

CMPT 361: Introduction to Computer Graphics

Programming Tutorial 1: Animating a small square

In this tutorial we are going to introduce Sublime, a suitable editor for JavaScript and HTML, and then learn how to make a small square to move downwards by pressing a button. For this tutorial we are going to start exactly from the basic triangle code written in class, which is available on the course webpage.

Part I Sublime

A. Getting Sublime

You can download Sublime from <https://www.sublimetext.com/3>, which you can use it for free, but any other editors you wish to use is ok.

B. Opening Sublime

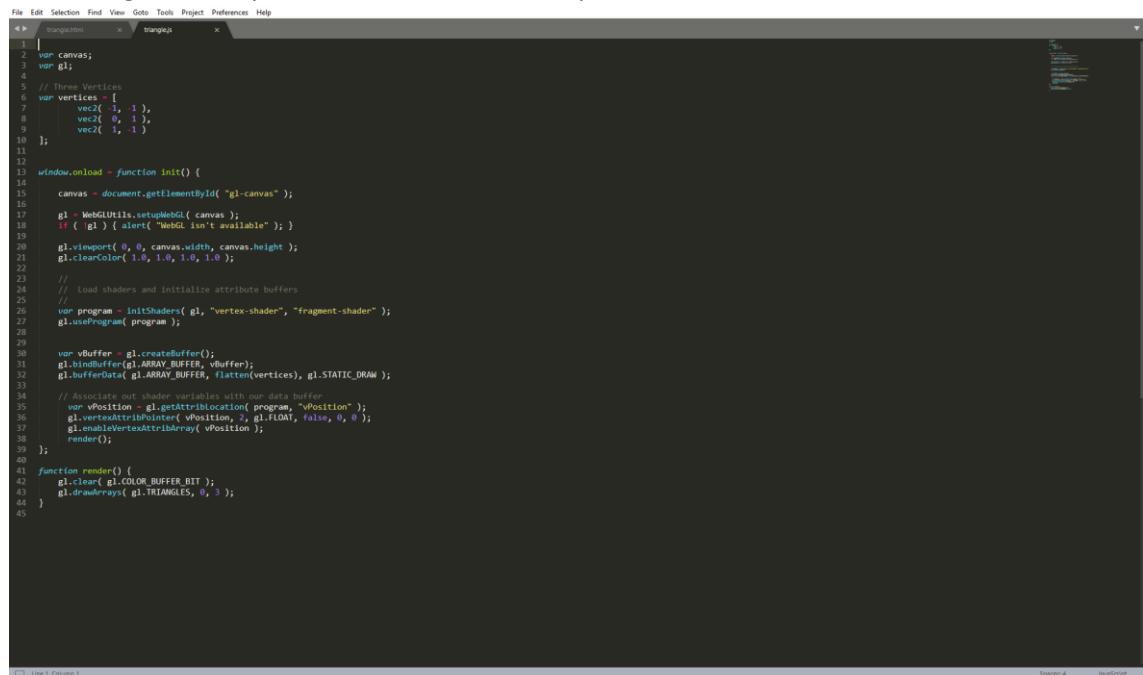
Open sublime_text.exe from the installation directory. By default it is C:\Program Files\Sublime Text 3.

C. Downloading the files

Download the triangle.html and triangle.js files from the course webpage, as well as the codes needed for common folder, and open triangle.html and triangle.js files from File menu.

D. Opening the files in Sublime

After doing these steps we will have our codes opened in our editor.



```
1 |
2 | var canvas;
3 | var gl;
4 |
5 | // Three Vertices
6 | var vertices = [
7 |     vec2( 1, 1 ),
8 |     vec2( 0, 1 ),
9 |     vec2( 1, 1 )
10 | ];
11 |
12 |
13 | window.onload = function init() {
14 |
15 |     canvas = document.getElementById( "gl-canvas" );
16 |
17 |     gl = WebGLUtils.setupWebGL( canvas );
18 |     if ( ! gl ) { alert( "WebGL isn't available" ); }
19 |
20 |     gl.viewport( 0, 0, canvas.width, canvas.height );
21 |     gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
22 |
23 |     //
24 |     // Load shaders and initialize attribute buffers
25 |
26 |     var program = initShaders( gl, "vertex-shader", "fragment-shader" );
27 |     gl.useProgram( program );
28 |
29 |     var vBuffer = gl.createBuffer();
30 |     gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
31 |     gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );
32 |
33 |     // Associate out shader variables with our data buffer
34 |     var vPosition = gl.getAttribLocation( program, "vPosition" );
35 |     gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
36 |     gl.enableVertexAttribArray( vPosition );
37 |     render();
38 | };
39 |
40 |
41 | function render() {
42 |     gl.clear( gl.COLOR_BUFFER_BIT );
43 |     gl.drawArrays( gl.TRIANGLES, 0, 3 );
44 | }
45 |
```

Part II Drawing a Square

First, we are going to draw a basic square at the top middle of the screen by changing the position and number of vertices.

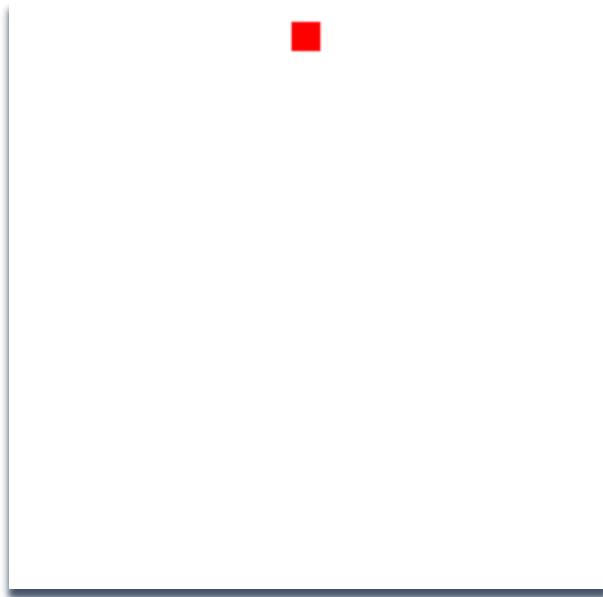
A. Defining vertices

We will redefine the vertices as below:

```
var vertices = [  
    vec2 (-0.05, 0.95),  
    vec2 (0.05, 0.95),  
    vec2 (0.05, 0.85),  
    vec2 (-0.05, 0.95),  
    vec2 (0.05, 0.85),  
    vec2 (-0.05, 0.85)  
];
```

Second, to draw this square, we should change the number of drawing points from 3 to 6 in the render function.

After these modifications, we will get this output.



B. Moving the square

We have two methods to move the square. In order to move our square a little at every frame, we should have a repetitive function, which runs once at each frame.

We will put the last eight lines except render function call in our “render” function, and call it with `window.requestAnimationFrame(render)` command at the end of our render function to call this function again at the next frame. After doing this we have a fixed square, which is redrawn at every frame. The first way to make this square move is to put the definition of our vertices in our “tick” function, and define a global variable named `height` with an initial value of 0.95. Then we should replace 0.95 with `height` and 0.85 with `height-0.1` in vertices array, and then decrease `height` value at every frame by 0.01.

So, our init function would be:

```

window.onload = function init() {

    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );

    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );

    gl.clearColor( 0.0, 0.0, 0.0, 1.0 );

    vBuffer = gl.createBuffer();

    // Load shaders and initialize attribute buffers

    program = initShaders( gl, "vertex-shader", "fragment-shader" );

    gl.useProgram( program );

    // Binding the vertex buffer

    gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);

    // Associate our shader variables with our data buffer

    var vPosition = gl.getAttribLocation( program, "vPosition" );

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

    gl.enableVertexAttribArray( vPosition );

    render();

}

```

And our render function would be like this:

```

function render() {

    // Six Vertices

    var vertices = [

        vec2( -0.05, height),

        vec2( 0.05, height),

        vec2( 0.05, height - 0.1 ),

        vec2( -0.05, height),

        vec2( 0.05, height - 0.1 ),

        vec2( -0.05, height - 0.1 )

    ];

}

```

```

    // Changing the height value for moving the square
    height = height - 0.01;

    // For debugging
    console.log(height);

    document.getElementById("debug").innerHTML = height;

    // Binding the vertex buffer

    gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);

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

    // Clearing the buffer and drawing the square

    gl.clear( gl.COLOR_BUFFER_BIT );

    gl.drawArrays( gl.TRIANGLES, 0, 6 );

    window.requestAnimationFrame(render);
}

```

By doing this, we will see that our square moves downwards until it gets out of the screen, and we can prevent this by using only a simple if.

For the second method, we can move the definition of vertices out of render function, and only pass a height value, which is a uniform variable, to our vertex shader.

C. Adding keyboard listener

For this part we want to start the square movement by pressing the down arrow button.

For this manner we add this command at the top of our script.

```

window.addEventListener("keydown", getKey, false);

```

Then we should check if the arrow down button has been pressed in a new function named getKey so the code for the getKey function would be like this:

```

var pressed = 0;
function getKey(key) {
    if (key.key == "ArrowDown")
        pressed = 1;
}

```

Now, we should only add the condition `pressed == 1` to the particular if for moving the square to make this program work.

D. Debugging code

If there is any error, you can see it in the developer tools menu on the Console tab, but if you want to see the value of a parameter, for example height, you can use `console.log(height);` command, which outputs the value of your parameter into the Console, also one another way is

to define a paragraph (like `<p id="debug"></p>`) in HTML file, and assign its value from JavaScript by using `document.getElementById("debug").innerHTML = height;` command.