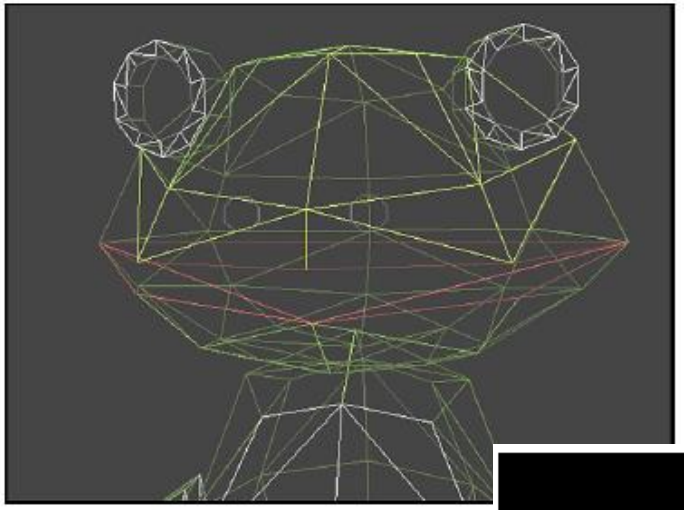
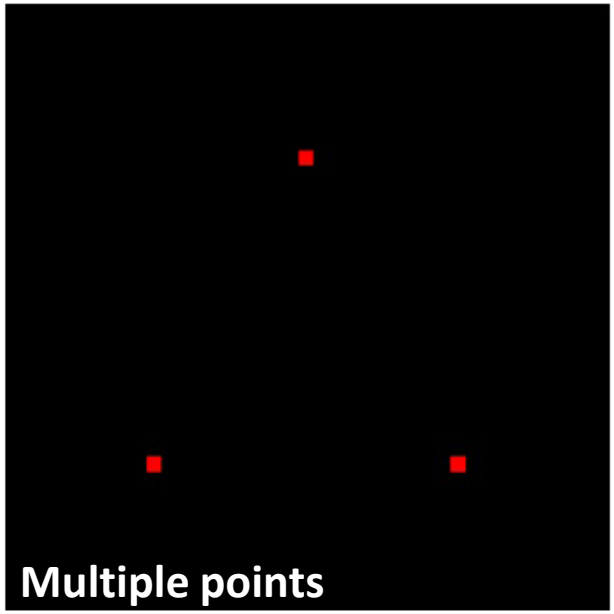
WebGL III

Drawing Multiple Points



3D model: simple color shade model and
wireframe model with triangles



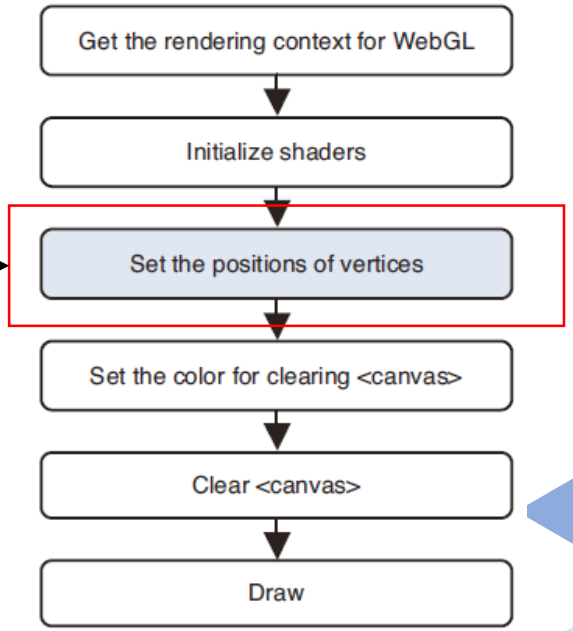
Drawing Multiple Points

ClickedPoints.js … **Only for single points**

```
65   for(var i = 0; i<len; i+=2) {
66     // Pass the position of a point to a_Position variable
67     gl.vertexAttrib3f(a_Position, g_points[i], g_points[i+1], 0.0);
68
69     // Draw a point
70     gl.drawArrays(gl.POINTS, 0, 1);
71   }
```

단일 지점에만 유용

여러 정점을 동시에 vertex shader로 전달



Drawing Multiple Points (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="..."></script>
 </body>
</html>
```

Drawing Multiple Points (js)

```
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() {
  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;
  }
```

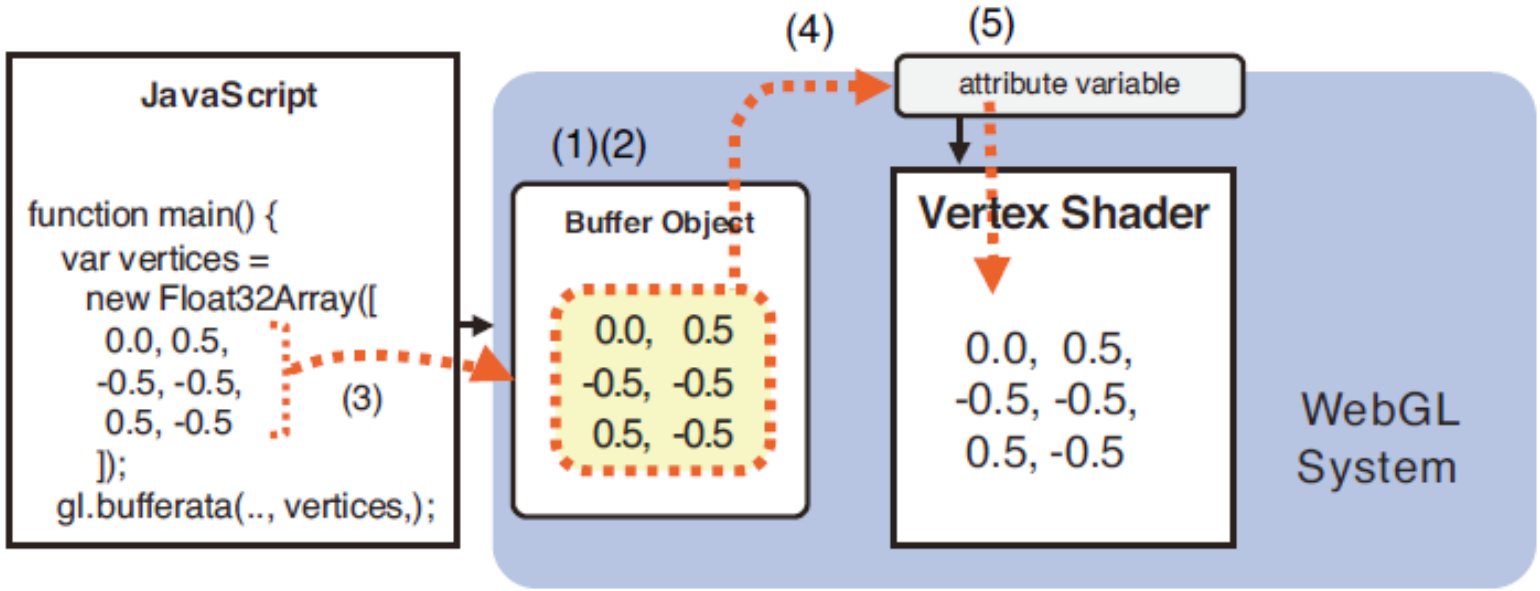
```
var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of
the vertices');
    return;
  }

 gl.clearColor(0, 0, 0, 1);

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

  // Draw three points
  gl.drawArrays(gl.POINTS, 0, n);
}
```

Drawing Multiple Points (js)

```
function initVertexBuffers(gl) {
  var vertices = new Float32Array([  0.0, 0.5,   -0.5, -0.5,   0.5, -0.5    ]);

  var n = 3; // The number of vertices

  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

  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 -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(a_Position);

  return n;
}
```

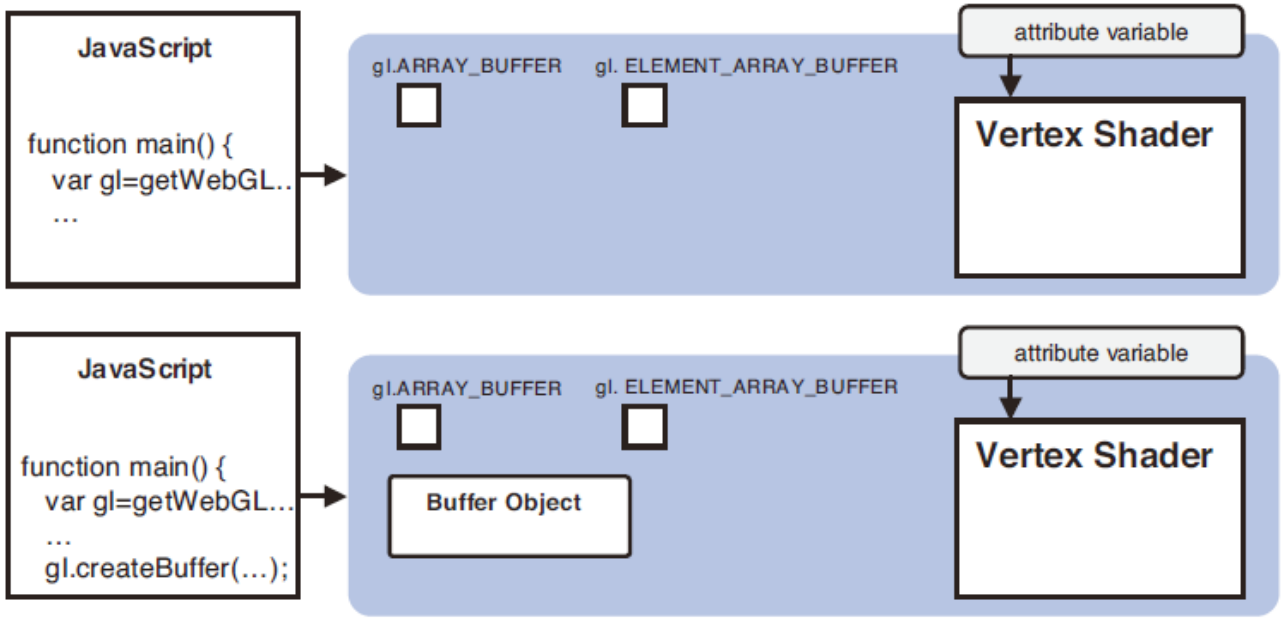
Drawing Multiple Points (js)

1. 버퍼 객체 생성 — gl.createBuffer()
2. 생성한 버퍼 객체를 target에 바인드 — gl.bindBuffer()
3. 버퍼 객체에 데이터를 입력 — gl.bufferData()
4. 속성 변수에 버퍼 할당 — gl.vertexAttribPointer()
5. 속성 배열 버텍스 사용 활성화 — gl.enableVertexAttribArray()



Passing multiple vertices to a vertex shader by using a buffer object

createBuffer()



`gl.createBuffer()`

Create a buffer object.

Return value	non-null	The newly created buffer object.
	null	Failed to create a buffer object.
Errors	None	

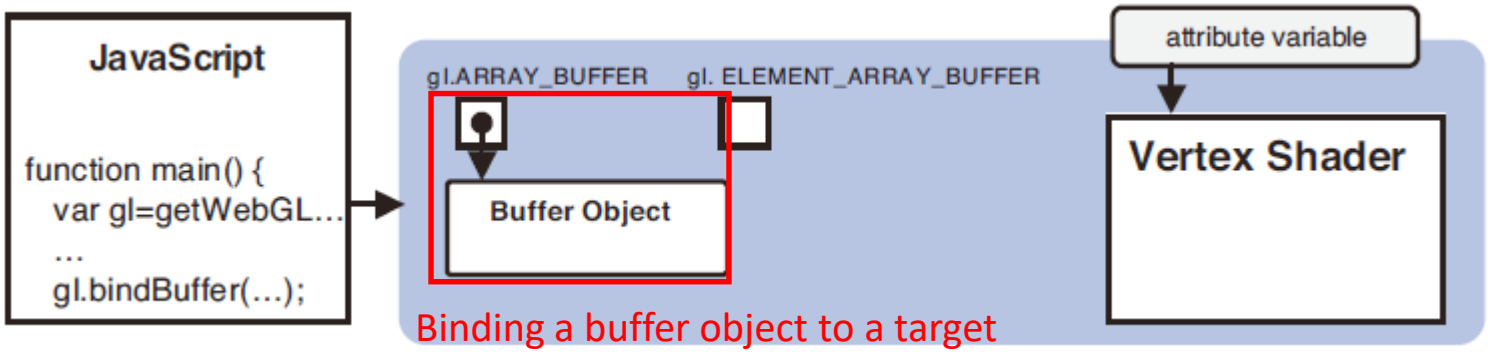
`gl.deleteBuffer(buffer)`

Delete the buffer object specified by *buffer*.

Parameters	buffer	Specifies the buffer object to be deleted.
Return Value	None	
Errors	None	

bindBuffer()

After creating a buffer object, then binding it to the target



gl.bindBuffer(target, buffer)

Enable the buffer object specified by *buffer* and bind it to the *target*.

Parameters Target can be one of the following:

`gl.ARRAY_BUFFER`	Specifies that the buffer object contains vertex data.
`gl.ELEMENT_ARRAY_BUFFER`	Specifies that the buffer object contains index values pointing to vertex data. (See Chapter 6, "The OpenGL ES Shading Language [GLSL ES].)
buffer	Specifies the buffer object created by a previous call to `gl.createBuffer()`. When `null` is specified, binding to the *target* is disabled.

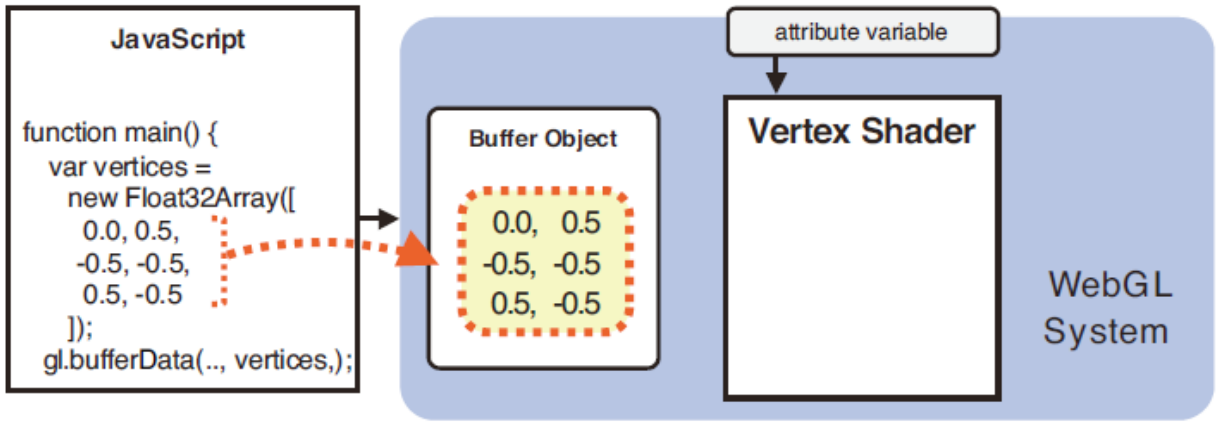
Return Value None

Errors INVALID_ENUM *target* is none of the above values. In this case, the current binding is maintained.

gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);

WebGL

bufferData()



`gl.bufferData(target, data, usage)`

Allocate storage and write the data specified by *data* to the buffer object bound to *target*.

Parameters	target	Specifies `gl.ARRAY_BUFFER` or `gl.ELEMENT_ARRAY_BUFFER`.
	data	Specifies the data to be written to the buffer object (typed array; see the next section).
	usage	Specifies a hint about how the program is going to use the data stored in the buffer object. This hint helps WebGL optimize performance but will not stop your program from working if you get it wrong.
	`gl.STATIC_DRAW`	The buffer object data will be specified once and used many times to draw shapes.
	`gl.STREAM_DRAW`	The buffer object data will be specified once and used a few times to draw shapes.
	`gl.DYNAMIC_DRAW`	The buffer object data will be specified repeatedly and used many times to draw shapes.
Return value	None	
Errors	INVALID_ENUM	*target* is none of the preceding constants

gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

WebGL

Typed Arrays

```
51   var vertices = new Float32Array([
52     0.0, 0.5,   -0.5, -0.5,   0.5, -0.5
53   ]);
54   var n = 3; // The number of vertices
```

Typed Array	Number of Bytes per Element	Description (C Types)
Int8Array	1	8-bit signed integer (signed char)
Uint8Array	1	8-bit unsigned integer (unsigned char)
Int16Array	2	16-bit signed integer (signed short)
Uint16Array	2	16-bit unsigned integer (unsigned short)
Int32Array	4	32-bit signed integer (signed int)
Uint32Array	4	32-bit unsigned integer (unsigned int)
Float32Array	4	32-bit floating point number (float)
Float64Array	8	64-bit floating point number (double)

Typed Arrays have a set of methods, a property, and a constant available

Methods, Properties, and Constants	Description
`get(index)`	Get the *index*-th element
`set(index, value)`	Set *value* to the *index*-th element
`set(array, offset)`	Set the elements of *array* from *offset*-th element
`length`	The length of the array
`BYTES_PER_ELEMENT`	The number of bytes per element in the array

~~Push() / pop()~~

WebGL

vertexAttribPointer()

vertexAttrib1~4f(…)

Can only be used to assign a single data value to an attribute Variable.

gl.vertexAttribPointer()

a way to assign an array of values to an attribute variable.

gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);

`gl.vertexAttribPointer(location, size, type, normalized, stride, offset)`

Assign the buffer object bound to `gl.ARRAY_BUFFER` to the attribute variable specified by *location*.

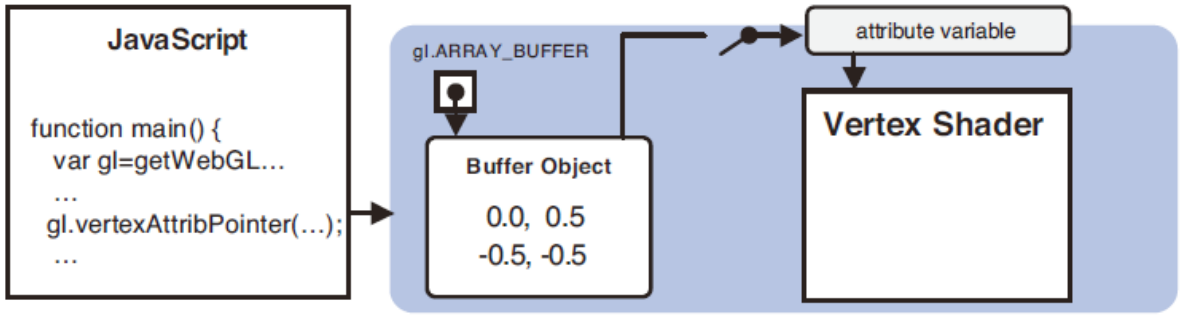
Parameters	location	Specifies the storage location of an attribute variable.
	size	Specifies the number of components per vertex in the buffer object (valid values are 1 to 4). If *size* is less than the number of components required by the attribute variable, the missing components are automatically supplied just like `gl.vertexAttrib[1234]f()`. For example, if *size* is 1, the second and third components will be set to 0, and the fourth component will be set to 1.
	type	Specifies the data format using one of the following:

`gl.UNSIGNED_BYTE`	unsigned byte	for `Uint8Array`
`gl.SHORT`	signed short integer	for `Int16Array`
`gl.UNSIGNED_SHORT`	unsigned short integer	for `Uint16Array`
`gl.INT`	signed integer	for `Int32Array`
`gl.UNSIGNED_INT`	unsigned integer	for `Uint32Array`
`gl.FLOAT`	floating point number	for `Float32Array`

	normalized	Either `true` or `false` to indicate whether nonfloating data should be normalized to [0, 1] or [–1, 1].
	stride	Specifies the number of bytes between different vertex data elements, or zero for default stride (see Chapter 4).
	offset	Specifies the offset (in bytes) in a buffer object to indicate what number-th byte the vertex data is stored from. If the data is stored from the beginning, *offset* is 0.
Return value	None	
Errors	INVALID_OPERATION	There is no current program object.
	INVALID_VALUE	*location* is greater than or equal to the maximum number of attribute variables (8, by default). *stride* or *offset* is a negative value.

WebGL

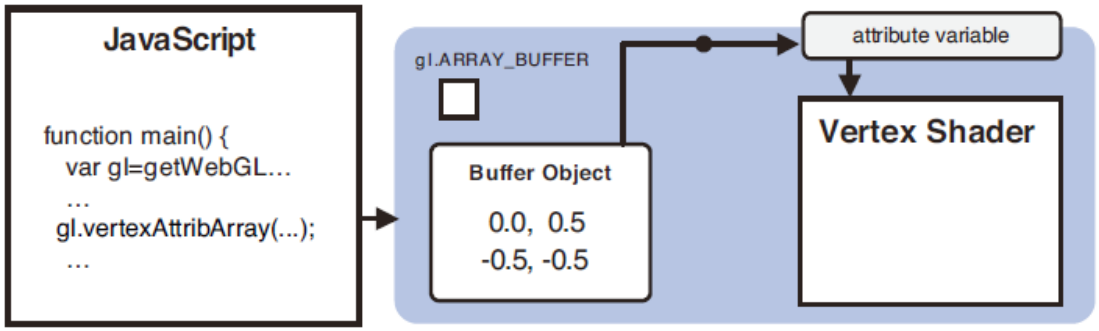
enableVertexAttribArray()



`gl.enableVertexAttribArray(location)`

Enable the assignment of a buffer object to the attribute variable specified by *location*.

Parameters	location	Specifies the storage location of an attribute variable.
Return value	None	
Errors	INVALID_VALUE	*location* is greater than or equal to the maximum number of attribute variables (8 by default).



`gl.disableVertexAttribArray(location)`

Disable the assignment of a buffer object to the attribute variable specified by *location*.

Parameters	location	Specifies the storage location of an attribute variable.
Return Value	None	
Errors	INVALID_VALUE	*location* is greater than or equal to the maximum number of attribute variables (8 by default).

drawArrays()

`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 what number-th vertex is used to draw from (integer).
	count	Specifies the number of vertices to be used (integer).

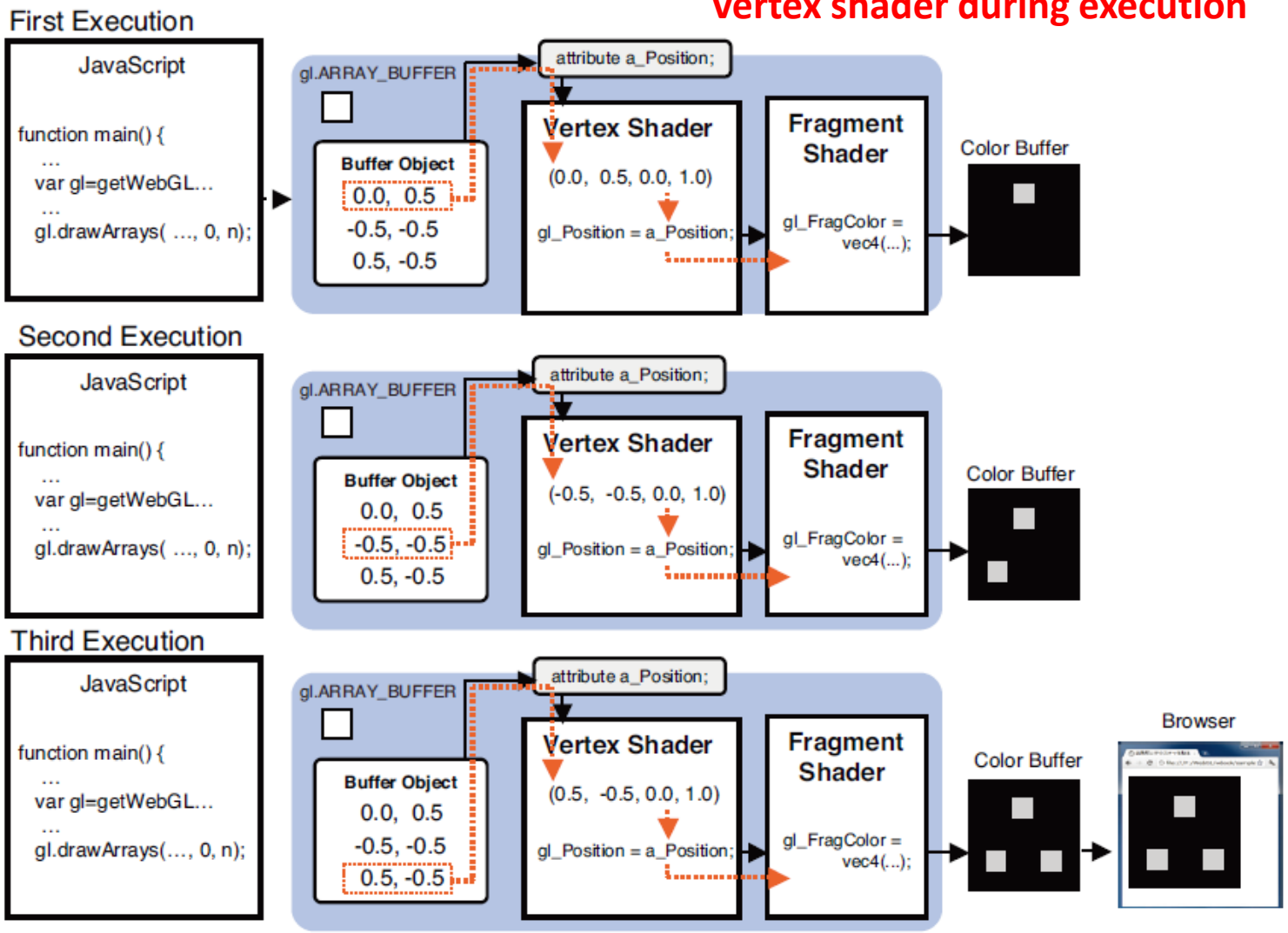
gl.drawArrays(gl.POINTS, 0, n);

- 점 세개를 그리기 때문에 여전히 PONITS
- 버퍼의 첫 번째 좌표에서 그리는 것이므로 0
- 세 번째 점까지 그리기 위한 점 개수 3

WebGL

drawArrays()

How the data in a buffer object is passed to a vertex shader during execution



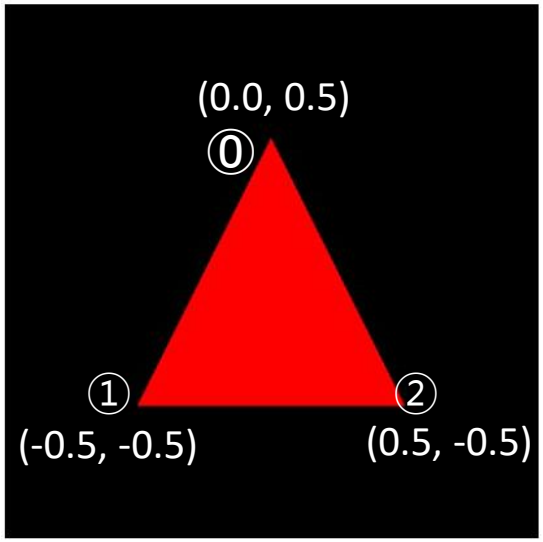
HelloTriangle (js)

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

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

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 n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of
the vertices');
    return;
  }

 gl.clearColor(0, 0, 0, 1);

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

  // Draw three points
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

HelloTriangle (js)

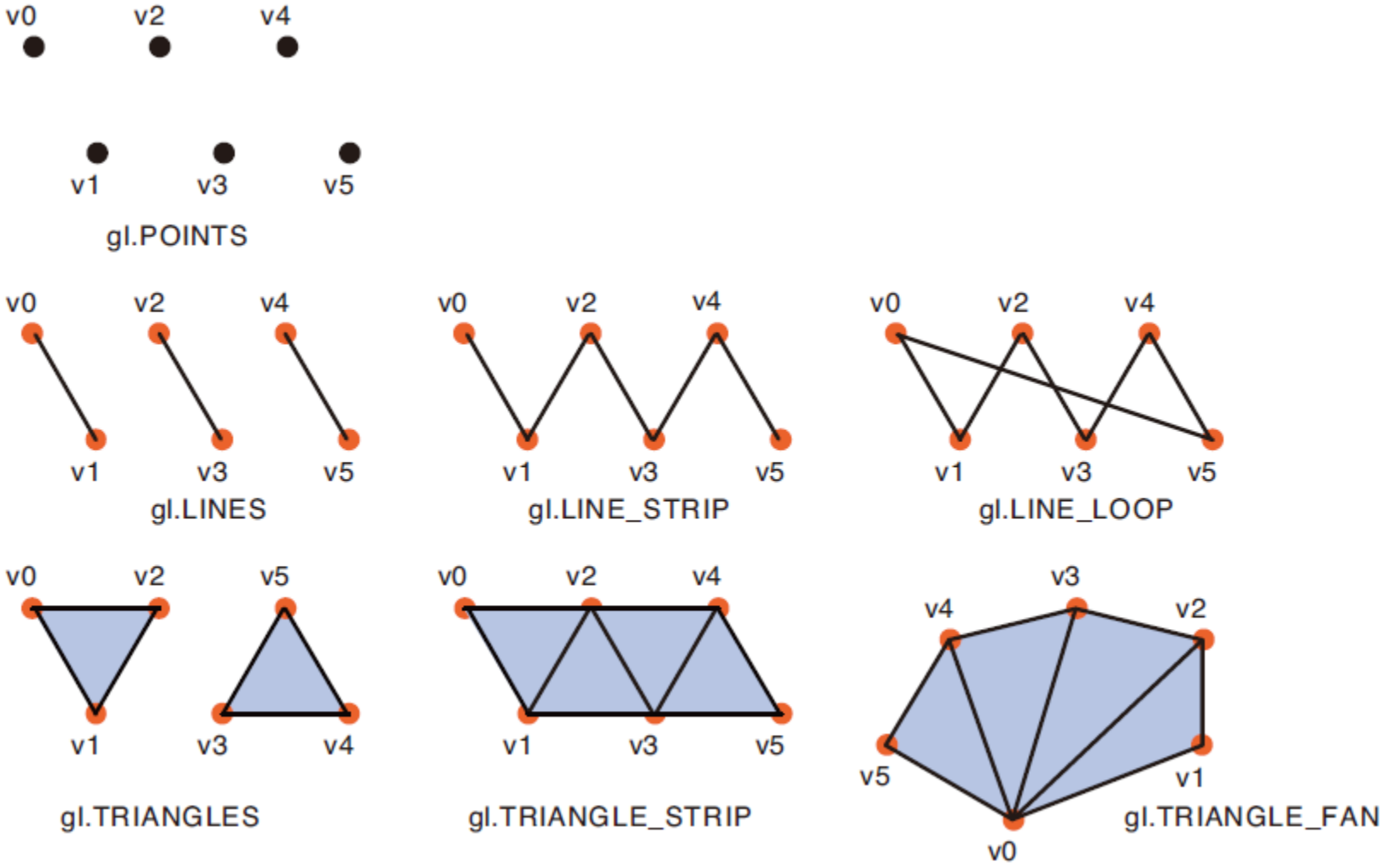
```
function initVertexBuffers(gl) {          ⓪          ①          ②
  var vertices = new Float32Array([  0.0, 0.5,   -0.5, -0.5,   0.5, -0.5   ]);

  var n = 3; // The number of vertices

  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

  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 -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(a_Position);

  return n;
}
```



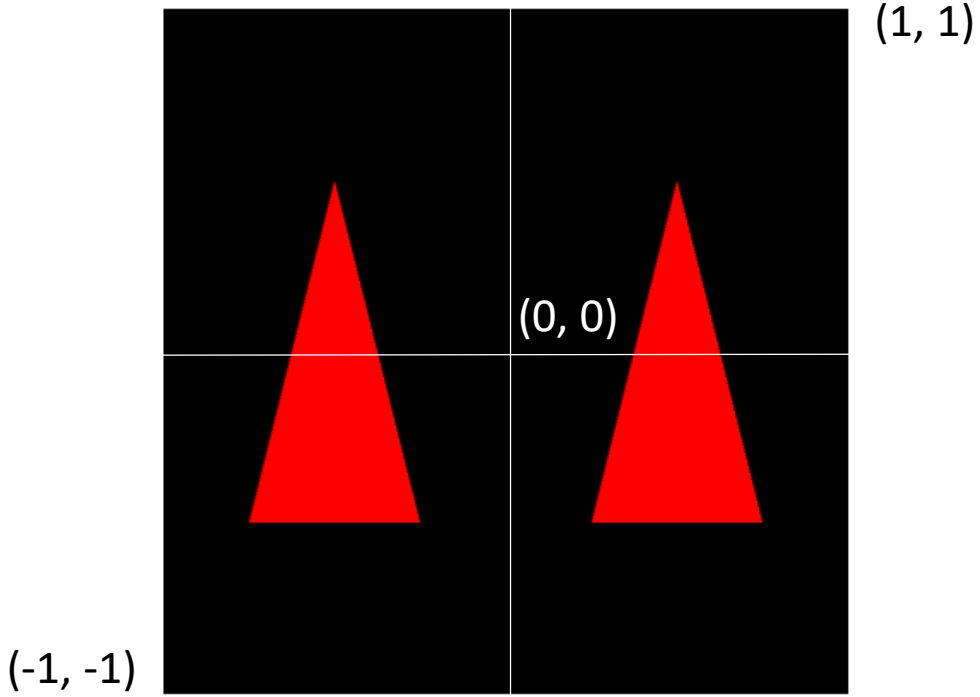
Basic Shape

Basic Shape	Mode	Description
Points	gl.POINTS	v0, v1, v2 …와 같은 좌표 점을 그림
Line segments	gl.LINES	연결되지 않은 선 (v0, v1), (v2, v3), (v4, v5)…
Line strips	gl.LINE_STRIP	연결된 선 (v0, v1), (v1, v2), (v2, v3) …
Line loops	gl.LINE_LOOP	연결된 선 (끝점과 첫점) (v0, v1), (v1, v2), (v2, v3), …, (vn, v0)
Triangles	gl.TRIANGLES	분리된 삼각형 (v0, v1, v2), (v3, v4, v5) …
Triangle strips	gl.TRIANGLE_STRIP	연결된 삼각형 (v0, v1, v2), (v2, v1, v3), (v2, v3, v4)
Triangle fans	gl.TRIANGLE_FAN	연결된 삼각형 (v0, v1, v2), (v0, v2, v3), (v0, v3, v4)

Basic Shape


v0
v2
v4
v1
v3
v5
gl.POINTS
gl.LINES
gl.LINE_STRIP
gl.LINE_LOOP
gl.TRIANGLES
gl.TRIANGLE_STRIP
gl.TRIANGLE_FAN

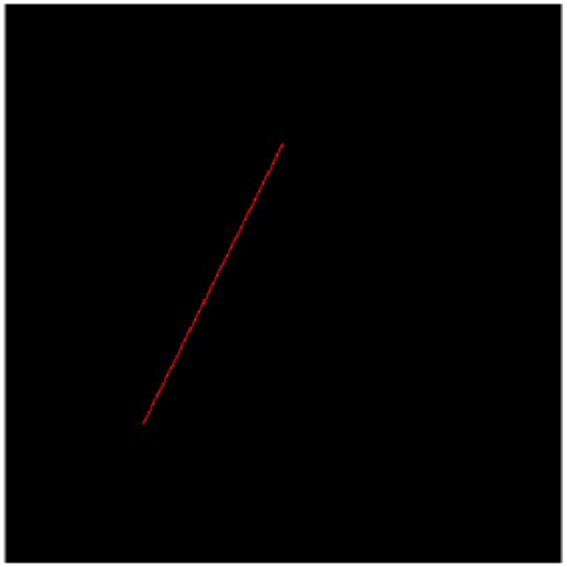
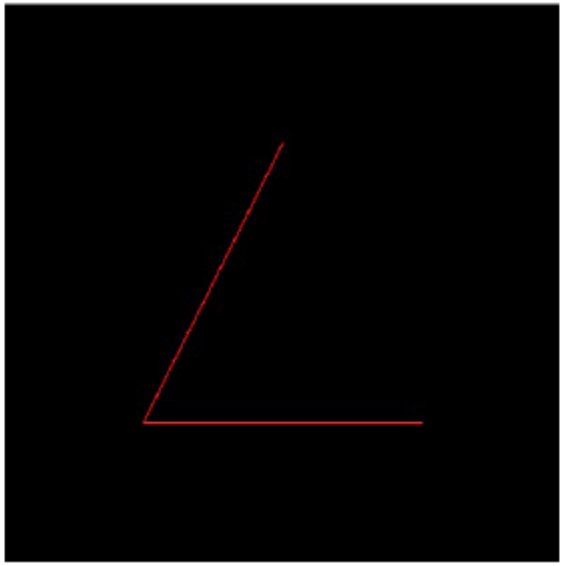
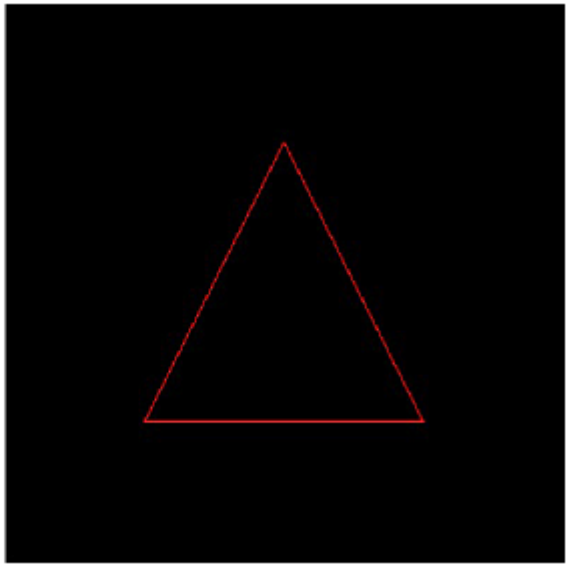
HelloTriangles




```
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    -0.25, -0.5, -0.75, -0.5, -0.5, 0.5, 0.25, -0.5, 0.75, -0.5, 0.5, 0.5
  ]);
  var n = 6; // The number of vertices
```

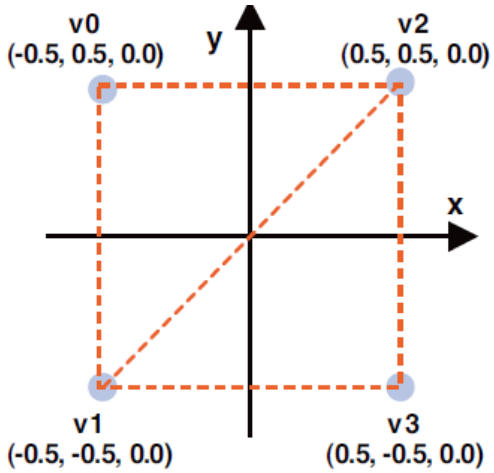
Lines

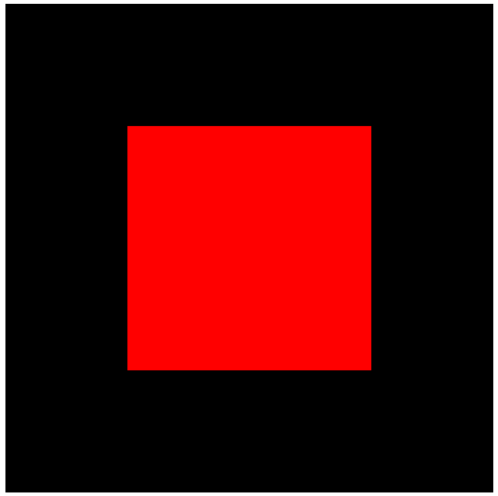
```
46    gl.drawArrays(gl.LINES, 0, n);
46    gl.drawArrays(gl.LINE_STRIP, 0, n);
46    gl.drawArrays(gl.LINE_LOOP, 0, n);
```

Triangles

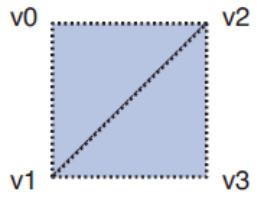
```
50   var vertices = new Float32Array([
51     -0.5, 0.5,   -0.5, -0.5,   0.5, 0.5,   0.5, -0.5
52   ]);
```



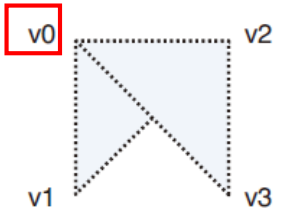


```
46   gl.drawArrays(gl.TRIANGLE_STRIP, 0, n);
```

```
46   gl.drawArrays(gl.TRIANGLE_FAN, 0, n);
```



gl.TRIANGLE_STRIP



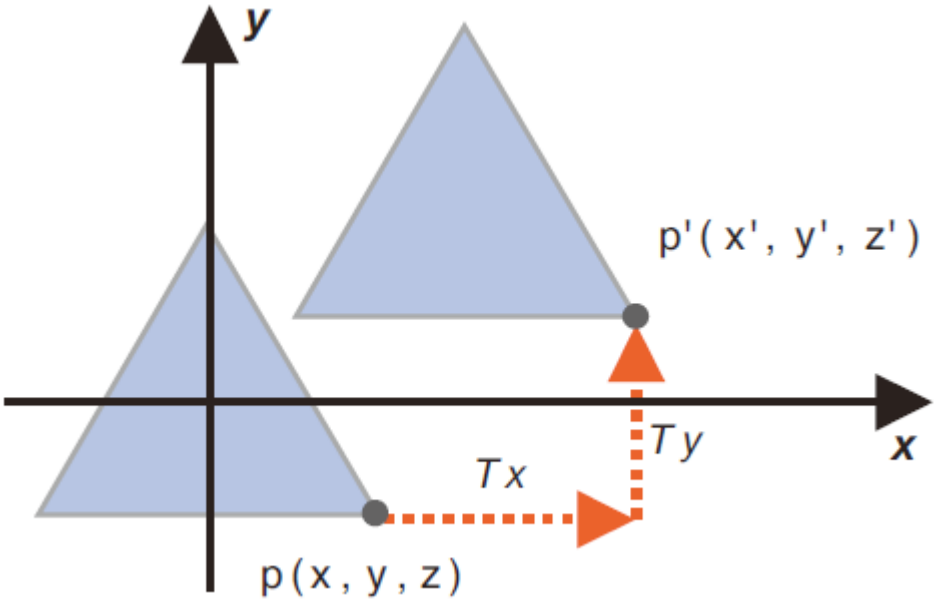
gl.TRIANGLE_FAN

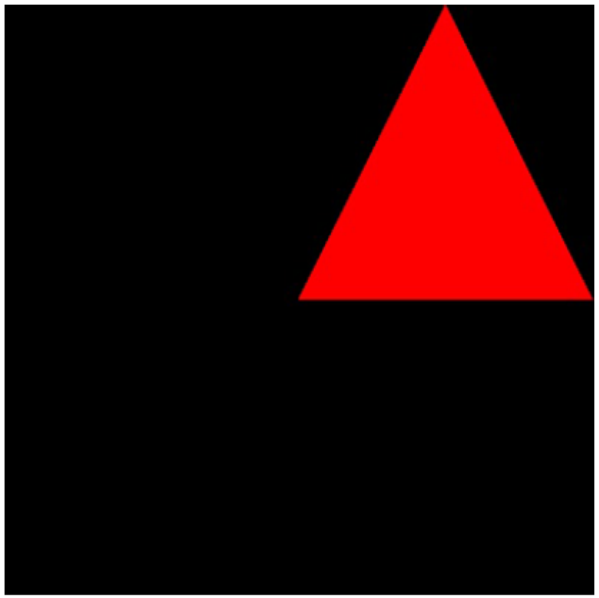
Translation

$$x' = x + Tx$$

$$y' = y + Ty$$

$$z' = z + Tz$$



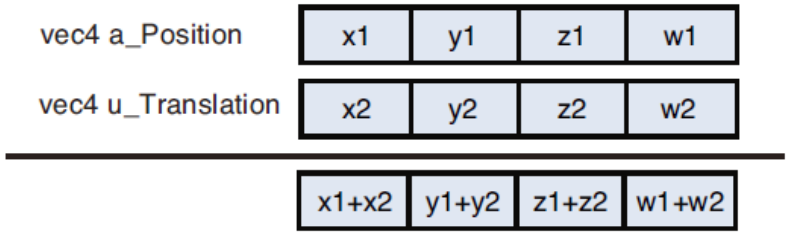


per-vertex operation

Applying to each vertex coordinate of a shape to translate the shape

No per-fragment operation

WebGL



Translation (js)

```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform vec4 u_Translation;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position + u_Translation;\n' +
  '}\n';
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';

var Tx = 0.5, Ty = 0.5, Tz = 0.0;
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 n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the
vertices');
    return;
  }

  var u_Translation = gl.getUniformLocation(gl.program,
'u_Translation');
  if (!u_Translation) {
    console.log('Failed to get the storage location of
                                          u_Translation');
    return;
  }
  gl.uniform4f(u_Translation, Tx, Ty, Tz, 0.0);

  gl.clearColor(0, 0, 0, 1);

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

  // Draw three points
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

Translation (js)

```
function initVertexBuffers(gl) {
  var vertices = new Float32Array([  0.0, 0.5,   -0.5, -0.5,   0.5, -0.5    ]);

  var n = 3; // The number of vertices

  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

  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 -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(a_Position);

  return n;
}
```

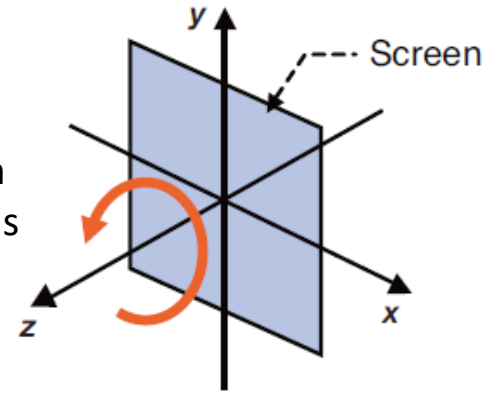
Rotation

- 회전을 위해서는 3가지 요소 필요
  > axis, direction, angle

- Rotation axis (the axis the shape will be rotated around)
- Rotation direction (the direction: clockwise or counterclockwise)
- Rotation angle (the number of degrees the shape will be rotated through)

- 시계 반대방향으로 회전 (z축 중심으로 가정)

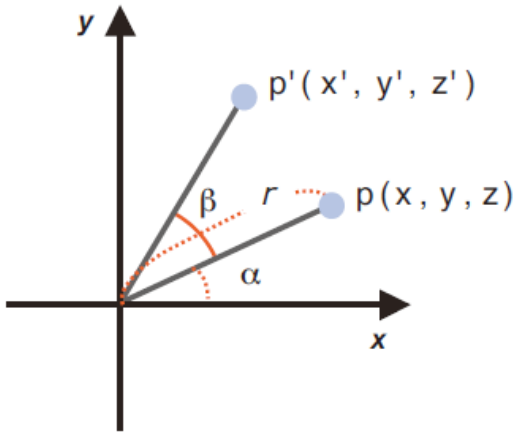
Positive rotation around the z-axis



+ direction: CCW (counter-clockwise) direction
Right-handed system

Rotation

- Translation 과 같은 방식으로 rotation 계산 표현식



r : p 까지의 거리
α: x 축에서 p 점까지의 회전 각
β: p 점에서 p′ 점까지의 회전 각

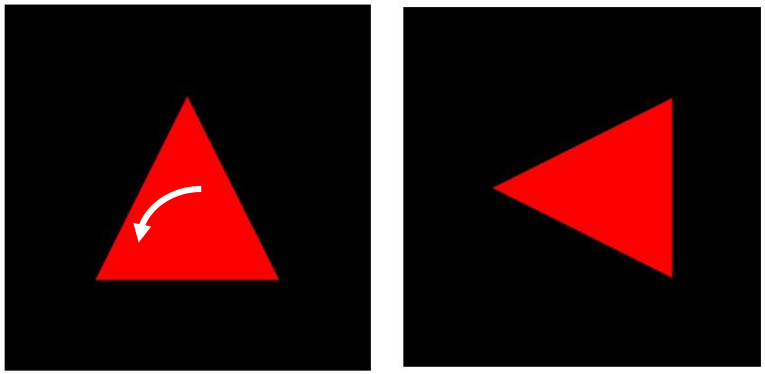
$$x = r * cos\,\alpha \qquad x' = r * cos\,(\alpha + \beta)$$

$$y = r * sin\,\alpha \qquad y' = r * sin\,(\alpha + \beta)$$

삼각함수 덧셈 정리를 사용하여 얻은 공식

$$x' = r * (cos\,\alpha * cos\beta - sin\alpha * sin\beta)$$

$$y' = r * (sin\,\alpha * cos\beta + cos\alpha * sin\beta)$$



$$sin(a \pm b) = sin(a) * cos(b) \mp cos(a) * sin(b)$$

$$cos(a \pm b) = cos(a) * cos(b) \mp sin(a) * sin(b)$$

Rotation

$$x = r * cos\,\alpha$$
$$y = r * sin\,\alpha$$

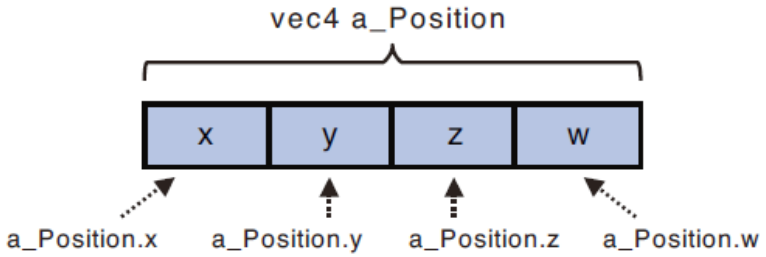
$$x' = r * cos\,(\alpha + \beta)$$
$$y' = r * sin\,(\alpha + \beta)$$

$$x' = r * (cos\,\alpha * cos\beta - sin\alpha * sin\beta)$$
$$y' = r * (sin\,\alpha * cos\beta + cos\alpha * sin\beta)$$

$$x' = x * cos\beta - y * sin\beta$$
$$y' = y * sin\,\beta + y * cos\beta$$
$$z' = z$$

WebGL

Rotation (js)




```
var VSHADER_SOURCE =
'attribute vec4 a_Position;\n' +
 'uniform float u_CosB, u_SinB;\n' +
 'void main() {\n' +
 '  gl_Position.x = a_Position.x * u_CosB - a_Position.y * u_SinB;\n' +
 '  gl_Position.y = a_Position.x * u_SinB + a_Position.y * u_CosB;\n' +
 '  gl_Position.z = a_Position.z;\n' +
 '  gl_Position.w = 1.0;\n' +
 '}\n';
var FSHADER_SOURCE =
 'void main() {\n' +
 '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
 '}\n';

var ANGLE = 90.0;
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;
 }
```

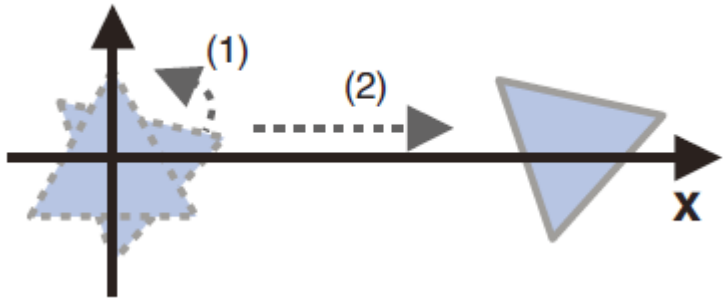
```
var n = initVertexBuffers(gl);
 if (n < 0) {
  console.log('Failed to set the positions of the vertices');
  return;
 }

 var radian = Math.PI * ANGLE / 180.0;
 var cosB = Math.cos(radian);
 var sinB = Math.sin(radian);
 var u_CosB = gl.getUniformLocation(gl.program, 'u_CosB');
 var u_SinB = gl.getUniformLocation(gl.program, 'u_SinB');
 if (!u_CosB || !u_SinB) {
  console.log('Failed to get the storage location of u_CosB
or u_SinB');
  return;
 }
 gl.uniform1f(u_CosB, cosB);
 gl.uniform1f(u_SinB, sinB);

 gl.clearColor(0, 0, 0, 1);

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

 // Draw three points
 gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

Rotation (js)

```
function initVertexBuffers(gl) {
  var vertices = new Float32Array([  0.0, 0.5,   -0.5, -0.5,   0.5, -0.5    ]);

  var n = 3; // The number of vertices

  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

  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 -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(a_Position);

  return n;
}
```

Transformation Matrix

- 간단한 변환의 경우 수학 표현식 사용 가능 하지만 복잡할경우?
  예) 회전 후 이동...



- 행렬을 활용해 적절히 그래픽 조작 가능

$$x' = x \cos\beta - y \sin\beta$$

$$y' = x \sin\beta + y \cos\beta$$

$$z' = z$$

$$x' = x + T_x$$

$$y' = y + T_y$$

$$z' = z + T_z$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Transformation Matrix

$$x' = ax + by + cz \qquad x' = x \cos\beta - y \sin\beta$$

$$y' = dx + ey + fz \qquad y' = x \sin\beta + y \cos\beta$$

$$z' = gx + hy + iz \qquad z' = z$$

$$x' = ax + by + cz \qquad y' = dx + ey + fz$$

$$x' = x \cos\beta - y \sin\beta \qquad y' = x \sin\beta + y \cos\beta$$

Rotation Matrix

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$x' = ax + by + cz \qquad x' = x + T_x$$

$$y' = dx + ey + fz \qquad y' = y + T_y$$

$$z' = gx + hy + iz \qquad z' = z + T_z$$

$$x' = ax + by + cz + d \qquad x' = x + T_x$$

$$y' = ex + fy + gz + h \qquad y' = y + T_y$$

$$z' = ix + jy + kz + l \qquad z' = z + T_z$$

$$1 = mx + ny + oz + p$$

Translation Matrix

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Transformation Matrix

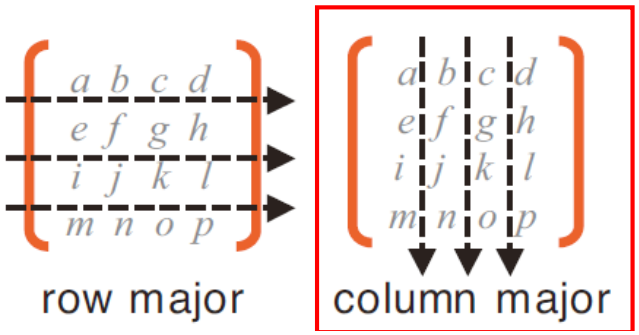
```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_xformMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_xformMatrix * a_Position;\n' +
  '}\n';

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

var ANGLE = 90.0;
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 n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Create a rotation matrix
  var radian = Math.PI * ANGLE / 180.0; // Convert to
radians
  var cosB = Math.cos(radian), sinB = Math.sin(radian);

  // Note: WebGL is column major order
  var xformMatrix = new Float32Array([
     cosB, sinB, 0.0, 0.0,
    -sinB, cosB, 0.0, 0.0,
      0.0,  0.0, 1.0, 0.0,
      0.0,  0.0, 0.0, 1.0
  ]);
  // Pass the rotation matrix to the vertex shader
  var u_xformMatrix = gl.getUniformLocation(gl.program,
'u_xformMatrix');
  if (!u_xformMatrix) {
    console.log('Failed to get the storage location of
u_xformMatrix');
    return;
  }
  gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix);
 gl.clearColor(0, 0, 0, 1);

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

  // Draw three points
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

Transformation Matrix

```
function initVertexBuffers(gl) {
  var vertices = new Float32Array([  0.0, 0.5,   -0.5, -0.5,   0.5, -0.5    ]);

  var n = 3; // The number of vertices

  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

  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 -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(a_Position);

  return n;
}
```

WebGL / OpenGL Array



Web GL: column major order

in an array as follows: [*a, e, i, m, b, f, j, n, c, g, k, o, d, h, l, p*]

```
48   var xformMatrix = new Float32Array([
49      cosB, sinB, 0.0, 0.0,
50     -sinB, cosB, 0.0, 0.0,
51      0.0,  0.0, 1.0, 0.0,
52      0.0,  0.0, 0.0, 1.0
53    ]);
```

Web GL 배열 저장

$$\begin{bmatrix} \cos\beta & -\sin\beta & 0 & 0 \\ \sin\beta & \cos\beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

실제 계산에 이용되는 행렬

uniformMatrix4fv

gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix);

`gl.uniformMatrix4fv``(location, transpose, array)`

Assign the 4×4 matrix specified by *array* to the uniform variable specified by *location*.

Parameters	location	Specifies the storage location of the uniform variable.
	Transpose	Must be `false` in WebGL.[3]
	array	Specifies an array containing a 4×4 matrix in column major order (typed array).
Return value	None	
Errors	INVALID_OPERATION	There is no current program object.
	INVALID_VALUE	*transpose* is not `false`, or the length of *array* is less than 16.

이동에 대한 행렬 재사용

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 & 0 \\ \sin\beta & \cos\beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

<new coordinates> = <transformation matrix> * <original coordinates>

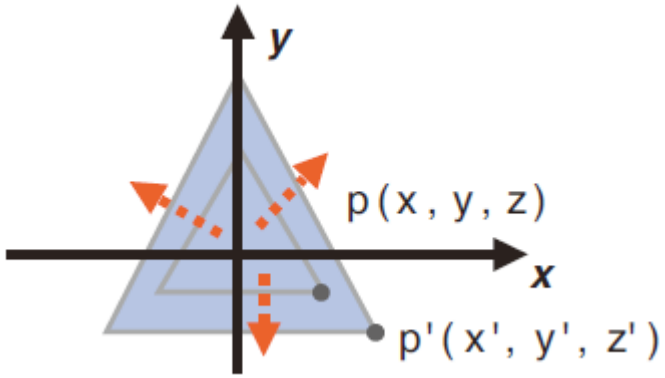
```
' gl_Position = u_xformMatrix * a_Position;\n' +
```

```
48  var xformMatrix = new Float32Array([
49    cosB, sinB, 0.0, 0.0,
50   -sinB, cosB, 0.0, 0.0,
51    0.0,  0.0, 1.0, 0.0,
52    0.0,  0.0, 0.0, 1.0
53  ]);
```

```
16  // The translation distance for x, y, and z direction
17  var Tx = 0.5, Ty = 0.5, Tz = 0.0;
```

```
43  // Create a rotation matrix
44  // var radian = Math.PI * ANGLE / 180.0; // Convert to radians
45  // var cosB = Math.cos(radian), sinB = Math.sin(radian);
46
47   // Note: WebGL is column major order
48   var xformMatrix = new Float32Array([
49      1.0, 0.0, 0.0, 0.0,
50      0.0, 1.0, 0.0, 0.0,
51      0.0, 0.0, 1.0, 0.0,
52      Tx, Ty, Tz, 1.0
53   ]);
```

Scaling



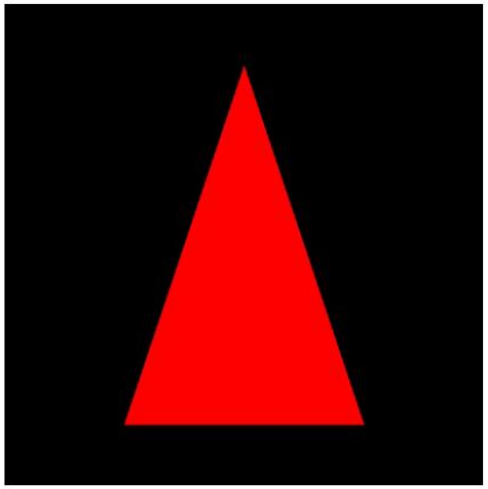
$$x' = S_x \times x$$

$$y' = S_y \times y$$

$$z' = S_z \times z$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

```
17  varSx = 1.0, Sy = 1.5, Sz = 1.0;
     ...
47    // Note: WebGL is column major order
48    var xformMatrix = new Float32Array([
49       Sx,   0.0,   0.0,   0.0,
50       0.0,   Sy,   0.0,   0.0,
51       0.0,   0.0,   Sz,   0.0,
52       0.0,   0.0,   0.0,   1.0
53    ]
```



Matrix.js

```
var Matrix4 = function(opt_src) {
…
Matrix4.prototype.setIdentity = function() {…}
Matrix4.prototype.set = function(src) {…}
Matrix4.prototype.concat = function(other) {…}
Matrix4.prototype.multiplyVector3 = function(pos) {…}
Matrix4.prototype.multiplyVector4 = function(pos) {…}
Matrix4.prototype.transpose = function() {…}
Matrix4.prototype.setInverseOf = function(other) {…}
Matrix4.prototype.invert = function() {…}
Matrix4.prototype.setOrtho = function(left, right, bottom, top, near, far) {…}
Matrix4.prototype.setScale = function(x, y, z) {...}
Matrix4.prototype.setTranslate = function(x, y, z) {...}
Matrix4.prototype.setRotate = function(angle, x, y, z) {...}
…
}
```

Matrix.js

```
13      <script src="../lib/webgl-debug.js"></script>
14      <script src="../lib/cuon-utils.js"></script>
15      <script src="../lib/cuon-matrix.js"></script>
16      <script src="RotatedTriangle_Matrix4.js"></script>
```

```
1  // RotatedTriangle_Matrix.js
     ...
43    // Create a rotation matrix
44    var radian = Math.PI * ANGLE / 180.0; // Convert to radians
45    varcosB = Math.cos(radian), sinB = Math.sin(radian);
46
47    // Note: WebGL is column major order
48    var xformMatrix = new Float32Array([
49       cosB, sinB, 0.0, 0.0,
50      -sinB, cosB, 0.0, 0.0,
51        0.0,  0.0, 1.0, 0.0,
52        0.0,  0.0, 0.0, 1.0
53    ]);
      ...
61    gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix);
```

X축: ANGLES, 1, 0, 0
Y축: ANGLES, 0, 1, 0

```
1  // RotatedTriangle_Matrix4.js
     ...
47    // Create Matrix4 object for a rotation matrix
48    var xformMatrix = new Matrix4();
49    // Set the rotation matrix to xformMatrix
50    xformMatrix.setRotate(ANGLE, 0, 0, 1);
      ...
56    // Pass the rotation matrix to the vertex shader
57    gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix.elements);
```

Matrix4 object

Matrix.js

Methods and Properties supported by Matrix4

Methods and Properties	Description
`Matrix4.setIdentity()`	Initialize a matrix (to the identity matrix*).
`Matrix4.setTranslate(x, y, z)`	Set `Matrix4` to the translation matrix, which translates *x* units in the direction of the x-axis, *y* units in the direction of the y-axis, and *z* units in the direction of the z-axis.
`Matrix4.setRotate(angle, x, y, z)`	Set `Matrix4` to the rotation matrix, which rotates *angle* degrees around the rotation axis (*x*, *y*, *z*). The (*x*, *y*, *z*) coordinates do not need to be **normalized**. (See Chapter 8, "Lighting Objects.")
`Matrix4.setScale(x, y, z)`	Set `Matrix4` to the scaling matrix with scaling factors *x*, *y*, and *z*.
`Matrix4.translate (x, y, z)`	Multiply the matrix stored in `Matrix4` by the translation matrix, which translates *x* units in the direction of the x-axis, *y* units in the direction of the y-axis, and *z* units in the direction of the z-axis, storing the result back into `Matrix4`.
`Matrix4.rotate(angle, x, y, z)`	Multiply the matrix stored in `Matrix4` by the rotation matrix, which rotates *angle* degrees around the rotation axis (*x*, *y*, *z*), storing the results back into `Matrix4`. The (*x*, *y*, *z*) coordinates do not need to be normalized. (See Chapter 8.)
`Matrix4.scale(x, y, z)`	Multiply the matrix stored in `Matrix4` by the scaling matrix, with scaling factors *x*, *y*, and *z*, storing the results back into `Matrix4`.
`Matrix4.set(m)`	Set the matrix *m* to `Matrix4`. *m* must be a `Matrix4` object.
`Matrix4.elements`	The typed array (`Float32Array`) containing the elements of the matrix stored in `Matrix4`.

Rotated & Translated Triangle

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <title>Translate And Then Rotate A Triangle</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="../lib/cuon-matrix.js"></script>
  <script src="06 RotatedTranslatedTriangle.js"></script>
 </body>
</html>
```

WebGL

Rotated & Translated Triangle

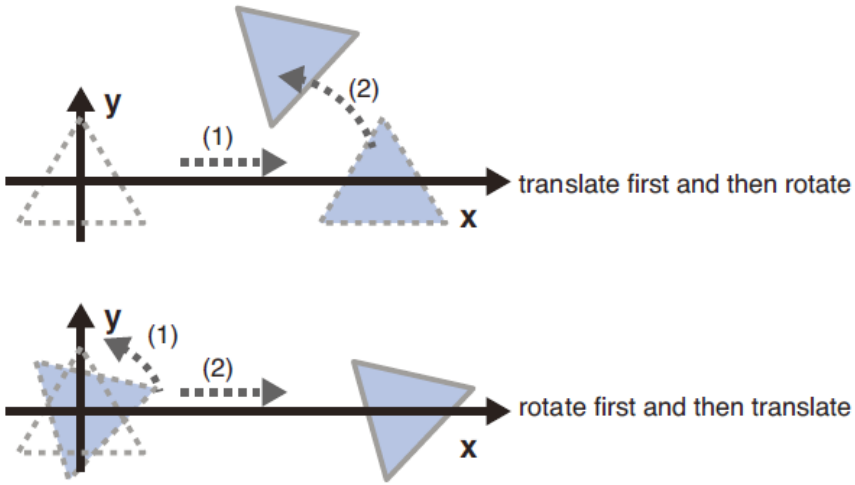
```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_ModelMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_ModelMatrix * a_Position;\n' +
  '}\n';
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
function main() {
  // 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;
  }
```

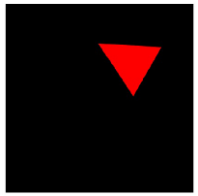
```
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the
vertices');
    return;
  }

  var modelMatrix = new Matrix4();

  var ANGLE = 60.0; // The rotation angle
  var Tx = 0.5;     // Translation distance
  modelMatrix.setRotate(ANGLE, 0, 0, 1);
  modelMatrix.translate(Tx, 0, 0);

  var u_ModelMatrix =
gl.getUniformLocation(gl.program,
'u_ModelMatrix');
  if (!u_ModelMatrix) {
    console.log('Failed to get the storage location
of u_xformMatrix');
    return;
  }
  gl.uniformMatrix4fv(u_ModelMatrix, false,
modelMatrix.elements);
  gl.clearColor(0, 0, 0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT);
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

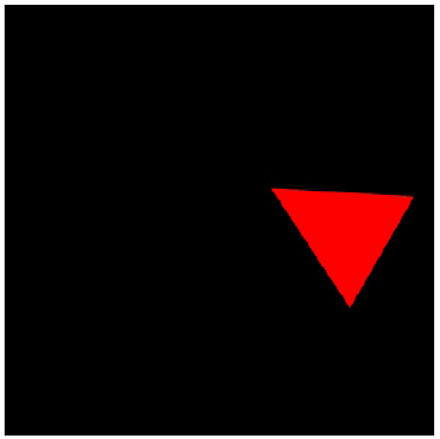
Rotated & Translated Triangle

```
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0, 0.3,   -0.3, -0.3,   0.3, -0.3
  ]);
  var n = 3; // The number of vertices
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return false;
  }
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

  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 -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(a_Position);

  return n;
}
```

Rotated & Translated Triangle



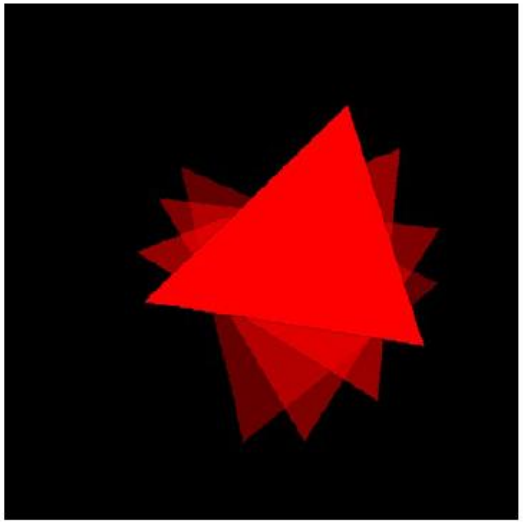




```
39
40  // Create Matrix4 object for model transformation
41  var modelMatrix = new Matrix4();
42
43  // Calculate a model matrix
44  var ANGLE = 60.0; // The rotation angle
45  var Tx = 0.5;     // Translation distance
46  modelMatrix.setTranslate(Tx, 0, 0);  // Set translation matrix
47  modelMatrix.rotate(ANGLE, 0, 0, 1);  // Multiply modelMatrix by the calculated rotation matrix
48
```

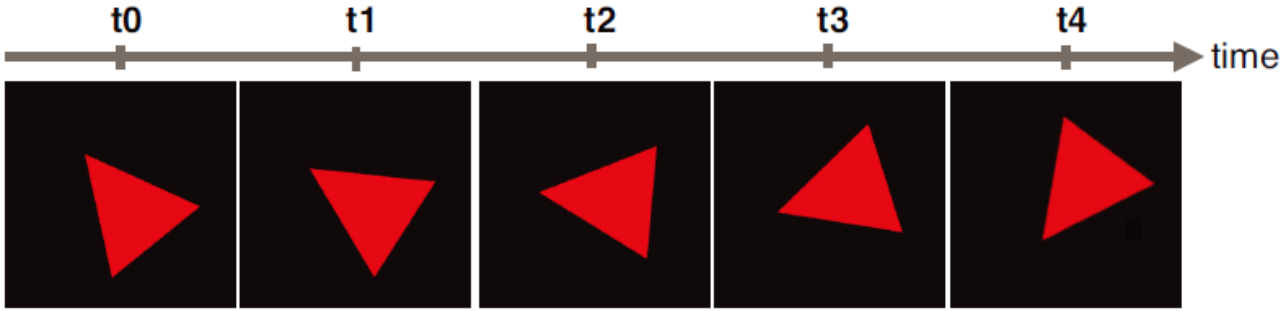
```
43  // Calculate a model matrix
44  var ANGLE = 60.0; // The rotation angle
45  var Tx = 0.5;     // Translation distance
46  modelMatrix.rotate(ANGLE, 0, 0, 1);  // Set
47  modelMatrix.setTranslate(Tx, 0, 0);       /
48
```

Basic Animation



Rotating triangle: continually rotates a triangle at a Constant rotation speed (45 degree/second)

To animate a rotating triangle, need to redraw the triangle at a slightly different angle each time it draws



Rotating Triangle

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <title>Continually Rotate A Triangle</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="../lib/cuon-matrix.js"></script>
  <script src="07 RotatingTriangle.js"></script>
 </body>
</html>
```

Rotating Triangle

```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_ModelMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_ModelMatrix * a_Position;\n' +
  '}\n';
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
var ANGLE_STEP = 45.0;
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 n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
```

```
  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  var u_ModelMatrix =
gl.getUniformLocation(gl.program, 'u_ModelMatrix');
  if (!u_ModelMatrix) {
    console.log('Failed to get the storage location of
u_ModelMatrix');
    return;
  }

  var currentAngle = 0.0;
  var modelMatrix = new Matrix4();

  var tick = function() {
    currentAngle = animate(currentAngle);
    draw(gl, n, currentAngle,
                          modelMatrix, u_ModelMatrix);
    requestAnimationFrame(tick, canvas);
  };
  tick();
}
```

Rotating Triangle

```
function initVertexBuffers(gl) {
  var vertices = new Float32Array ([
    0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3;   // The number of vertices

  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

  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 -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(a_Position);

  return n;
}
```

```
function draw(gl, n, currentAngle, modelMatrix,
                                        u_ModelMatrix) {
  modelMatrix.setRotate(currentAngle, 0, 0, 1);
  gl.uniformMatrix4fv(u_ModelMatrix,
                      false, modelMatrix.elements);
  gl.clear(gl.COLOR_BUFFER_BIT);
  gl.drawArrays(gl.TRIANGLES, 0, n);
}


var g_last = Date.now();
function animate(angle) {
  var now = Date.now();
  var elapsed = now - g_last;
  g_last = now;
  var newAngle = angle + (ANGLE_STEP * elapsed)
                                          / 1000.0;
  return newAngle %= 360;
}
```

Rotating Triangle

JavaScript 에서 특정한 함수를 반복적으로 실행하는 데 이용하는 메소드: setInterval

`setInterval(func, delay)`

Call the function specified by *func* multiple times with intervals specified by *delay*.

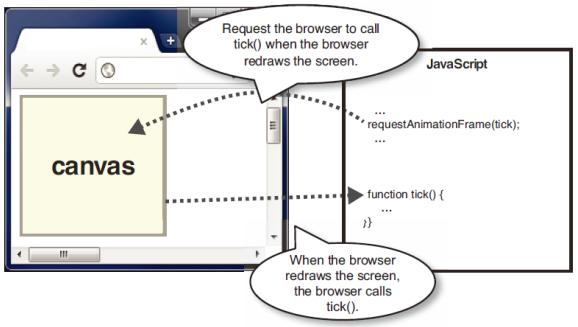
Parameters	func	Specifies the function to be called multiple times.
	delay	Specifies the intervals (in milliseconds).
Return value	Timer id	

Tab이 활성화된 상태에서만 함수를 실행하도록 하는 메소드: requestAnimationFrame

`requestAnimationFrame(func)`

Requests the function specified by *func* to be called on redraw (see Figure 4.9). This request needs to be remade after each callback.

Parameters	func	Specifies the function to be called later. The function takes a "time" parameter, indicating the timestamp of the callback.
Return value	Request id	



`cancelAnimationFrame(requestID)`

Cancel the function registered by `requestAnimationFrame()`.

Parameter	requestID	Specifies the return value of `requestAnimationFrame()`.
Return value	None	

requestAnimationFrame

Microsoft | Developer Network 로그인 MS

다운로드 프로그램 커뮤니티 설명서

··· > 웹 개발 > Internet Explorer/Microsoft Edge > 웹 플랫폼 기능의 A-Z 색인

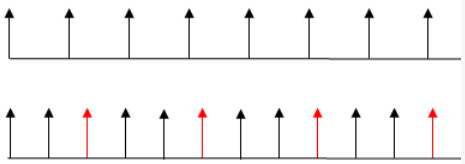
···

- 캔버스6
- 응용 프로그램 캐시 API("AppCache")01
- 응용 프로그램 캐시 API("AppCache")02
- 응용 프로그램 캐시 API("AppCache")09
- 응용 프로그램 캐시 API("AppCache")10
- 응용 프로그램 캐시 API("AppCache")13
- 응용 프로그램 캐시 API("AppCache")26
- 응용 프로그램 캐시 API("AppCache")27
- 응용 프로그램 캐시 API("AppCache")28
- 응용 프로그램 캐시 API("AppCache")29
- 응용 프로그램 캐시 API("AppCache")30
- 응용 프로그램 캐시 API("AppCache")31

스크립트 기반 애니메이션용 타이밍 컨트롤("requestAnimationFrame")

Internet Explorer 10 및 JavaScript로 작성한 Windows 스토어 앱은 **requestAnimationFrame** 메서드를 새롭게 지원합니다. 이 메서드는 시스템이 프레임을 그릴 준비가 되었을 때 애니메이션 프레임을 호출하여 애니메이션 웹 페이지를 만들 수 있는 더 유연하고 효율적인 방법을 제공합니다. 이 API 전에 **setTimeout** 및 **setInterval**을 사용하여 그린 애니메이션은 웹 개발자에게 애니메이션의 그래픽 타이머를 예약할 수 있는 효율적인 방법을 제공하지 않았습니다. 따라서 애니메이션이 과도하게 그려지고, CPU 주기가 낭비되었으며, 추가 전력이 사용되었습니다. 또한 웹 사이트가 보이지 않을 때, 특히 웹 사이트가 배경 탭의 페이지를 사용하거나 브라우저가 최소화된 경우에도 애니메이션이 종종 발생합니다.

애니메이션에서 10밀리초의 JavaScript 타이머 확인을 사용할 경우 여기 표시된 대로 타이밍 불일치가 발생합니다.



맨 위 행은 대부분의 모니터에 표시되는 16.7밀리초 디스플레이 주
적인 10밀리초 setTimeout을 나타냅니다. 디스플레이 새로 고침 간
생하기 때문에 매 세 번째 그림은 그릴 수 없습니다(빨간색 화살표
세 번째 프레임이 손실되기 때문에 애니메이션이 끊어지는 현상을
소는 또한 배터리 수명에도 좋지 않은 영향을 미치고 다른 앱의 성

기사 내용

requestAnimationFrame 사용

자바스크립트 타이머 - setTimeout, setInterval, clearInterval 함수

□ 개발&컴퓨터/JQuery & JS □ 2015.03.17 00:11

오늘은 Javascript에서 제공하는 타이머(Timer) 함수 활용에 대한 간단한 포스팅입니다.

Javascript 를 이용할 때,
1) 종종 특정 함수나 기능을 페이지가 로드되거나 버튼이 클릭되었을 때, 바로 실행하지 않고, 약간의 시간이 지난후에 실행되게 하고 싶은 경우가 있습니다. 또는
2) 특정 함수를 지속적으로 반복하여 실행하고 싶은 경우도 있구요.

예를 들면 특정 정보를 화면에 표시하여 사용자에게 안내하고, 5초 후에 다른 페이지로 이동시키고 싶은 것이라면 위의 첫번째 경우이겠죠?
그리고 10초마다 새로운 정보를 보여주기 위해 페이지의 특정 영역 프레임을 AJAX호출을 통해 지속적으로 갱신해주고 싶은 경우라면 두번째 경우일 것 같구요.

이런 처리를 위해 Javascript는 어떤 함수를 제공하고 있을까요?

바로 **setTimeout** 과 **setInterval** 함수입니다.

바로 함수 정의와 사용 방법에 대해 알아 보겠습니다.

1) setTimeout([Function], [Milliseconds])

- 특정 시간 이 후, 단 한번만 특정 함수 또는 코드를 실행시킬 때 사용합니다.
- 2번째 인자의 시간(Milliseconds)이 경과하면 1번째 인자의 함수를 실행합니다.

Rotating Trangle with Buttons

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <title>Continually Rotate A Triangle (Button)</title>
 </head>

 <body onload="main()">
  <canvas id="webgl" width="400" height="400">
  Please use a browser that supports "canvas"
  </canvas>
  <p>
   <button type="button" onclick="up()">UP</button>
   <button type="button" onclick="down()">DOWN</button>
  </p>
  <script src="../lib/webgl-utils.js"></script>
  <script src="../lib/webgl-debug.js"></script>
  <script src="../lib/cuon-utils.js"></script>
  <script src="../lib/cuon-matrix.js"></script>
  <script src="07_2 RotatingTriangle_withButtons.js"></script>
 </body>
</html>
```

WebGL

Rotating Trangle with Buttons

```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_ModelMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_ModelMatrix * a_Position;\n' +
  '}\n';
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
var ANGLE_STEP = 45.0;
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 n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the
vertices');
    return;
  }
  gl.clearColor(0, 0, 0, 1);

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

  var currentAngle = 0.0;
  var modelMatrix = new Matrix4();
  var tick = function() {
    currentAngle = animate(currentAngle);
    draw(gl, n, currentAngle, modelMatrix,
u_ModelMatrix);   // Draw the triangle
    requestAnimationFrame(tick, canvas);
  };
  tick();
}
```

Rotating Trangle with Buttons

```
function initVertexBuffers(gl) {
  var vertices = new Float32Array ([
    0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3;   // The number of vertices
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  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 -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(a_Position);
  return n;
}
```

Rotating Trangle with Buttons

```
function draw(gl, n, currentAngle, modelMatrix, u_ModelMatrix) {
  modelMatrix.setRotate(currentAngle, 0, 0, 1);
  modelMatrix.translate(0.35, 0, 0);
  gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
  gl.clear(gl.COLOR_BUFFER_BIT);
  gl.drawArrays(gl.TRIANGLES, 0, n);
}


var g_last = Date.now();
function animate(angle) {
  // Calculate the elapsed time
  var now = Date.now();
  var elapsed = now - g_last;
  g_last = now;
  var newAngle = angle + (ANGLE_STEP * elapsed) / 1000.0;
  return newAngle %= 360;
}
function up() {     ANGLE_STEP += 10;    }
function down() {    ANGLE_STEP -= 10;   }
```