

COMP 370 Individual Project Part 2: Final Version

Thomas Williamson

id: 588206

2021/12/07

Abstract:

Interactive webgame the goal of the game is dodge the incoming blocks of “space debris” the high score and score are present at the top of the screen, there where little to no changes in from the original

Setup:

In order to load you must access the assignment7.html through a server, suggested application serverz

Controls:

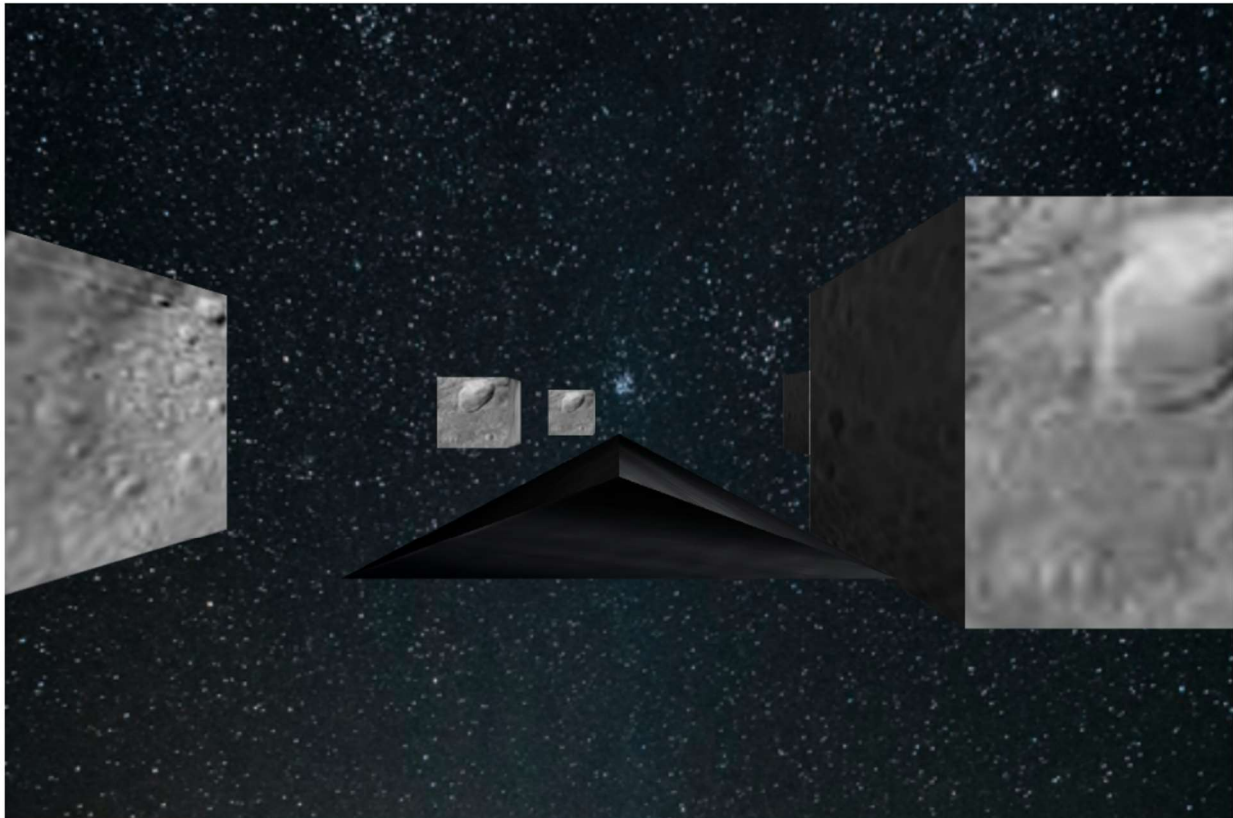
to move left and right either use A and D or the left and right arrow keys, to speed up and slow down press W and S or the up and down arrow keys.

Take away:

It was quite a lot easier than I anticipated. In the process of the development, I learned how to implement multiple different shapes I made use of instancing as well as unseen surfaces in order to put the background.

Gameplay:

score: 1762 | highScore: 0



impact:

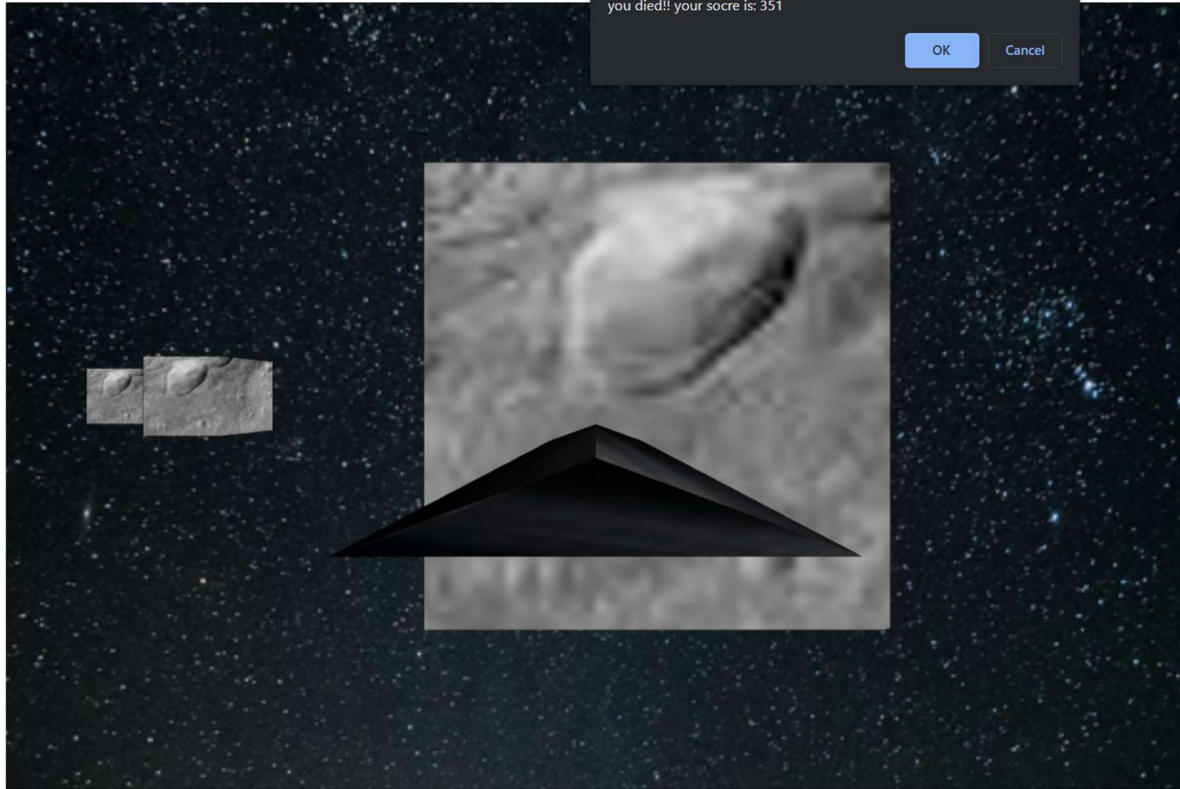
score: 351 | highScore: 0

localhost:8080 says

you died!! your socre is: 351

OK

Cancel



high Score

score: 194 | highScore: 1929



code:

```

1  <!--
2  COMP 370 Individual Project Part 2: Final Version: dodge cube
3  Thomas Williamson
4  id: 588206
5  2021/12/01
6  -->
7  <!DOCTYPE html>
8  <html>
9  <head>
10 |   <title>Texture Mapping</title>
11 </head>
12 <body>
13 <p id="score"></p>
14 <canvas id="gl-canvas" width="1080" height="720"> </canvas>
15 <img id = "texImage" src = "block.png" hidden></img>
16 <img id = "playerImage" src = "player.png" hidden></img>
17
18
19 <script id="vertex-shader" type="x-shader/x-vertex">
20 #version 300 es
21
22 in vec4 aPosition;
23 in vec2 aTexCoord;
24 in vec3 aNormal;
25 in vec4 aColor;
26
27 out vec4 vColor;
28 out vec2 vTexCoord;
29
30 uniform mat4 uModelViewMatrix;
31 uniform mat4 uProjectionMatrix;
32 uniform vec4 uAmbientProduct, uDiffuseProduct, uSpecularProduct;
33 uniform vec4 uLightPosition;
34 uniform float uShininess;
35
36 void main()
37 {
38     vec3 pos = aPosition.xyz;
39
40     //light position
41     vec3 light = uLightPosition.xyz;
42
43     vec3 L = normalize(light - pos);
44     vec3 E = normalize(-pos);
45     vec3 H = normalize(L + E);
46     vec4 NN = vec4(aNormal,0);
47
48     // Transform vertex normal into eye coordinates
49     vec3 N = normalize((uModelViewMatrix*NN).xyz);
50
51     // Compute terms in the illumination equation
52     vec4 ambient = uAmbientProduct;
53
54     float Kd = max(dot(L, N), 0.0);

```

```

55     vec4 diffuse = Kd * uDiffuseProduct;
56
57     float Ks = pow( max(dot(N, H), 0.0), uShininess );
58     vec4 specular = Ks * uSpecularProduct;
59
60     if( dot(L, N) < 0.0 ) {
61         specular = vec4(0.0, 0.0, 0.0, 1.0);
62     }
63     gl_Position = uProjectionMatrix * uModelViewMatrix * aPosition;
64     vTexCoord = aTexCoord;
65     vColor = ambient + diffuse + specular + aColor;
66     vColor.a = 1.0;
67 }
68 </script>
69
70 <script id="fragment-shader" type="x-shader/x-vertex">
71 #version 300 es
72
73 precision mediump float;
74 in vec4 vColor;
75 in vec2 vTexCoord;
76
77 out vec4 fColor;
78
79 uniform sampler2D uTextureMap;
80 uniform vec4 uDiffuseProduct;
81
82 void main()
83 {
84     vec4 T = texture(uTextureMap, vTexCoord);
85     fColor = T * vColor;
86 }
87 </script>
88
89 <script src="../../Common/initShaders.js"></script>
90 <script src="../../Common/MV.js"></script>
91 <script src="Assignment7.js"></script>
92 </body>
93 </html>
94

```



```

1  /*<!--
2  COMP 370 Individual Project Part 2: Final Version :dodge cube
3  Thomas Williamson
4  id: 588206
5  2021/12/07
6  -->*/
7  "use strict";
8  /*global variable defined*/
9  var canvas;
10 var gl;
11 var numPositions = 36;
12 //var texSize = 64;
13 var program;
14 var programP;
15 var alive = true
16 var speed = .02
17 var positionsArray = [];
18 var score =0;
19 var hiScore =0;
20 var camAtx = 0
21
22 ▼ var vertices = [
23     vec3(-0.5, -0.5, 0.5),
24     vec3(-0.5, 0.5, 0.5),
25     vec3(0.5, 0.5, 0.5),
26     vec3(0.5, -0.5, 0.5),
27     vec3(-0.5, -0.5, -0.5),
28     vec3(-0.5, 0.5, -0.5),
29     vec3(0.5, 0.5, -0.5),
30     vec3(0.5, -0.5, -0.5)];
31 ▼ var texCoordsArray = new Float32Array([
32     // select the top left image
33     0 , 0,
34     0 , 0.5,
35     0.25, 0.5,
36     0 , 0 ,
37     0.25, 0.5,
38     0.25, 0 ,
39     // select the top middle image
40     0.25, 0 ,
41     0.25, 0.5,
42     0.5 , 0.5,
43     0.25, 0 ,
44     0.5 , 0.5,
45     0.5 , 0 ,
46     // select to top right image
47     0.5 , 0.5,
48     0.75, 0.5,
49     0.75, 0 ,
50     0.5 , 0.5,
51     0.75, 0 ,
52     0.5, 0,
53

```

```

54 // select the bottom left image
55 0.25 , 0.5 ,
56 0 , 0.5,
57 0 , 1 ,
58 0.25 , 0.5 ,
59 0 , 1,
60 0.25 , 1 ,
61 // select the bottom middle image
62 0.5, 1 ,
63 0.5 , 0.5,
64 0.25, 0.5,
65 0.5 , 1 ,
66 0.25, 0.5,
67 0.25, 1 ,
68 // select the bottom right image
69
70 0.5 , 0.5,
71 0.5 , 1 ,
72 0.75, 1 ,
73 0.5 , 0.5,
74 0.75 , 1 ,
75 0.75, 0.5,
76 ]);
77 ▼ var normalsArray = [
78 //front
79 vec3(0.0, 0.0, 1.0),
80 vec3(0.0, 0.0, 1.0),
81 vec3(0.0, 0.0, 1.0),
82 vec3(0.0, 0.0, 1.0),
83 vec3(0.0, 0.0, 1.0),
84 vec3(0.0, 0.0, 1.0),
85
86 //right
87 vec3(1.0, 0.0, 0.0),
88 vec3(1.0, 0.0, 0.0),
89 vec3(1.0, 0.0, 0.0),
90 vec3(1.0, 0.0, 0.0),
91 vec3(1.0, 0.0, 0.0),
92 vec3(1.0, 0.0, 0.0),
93 // Bottom
94
95 vec3(0.0, -1, 0.0),
96 vec3(0.0, -1, 0.0),
97 vec3(0.0, -1, 0.0),
98 vec3(0.0, -1, 0.0),
99 vec3(0.0, -1, 0.0),
100 vec3(0.0, -1, 0.0),
101 //top
102 vec3(0.0, 1, 0.0),
103 vec3(0.0, 1, 0.0),
104 vec3(0.0, 1, 0.0),
105 vec3(0.0, 1, 0.0),
106 vec3(0.0, 1, 0.0),

```



```

107     vec3(0.0, 1, 0.0),
108     //back
109
110     vec3(0.0, 0.0, 1),
111     vec3(0.0, 0.0, 1),
112     vec3(0.0, 0.0, 1),
113     vec3(0.0, 0.0, 1),
114     vec3(0.0, 0.0, 1),
115     vec3(0.0, 0.0, 1),
116
117     //left
118
119     vec3(-1.0, 0.0, 0.0),
120     vec3(-1.0, 0.0, 0.0),
121     vec3(-1.0, 0.0, 0.0),
122     vec3(-1.0, 0.0, 0.0),
123     vec3(-1.0, 0.0, 0.0),
124     vec3(-1.0, 0.0, 0.0),
125 ];
126 var texture;
127 var xAxis = 0;
128 var yAxis = 1;
129 var zAxis = 2;
130 var axis = xAxis;
131 var theta = vec3(0.0, 0.0, 0.0);
132 var flag = false;
133 var modelViewMatrixLoc;
134 var projectionMatrixLoc;
135 var textureLocation;
136 var viewMatrix;
137 var time = 0;
138 var projectionMatrix;
139 var modelViewMatrix;
140 function getRand(min, max) {
141     return Math.random() * (max - min) + min;
142 }
143
144 var lightPosition = vec4(0.0, 1.0, 2.0, 0.0);
145 var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0);
146 var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
147 var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0);
148 var materialAmbient = vec4(1.0, 1.0, 1.0, 1.0 );
149 var materialDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
150 var materialSpecular = vec4(1.0, 1.0, 1.0, 1.0 );
151 var materialShininess = 100;
152
153 var cubes =[];
154
155 function quad(a, b, c, d) {
156     positionsArray.push(vertices[a]);
157
158     positionsArray.push(vertices[b]);
159

```

```

160     positionsArray.push(vertices[c]);
161
162
163     positionsArray.push(vertices[a]);
164
165     positionsArray.push(vertices[c]);
166
167     positionsArray.push(vertices[d]);
168 }
169
170
171 function colorCube()
172 {
173     quad(1, 0, 3, 2);
174     quad(2, 3, 7, 6);
175     quad(3, 0, 4, 7);
176     quad(6, 5, 1, 2);
177     quad(4, 5, 6, 7);
178     quad(5, 4, 0, 1);
179 }
180
181
182
183 //Execute a JavaScript immediately after a page has been loaded
184 window.onload = function init(){
185
186     //Initialize the canvas by document.getElementById method
187     canvas = document.getElementById("gl-canvas");
188     gl = canvas.getContext('webgl2');
189     if (!gl){
190         alert("WebGL 2.0 isn't available");
191     }
192     //set the viewport and canvas background color
193     gl.viewport(0, 0, canvas.width, canvas.height);
194     gl.clearColor(1.0, 1.0, 1.0, 1);
195
196     gl.enable(gl.DEPTH_TEST);
197
198     //Load shaders and initialize attribute buffers
199     program = initShaders(gl,"vertex-shader", "fragment-shader");
200     programP = initShaders(gl,"vertex-shader", "fragment-shader");
201
202     gl.useProgram(program);
203
204     colorCube();
205     //Create buffer for normals
206     cubeBuffer();
207
208
209     //set the perspective projection
210     var fieldOfView = 75; //Change the value
211     var aspect = canvas.width/canvas.height;
212     var zNear = .01; //Change the value

```

```

213     var zFar = 30; //Change the value
214     projectionMatrix = perspective(fieldOfView, aspect, zNear, zFar);
215
216     //set event to the buttons
217     window.addEventListener('keydown', this.checkKey);
218
219     render();
220 }
221
222 function cubeBuffer(){
223     gl.useProgram(program);
224     var nBuffer = gl.createBuffer();
225     gl.bindBuffer(gl.ARRAY_BUFFER, nBuffer);
226     gl.bufferData(gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW);
227     var normalLoc = gl.getAttribLocation(program, "aNormal");
228     gl.vertexAttribPointer(normalLoc, 3, gl.FLOAT, false, 0, 0);
229     gl.enableVertexAttribArray(normalLoc);
230
231     //Create buffer for vertex
232     var vBuffer = gl.createBuffer();
233     gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
234     gl.bufferData(gl.ARRAY_BUFFER, flatten(positionsArray), gl.STATIC_DRAW);
235     var positionLoc = gl.getAttribLocation(program, "aPosition");
236     gl.vertexAttribPointer(positionLoc, 3, gl.FLOAT, false, 0, 0);
237     gl.enableVertexAttribArray(positionLoc);
238
239     //Create buffer for texture coordinate
240     var tBuffer = gl.createBuffer();
241     gl.bindBuffer(gl.ARRAY_BUFFER, tBuffer);
242     gl.bufferData(gl.ARRAY_BUFFER, texCoordsArray, gl.STATIC_DRAW);
243     var texCoordLoc = gl.getAttribLocation(program, "aTexCoord");
244     gl.vertexAttribPointer(texCoordLoc, 2, gl.FLOAT, false, 0, 0);
245     gl.enableVertexAttribArray(texCoordLoc);
246     //load an image
247     var image = document.getElementById("texImage");
248     configureTexture(image);
249     modelViewMatrixLoc = gl.getUniformLocation(program, "uModelViewMatrix");
250     projectionMatrixLoc = gl.getUniformLocation(program, "uProjectionMatrix");
251 }
252 var positionsArrayP = [
253     vec3(-0.25, -0.25, 0.26),
254     vec3(0, -0.1, 0.25),
255     vec3(0, -0.1, -0.25),
256
257     vec3(0.25, -0.25, 0.26),
258     vec3(0, -0.1, 0.25),
259     vec3(0, -0.1, -0.25),
260
261     vec3(0, -0.1, 0.25),
262     vec3(0.25, -0.25, 0.26),
263     vec3(-0.25, -0.25, 0.26)
264 ]
265 var normalsArrayP = [

```

```

266     vec3(-1, 1, 0),
267     vec3(-1, 1, 0),
268     vec3(-1, 1, 0),
269
270     vec3(1, 1, 0),
271     vec3(1, 1, 0),
272     vec3(1, 1, 0),
273
274     vec3(0, 0, 1),
275     vec3(0, 0, 1),
276     vec3(0, 0, 1)
277 ]
278
279
280
281
282
283 var texCoordsArrayP = new Float32Array([
284     0,1,
285     .5,0,
286     .5,1,
287
288     .5,1,
289     .5,0,
290     1,1,
291
292     1,1,
293     .5,.5,
294     0,1
295 ]);
296 var modelViewMatrixPLoc
297 var projectionMatrixPLoc;
298 //load player
299 function playerbuffer(){
300     gl.useProgram(programP);
301
302     colorCube()
303     //Create buffer for normals
304     var nBufferP = gl.createBuffer();
305     gl.bindBuffer(gl.ARRAY_BUFFER, nBufferP);
306     gl.bufferData(gl.ARRAY_BUFFER, flatten(normalsArrayP), gl.STATIC_DRAW);
307     var normalLocP = gl.getAttribLocation(programP, "aNormal");
308     gl.vertexAttribPointer(normalLocP, 3, gl.FLOAT, false, 0, 0);
309     gl.enableVertexAttribArray(normalLocP);
310
311     //Create buffer for vertex
312     var vBufferP = gl.createBuffer();
313     gl.bindBuffer(gl.ARRAY_BUFFER, vBufferP);
314     gl.bufferData(gl.ARRAY_BUFFER, flatten(positionsArrayP), gl.STATIC_DRAW);
315     var positionLocP = gl.getAttribLocation(programP, "aPosition");
316     gl.vertexAttribPointer(positionLocP, 3, gl.FLOAT, false, 0, 0);
317     gl.enableVertexAttribArray(positionLocP);
318

```



```

319 //Create buffer for texture coordinate
320 var tBufferP = gl.createBuffer();
321 gl.bindBuffer(gl.ARRAY_BUFFER, tBufferP);
322 gl.bufferData(gl.ARRAY_BUFFER, texCoordsArrayP, gl.STATIC_DRAW);
323 var texCoordLocP = gl.getAttribLocation(programP, "aTexCoord");
324 gl.vertexAttribPointer(texCoordLocP, 2, gl.FLOAT, false, 0, 0);
325 gl.enableVertexAttribArray(texCoordLocP);
326
327 //load an image
328 var imageP = document.getElementById("playerImage");
329 configureTexture(imageP);
330 modelViewMatrixPLoc = gl.getUniformLocation(programP, "uModelViewMatrix");
331 projectionMatrixPLoc = gl.getUniformLocation(programP, "uProjectionMatrix");
332 }
333
334 //function for setting the texture
335 ▼ function configureTexture(image){
336     texture = gl.createTexture();
337     gl.bindTexture(gl.TEXTURE_2D, texture);
338     gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
339         new Uint8Array([0, 0, 255, 255]));
340     gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
341     //generate the Mipmap
342     gl.generateMipmap(gl.TEXTURE_2D);
343     gl.bindTexture(gl.TEXTURE_2D, texture);
344     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);
345     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
346 }
347
348 var tiltTime = 0;
349 //render function
350 ▼ function render(){
351     cubeBuffer();
352     //set the model-view matrix
353     var cameraPosition = vec3(camAtx, 0, 1);
354     var up = vec3(0.0, 1.0, 0.0);
355     var target = vec3(camAtx, 0.0, 0.0);
356     modelViewMatrix = lookAt(cameraPosition, target, up);
357
358     gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
359     if(flag){
360         theta[axis] += 0.3; //Change the value
361     }
362     //rotating the light
363     lightPosition[0] = 5.5 ;
364     lightPosition[2] = 5.5;
365     time += .5; //Change the value
366
367     //generate two cubes, one is closer to the viewer and the other farther from the viewer
368 ▼ for(var index = 0; index < 10; index++){
369 ▼     if( cubes[index] == undefined){
370         cubes.push(translate(0,0,0));
371 ▼     if(!index > 0){

```

```

368     for(var index = 0; index < 10; index++){
369         if( cubes[index] == undefined){
370             cubes.push(translate(0,0,0));
371             if(!index > 0){
372                 //console.log('sky')
373                 cubes[index] = translate(camAtx+getRand(-10,10), 0, getRand(-10, -20))
374             }else{cubes[index] = translate(camAtx+getRand(-15,15), 0, getRand(-10, -20));}
375             //console.log(cubes[index]);
376         }
377     }
378     gl.uniform4fv( gl.getUniformLocation(program, "uLightPosition"), lightPosition);
379
380     if(index == 0){
381
382         viewMatrix = mult(modelViewMatrix, translate(cameraPosition[0],0,1));
383
384         viewMatrix = mult(scale(2.5,2,2),viewMatrix);
385     }else{
386         //console.log(cubes[index]);
387
388         if(cubes[index][2][3] >= 1){
389             cubes[index] = translate(camAtx+getRand(-5,5), 0, getRand(-10, -30))
390         }else{
391             cubes[index] = mult(cubes[index], translate(0,0,speed));
392         }
393         //console.log('cubes[index]')
394         viewMatrix = mult(modelViewMatrix, cubes[index]);
395         viewMatrix = mult(scale(.05,.05,.05),viewMatrix);
396         if(cubes[index][0][3] +1 >= camAtx && cubes[index][0][3] -1 <= camAtx && cubes
397             console.log('aaaaa!');
398             if (hiScore < score){hiScore = score;}
399
400             window.confirm("you died!! your score is: "+ Math.floor(score));
401             score = 0;
402             speed = 0.2;
403             cubes = []
404         }
405     }
406 };
407 // viewMatrix = mult(viewMatrix, rotate(theta[xAxis], vec3(1, 0, 0)));
408 // viewMatrix = mult(viewMatrix, rotate(theta[yAxis], vec3(0, 1, 0)));
409 // viewMatrix = mult(viewMatrix, rotate(theta[zAxis], vec3(0, 0, 1)));
410 gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(viewMatrix));
411 gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));
412
413 var diffuseProduct = mult(lightDiffuse, materialDiffuse);
414 gl.uniform4fv(gl.getUniformLocation(program, "uDiffuseProduct"), diffuseProduct);
415
416 var ambientProduct = mult(lightAmbient, materialAmbient);
417 gl.uniform4fv(gl.getUniformLocation(program, "uAmbientProduct"), ambientProduct);
418
419 var specularProduct = mult(lightSpecular, materialSpecular);
420 gl.uniform4fv(gl.getUniformLocation(program, "uSpecularProduct"), specularProduct);

```



```

421     gl.uniform1f(gl.getUniformLocation(program, "uShininess"), materialShininess);
422     gl.uniform1i(textureLocation, index);
423     gl.drawArrays(gl.TRIANGLES, 0, numPositions);
424 }
425
426
427
428
429 //generate player
430 playerbuffer();
431
432 gl.uniform4fv( gl.getUniformLocation(programP, "uLightPosition"), lightPosition);
433
434 viewMatrix = mult(modelViewMatrix, translate(cameraPosition[0],0,.5));
435 viewMatrix = mult(scale(.05,.03,.1),viewMatrix);
436 viewMatrix = mult(viewMatrix, rotate(theta[zAxis], vec3(0, 0, 1)));
437
438 if (tiltTime > 0){
439     tiltTime -= 1;
440 }else{
441     if(theta[zAxis] > 0){
442         theta[zAxis] -=1;
443     }
444     if(theta[zAxis] < 0){
445         theta[zAxis] +=1;
446     }
447 }
448 gl.uniformMatrix4fv(modelViewMatrixPLoc, false, flatten(viewMatrix));
449 gl.uniformMatrix4fv(projectionMatrixPLoc, false, flatten(projectionMatrix));
450
451 var diffuseProduct = mult(lightDiffuse, materialDiffuse);
452 gl.uniform4fv(gl.getUniformLocation(programP, "uDiffuseProduct"), diffuseProduct);
453
454 var ambientProduct = mult(lightAmbient, materialAmbient);
455 gl.uniform4fv(gl.getUniformLocation(programP, "uAmbientProduct"), ambientProduct);
456
457 var specularProduct = mult(lightSpecular, materialSpecular);
458 gl.uniform4fv(gl.getUniformLocation(programP, "uSpecularProduct"), specularProduct);
459
460 gl.uniform1f(gl.getUniformLocation(programP, "uShininess"), materialShininess);
461 //gl.uniform1i(textureLocationP, 0);
462 gl.drawArrays(gl.TRIANGLES, 0, 9);
463
464 score += 1*speed +1;
465 const element = document.getElementById("score");
466 element.innerHTML = "score: "+ String(Math.floor(score))+ " | highScore: "+String(Math.
467
468 requestAnimationFrame(render);
469 }
470
471 function checkKey(e){
472     if(alive == true){
473         //console.log(e.keyCode);

```

```
474     switch(e.keyCode){
475         case 68:
476         case 39:
477             camAtx += .05;
478             if(theta[zAxis] < 15){
479                 theta[zAxis] += 2
480             }
481             tiltTime = 2;
482             // console.log('a');
483             break
484         case 65:
485         case 37:
486             camAtx -= .05;
487             if(theta[zAxis] > -15){
488                 theta[zAxis] -= 2
489             }
490             tiltTime = 2;
491             break
492         case 87:
493         case 38:
494             speed += .01;
495             break
496         case 83:
497         case 40:
498             if(speed > .02){
499                 speed -= .01;
500             }
501
502             break
503     }
504 }
505 }
506
```