COSC363 Computer Graphics

13

WebGL 2.0

Where's the teapot?

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Introduction

- 3D graphics in a browser.
  - Supported browsers: Firefox, Chrome.
  - To test if a browser supports WebGL 2, load the following webpage and you should see a spinning cube.

    https://get.webgl.org/webgl2/
- WebGL 2.0 requires hardware with OpenGL ES 3.0 support.
- No software installation or plugins requried.
- Allows integration with HTML elements.
- WebGL Wiki: https://www.khronos.org/webgl/wiki/Main_Page
- WebGL 2.0 Specs:

  https://www.khronos.org/registry/webgl/specs/latest/2.0/

Introduction

- WebGL programs are written in Javascript (see API Ref)
- Allows easy integration with HTML Document Object Model (DOM) elements
- Scene rendered on a canvas (an HTML5 element) which acts as the viewport.
- JavaScript can register different types of even handlers to process mouse and timer events.
- We will be using the matrix library `gl-matrix-min.js` for vector and matrix operations (glmatrix.net)
- The webpage provides a debugging console. This Javascript console is a useful tool for testing code segments and expressions: https://jsconsole.com/

Limitations

- Not supported by all browsers
- Very slow compared to native OpenGL-4 applications
- Keyboard and special key events cannot be handled
- OpenGL ES 3.0 does not support tessellation or geometry shaders. Hence these shader stages are not currently available in WebGL

Sample Files

WebGL_Files.zip:

- js
- BasicCanvas.html
- Cube1.html
- Cube1Anim.html
- Dolphin.html

- BasicCanvas.js
- Cube1.js
- Cube1Anim.js
- Dolphin.js
- DolphinObj.js
- gl-matrix-min.js
- objparser.js

BasicCanvas.html

- Creates an HTML5 Canvas element of size 500x400 pixels, with id “glCanvas”
- Uses the <script> tag to embed to JavaScript files:
  - BasicCanvas.java
  - gl-matrix-min.js

```html
<html>
    <head>
        <title>COSC363:BasicCanvas</title>
        <script src="js/gl-matrix-min.js" defer></script>
        <script src="js/BasicCanvas.js" defer></script>
    </head>

    <body>
        <canvas id="glCanvas" width = "500" height = "400"></canvas>
    </body>
</html>
```

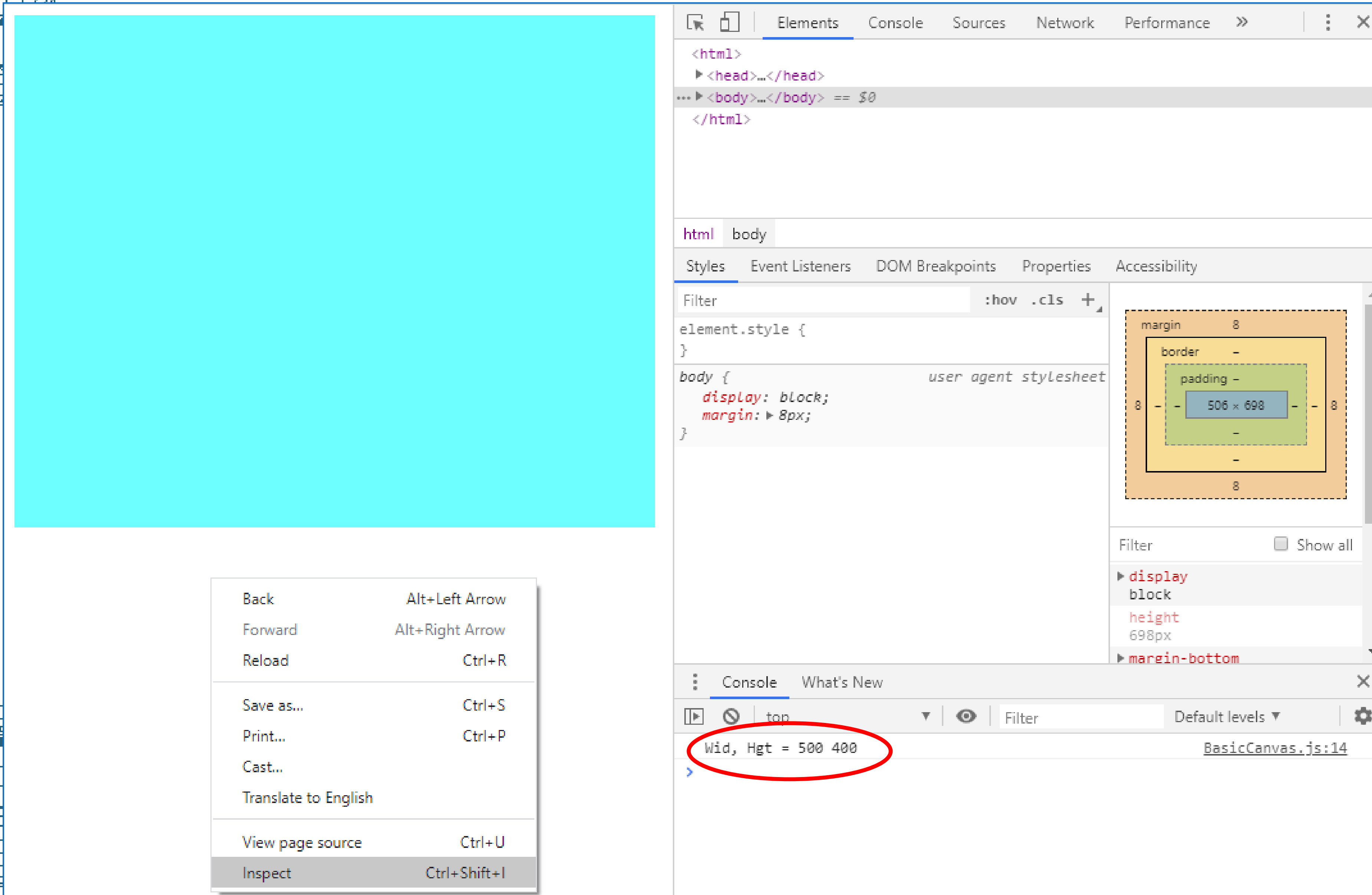
BasicCanvas.js

- Uses the canvas object to accesses the WebGL context.
- The WebGL context provides a handle through which we can access WebGL functions.
- The canvas acts as the viewport for WebGL display commands.

```
function main() {
    const { mat4, mat3, vec3 } = glMatrix;
    const canvas = document.getElementById("glCanvas");
    const gl = canvas.getContext("webgl2");
    if(!gl) {
        alert("Error enabling WebGL context!");
        return;
    }
    canvas.width = canvas.clientWidth;
    canvas.height = canvas.clientHeight;
    gl.viewport(0, 0, canvas.width, canvas.height);
    gl.clearColor(0, 1, 1, 1);
    gl.clear(gl.COLOR_BUFFER_BIT);
    console.log("Wid, Hgt = " + canvas.width + " " + canvas.height);
}

main();
```

BasicCanvas Output



WebGL Buffer Objects

- VAO:

```
vao = gl.createVertexArray();
gl.bindVertexArray(vao);
```

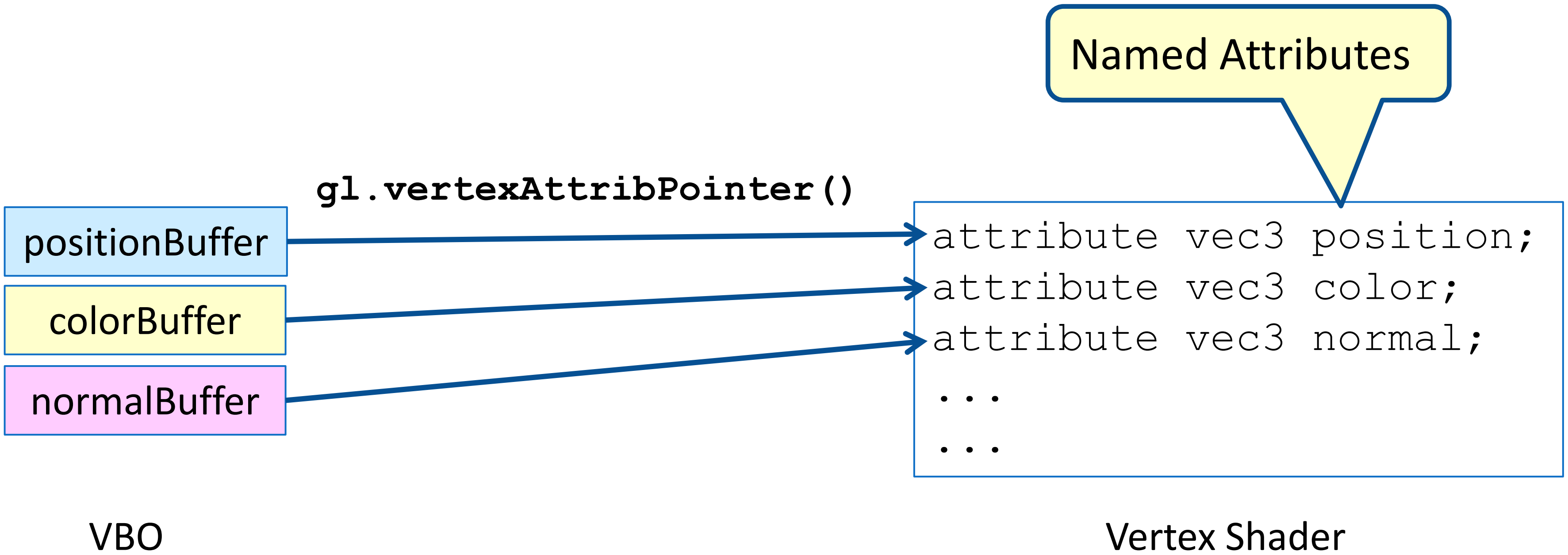
- VBOs (Examples)

```
const positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexData),
              gl.STATIC_DRAW);
const colorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colorData),
              gl.STATIC_DRAW);
```

- Attribute Pointers

```
const positionLoc = gl.getAttribLocation(program, 'position');
gl.enableVertexAttribArray(positionLoc);
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.vertexAttribPointer(positionLoc, 3, gl.FLOAT, false, 0, 0);
const colorLoc = gl.getAttribLocation(program, 'color');
gl.enableVertexAttribArray(colorLoc);
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
gl.vertexAttribPointer(colorLoc, 3, gl.FLOAT, false, 0, 0);
```

WebGL Buffer Objects



Cube1.html

- Includes the code for vertex and fragment shaders

```
<html>
    <head>
        <title>COSC363:Cube1</title>
        <!--  VERTEX SHADER -->
        <script id="vertexShader"  type="notjs">
            precision mediump float;
            attribute vec3 position;
            attribute vec3 color;
            uniform mat4 mvpMatrix;
            varying vec3 oColor;
            void main() {
               oColor = color;
               gl_Position = mvpMatrix * vec4(position, 1);
            }
        </script>
        <!-- FRAGMENT SHADER -->
        <script id="fragmentShader" type="notjs">
            precision mediump float;
            varying vec3 oColor;
            void main() {
               gl_FragColor = vec4(oColor, 1);
            }
        </script>
        <script src="js/gl-matrix-min.js" defer></script>
        <script src="js/Cube1.js" defer></script>
    </head>

    <body>
        <canvas id="glCanvas" width = "500" height = "500"></canvas>
    </body>
</html>
```

Transformations Using glMatrix

- Identity Matrix

```
const idMatrix = mat4.create();
```

- The first two parameters of transformation functions must be previously allocated matrix variables (outMatrix, inMatrix).   Result: outMatrix = inMatrix * M

```
mat4.rotateY(outMatrix, inMatrix, angle*Math.PI/180.0);
mat4.translate(outMatrix, inMatrix, [2, 5, 1]);
mat4.scale(outMatrix, inMatrix, [0.5, 1, 0.5]);
```

- Example:

```
const matrix = mat4.create();
mat4.rotateY(matrix, matrix, angle*Math.PI/180.0);  //m = I*R
mat4.translate(matrix, matrix, [2, 5, 1]);    //m = m * T = R * T
mat4.scale(matrix, matrix, [0.5, 1, 0.5]); //m = m * S = R*T*S
```

Scale transformation followed by translation followed by rotation

View and Projection Using glMatrix

- View Matrix

```
const viewMatrix = mat4.create();
mat4.lookAt(viewMatrix, [0, 0, 20], [0,0,0], [0,1,0]);
```

- Projection Matrix

```
projMatrix = mat4.create();
mat4.perspective(projMatrix, 75 * Math.PI/180, 1, 10, 100);
```

- Model-View-Projection Matrix

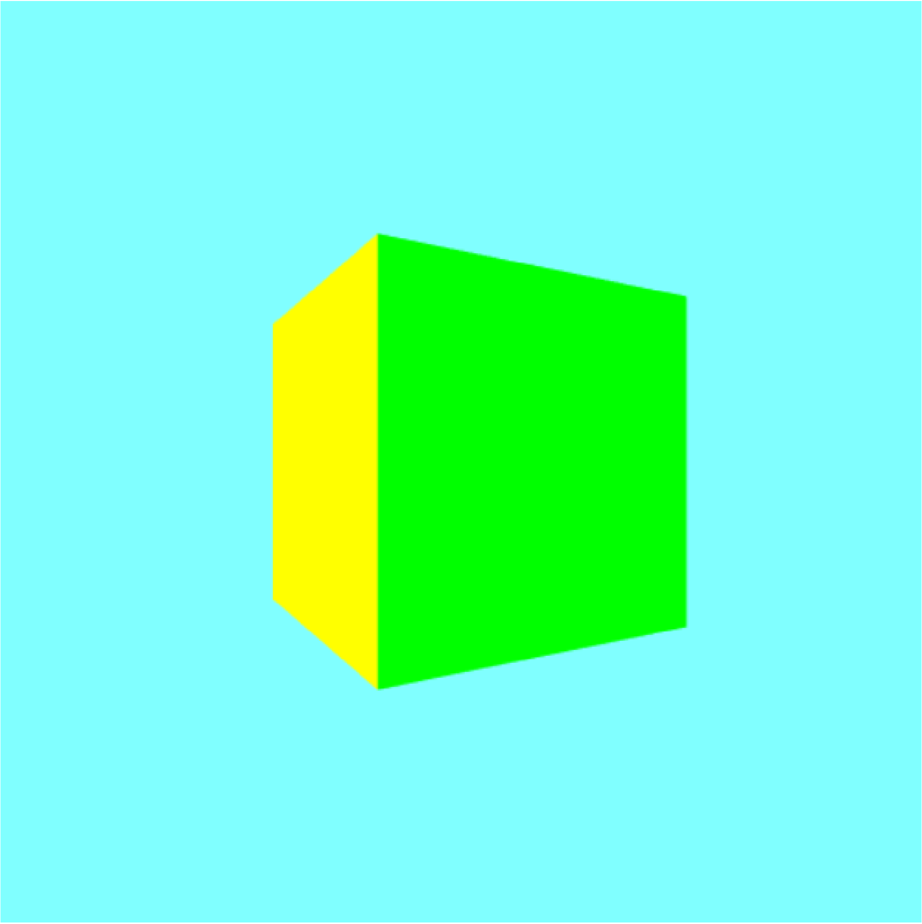
```
const mvMatrix = mat4.create();
mat4.multiply(mvMatrix, viewMatrix, matrix);

const mvpMatrix = mat4.create();
mat4.multiply(mvpMatrix, projMatrix, mvMatrix);
gl.uniformMatrix4fv(mvpMatrixLoc, false, mvpMatrix);
```

Cube1.js

- Contains code that has a structure similar to an OpenGL-4 program, to define VAO, VBOs and display commands.

Output:



Cube1Anim.html

- Uses an HTML DOM element button to execute a JavaScript when a button is clicked.
- The button is used to start and stop the cube's rotation.

```
<html>
    <head>
        <title>COSC363:Cube1Anim</title>

        <!--  VERTEX SHADER -->
        <script id="vertexShader"  type="notjs">
            precision mediump float;
       ...

       ...
      <script src="js/gl-matrix-min.js" defer></script>
      <script src="js/Cube1Anim.js" defer></script>

  </head>

  <body>
      <canvas id="glCanvas" width = "500" height = "500"></canvas>
      <p><button id = "demo" onclick="myButtonListener()">Start Animation</button></p>
  </body>
</html>
```

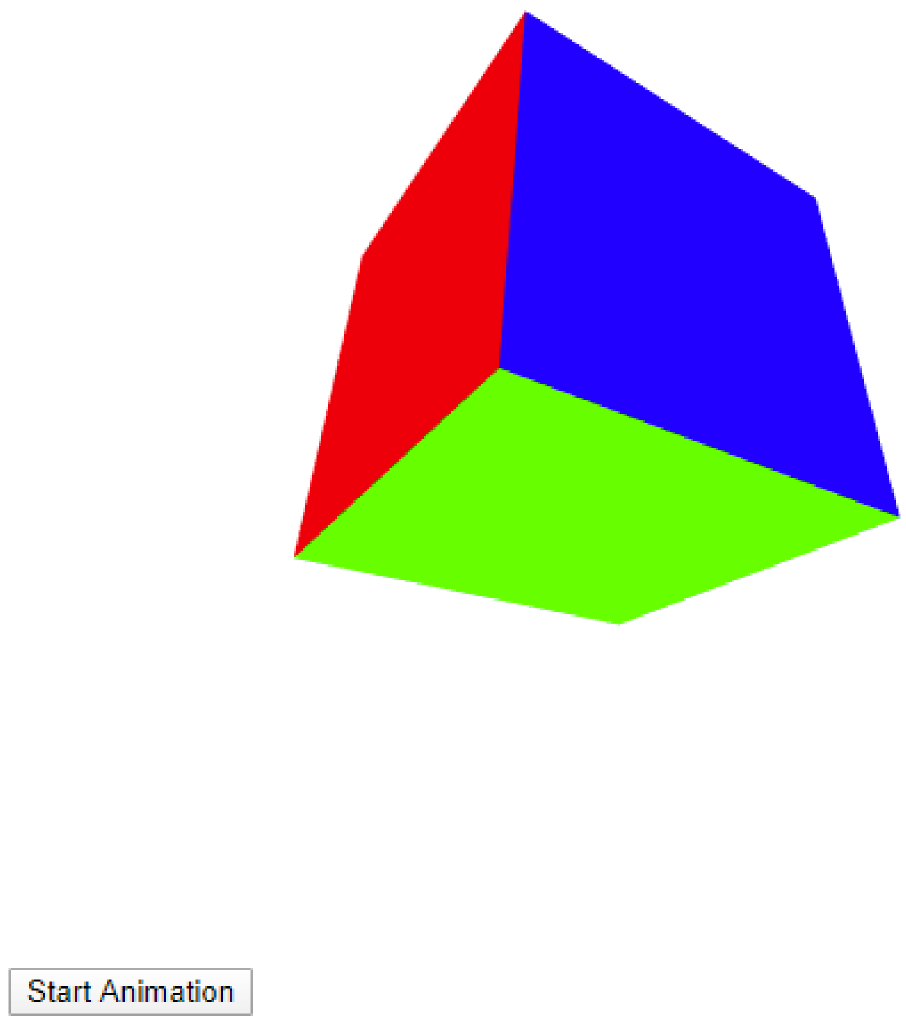
Cube1Anim.js

- Includes a button onClick event callback function to start and stop animation
- Includes JavaScript timer event callback for continuously rotating the cube.

```
function updateParams() {
     angle +=5;
     if(angle > 360) angle = 0;
     display();
}

function myButtonListener(){
   if(flag){
     flag = false;
     document.getElementById("demo").innerHTML = "Start Animation";
     clearInterval(timerVar);
   }
   else{
     flag = true;
     document.getElementById("demo").innerHTML = "Stop Animation";
     timerVar = setInterval(updateParams, 100);
   }
}
```

Cube1Anim: Output



Dolphin.html

- Uses a JavaScript file containing the model definition in OBJ format, and an OBJ file parser.
- The shaders are stored in the JavaScript file

```
<!DOCTYPE html> <html lang="en">
    <head>
    <meta http-equiv="Access-Control-Allow-Origin" content="*" />
        <title>COSC363:Dolphin</title>

        <script src="js/gl-matrix-min.js" defer></script>
        <script src="js/DolphinObj.js" defer></script>
        <script src="js/objparser.js" defer></script>
        <script src="js/Dolphin.js" defer></script>

    </head>

    <body>

        <canvas id="glCanvas" width = "600" height = "600"></canvas>

    </body>
</html>
```

Dolphin.js

- Defines vertex and fragment shaders as text blocks.
- The model data is also defined in OBJ format as a text block in DolphinObj.js. Uses a OBJ parser to get vertex and normal data from this model data.

Model Data (DolphinObj.js)

```
var dolphinData = `
# 1226 vertices,  2416 triangles
# All groups merged
# NOTE: Ignore normal indices when

v -0.02677 2.79412 -0.61210
v -0.09954 2.76945 -0.68876
v -0.11262 2.54082 -0.28951
v -0.03985 2.56549 -0.21285
v 0.00769 2.96169 -1.83568
v 0.00769 2.89151 -2.01892
v -0.00410 2.89151 -2.01892
v -0.00410 2.96169 -1.83568
```

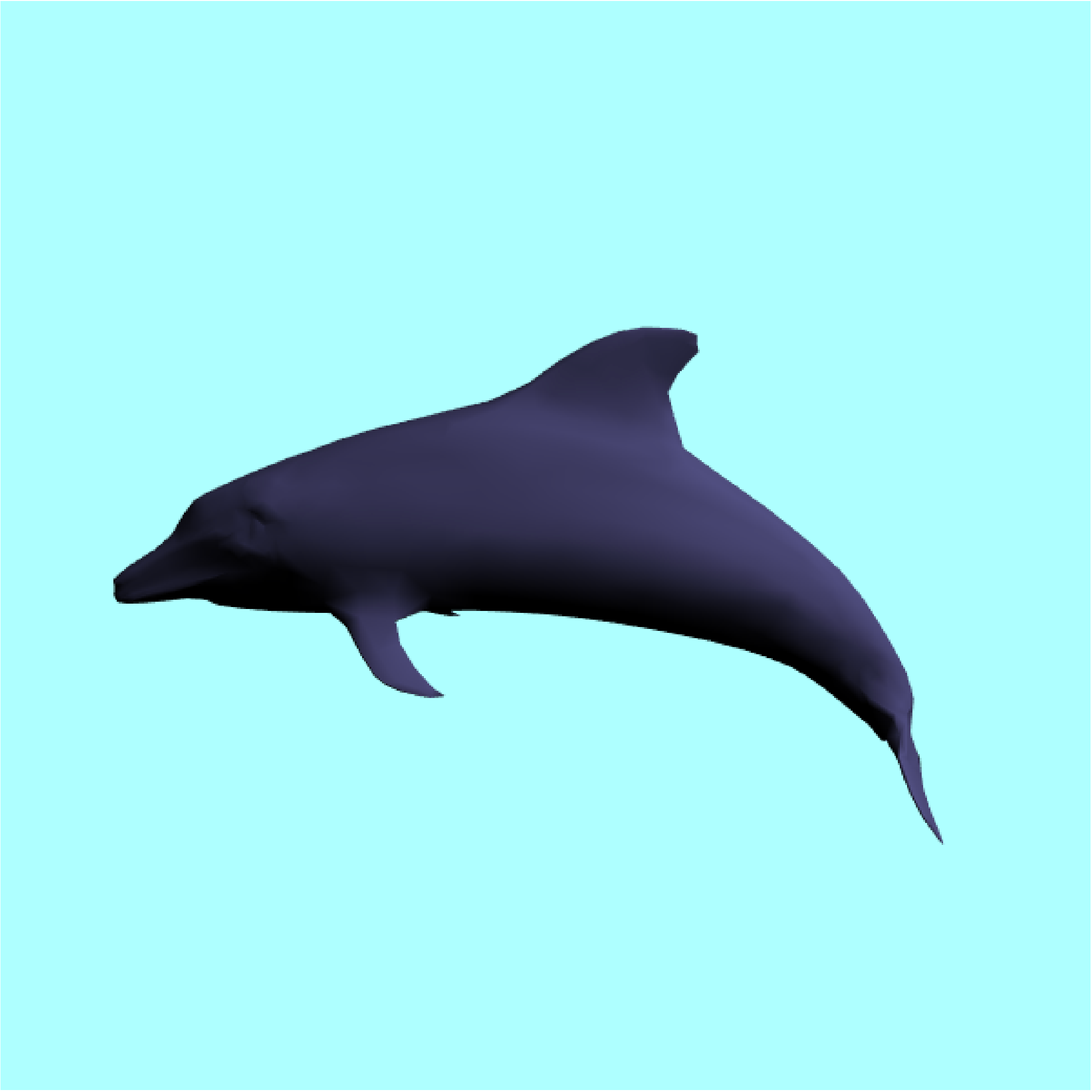
```
var vsSource = `#version 300 es
    precision mediump float;
    in vec3 position;
    in vec3 normal;
    uniform mat4 mvpMatrix;
    uniform mat4 norMatrix;
    out float diffuse;

    const vec3 lightDirection = normalize(vec3(1.0, 1.0, 1.0));
    void main() {
     vec3 norm = (norMatrix * vec4(normal, 0) ).xyz;
     norm = normalize(norm);
     diffuse = max(0.0, dot(norm, lightDirection));
     gl_Position = mvpMatrix * vec4(position, 1);
    }
  `;

var fsSource = `#version 300 es
    precision mediump float;
    uniform vec3 matColor;
    in float diffuse;
    out vec4 fragColor;

    void main() {
       vec3 col = diffuse * matColor/255.0;
       fragColor = vec4(col, 1);
    }
  `;
```

Dolphin: Output



Higher Level WebGL Frameworks

- High level WebGL libraries allow rapid 3D development
- They provide high level functions that encapsulate the complexity of raw WebGL development
- Some of them even have built-in game engines.

Tiny WebGL: http://twgljs.org/

Three.js: https://threejs.org/

sceneJS: http://scenejs.org/ (A 3D Visualization Engine)

playCanvas: https://playcanvas.com/ (WebGL Game Engine)