

# Blending

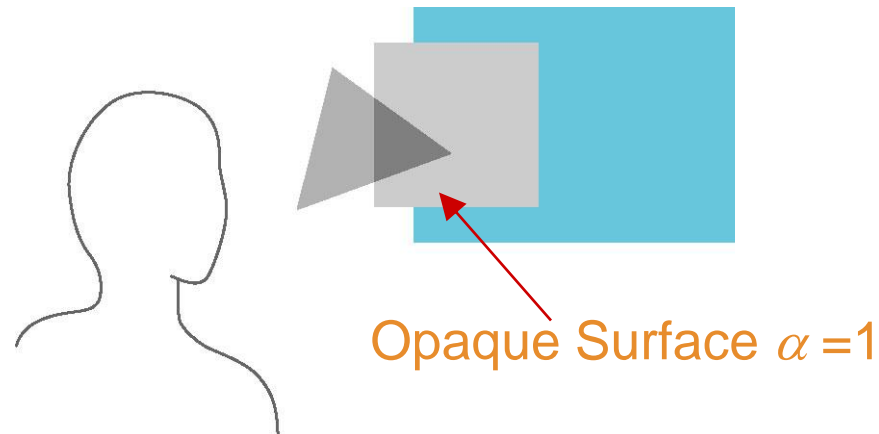
13<sup>TH</sup> WEEK, 2022



# Opacity and Transparency

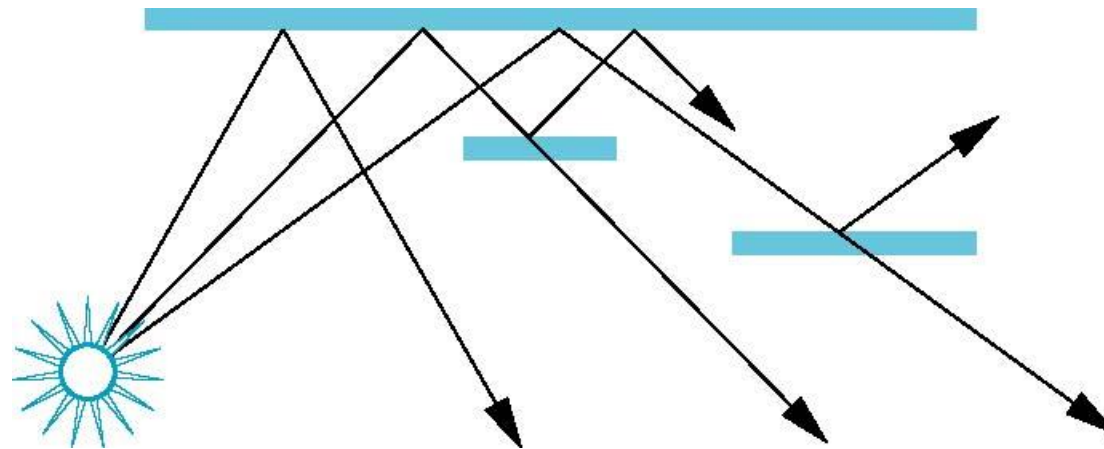
- Opaque surfaces permit no light to pass through
- Transparent surfaces permit all light to pass
- Translucent surfaces pass some light

$$\text{Translucency} = 1 - \text{Opacity}(\alpha)$$



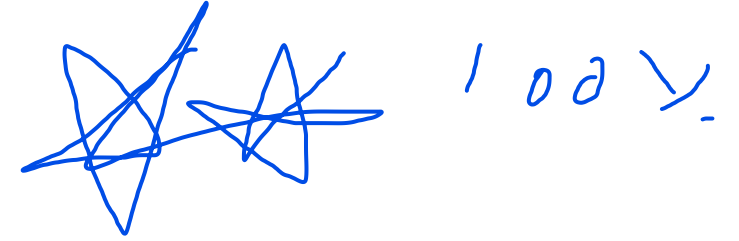
# Physical Models

- Dealing with translucency in a physically correct manner is difficult due to
  - The complexity of the internal interactions of light and matter
  - Using a pipeline renderer

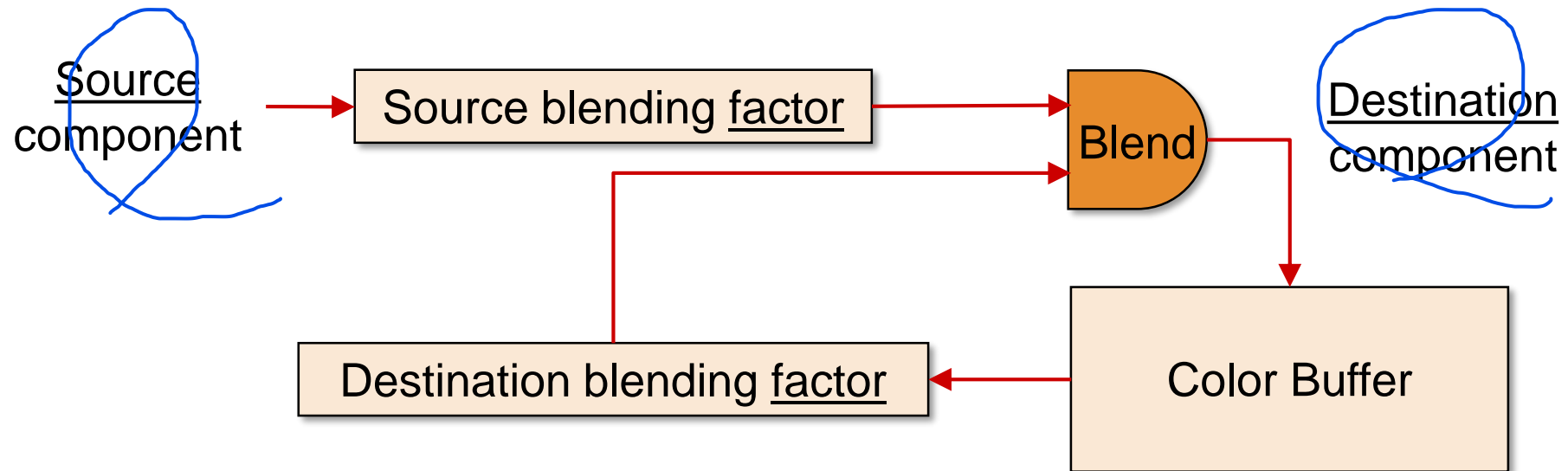


Scene with translucent objects

# Writing Model for Blending



- Use A component of RGBA (or  $\text{RGB}\alpha$ ) color to store opacity
- During rendering we can expand our writing model to use RGBA values



# Blending Equation

- We can define source and destination blending factors for each RGBA component

$$\mathbf{s} = [s_r, s_g, s_b, s_\alpha]$$

$$\mathbf{d} = [d_r, d_g, d_b, d_\alpha]$$

- Suppose that the source and destination colors are

$$\mathbf{b} = [b_r, b_g, b_b, b_\alpha]$$

$$\mathbf{c} = [c_r, c_g, c_b, c_\alpha]$$

- Blend as

$$\mathbf{c}' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_\alpha s_\alpha + c_\alpha d_\alpha]$$

# WebGL Blending and Compositing

- Must enable blending and pick source and destination factors

```
gl.enable( gl.BLEND );  
gl.blendFunc( source_factor, destination_factor );
```

- Only certain factors supported

- `gl.ZERO`, `gl.ONE`
- `gl.SRC_ALPHA`, `gl.ONE_MINUS_SRC_ALPHA`
- `gl.DST_ALPHA`, `gl.ONE_MINUS_DST_ALPHA`
- `gl.SRC_COLOR`, `gl.ONE_MINUS_SRC_COLOR`
- `gl.DST_COLOR`, `gl.ONE_MINUS_DST_COLOR`
- `gl.SRC_ALPHA_SATURATE`

# Example: Blending

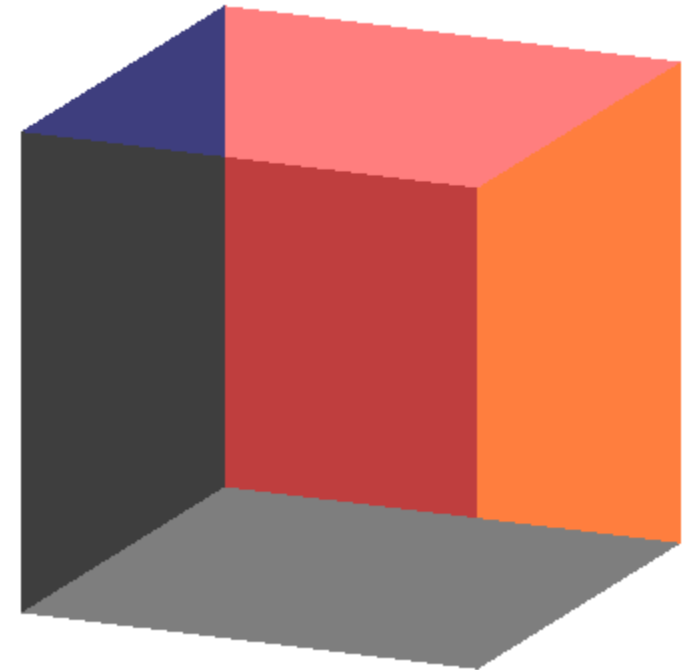
- Suppose that we start with the opaque background color  $(R_0, G_0, B_0, 1)$ 
  - This color becomes the initial destination color
- We now want to blend in a translucent polygon with color  $(R_1, G_1, B_1, \alpha_1)$
- Select **gl.SRC\_ALPHA** and **gl.ONE\_MINUS\_SRC\_ALPHA** as the source and destination blending factors

$$R'_1 = \alpha_1 R_1 + (1 - \alpha_1) R_0 \quad G'_1 = \alpha_1 G_1 + (1 - \alpha_1) G_0 \quad B'_1 = \alpha_1 B_1 + (1 - \alpha_1) B_0$$

- Note that this formula is correct if polygon is either opaque or transparent

# Order Dependency

- Is this image correct?
  - Probably not
  - Polygons are rendered in the order they pass down the pipeline
  - Blending functions are order dependent





# Opaque and Translucent Polygons

- Suppose that we have a group of polygons some of which are opaque and some translucent
  - How do we use hidden-surface removal?
  - Opaque polygons block all polygons behind them and affect the depth buffer
  - Translucent polygons should not affect depth buffer
    - Render with `gl.depthMask(false)` which makes depth buffer read-only
- Sort polygons first to remove order dependency

정렬

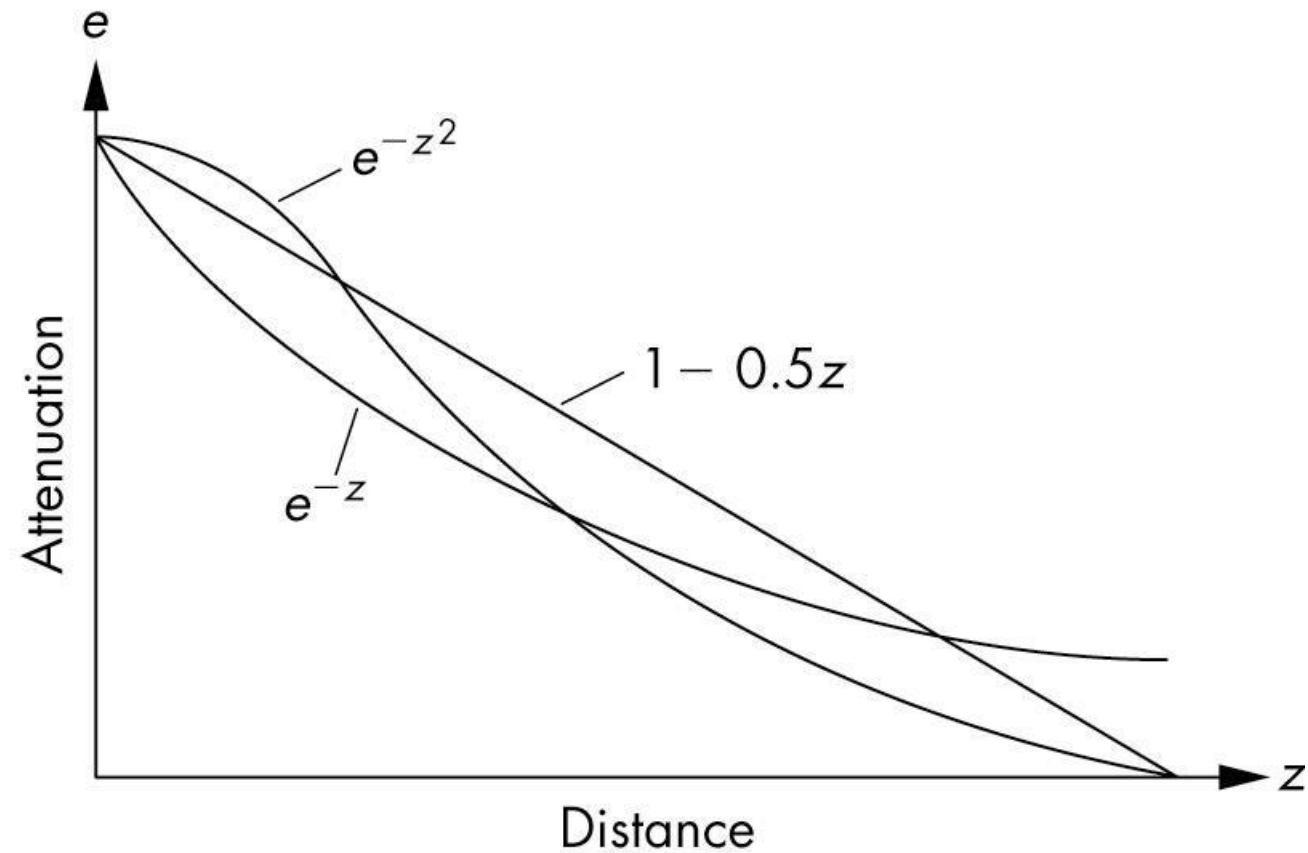
# Fog

- We can composite with a fixed color and have the blending factors depend on depth
  - Simulates a fog effect
- Blend source color  $C_s$  and fog color  $C_f$  by

$$C'_s = f C_s + (1-f) C_f$$

- $f$  is the fog factor
  - Exponential
  - Gaussian
  - Linear (depth cueing)

# Fog Functions

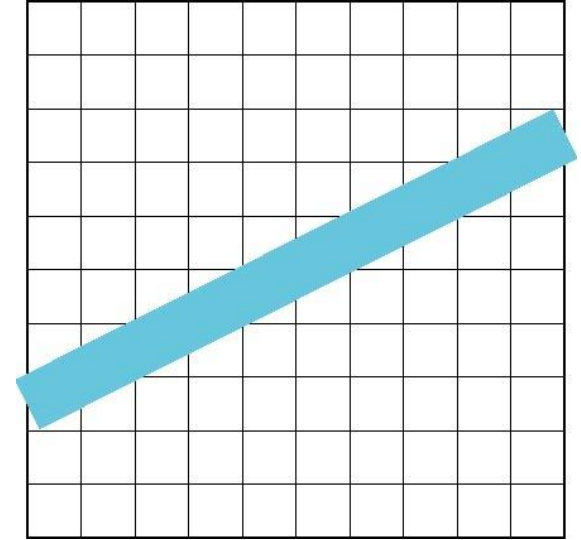


# Compositing and HTML

- In desktop OpenGL, the A component has no effect unless blending is enabled
- In WebGL, an A other than 1.0 has an effect because WebGL works with the HTML5 Canvas element
- $A = 0.5$  will cut the RGB values by  $\frac{1}{2}$  when the pixel is displayed
- Allows other applications to be blended into the canvas along with the graphics

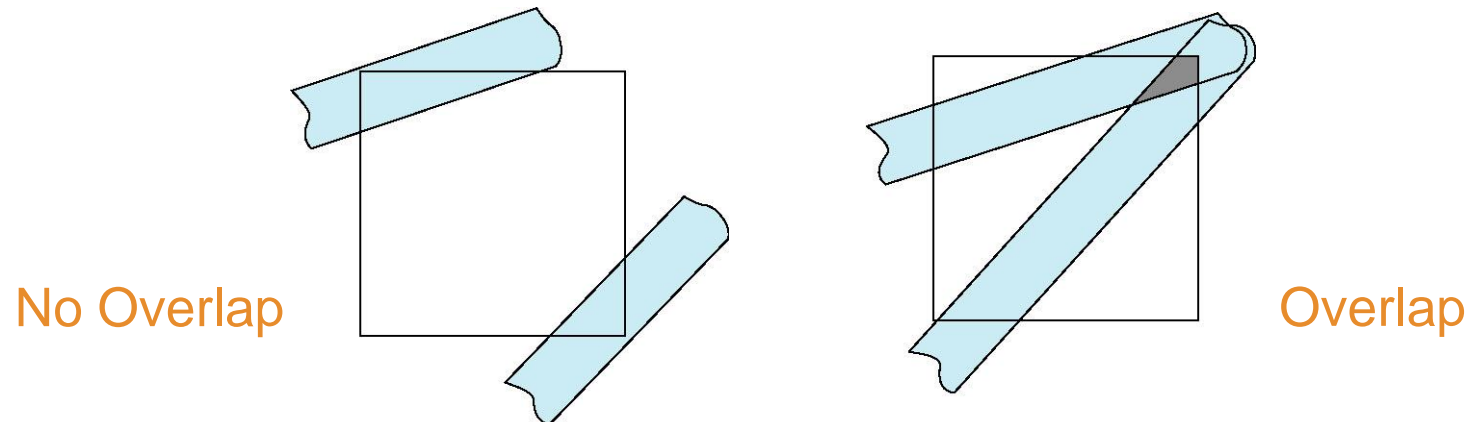
# Line Aliasing

- Ideal raster line is one pixel wide
- All line segments, other than vertical and horizontal segments, partially cover pixels
- Simple algorithms color only whole pixels
- Lead to the “jaggies” or aliasing
- Similar issue for polygons



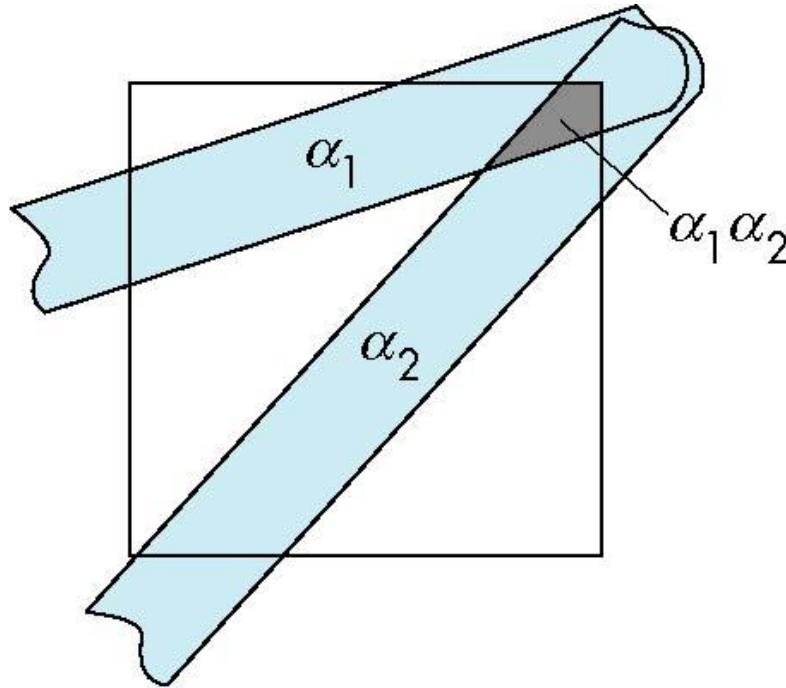
# Antialiasing

- Can try to color a pixel by adding a fraction of its color to the frame buffer
  - Fraction depends on percentage of pixel covered by fragment
    - Setting the alpha value for the corresponding pixel to be a number between 0 and 1 that is the amount of that pixel covered by the fragment
  - Fraction depends on whether there is overlap



# Area Averaging

- Use average area  $\alpha_1 + \alpha_2 - \alpha_1 \alpha_2$  as blending factor





# Example: Antialiasing



Without Antialiasing



Antialiasing



# OpenGL Antialiasing

- Not (yet) supported in WebGL
- Can enable separately for points, lines, or polygons

```
glEnable( GL_POINT_SMOOTH );  
glEnable( GL_LINE_SMOOTH );  
glEnable( GL_POLYGON_SMOOTH );
```

```
glEnable( GL_BLEND );  
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
```

- Note most hardware will automatically antialias

alphaCube.html x JS alphaCube.js

C: > Users > GVE > Desktop > CG > Week13 > <> alphaCube.html > html > head > script

1 <!DOCTYPE html>

2 <html>

3 <head>

4 <title>학번 이름 - Alpha Blending</title>

5 <script id="vertex-shader" type="x-shader/x-vertex">

6 attribute vec4 vPosition;

7 attribute vec4 vColor;

8

9 uniform mat4 modelMatrix, viewMatrix, projectionMatrix;

10

11 varying vec4 fColor;

12

13 void main() {

14 | gl\_Position = projectionMatrix \* viewMatrix \* modelMatrix \* vPosition;

15 |

16 | fColor = vColor;

17 | }

18 </script>

19

20 <script id="fragment-shader" type="x-shader/x-fragment">

21 precision mediump float;

22

23 varying vec4 fColor;

24

25 void main() {

26 | gl\_FragColor = fColor;

27 | gl\_FragColor.a = 1.0;

28 | }

29 </script>

30

31 <script type="text/javascript" src="../Common/webgl-utils.js"></script>

32 <script type="text/javascript" src="../Common/initShaders.js"></script>

33 <script type="text/javascript" src="../Common/MV.js"></script>

34 <script type="text/javascript" src="../trackball.js"></script>

35 <script type="text/javascript" src="alphaCube.js"></script>

Restricted Mode 0 0 Ln 35, Col 54 Spaces: 4 UTF-8 CRLF HTML

alphaCube.html x JS alphaCube.js

C: > Users > GVE > Desktop > CG > Week13 > <> alphaCube.html > html > head > script

```
30
31     <script type="text/javascript" src="../../Common/webgl-utils.js"></script>
32     <script type="text/javascript" src="../../Common/initShaders.js"></script>
33     <script type="text/javascript" src="../../Common/MV.js"></script>
34     <script type="text/javascript" src="../../trackball.js"></script>
35     <script type="text/javascript" src="alphaCube.js"></script>
36 </head>
37 <body>
38     <canvas id="gl-canvas" width="512" height="512">
39         |   Oops... your browser doesn't support the HTML5 canvas element!
40     </canvas>
41 </body>
42 </html>
```

alphaCube.html

alphaCube.html JS alphaCube.js X

C: &gt; Users &gt; GVE &gt; Desktop &gt; CG &gt; Week13 &gt; JS alphaCube.js &gt; ...

```
1  var gl;
2  var points = [];
3  var colors = [];
4
5  //var modelMatrix;
6  var modelMatrixLoc;
7
8  var trballMatrix = mat4(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
9  var numVertCubeTri;
10
11 window.onload = function init()
12 {
13     var canvas = document.getElementById("gl-canvas");
14
15     gl = WebGLUtils.setupWebGL(canvas);
16     if( !gl ) {
17         alert("WebGL isn't available!");
18     }
19
20     generateColorCube();
21
22     // virtual trackball
23     var trball = trackball(canvas.width, canvas.height);
24     var mouseDown = false;
25
26     canvas.addEventListener("mousedown", function (event) {
27         trball.start(event.clientX, event.clientY);
28
29         mouseDown = true;
30     });
31
32     canvas.addEventListener("mouseup", function (event) {
33         mouseDown = false;
34     });
35 }
```

```
1 // WebGL 1.0
2 // WebGL 1.0
3 // WebGL 1.0
4 // WebGL 1.0
5 // WebGL 1.0
6 // WebGL 1.0
7 // WebGL 1.0
8 // WebGL 1.0
9 // WebGL 1.0
10 // WebGL 1.0
11 // WebGL 1.0
12 // WebGL 1.0
13 // WebGL 1.0
14 // WebGL 1.0
15 // WebGL 1.0
16 // WebGL 1.0
17 // WebGL 1.0
18 // WebGL 1.0
19 // WebGL 1.0
20 // WebGL 1.0
21 // WebGL 1.0
22 // WebGL 1.0
23 // WebGL 1.0
24 // WebGL 1.0
25 // WebGL 1.0
26 // WebGL 1.0
27 // WebGL 1.0
28 // WebGL 1.0
29 // WebGL 1.0
30 // WebGL 1.0
31 // WebGL 1.0
32 // WebGL 1.0
33 // WebGL 1.0
34 // WebGL 1.0
35 // WebGL 1.0
36 // WebGL 1.0
37 // WebGL 1.0
38 // WebGL 1.0
39 // WebGL 1.0
40 // WebGL 1.0
41 // WebGL 1.0
42 // WebGL 1.0
43 // WebGL 1.0
44 // WebGL 1.0
45 // WebGL 1.0
46 // WebGL 1.0
47 // WebGL 1.0
48 // WebGL 1.0
49 // WebGL 1.0
50 // WebGL 1.0
51 // WebGL 1.0
52 // WebGL 1.0
53 // WebGL 1.0
54 // WebGL 1.0
55 // WebGL 1.0
56 // WebGL 1.0
57 // WebGL 1.0
58 // WebGL 1.0
59 // WebGL 1.0
60 // WebGL 1.0
61 // WebGL 1.0
62 // WebGL 1.0
63 // WebGL 1.0
64 // WebGL 1.0
65 // WebGL 1.0
66 // WebGL 1.0
67 // WebGL 1.0
68 // WebGL 1.0
69 // WebGL 1.0
70 // WebGL 1.0
71 // WebGL 1.0
72 // WebGL 1.0
73 // WebGL 1.0
74 // WebGL 1.0
75 // WebGL 1.0
76 // WebGL 1.0
77 // WebGL 1.0
78 // WebGL 1.0
79 // WebGL 1.0
80 // WebGL 1.0
81 // WebGL 1.0
82 // WebGL 1.0
83 // WebGL 1.0
84 // WebGL 1.0
85 // WebGL 1.0
86 // WebGL 1.0
87 // WebGL 1.0
88 // WebGL 1.0
89 // WebGL 1.0
90 // WebGL 1.0
91 // WebGL 1.0
92 // WebGL 1.0
93 // WebGL 1.0
94 // WebGL 1.0
95 // WebGL 1.0
96 // WebGL 1.0
97 // WebGL 1.0
98 // WebGL 1.0
99 // WebGL 1.0
100 // WebGL 1.0
```

```

35
36 canvas.addEventListener("mousemove", function (event) {
37     if (mouseDown) {
38         trball.end(event.clientX, event.clientY);
39
40         trballMatrix = mat4(trball.rotationMatrix);
41     }
42 });
43
44 // Configure WebGL
45 gl.viewport(0, 0, canvas.width, canvas.height);
46 gl.clearColor(0.9, 0.9, 0.9, 1.0);
47
48 // Enable hidden-surface removal
49 gl.enable(gl.DEPTH_TEST);
50
51 // Load shaders and initialize attribute buffers
52 var program = initShaders(gl, "vertex-shader", "fragment-shader");
53 gl.useProgram(program);
54
55 // Load the data into the GPU
56 var bufferId = gl.createBuffer();
57 gl.bindBuffer(gl.ARRAY_BUFFER, bufferId);
58 gl.bufferData(gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW);
59
60 // Associate our shader variables with our data buffer
61 var vPosition = gl.getAttribLocation(program, "vPosition");
62 gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
63 gl.enableVertexAttribArray(vPosition);
64
65 // Create a buffer object, initialize it, and associate it with
66 // the associated attribute variable in our vertex shader
67 var cBufferId = gl.createBuffer();
68 gl.bindBuffer(gl.ARRAY_BUFFER, cBufferId);
69 gl.bufferData(gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW);

```

[illegible]

alphaCube.html JS alphaCube.js X

C: &gt; Users &gt; GVE &gt; Desktop &gt; CG &gt; Week13 &gt; JS alphaCube.js &gt; ...

```
170
171     var vColor = gl.getAttribLocation(program, "vColor");
172     gl.vertexAttribPointer(vColor, 4, gl.FLOAT, false, 0, 0);
173     gl.enableVertexAttribArray(vColor);
174
175     //var modelMatrix = mat4(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
176     modelMatrixLoc = gl.getUniformLocation(program, "modelMatrix");
177     //gl.uniformMatrix4fv(modelMatrixLoc, false, flatten(modelMatrix));
178
179     var viewMatrix = lookAt(vec3(0.0, 0.0, 1.0), vec3(0.0, 0.0, 0.0), vec3(0.0, 1.0, 0.0));
180     var viewMatrixLoc = gl.getUniformLocation(program, "viewMatrix");
181     gl.uniformMatrix4fv(viewMatrixLoc, false, flatten(viewMatrix));
182
183     // 3D orthographic viewing
184     var viewLength = 1.0;
185     var projectionMatrix;
186     if (canvas.width > canvas.height) {
187         var aspect = viewLength * canvas.width / canvas.height;
188         projectionMatrix = ortho(-aspect, aspect, -viewLength, viewLength, -viewLength, 1000);
189     }
190     else {
191         var aspect = viewLength * canvas.height / canvas.width;
192         projectionMatrix = ortho(-viewLength, viewLength, -aspect, aspect, -viewLength, 1000);
193     }
194     /*
195     // 3D perspective viewing
196     var aspect = canvas.width / canvas.height;
197     var projectionMatrix = perspective(90, aspect, 0.1, 1000);
198     */
199     var projectionMatrixLoc = gl.getUniformLocation(program, "projectionMatrix");
200     gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));
201
202     render();
203 };
204
```

```
170
171     var vColor = gl.getAttribLocation(program, "vColor");
172     gl.vertexAttribPointer(vColor, 4, gl.FLOAT, false, 0, 0);
173     gl.enableVertexAttribArray(vColor);
174
175     //var modelMatrix = mat4(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
176     modelMatrixLoc = gl.getUniformLocation(program, "modelMatrix");
177     //gl.uniformMatrix4fv(modelMatrixLoc, false, flatten(modelMatrix));
178
179     var viewMatrix = lookAt(vec3(0.0, 0.0, 1.0), vec3(0.0, 0.0, 0.0), vec3(0.0, 1.0, 0.0));
180     var viewMatrixLoc = gl.getUniformLocation(program, "viewMatrix");
181     gl.uniformMatrix4fv(viewMatrixLoc, false, flatten(viewMatrix));
182
183     // 3D orthographic viewing
184     var viewLength = 1.0;
185     var projectionMatrix;
186     if (canvas.width > canvas.height) {
187         var aspect = viewLength * canvas.width / canvas.height;
188         projectionMatrix = ortho(-aspect, aspect, -viewLength, viewLength, -viewLength, 1000);
189     }
190     else {
191         var aspect = viewLength * canvas.height / canvas.width;
192         projectionMatrix = ortho(-viewLength, viewLength, -aspect, aspect, -viewLength, 1000);
193     }
194     /*
195     // 3D perspective viewing
196     var aspect = canvas.width / canvas.height;
197     var projectionMatrix = perspective(90, aspect, 0.1, 1000);
198     */
199     var projectionMatrixLoc = gl.getUniformLocation(program, "projectionMatrix");
200     gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));
201
202     render();
203 };
204
```

```

alphaCube.html JS alphaCube.js X
C: > Users > GVE > Desktop > CG > Week13 > JS alphaCube.js > ...
104
105 function render() {
106     gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
107
108     gl.uniformMatrix4fv(modelMatrixLoc, false, flatten(trballMatrix));
109
110     gl.drawArrays(gl.TRIANGLES, 0, numVertCubeTri);
111
112     requestAnimationFrame(render);
113 }
114
115 function generateColorCube() {
116     numVertCubeTri = 0;
117     quad(1, 0, 3, 2);
118     quad(2, 3, 7, 6);
119     quad(3, 0, 4, 7);
120     quad(4, 5, 6, 7);
121     quad(5, 4, 0, 1);
122     quad(6, 5, 1, 2);
123 }
124
125 function quad(a, b, c, d) {
126     vertexPos = [
127         vec4(-0.5, -0.5, -0.5, 1.0),
128         vec4( 0.5, -0.5, -0.5, 1.0),
129         vec4( 0.5,  0.5, -0.5, 1.0),
130         vec4(-0.5,  0.5, -0.5, 1.0),
131         vec4(-0.5, -0.5,  0.5, 1.0),
132         vec4( 0.5, -0.5,  0.5, 1.0),
133         vec4( 0.5,  0.5,  0.5, 1.0),
134         vec4(-0.5,  0.5,  0.5, 1.0)
135     ];
136
137     vertexColor = [
138         vec4(0.0, 0.0, 0.0, 1.0), // black

```

alphaCube.html

JS alphaCube.js

C: > Users > GVE > Desktop > CG > Week13 > JS alphaCube.js > ...

137

vertexColor = [

138

vec4(0.0, 0.0, 0.0, 1.0),    // black

139

vec4(1.0, 0.0, 0.0, 1.0),    // red

140

vec4(1.0, 1.0, 0.0, 1.0),    // yellow

141

vec4(0.0, 1.0, 0.0, 1.0),    // green

142

vec4(0.0, 0.0, 1.0, 1.0),    // blue

143

vec4(1.0, 0.0, 1.0, 1.0),    // magenta

144

vec4(1.0, 1.0, 1.0, 1.0),    // white

145

vec4(0.0, 1.0, 1.0, 1.0)    // cyan

146

];

147

148

// two triangles: (a, b, c) and (a, c, d)

149

// solid colored faces

150

points.push(vertexPos[a]);

151

colors.push(vertexColor[a]);

152

numVertCubeTri++;

153

154

points.push(vertexPos[b]);

155

colors.push(vertexColor[a]);

156

numVertCubeTri++;

157

158

points.push(vertexPos[c]);

159

colors.push(vertexColor[a]);

160

numVertCubeTri++;

161

162

points.push(vertexPos[a]);

163

colors.push(vertexColor[a]);

164

numVertCubeTri++;

165

166

points.push(vertexPos[c]);

167

colors.push(vertexColor[a]);

168

numVertCubeTri++;

169

170

points.push(vertexPos[d]);

171

colors.push(vertexColor[a]);



alphaCube.html

JS alphaCube.js

C: > Users > GVE > Desktop > CG > Week13 > JS alphaCube.js > ...

148

// two triangles: (a, b, c) and (a, c, d)

149

// solid colored faces

150

points.push(vertexPos[a]);

151

colors.push(vertexColor[a]);

152

numVertCubeTri++;

153

154

points.push(vertexPos[b]);

155

colors.push(vertexColor[a]);

156

numVertCubeTri++;

157

158

points.push(vertexPos[c]);

159

colors.push(vertexColor[a]);

160

numVertCubeTri++;

161

162

points.push(vertexPos[a]);

163

colors.push(vertexColor[a]);

164

numVertCubeTri++;

165

166

points.push(vertexPos[c]);

167

colors.push(vertexColor[a]);

168

numVertCubeTri++;

169

170

points.push(vertexPos[d]);

171

colors.push(vertexColor[a]);

172

numVertCubeTri++;

173

}

174

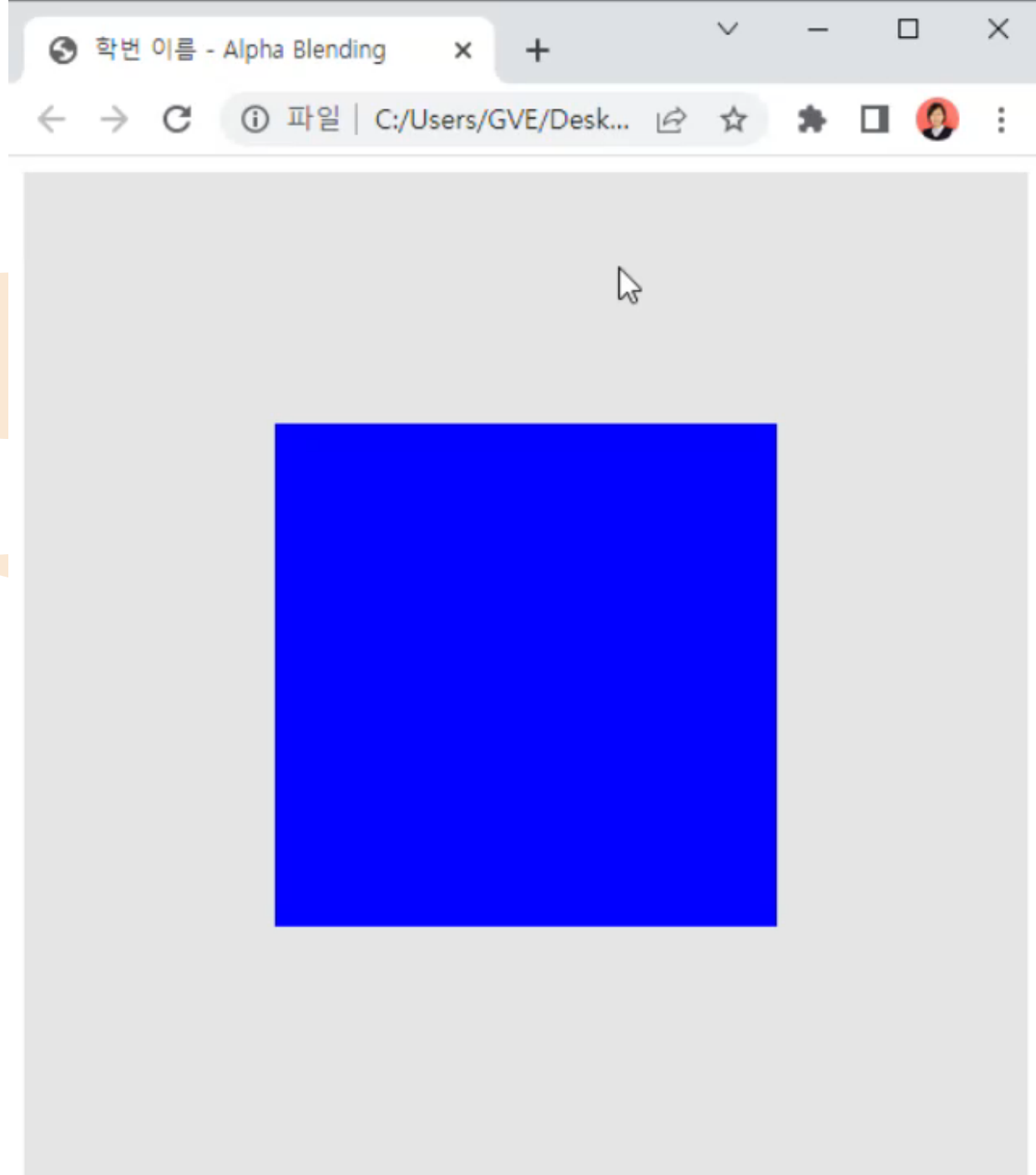
173

174

173

174

Restricted Mode 0 0 Ln 174, Col 1 Spaces: 4 UTF-8 CRLF {} JavaScript



&lt;&gt; alphaCube.html X JS alphaCube.js

C: &gt; Users &gt; GVE &gt; Desktop &gt; CG &gt; Week13 &gt; &lt;&gt; alphaCube.html &gt; html &gt; head &gt; script#fragment-shader

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>학번 이름 - Alpha Blending</title>
5      <script id="vertex-shader" type="x-shader/x-vertex">
6        attribute vec4 vPosition;
7        attribute vec4 vColor;
8
9        uniform mat4 modelMatrix, viewMatrix, projectionMatrix;
10
11        varying vec4 fColor;
12
13        void main() {
14          gl_Position = projectionMatrix * viewMatrix * modelMatrix * vPosition;
15
16          fColor = vColor;
17        }
18      </script>
19
20      <script id="fragment-shader" type="x-shader/x-fragment">
21        precision mediump float;
22
23        varying vec4 fColor;
24
25        void main() {
26          gl_FragColor = fColor;
27          //gl_FragColor.a = 1.0;
28        }
29      </script>
30
31      <script type="text/javascript" src="../../Common/webgl-utils.js"></script>
32      <script type="text/javascript" src="../../Common/initShaders.js"></script>
33      <script type="text/javascript" src="../../Common/MV.js"></script>
34      <script type="text/javascript" src="../../trackball.js"></script>
35      <script type="text/javascript" src="alphaCube.js"></script>
```

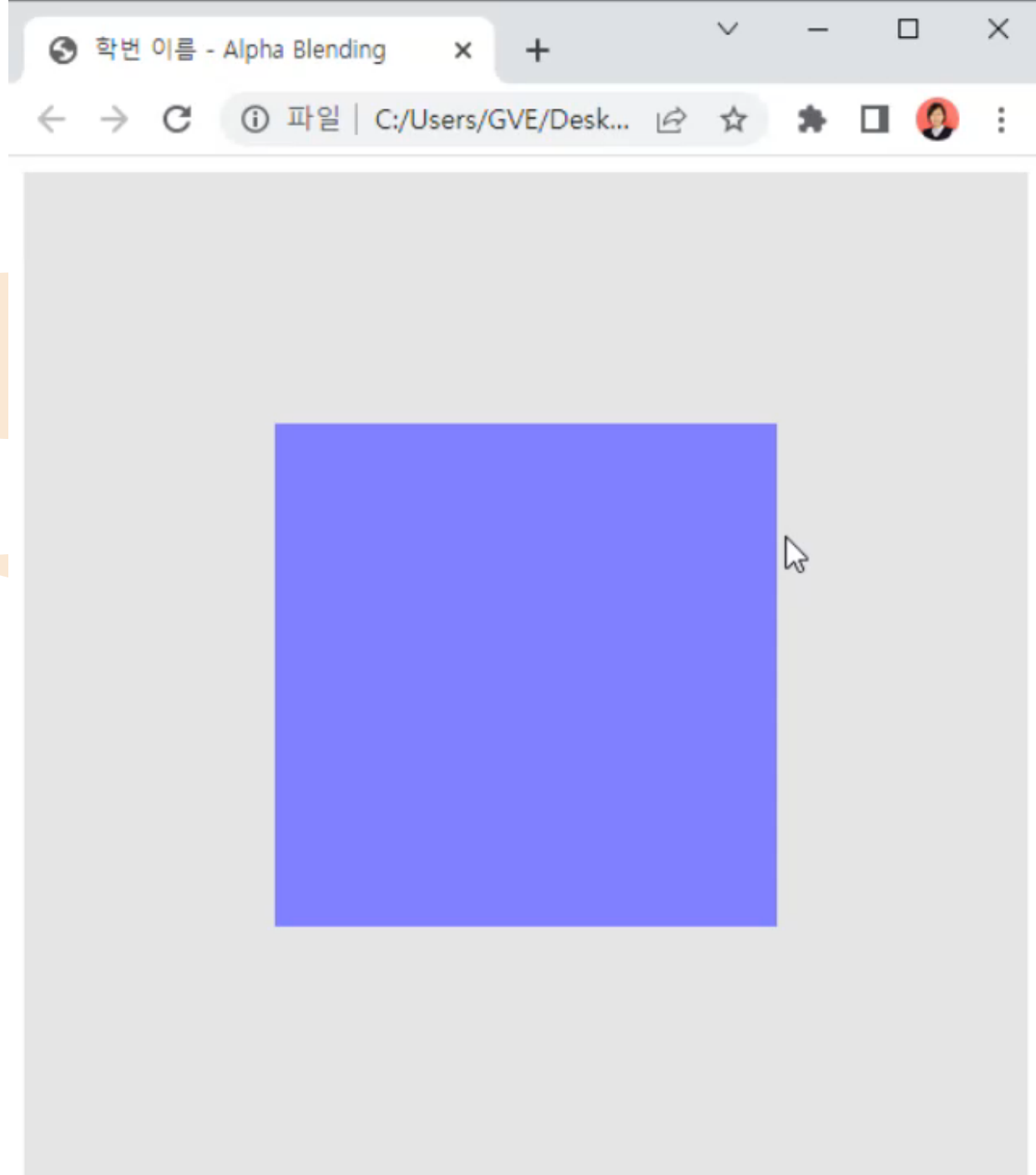


```

alphaCube.html JS alphaCube.js X
C: > Users > GVE > Desktop > CG > Week13 > JS alphaCube.js > quad

124
125 function quad(a, b, c, d) {
126     vertexPos = [
127         vec4(-0.5, -0.5, -0.5, 1.0),
128         vec4( 0.5, -0.5, -0.5, 1.0),
129         vec4( 0.5,  0.5, -0.5, 1.0),
130         vec4(-0.5,  0.5, -0.5, 1.0),
131         vec4(-0.5, -0.5,  0.5, 1.0),
132         vec4( 0.5, -0.5,  0.5, 1.0),
133         vec4( 0.5,  0.5,  0.5, 1.0),
134         vec4(-0.5,  0.5,  0.5, 1.0)
135     ];
136
137     vertexColor = [
138         vec4(0.0, 0.0, 0.0, 0.5), // black
139         vec4(1.0, 0.0, 0.0, 0.5), // red
140         vec4(1.0, 1.0, 0.0, 0.5), // yellow
141         vec4(0.0, 1.0, 0.0, 0.5), // green
142         vec4(0.0, 0.0, 1.0, 0.5), // blue
143         vec4(1.0, 0.0, 1.0, 0.5), // magenta
144         vec4(1.0, 1.0, 1.0, 0.5), // white
145         vec4(0.0, 1.0, 1.0, 0.5) // cyan
146     ];
147
148     // two triangles: (a, b, c) and (a, c, d)
149     // solid colored faces
150     points.push(vertexPos[a]);
151     colors.push(vertexColor[a]);
152     numVertCubeTri++;
153
154     points.push(vertexPos[b]);
155     colors.push(vertexColor[a]);
156     numVertCubeTri++;
157
158     points.push(vertexPos[c]);

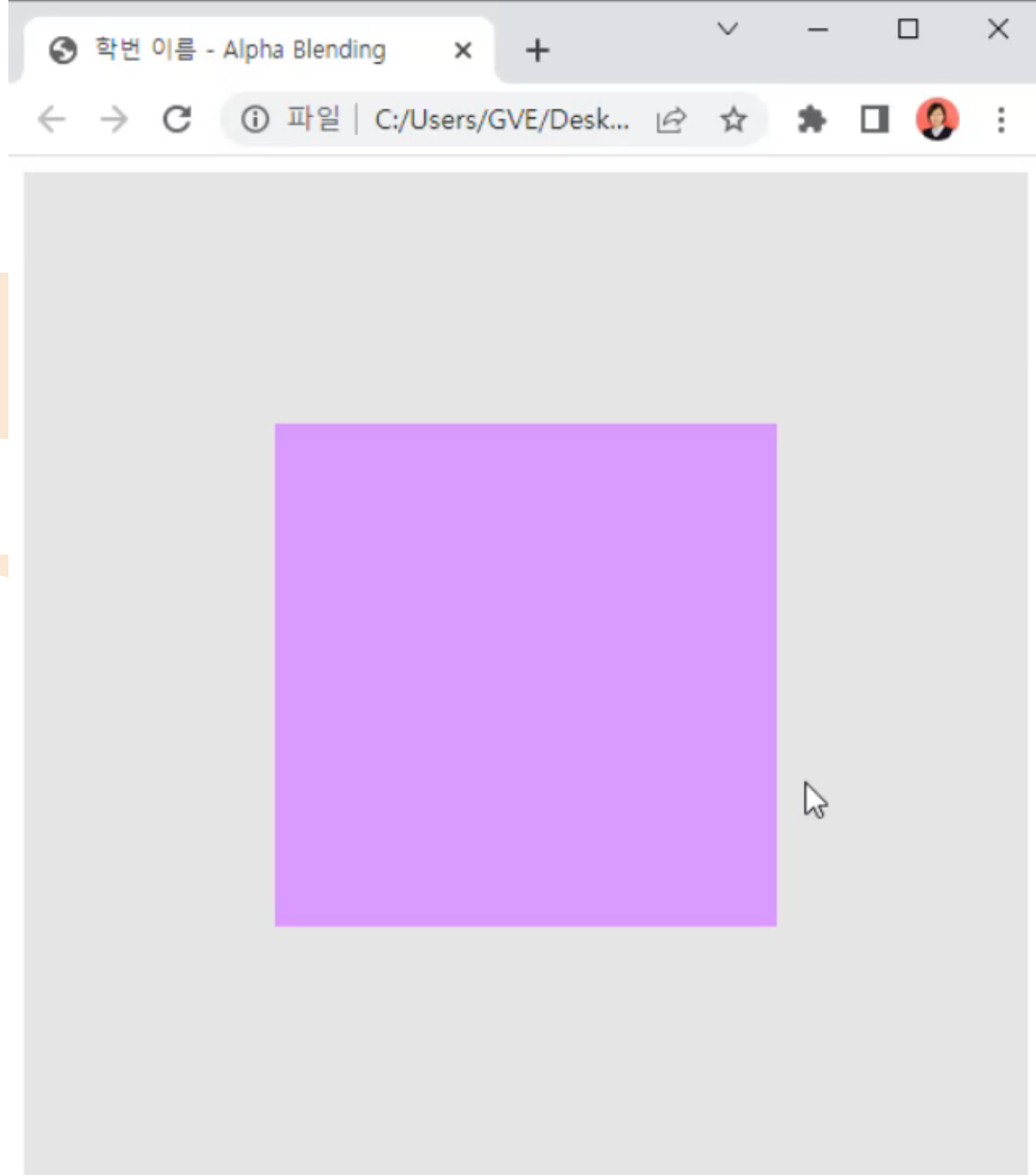
```



```

alphaCube.html JS alphaCube.js X
C: > Users > GVE > Desktop > CG > Week13 > JS alphaCube.js > init
43
44 // Configure WebGL
45 gl.viewport(0, 0, canvas.width, canvas.height);
46 gl.clearColor(0.9, 0.9, 0.9, 1.0);
47
48 // Enable hidden-surface removal
49 gl.enable(gl.DEPTH_TEST);
50 gl.enable(gl.BLEND);
51 gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
52
53 // Load shaders and initialize attribute buffers
54 var program = initShaders(gl, "vertex-shader", "fragment-shader");
55 gl.useProgram(program);
56
57 // Load the data into the GPU
58 var bufferId = gl.createBuffer();
59 gl.bindBuffer(gl.ARRAY_BUFFER, bufferId);
60 gl.bufferData(gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW);
61
62 // Associate our shader variables with our data buffer
63 var vPosition = gl.getAttribLocation(program, "vPosition");
64 gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
65 gl.enableVertexAttribArray(vPosition);
66
67 // Create a buffer object, initialize it, and associate it with
68 // the associated attribute variable in our vertex shader
69 var cBufferId = gl.createBuffer();
70 gl.bindBuffer(gl.ARRAY_BUFFER, cBufferId);
71 gl.bufferData(gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW);
72
73 var vColor = gl.getAttribLocation(program, "vColor");
74 gl.vertexAttribPointer(vColor, 4, gl.FLOAT, false, 0, 0);
75 gl.enableVertexAttribArray(vColor);
76
77 //var modelMatrix = mat4(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);

```



```

alphaCube.html JS alphaCube.js X
C: > Users > GVE > Desktop > CG > Week13 > JS alphaCube.js > init
43
44 // Configure WebGL
45 gl.viewport(0, 0, canvas.width, canvas.height);
46 gl.clearColor(0.9, 0.9, 0.9, 1.0);
47
48 // Enable hidden-surface removal
49 //gl.enable(gl.DEPTH_TEST);
50 gl.enable(gl.BLEND);
51 gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
52
53 // Load shaders and initialize attribute buffers
54 var program = initShaders(gl, "vertex-shader", "fragment-shader");
55 gl.useProgram(program);
56
57 // Load the data into the GPU
58 var bufferId = gl.createBuffer();
59 gl.bindBuffer(gl.ARRAY_BUFFER, bufferId);
60 gl.bufferData(gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW);
61
62 // Associate our shader variables with our data buffer
63 var vPosition = gl.getAttribLocation(program, "vPosition");
64 gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
65 gl.enableVertexAttribArray(vPosition);
66
67 // Create a buffer object, initialize it, and associate it with
68 // the associated attribute variable in our vertex shader
69 var cBufferId = gl.createBuffer();
70 gl.bindBuffer(gl.ARRAY_BUFFER, cBufferId);
71 gl.bufferData(gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW);
72
73 var vColor = gl.getAttribLocation(program, "vColor");
74 gl.vertexAttribPointer(vColor, 4, gl.FLOAT, false, 0, 0);
75 gl.enableVertexAttribArray(vColor);
76
77 //var modelMatrix = mat4(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);

```



