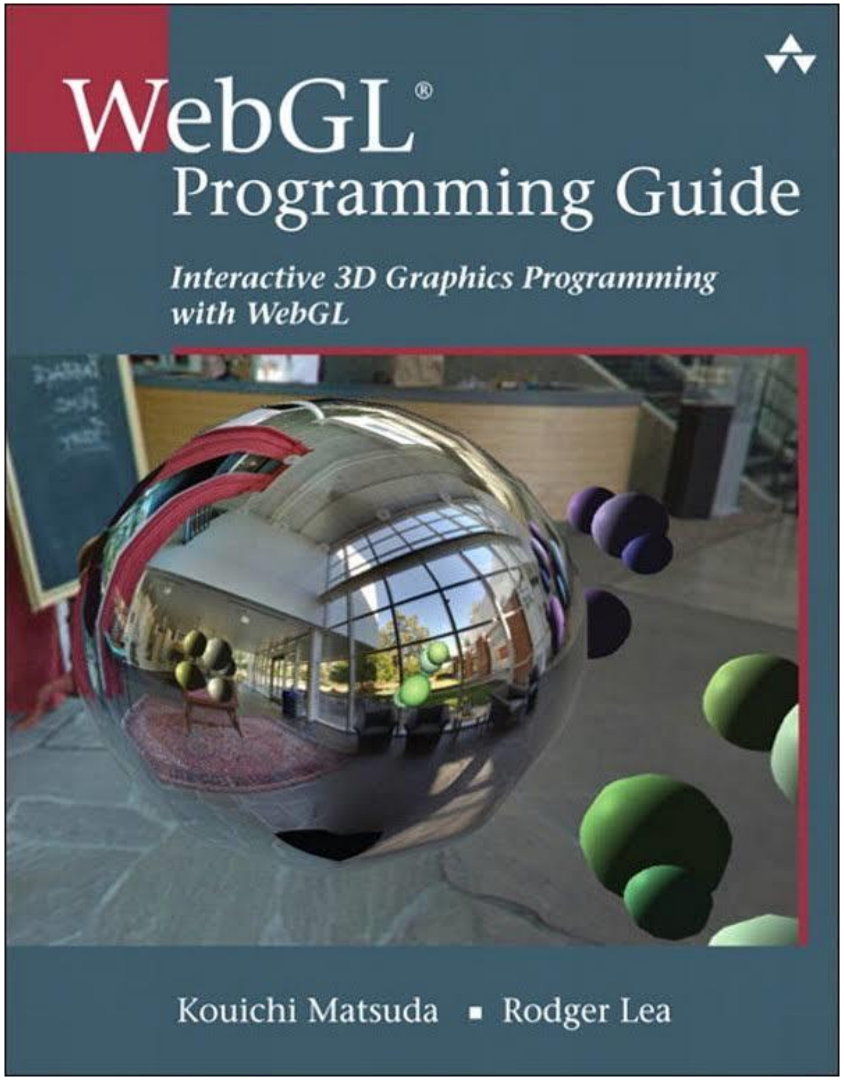
WebGL II

그래픽 정보 처리 (Graphic Information Processing)

WebGL

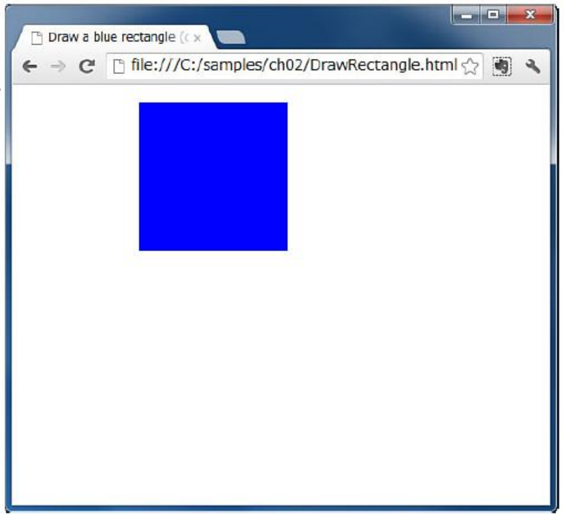
실습 참고 책


WebGL®
Programming Guide
Interactive 3D Graphics Programming with WebGL
Kouichi Matsuda ▪ Rodger Lea

WebGL

DrawRectangle 실습 (Canvas 버전)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Draw a blue rectangle (canvas version)</title>
  </head>

  <body onload="main()">
    <canvas id="example" width="400" height="400">
      Please use a browser that supports "canvas"
    </canvas>
    <script src="01 DrawRectangle.js"></script>
  </body>
</html>
```



```
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('example');
  if (!canvas) {
    console.log('Failed to retrieve the <canvas> element');
    return false;
  }

  // Get the rendering context for 2DCG
  var ctx = canvas.getContext('2d');

  // Draw a blue rectangle
  ctx.fillStyle = 'rgba(0, 0, 255, 1.0)'; // Set color to blue
  ctx.fillRect(120, 10, 150, 150);        // Fill a rectangle
with the color
}
```

WebGL

WebGL 기본 실습

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Clear "canvas"</title>
  </head>

  <body onload="main()">
    <canvas id="webgl" width="400" height="400">
    Please use a browser that supports "canvas"
    </canvas>

    <script src="lib/webgl-utils.js"></script>
    <script src="lib/webgl-debug.js"></script>
    <script src="lib/cuon-utils.js"></script>
    <script src="02 HelloCanvas.js"></script>
  </body>
</html>
```

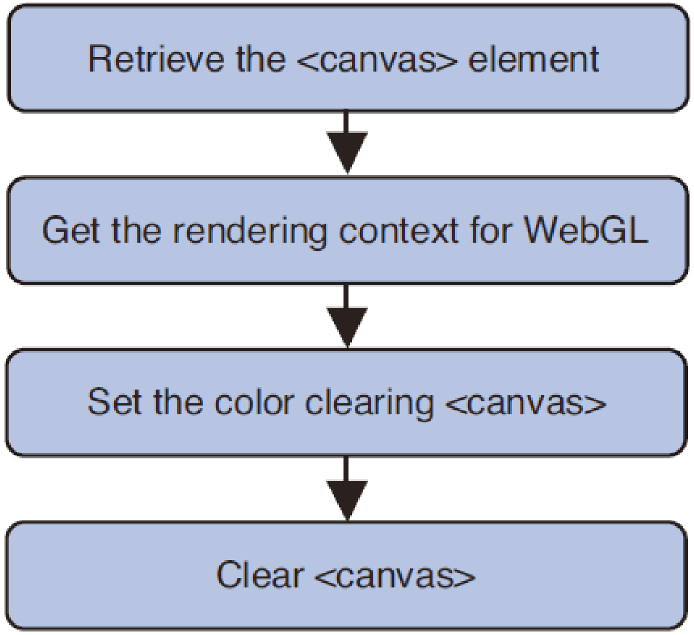
```
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');

  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for
WebGL');
    return;
  }

  // Set clear color
  gl.clearColor(0.0, 0.0, 0.0, 1.0);

  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
}
```

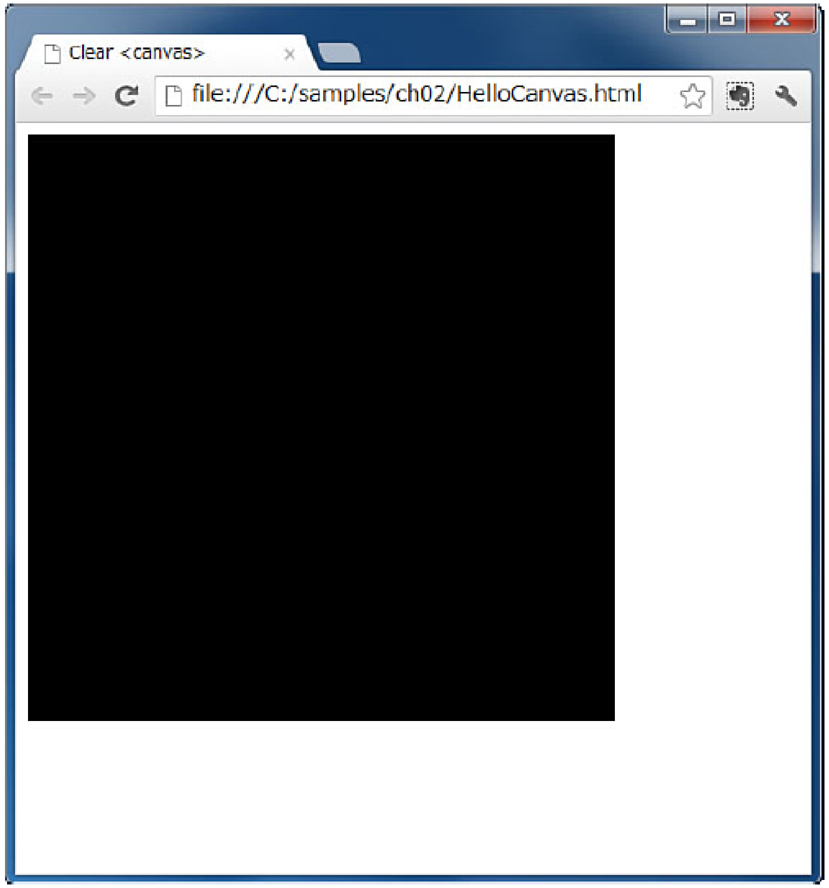
https://github.com/kmindi/webgl-utils.js

https://github.com/KhronosGroup/WebGLDeveloperTools

WebGL

WebGL 기본 실습

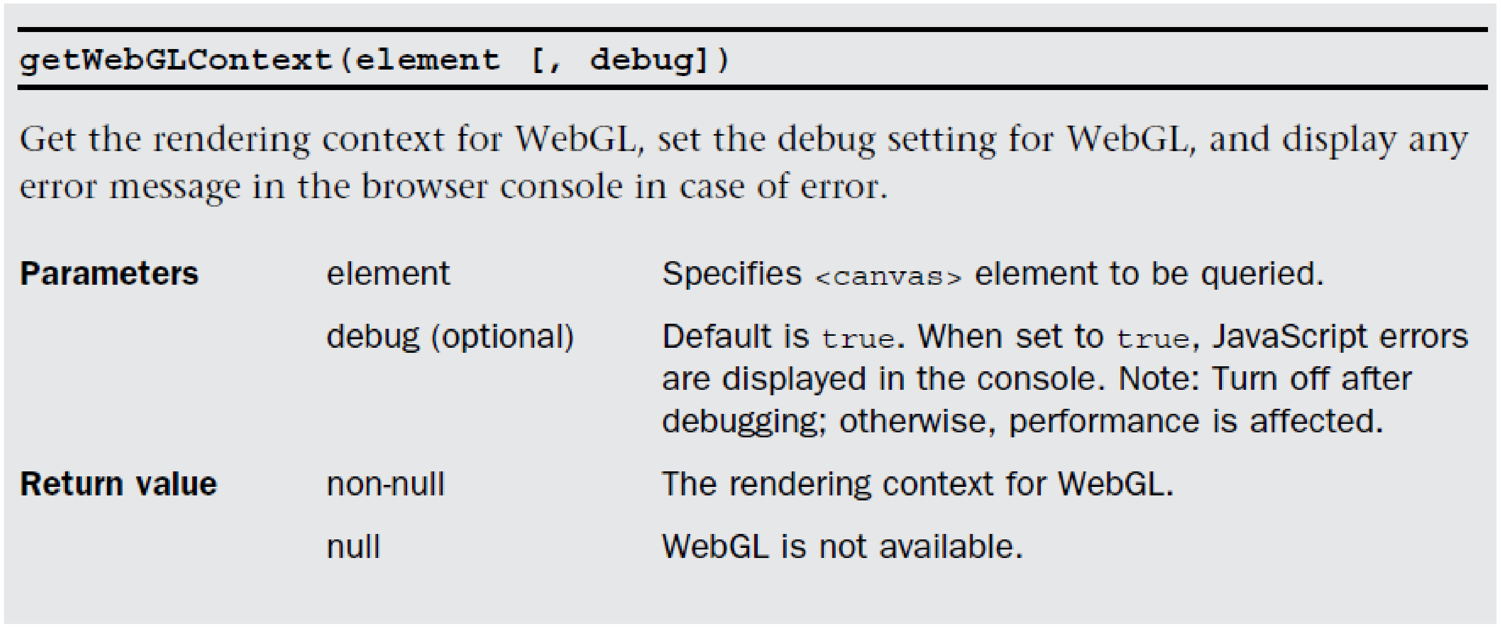

Retrieve the <canvas> element
Get the rendering context for WebGL
Set the color clearing <canvas>
Clear <canvas>



WebGL

getWebGLContext(element, [, debug])

- WebGL에 대한 렌더링 Context를 받아옴
- 선택적 파라미터로 debug 설정을 할 수 있음
- 참고 교제에서 사용하고 있는 js 파일 함수



`getWebGLContext(element [, debug])`

Get the rendering context for WebGL, set the debug setting for WebGL, and display any error message in the browser console in case of error.

Parameters	element	Specifies `<canvas>` element to be queried.
	debug (optional)	Default is `true`. When set to `true`, JavaScript errors are displayed in the console. Note: Turn off after debugging; otherwise, performance is affected.
Return value	non-null	The rendering context for WebGL.
	null	WebGL is not available.

WebGL

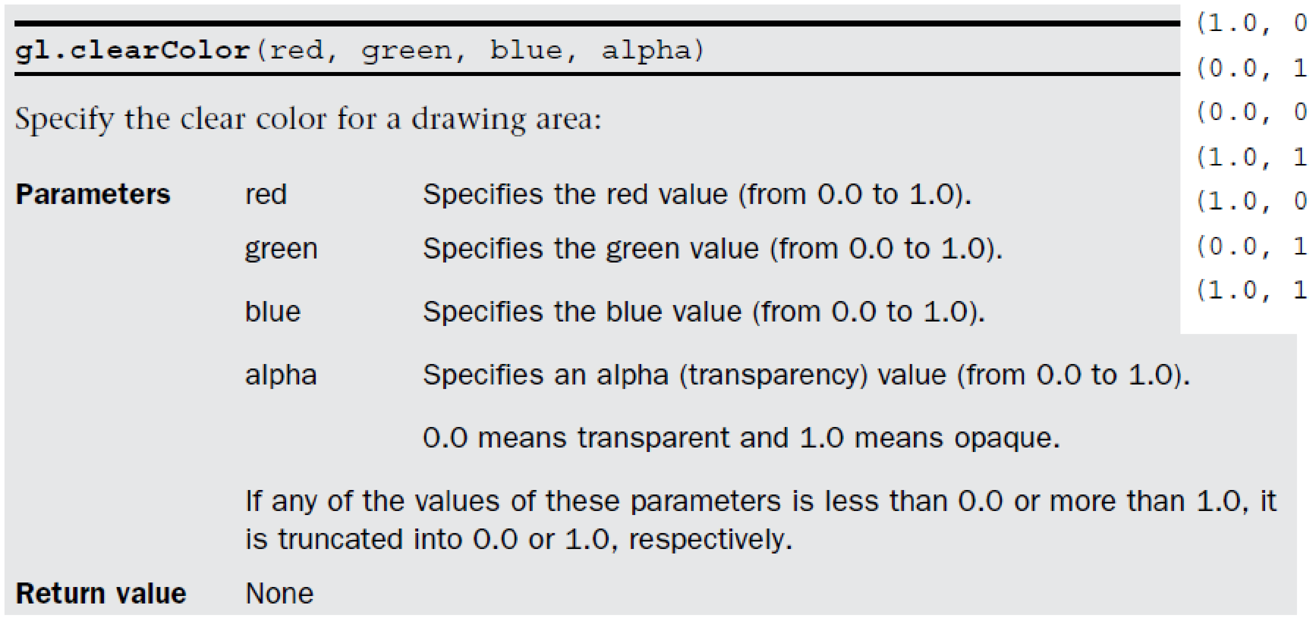
getWebGLContext(element, [, debug])

```
/**
 * Initialize and get the rendering for WebGL
 * @param canvas <cavnas> element
 * @param opt_debug flag to initialize the context for debugging
 * @return the rendering context for WebGL
 */
function getWebGLContext(canvas, opt_debug) {
  // Get the rendering context for WebGL
  var gl = WebGLUtils.setupWebGL(canvas);
  if (!gl) return null;

  // if opt_debug is explicitly false, create the context for debugging
  if (arguments.length < 2 || opt_debug) {
    gl = WebGLDebugUtils.makeDebugContext(gl);
  }

  return gl;
}
```

WebGL

gl.clearColor(red, green, blue, alpha)

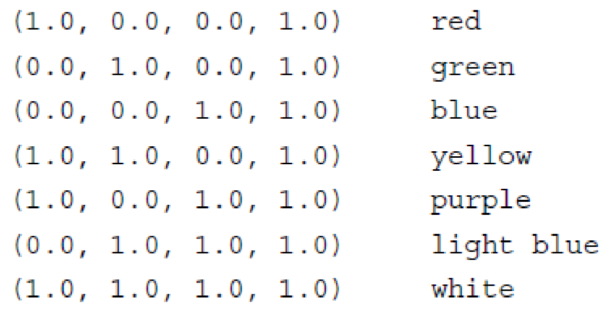
- 도면 영역에 색상 적용



gl.clearColor(red, green, blue, alpha)

Specify the clear color for a drawing area:

Parameters	red	Specifies the red value (from 0.0 to 1.0).
	green	Specifies the green value (from 0.0 to 1.0).
	blue	Specifies the blue value (from 0.0 to 1.0).
	alpha	Specifies an alpha (transparency) value (from 0.0 to 1.0). 0.0 means transparent and 1.0 means opaque.
	If any of the values of these parameters is less than 0.0 or more than 1.0, it is truncated into 0.0 or 1.0, respectively.	
Return value	None	

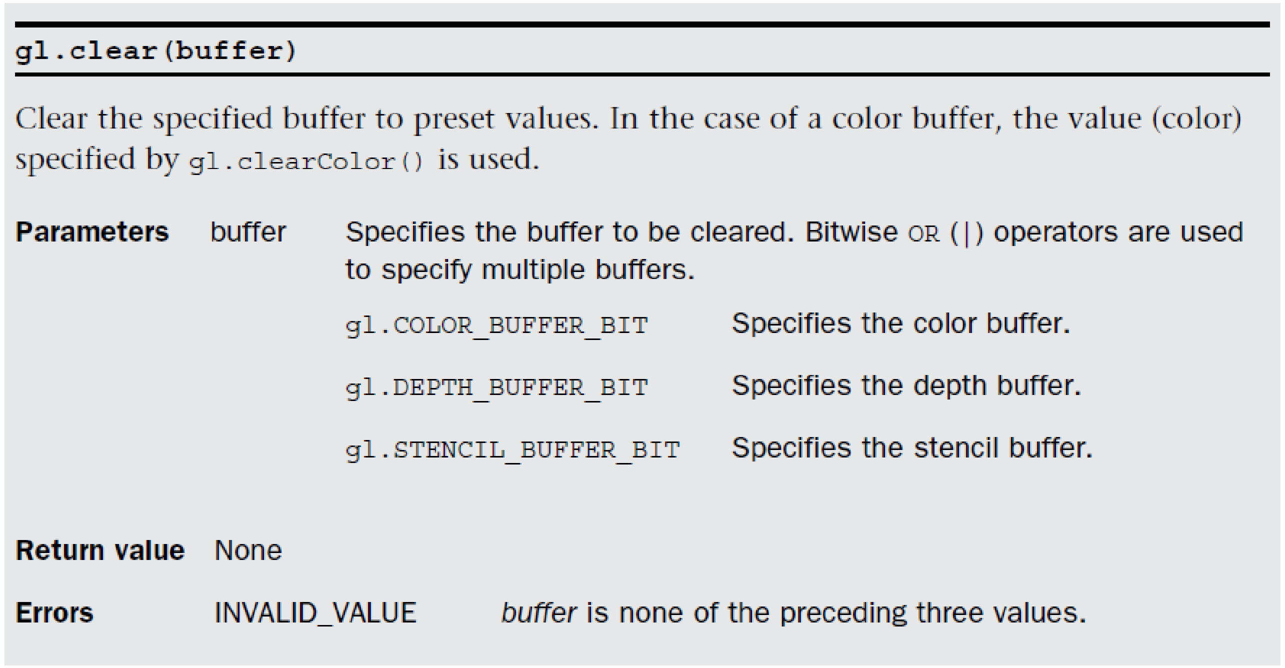


(1.0, 0.0, 0.0, 1.0)	red
(0.0, 1.0, 0.0, 1.0)	green
(0.0, 0.0, 1.0, 1.0)	blue
(1.0, 1.0, 0.0, 1.0)	yellow
(1.0, 0.0, 1.0, 1.0)	purple
(0.0, 1.0, 1.0, 1.0)	light blue
(1.0, 1.0, 1.0, 1.0)	white

WebGL

gl.clear(buffer);

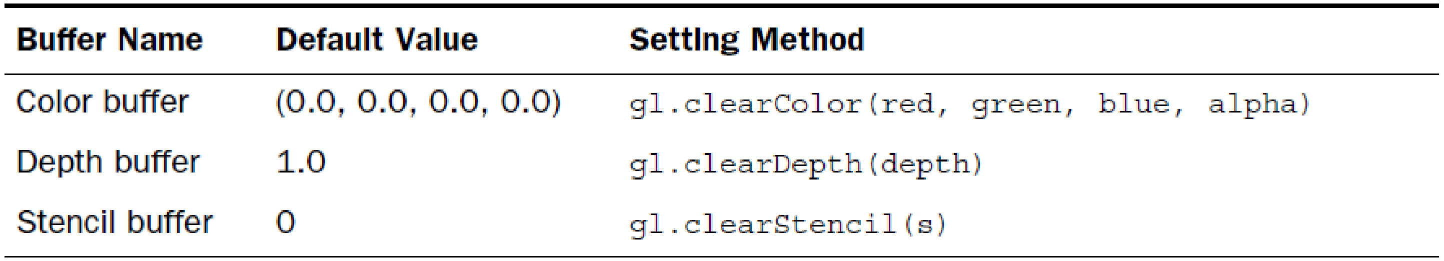
- COLOR_BUFFER_BIT, DEPTH_BUFFER_BIT, STENCIL_BUFFER_BIT



`gl.clear(buffer)`

Clear the specified buffer to preset values. In the case of a color buffer, the value (color) specified by `gl.clearColor()` is used.

Parameters	buffer	Specifies the buffer to be cleared. Bitwise OR (|) operators are used to specify multiple buffers.	
		`gl.COLOR_BUFFER_BIT`	Specifies the color buffer.
		`gl.DEPTH_BUFFER_BIT`	Specifies the depth buffer.
		`gl.STENCIL_BUFFER_BIT`	Specifies the stencil buffer.
Return value	None		
Errors	INVALID_VALUE	*buffer* is none of the preceding three values.	



Buffer Name	Default Value	Setting Method
Color buffer	(0.0, 0.0, 0.0, 0.0)	`gl.clearColor(red, green, blue, alpha)`
Depth buffer	1.0	`gl.clearDepth(depth)`
Stencil buffer	0	`gl.clearStencil(s)`

WebGL

Draw a Point (html)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Draw a point (1)</title>
  </head>

  <body onload="main()">
    <canvas id="webgl" width="400" height="400">
    Please use a browser that supports "canvas"
    </canvas>

    <script src="lib/webgl-utils.js"></script>
    <script src="lib/webgl-debug.js"></script>
    <script src="lib/cuon-utils.js"></script>
    <script src="03 HelloPoint1.js"></script>
  </body>
</html>
```

WebGL

Draw a Point (js)

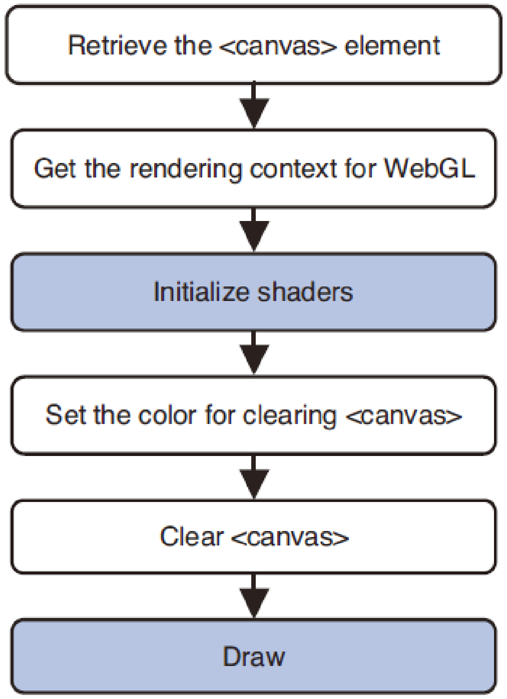
```
// Vertex shader program
var VSHADER_SOURCE =
  'void main() {\n' +
  '  gl_Position = vec4(0.0, 0.0, 0.0, 1.0);\n' + // Set the vertex coordinates of the point
  '  gl_PointSize = 10.0;\n' +                    // Set the point size
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' + // Set the point color
  '}\n';

function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');

  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }}
```

```
// Initialize shaders
if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
  console.log('Failed to intialize shaders.');
  return;
}

// Specify the color for clearing <canvas>
gl.clearColor(0.0, 0.0, 0.0, 1.0);

// Clear <canvas>
gl.clear(gl.COLOR_BUFFER_BIT);

// Draw a point
gl.drawArrays(gl.POINTS, 0, 1);
```

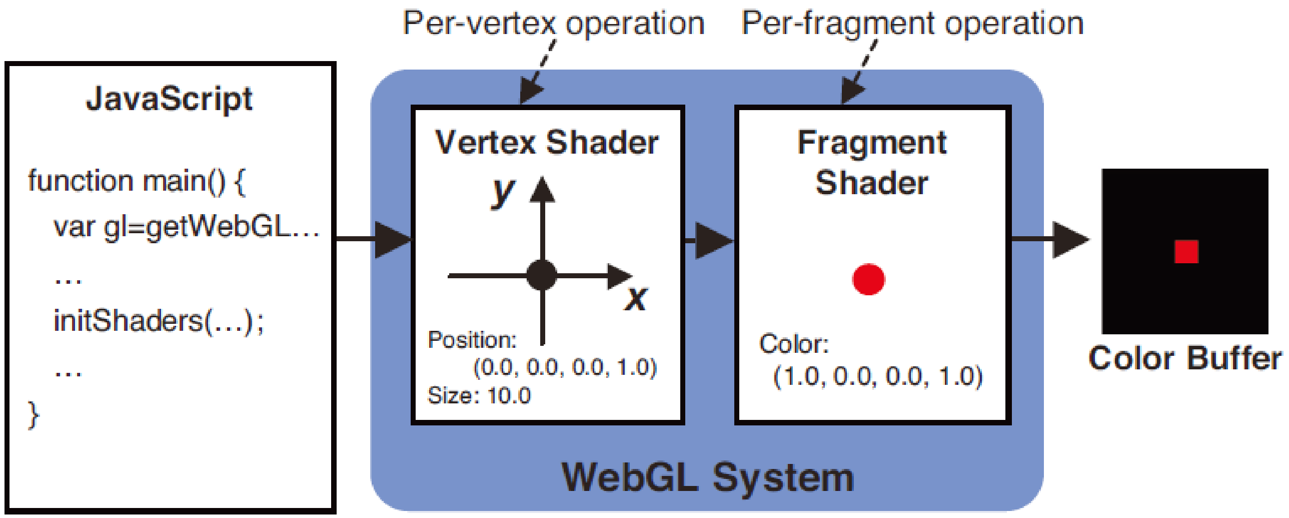
WebGL

Draw a Point

```
gl_Position = vec4(0.0, 0.0, 0.0, 1.0);
gl_PointSize = 10.0;

gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
```



Retrieve the <canvas> element
Get the rendering context for WebGL
Initialize shaders
Set the color for clearing <canvas>
Clear <canvas>
Draw

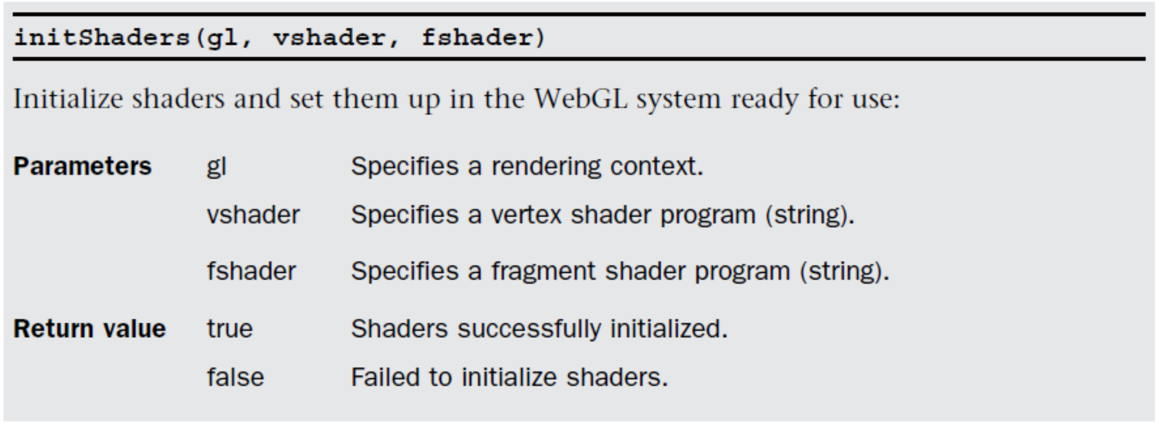


```
function main() {
  var gl=getWebGL…
  …
  initShaders(…);
  …
}
```

WebGL

initShaders(gl, vshader, fshader)

- 참고 교제에서 사용하고 있는 js 파일 함수



`initShaders(gl, vshader, fshader)`

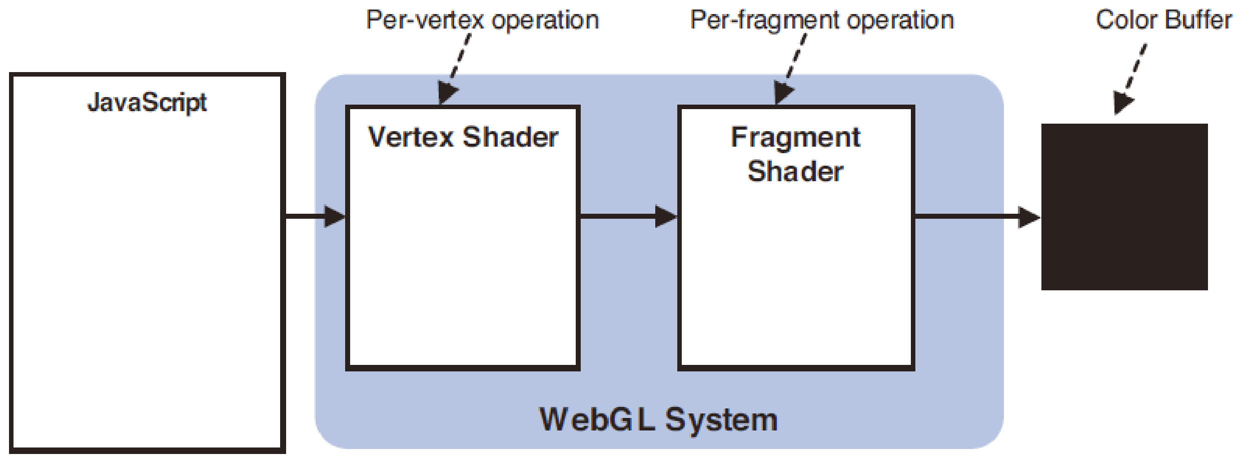
Initialize shaders and set them up in the WebGL system ready for use:

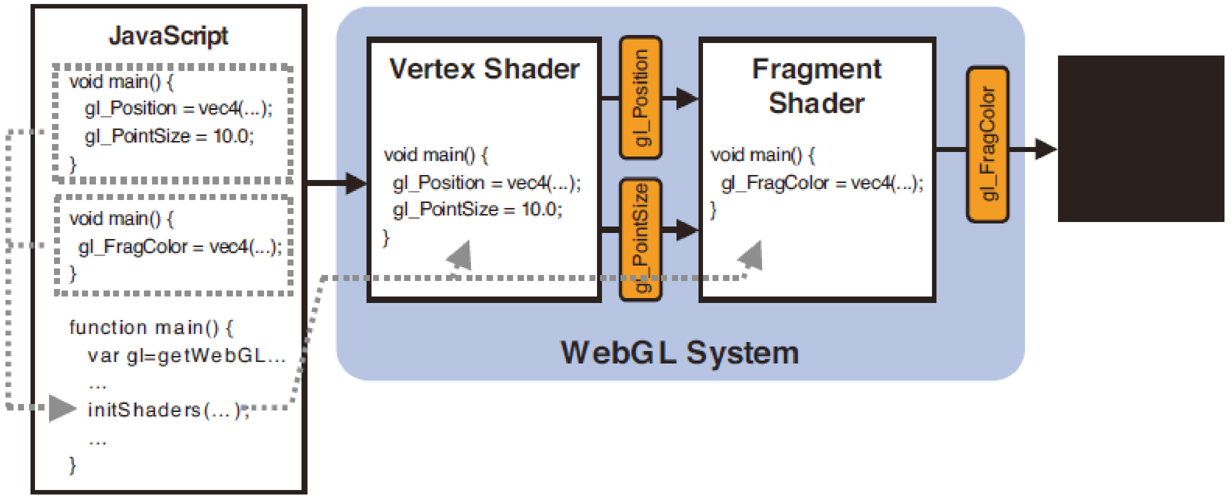
Parameters	gl	Specifies a rendering context.
	vshader	Specifies a vertex shader program (string).
	fshader	Specifies a fragment shader program (string).
Return value	true	Shaders successfully initialized.
	false	Failed to initialize shaders.

```
function initShaders(gl, vshader, fshader) {
  var program = createProgram(gl, vshader, fshader);
  if (!program) {
    console.log('Failed to create program');
    return false;
  }

  gl.useProgram(program);
  gl.program = program;

  return true;
}
```

WebGL

initShaders(gl, vshader, fshader)





```
void main() {
  gl_Position = vec4(...);
  gl_PointSize = 10.0;
}
void main() {
  gl_FragColor = vec4(...);
}
function main() {
  var gl=getWebGL...
  ...
  initShaders(...);
  ...
}
```

```
void main() {
 gl_Position = vec4(...);
 gl_PointSize = 10.0;
}
```

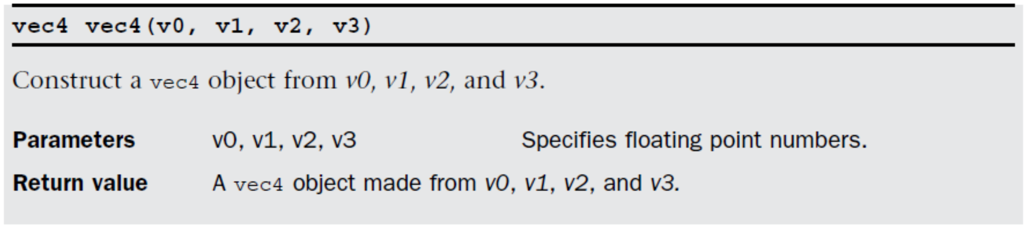
```
void main() {
 gl_FragColor = vec4(...);
}
```

WebGL

Vertex Shader

```
var VSHADER_SOURCE =
  'void main() {\n' +
  '  gl_Position = vec4(0.0, 0.0, 0.0, 1.0);\n' + // Set the vertex coordinates of the point
  '  gl_PointSize = 10.0;\n' +                    // Set the point size
  '}\n';
```

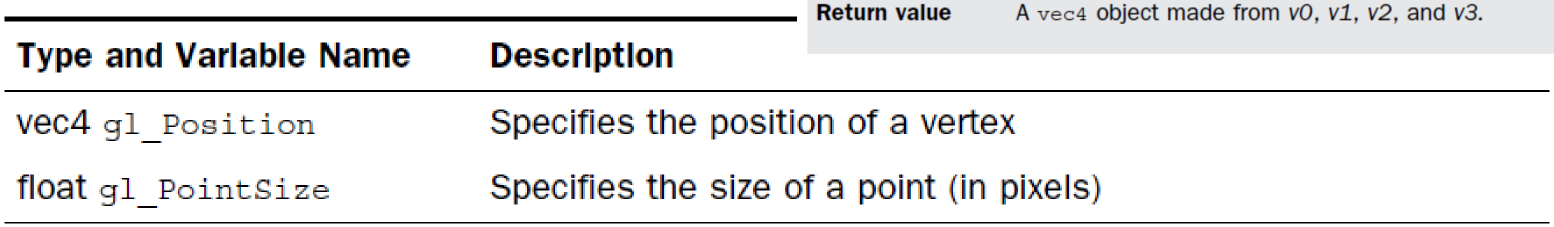
- C 언어와 비슷
- 기본 void main() { }



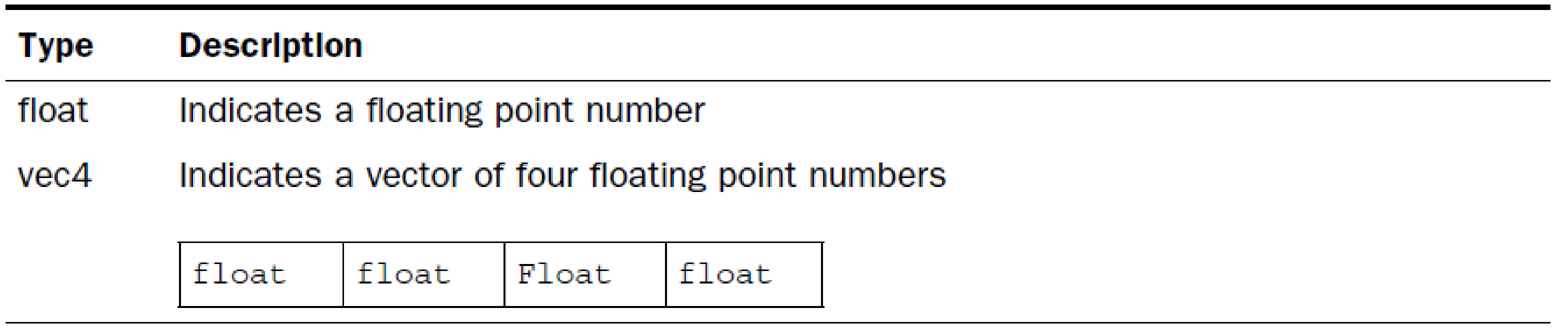
`vec4 vec4(v0, v1, v2, v3)`

Construct a `vec4` object from *v0*, *v1*, *v2*, and *v3*.

Parameters	v0, v1, v2, v3	Specifies floating point numbers.
Return value	A `vec4` object made from *v0*, *v1*, *v2*, and *v3*.	



Type and Variable Name	Description
vec4 `gl_Position`	Specifies the position of a vertex
float `gl_PointSize`	Specifies the size of a point (in pixels)



Type	Description
float	Indicates a floating point number
vec4	Indicates a vector of four floating point numbers

`float`	`float`	`Float`	`float`

WebGL

Fragment Shader

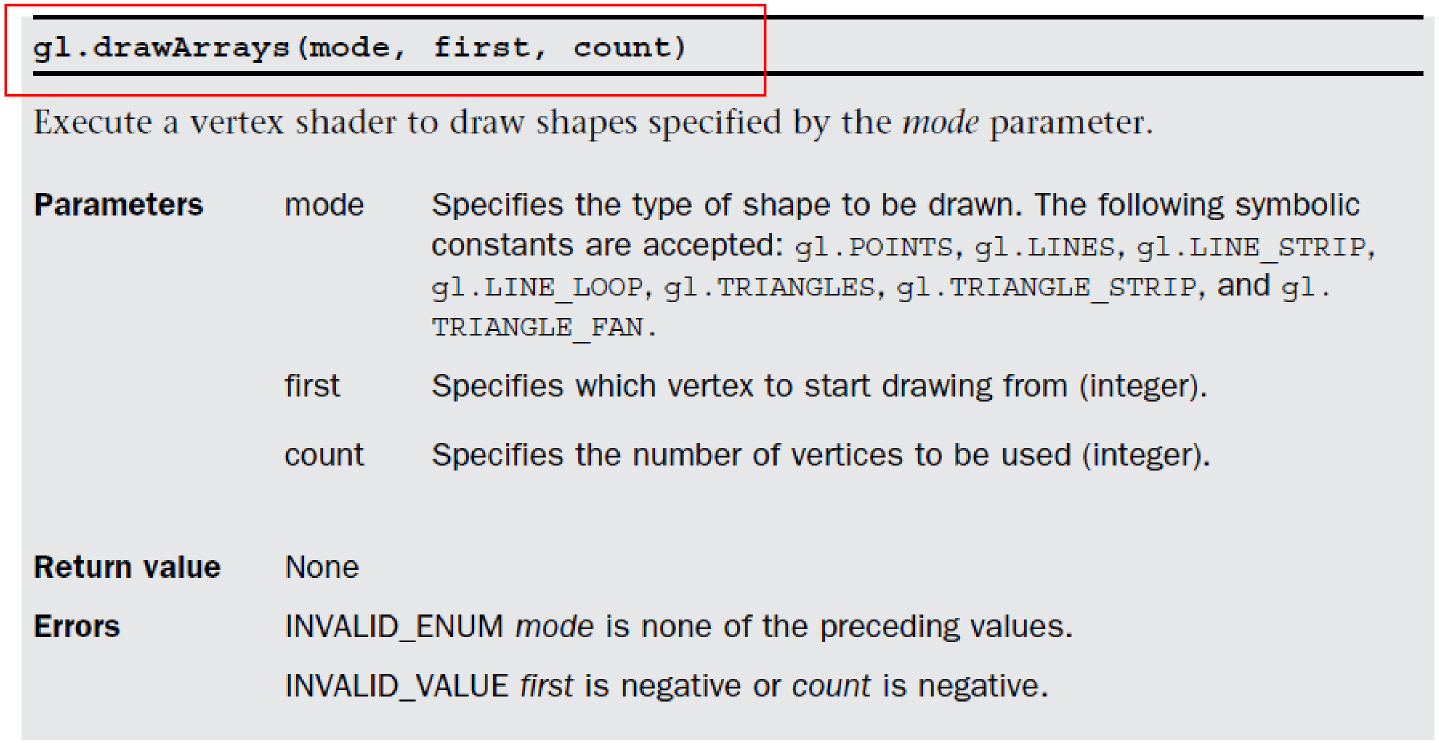
```
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' + // Set the point color
  '}\n';
```

Type and Variable Name	Description
vec4 `gl_FragColor`	Specify the color of a fragment (in RGBA)

WebGL

Draw Operation

```
gl.drawArrays(gl.POINTS, 0, 1);
```



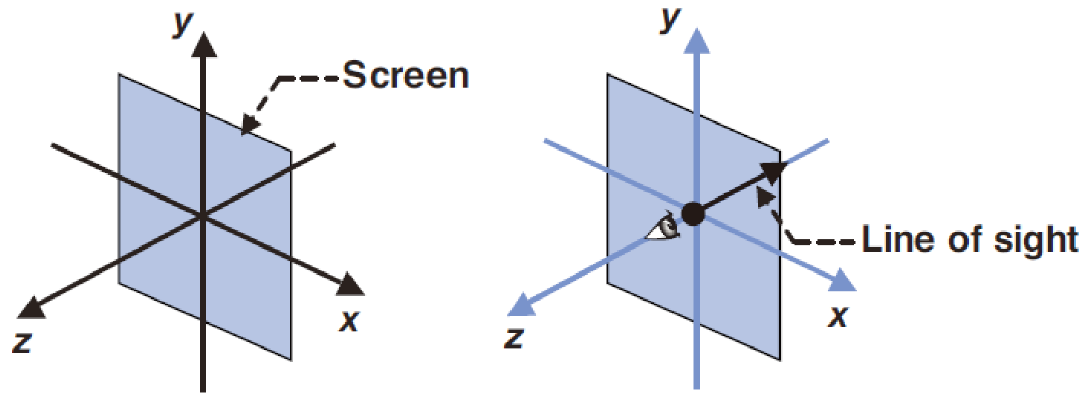
```
gl.drawArrays(mode, first, count)
```

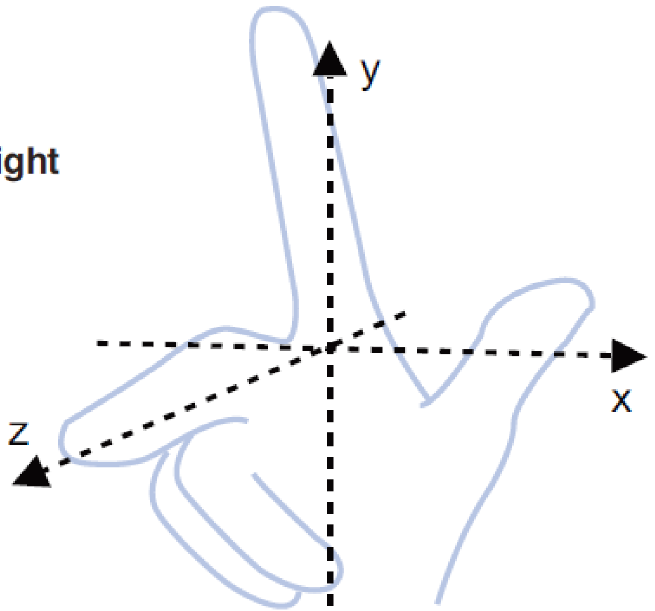
Execute a vertex shader to draw shapes specified by the *mode* parameter.

Parameters	mode	Specifies the type of shape to be drawn. The following symbolic constants are accepted: `gl.POINTS`, `gl.LINES`, `gl.LINE_STRIP`, `gl.LINE_LOOP`, `gl.TRIANGLES`, `gl.TRIANGLE_STRIP`, and `gl.TRIANGLE_FAN`.
	first	Specifies which vertex to start drawing from (integer).
	count	Specifies the number of vertices to be used (integer).
Return value	None	
Errors	INVALID_ENUM *mode* is none of the preceding values.	
	INVALID_VALUE *first* is negative or *count* is negative.	

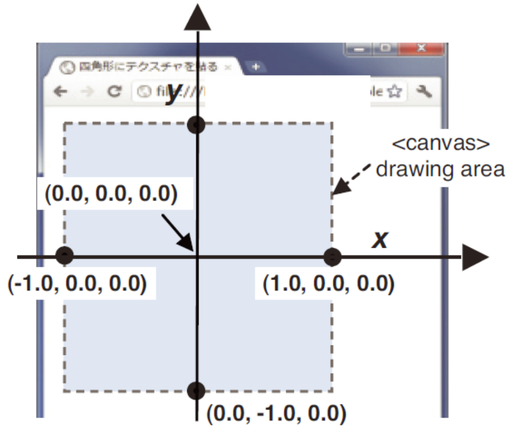
WebGL

The WebGL Coordinate System



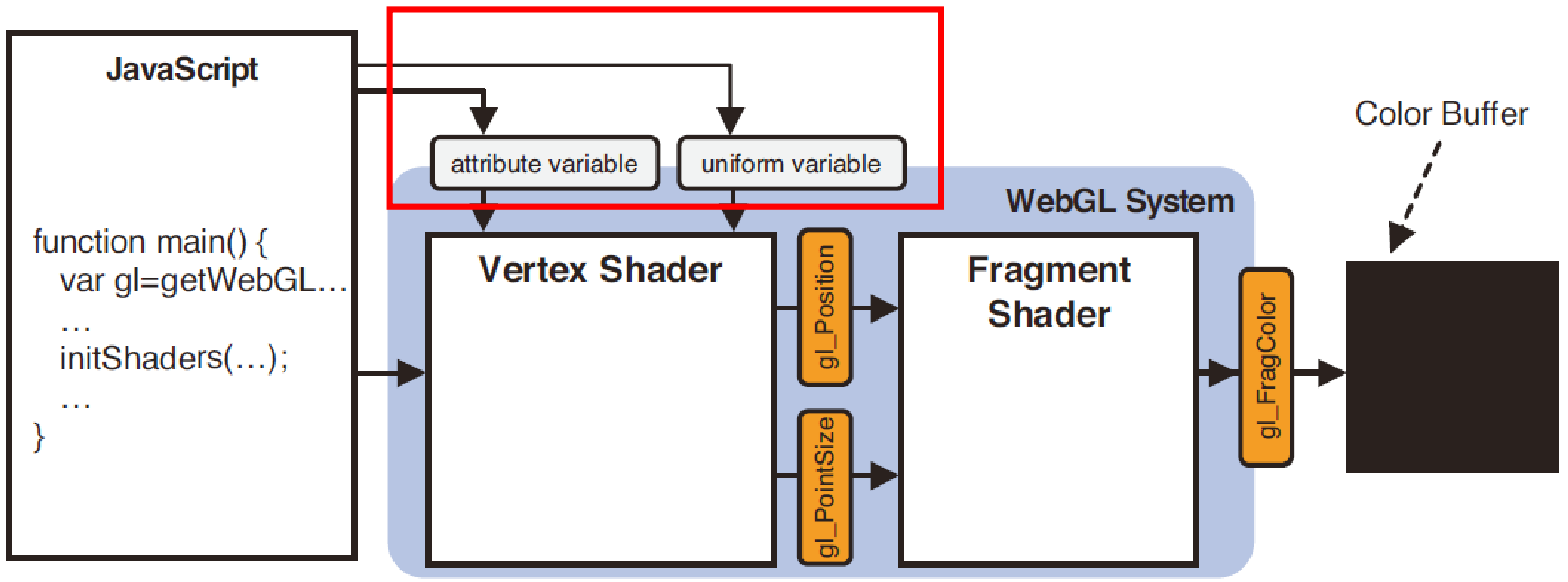


- The center position of a `<canvas>`: (0.0, 0.0, 0.0)
- The two edges of the x-axis of the `<canvas>`: (–1.0, 0.0, 0.0) and (1.0, 0.0, 0.0)
- The two edges of the y-axis of the `<canvas>`: (0.0, –1.0, 0.0) and (0.0, 1.0, 0.0)



WebGL

Attribute Variables



1. Vertext shader 정점 위치에 대한 속성 변수 준비
2. gl_Position 변수에 속성 변수 할당
3. 데이터를 속성 변수에 전달

How to pass data between JavaScript and the shaders: Two ways.

Attribute variable: passing data that differs for each vertex.

passing data from outside a vertex shader into the shader

Uniform variable: passing data that is the same in each vertex

WebGL

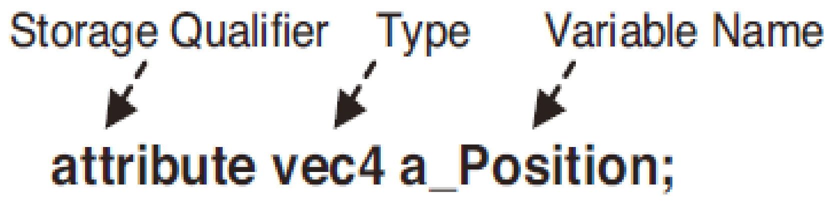
Drawing Point (Attribute Variables)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Hello Point (2)</title>
  </head>

  <body onload="main()">
    <canvas id="webgl" width="400" height="400">
    Please use a browser that supports "canvas"
    </canvas>

    <script src="../lib/webgl-utils.js"></script>
    <script src="../lib/webgl-debug.js"></script>
    <script src="../lib/cuon-utils.js"></script>
    <script src="04 HelloPoint2.js"></script>
  </body>
</html>
```

WebGL

Drawing Point (Attribute Variables)

```
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' + // attribute variable
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '  gl_PointSize = 10.0;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
```



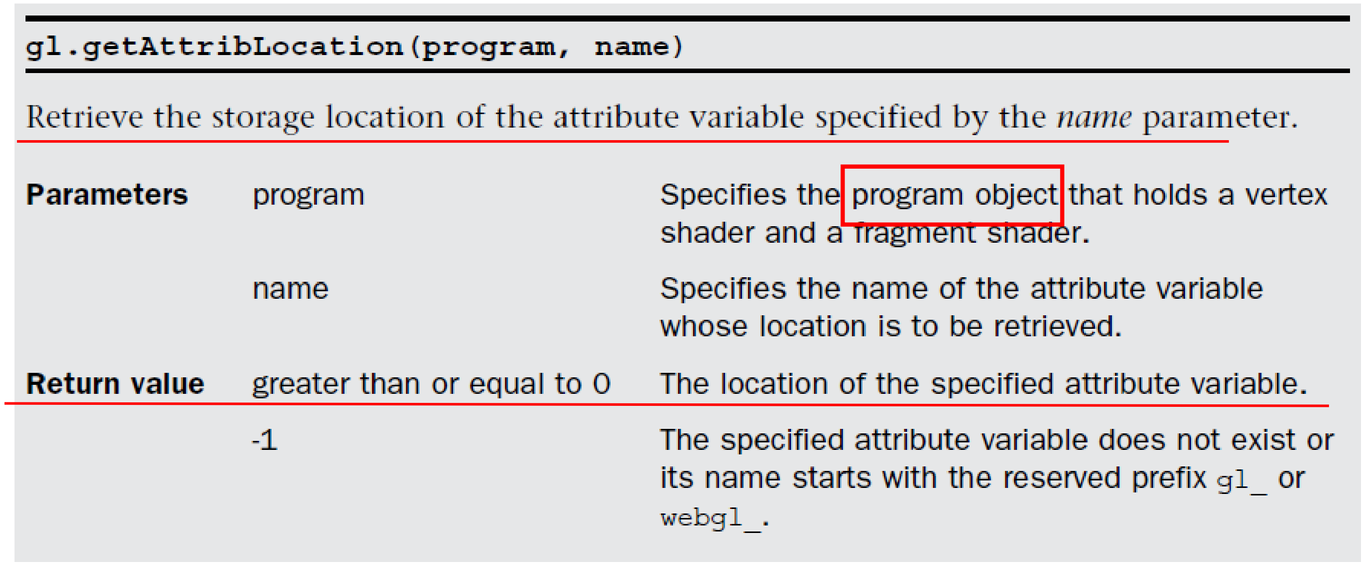
WebGL

Drawing Point (Attribute Variables)

```
function main() {
  var canvas = document.getElementById('webgl');
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
 if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
 var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return;
  }
  gl.vertexAttrib3f(a_Position, 0.0, 0.0, 0.0);
  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);
  gl.drawArrays(gl.POINTS, 0, 1);
}
```

WebGL

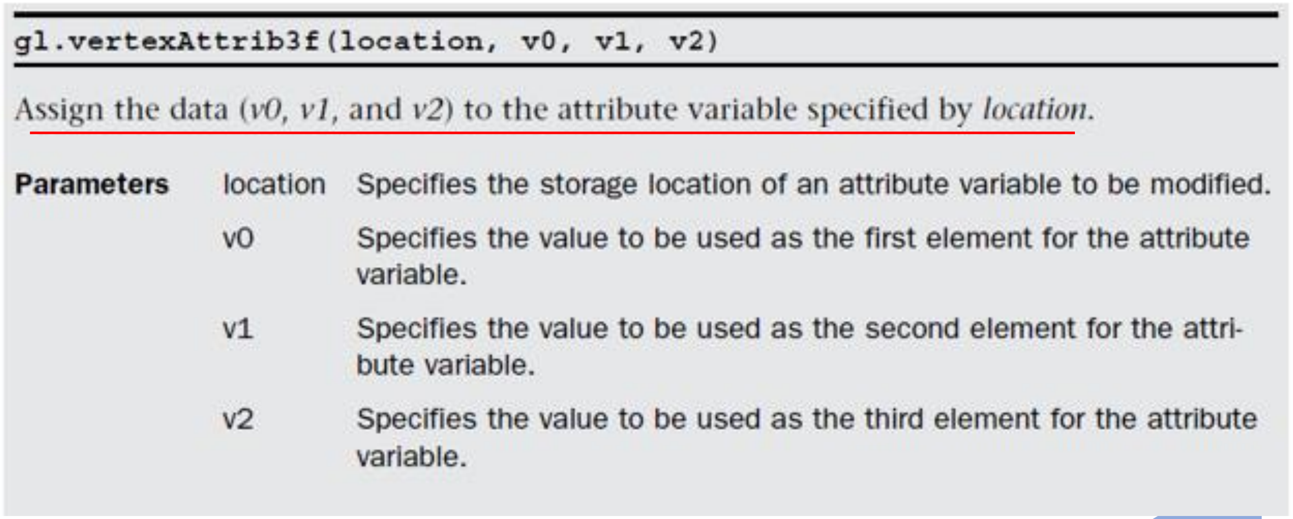
Drawing Point (Attribute Variables)



`gl.getAttribLocation(program, name)`

Retrieve the storage location of the attribute variable specified by the *name* parameter.

Parameters	program	Specifies the program object that holds a vertex shader and a fragment shader.
	name	Specifies the name of the attribute variable whose location is to be retrieved.
Return value	greater than or equal to 0	The location of the specified attribute variable.
	-1	The specified attribute variable does not exist or its name starts with the reserved prefix `gl_` or `webgl_`.



`gl.vertexAttrib3f(location, v0, v1, v2)`

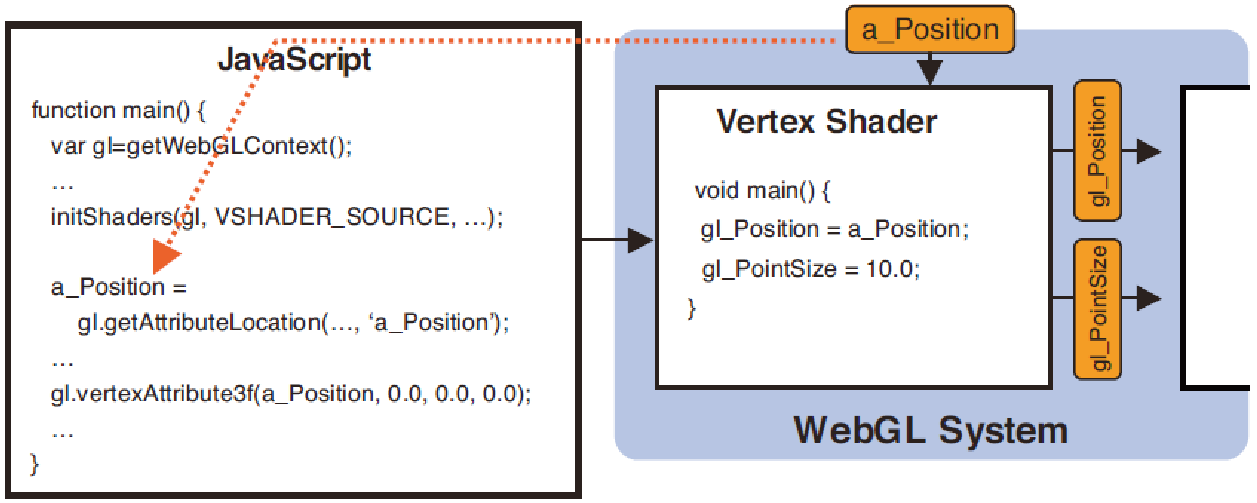
Assign the data (*v0*, *v1*, and *v2*) to the attribute variable specified by *location*.

Parameters	location	Specifies the storage location of an attribute variable to be modified.
	v0	Specifies the value to be used as the first element for the attribute variable.
	v1	Specifies the value to be used as the second element for the attribute variable.
	v2	Specifies the value to be used as the third element for the attribute variable.

WebGL

Drawing Point (Attribute Variables)

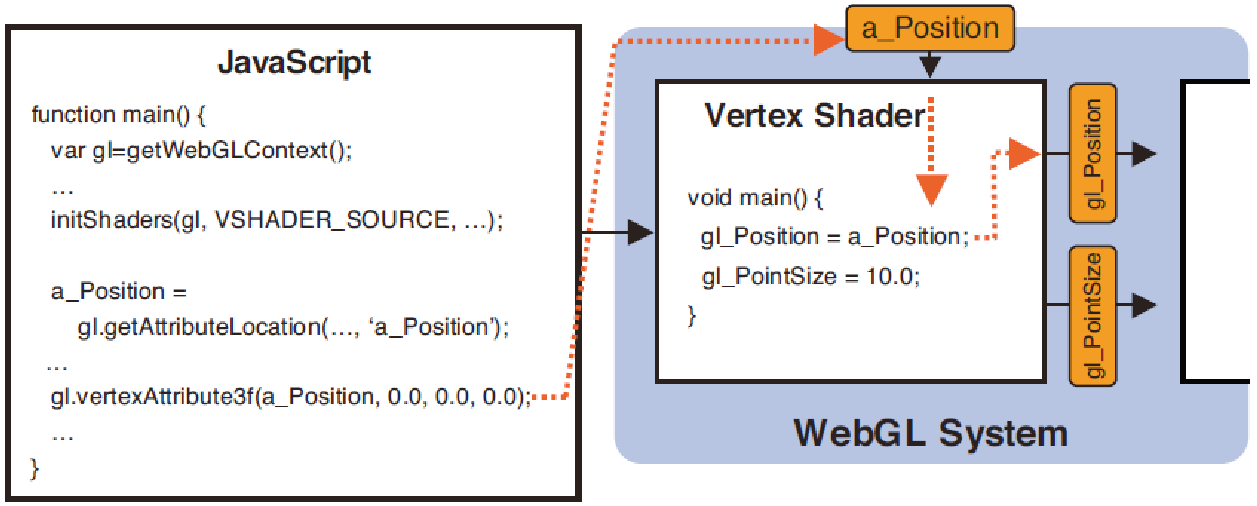
Storage location of an attribute variable



```
JavaScript
function main() {
  var gl=getWebGLContext();
  …
  initShaders(gl, VSHADER_SOURCE, …);

  a_Position =
     gl.getAttributeLocation(…, 'a_Position');
  …
  gl.vertexAttribute3f(a_Position, 0.0, 0.0, 0.0);
  …
}
```

```
Vertex Shader
void main() {
  gl_Position = a_Position;
  gl_PointSize = 10.0;
}
```

Writing a value to the variable

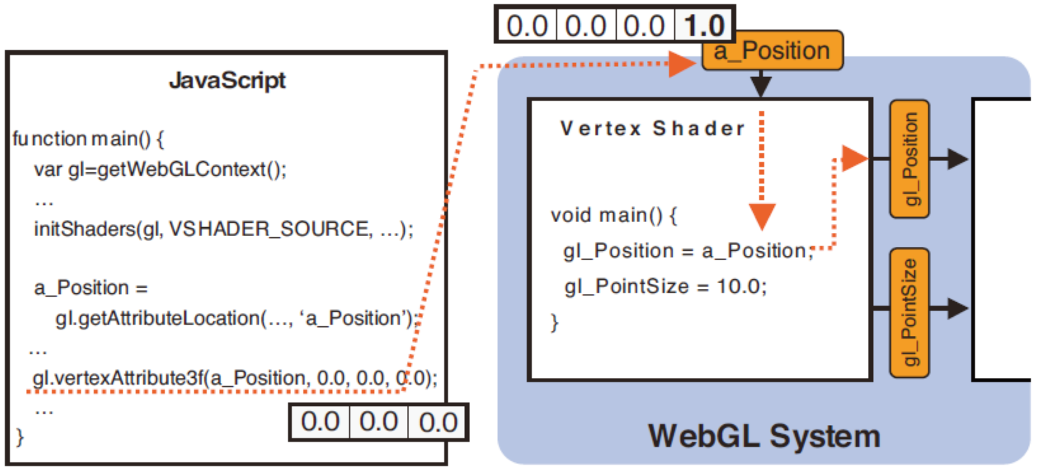


```
JavaScript
function main() {
  var gl=getWebGLContext();
  …
  initShaders(gl, VSHADER_SOURCE, …);

  a_Position =
     gl.getAttributeLocation(…, 'a_Position');
  …
  gl.vertexAttribute3f(a_Position, 0.0, 0.0, 0.0);
  …
}
```

```
Vertex Shader
void main() {
  gl_Position = a_Position;
  gl_PointSize = 10.0;
}
```

WebGL

Drawing Point (Attribute Variables)



0.0 | 0.0 | 0.0 | 1.0
a_Position
JavaScript
function main() {
var gl=getWebGLContext();
...
initShaders(gl, VSHADER_SOURCE, ...);
a_Position =
gl.getAttributeLocation(..., 'a_Position');
...
gl.vertexAttribute3f(a_Position, 0.0, 0.0, 0.0);
...
}
0.0 | 0.0 | 0.0
Vertex Shader
void main() {
gl_Position = a_Position;
gl_PointSize = 10.0;
}
gl_Position
gl_PointSize
WebGL System

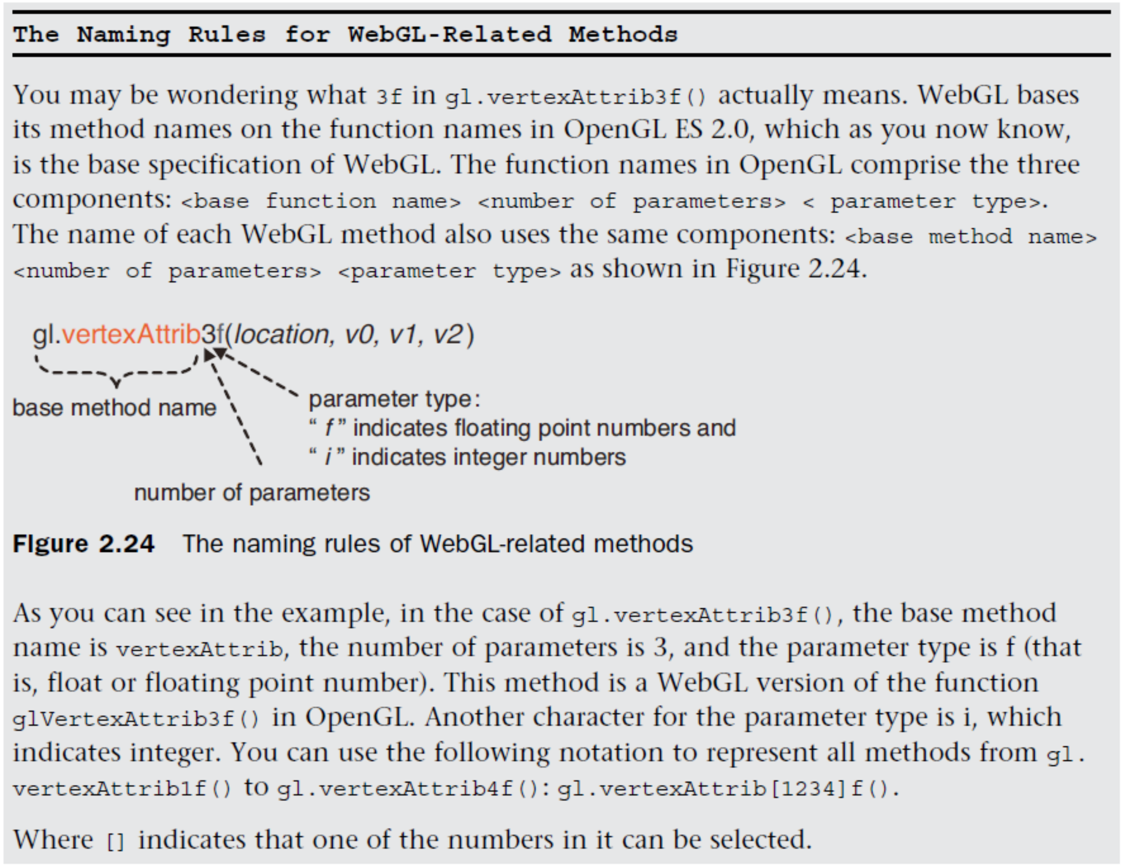
```
gl.vertexAttrib1f(location, v0)
gl.vertexAttrib2f(location, v0, v1)
gl.vertexAttrib3f(location, v0, v1, v2)
gl.vertexAttrib4f(location, v0, v1, v2, v3)
```

(v0, 0.0, 0.0, 1.0)

Parameters	location	Specifies the storage location of the attribute variable.
	v0, v1, v2, v3	Specifies the values to be assigned to the first, second, third, and fourth components of the attribute variable.

WebGL

Drawing Point (Attribute Variables)



The Naming Rules for WebGL-Related Methods

You may be wondering what `3f` in `gl.vertexAttrib3f()` actually means. WebGL bases its method names on the function names in OpenGL ES 2.0, which as you now know, is the base specification of WebGL. The function names in OpenGL comprise the three components: `<base function name> <number of parameters> < parameter type>`. The name of each WebGL method also uses the same components: `<base method name> <number of parameters> <parameter type>` as shown in Figure 2.24.

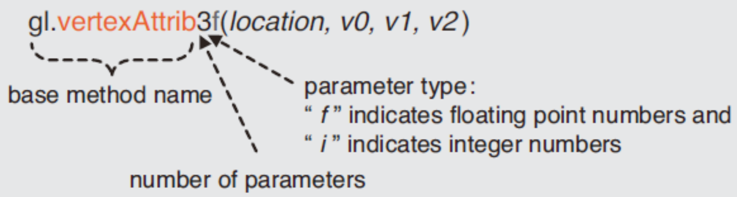
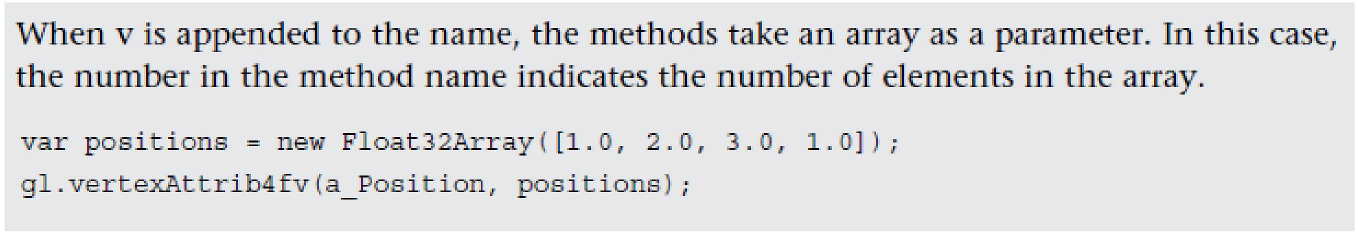

gl.vertexAttrib3f(location, v0, v1, v2)
base method name
parameter type:
" f " indicates floating point numbers and
" i " indicates integer numbers
number of parameters

Figure 2.24 The naming rules of WebGL-related methods

As you can see in the example, in the case of `gl.vertexAttrib3f()`, the base method name is `vertexAttrib`, the number of parameters is 3, and the parameter type is f (that is, float or floating point number). This method is a WebGL version of the function `glVertexAttrib3f()` in OpenGL. Another character for the parameter type is i, which indicates integer. You can use the following notation to represent all methods from `gl.vertexAttrib1f()` to `gl.vertexAttrib4f()`: `gl.vertexAttrib[1234]f()`.

Where `[]` indicates that one of the numbers in it can be selected.



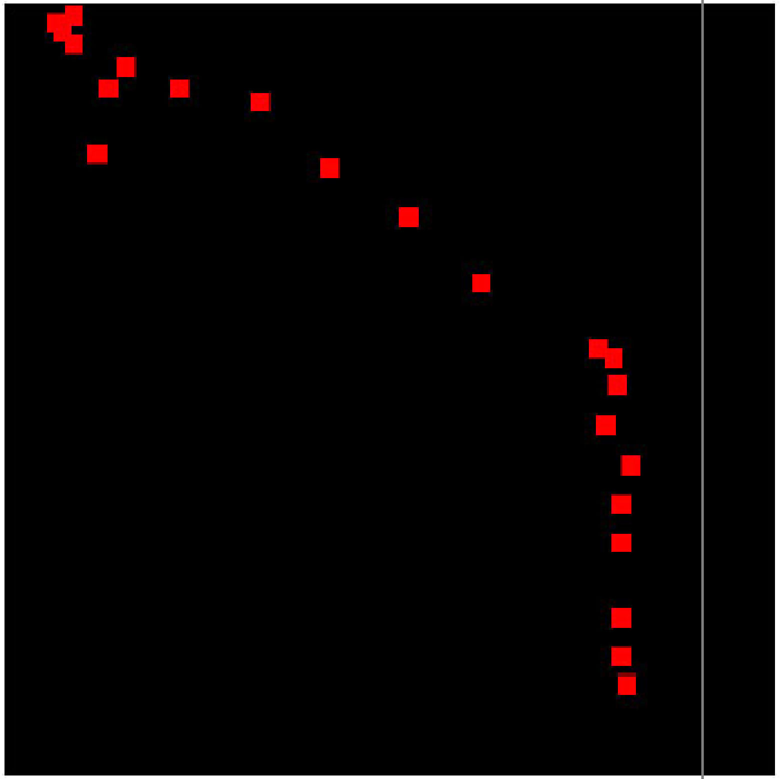
When v is appended to the name, the methods take an array as a parameter. In this case, the number in the method name indicates the number of elements in the array.

```
var positions = new Float32Array([1.0, 2.0, 3.0, 1.0]);
gl.vertexAttrib4fv(a_Position, positions);
```

WebGL

Draw a Point with a Mouse Click

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Draw a point with a mouse click</title>
  </head>

  <body onload="main()">
    <canvas id="webgl" width="400" height="400">
    Please use a browser that supports "canvas"
    </canvas>

    <script src="../lib/webgl-utils.js"></script>
    <script src="../lib/webgl-debug.js"></script>
    <script src="../lib/cuon-utils.js"></script>
    <script src="05 ClickedPoints.js"></script>
  </body>
</html>
```



WebGL

Draw a Point with a Mouse Click

```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '  gl_PointSize = 10.0;\n' +
  '}\n';

var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n’;

function main() {
  // Retrieve <canvas> element
  var canvas =
document.getElementById('webgl');

  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering
context for WebGL');
    return;
  }
```

```
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }

  // // Get the storage location of a_Position
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return;
  }
  // Register function (event handler) to be called on a mouse
press
  canvas.onmousedown = function(ev){
       click(ev, gl, canvas, a_Position);
  };

  // Specify the color for clearing <canvas>
  gl.clearColor(0.0, 0.0, 0.0, 1.0);

  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
}
```

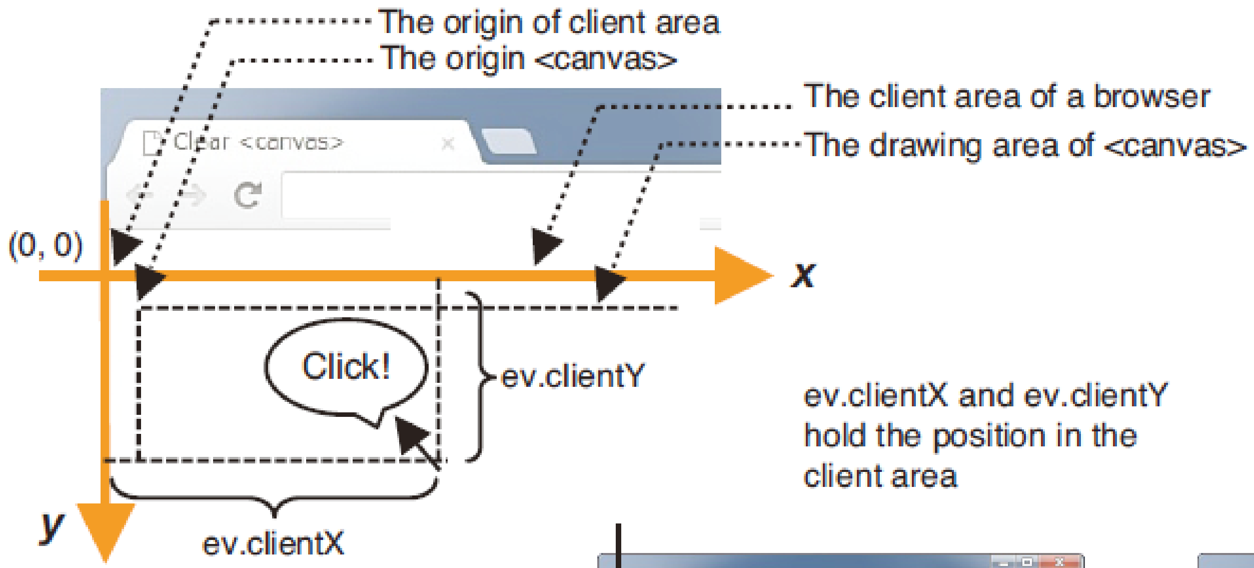
WebGL

Draw a Point with a Mouse Click

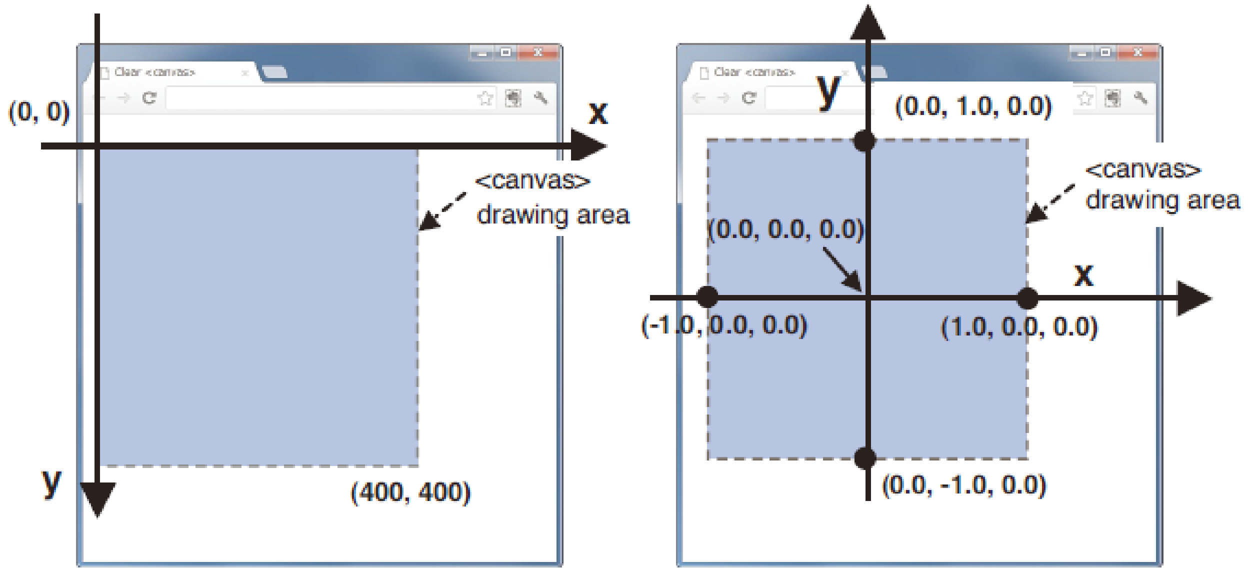
```
var g_points = []; // The array for the position of a mouse press
function click(ev, gl, canvas, a_Position) {
  var x = ev.clientX; // x coordinate of a mouse pointer
  var y = ev.clientY; // y coordinate of a mouse pointer
  var rect = ev.target.getBoundingClientRect() ;

  x = ((x - rect.left) - canvas.width/2)/(canvas.width/2);
  y = (canvas.height/2 - (y - rect.top))/(canvas.height/2);
  // Store the coordinates to g_points array
  g_points.push(x); g_points.push(y);

  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);

  var len = g_points.length;
  for(var i = 0; i < len; i += 2) {
    // Pass the position of a point to a_Position variable
    gl.vertexAttrib3f(a_Position, g_points[i], g_points[i+1], 0.0);

    // Draw
    gl.drawArrays(gl.POINTS, 0, 1);
  }
}
```

WebGL

이벤트 핸들러


The origin of client area
The origin <canvas>
The client area of a browser
The drawing area of <canvas>
Clear <canvas>
(0, 0)
x
y
Click!
ev.clientY
ev.clientX

ev.clientX and ev.clientY hold the position in the client area



WebGL

이벤트 핸들러

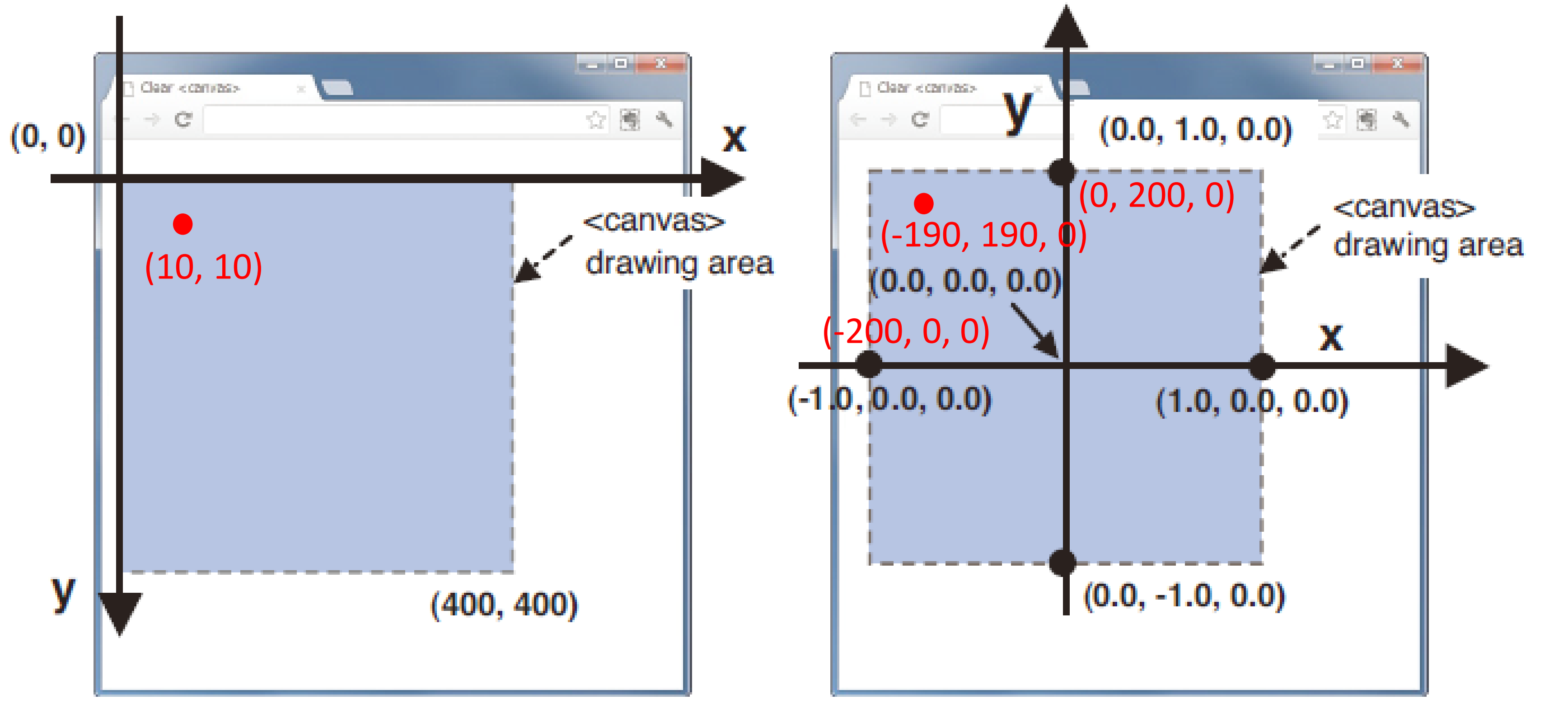
```
var g_points = []; // The array for the position of a mouse press
function click(ev, gl, canvas, a_Position) {
  var x = ev.clientX; // x coordinate of a mouse pointer
  var y = ev.clientY; // y coordinate of a mouse pointer
  var rect = ev.target.getBoundingClientRect() ;

  x = ((x - rect.left) - canvas.width/2)/(canvas.width/2);
  y = (canvas.height/2 - (y - rect.top))/(canvas.height/2);
  // Store the coordinates to g_points array
  g_points.push(x); g_points.push(y);

  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);

  var len = g_points.length;
  for(var i = 0; i < len; i += 2) {
    // Pass the position of a point to a_Position variable
    gl.vertexAttrib3f(a_Position, g_points[i], g_points[i+1], 0.0);

    // Draw
    gl.drawArrays(gl.POINTS, 0, 1);
  }
}
```

Get the position of the <canvas> in the client area

Move the origin of <canvas> into the origin of the WebGL
Range: -1.0 ~ +1.0

WebGL

이벤트 핸들러

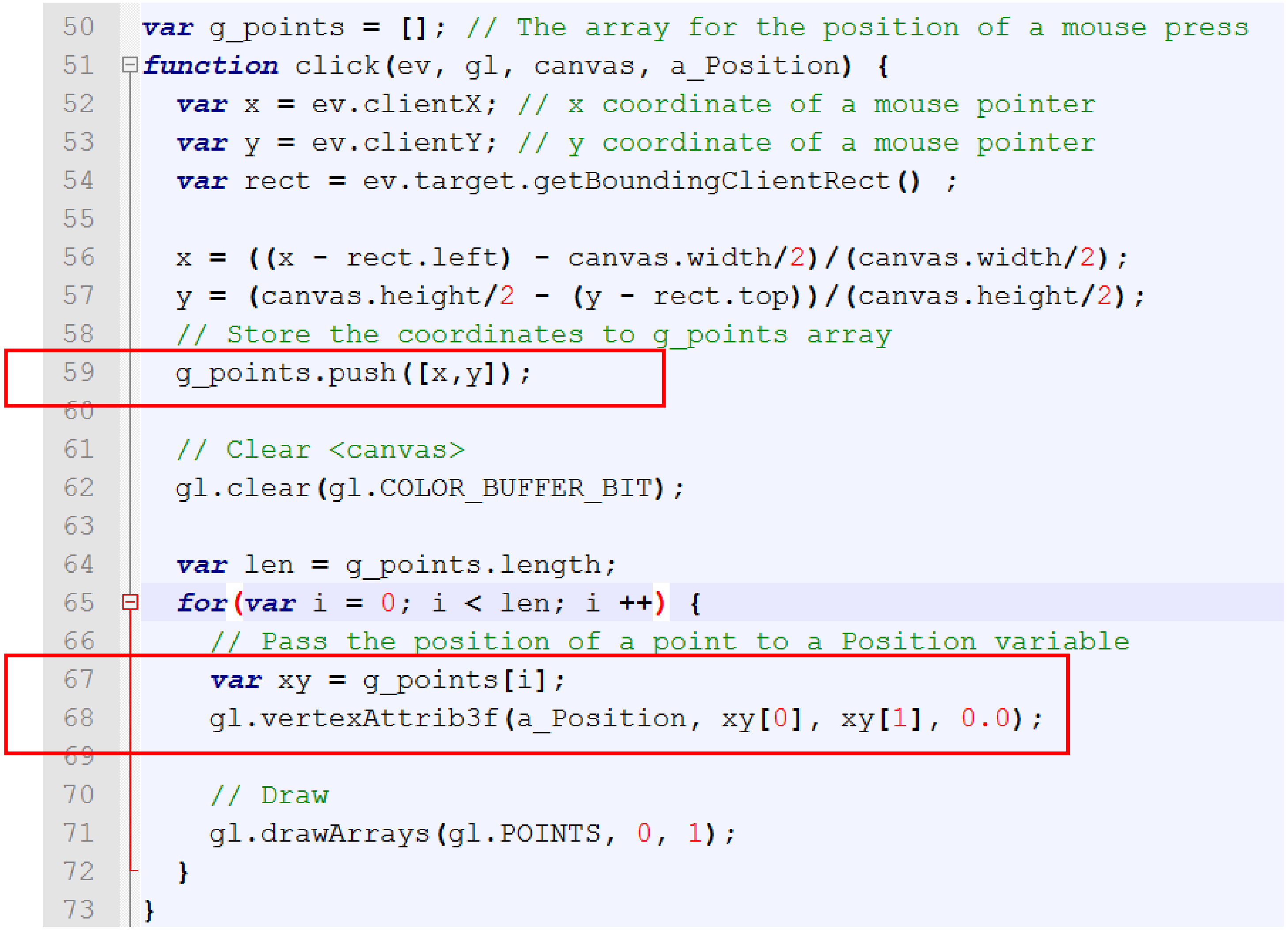


```
x = ((x - rect.left) - canvas.width/2)/(canvas.width/2);
y = (canvas.height/2 - (y - rect.top))/(canvas.height/2)
```

X=((10-0)-200)/200 = -0.95
Y=(200-(10-0))/200 = 0.95

WebGL

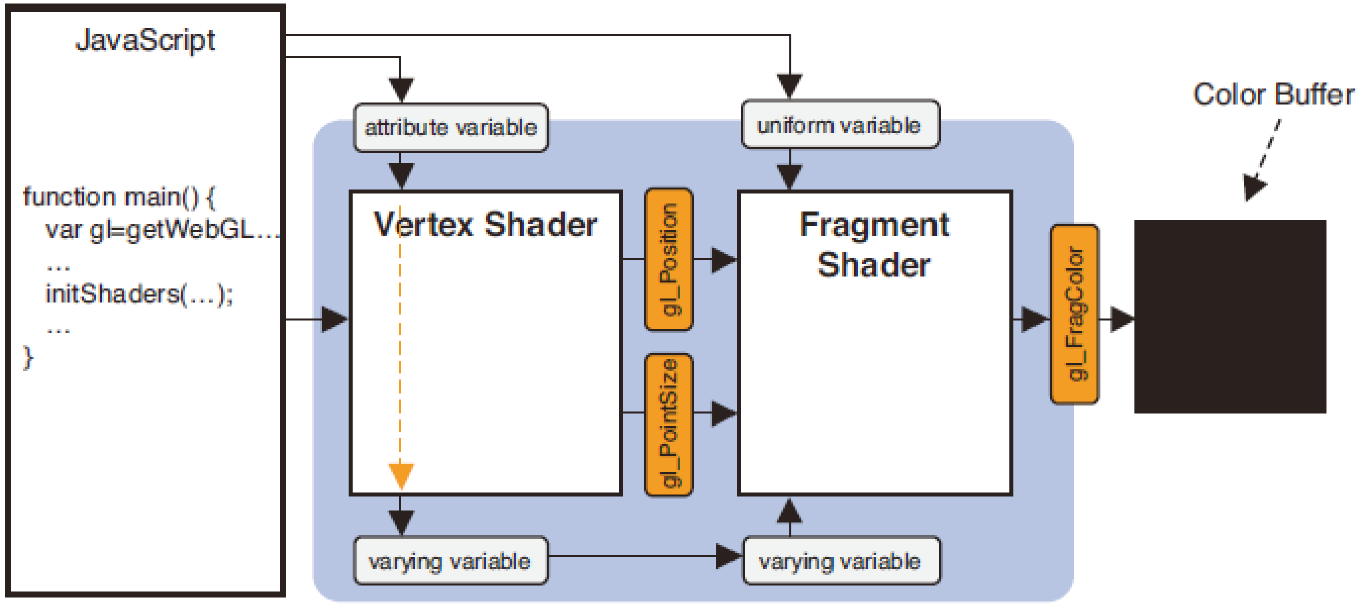
이벤트 핸들러



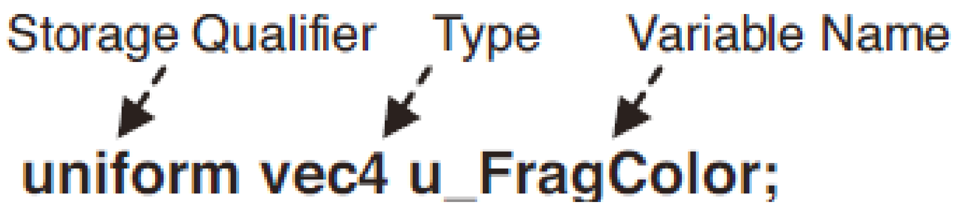
```
var g_points = []; // The array for the position of a mouse press
function click(ev, gl, canvas, a_Position) {
  var x = ev.clientX; // x coordinate of a mouse pointer
  var y = ev.clientY; // y coordinate of a mouse pointer
  var rect = ev.target.getBoundingClientRect() ;

  x = ((x - rect.left) - canvas.width/2)/(canvas.width/2);
  y = (canvas.height/2 - (y - rect.top))/(canvas.height/2);
  // Store the coordinates to g_points array
  g_points.push([x,y]);

  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);

  var len = g_points.length;
  for(var i = 0; i < len; i ++) {
    // Pass the position of a point to a Position variable
    var xy = g_points[i];
    gl.vertexAttrib3f(a_Position, xy[0], xy[1], 0.0);

    // Draw
    gl.drawArrays(gl.POINTS, 0, 1);
  }
}
```

WebGL

Change the Point Color



1. Fragment shader에 uniform 변수 준비
2. gl_FragColor 변수에 할당
3. JavaScript에서 색상 데이터 전달

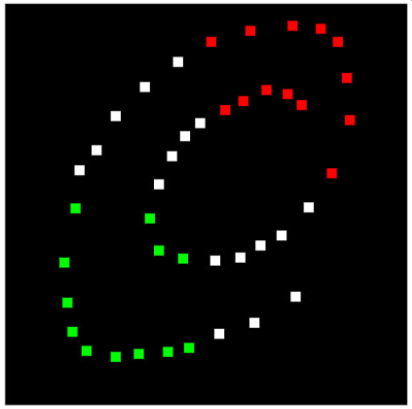


```
uniform vec4 u_FragColor;
```

WebGL

Change the Point Color

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Change a point color</title>
  </head>

  <body onload="main()">
    <canvas id="webgl" width="400" height="400">
    Please use a browser that supports "canvas"
    </canvas>

    <script src="../lib/webgl-utils.js"></script>
    <script src="../lib/webgl-debug.js"></script>
    <script src="../lib/cuon-utils.js"></script>
    <script src="06 ColoredPoints.js"></script>
  </body>
</html>
```



WebGL

Change the Point Color

```
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '  gl_PointSize = 10.0;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  'precision mediump float;\n' +
  'uniform vec4 u_FragColor;\n' +
  'void main() {\n' +
  '  gl_FragColor = u_FragColor;\n' +
  '}\n';
```

```
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');

  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }

  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }

  // // Get the storage location of a_Position
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return;
  }
```

WebGL

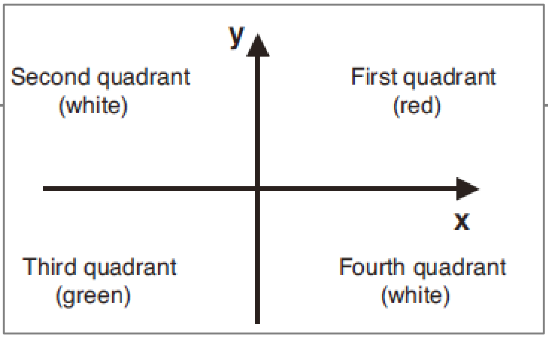
Change the Point Color

```
// Get the storage location of u_FragColor
 var u_FragColor = gl.getUniformLocation(gl.program, 'u_FragColor');
 if (!u_FragColor) {
   console.log('Failed to get the storage location of u_FragColor');
   return;
 }

 // Register function (event handler) to be called on a mouse press
 canvas.onmousedown = function(ev){ click(ev, gl, canvas, a_Position, u_FragColor) };

 // Specify the color for clearing <canvas>
 gl.clearColor(0.0, 0.0, 0.0, 1.0);

 // Clear <canvas>
 gl.clear(gl.COLOR_BUFFER_BIT);
}
```

WebGL

Change the Point Color

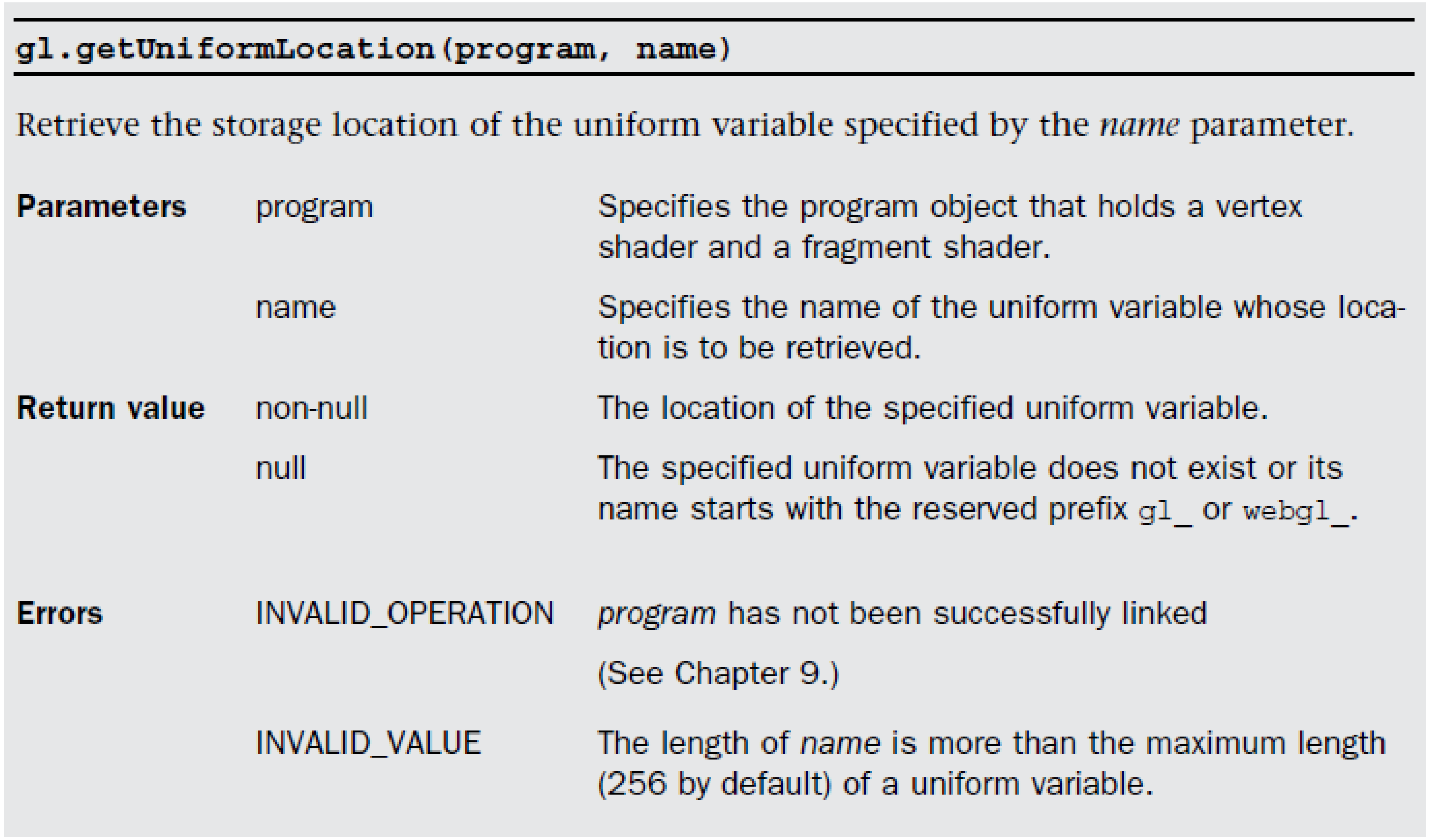
```
var g_points = [];
var g_colors = [];
function click(ev, gl, canvas, a_Position,
u_FragColor) {
 var x = ev.clientX;
 var y = ev.clientY;
 var rect = ev.target.getBoundingClientRect();

 x = ((x - rect.left) -
canvas.width/2)/(canvas.width/2);
 y = (canvas.height/2 - (y -
rect.top))/(canvas.height/2);

 // Store the coordinates to g_points array
 g_points.push([x, y]);
 // Store the coordinates to g_points array
 if (x >= 0.0 && y >= 0.0)
  g_colors.push([1.0, 0.0, 0.0, 1.0]);  // Red
 } else if (x < 0.0 && y < 0.0)
  g_colors.push([0.0, 1.0, 0.0, 1.0]);  // Green
 } else {                      // Others
  g_colors.push([1.0, 1.0, 1.0, 1.0]);  // White
 }
```



```
// Clear <canvas>
gl.clear(gl.COLOR_BUFFER_BIT);

var len = g_points.length;
for(var i = 0; i < len; i++) {
 var xy = g_points[i];
 var rgba = g_colors[i];


 gl.vertexAttrib3f(a_Position, xy[0], xy[1], 0.0);

 gl.uniform4f(u_FragColor, rgba[0], rgba[1], rgba[2], rgba[3]);
 // Draw
 gl.drawArrays(gl.POINTS, 0, 1);
 }
}
```

WebGL

getUniformLocation(program, name)



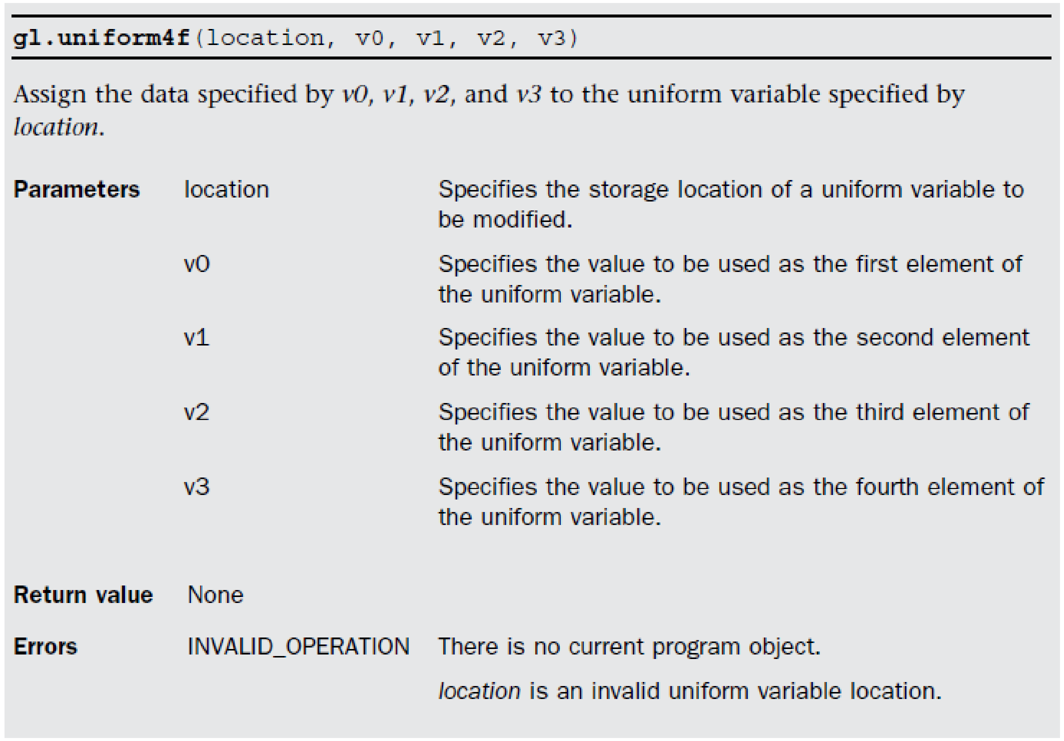
gl.getUniformLocation(program, name)

Retrieve the storage location of the uniform variable specified by the *name* parameter.

Parameters	program	Specifies the program object that holds a vertex shader and a fragment shader.
	name	Specifies the name of the uniform variable whose location is to be retrieved.
Return value	non-null	The location of the specified uniform variable.
	null	The specified uniform variable does not exist or its name starts with the reserved prefix `gl_` or `webgl_`.
Errors	INVALID_OPERATION	*program* has not been successfully linked (See Chapter 9.)
	INVALID_VALUE	The length of *name* is more than the maximum length (256 by default) of a uniform variable.

WebGL

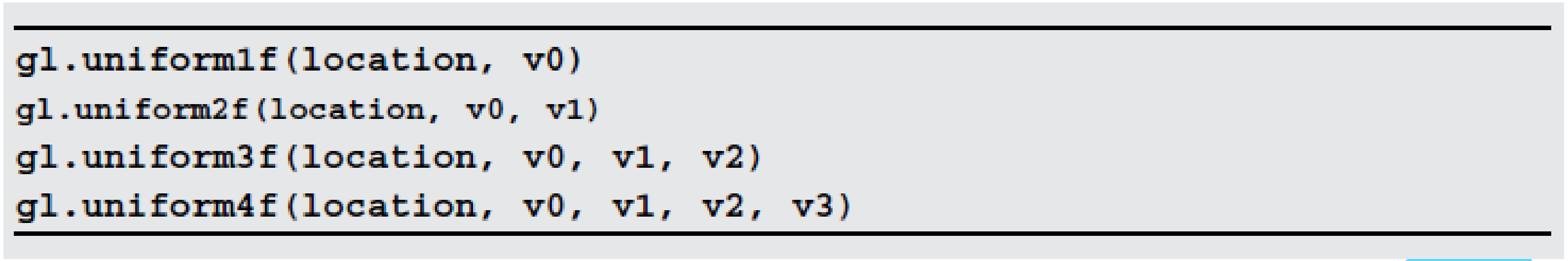
gl.uniform4f



```
gl.uniform4f(location, v0, v1, v2, v3)
```

Assign the data specified by *v0*, *v1*, *v2*, and *v3* to the uniform variable specified by *location*.

Parameters	location	Specifies the storage location of a uniform variable to be modified.
	v0	Specifies the value to be used as the first element of the uniform variable.
	v1	Specifies the value to be used as the second element of the uniform variable.
	v2	Specifies the value to be used as the third element of the uniform variable.
	v3	Specifies the value to be used as the fourth element of the uniform variable.
Return value	None	
Errors	INVALID_OPERATION	There is no current program object.
		location is an invalid uniform variable location.



```
gl.uniform1f(location, v0)
gl.uniform2f(location, v0, v1)
gl.uniform3f(location, v0, v1, v2)
gl.uniform4f(location, v0, v1, v2, v3)
```