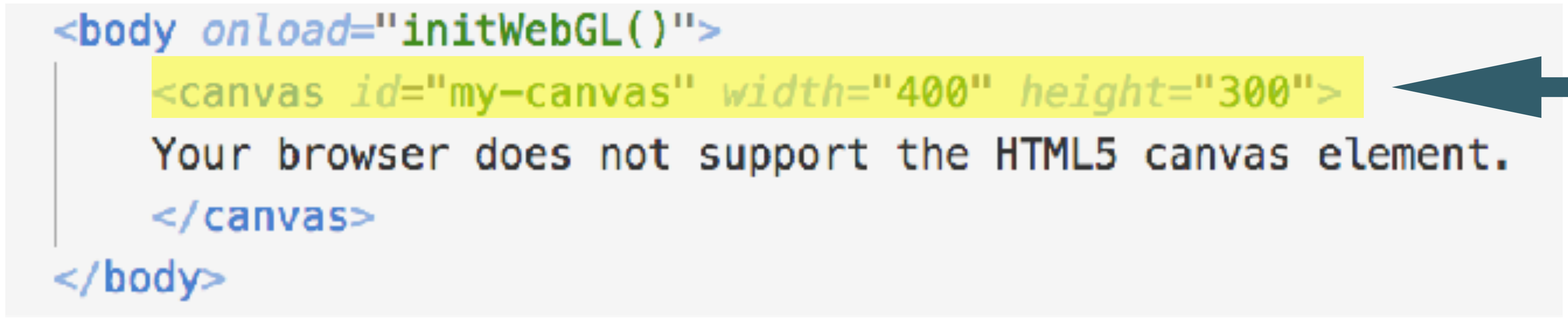
BCA611 Video Oyunları için 3B Grafik

WebGL - Shader Programming

Zümra Kavafoğlu

Canvas element

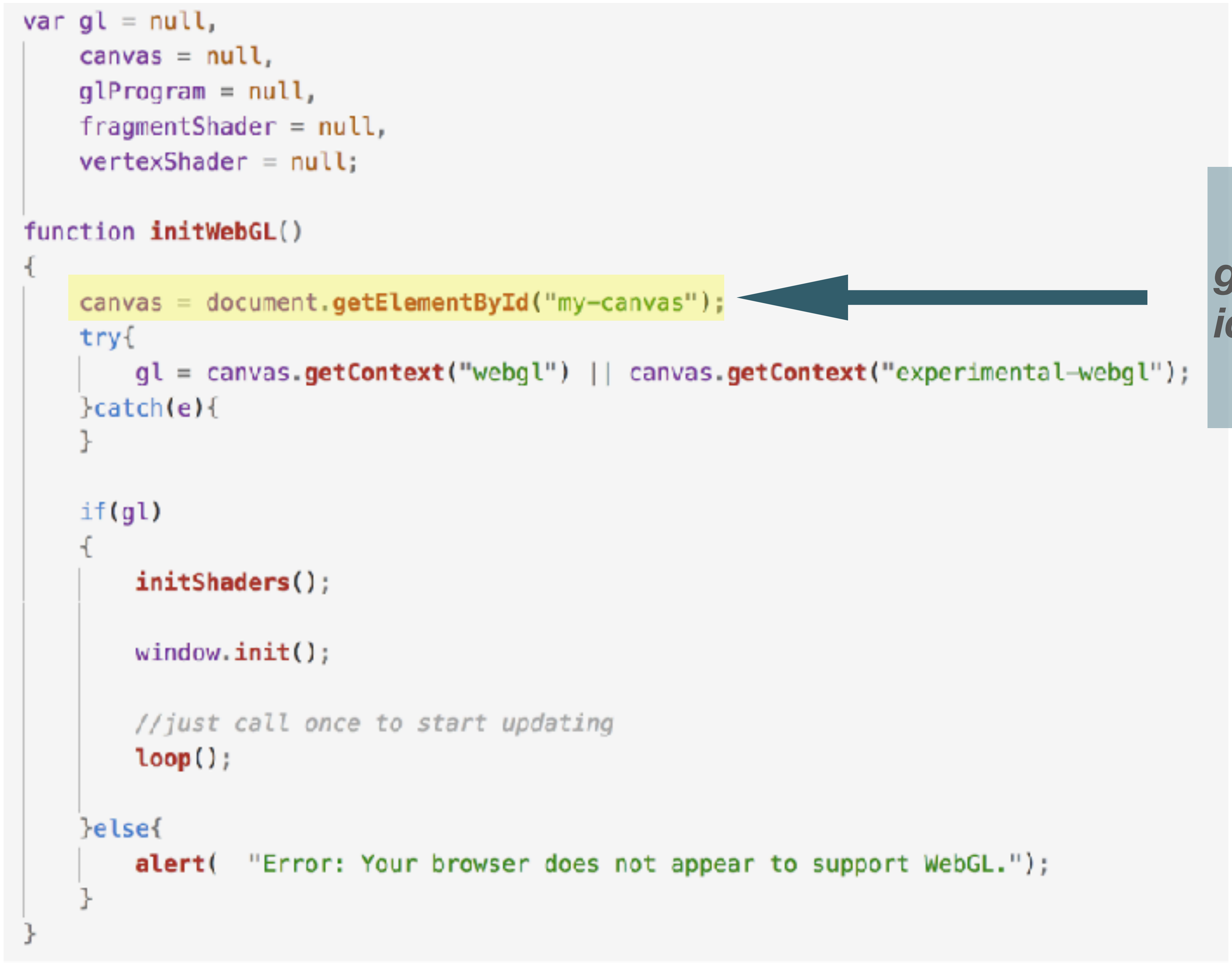
The HTML <canvas> element is used to draw graphics on a web page.



```
<body onload="initWebGL()">
    <canvas id="my-canvas" width="400" height="300">
    Your browser does not support the HTML5 canvas element.
    </canvas>
</body>
```

Create canvas with id, width and height

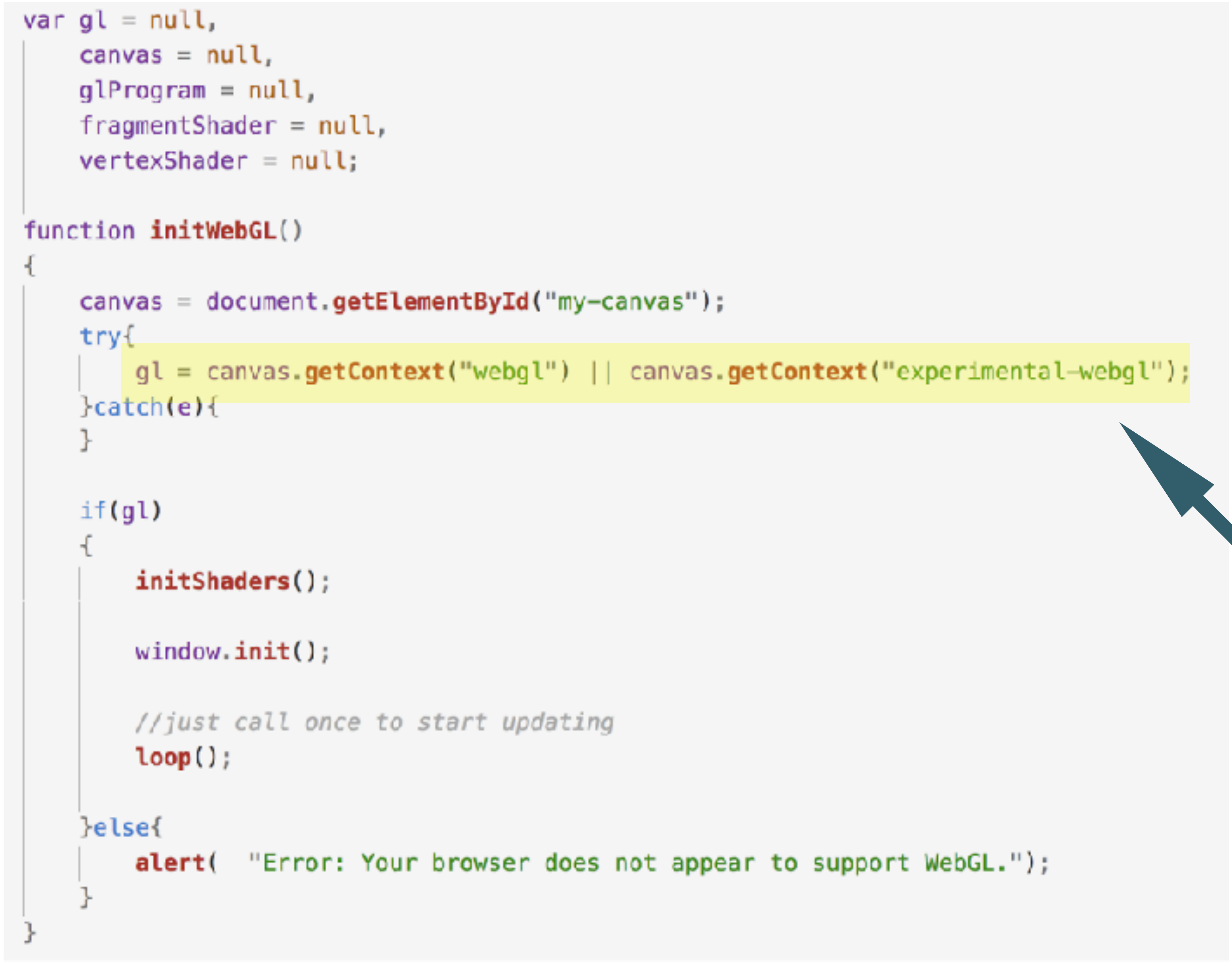
initWebGL()



```
var gl = null,
    canvas = null,
    glProgram = null,
    fragmentShader = null,
    vertexShader = null;

function initWebGL()
{
    canvas = document.getElementById("my-canvas");
    try{
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }catch(e){
    }

    if(gl)
    {
        initShaders();

        window.init();

        //just call once to start updating
        loop();

    }else{
        alert(  "Error: Your browser does not appear to support WebGL.");
    }
}
```

get the canvas with id “my-canvas”

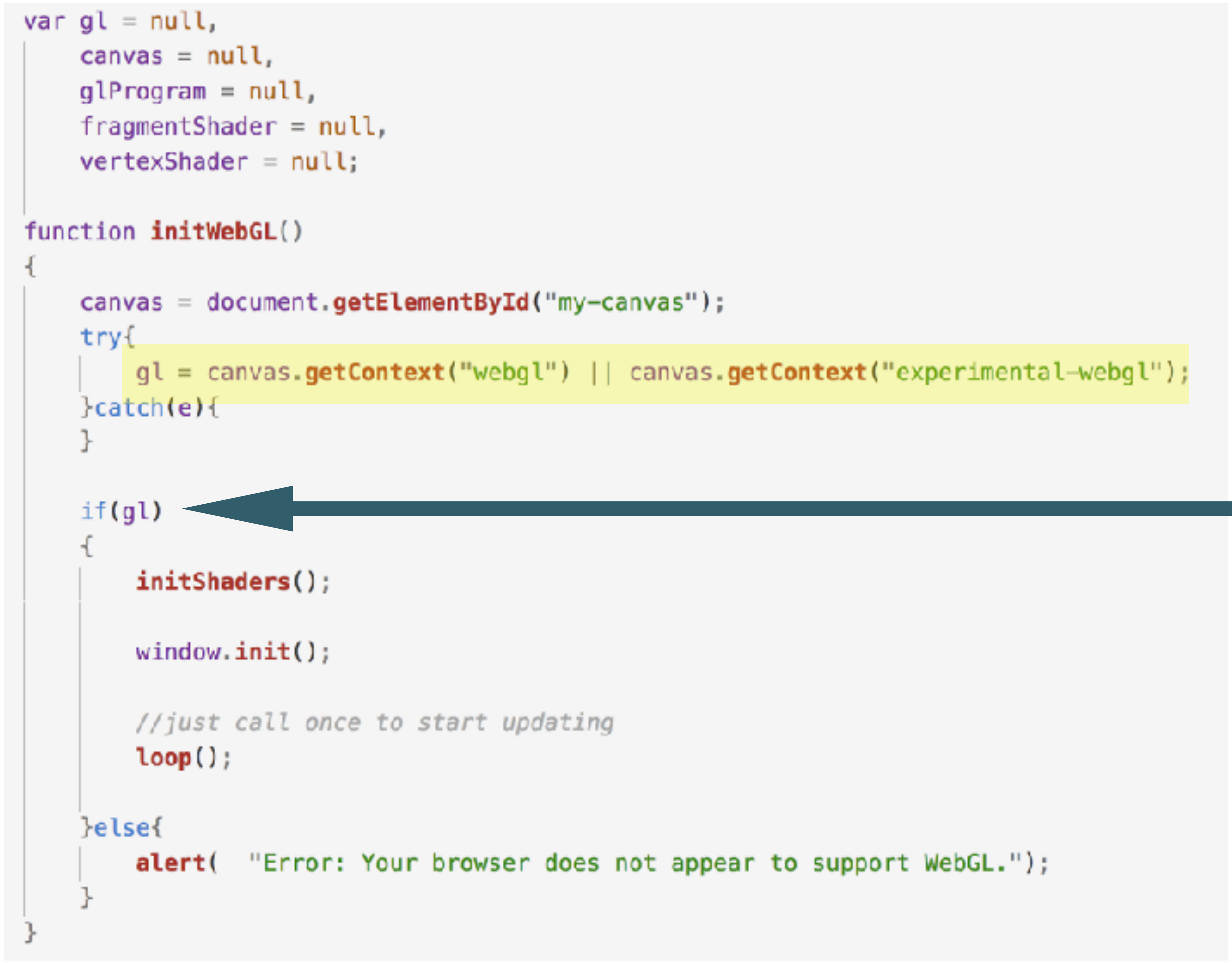
initWebGL()



```
var gl = null,
    canvas = null,
    glProgram = null,
    fragmentShader = null,
    vertexShader = null;

function initWebGL()
{
    canvas = document.getElementById("my-canvas");
    try{
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }catch(e){
    }

    if(gl)
    {
        initShaders();

        window.init();

        //just call once to start updating
        loop();

    }else{
        alert(  "Error: Your browser does not appear to support WebGL.");
    }
}
```

get webGL context if it's supported by the browser

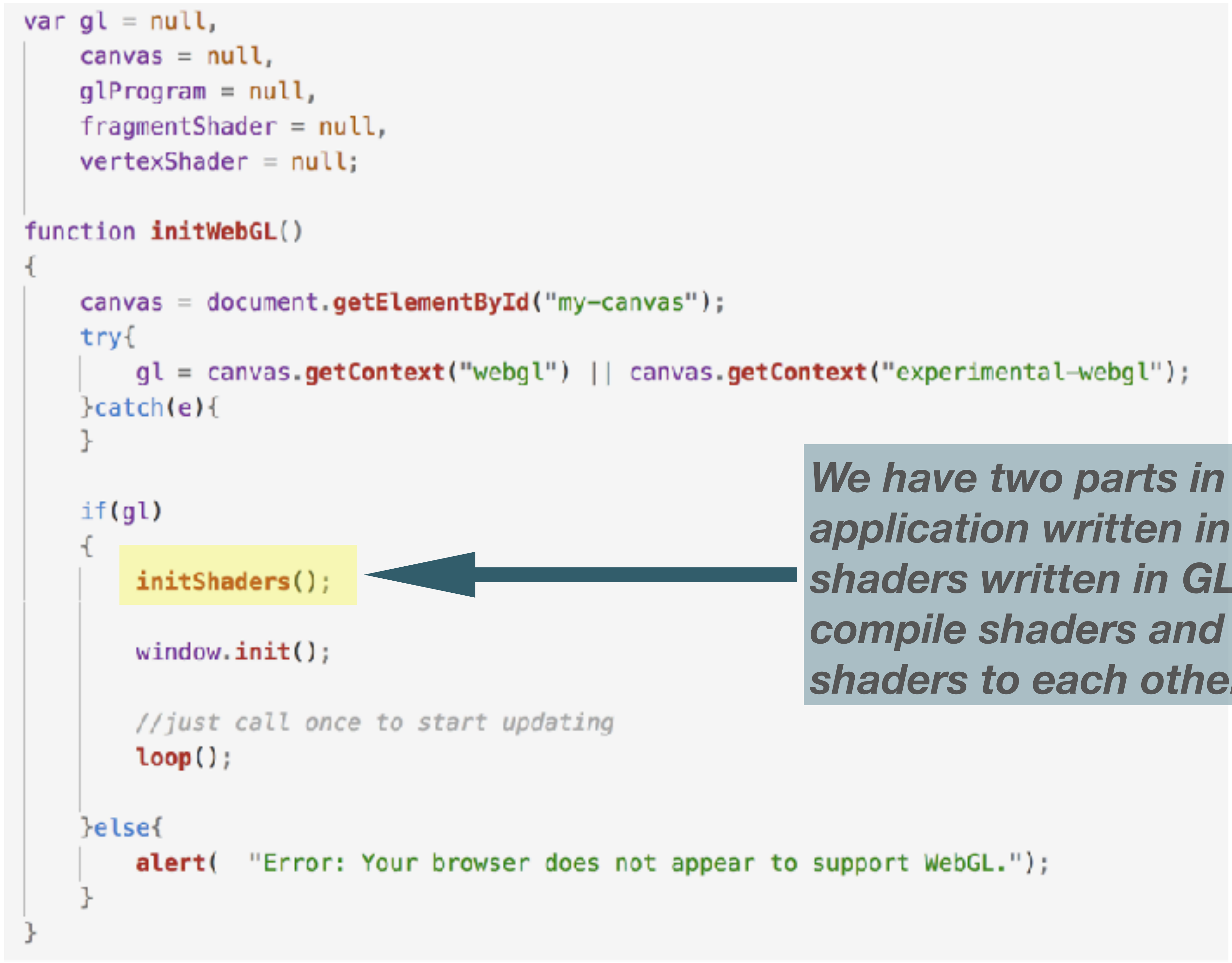
initWebGL()



```
var gl = null,
    canvas = null,
    glProgram = null,
    fragmentShader = null,
    vertexShader = null;

function initWebGL()
{
    canvas = document.getElementById("my-canvas");
    try{
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }catch(e){
    }

    if(gl)
    {
        initShaders();

        window.init();

        //just call once to start updating
        loop();

    }else{
        alert(  "Error: Your browser does not appear to support WebGL.");
    }
}
```

if your browser supports webgl

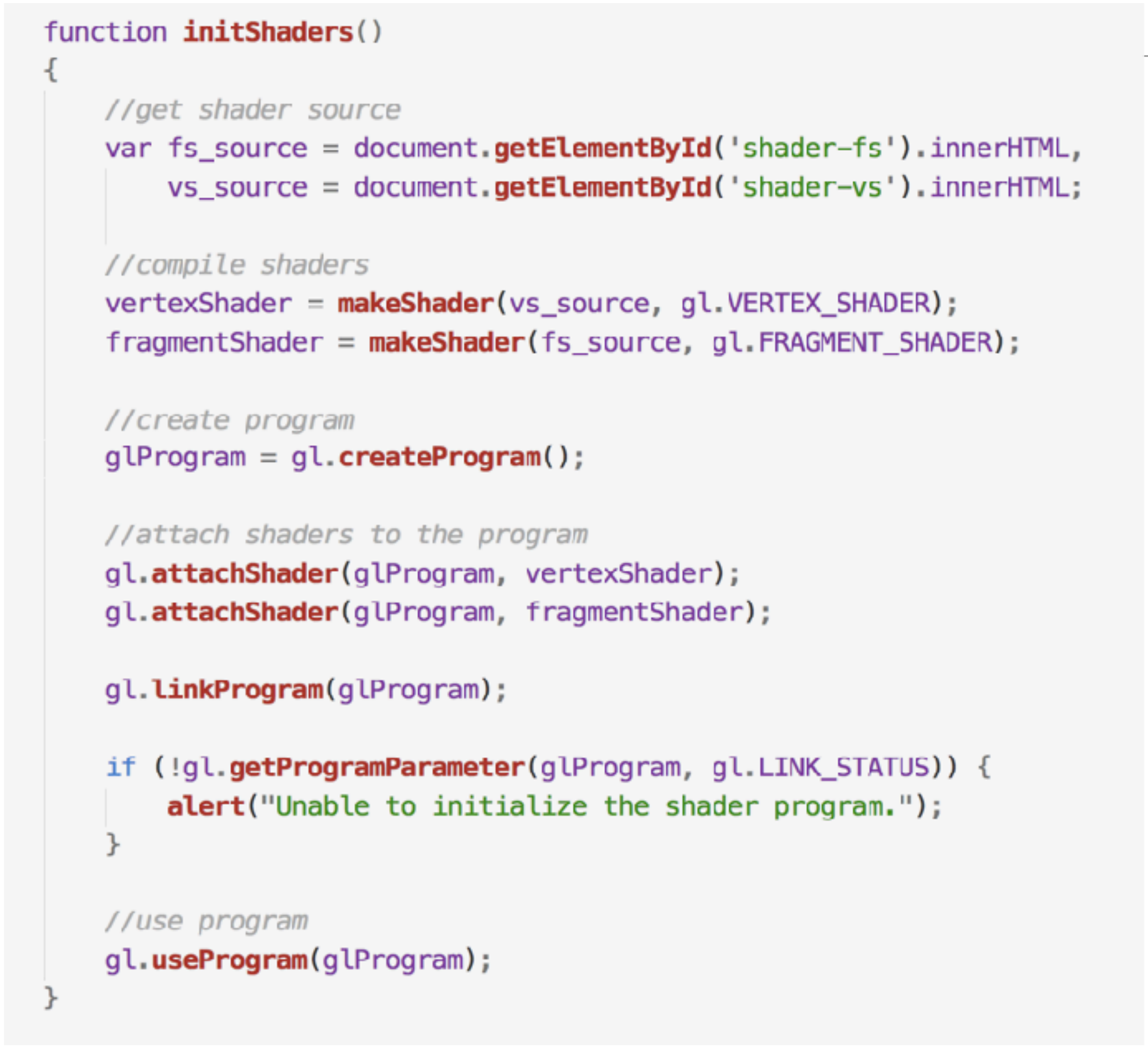
initWebGL()



```
var gl = null,
    canvas = null,
    glProgram = null,
    fragmentShader = null,
    vertexShader = null;

function initWebGL()
{
    canvas = document.getElementById("my-canvas");
    try{
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }catch(e){
    }

    if(gl)
    {
        initShaders();

        window.init();

        //just call once to start updating
        loop();

    }else{
        alert(  "Error: Your browser does not appear to support WebGL.");
    }
}
```

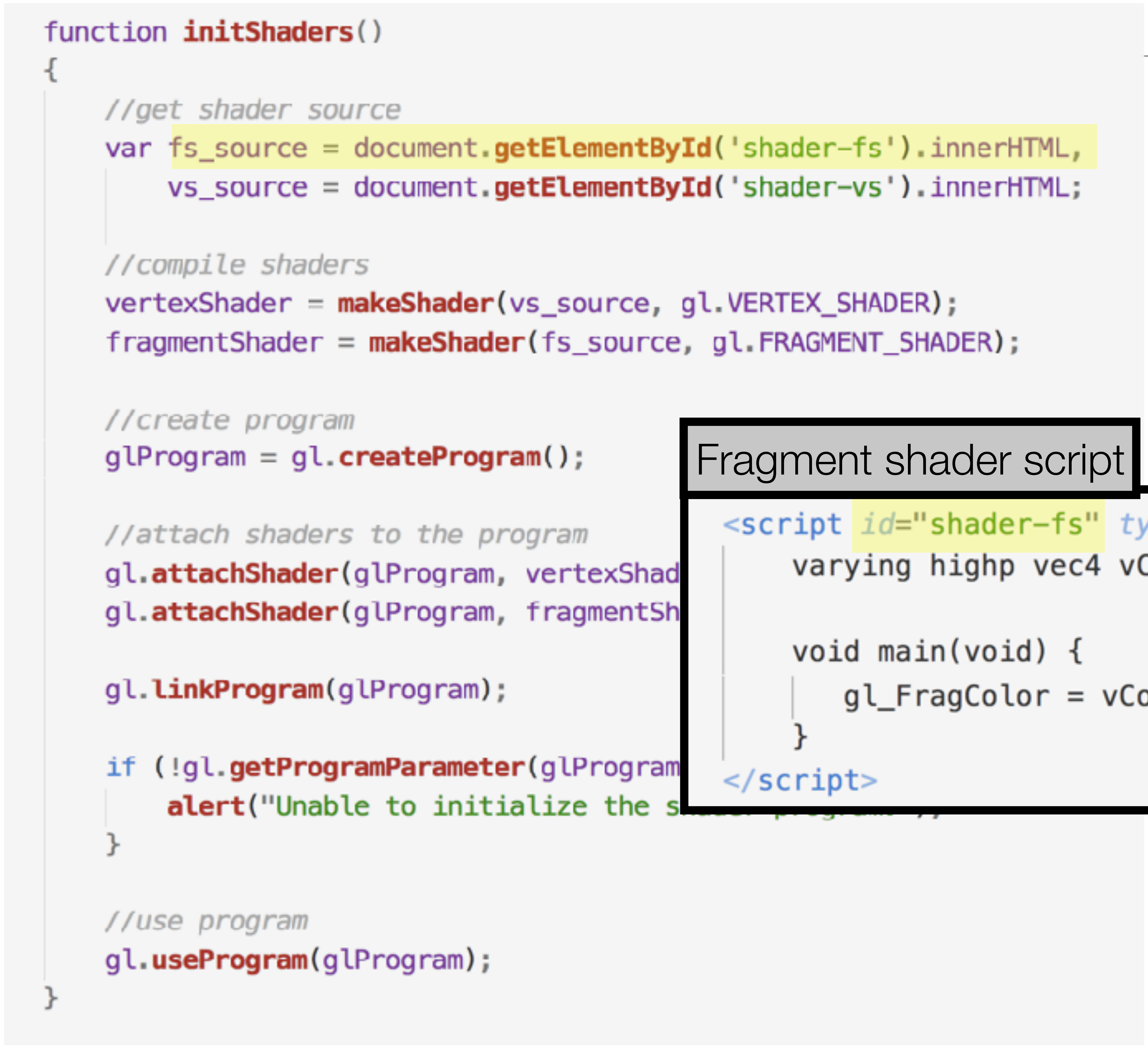
We have two parts in the program, the application written in javaScript and the shaders written in GLSL. initShaders compile shaders and link application and shaders to each other.

initShaders()



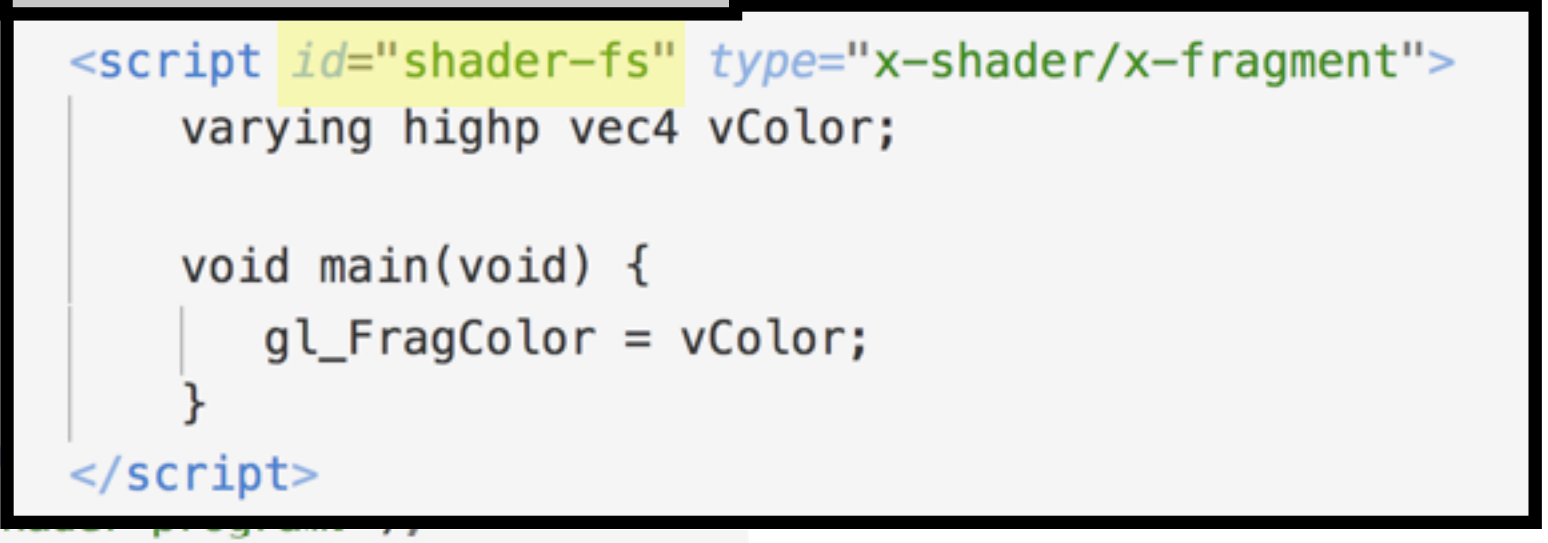
```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

initShaders()



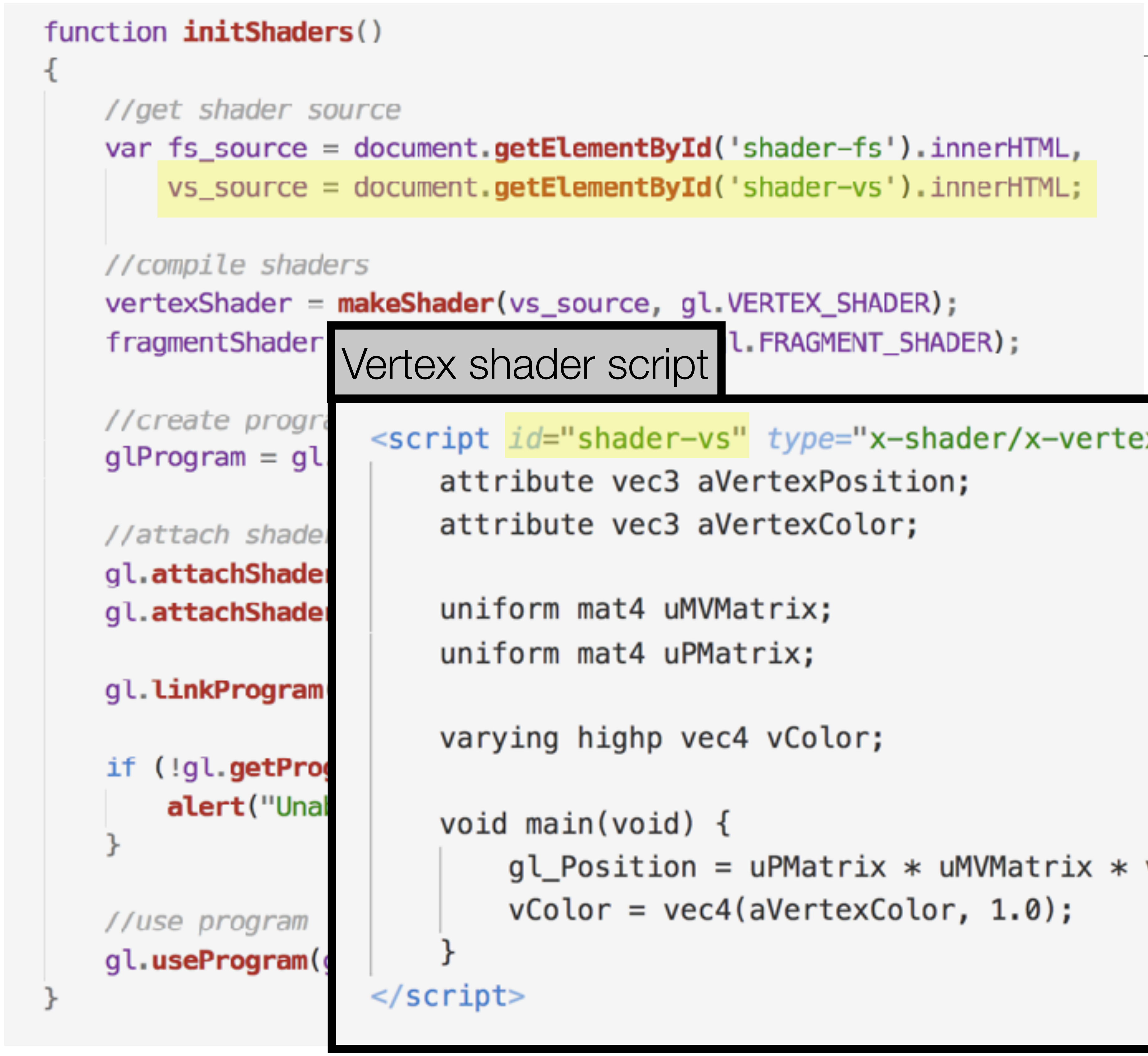
```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShad
    gl.attachShader(glProgram, fragmentSh

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram
        alert("Unable to initialize the s
    }

    //use program
    gl.useProgram(glProgram);
}
```

Fragment shader script



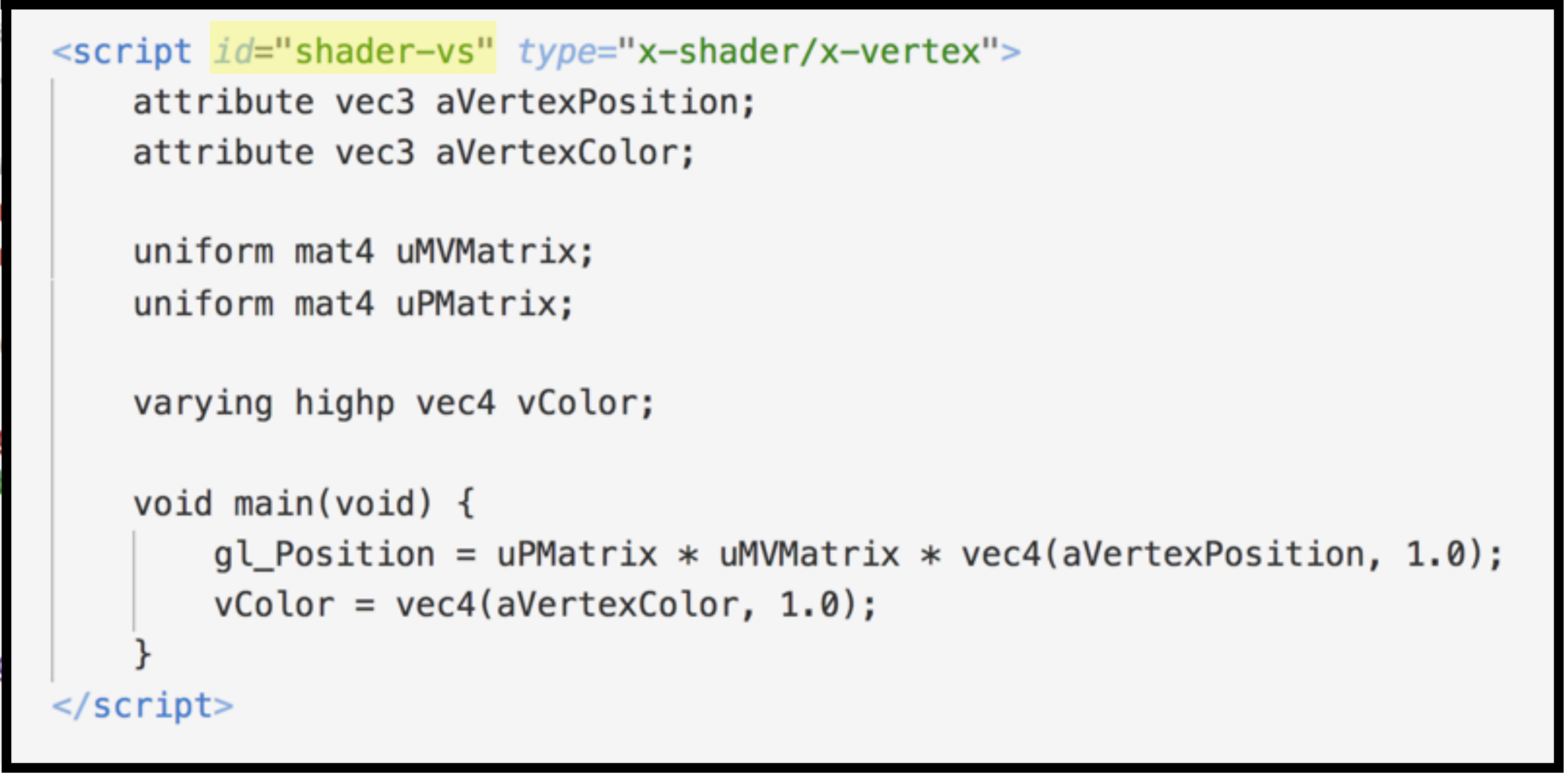
```
<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;

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

initShaders()



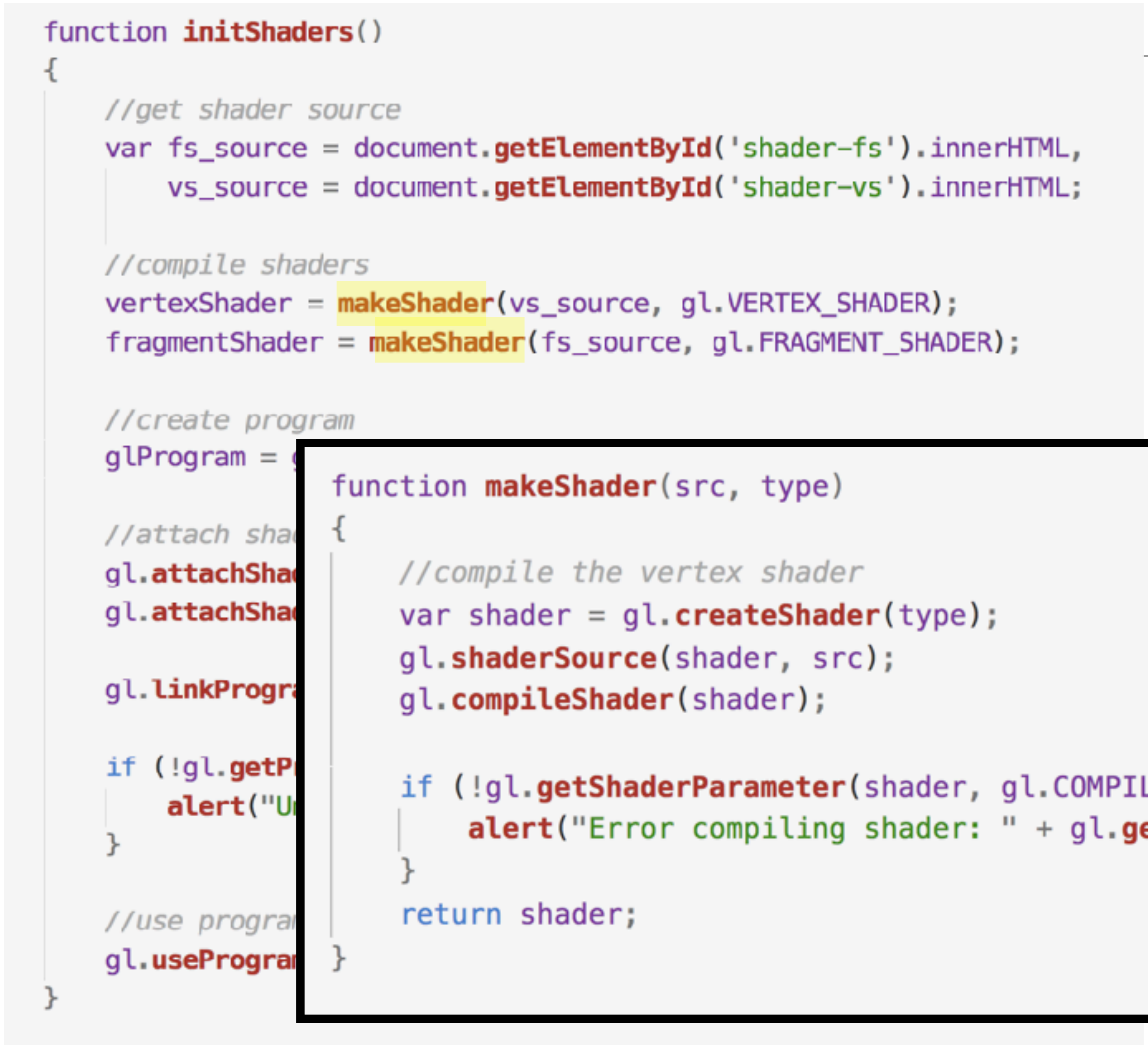
```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader                                 l.FRAGMENT_SHADER);

    //create progr
    glProgram = gl

    //attach shade
    gl.attachShade
    gl.attachShade

    gl.linkProgram

    if (!gl.getProg
        alert("Unal
    }

    //use program
    gl.useProgram(
}
```

Vertex shader script

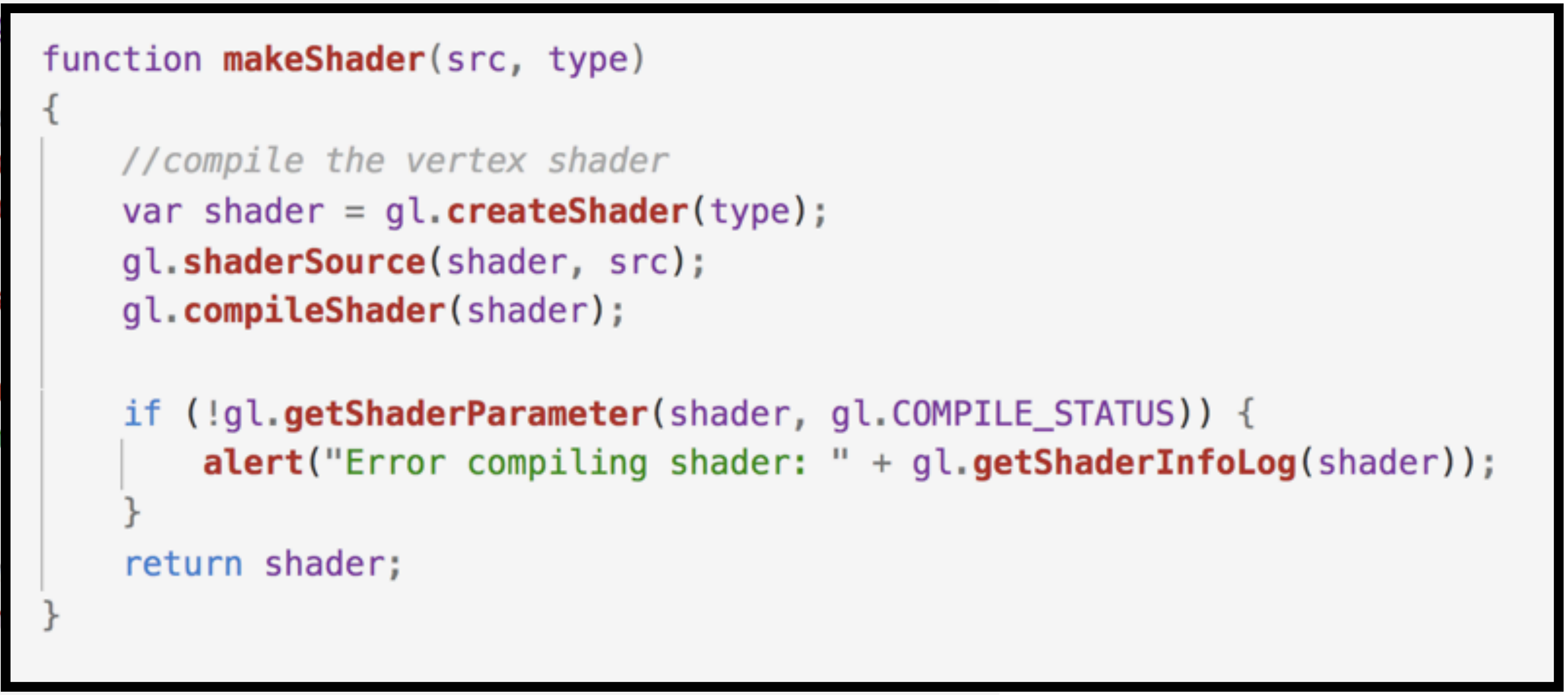


```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

initShaders()

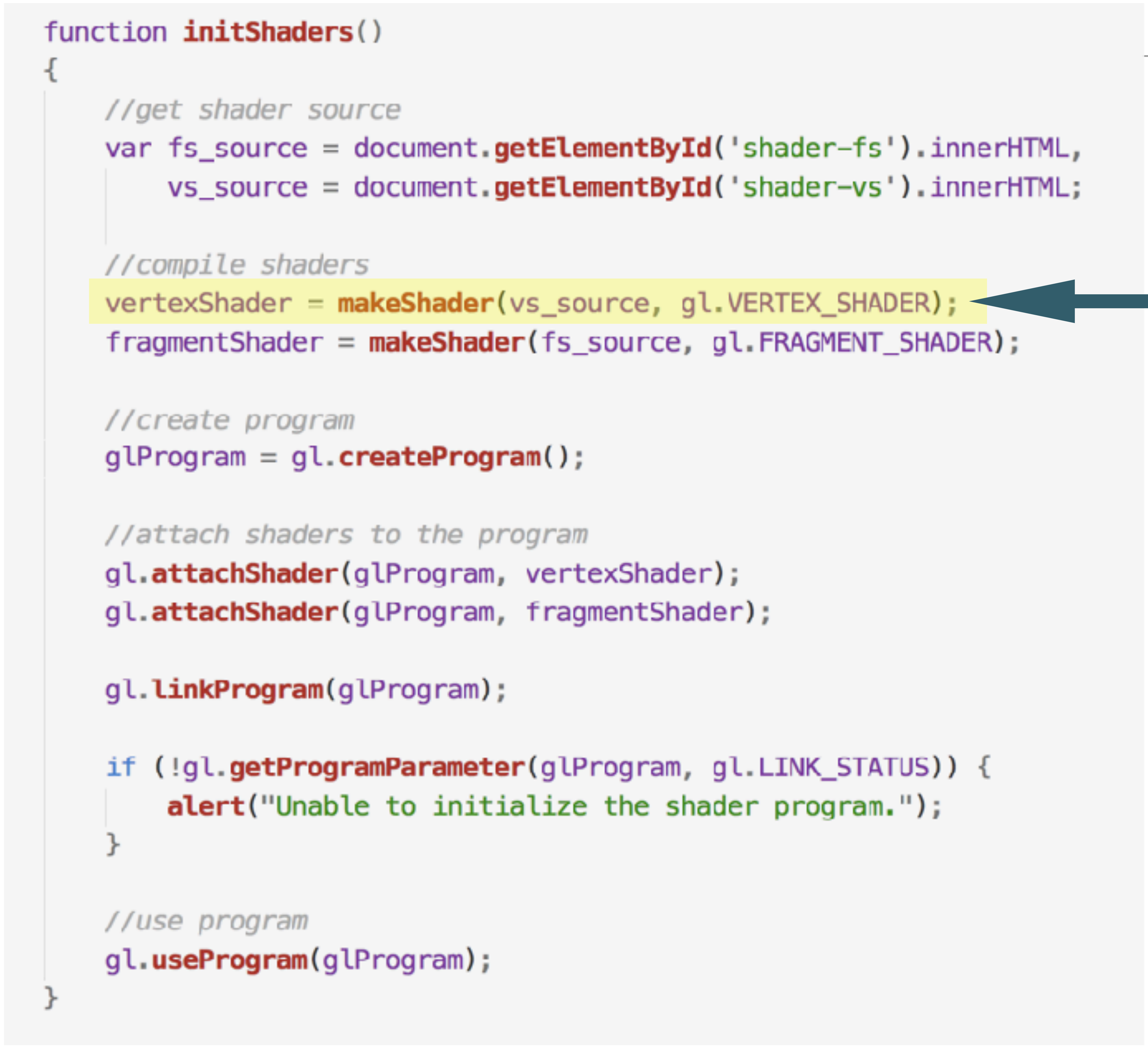


```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = g

    //attach sha
    gl.attachSha
    gl.attachSha

    gl.linkProgra

    if (!gl.getP
        alert("U
    }

    //use progra
    gl.usePrograi
}
```



```
function makeShader(src, type)
{
    //compile the vertex shader
    var shader = gl.createShader(type);
    gl.shaderSource(shader, src);
    gl.compileShader(shader);

    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert("Error compiling shader: " + gl.getShaderInfoLog(shader));
    }
    return shader;
}
```

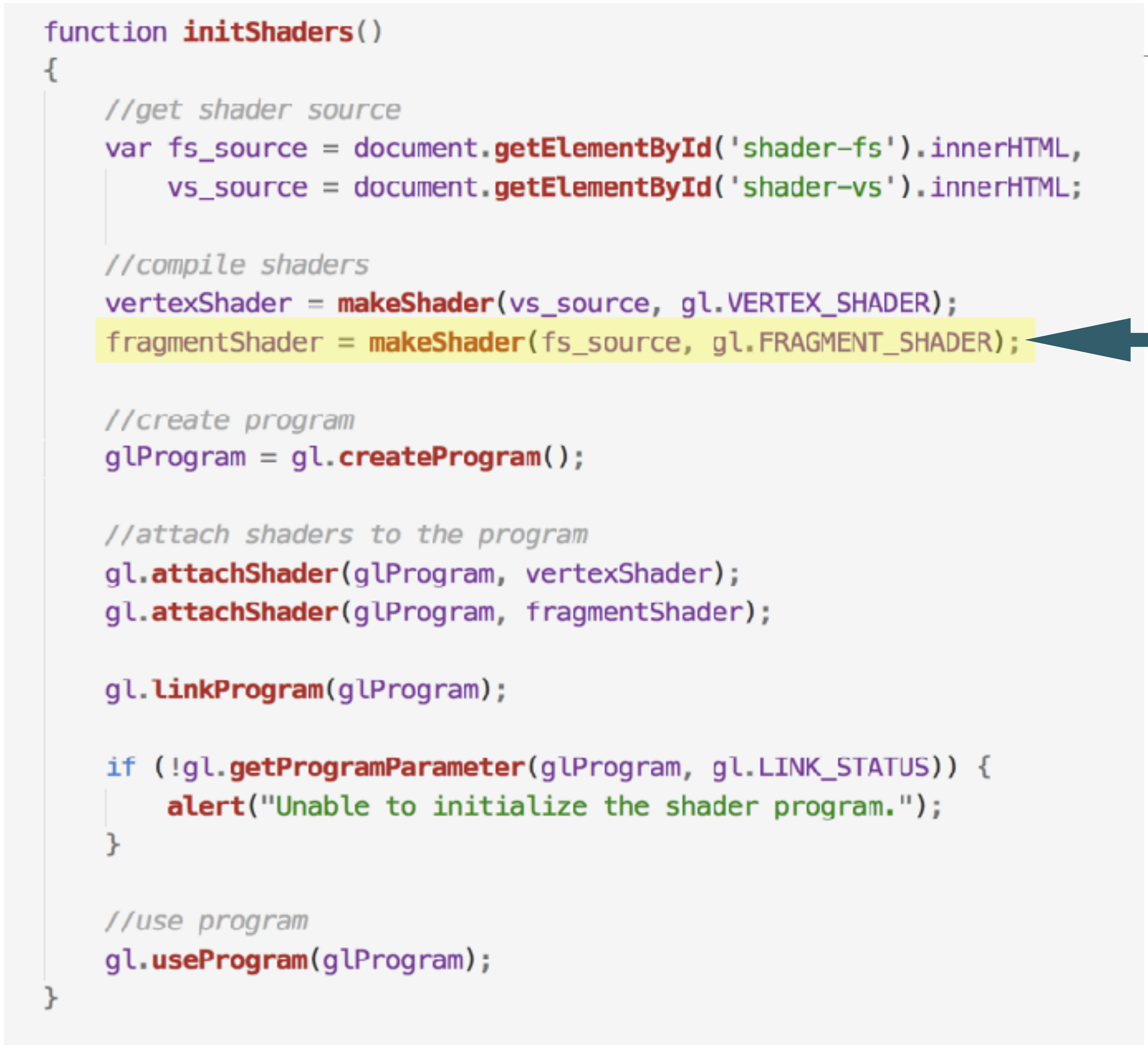
initShaders()



```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

compile vertex shader

initShaders(): compile shaders



```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

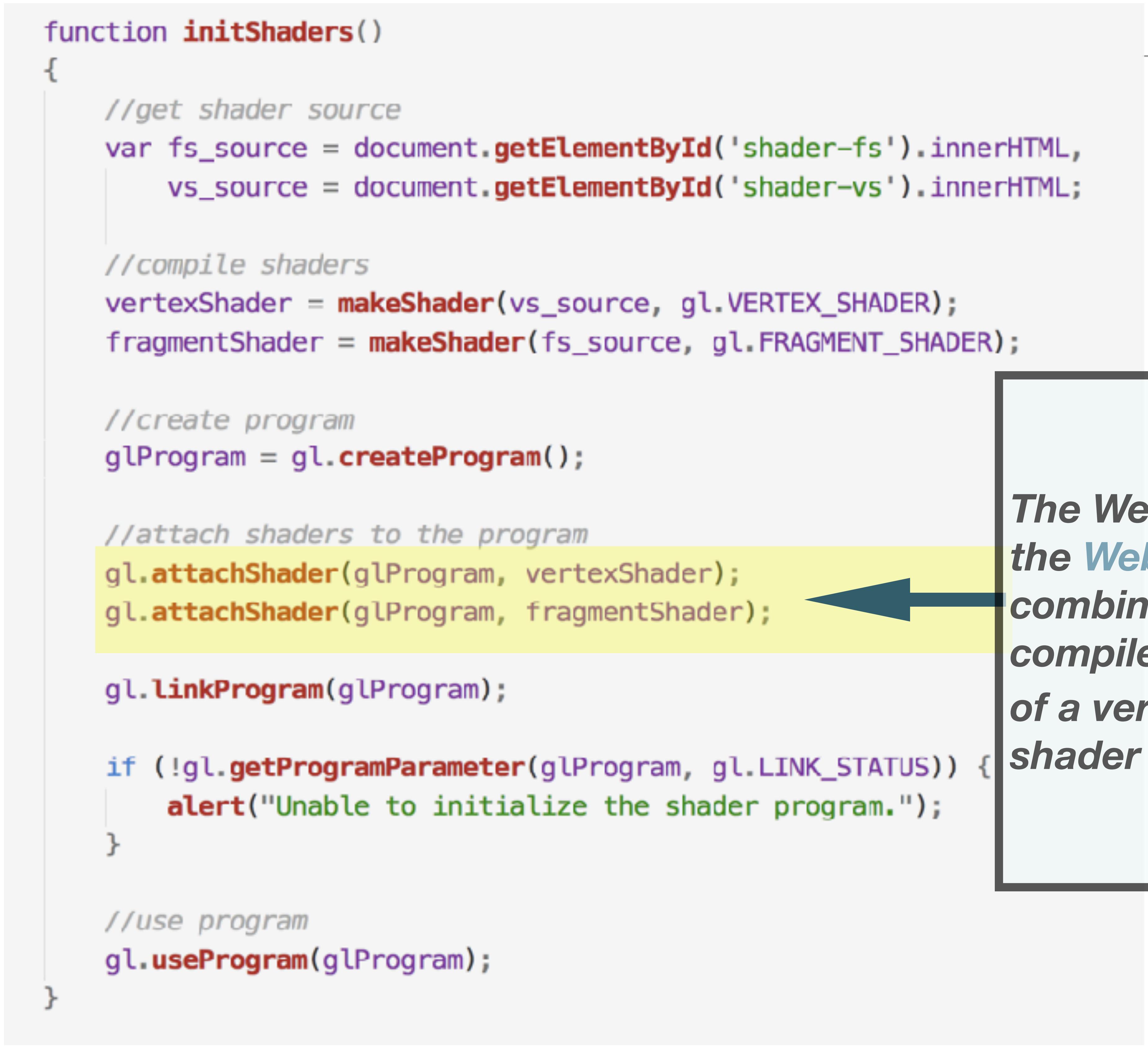
compile fragment shader

initShaders(): WebGLProgram

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

create and initialize a WebGLProgram object

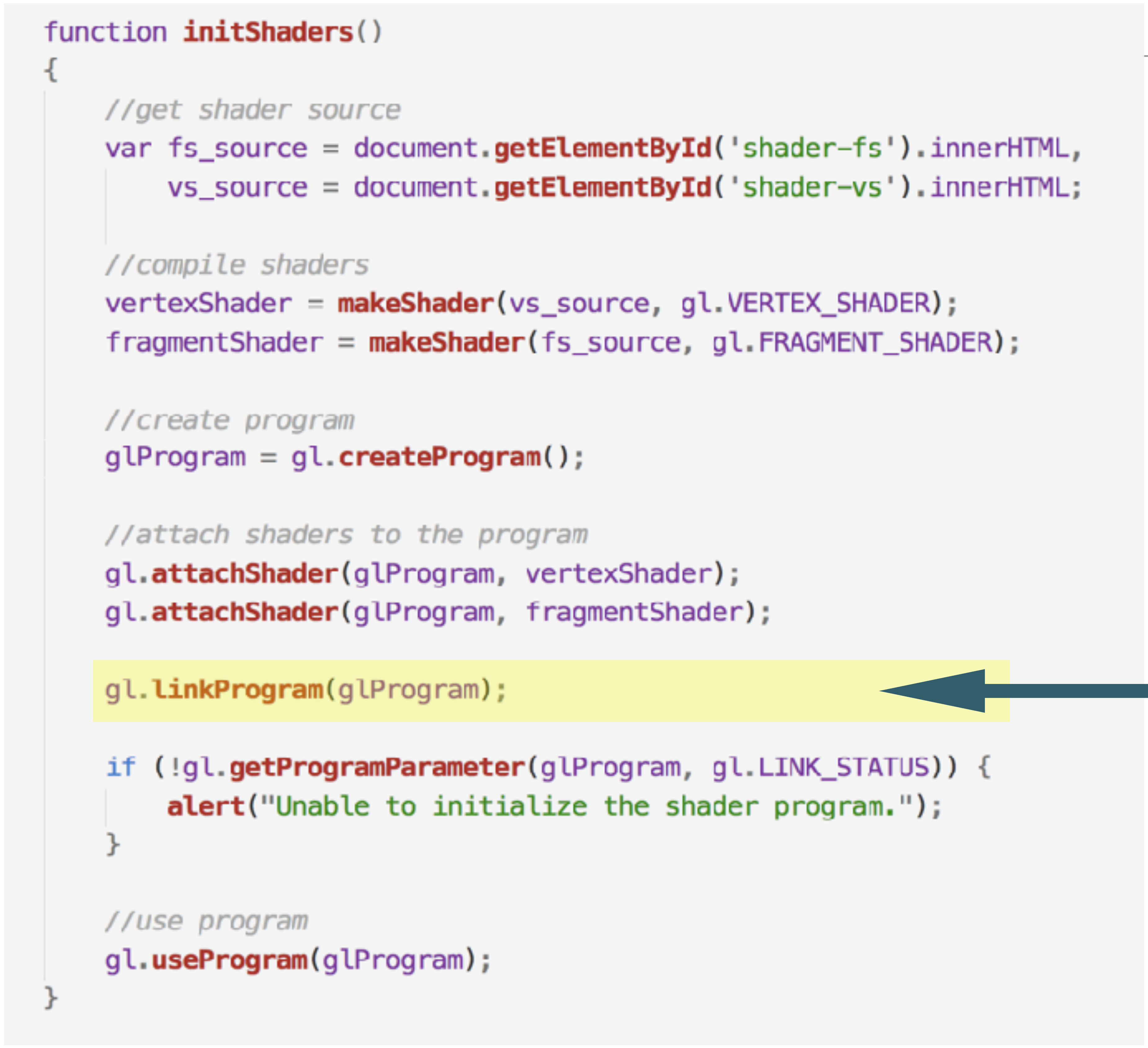
initShaders(): WebGLProgram



```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

The WebGLProgram is part of the WebGL API and is a combination of two compiled `WebGLShader`s consisting of a vertex shader and a fragment shader (both written in GLSL).

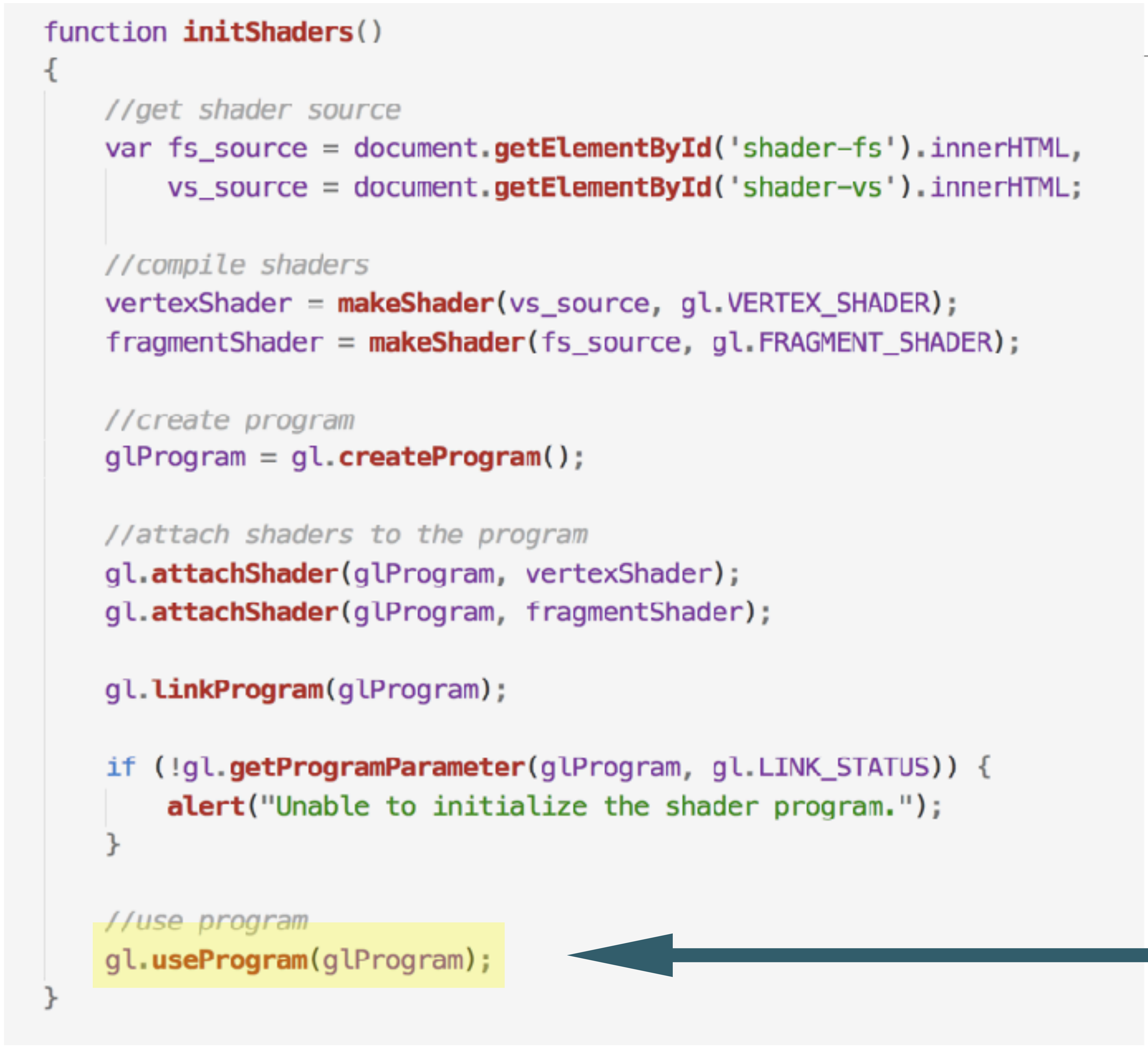
initShaders(): WebGLProgram



```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

if shader compilation is successful, the application and shaders can be linked together

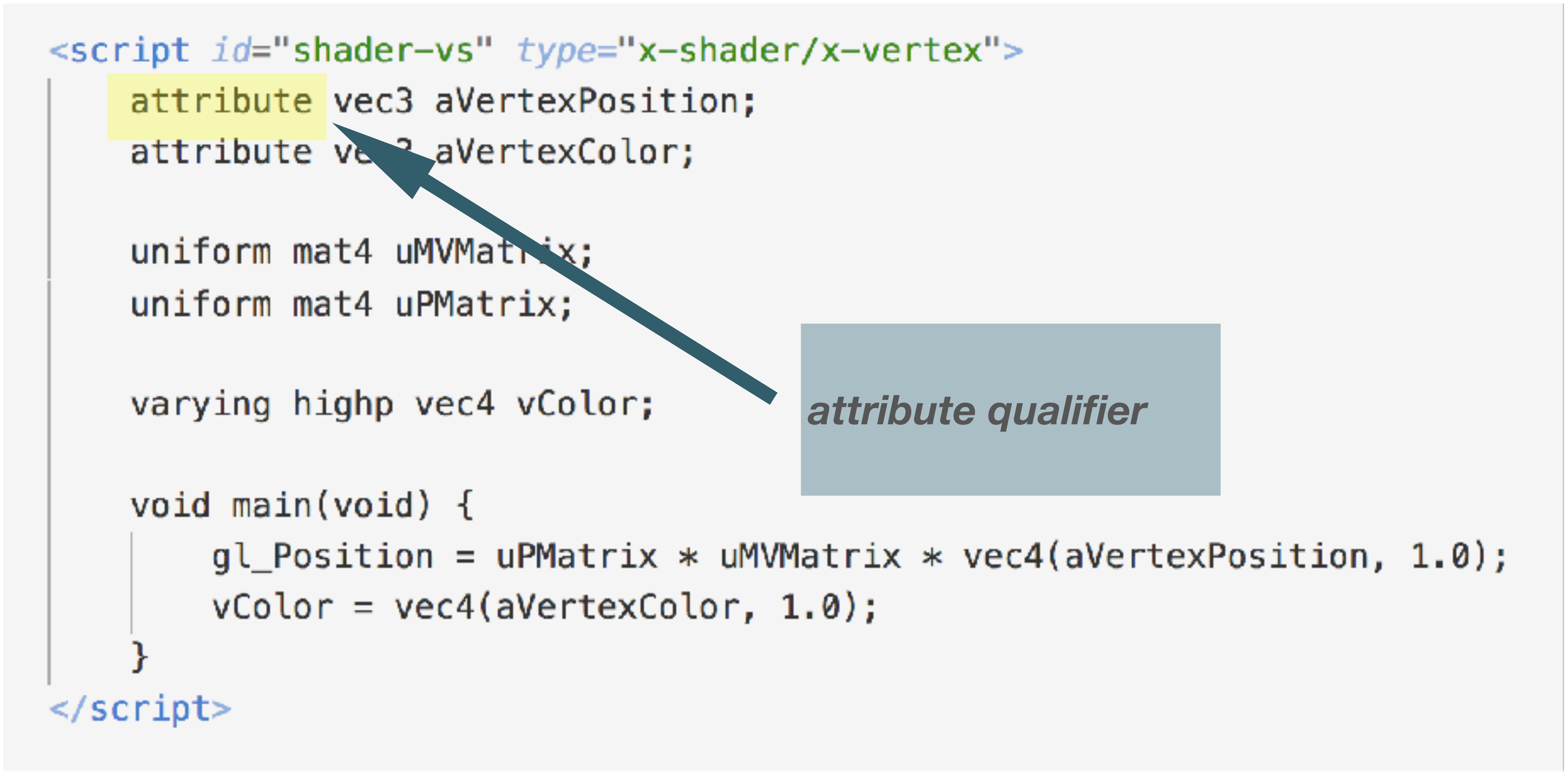
initShaders(): WebGLProgram



```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

tell GPU to use the program

Vertex Attributes



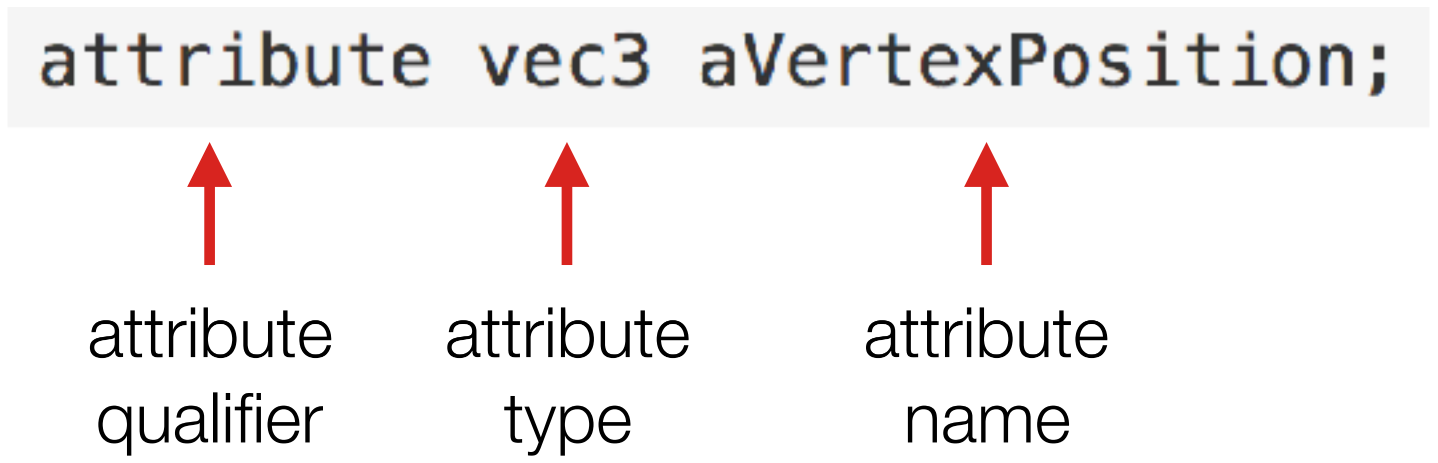
```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

attribute qualifier

Vertex Attributes

Vertex attributes are used to communicate from "outside" to the vertex shader. Values are provided per vertex (and not globally for all vertices). Attributes can't be defined in the fragment shader.

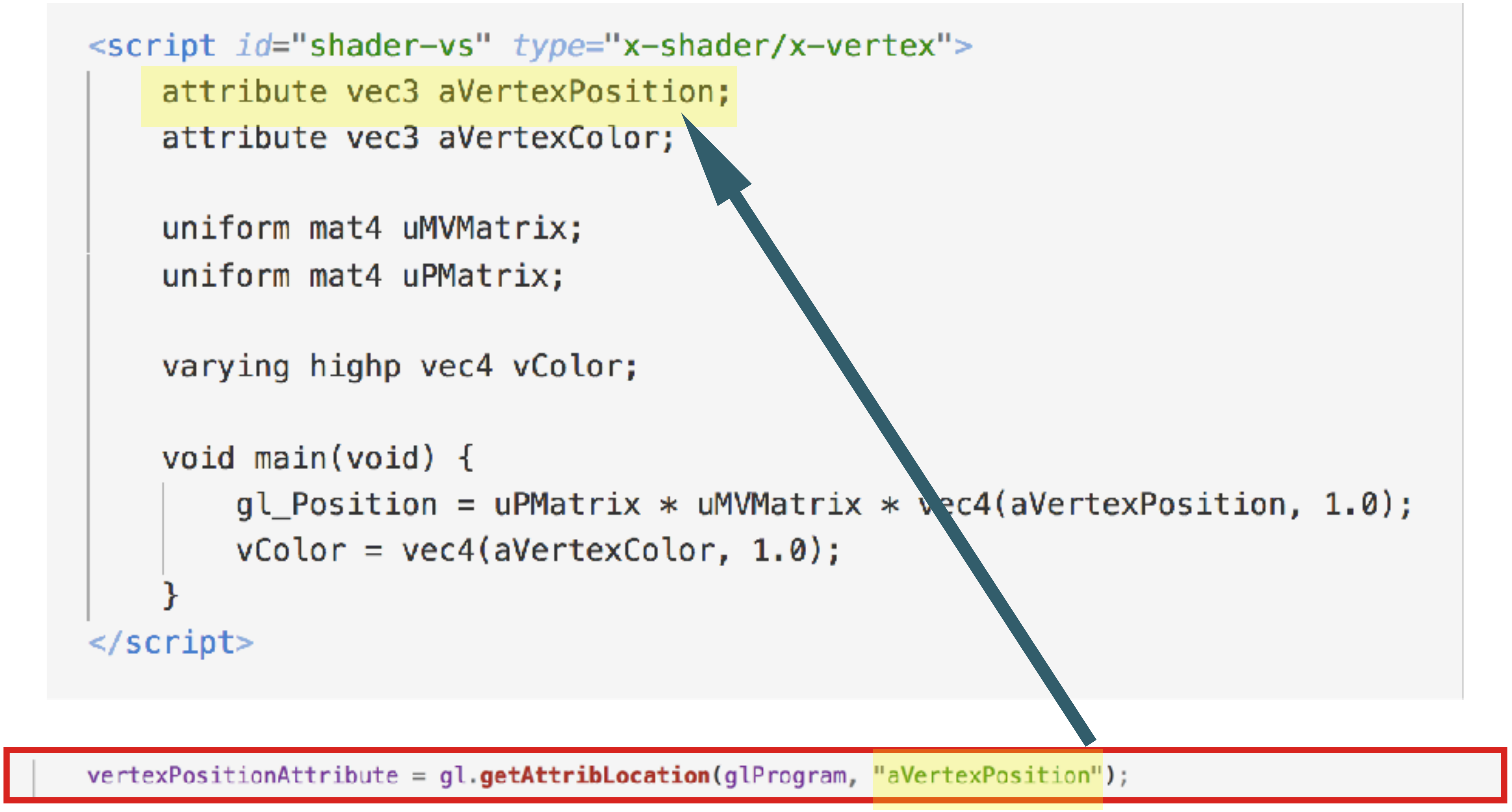
Attributes can be defined in the vertex shader using the "attribute" qualifier*:



```
attribute vec3 aVertexPosition;
```

*https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/attributes.php

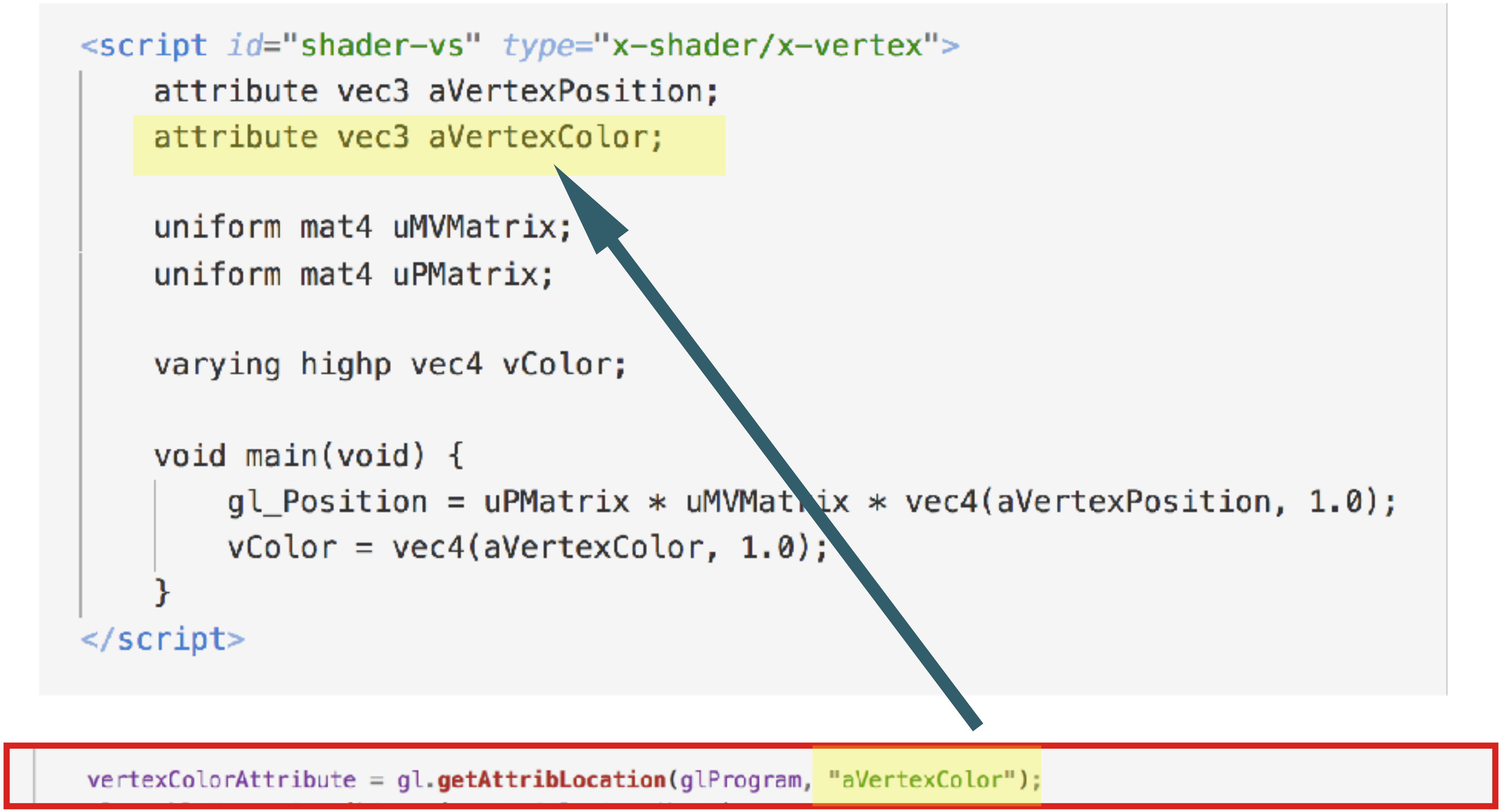
Vertex Attributes



```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

```
vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
```

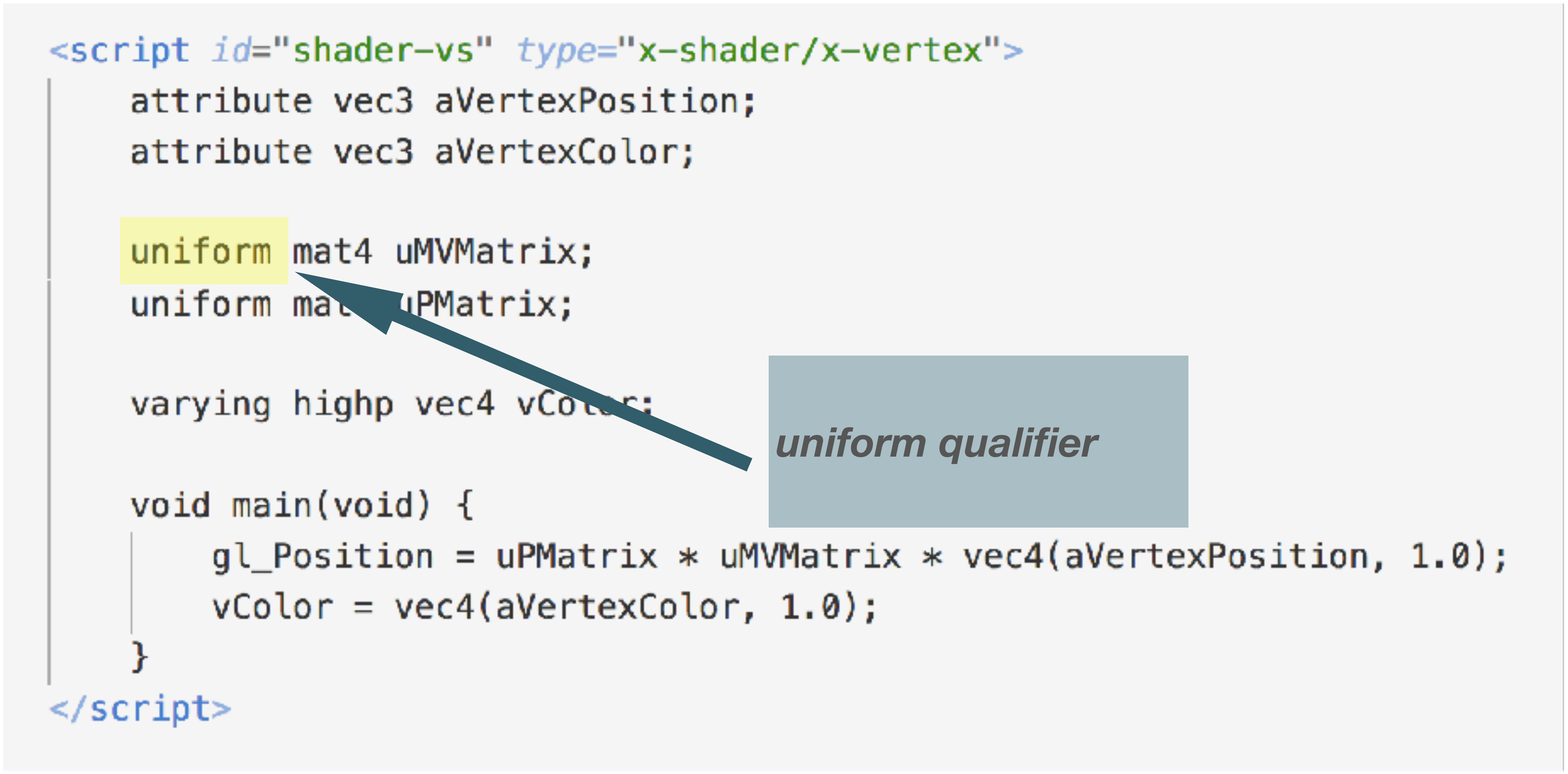
Vertex Attributes



```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

```
vertexColorAttribute = gl.getAttribLocation(glProgram, "aVertexColor");
```

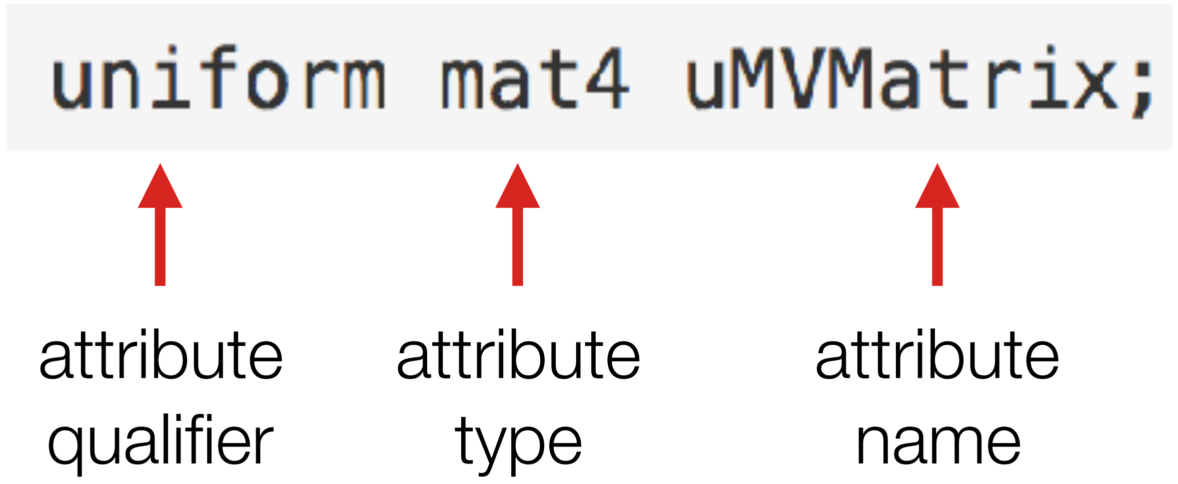
Uniform Variables



```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

Uniform Variables

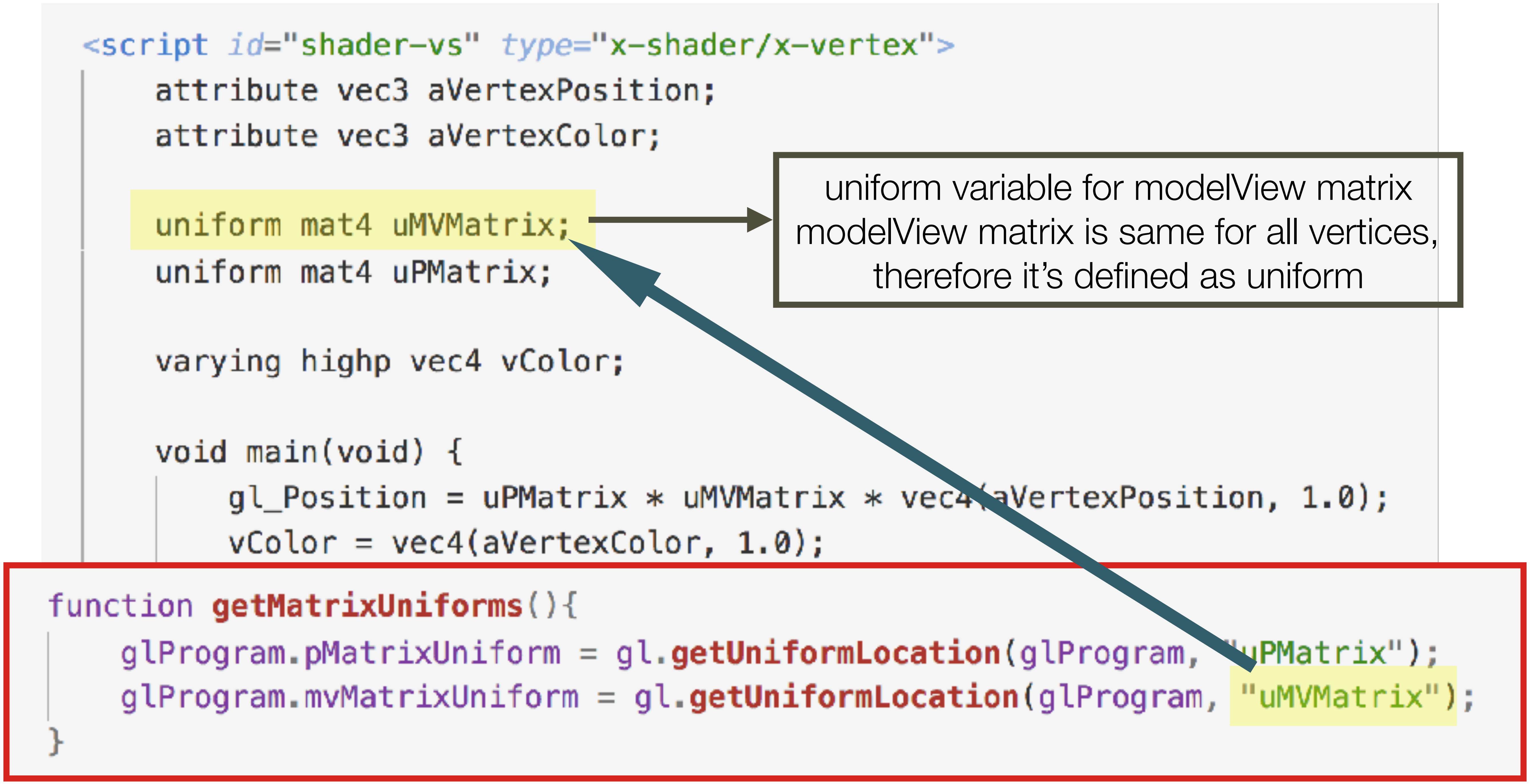
Uniform variables are used to communicate with your vertex or fragment shader from "outside". Uniform variables are **read-only** and have the same value among all processed vertices. You can only change them within your javascript program*



```
uniform mat4 uMVMatrix;
```

*https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/uniform.php

Uniform Variables

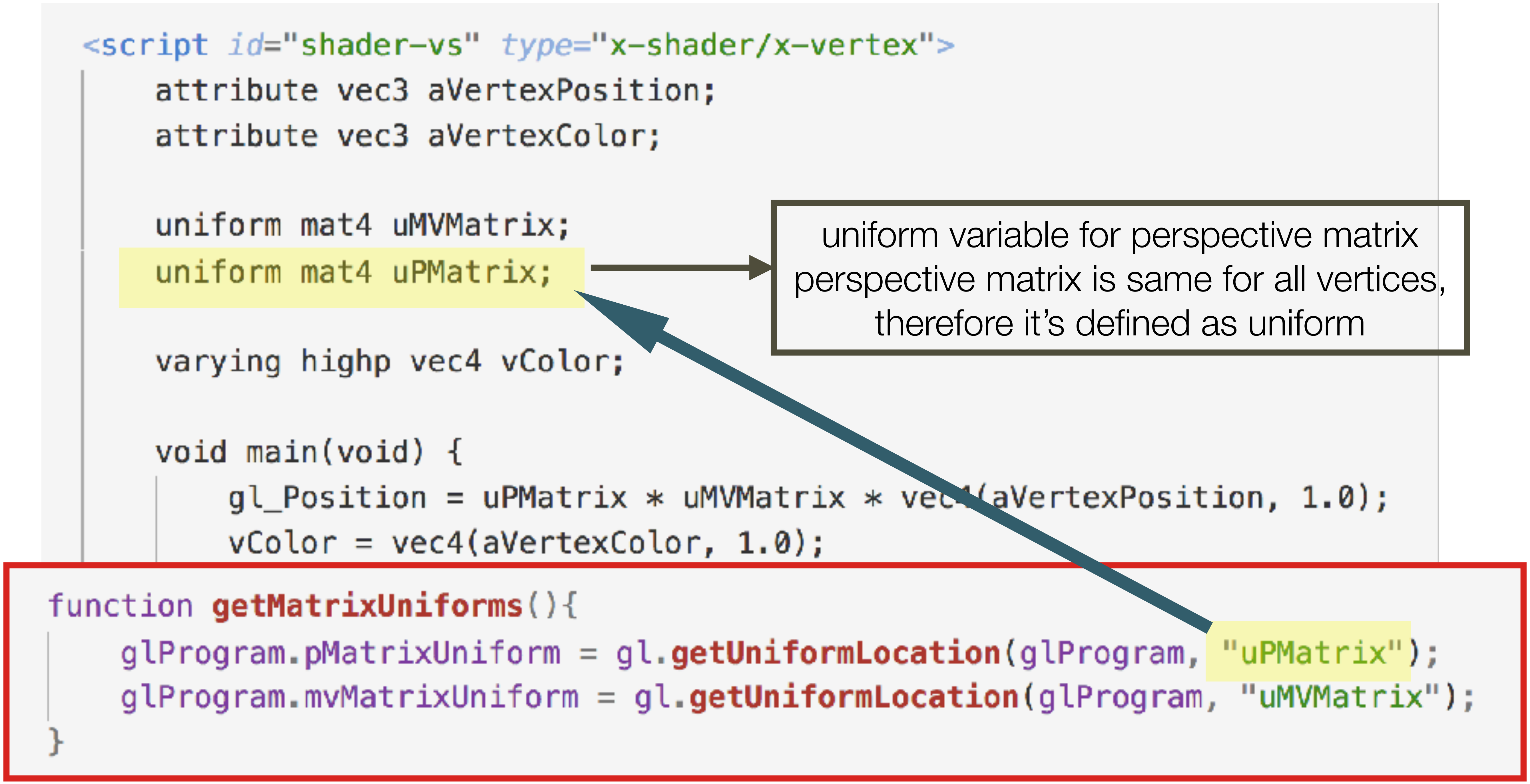


```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
```

```
function getMatrixUniforms(){
    glProgram.pMatrixUniform = gl.getUniformLocation(glProgram, "uPMatrix");
    glProgram.mvMatrixUniform = gl.getUniformLocation(glProgram, "uMVMatrix");
}
```

uniform variable for modelView matrix
modelView matrix is same for all vertices,
therefore it’s defined as uniform

Uniform Variables

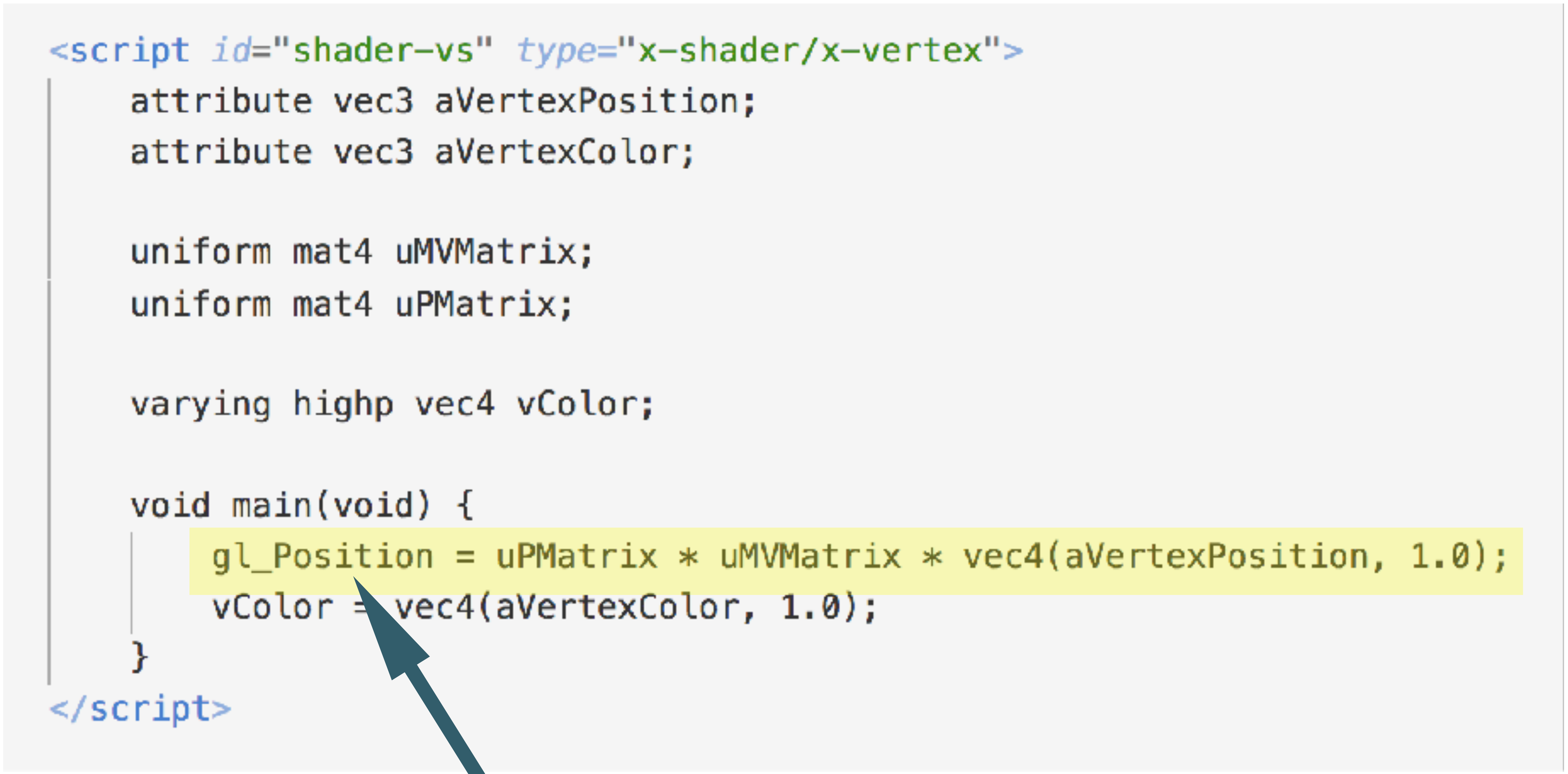


```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
```

uniform variable for perspective matrix
perspective matrix is same for all vertices,
therefore it's defined as uniform

```
function getMatrixUniforms(){
    glProgram.pMatrixUniform = gl.getUniformLocation(glProgram, "uPMatrix");
    glProgram.mvMatrixUniform = gl.getUniformLocation(glProgram, "uMVMatrix");
}
```

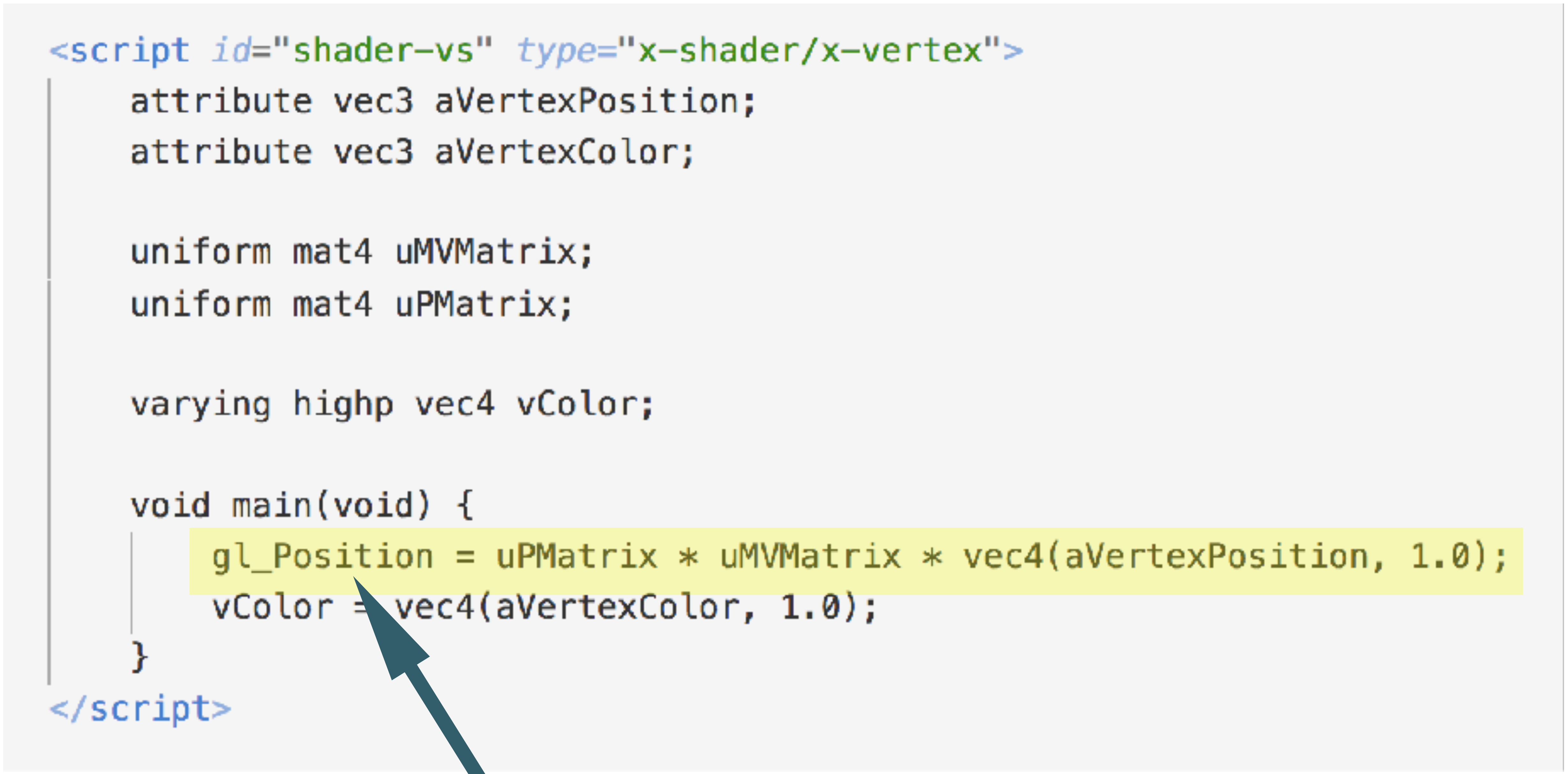
Built-in variables: gl_Position



```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

built-in gl_Position variable

Built-in variables



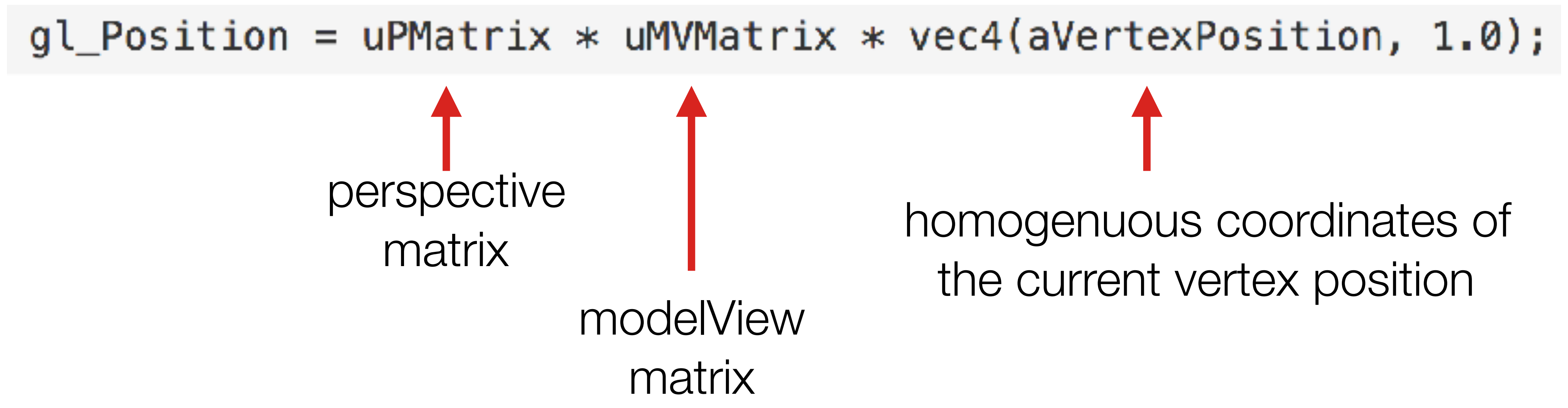
```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

built-in gl_Position variable

Built-in variables: gl_Position

The OpenGL Shading Language defines a number of special variables for the various shader stages. These **built-in variables** (or built-in variables) have special properties. They are usually for communicating with certain fixed-functionality. By convention, all predefined variables start with "gl_"; no user-defined variables may start with this.*

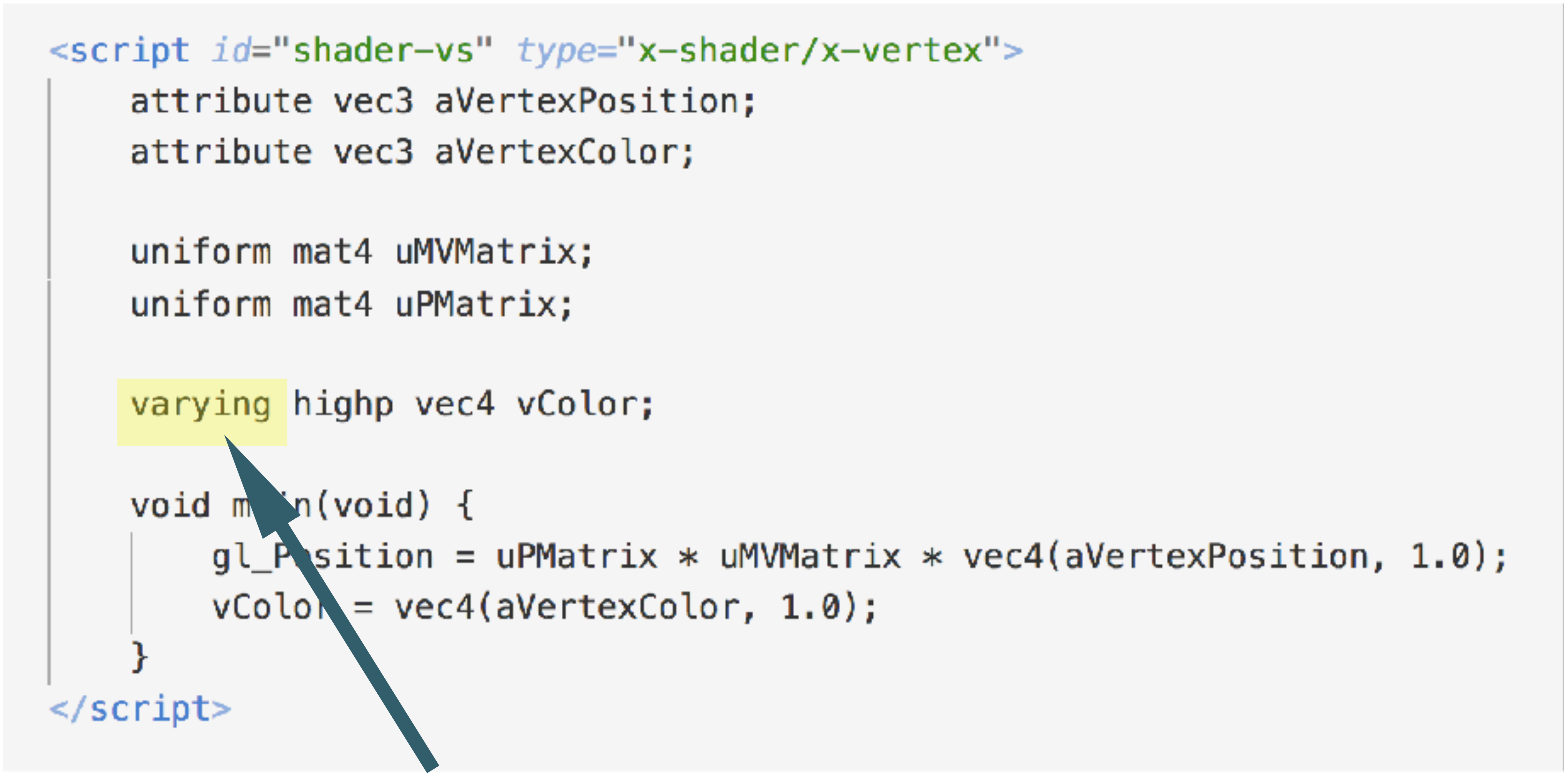
gl_Position — contains the position of the current vertex



```
gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
```

*https://www.khronos.org/opengl/wiki/Built-in_Variable_(GLSL)

Varying variables

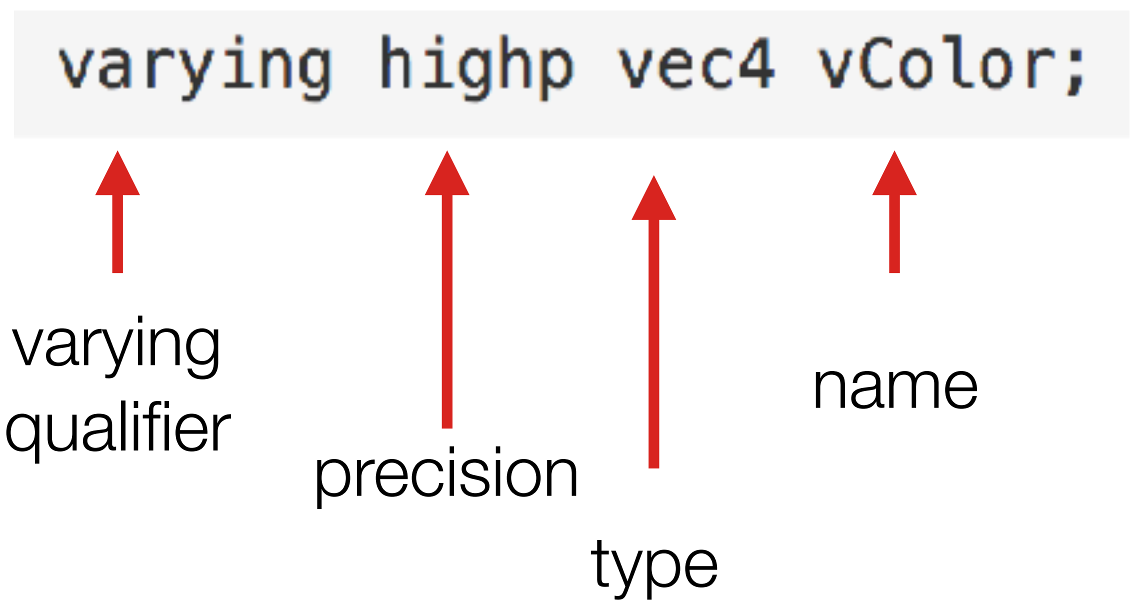


```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

varying qualifier

Varying variables

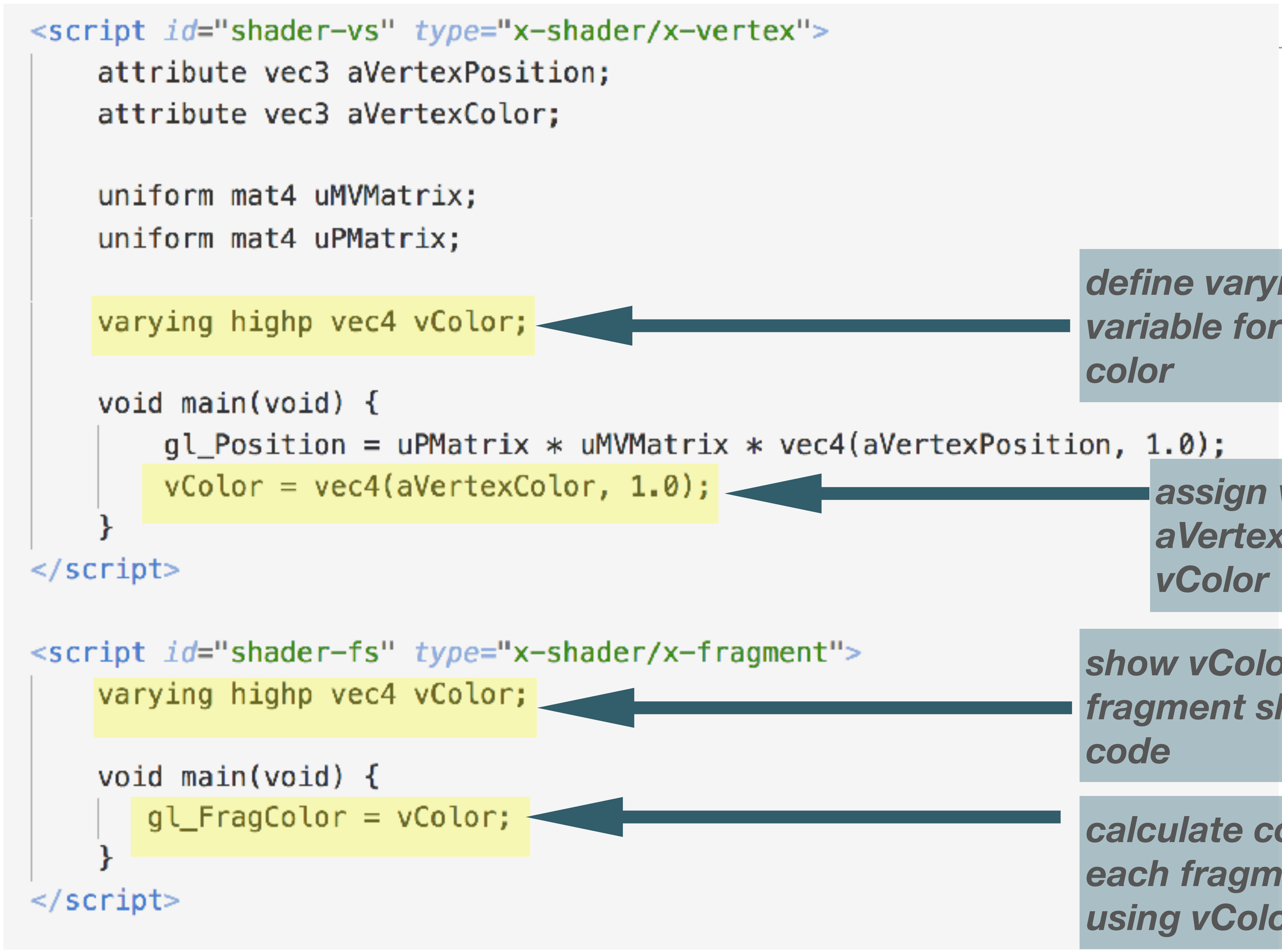
Varying variables provide an interface between Vertex and Fragment Shader. Vertex Shaders compute values per vertex and fragment shaders compute values per fragment. If you define a varying variable in a vertex shader, its value will be interpolated (perspective-correct) over the primitive being rendered and you can access the interpolated value in the fragment shader.*



```
varying highp vec4 vColor;
```

https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/varying.php

Varying variables



```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>

<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;

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

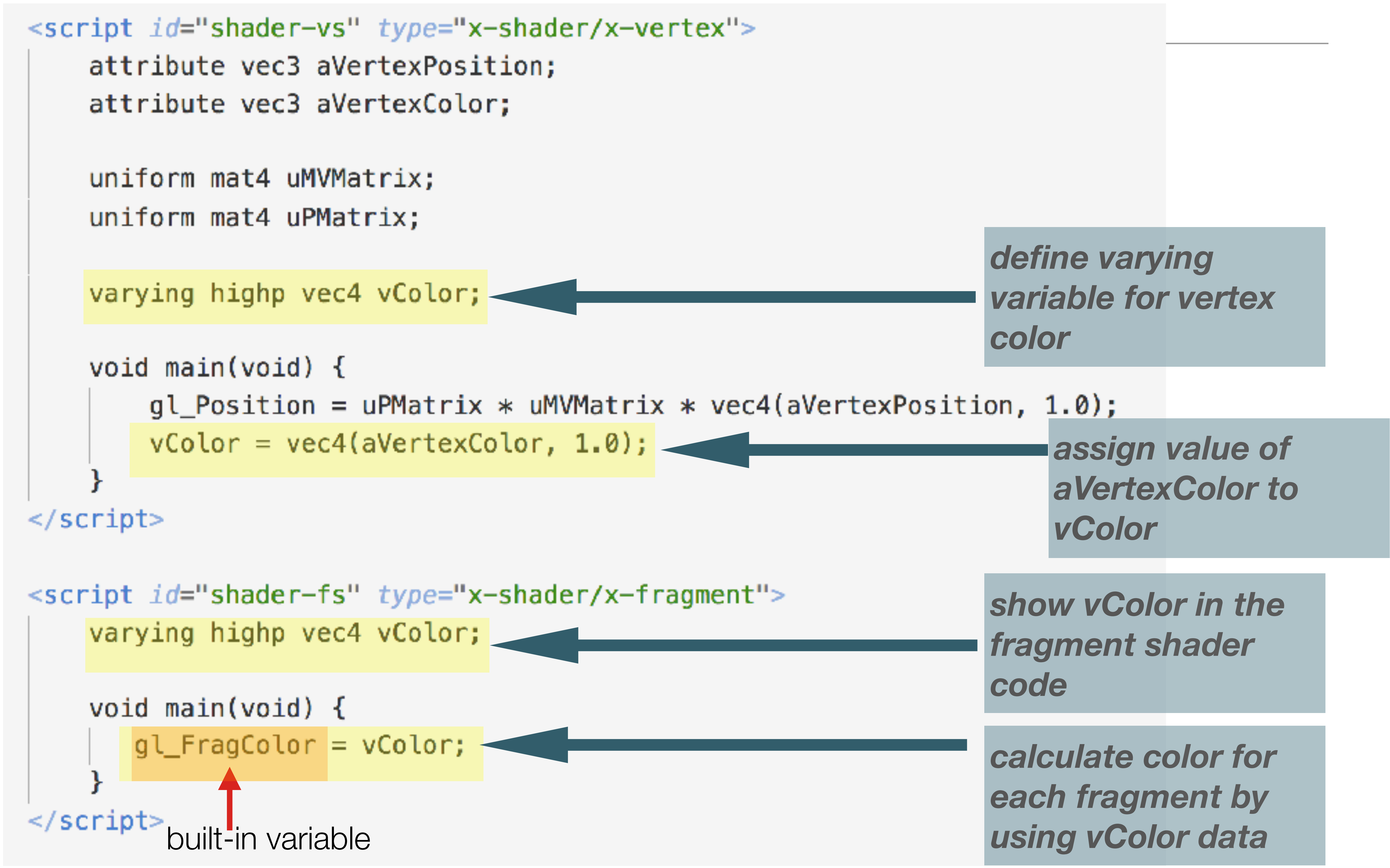
define varying variable for vertex color

assign value of aVertexColor to vColor

show vColor in the fragment shader code

calculate color for each fragment by using vColor data

Varying variables



```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>

<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;

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

define varying variable for vertex color
assign value of aVertexColor to vColor
show vColor in the fragment shader code
calculate color for each fragment by using vColor data
built-in variable