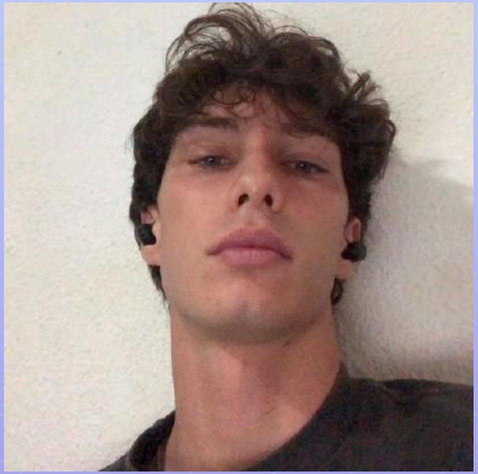
WebGL™



Adrián Grassin Luis
adrian.grassin.31@ull.edu.es

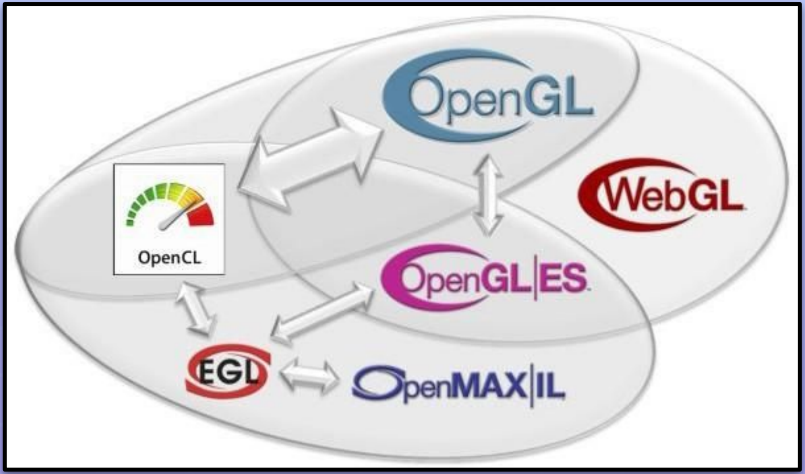
Gian Luis Bolivar Diana
gian.diana.28@ull.edu.es

CONTENT

- What 's WebGL?
- Motivación
- Shaders
- Getting started with WebGL
- Working with WebGL

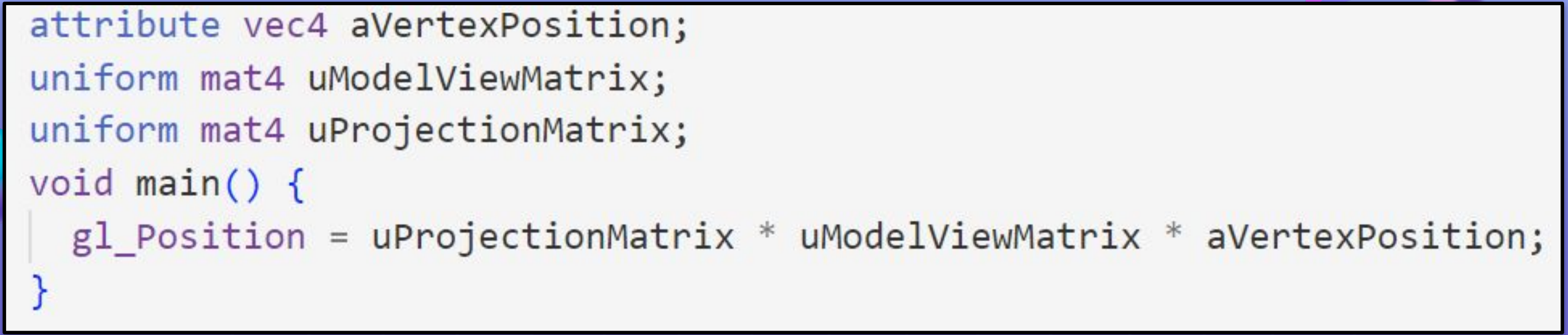
What's WebGL?

Compatible with firefox, safari edge, chrome.



Standard specification that defines an API implemented in JavaScript for rendering 3D graphics in the web.

Optimizing CPU



```
attribute vec4 aVertexPosition;
uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;
void main() {
  gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
}
```

Uses the GPU to perform calculations and parallel processing, this allows a **fast** and more **efficient rendering** of complex 3d scenes.

WebGL programs are written in JavaScript and GLSL.

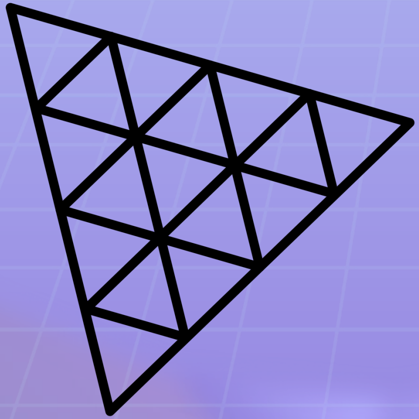
Libraries

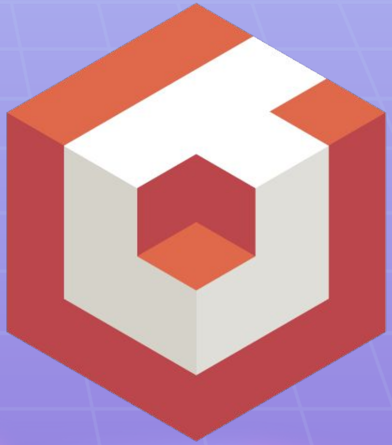
Pros

- Extra functionalities.
- Abstraction.
- Development speed

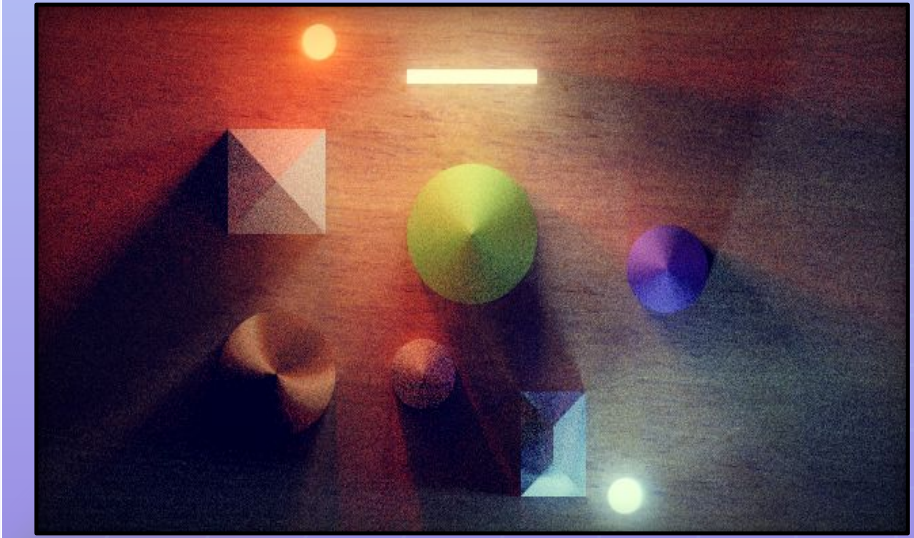
Cons

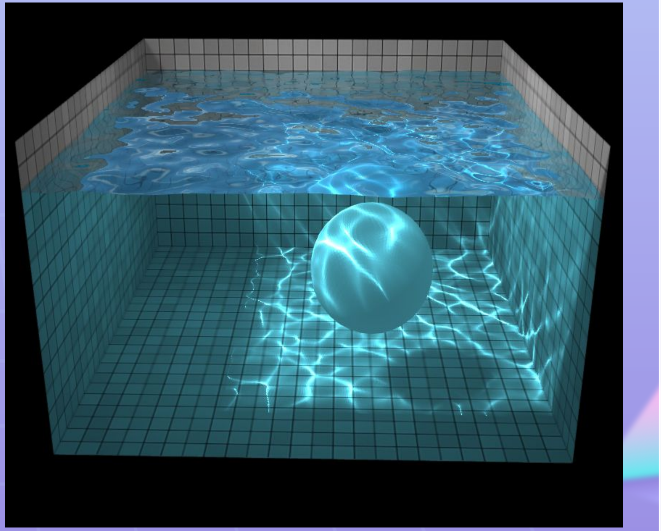
- Limited customization.
- Performance





WebGL Applications

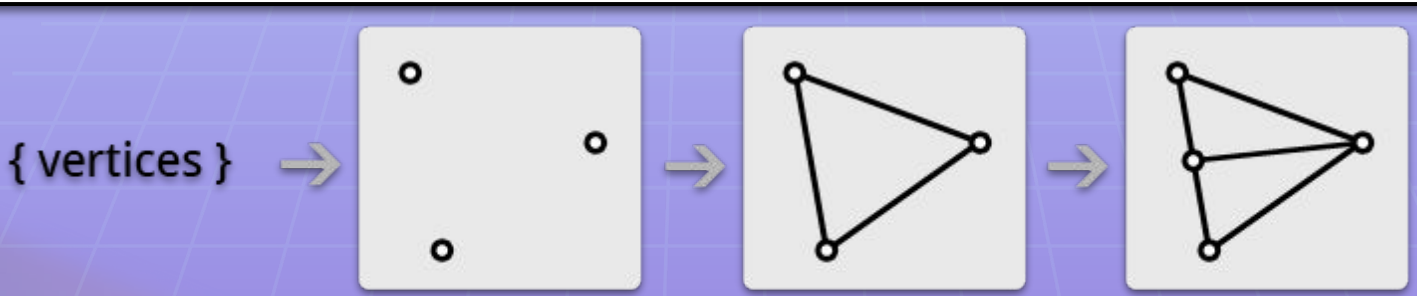




Shaders

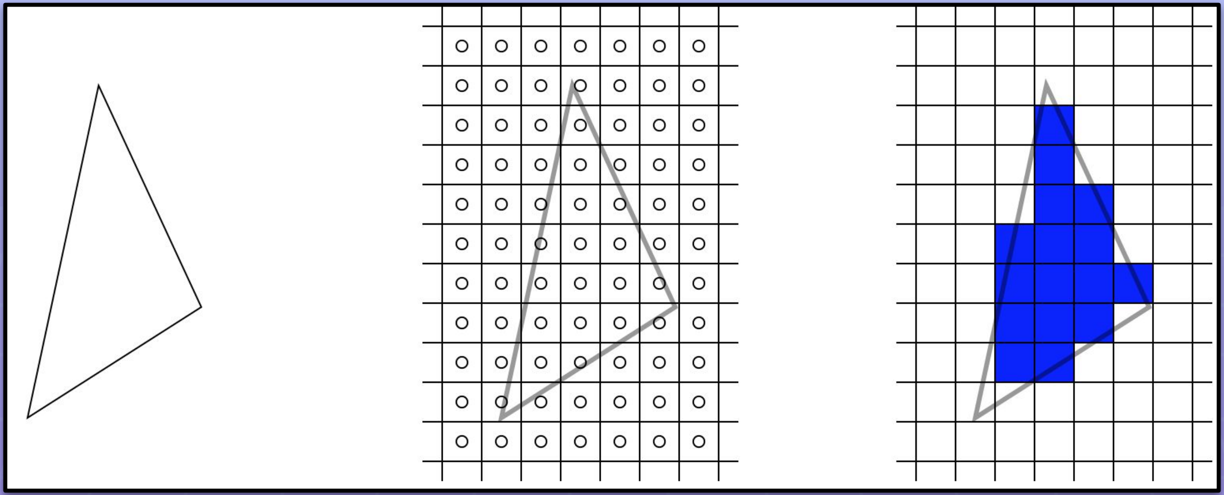
Vertex

Run for every vertex on the figure. Transform the vertex input from the original coordinate system into the clip coordinate system.



Vertex Processing

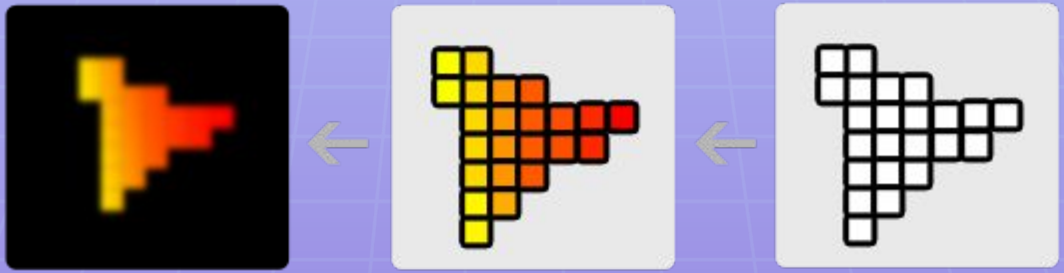
Rasterization process



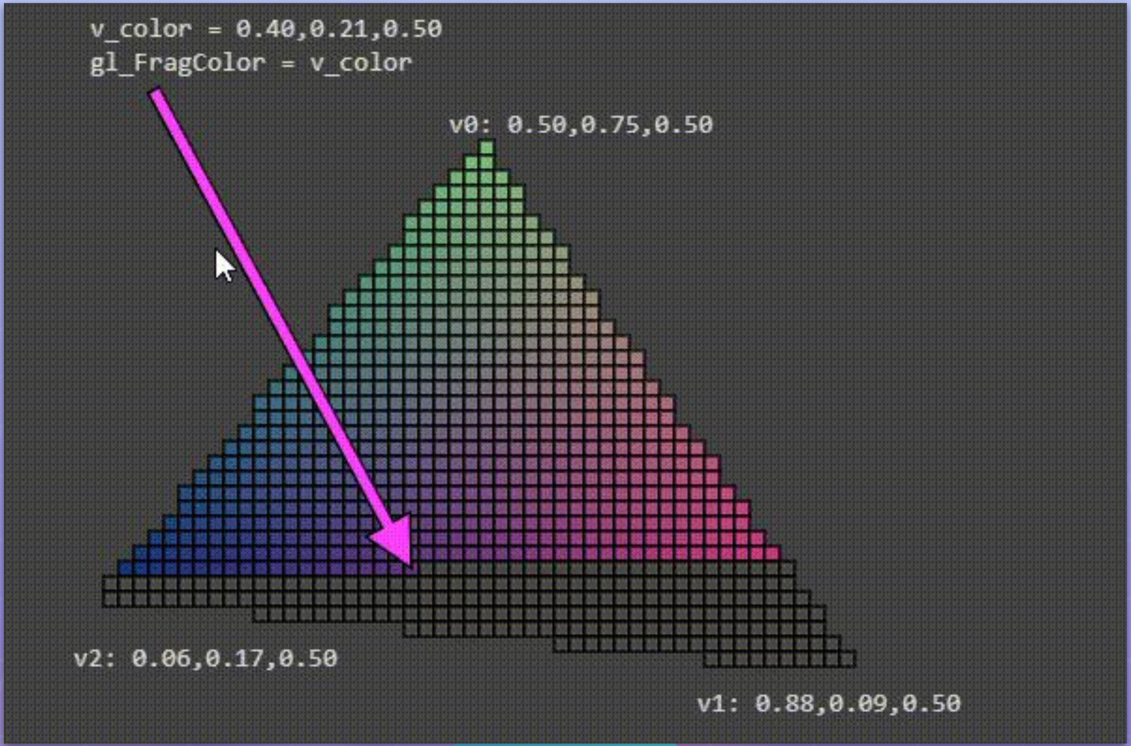
Shaders

Fragment

Is called once for every pixel on shape to be drawn. Determinantes the color and lighting to apply to that texel.



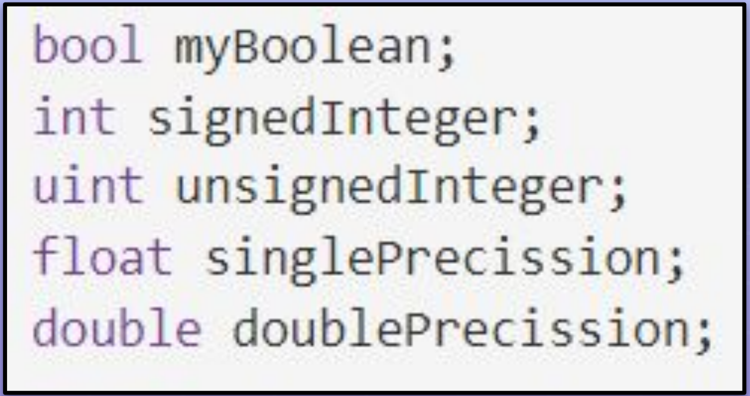
Shaders



v_color = 0.40,0.21,0.50
gl_FragColor = v_color
v0: 0.50,0.75,0.50
v2: 0.06,0.17,0.50
v1: 0.88,0.09,0.50

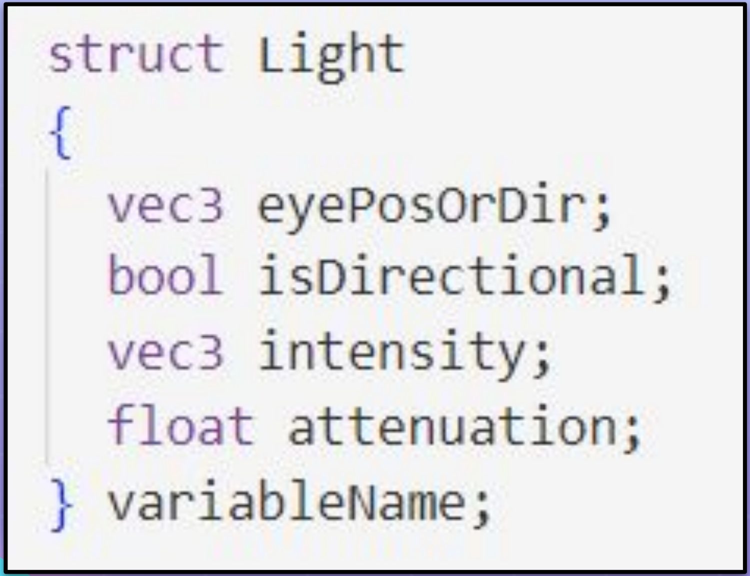
Data types

Scalars



```
bool myBoolean;
int signedInteger;
uint unsignedInteger;
float singlePrecission;
double doublePrecission;
```

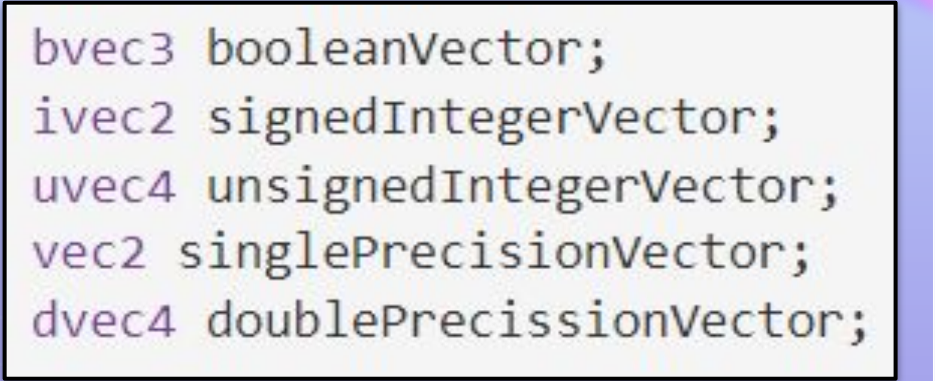
Struct



```
struct Light
{
    vec3 eyePosOrDir;
    bool isDirectional;
    vec3 intensity;
    float attenuation;
} variableName;
```

Khronos Docs

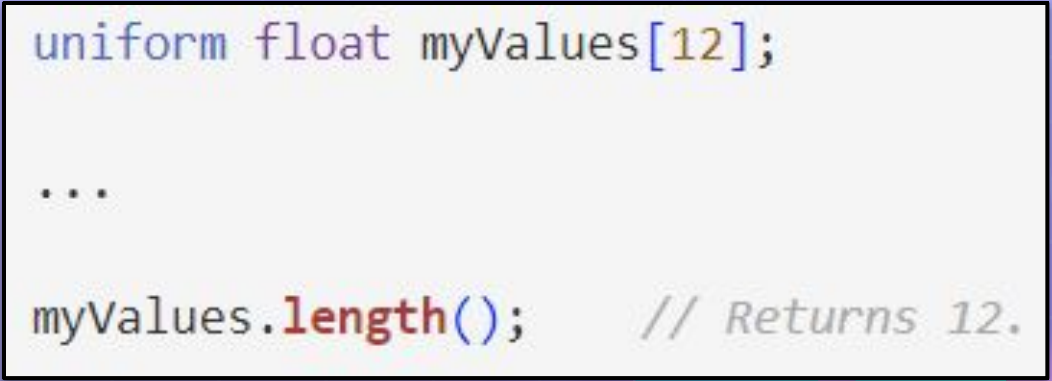
Data types (II)



```
bvec3 booleanVector;
ivec2 signedIntegerVector;
uvec4 unsignedIntegerVector;
vec2 singlePrecisionVector;
dvec4 doublePrecissionVector;
```

Vector

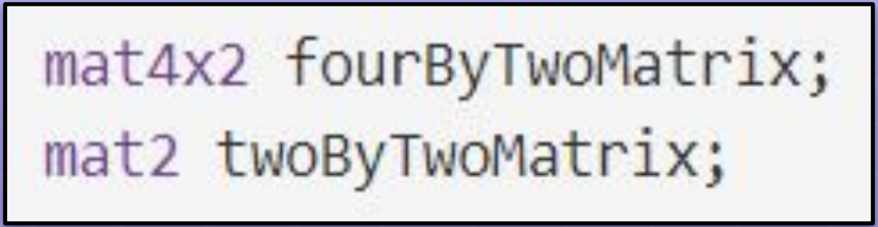
Array



```
uniform float myValues[12];

...

myValues.length();    // Returns 12.
```

Data types (III)

Matrix

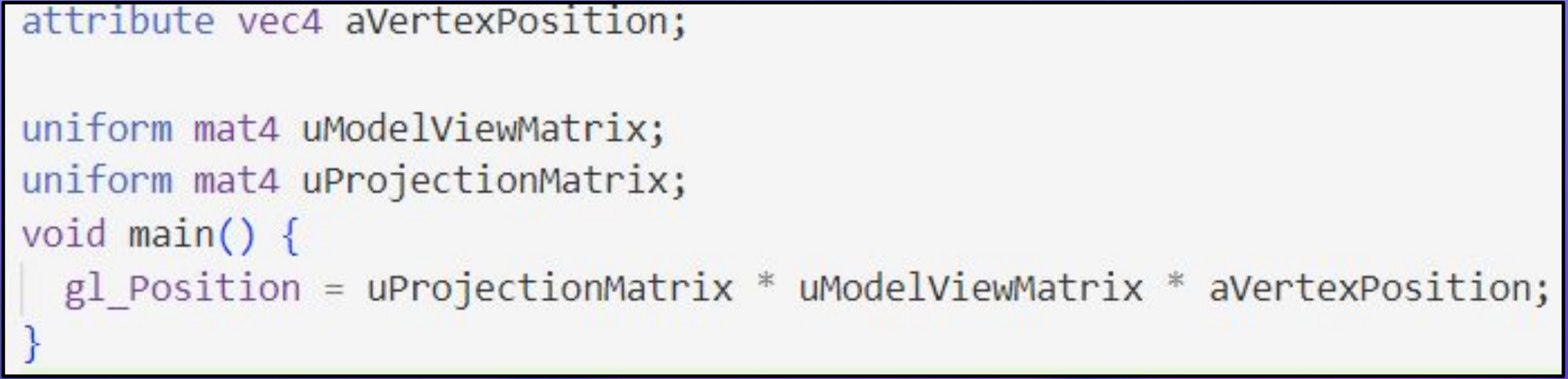


```
mat4x2 fourByTwoMatrix;
mat2 twoByTwoMatrix;
```

Data types (IV)

Attributes

Only available to the **vertex shader** and the JavaScript code. Used to store **color information, texture coordinates**, any kind of information.

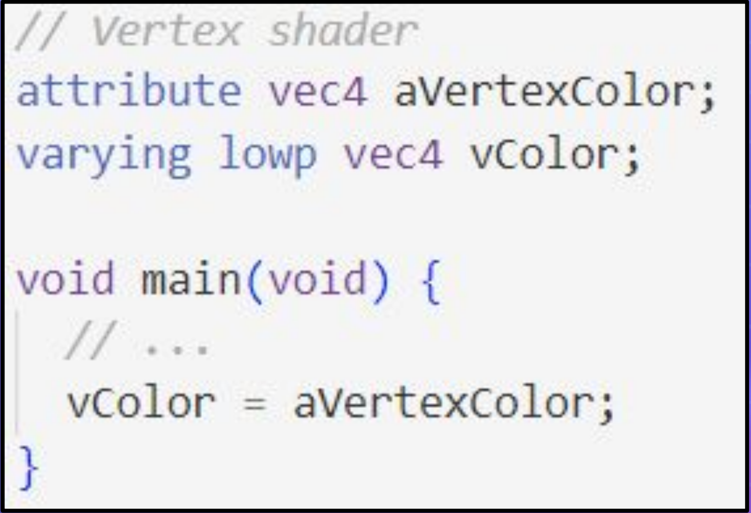


```
attribute vec4 aVertexPosition;


uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;
void main() {
  gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
}
```

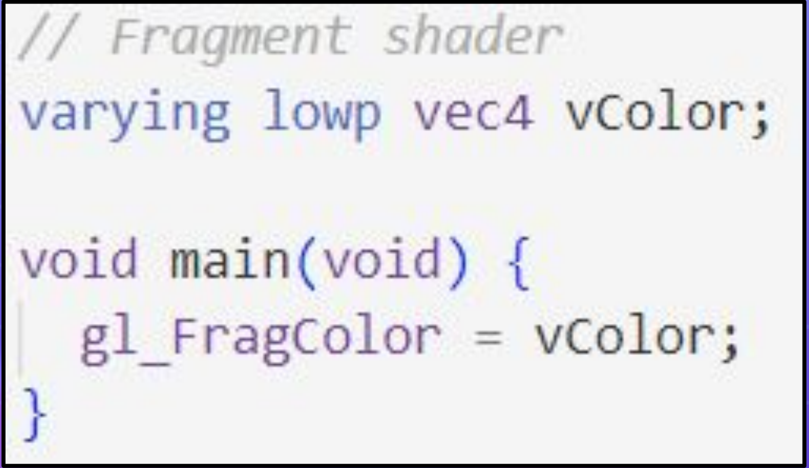
Data types (V)

Varying

Declared by the vertex shader and used to pass data from the vertex shader to the fragment shader.



```
// Vertex shader
attribute vec4 aVertexColor;
varying lowp vec4 vColor;

void main(void) {
  // ...
  vColor = aVertexColor;
}
```

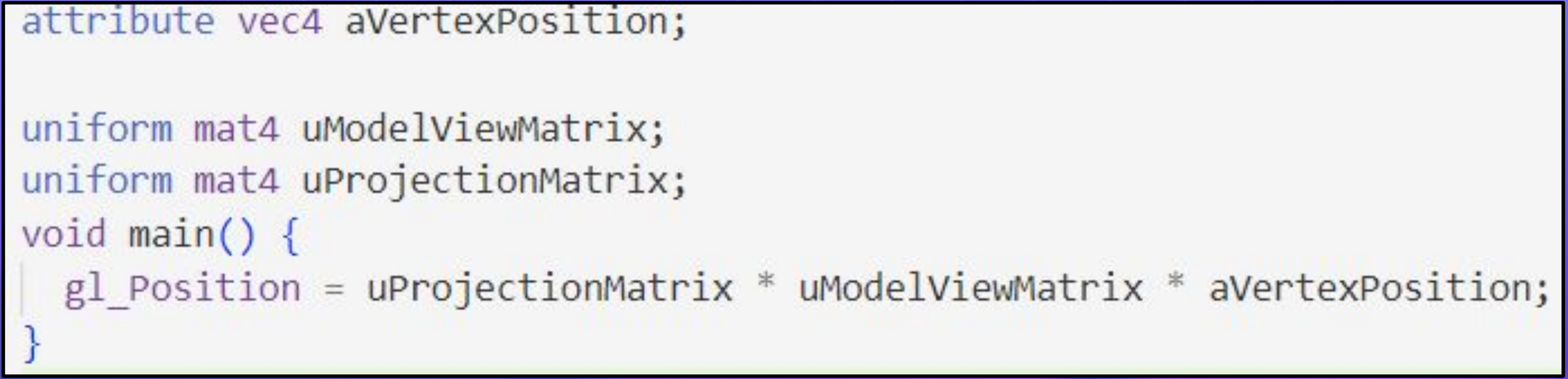


```
// Fragment shader
varying lowp vec4 vColor;

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

Data types (VI)

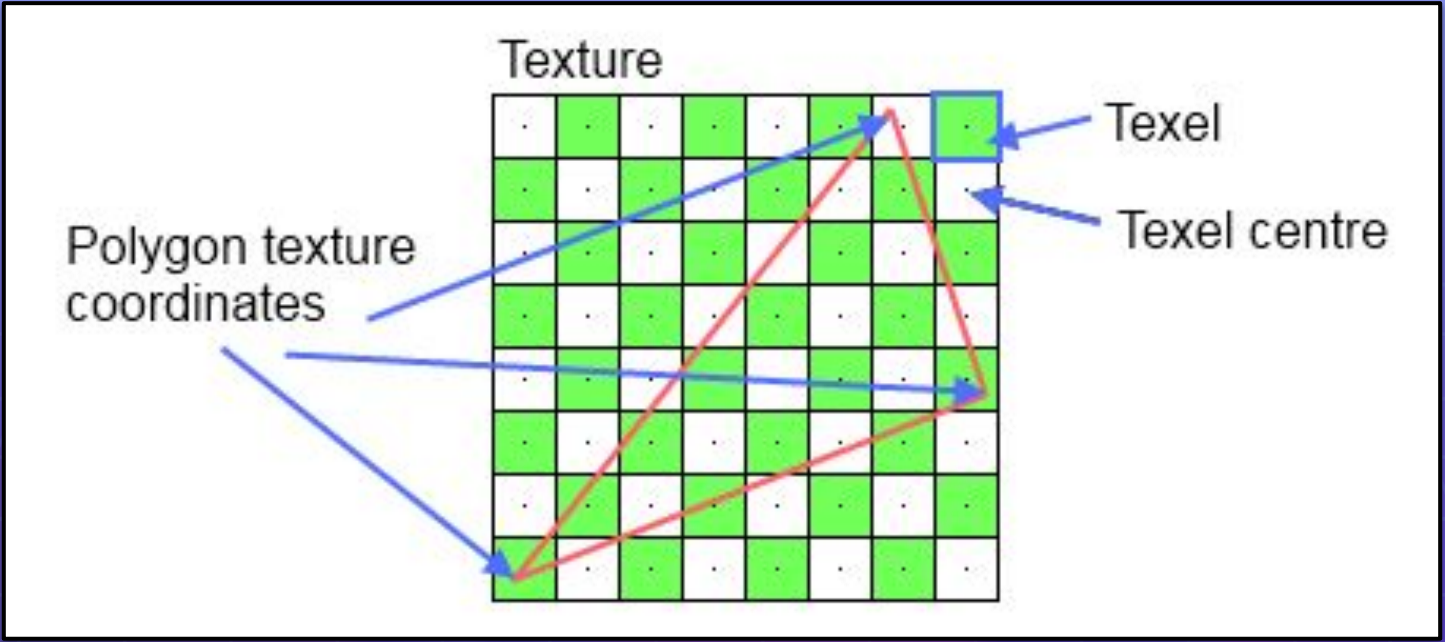
Uniform

Set by the JavaScript code and are available to both the **vertex** and **fragment shaders**. Typically for values that does **not change** over time, like **lighting position** and **perspective details**.

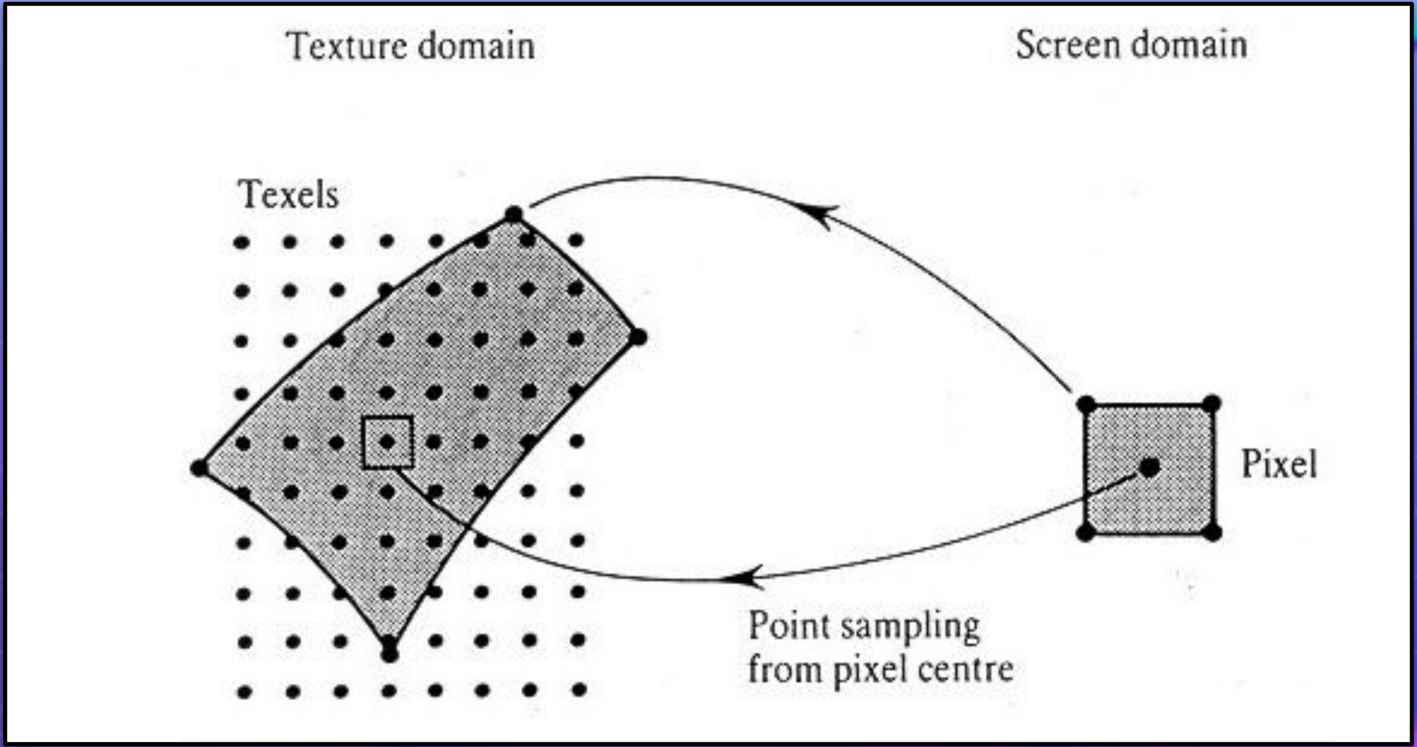


```
attribute vec4 aVertexPosition;


uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;
void main() {
  gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
}
```

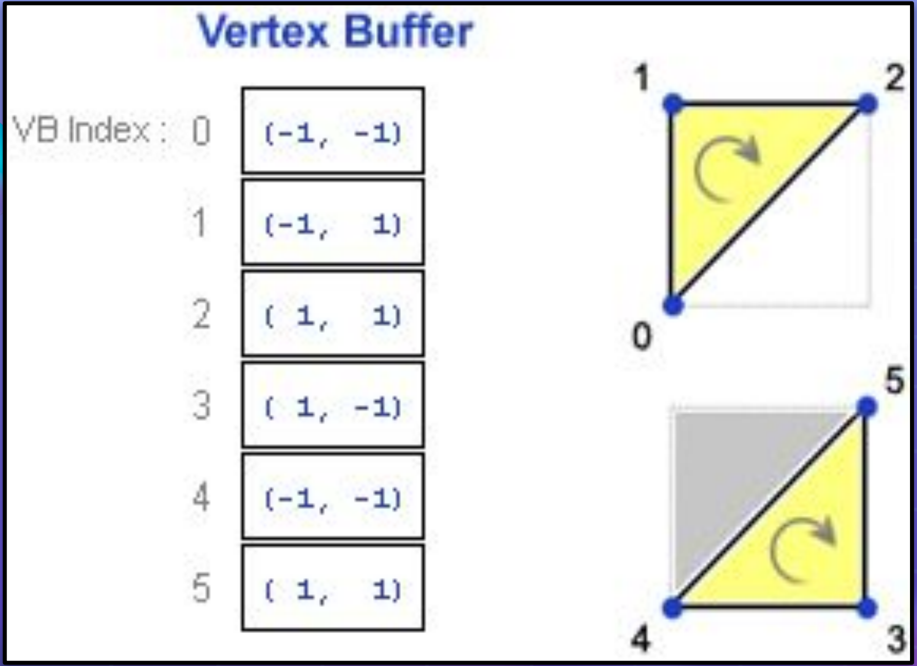
Texture Pixel (Texel)



Texel (II)

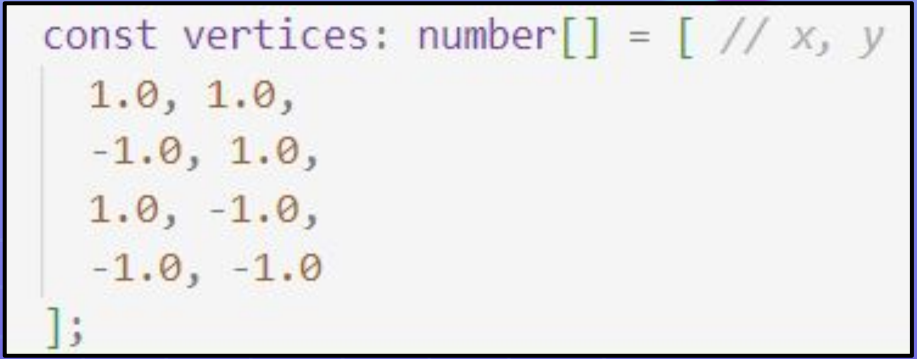

Texture domain
Screen domain
Texels
Pixel
Point sampling
from pixel centre

Vertex Buffer



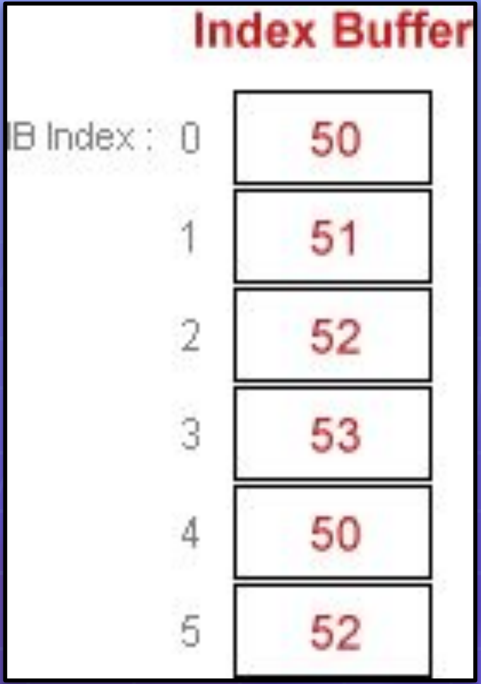
Vertex Buffer

VB Index :	
0	(-1, -1)
1	(-1, 1)
2	(1, 1)
3	(1, -1)
4	(-1, -1)
5	(1, 1)



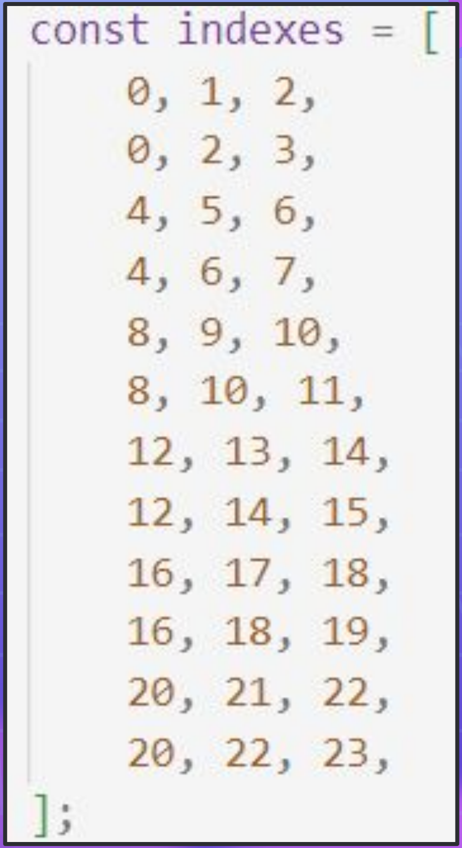
```
const vertices: number[] = [ // x, y
  1.0, 1.0,
  -1.0, 1.0,
  1.0, -1.0,
  -1.0, -1.0
];
```

Index Buffer



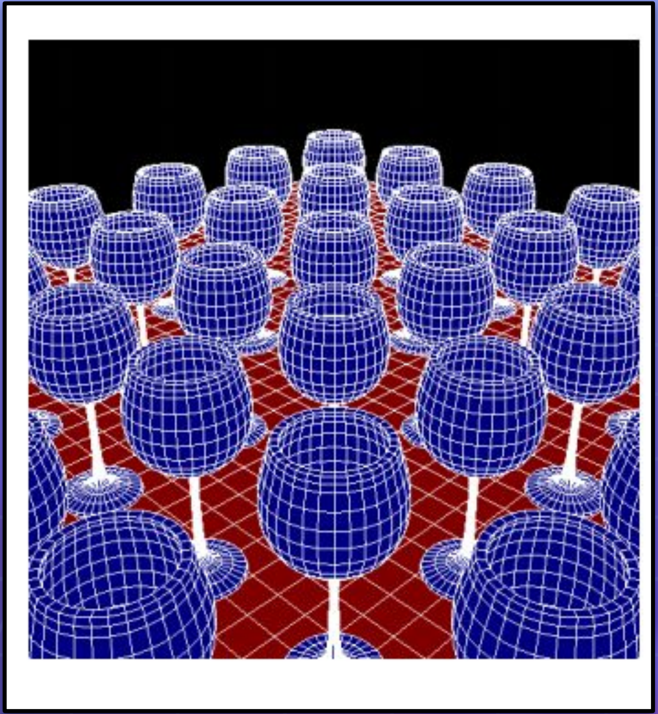
Index Buffer

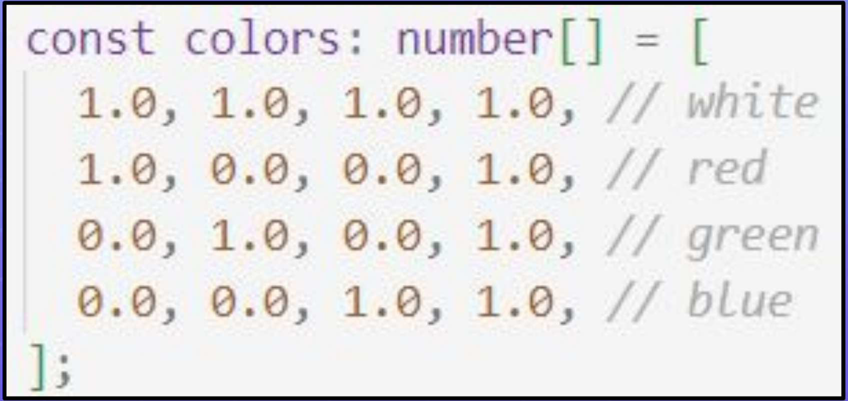
IB Index :	
0	50
1	51
2	52
3	53
4	50
5	52



```
const indexes = [
    0, 1, 2,
    0, 2, 3,
    4, 5, 6,
    4, 6, 7,
    8, 9, 10,
    8, 10, 11,
    12, 13, 14,
    12, 14, 15,
    16, 17, 18,
    16, 18, 19,
    20, 21, 22,
    20, 22, 23,
];
```

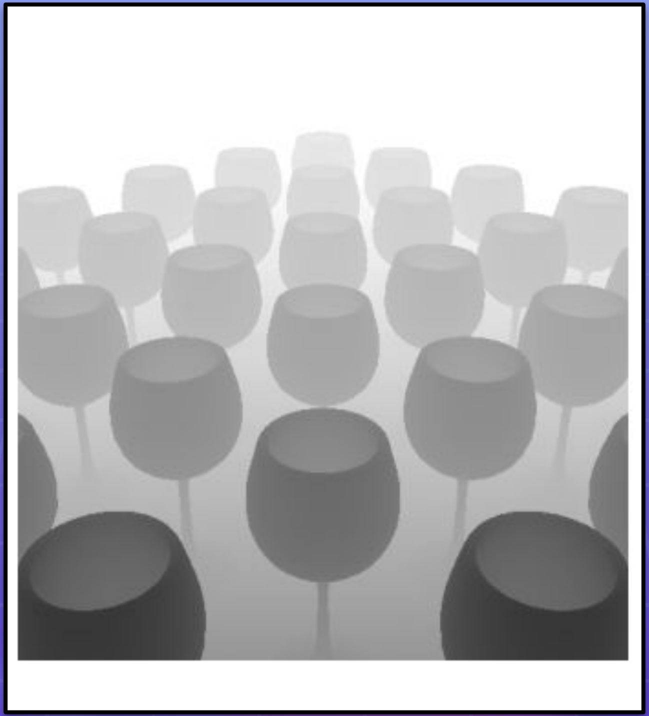
Color Buffer



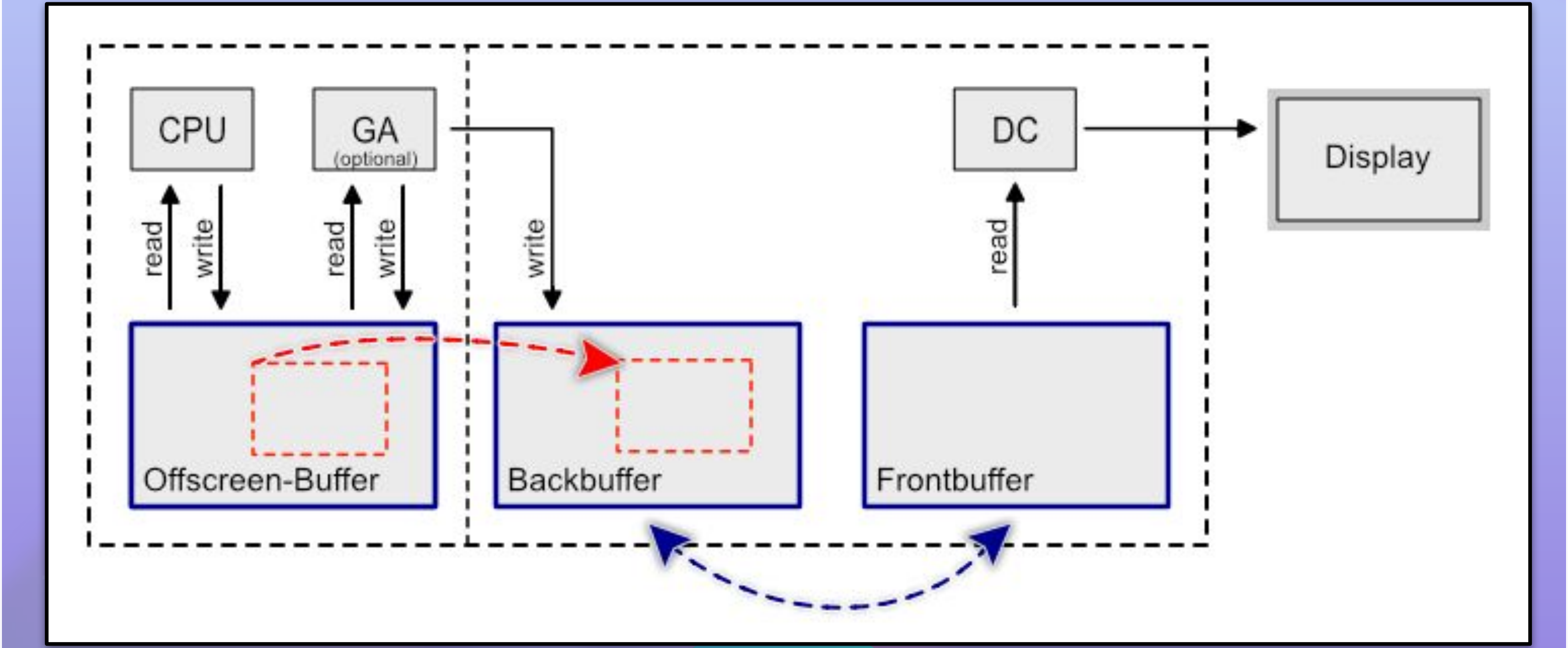


```
const colors: number[] = [
  1.0, 1.0, 1.0, 1.0, // white
  1.0, 0.0, 0.0, 1.0, // red
  0.0, 1.0, 0.0, 1.0, // green
  0.0, 0.0, 1.0, 1.0, // blue
];
```

Depth Buffer



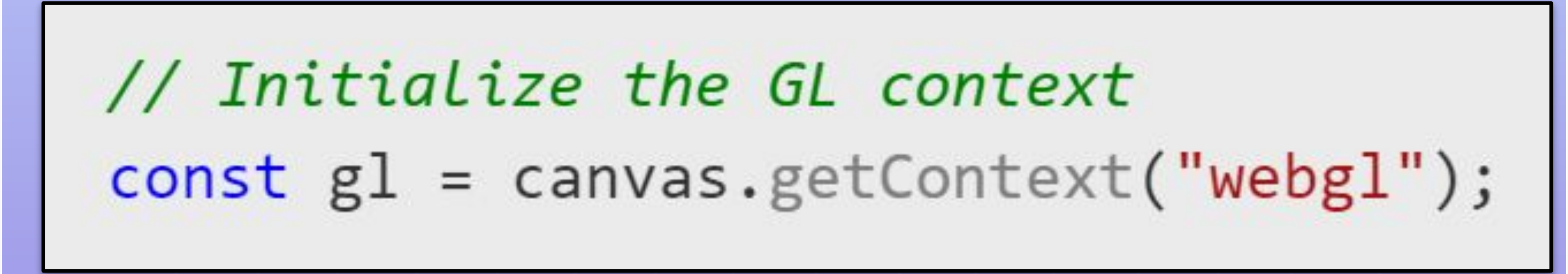
Framebuffer (II)



Getting Started

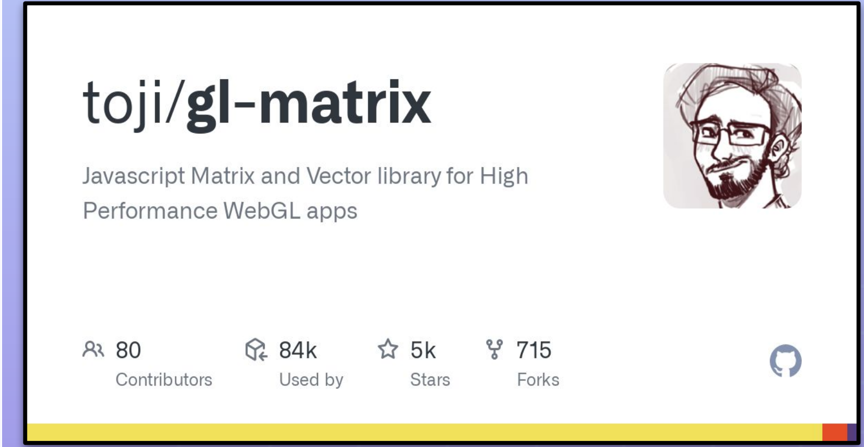
1. Create / get the canvas.
2. Get the webgl context.
3. Initialize the program with the shaders (compiled and linked).
4. Specify the GLSL variables on JS.
5. Initialize the buffers.
6. Draw the scene.

Rendering Context



```
// Initialize the GL context
const gl = canvas.getContext("webgl");
```

GL-Matrix



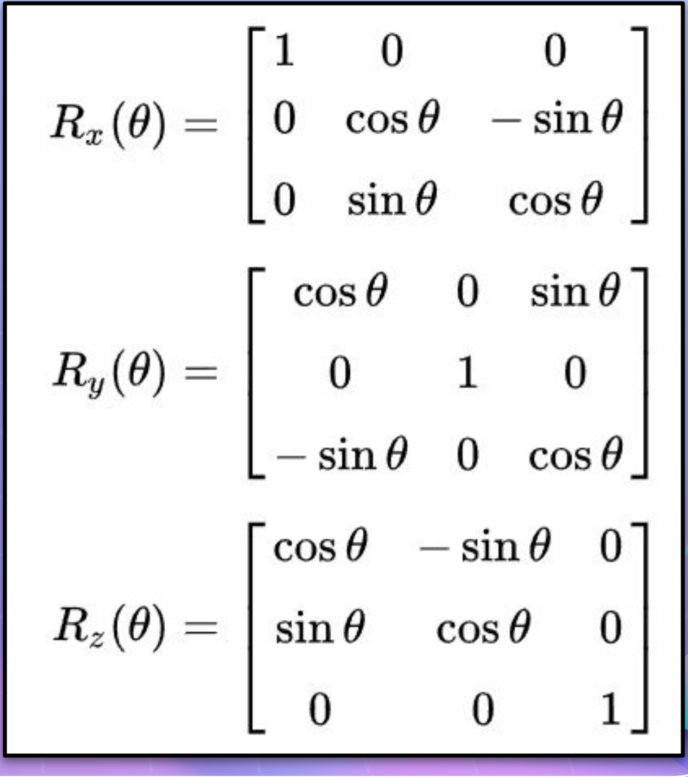
toji/**gl-matrix**

Javascript Matrix and Vector library for High Performance WebGL apps

80	84k	5k	715
Contributors	Used by	Stars	Forks

How to include this library in your project

github.com/gl-matrix



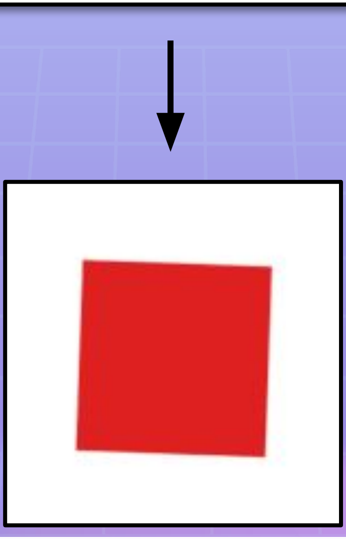
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

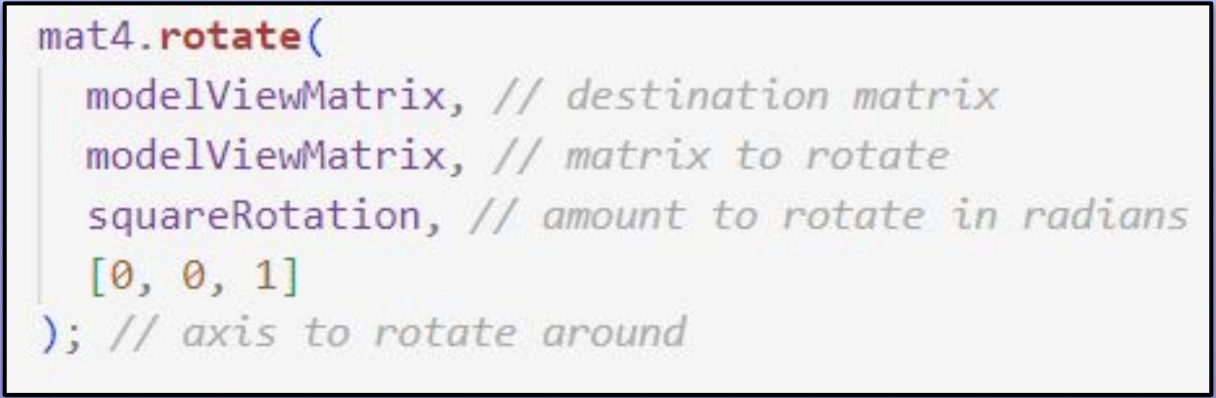
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotations

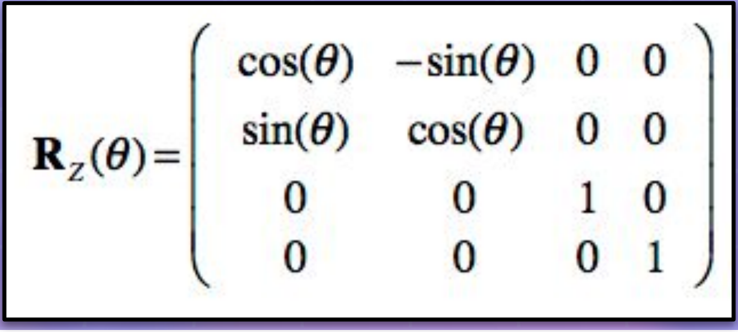
$$R\mathbf{v} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$



Rotations (II)

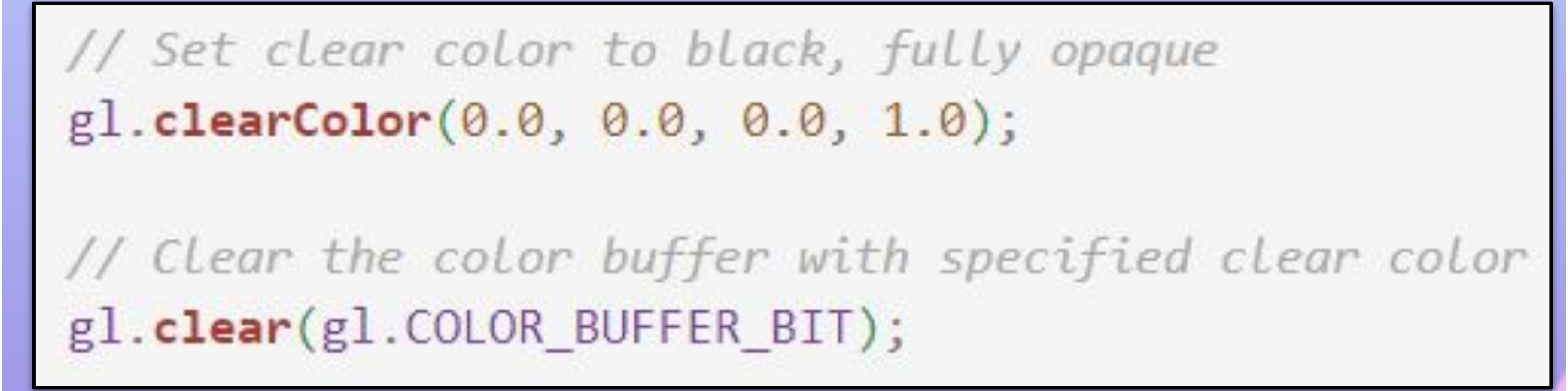


```
mat4.rotate(
  modelViewMatrix, // destination matrix
  modelViewMatrix, // matrix to rotate
  squareRotation, // amount to rotate in radians
  [0, 0, 1]
); // axis to rotate around
```



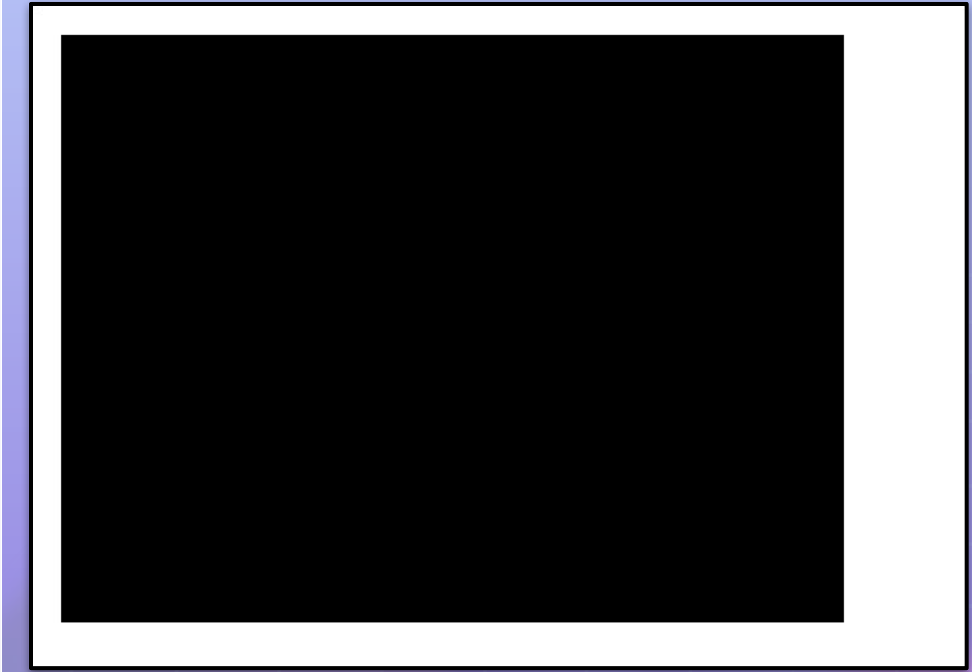
$$\mathbf{R}_Z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

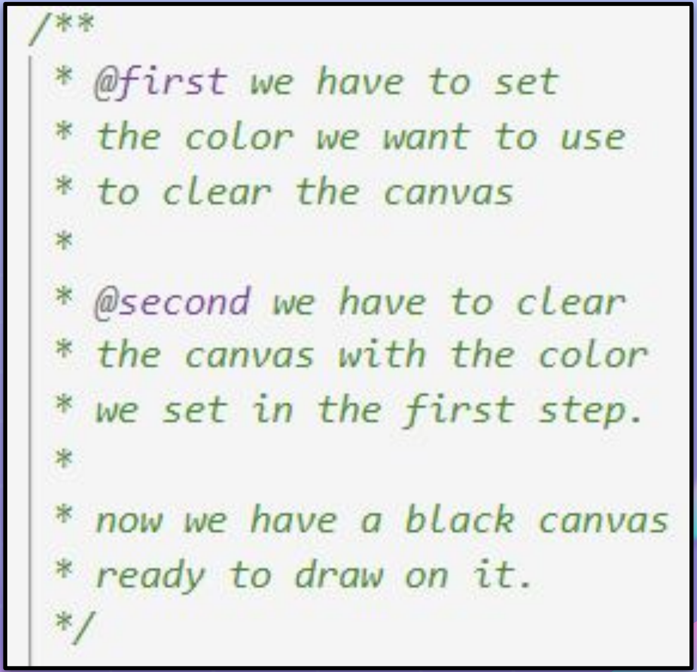
Canvas Setup



```
// Set clear color to black, fully opaque
gl.clearColor(0.0, 0.0, 0.0, 1.0);

// Clear the color buffer with specified clear color
gl.clear(gl.COLOR_BUFFER_BIT);
```

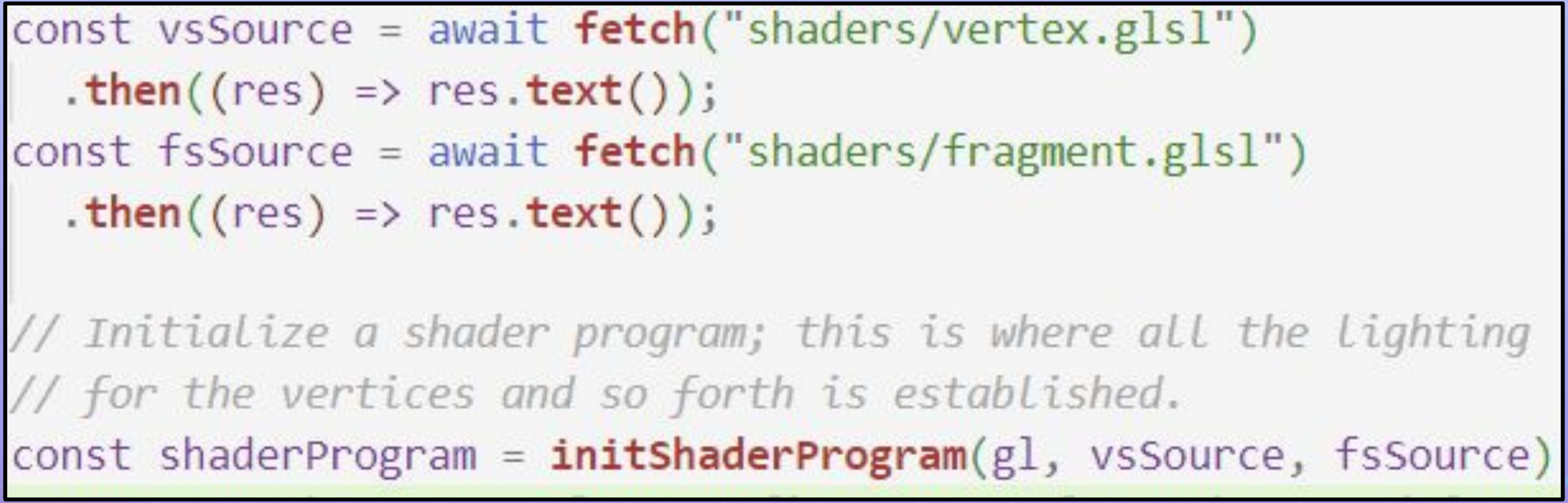
Canvas Setup (II)





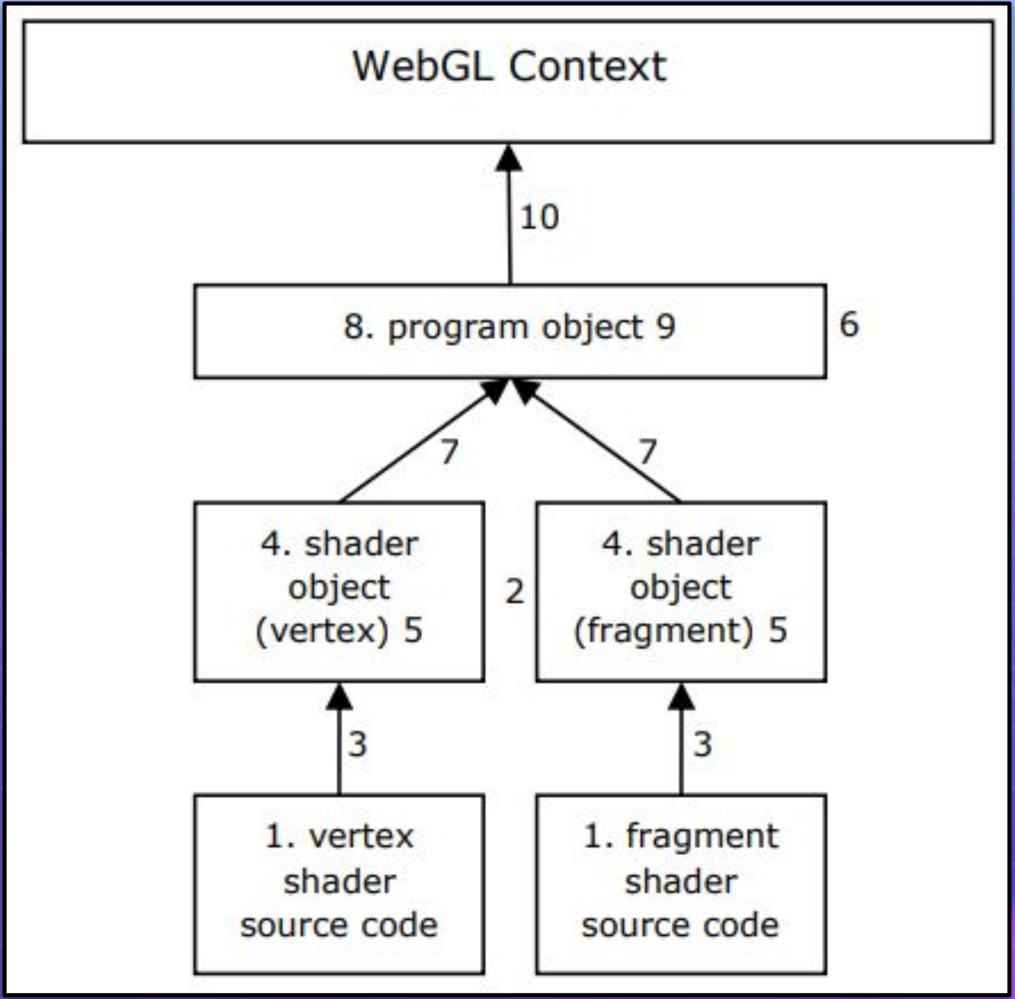
```
/**
 * @first we have to set
 * the color we want to use
 * to clear the canvas
 *
 * @second we have to clear
 * the canvas with the color
 * we set in the first step.
 *
 * now we have a black canvas
 * ready to draw on it.
 */
```

Shader Initialization

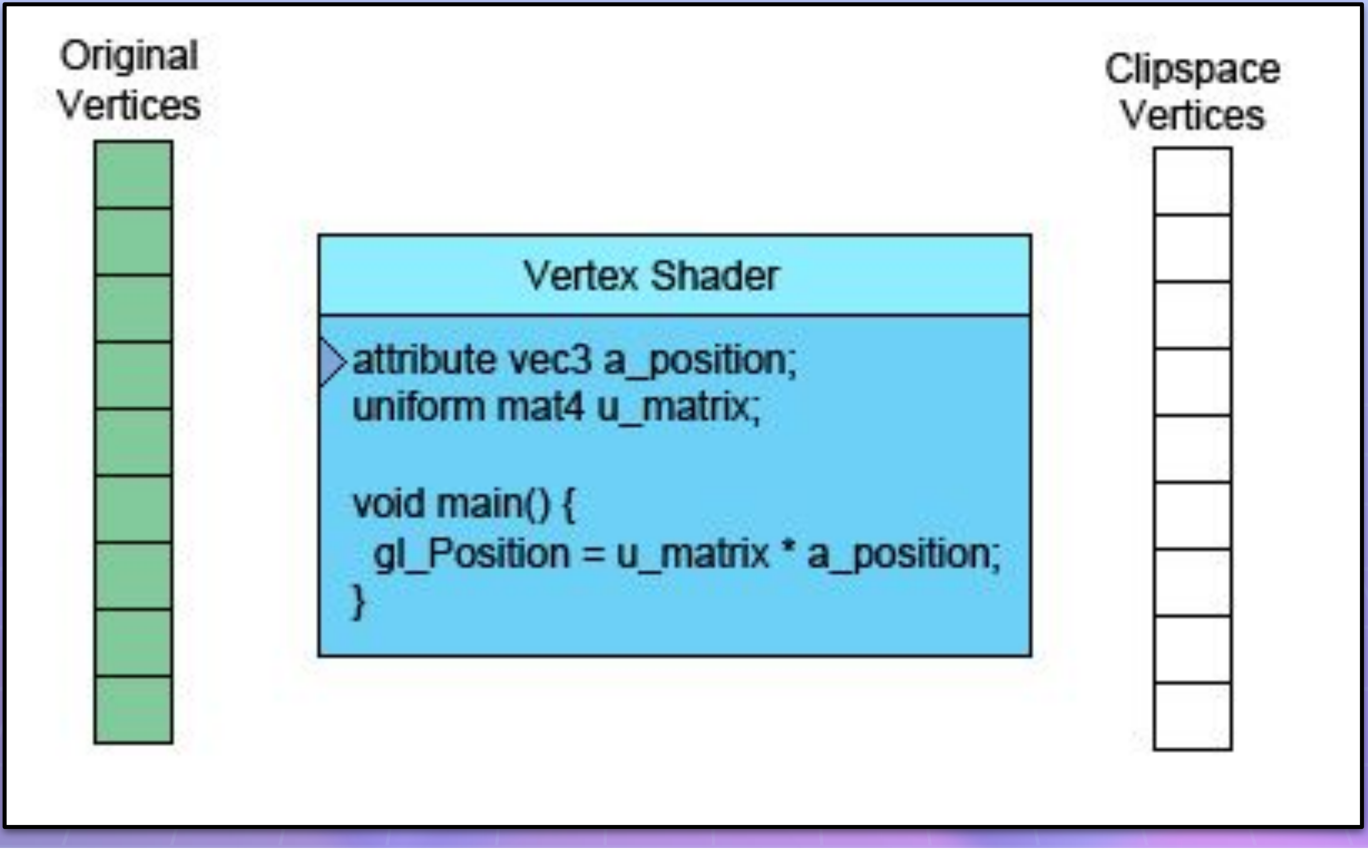


```
const vsSource = await fetch("shaders/vertex.glsl")
  .then((res) => res.text());
const fsSource = await fetch("shaders/fragment.glsl")
  .then((res) => res.text());

// Initialize a shader program; this is where all the lighting
// for the vertices and so forth is established.
const shaderProgram = initShaderProgram(gl, vsSource, fsSource)
```

Shader Initialization（II）


WebGL Context
10
8. program object 9
6
7
7
4. shader object (vertex) 5
2
4. shader object (fragment) 5
3
3
1. vertex shader source code
1. fragment shader source code

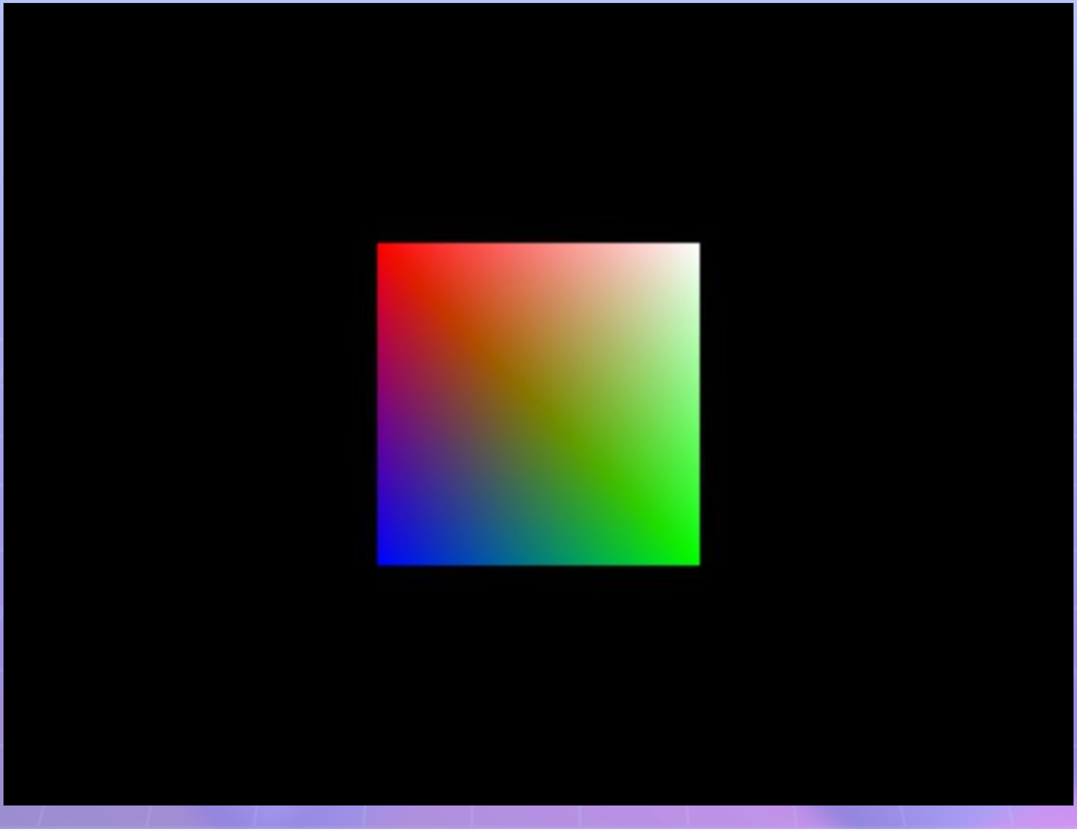
Shader Initialization（III）



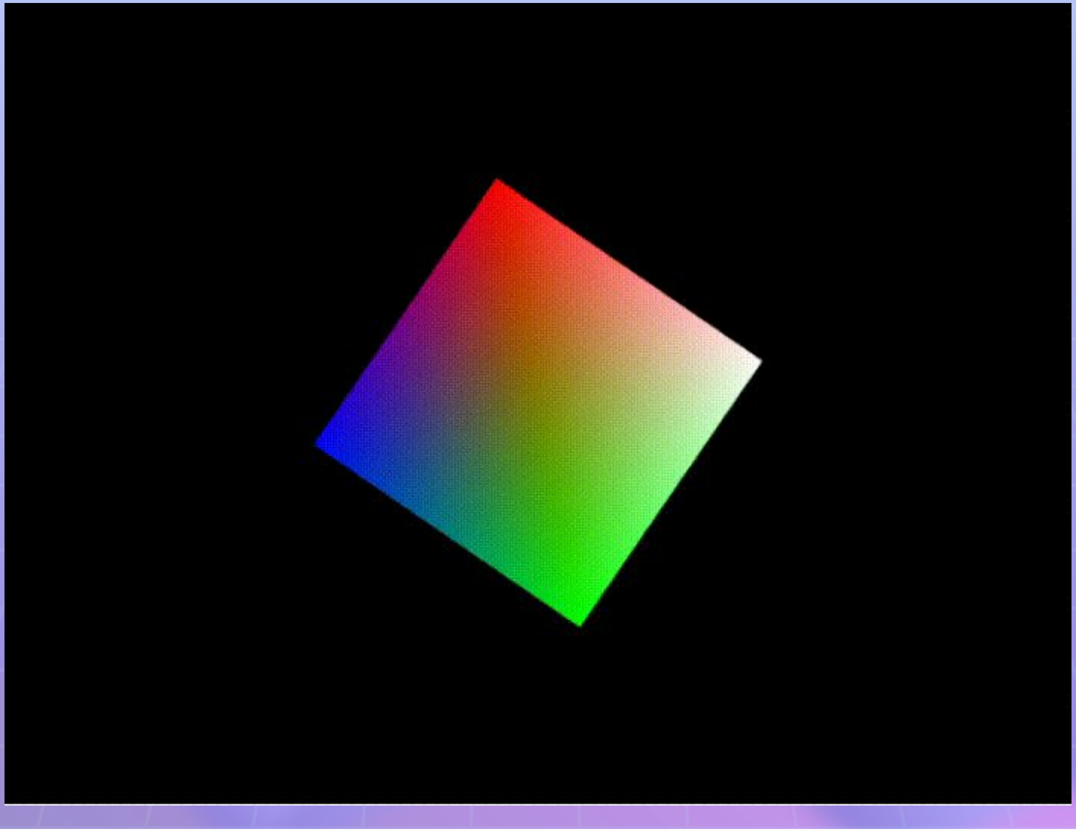
```
attribute vec3 a_position;
uniform mat4 u_matrix;

void main() {
  gl_Position = u_matrix * a_position;
}
```

2D Shapes

Coloring



Animation



Bibliography

- How to include shaders as external files
- MDN Tutorial
- Three.js
- WebGL Concept
- WebGL Websites
- GitHub Repository
- GLMatrix