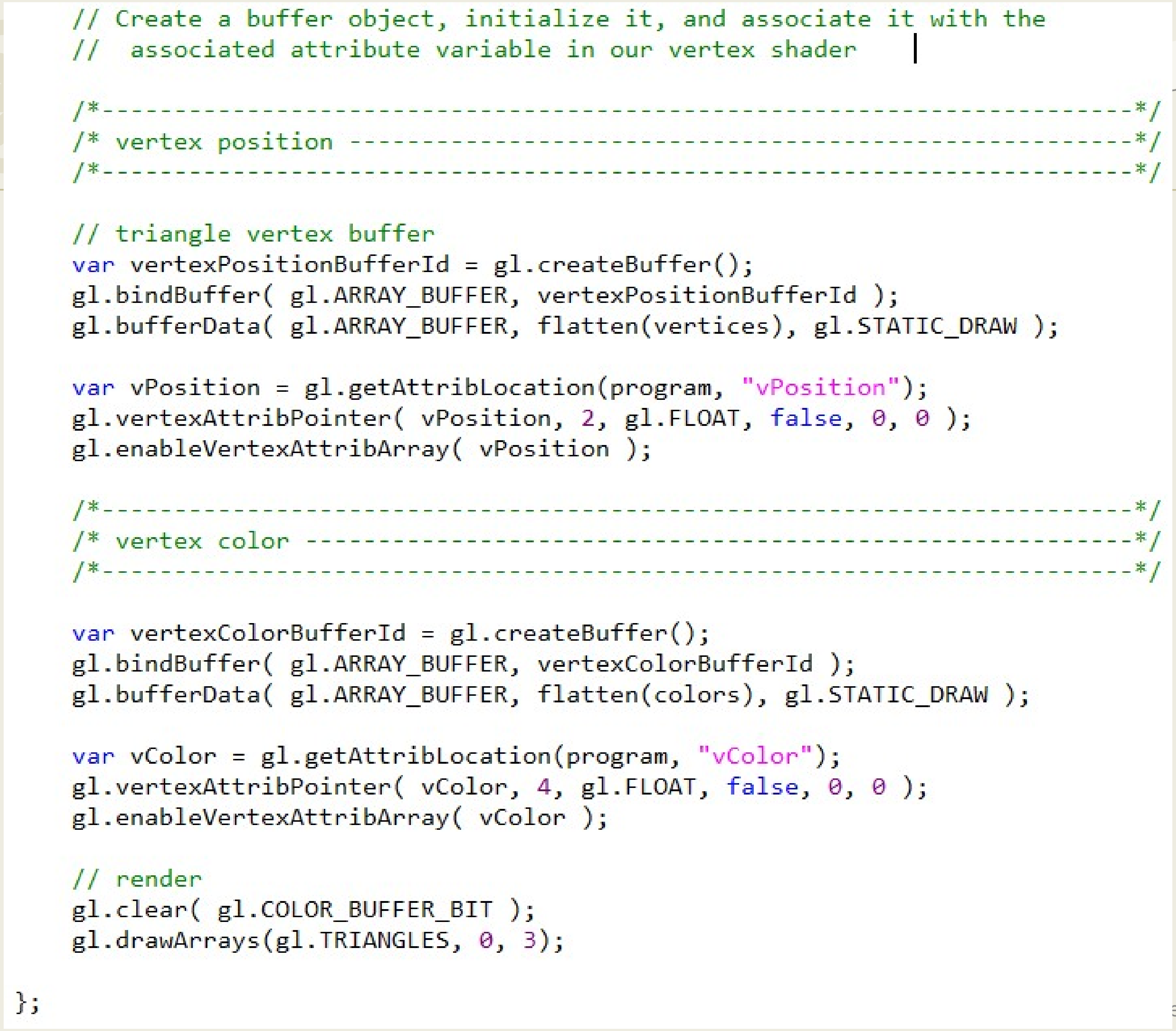


```
// vertex shader
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;

void
main()
{
	fColor = vColor;
	gl_Position = vPosition;
}
```

```
// fragment shager
precision mediump float;
varying vec4 fColor;

void
main()
{
	gl_FragColor = fColor;
}
```



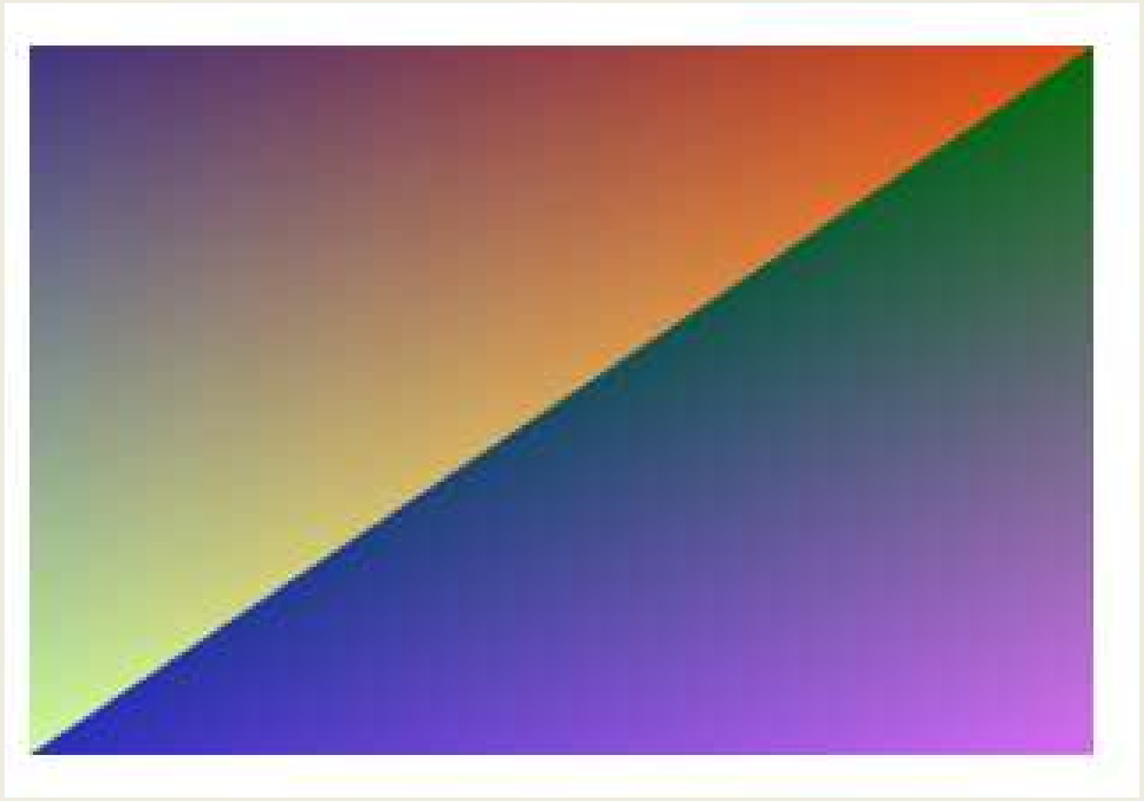
```
var points;

window.onload = function init()
{
    var canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    // vertex position
    var vertices = [
        vec2(0, 0.5), //v0
        vec2(-0.5, -0.5), //v1
        vec2(0.5, -0.5), //v2
    ];

    // vertex color (R, G, B, A)
    var colors = [
        vec4(1.0, 0.0, 0.0, 1.0), //v0
        vec4(0.0, 1.0, 0.0, 1.0), //v1
        vec4(0.0, 0.0, 1.0, 1.0)  //v2
    ];

    //  Configure WebGL
    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 0.0, 0.0, 0.0, 1.0 );

    //  Load shaders and initialize attribute buffers
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );
```



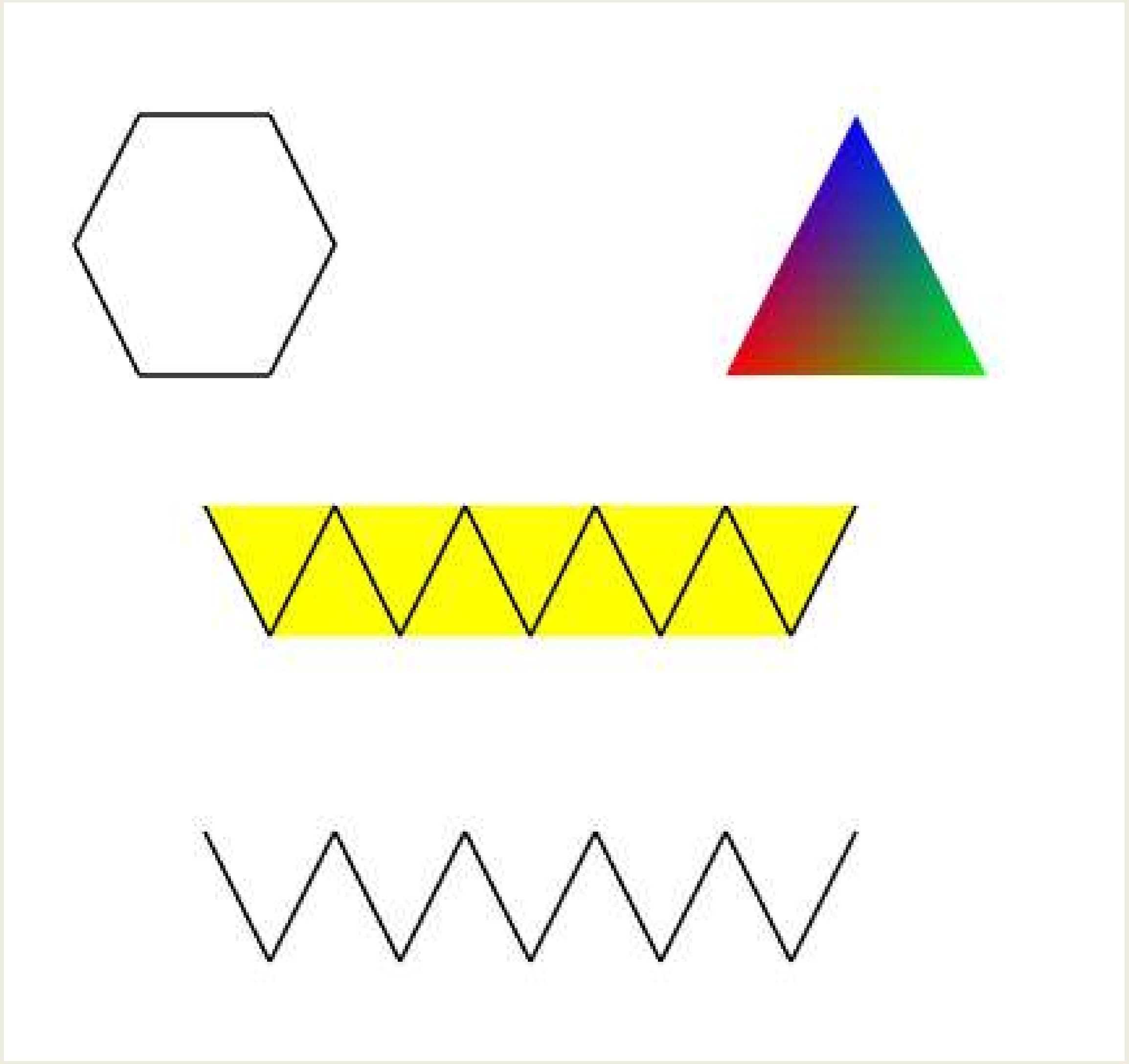
```
    // Create a buffer object, initialize it, and associate it with the
    //  associated attribute variable in our vertex shader

    /*-----------------------------------------------------------------------*/
    /* vertex position ------------------------------------------------------*/
    /*-----------------------------------------------------------------------*/

    // triangle vertex buffer
    var vertexPositionBufferId = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, vertexPositionBufferId );
    gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

    var vPosition = gl.getAttribLocation(program, "vPosition");
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );

    /*-----------------------------------------------------------------------*/
    /* vertex color ---------------------------------------------------------*/
    /*-----------------------------------------------------------------------*/

    var vertexColorBufferId = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, vertexColorBufferId );
    gl.bufferData( gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW );

    var vColor = gl.getAttribLocation(program, "vColor");
    gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vColor );

    // render
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays(gl.TRIANGLES, 0, 3);

};
```

Exercise 4

If we make each color different we'll see the interpolation. Now we see the interpolated varying.



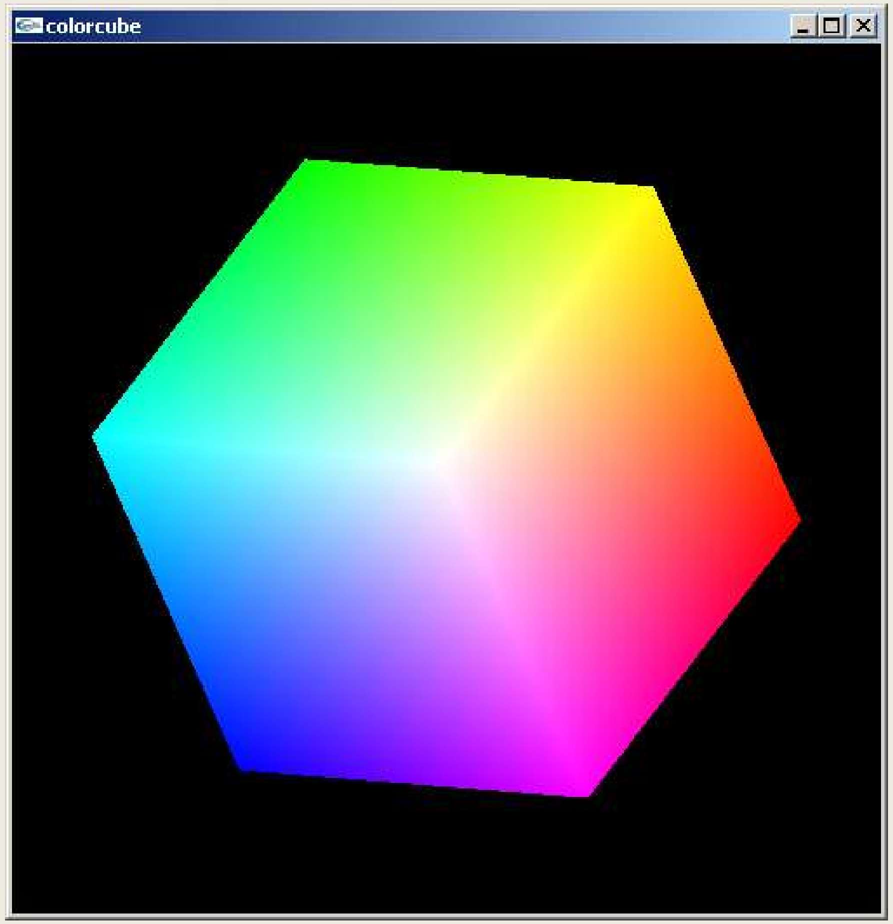
Exercise 5



```
<!DOCTYPE html>
<html>
<head>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;

void
main()
{
    fColor = vColor;
    gl_Position = vPosition;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

varying vec4 fColor;

void
main()
{
    gl_FragColor = fColor;
}
</script>


<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="polygon_primitive_colors.js"></script>
</head>

<body>

<canvas id="gl-canvas" width="500" height="500">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```

```
// hexagon vertices
var hexagonVertices = [
    vec2(-0.3,  0.6), //v0
    vec2(-0.4,  0.8), //v1
    vec2(-0.6,  0.8), //v2
    vec2(-0.7,  0.6), //v3
    vec2(-0.6,  0.4), //v4
    vec2(-0.4,  0.4), //v5
    vec2(-0.3,  0.6), //v6
];


// triangle vertices
var triangleVertices = [
    vec2(0.3,  0.4), //v0
    vec2(0.7,  0.4), //v1
    vec2(0.5,  0.8), //v2
];


var colors = [
    vec4(1.0, 0.0, 0.0, 1.0), //v0
    vec4(0.0, 1.0, 0.0, 1.0), //v1
    vec4(0.0, 0.0, 1.0, 1.0)  //v2
];



// strip vertices
var stripVertices = [
    vec2(-0.5,  0.2), //v0
    vec2(-0.4,  0.0), //v1
    vec2(-0.3,  0.2), //v2
    vec2(-0.2,  0.0), //v3
    vec2(-0.1,  0.2), //v4
    vec2(0.0,  0.0), //v5
    vec2(0.1,  0.2), //v6
    vec2(0.2,  0.0), //v7
    vec2(0.3,  0.2), //v8
    vec2(0.4,  0.0), //v9
    vec2(0.5,  0.2), //v10
    // start second strip
    vec2(-0.5, -0.3), //v11
    vec2(-0.4, -0.5), //v12
    vec2(-0.3, -0.3), //v13
    vec2(-0.2, -0.5), //v14
    vec2(-0.1, -0.3), //v15
    vec2(0.0, -0.5), //v16
    vec2(0.1, -0.3), //v17
    vec2(0.2, -0.5), //v18
    vec2(0.3, -0.3), //v19
    vec2(0.4, -0.5), //v20
    vec2(0.5, -0.3)  //v21
];
```

Assignment #2



Operators and Functions

Data Types

- C types: int, float, bool
- Vectors:
  - float vec2, vec3, vec4
  - Also int (ivec) and boolean (bvec)
- Matrices: mat2, mat3, mat4
  - Stored by columns
  - Standard referencing m[row][column]
- C++ style constructors
  - vec3 a =vec3(1.0, 2.0, 3.0)
  - vec2 b = vec2(a)

No Pointers

- There are no pointers in GLSL
- We can use C structs which can be copied back from functions
- Because matrices and vectors are basic types they can be passed into and output from GLSL functions, e.g.

  mat3 func(mat3 a)
- variables passed by copying

Operators and Functions

- Standard C functions
  - Trigonometric
  - Arithmetic
  - Normalize, reflect, length

- Overloading of vector and matrix types

```
mat4 a;
vec4 b, c, d;
c = b*a; // a column vector stored as a 1d array
d = a*b; // a row vector stored as a 1d array
```

Swizzling and Selection

- Can refer to array elements by element using [] or selection (.) operator with
  - x, y, z, w
  - r, g, b, a
  - s, t, p, q
  - **`a[2], a.b, a.z, a.p`** are the same

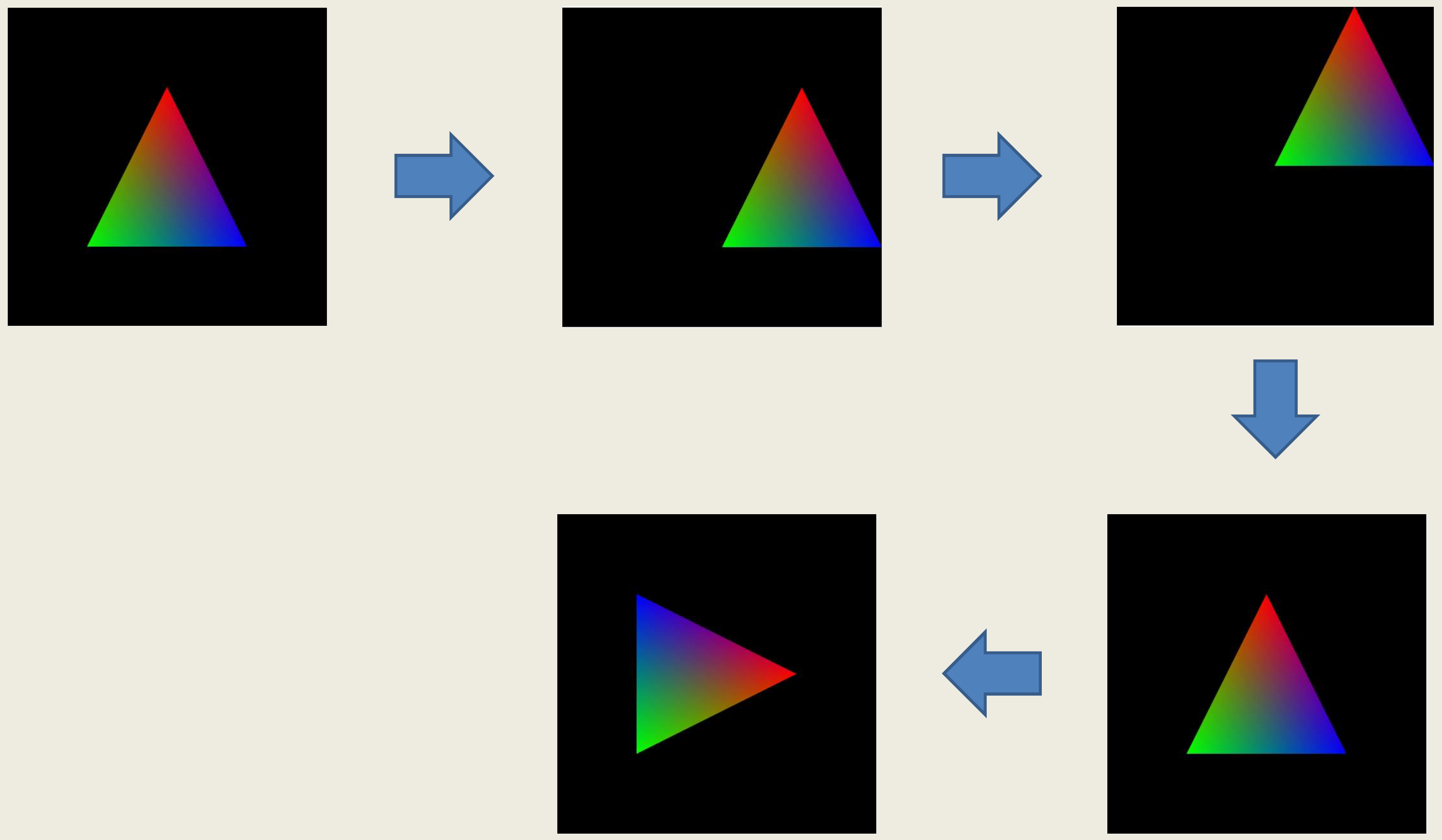
- **Swizzling** operator lets us manipulate components

```
vec4 a, b;
a.yz = vec2(1.0, 2.0, 3.0, 4.0);
b = a.yxzw;
```

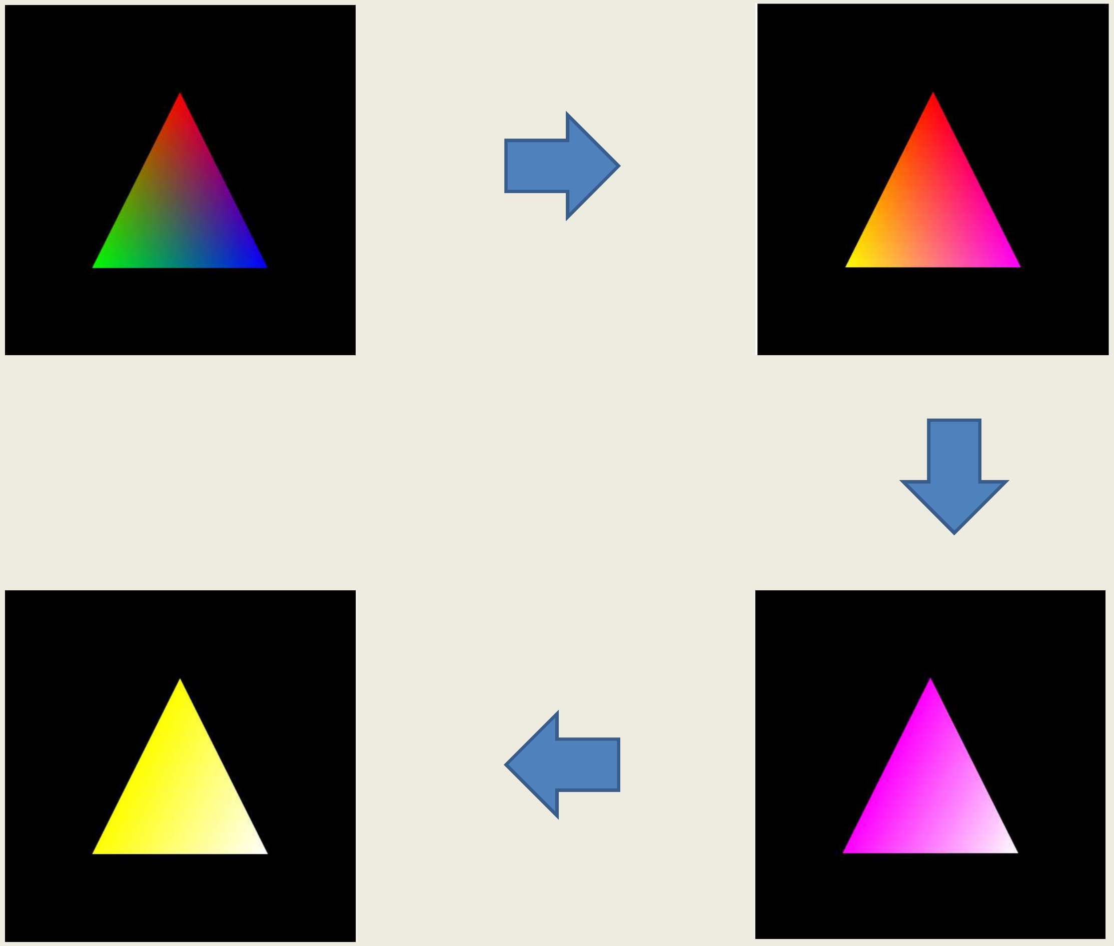
triangle_colors.html

```
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;

void
main()
{
        fColor = vColor;
        gl_Position = vPosition;
        //gl_Position.x = gl_Position.x + 0.5;
        //gl_Position.g = gl_Position.g + 0.5;
        //gl_Position.xy = gl_Position.xy + vec2(-0.5, -0.5);
        //gl_Position.st = gl_Position.ts;
}
```

triangle_colors.html



Exercise 6



Solution to Exercise 5

```
precision mediump float;

varying vec4 fColor;

void
main()
{
        gl_FragColor = fColor;
        //gl_FragColor.r = 1.0;
        //gl_FragColor.g = 1.0;
        //gl_FragColor.rgb = gl_FragColor.gbr;
}
```

Programming with WebGL: Operators and Functions

Clipping

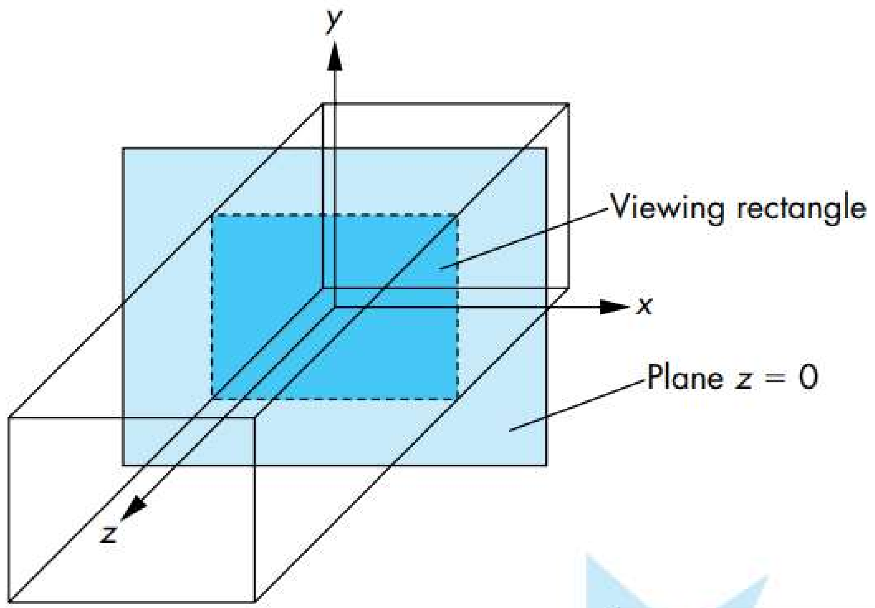


FIGURE 2.34 Viewing volume.

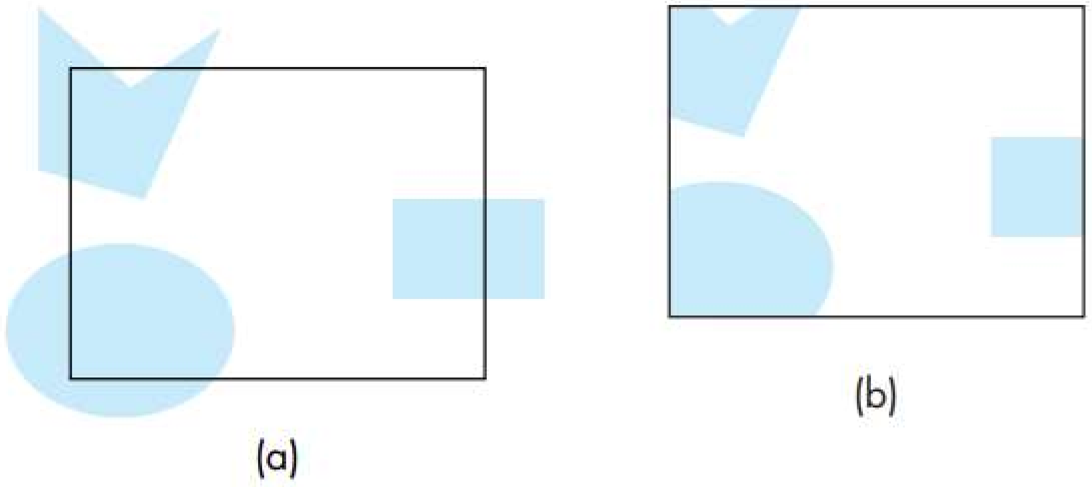
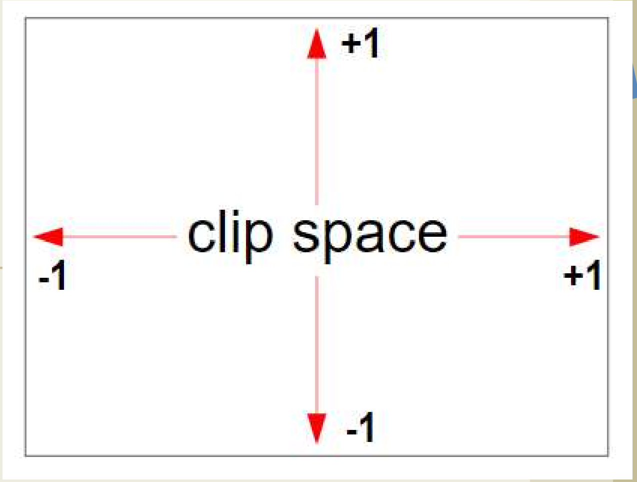
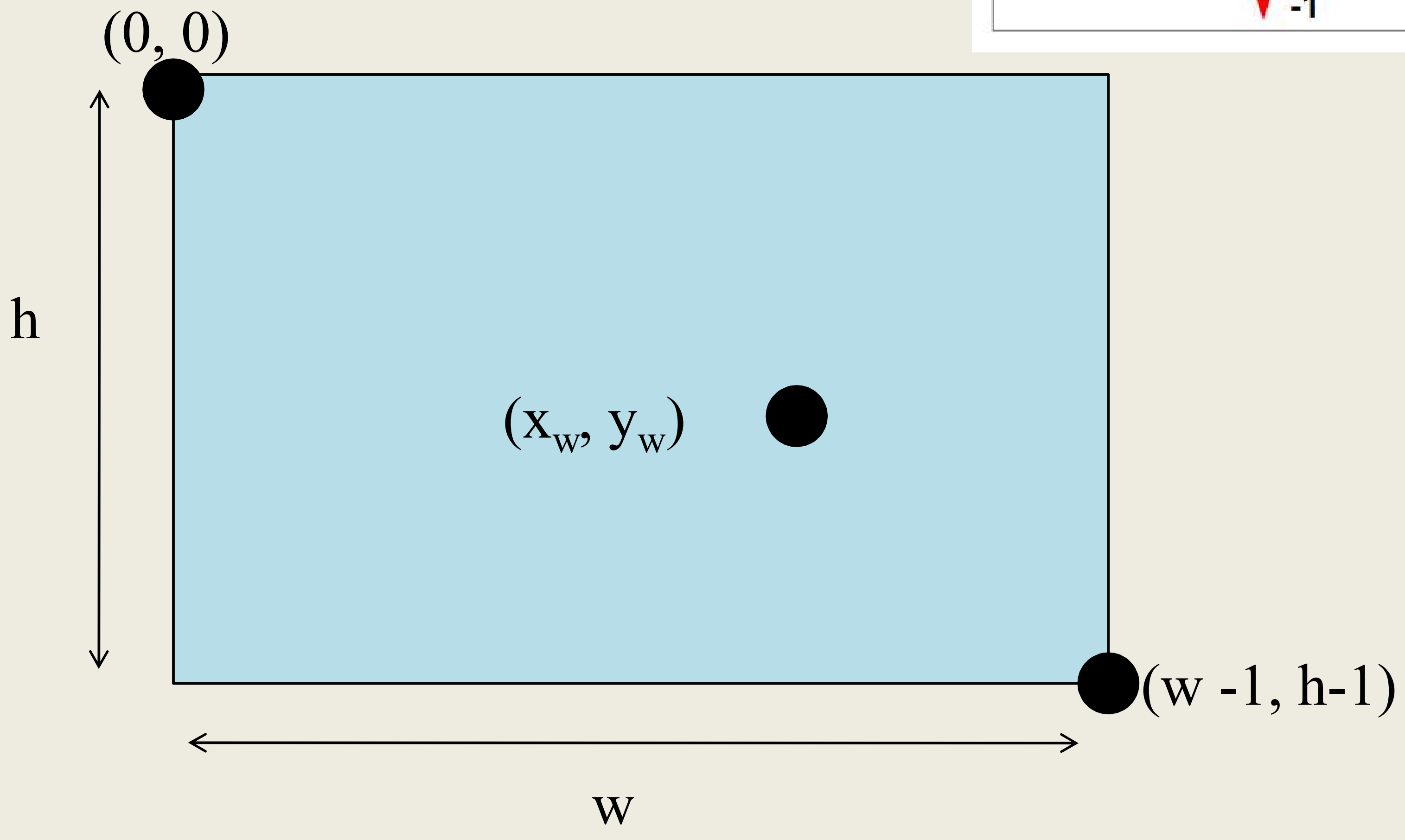


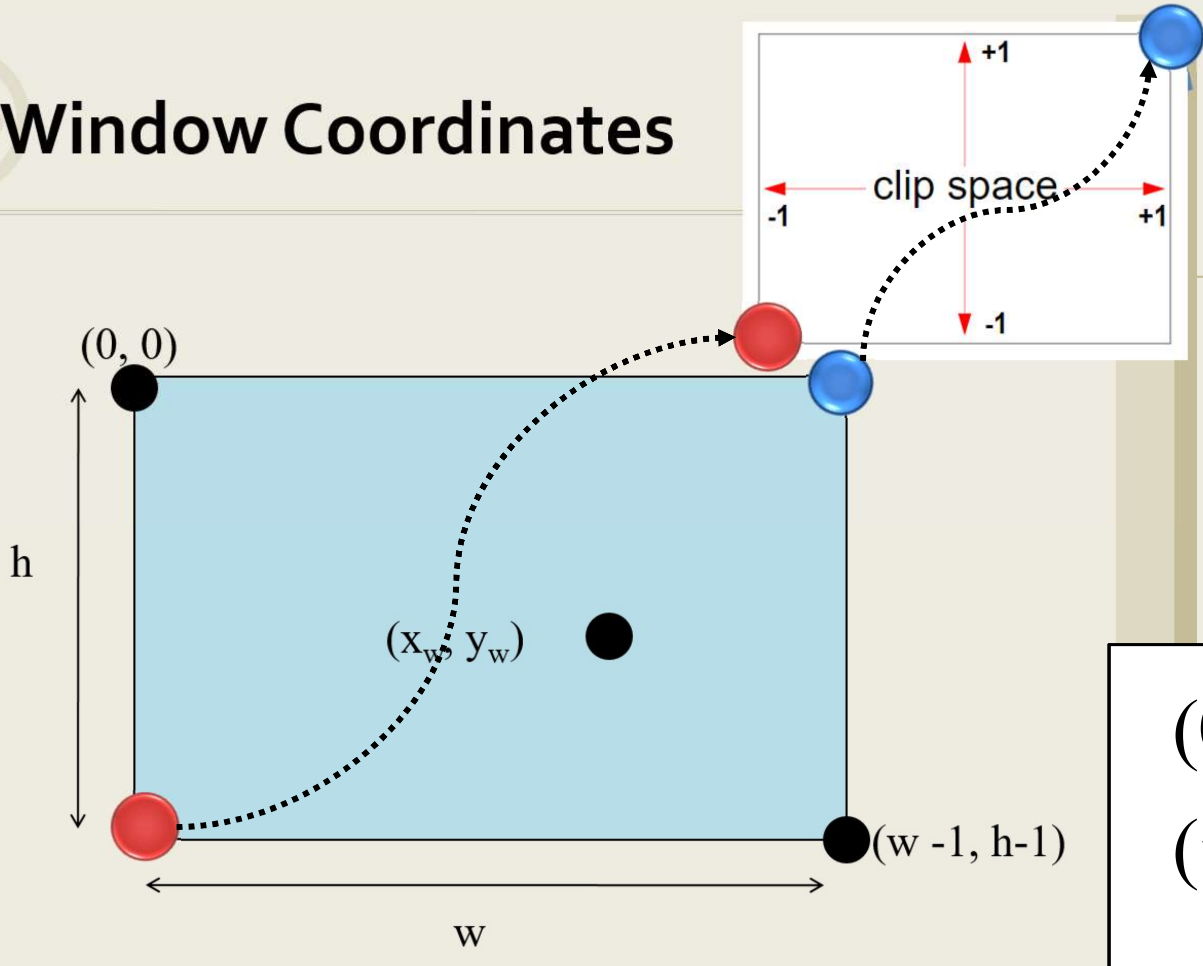
FIGURE 2.35 Two-dimensional viewing. (a) Objects before clipping. (b) Image after clipping.

Window Coordinates





Window Coordinates



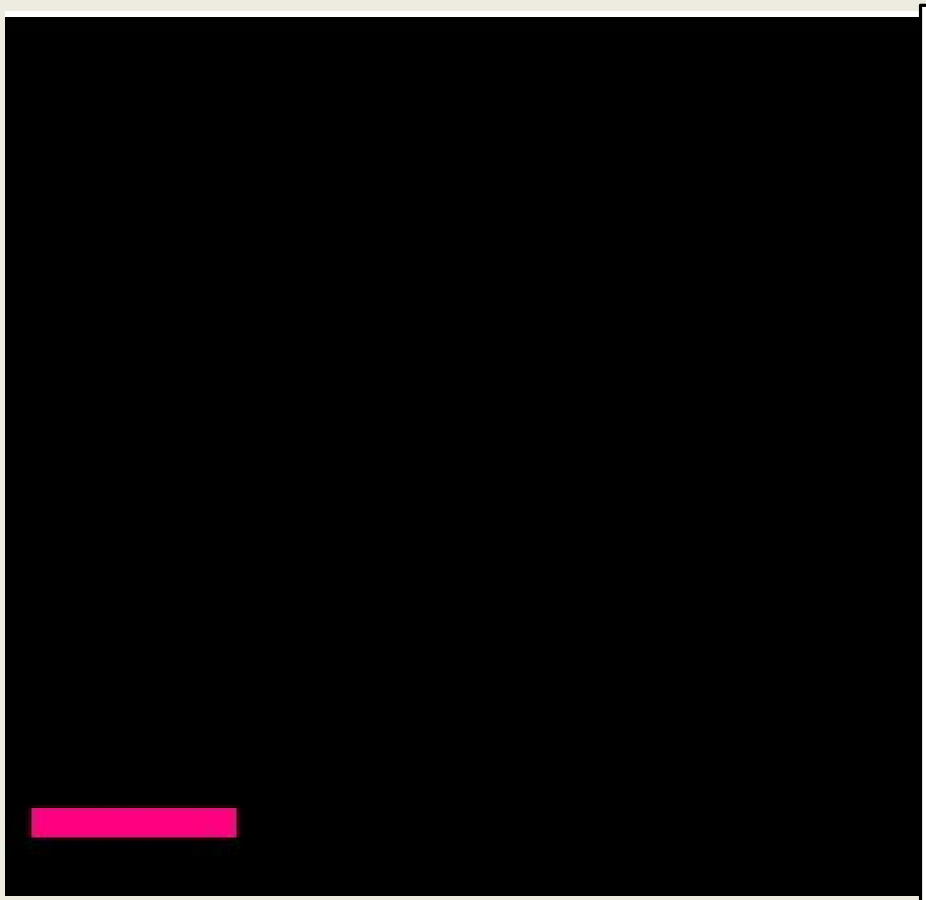
$$(0,h) \rightarrow (-1,-1)$$

$$(w,0) \rightarrow (1,1)$$

$$x = -1 + \frac{2 * x_w}{w}$$

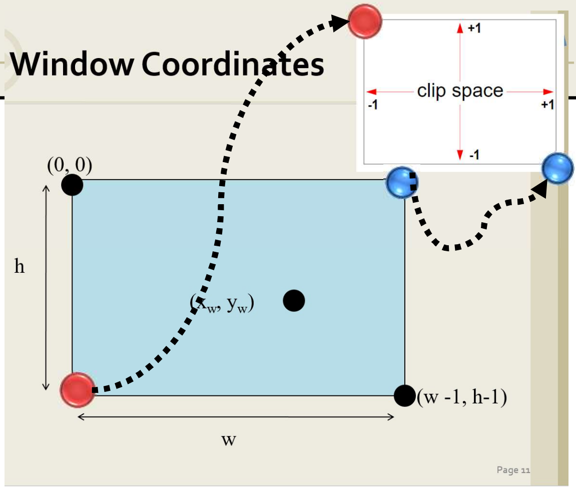
$$y = -1 + \frac{2 * (h - y_w)}{h}$$

[uniform] vertex_ex1.html



```
 var vertices = new
Float32Array([
                10, 20,
                80, 20,
                10, 30,
                10, 30,
                80, 20,
                80, 30,
      ]);
```

vertex_ex1.html



```
attribute vec2 vPosition;
uniform vec2 vResolution;
void
main()
{
        // convert the position from pixels to 0.0 to 1.0
        vec2 zeroToOne = vPosition / vResolution;

        // convert from 0->1 to 0->2
        vec2 zeroToTwo = zeroToOne * 2.0;

        // convert from 0->2 to -1->+1 (clip space)
        vec2 clipSpace = zeroToTwo - 1.0;
        gl_Position = vec4(clipSpace, 0.0, 1.0);
```

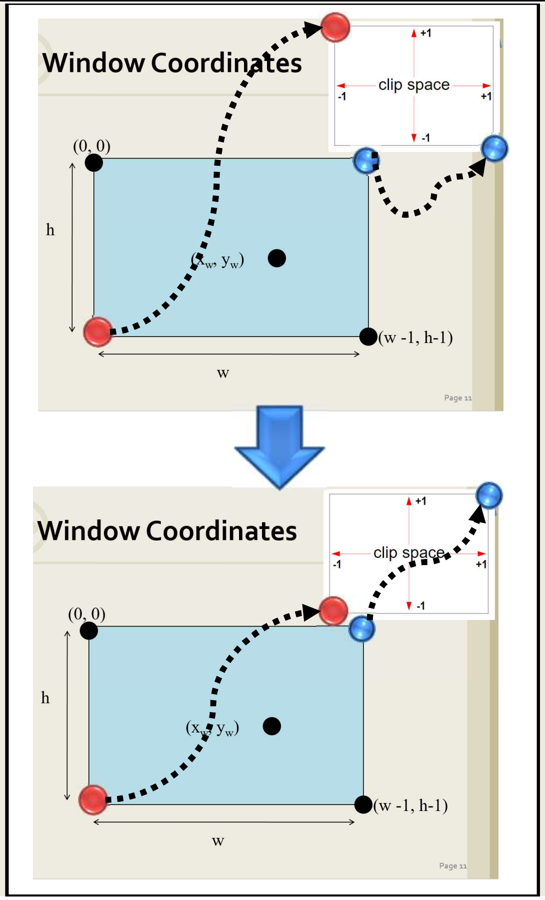
vertex_ex1.js

```
var vertices = new Float32Array([
                        10, 20,  80, 20, 10, 30, 10, 30,  80, 20, 80, 30,
            ]);
```

```
// we added a uniform called vResolution.
var vResolution = gl.getUniformLocation(program, "vResolution");

// set the resolution
gl.uniform2f(vResolution, gl.canvas.width, gl.canvas.height);
```

[redacted]

] vertex_ex2.html

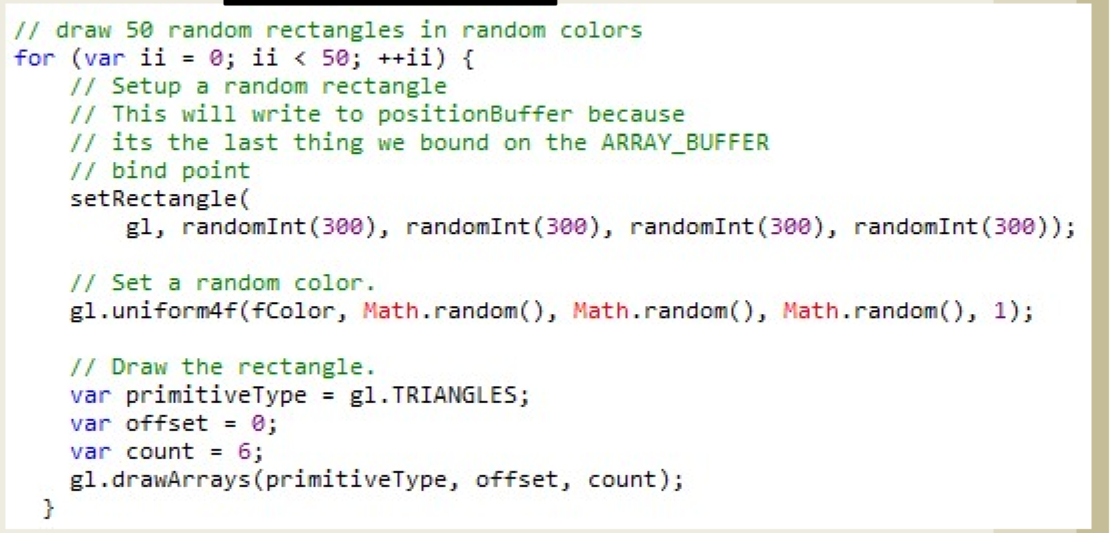


```
attribute vec2 vPosition;
uniform vec2 vResolution;
void
main()
{
  // convert the position from pixels to 0.0 to 1.0
          vec2 zeroToOne = vPosition / vResolution;

          // convert from 0->1 to 0->2
          vec2 zeroToTwo = zeroToOne * 2.0;

          // convert from 0->2 to -1->+1 (clip space)
          vec2 clipSpace = zeroToTwo - 1.0;

          //gl_Position = vec4(clipSpace, 0.0, 1.0);
          // To get it to be the more traditional top left
corner used for 2d graphics APIs we can just flip the clip
space y coordinate.
          gl_Position = vec4(clipSpace * vec2(1, -1), 0, 1);
}
```

Exercise 7

Let's make the code that defines a rectangle into a function so we can call it for different sized rectangles. While we're at it we'll make the color settable.

Hint



```
// draw 50 random rectangles in random colors
for (var ii = 0; ii < 50; ++ii) {
    // Setup a random rectangle
    // This will write to positionBuffer because
    // its the last thing we bound on the ARRAY_BUFFER
    // bind point
    setRectangle(
        gl, randomInt(300), randomInt(300), randomInt(300), randomInt(300));

    // Set a random color.
    gl.uniform4f(fColor, Math.random(), Math.random(), Math.random(), 1);

    // Draw the rectangle.
    var primitiveType = gl.TRIANGLES;
    var offset = 0;
    var count = 6;
    gl.drawArrays(primitiveType, offset, count);
  }
```