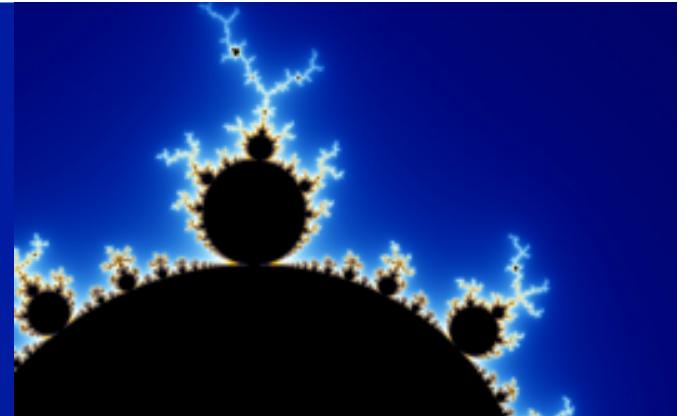
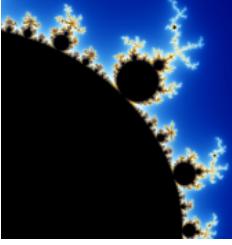


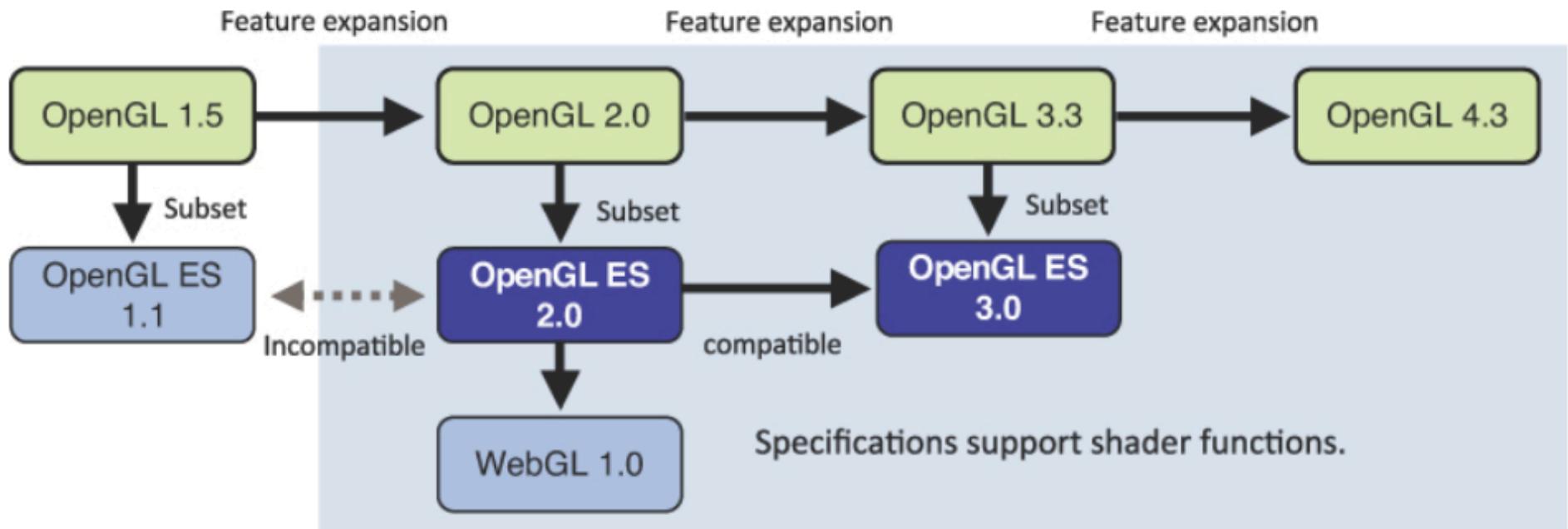
Computer Graphics

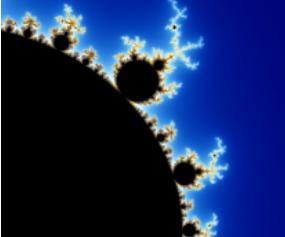


Programming with WebGL Getting Started

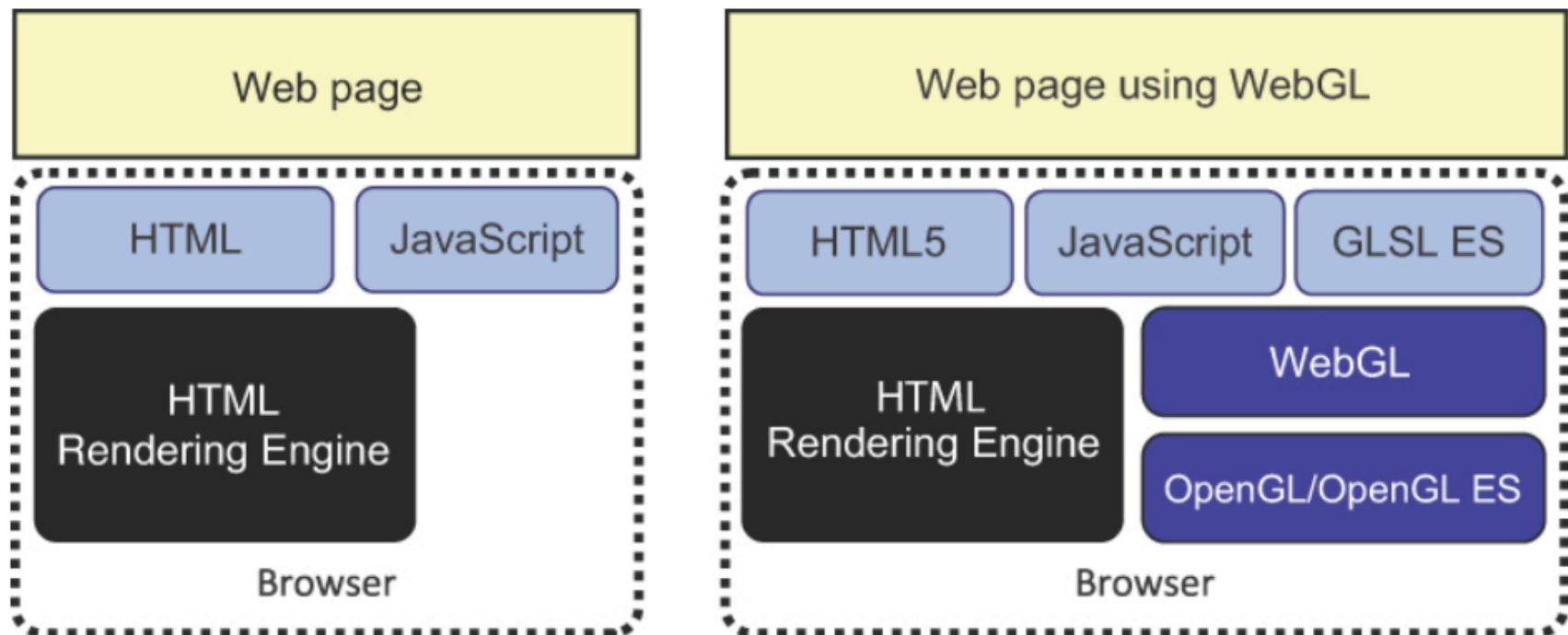


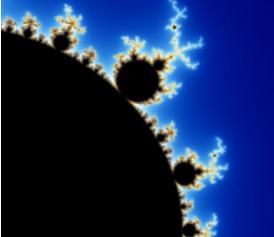
WebGL



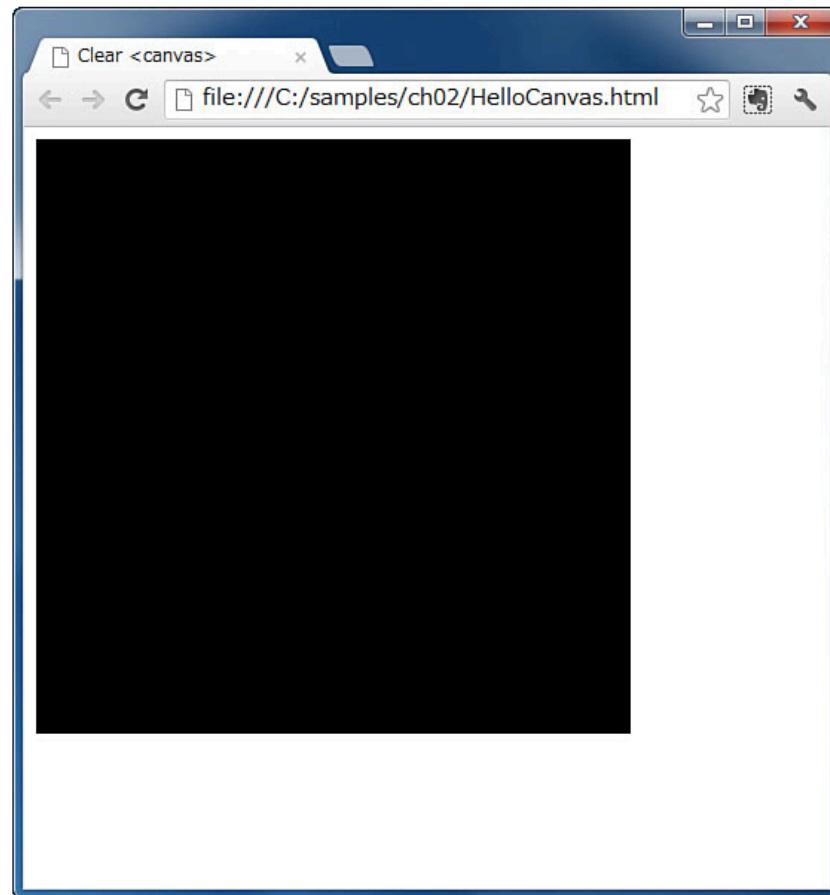


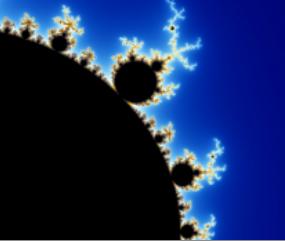
Execution in Browser





Example: Hello Canvas





Example: Hello Canvas

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

<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main() {
    gl_Position = vPosition;
}
</script>

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

void main() {
    gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}
</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="1-HelloCanvas.js"></script>
</head>

<body onload="main()">
<canvas id="webgl" width="400" height="400">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```

HelloCanvas.html

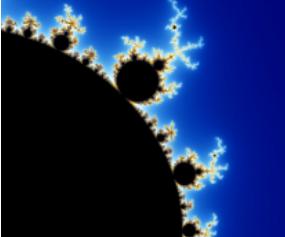
HelloCanvas.js

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

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

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

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



Example: Hello Canvas

HelloCanvas.html

- onload function
- Supporting libraries/files
- javascript source file
- Console in Chrome

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

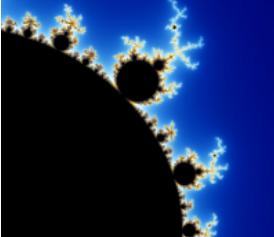
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main() {
    gl_Position = vPosition;
}
</script>

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

void main() {
    gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}
</script>

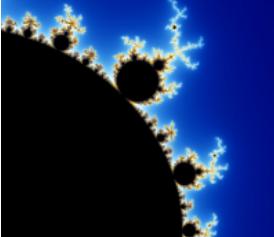
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="l-HelloCanvas.js"></script>
</head>

<body onload="main()">
<canvas id="webgl" width="400" height="400">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



Files

- `./Common/webgl-utils.js`: Standard utilities for setting up WebGL context in Common directory on website
- `./Common/initShaders.js`: contains JS and WebGL code for reading, compiling and linking the shaders
- `./Common/MV.js`: our matrix-vector package
- `square.js`: the application file



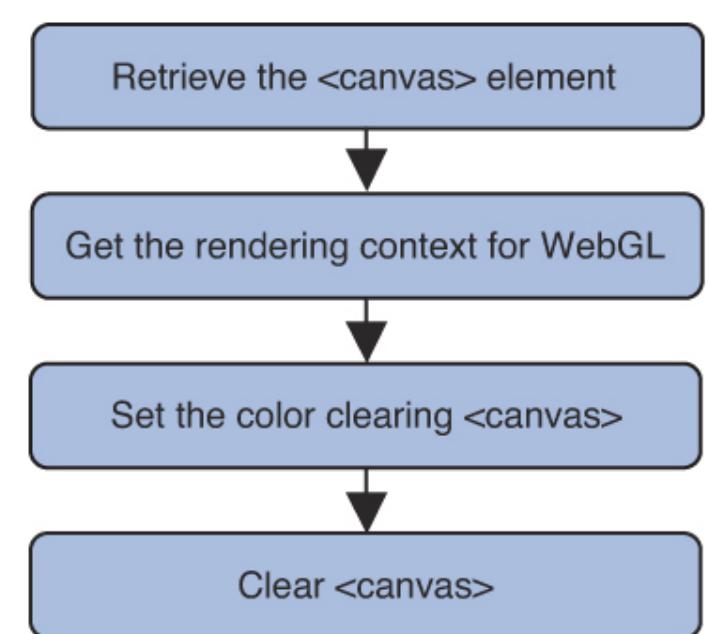
Example: Hello Canvas

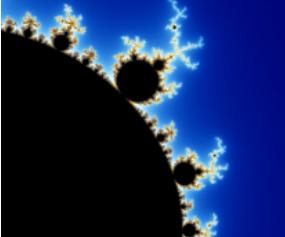
```
function main() {
    // Retrieve <canvas> element
    var canvas = document.getElementById('webgl');

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

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

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



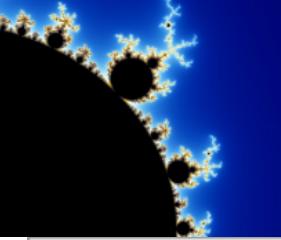


gl.clearColor

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 is truncated into
		(1.0, 0.0, 0.0, 1.0) red
Return value	None	(0.0, 1.0, 0.0, 1.0) green
Errors²	None	(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



gl.clear(buffer)

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.

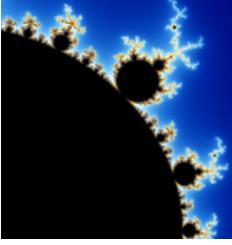
`gl.COLOR_BUFFER_BIT` Specifies the color buffer.

`gl.DEPTH_BUFFER_BIT` Specifies the depth buffer.

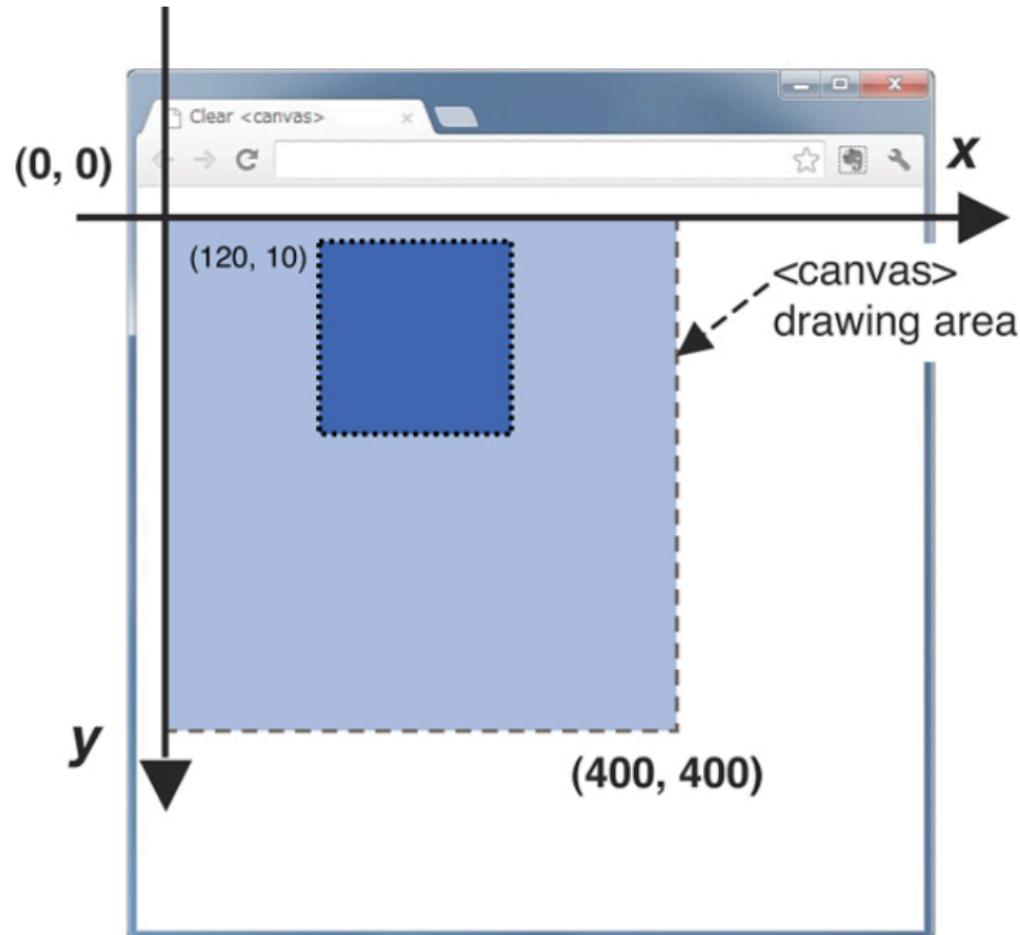
`gl.STENCIL_BUFFER_BIT` Specifies the stencil buffer.

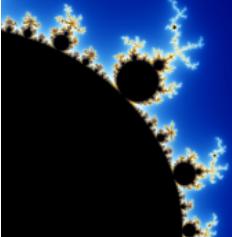
Return value None

Errors `INVALID_VALUE` *buffer* is none of the preceding three values.



Example: Draw Rectangle





Example: Draw Rectangle

```
<!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="200" height="150">
      Please use a browser that supports "canvas"
    </canvas>
    <script src="DrawRectangle.js"></script> ←
  </body>
</html>
```

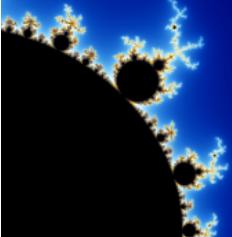
DrawRectangle.html

```
// DrawRectangle.js
function main() {
  // Retrieve the <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');

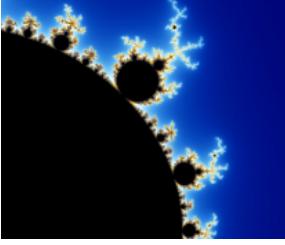
  // Draw a blue rectangle
  ctx.fillStyle = 'rgba(0, 0, 255, 1.0)'; // Set a blue color
  ctx.fillRect(120, 10, 150, 150);      // Fill a rectangle with the color
}
```

DrawRectangle.js



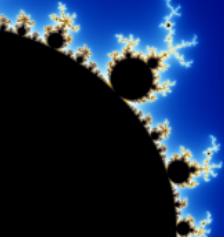
Example: Hello Point

- Adding the Vertex Shader and the Fragment Shader
- Vertex Shader: Program that describes the traits (position, colors, and others) of a vertex
 - Examples of a vertex
- Fragment Shader: Program that deals with per fragment processing, such as lighting
 - What is a fragment?
 - Must set precision in fragment shader

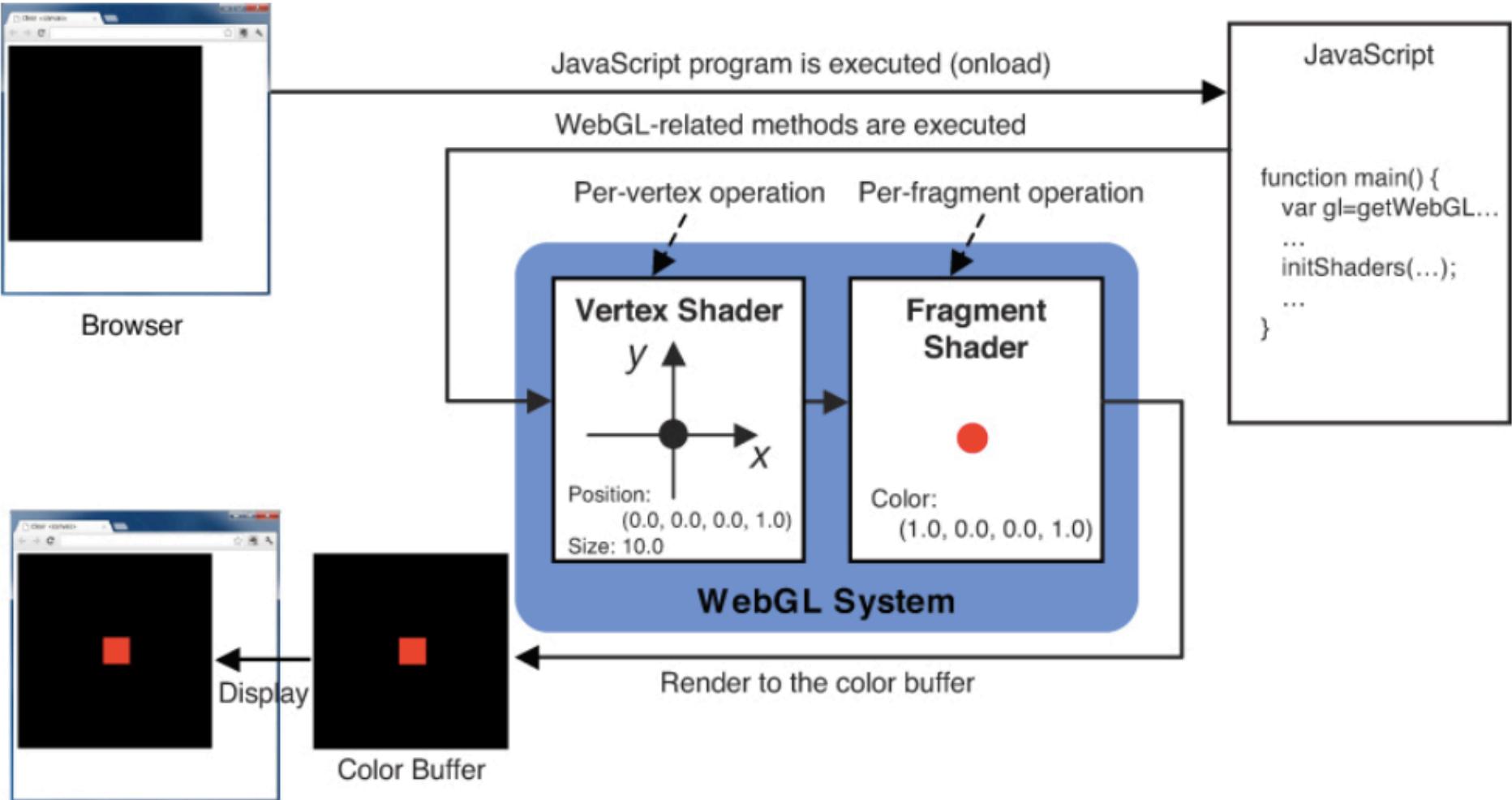


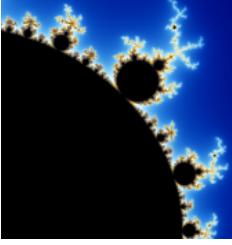
GLSL

- OpenGL Shading Language
- C-like with
 - Matrix and vector types (2, 3, 4 dimensional)
 - Overloaded operators
 - C++ like constructors
- Similar to Nvidia's Cg and Microsoft HLSL
- Code sent to shaders as source code
- WebGL functions compile, link and get information to shaders

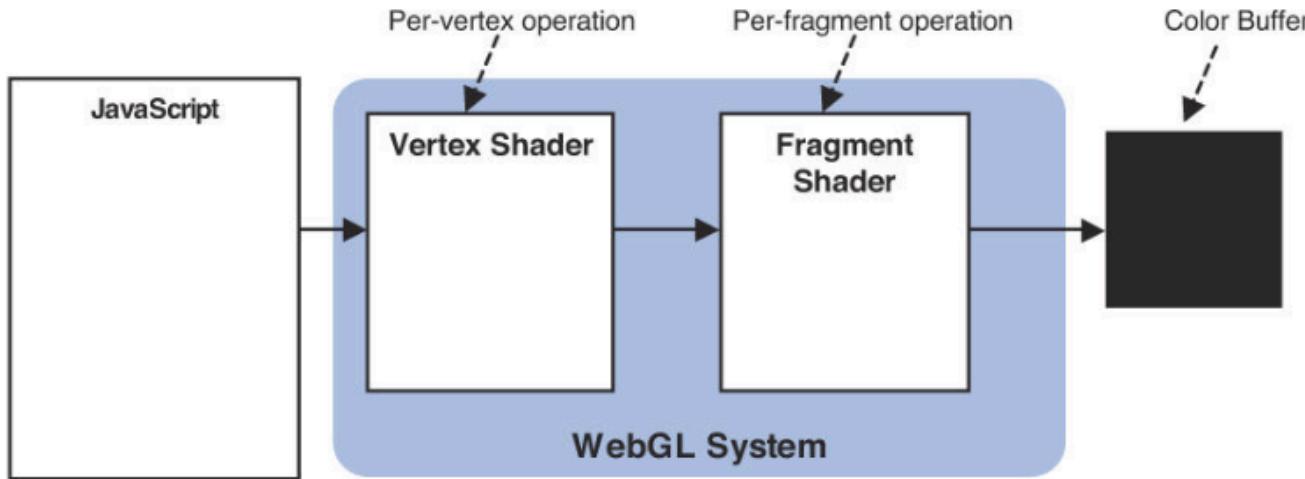


Example: Hello Point

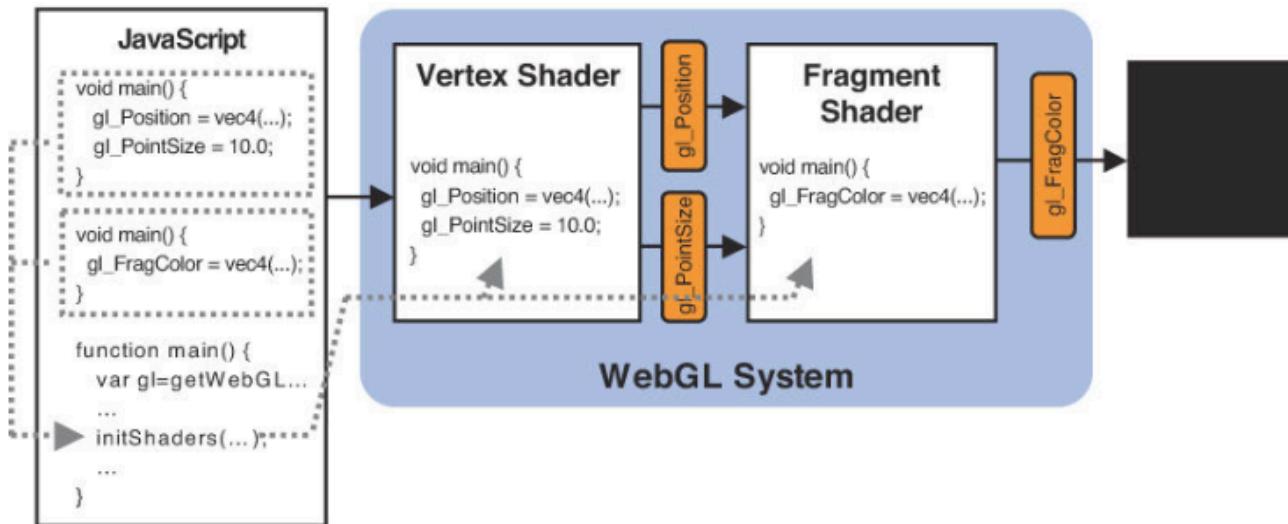


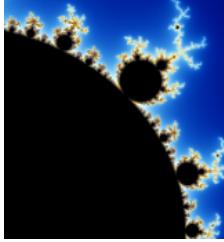


Behavior of initShader



WebGL applications consist of a javascript program that is executed by the browser and the shader program executed within the WebGL system



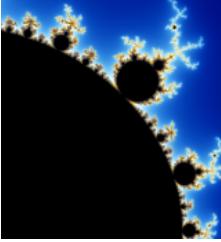


Hello Point (shader programs)

HelloPoint1.html

```
<script id="vertex-shader" type="x-shader/x-vertex">
// Vertex shader program
void main() {
    gl_Position = vec4(0.0, 0.0, 0.0, 1.0); // Set the vertex coordinate
    gl_PointSize = 10.0; // Set the point size
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
// Fragment shader program
void main() {
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0); // Set the point color
}
</script>
```



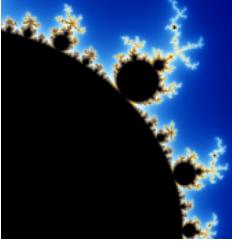
Hello Point (shader programs)

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)

[Built in values in the vertex shader](#)

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

[Built in values in the fragment shader](#)



Hello Point (shader programs)

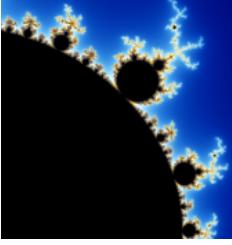
Type	Description				
float	Indicates a floating point number				
vec4	Indicates a vector of four floating point numbers				
<table border="1"><tr><td>float</td><td>float</td><td>Float</td><td>float</td></tr></table>		float	float	Float	float
float	float	Float	float		

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

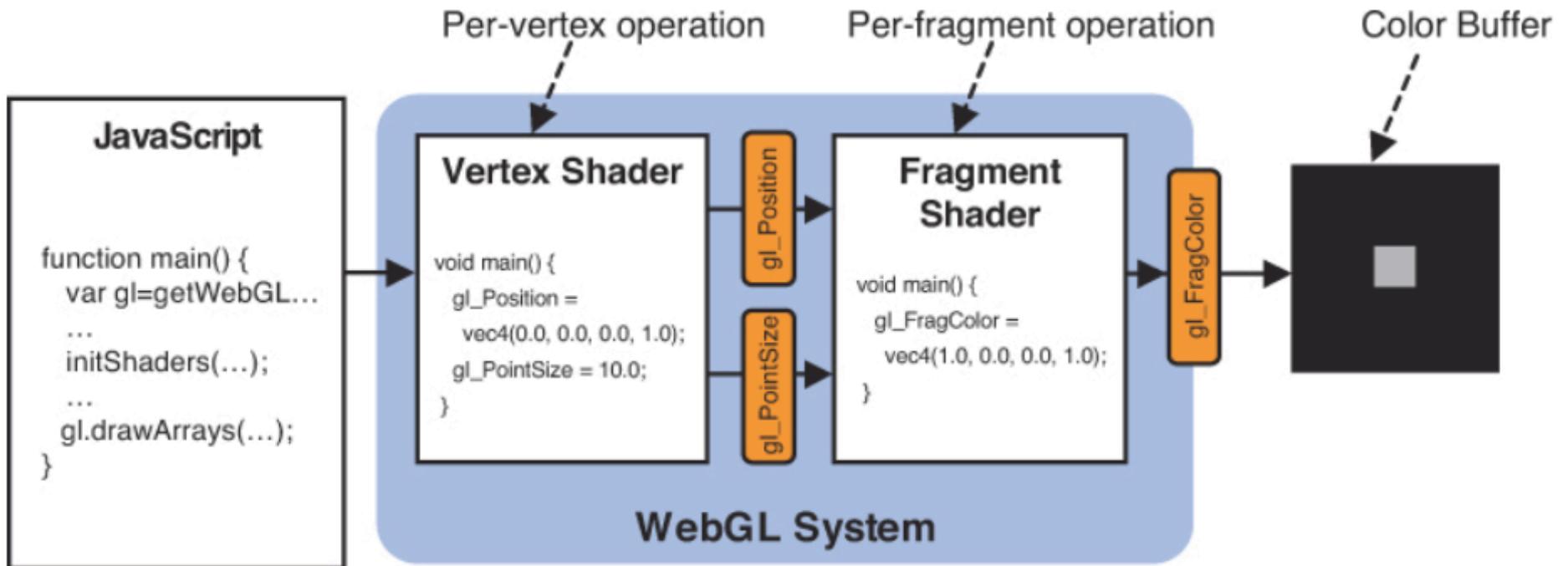
Construct a `vec4` object from v_0 , v_1 , v_2 , and v_3 .

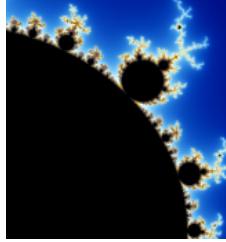
Parameters v_0, v_1, v_2, v_3 Specifies floating point numbers.

Return value A `vec4` object made from v_0 , v_1 , v_2 , and v_3 .

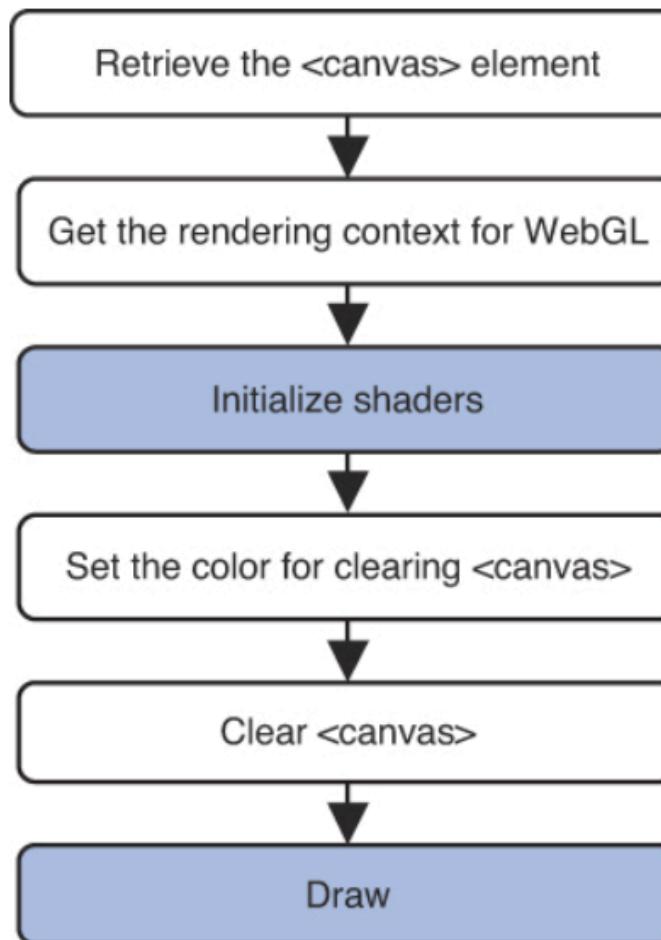


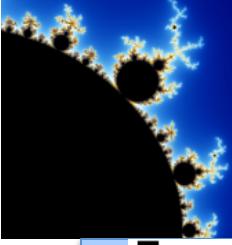
Variables in shader programs





Hello Point (Javascript)





Hello Point (Javascript)

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

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

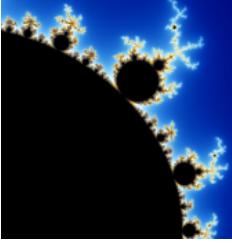
    // Initialize shaders
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    if (!program) {
        console.log('Failed to initialize shaders.');
        return;
    }
    gl.useProgram( program );

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

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

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

HelloPoint1.js



Hello Point : gl.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 which vertex to start drawing from (integer).
-------	---

count	Specifies the number of vertices to be used (integer).
-------	--

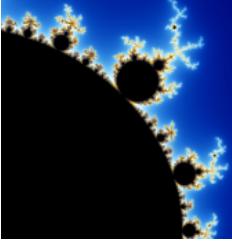
Return value

None

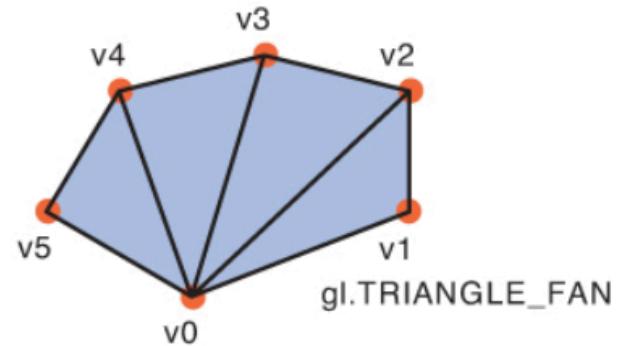
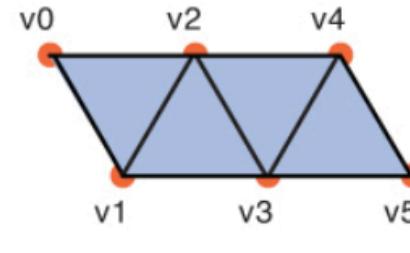
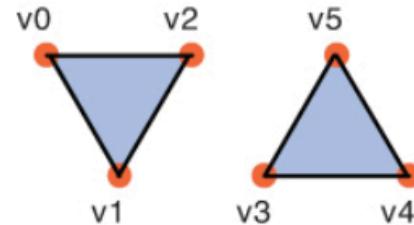
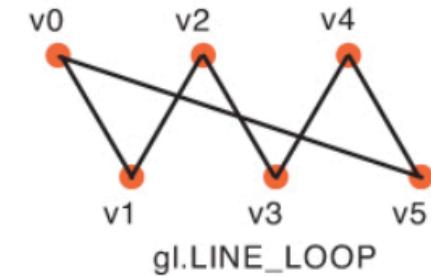
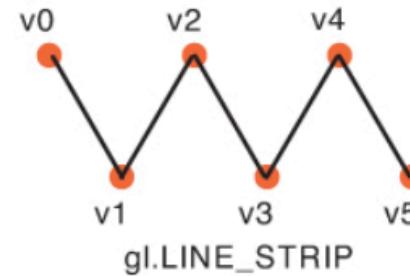
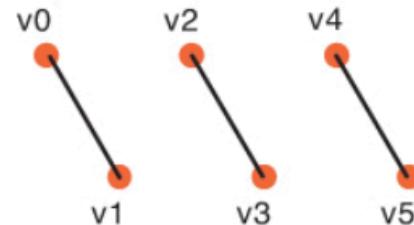
Errors

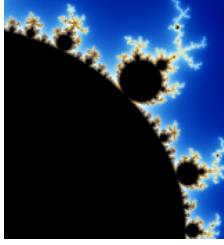
INVALID_ENUM *mode* is none of the preceding values.

INVALID_VALUE *first* is negative or *count* is negative.



Hello Point : gl.drawArrays



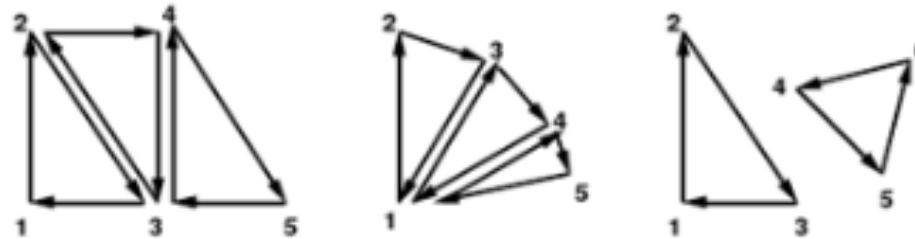


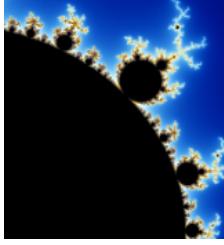
Hello Point : gl.drawArrays

Points

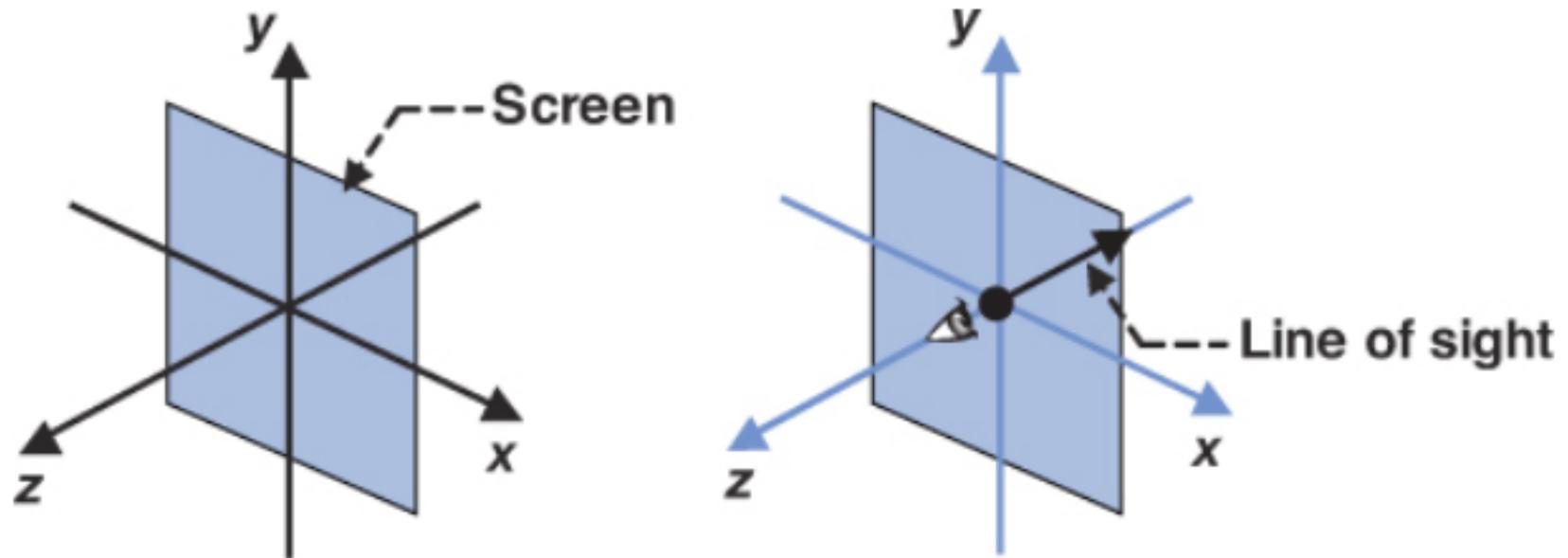
Lines: Strips (connected), Loops (closed), Lines (separate)

Triangle: Strips (shared common edge), Fans (shared first vertex), Triangles (separate)

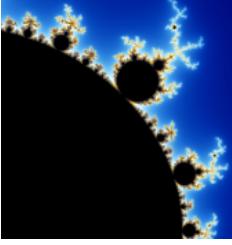




WebGL Coordinate System

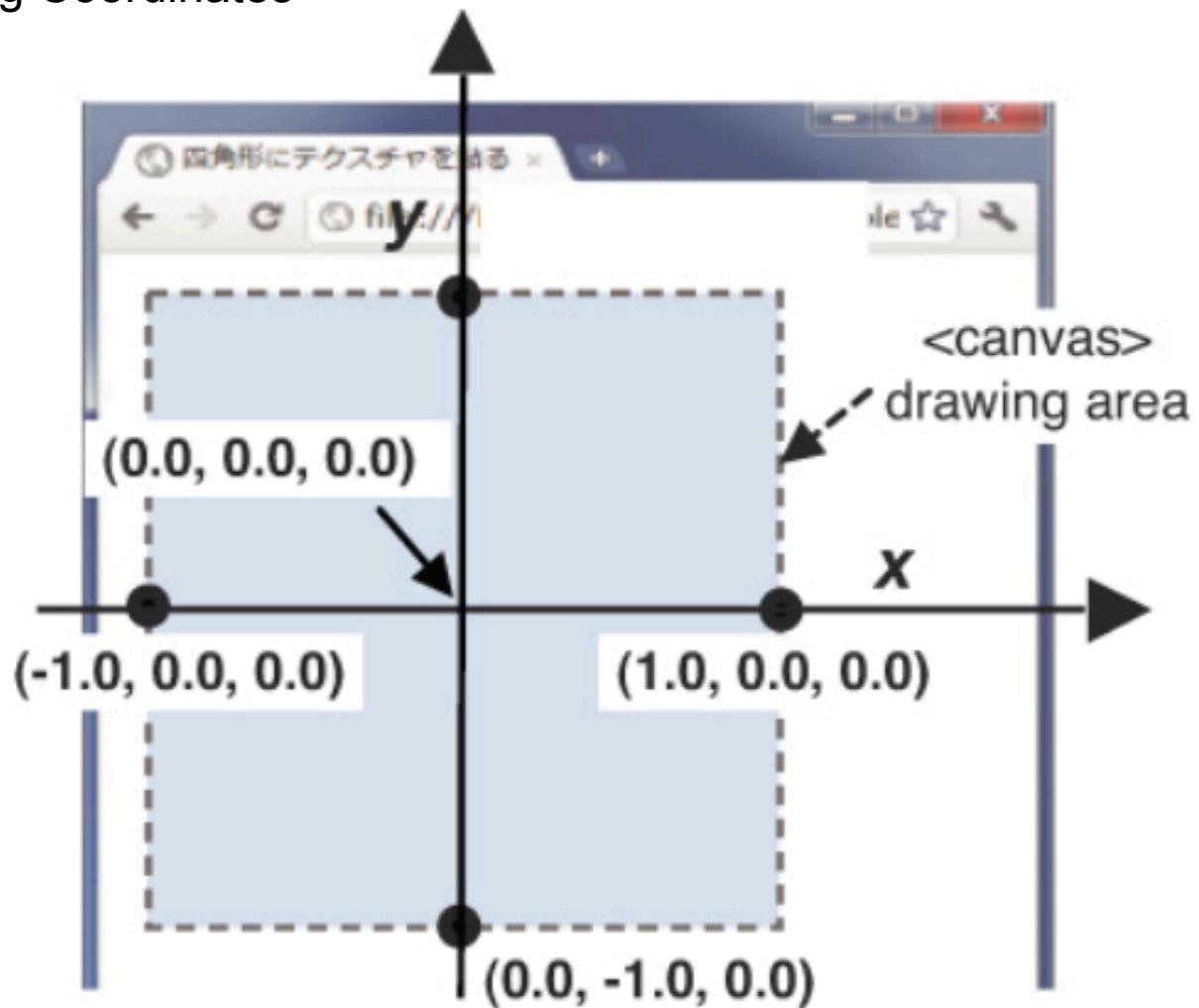


Use Right-Hand-Rule



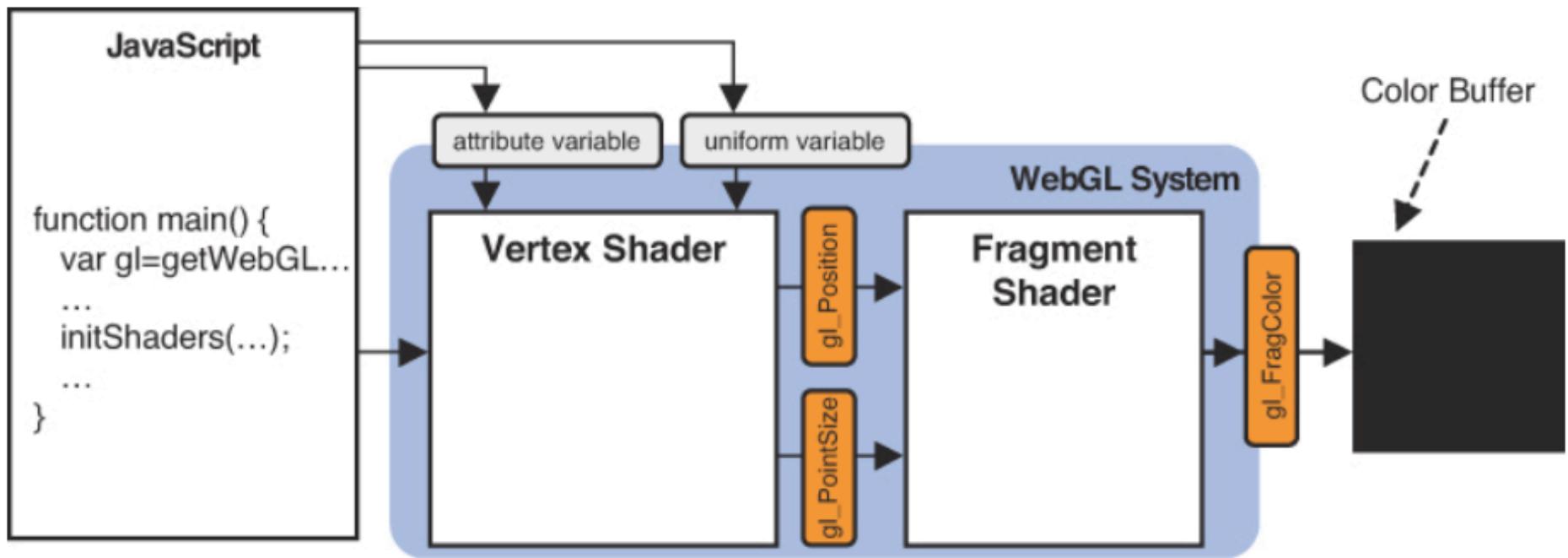
WebGL Coordinate System

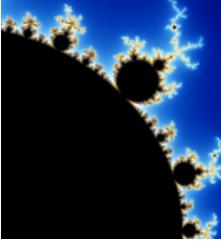
The Clipping Coordinates



Example: Hello Point 2 program

- How to pass data to the Vertex Shader?
 - attribute variable
 - uniform variable



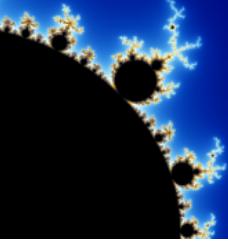


Example: Hello Point 2 program

- How to pass data to the Vertex Shader?
 - attribute variable: pass data that is different for each vertex
 - Examples:

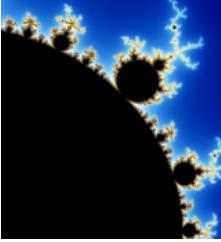
```
attribute float temperature;
attribute vec3 velocity;
```
 - uniform variable: pass data that is the same (or the uniform) for each vertex
 - Variables do not change across the primitive being processed
 - Used to pass information to shader such as the time or a bounding box of a primitive or transformation matrices
 - Examples:

```
uniform float angle;
```



Example: Hello Point 2 program

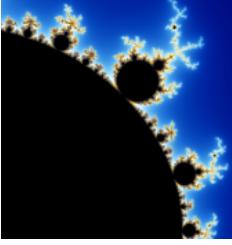
- Steps to pass data to the Vertex Shader
 1. Prepare the attribute variable for the vertex position in the vertex shader
 2. Assign the attribute variable to the `gl_Position` variable
 3. Pass the data to the attribute variable



Hello Point 2 : Vertex Shader

- Step 1:

```
<script id="vertex-shader" type="x-shader/x-vertex">
// Vertex shader program
attribute vec4 a_Position;
void main() {
    gl_Position = a_Position; // Set the vertex coordinates
    gl_PointSize = 15.0; // Set the point size
}
</script>
```



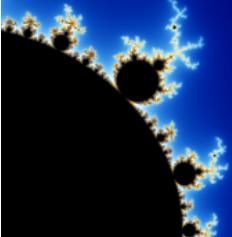
Hello Point 2 : Javascript program

- Step 2:

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

- Step 3:

```
// Pass vertex position to attribute variable
gl.vertexAttrib3f(a_Position, 0.0, 0.0, 0.0);
```

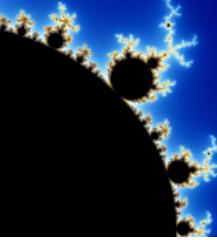


gl.getAttribLocation

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 <code>gl_</code> or <code>webgl_</code> .
Errors	INVALID_OPERATION	<i>program</i> has not been successfully linked (See Chapter 9.)
	INVALID_VALUE	The length of <i>name</i> is more than the maximum length (256 by default) of an attribute variable name.



gl.vertexAttrib3f

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

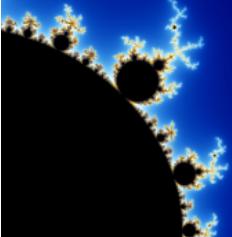
Assign the data (v_0 , v_1 , and v_2) to the attribute variable specified by *location*.

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.

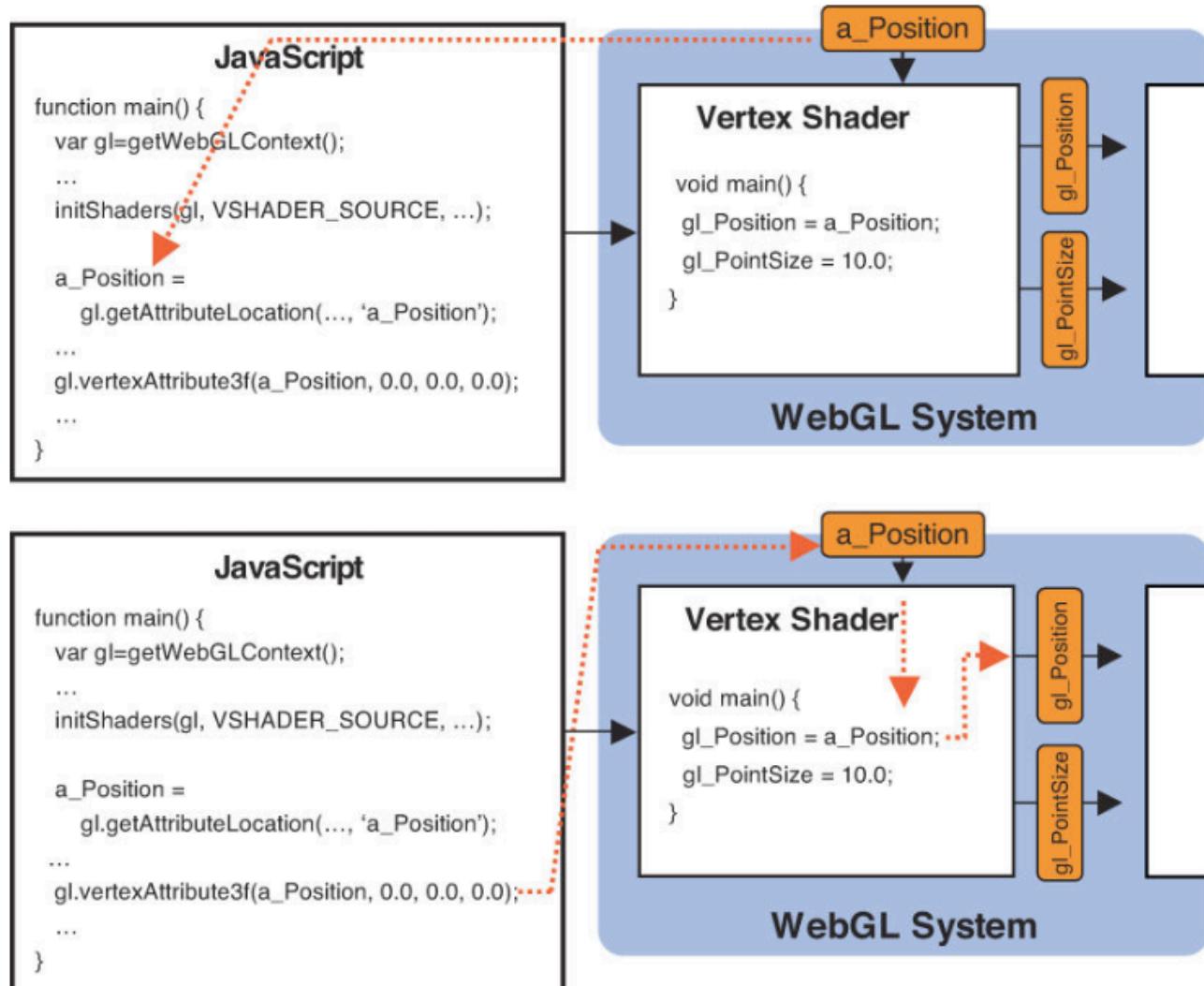
Return value None

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).



Getting the storage location and writing a value to the variable



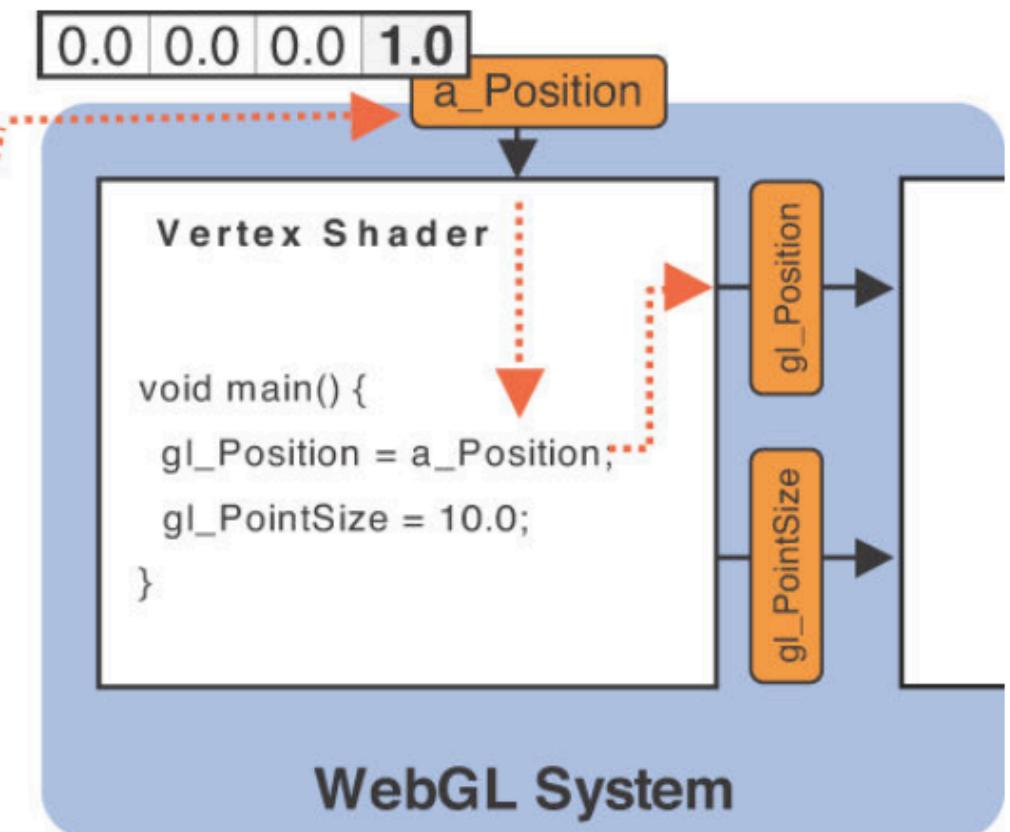
The missing data is auto-supplied

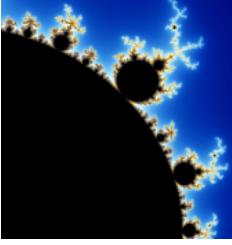
```
JavaScript

function main() {
    var gl=getWebGLContext();
    ...
    initShaders(gl, VS_SHADER_SOURCE, ...);

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

0.0 | 0.0 | 0.0





gl.vertexAttrib3f

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

base method name

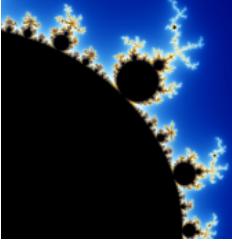
parameter type:

“*f*” indicates floating point numbers and

“*i*” indicates integer numbers

number of parameters

```
gl.vertexAttrib1f(a_Position, 0.5);  
gl.vertexAttrib2f(a_Position, 0.5, 0.0);  
gl.vertexAttrib4f(a_Position, 0.5, 0.0, 0.0, 1.0);
```



Example: ClickPoints

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

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);

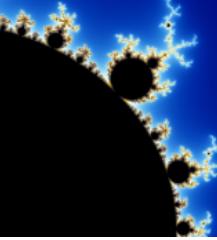
    // Store the coordinates to the "points" array
    points.push(x); points.push(y);

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

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

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

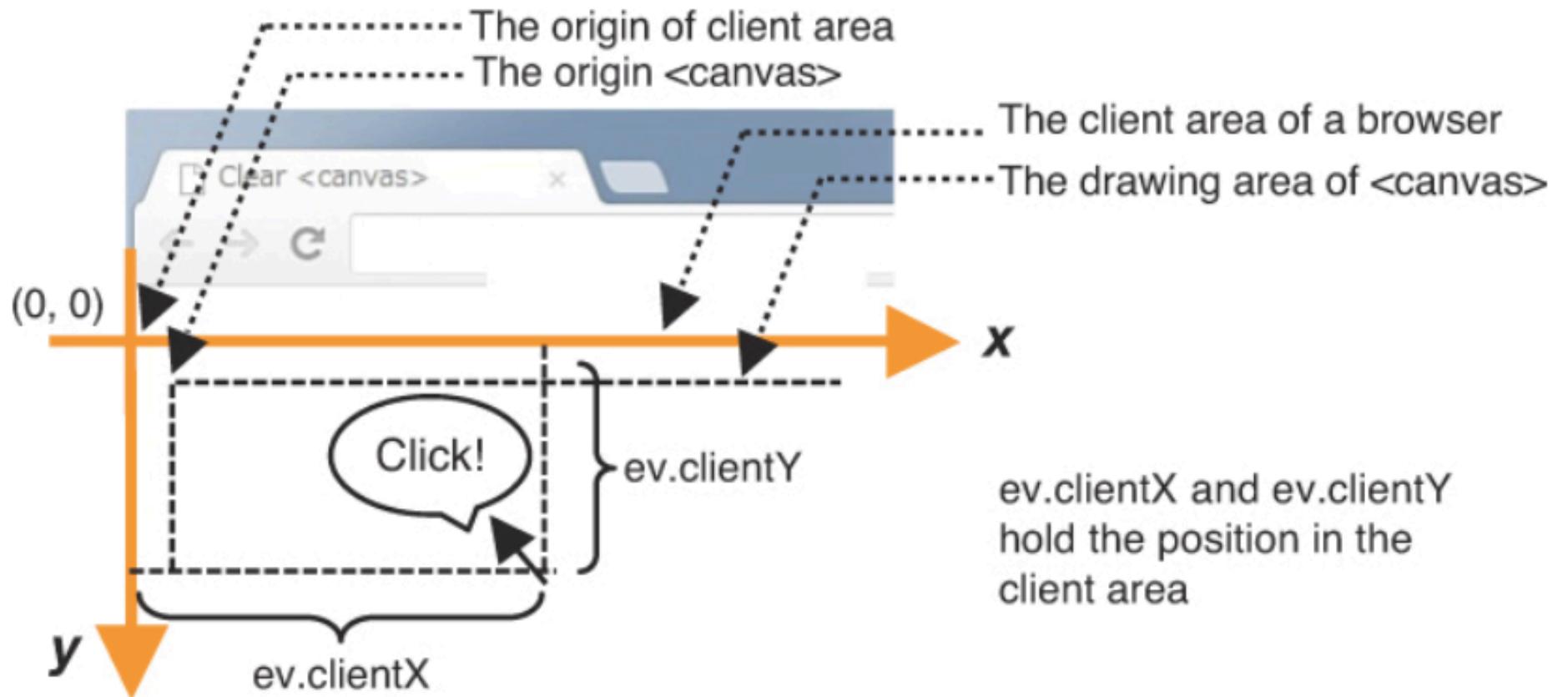
Register event handler

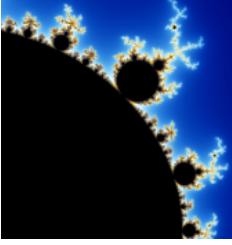


Example: ClickPoints

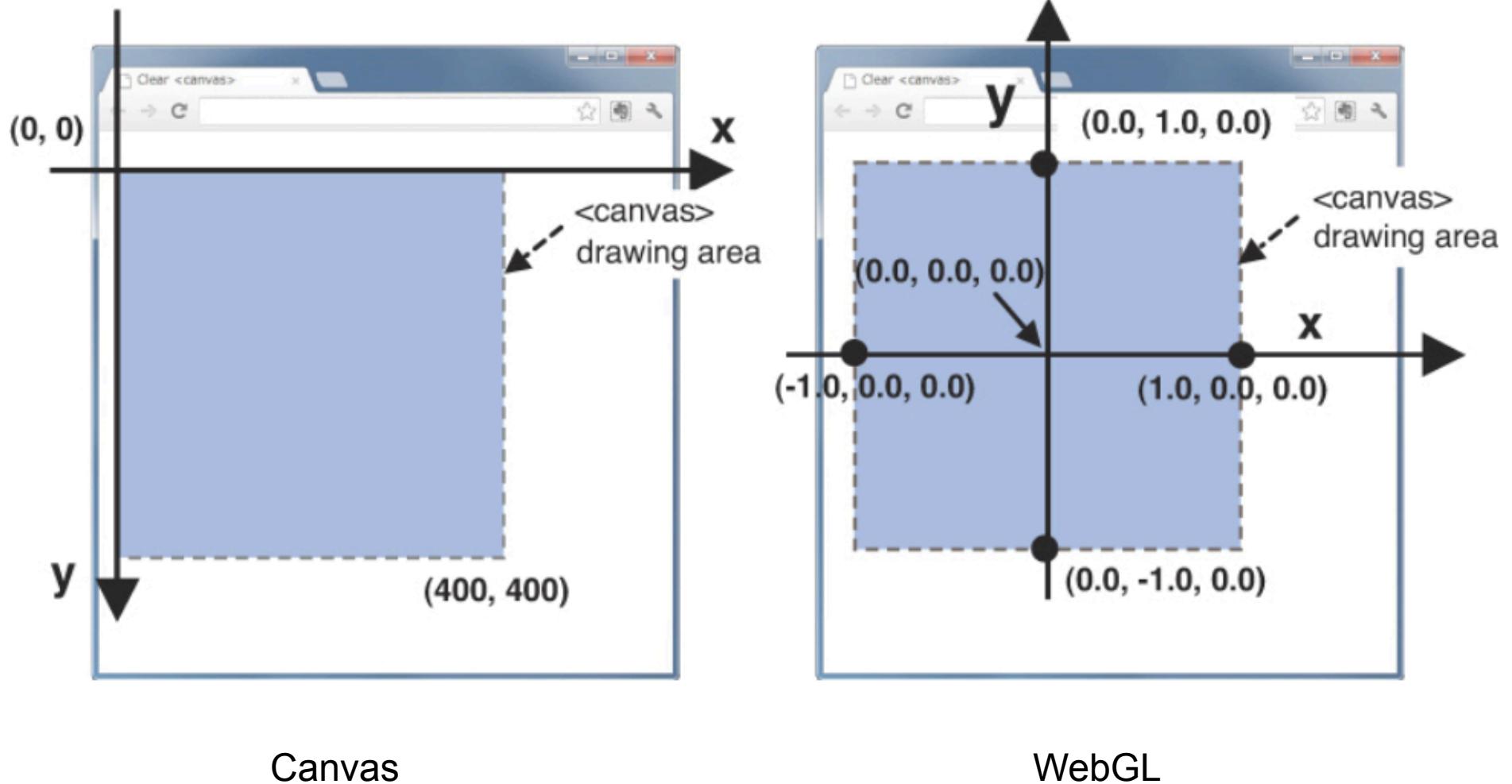
- How to transform the mouse click coordinates into the webGL coordinates and display the corresponding points?
- Can not use the mouse click coordinates directly
 - The coordinate is the position in the “client area” in the browser, not in the “canvas”
 - The coordinate system of the “canvas” is different from that of WebGL in terms of its origin and the direction of the y-axis

Coordinate system of the browser's client area and the position of the canvas



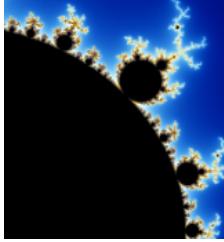


The coordinate system of the canvas and that of WebGL on canvas

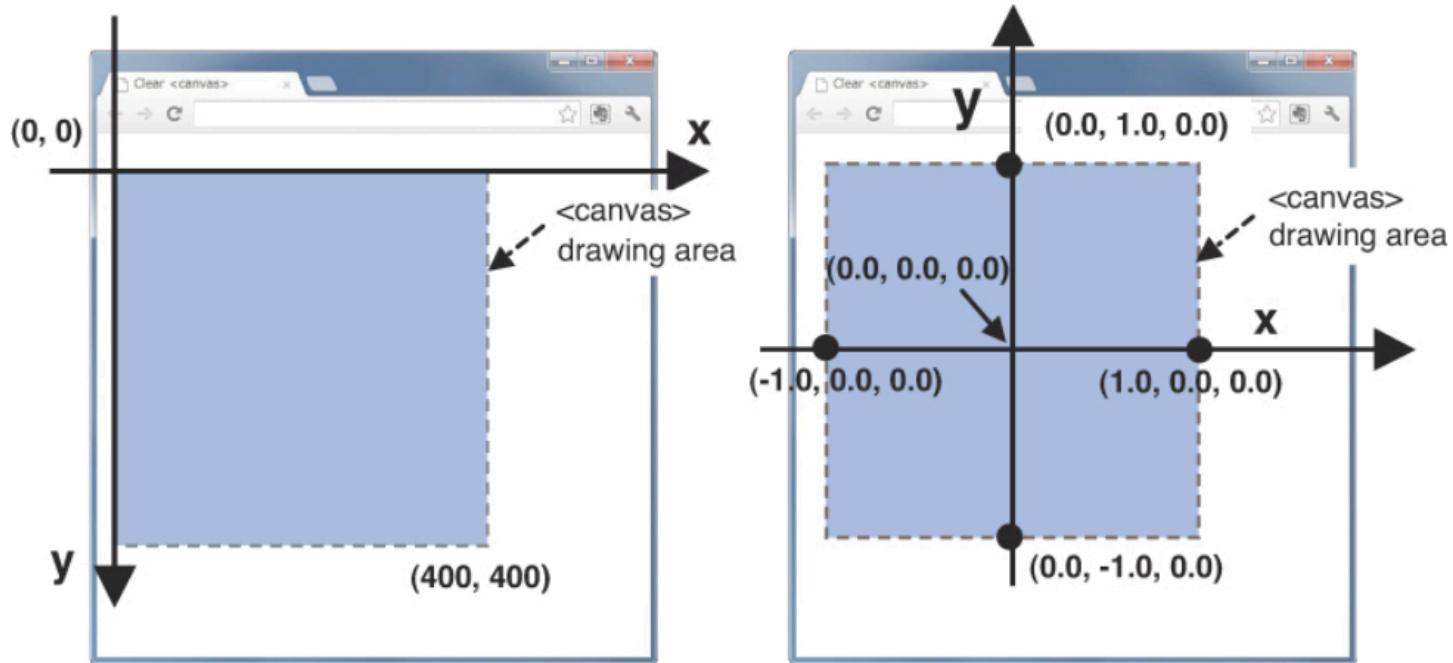


Canvas

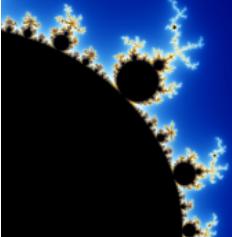
WebGL



The coordinate system of the canvas and that of WebGL on canvas

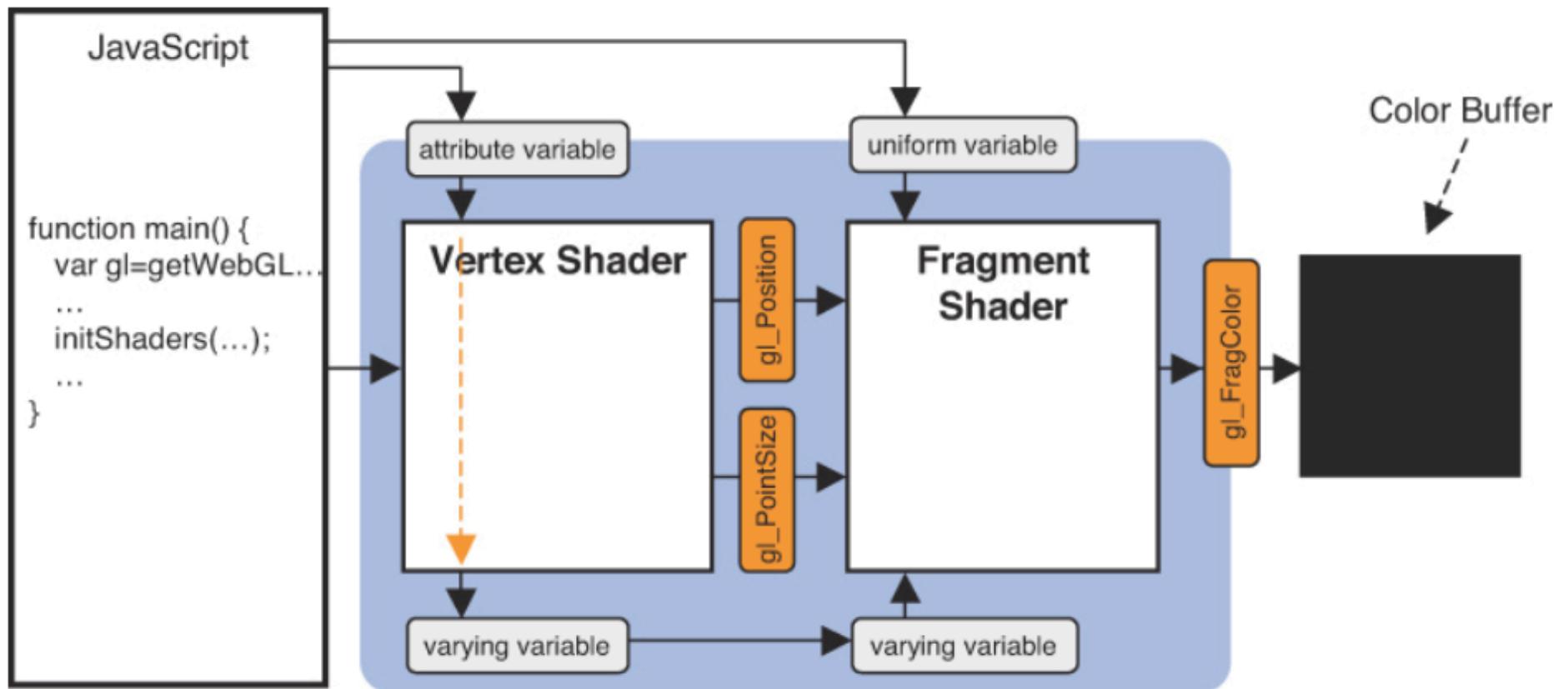


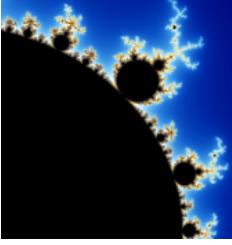
```
var x = ev.clientX; // x coordinate of a mouse pointer  
var y = ev.clientY; // y coordinate of a mouse pointer  
var rect = ev.target.getBoundingClientRect();  
  
x = ((x - rect.left) - canvas.width/2)/(canvas.width/2);  
y = (canvas.height/2 - (y - rect.top))/(canvas.height/2);
```



Example: Color Points

- Adding uniform variable to Fragment Shader → two methods

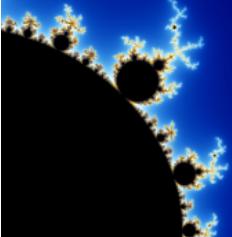




Example: Color Points

Fragment Shader:

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

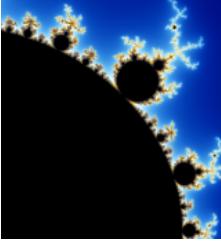


Example: Color Points – js

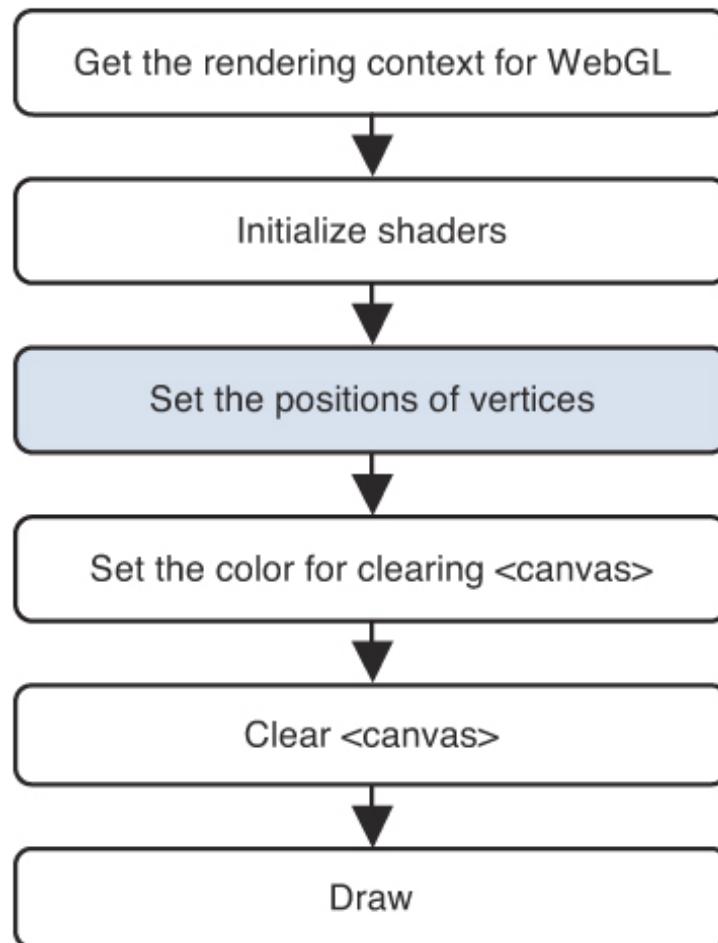
```
// Get the storage location of u_FragColor
var u_FragColor = gl.getUniformLocation(program, "u_FragColor");
if (!u_FragColor) {
    console.log('Failed to get the storage location of u_FragColor');
    return;
}

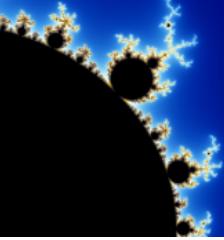
// Store the colors to the "colors" array
if (x >= 0.0 && y >= 0.0) {          // First quadrant
    colors.push([1.0, 0.0, 0.0, 1.0]);  // Red
} else if (x < 0.0 && y >= 0.0) { // Second quadrant
    colors.push([0.0, 0.0, 1.0, 1.0]); // Blue
} else if (x < 0.0 && y < 0.0) { // Third quadrant
    colors.push([0.0, 1.0, 0.0, 1.0]); // Green
} else if (x >=0.0 && y< 0.0) { // Fourth quadrant
    colors.push([1.0, 1.0, 1.0, 1.0]); // White
}

// Pass the color of a point to u_FragColor variable
gl.uniform4f(u_FragColor, rgba[0], rgba[1], rgba[2], rgba[3]);
```

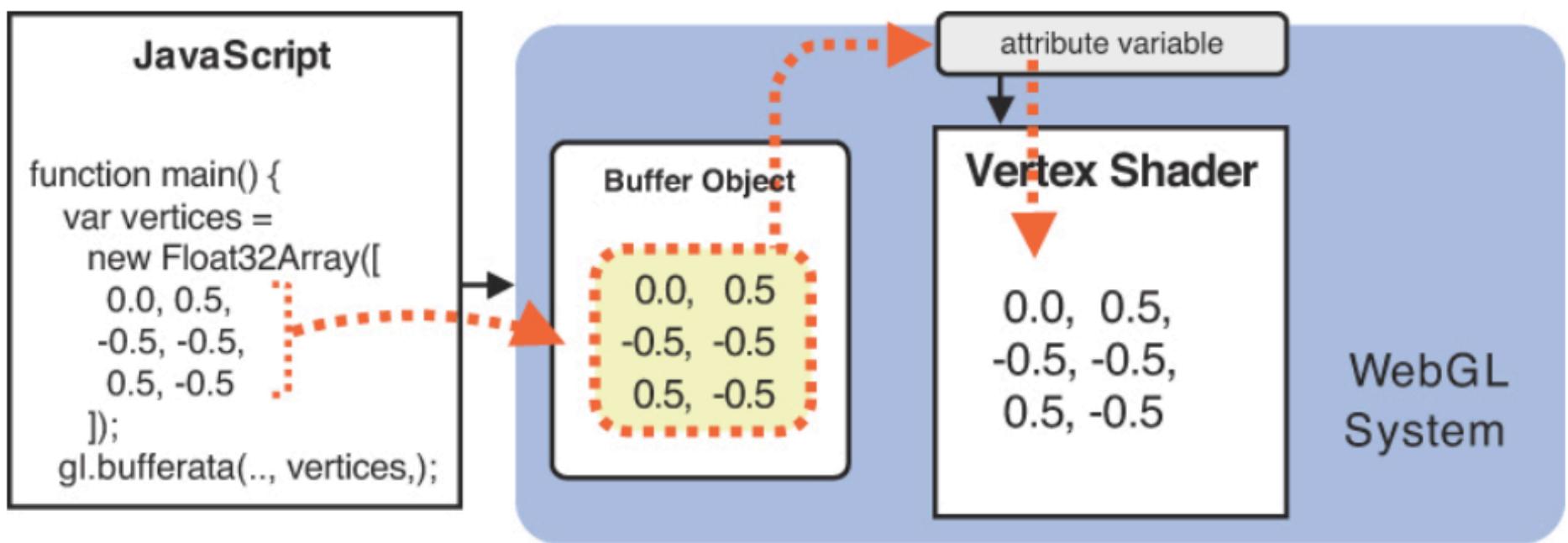


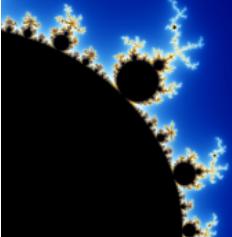
Example: Draw multiple points



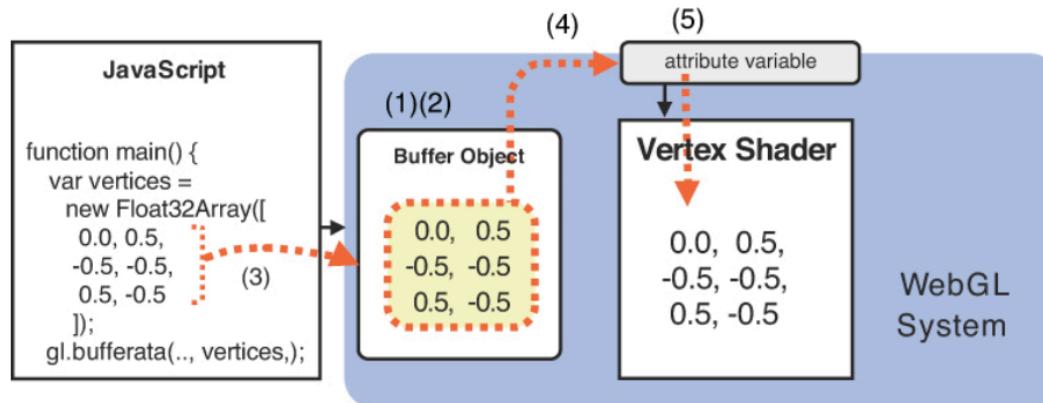


Example: Draw multiple points

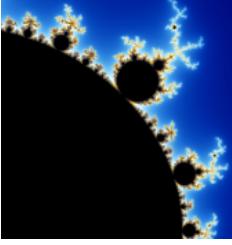




Steps to pass multiple data to vertex shader through a buffer object



1. Create a buffer object (`gl.createBuffer()`).
2. Bind the buffer object to a target (`gl.bindBuffer()`).
3. Write data into the buffer object (`gl.bufferData()`).
4. Assign the buffer object to an attribute variable (`gl.vertexAttribPointer()`).
5. Enable assignment (`gl.enableVertexAttribArray()`).

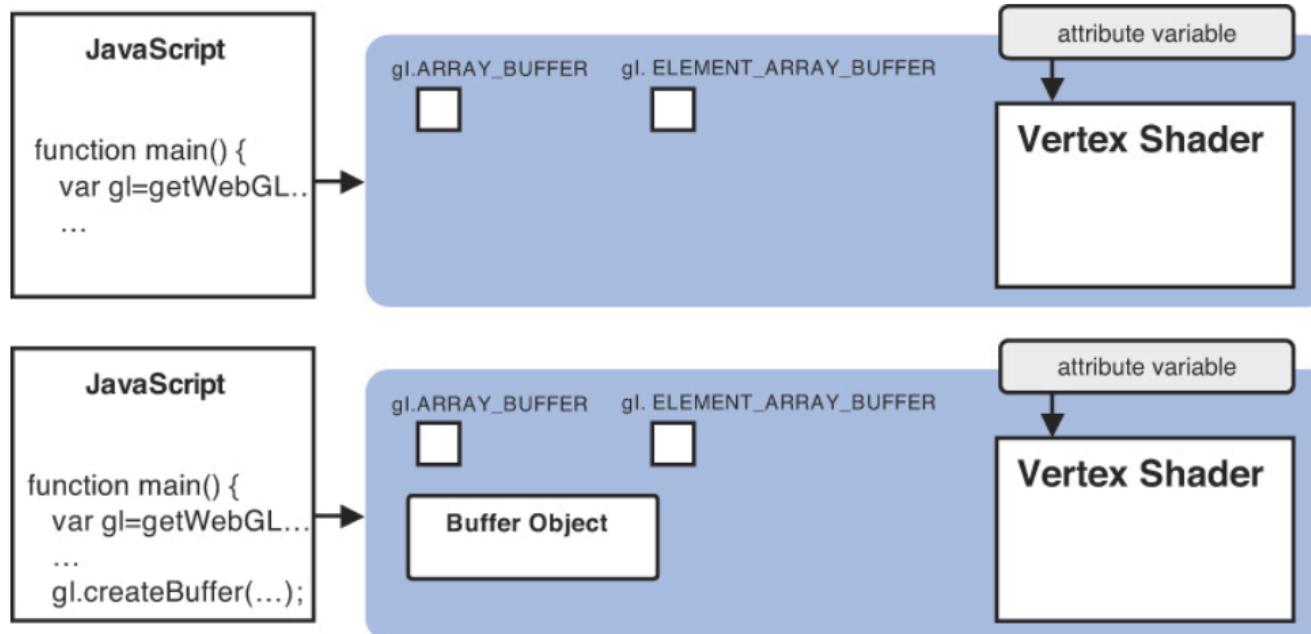


Create buffer

gl.createBuffer ()

Create a buffer object.

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



Bind Buffer

`gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);`

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.

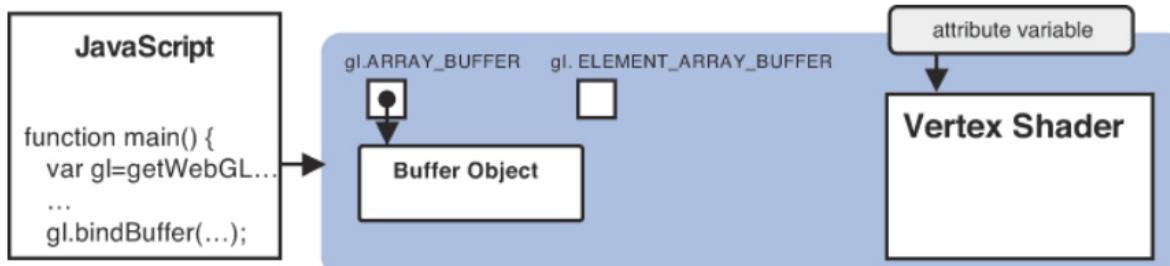
`gl.ELEMENT_ARRAY_BUFFER` Specifies that the buffer object contains index values pointing to vertex data. (See Chapter 6, “The OpenGL ES Shading Language [GLSL ES].”)

`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.



Write data into buffer

`gl.bufferData(gl.ARRAY_BUFFER, vertices, glSTATIC_DRAW);`

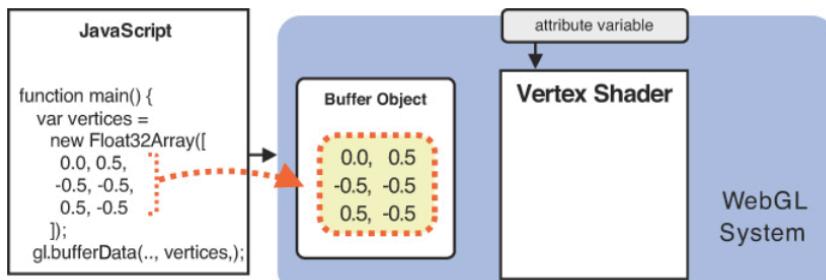
`gl.bufferData(target, data, usage)`

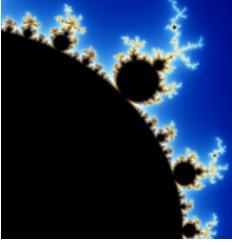
Allocate storage and write the *data* specified by *data* to the buffer object bound to *target*.

Parameters	target	Specifies <code>gl.ARRAY_BUFFER</code> or <code>gl.ELEMENT_ARRAY_BUFFER</code> .
	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.
	<code>glSTATIC_DRAW</code>	The buffer object data will be specified once and used many times to draw shapes.
	<code>glSTREAM_DRAW</code>	The buffer object data will be specified once and used a few times to draw shapes.
	<code>glDYNAMIC_DRAW</code>	The buffer object data will be specified repeatedly and used many times to draw shapes.

Return value None

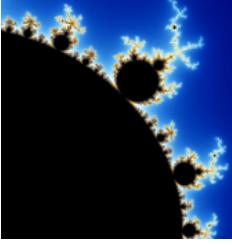
Errors `INVALID_ENUM` *target* is none of the preceding constants





Typed Arrays used in WebGL

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)



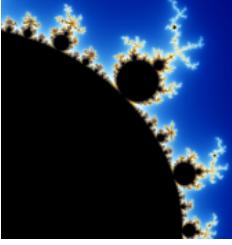
Assign the buffer object to an attribute variable

```
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 <code>size</code> is less than the number of components required by the attribute variable, the missing components are automatically supplied just like <code>gl.vertexAttrib[1234]f()</code> . For example, if <code>size</code> 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:
	<code>gl.UNSIGNED_BYTE</code>	unsigned byte for <code>Uint8Array</code>
	<code>gl.SHORT</code>	signed short integer for <code>Int16Array</code>
	<code>gl.UNSIGNED_SHORT</code>	unsigned short integer for <code>Uint16Array</code>
	<code>gl.INT</code>	signed integer for <code>Int32Array</code>
	<code>gl.UNSIGNED_INT</code>	unsigned integer for <code>Uint32Array</code>
	<code>gl.FLOAT</code>	floating point number for <code>Float32Array</code>



Assign the buffer object to an attribute variable

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

normalized Either `true` or `false` to indicate whether nonfloating data should be normalized to $[0, 1]$ or $[-1, 1]$.

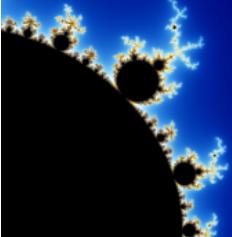
stride Specifies the number of bytes between different vertex data elements, or zero for default stride (see Chapter 4).

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.

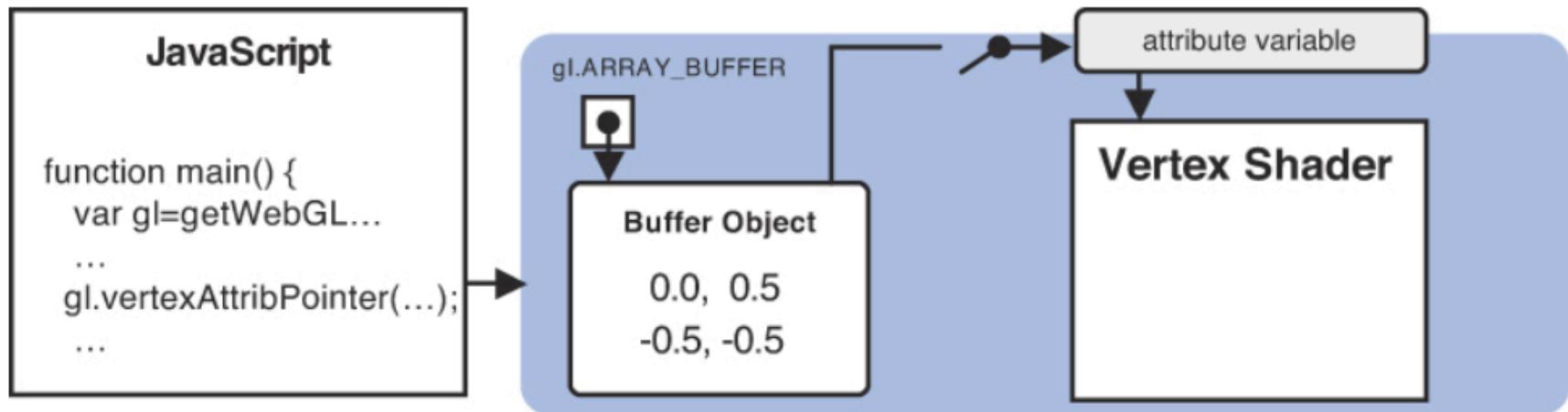
Return value None

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.



Enable the assignment to an attribute variable gl.enableVertexAttribArray(a_Position);



gl.enableVertexAttribArray(location)

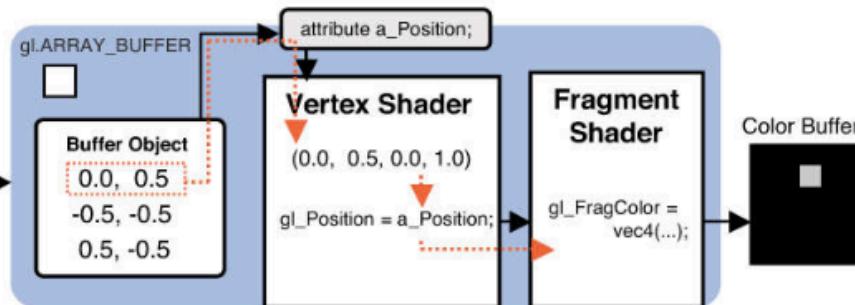
Enable the assignment of a buffer object to the attribute variable specified by *location*.

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

Execution Flow

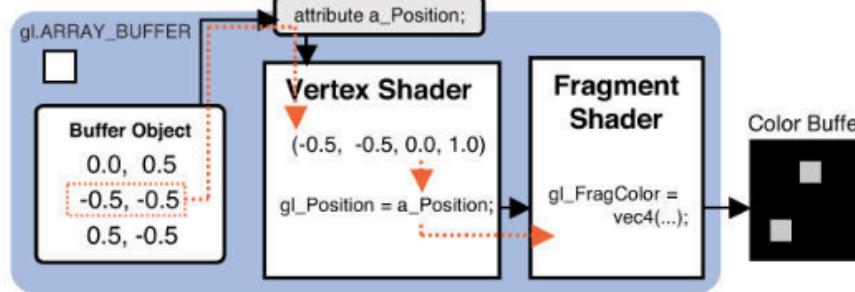
First Execution

```
JavaScript
function main() {
    ...
    var gl=getWebGL...
    ...
    gl.drawArrays( ..., 0, n);
```



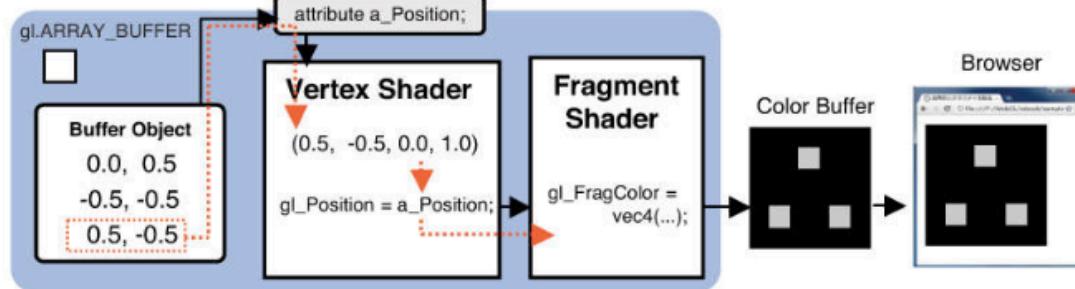
Second Execution

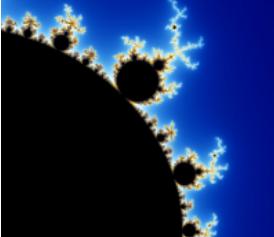
```
JavaScript
function main() {
    ...
    var gl=getWebGL...
    ...
    gl.drawArrays( ..., 0, n);
```



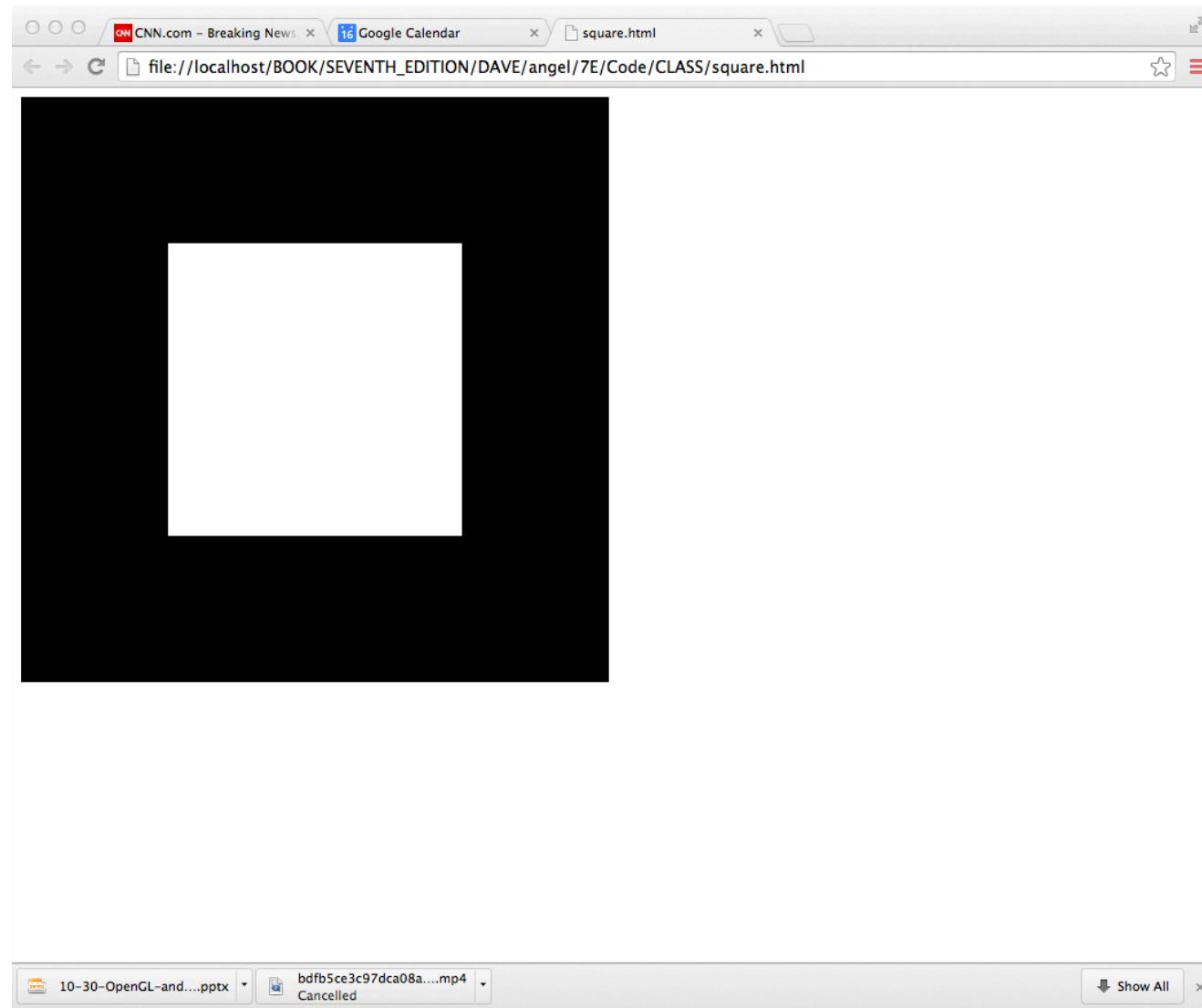
Third Execution

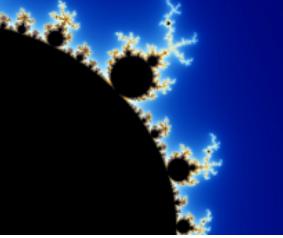
```
JavaScript
function main() {
    ...
    var gl=getWebGL...
    ...
    gl.drawArrays( ..., 0, n);
```





Example : Square Program





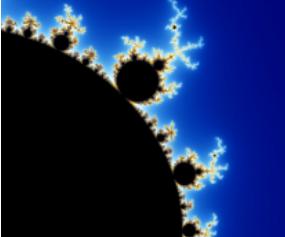
square.js

```
var gl;      // global
var points; // global

function main() {
    var canvas = document.getElementById( "gl-canvas" );

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

    // Define the four vertices  (values in clip coordinates)
    var vertices = [
        vec2( -0.5, -0.5 ),
        vec2( -0.5, 0.5 ),
        vec2( 0.5, 0.5 ),
        vec2( 0.5, -0.5 )
    ];
```



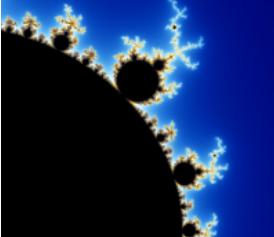
square.js (cont)

```
// 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 );

// Load the data into the GPU
var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

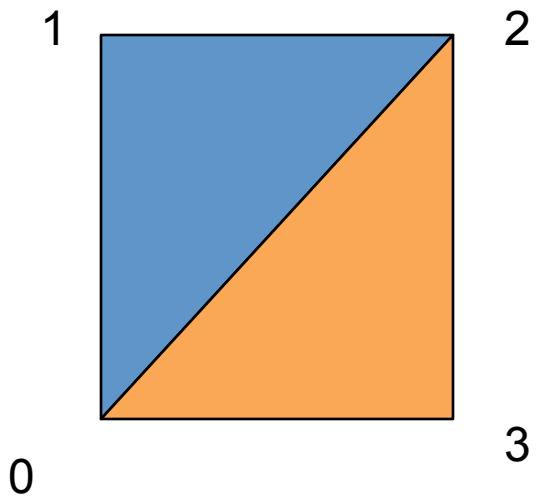
// Associate our shader variables with our data buffer
// flatten() to convert JS array to an array of float32's
var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```

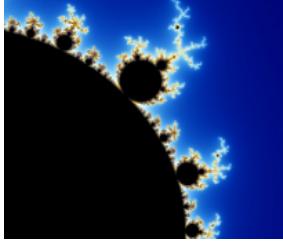


square.js (cont)

```
    render();
};

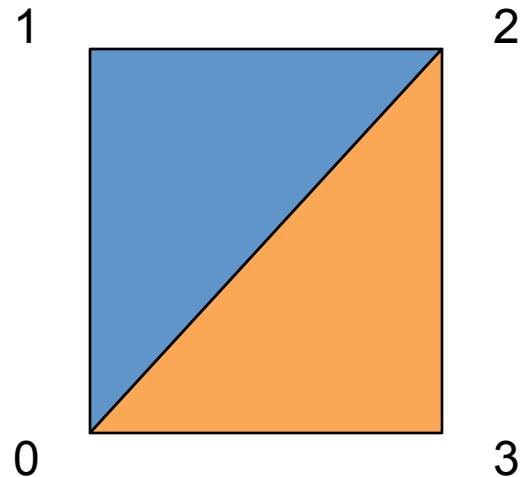
function render() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );
}
```





Triangles, Fans or Strips

```
gl.drawArrays( gl.TRIANGLES, 0, 6 ); // 0, 1, 2, 0, 2, 3  
gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 ); // 0, 1, 2, 3
```



```
gl.drawArrays( gl.TRIANGLE_STRIP, 0, 4 );  
// 0, 1, 2, 3
```

