3. Interaction and Animation

Outline

- Input and Interaction
- Animation
- Working with Callbacks
- Position Input
- Picking
- Sample Programs

Input and Interaction

Objectives

- Introduce the basic input devices
  - Physical Devices
  - Logical Devices
  - Input Modes
- Event-driven input
- Introduce double buffering for smooth animations
- Programming event input with WebGL

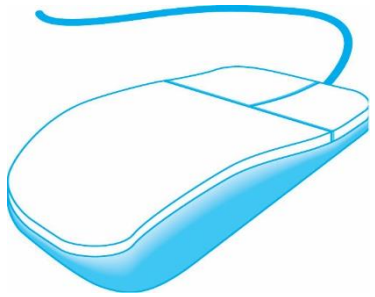
Project Sketchpad

- Ivan Sutherland (MIT 1963) established the basic interactive paradigm that characterizes interactive computer graphics:
  - User sees an *object* on the display
  - User points to (*picks*) the object with an input device (light pen, mouse, trackball)
  - Object changes (moves, rotates, morphs)
  - Repeat

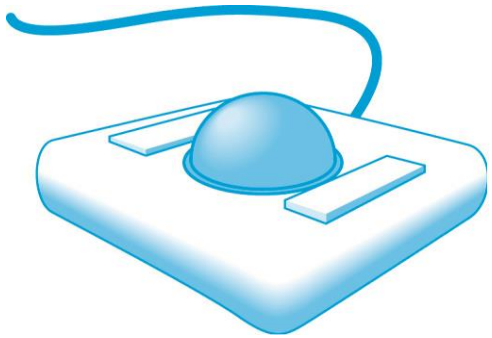
Graphical Input

- Devices can be described either by
  - Physical properties
    - Mouse
    - Keyboard
    - Trackball
  - Logical Properties
    - What is returned to program via API
      - A position
      - An object identifier
- Modes
  - How and when input is obtained
    - Request or event

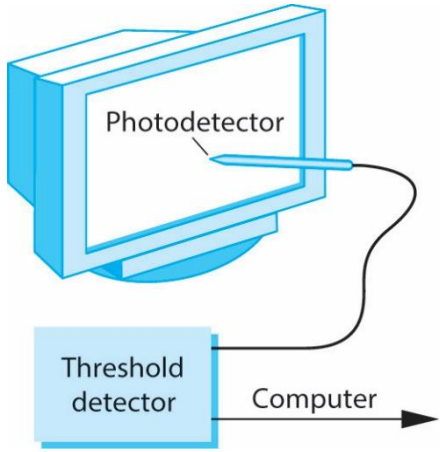
Physical Devices



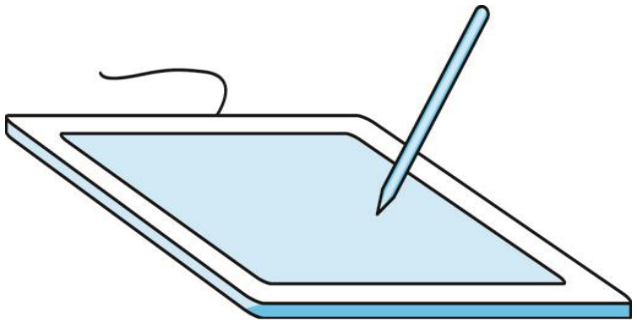
mouse



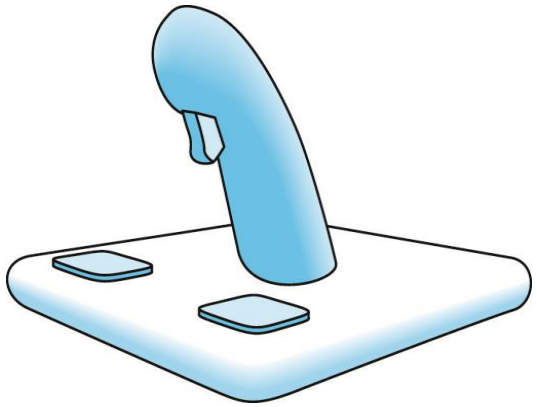
trackball



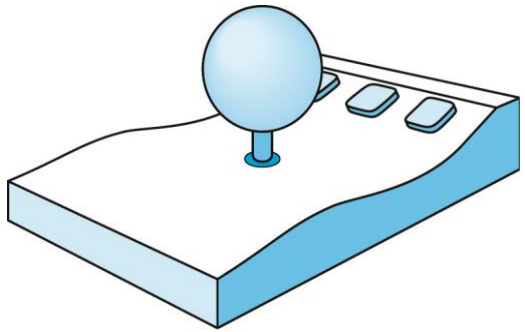
light pen



data tablet

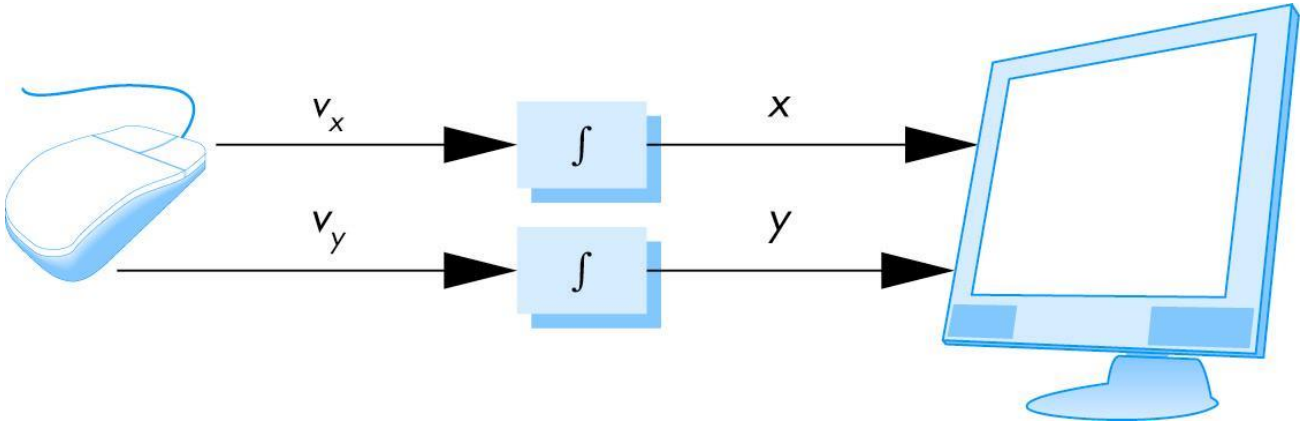


joy stick



space ball

Cursor Positioning



Incremental (Relative) Devices

- Devices such as the data tablet return a position directly to the operating system
- Devices such as the mouse, trackball, and joy stick return incremental inputs (or velocities) to the operating system
  - Must integrate these inputs to obtain an absolute position
    - Rotation of cylinders in mouse
    - Roll of trackball
    - Difficult to obtain absolute position
    - Can get variable sensitivity

Logical Devices

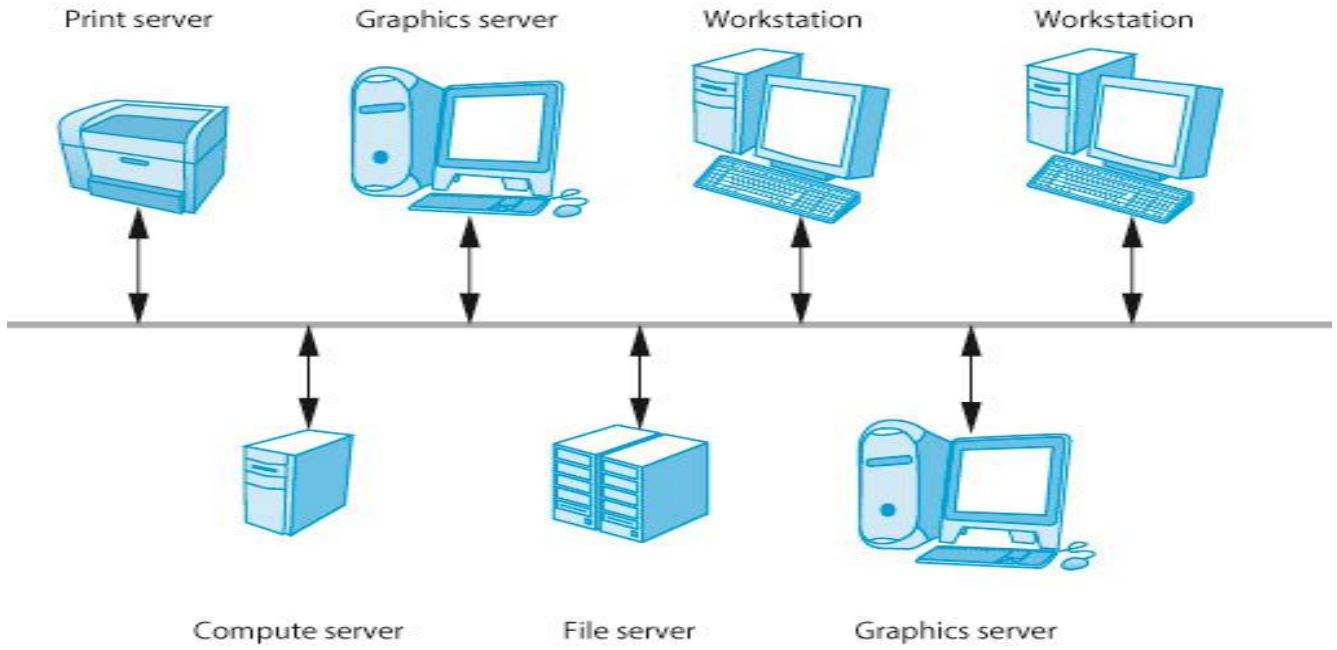
- Consider the C and C++ code
  - C++: **`cin >> x;`**
  - C: **`scanf ("%d", &x);`**
- What is the input device?
  - Can't tell from the code
  - Could be keyboard, file, output from another program
- The code provides *logical input*
  - A number (an **`int`**) is returned to the program regardless of the physical device

Graphical Logical Devices

- Graphical input is more varied than input to standard programs which is usually numbers, characters, or bits
- Two older APIs (GKS, PHIGS) defined six types of logical input
  - **Locator**: return a position
  - **Pick**: return ID of an object
  - **Keyboard**: return strings of characters
  - **Stroke**: return array of positions
  - **Valuator**: return floating point number
  - **Choice**: return one of n items

X Window Input

- The X Window System introduced a client-server model for a network of workstations
  - **Client**: OpenGL program
  - **Graphics Server**: bitmap display with a pointing device and a keyboard

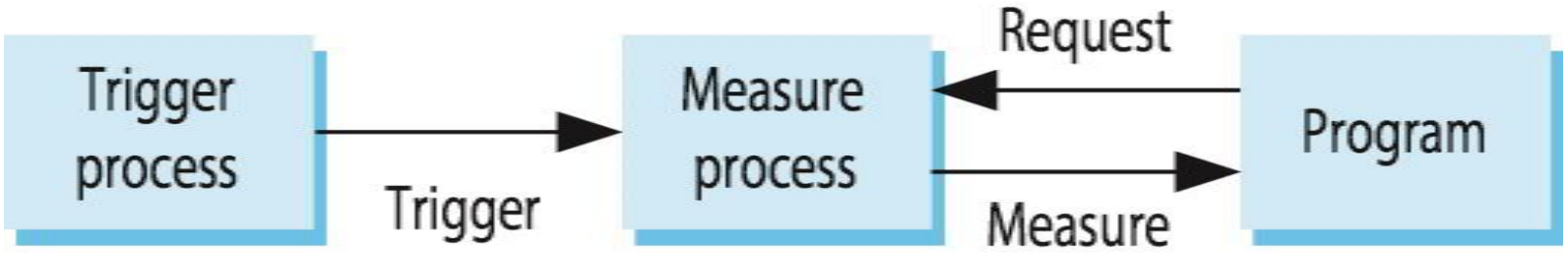


Input Modes

- Input devices contain a *trigger* which can be used to send a signal to the operating system
  - Button on mouse
  - Pressing or releasing a key
- When triggered, input devices return information (their *measure*) to the system
  - Mouse returns position information
  - Keyboard returns ASCII code

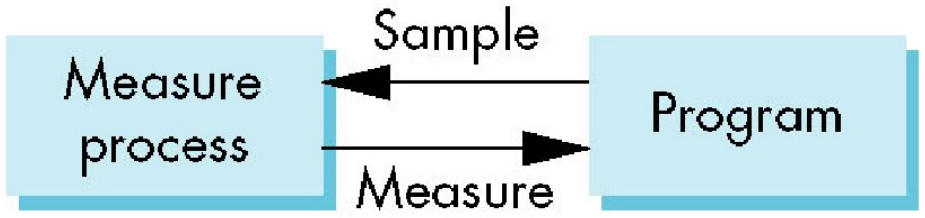
Request Mode

- Input provided to program only when user triggers the device
- Typical of keyboard input
  - Can erase (backspace), edit, correct until enter (return) key (the trigger) is depressed



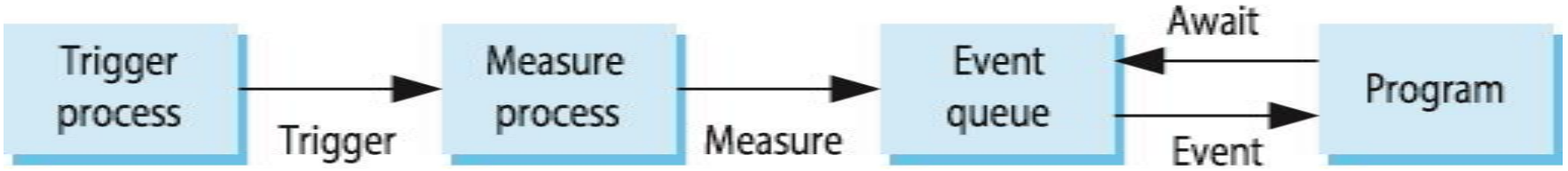
Sample Mode

- Input is immediate, no trigger necessary
- Users must have positioned pointing device or entered data using keyboard before the function is called



Event Mode

- Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user
- Each trigger generates an *event* whose measure is put in an *event queue* which can be examined by the user program



Event Types

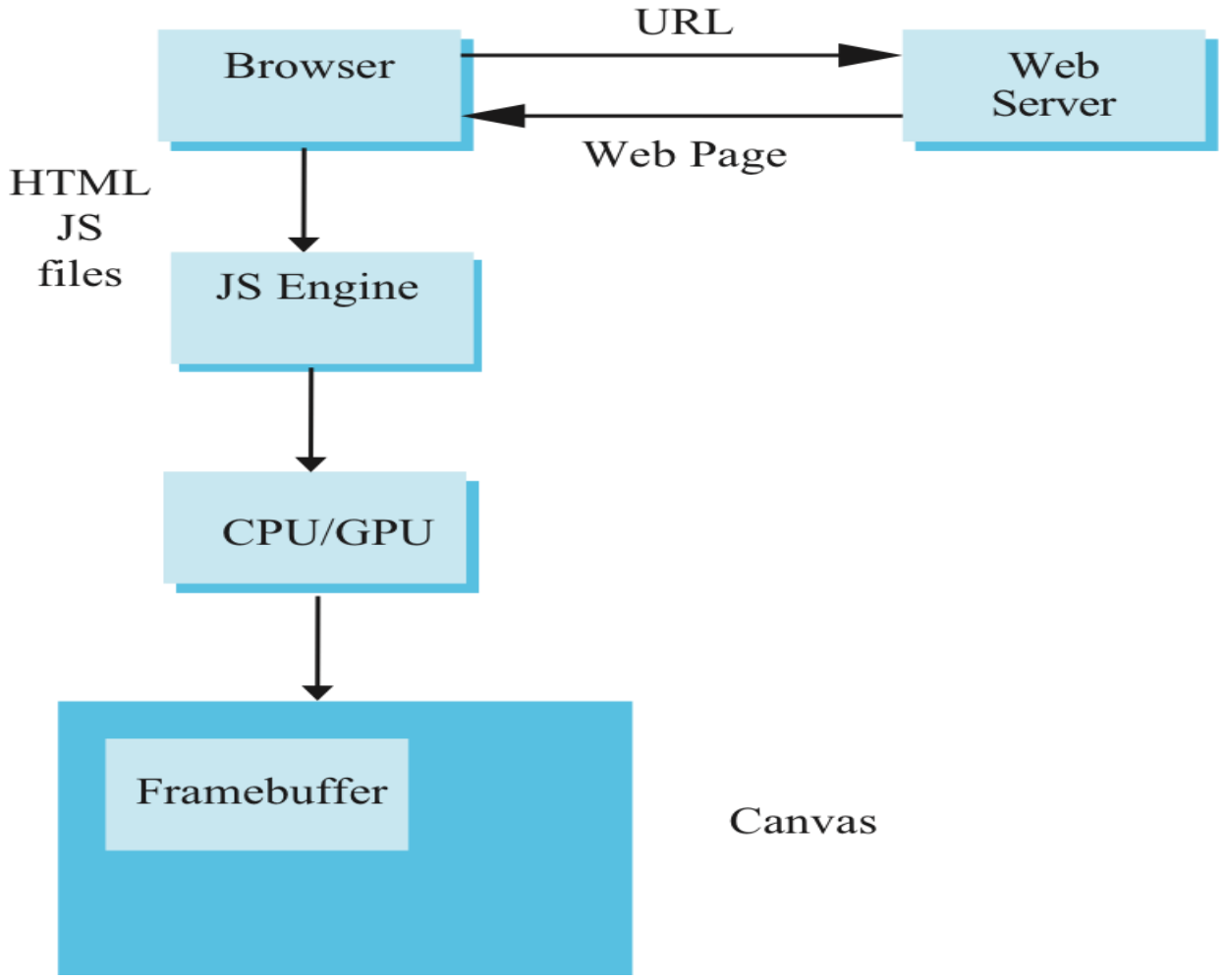
- Window: resize, expose, iconify
- Mouse: click one or more buttons
- Motion: move mouse
- Keyboard: press or release a key
- Idle: nonevent
  - Define what should be done if no other event is in queue

Animation

Callbacks

- Programming interface for event-driven input uses *callback functions* or *event listeners*
  - Define a callback for each event the graphics system recognizes
  - Browsers enters an event loop and responds to those events for which it has callbacks registered
  - The callback function is executed when the event occurs

Execution in a Browser



Execution in a Browser

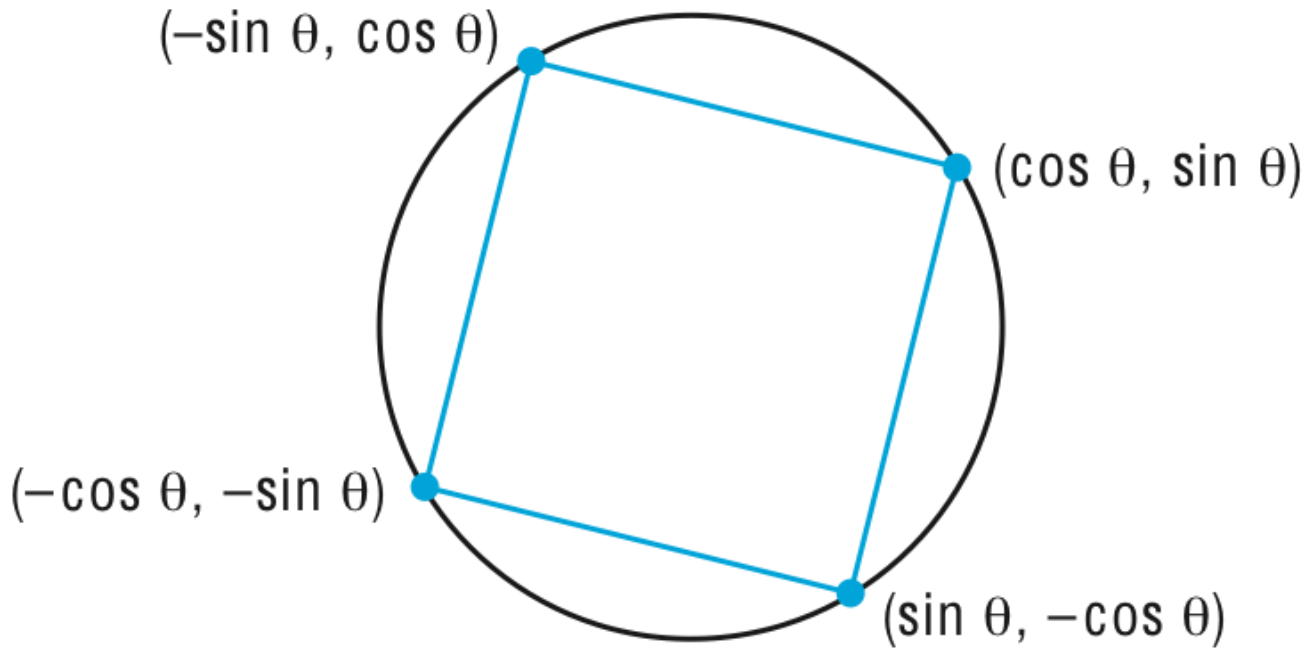
- Start with HTML file
  - Describes the page
  - May contain the shaders
  - Loads files
- Files are loaded asynchronously and JS code is executed
- Then what?
- Browser is in an event loop and waits for an event

onload Event

- What happens with our JS file containing the graphics part of our application?
  - All the “action” is within functions such as init() and render()
  - Consequently these functions are never executed and we see nothing
- Solution: use the onload window event to initiate execution of the init function
  - onload event occurs when all files read
  - window.onload = init;

Rotating Square

- Consider the four points



Animate display by rerendering with different values of θ

Simple but Slow Method

```
for(var theta = 0.0; theta <thetaMax; theta += dtheta; {

    vertices[0] = vec2(Math.sin(theta), Math.cos.(theta));
    vertices[1] = vec2(Math.sin(theta), -Math.cos.(theta));
    vertices[2] = vec2(-Math.sin(theta), -Math.cos.(theta));
    vertices[3] = vec2(-Math.sin(theta), Math.cos.(theta));

    gl.bufferSubData(……................

    render();
}
```

Better Way

- Send original vertices to vertex shader

- Send θ to shader as a uniform variable
- Compute vertices in vertex shader
- Render recursively

Render Function

```
var thetaLoc = gl.getUniformLocation(program, "theta");

function render()
{
    gl.clear(gl.COLOR_BUFFER_BIT);
    theta += 0.1;
    gl.uniform1f(thetaLoc, theta);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    render();
}
```

Vertex Shader

```
attribute vec4 vPosition;
uniform float theta;

void main()
{
   gl_Position.x = -sin(theta) * vPosition.x + cos(theta) * vPosition.y;
   gl_Position.y =  sin(theta) * vPosition.y + cos(theta) * vPosition.x;
   gl_Position.z = 0.0;
   gl_Position.w = 1.0;
}
```

Double Buffering

- Although we are rendering the square, it always into a buffer that is not displayed
- Browser uses double buffering
  - Always display front buffer
  - Rendering into back buffer
  - Need a buffer swap
- Prevents display of a partial rendering

Triggering a Buffer Swap

- Browsers refresh the display at ~60 Hz
  - redisplay of front buffer
  - not a buffer swap
- Trigger a buffer swap though an event
- Two options for rotating square
  - Interval timer
  - requestAnimFrame

Interval Timer

- Executes a function after a specified number of milliseconds
  - Also generates a buffer swap

```
setInterval(render, interval);
```

- Note an interval of 0 generates buffer swaps as fast as possible

requestAnimFrame

```
function render {
    gl.clear(gl.COLOR_BUFFER_BIT);
    theta += 0.1;
    gl.uniform1f(thetaLoc, theta);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    requestAnimFrame(render);
}
```

Add an Interval

```
function render()
{
    setTimeout( function() {
      requestAnimFrame(render);
      gl.clear(gl.COLOR_BUFFER_BIT);
      theta += 0.1;
      gl.uniform1f(thetaLoc, theta);
      gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    }, 100);
}
```

Working with Callbacks

Objectives

- Learn to build interactive programs using event listeners
  - Buttons
  - Menus
  - Mouse
  - Keyboard
  - Reshape

Adding a Button

- Let's add a button to control the rotation direction for our rotating cube
- In the render function we can use a var direction which is true or false to add or subtract a constant to the angle

```
var direction = true; // global initialization

// in render()

if(direction) theta += 0.1;
else theta -= 0.1;
```

The Button

- In the HTML file

```
<button id="DirectionButton">Change Rotation Direction
</button>
```

  - Uses HTML button tag
  - id gives an identifier we can use in JS file
  - Text “Change Rotation Direction” displayed in button
- Clicking on button generates a click event
- Note we are using default style and could use CSS or jQuery to get a prettier button

Button Event Listener

- We still need to define the listener
  - no listener and the event occurs but is ignored
- Two forms for event listener in JS file

```
var myButton = document.getElementById("DirectionButton");

myButton.addEventListener("click", function() {
    direction = !direction;
});
```

```
document.getElementById("DirectionButton").onclick =
function() { direction = !direction; };
```

onclick Variants

```
myButton.addEventListener("click", function() {
if (event.button == 0) { direction = !direction; }
});
```

```
myButton.addEventListener("click", function() {
if (event.shiftKey == 0) { direction = !direction; }
});
```

```
<button onclick="direction = !direction"></button>
```

Controling Rotation Speed

```
var delay = 100;

function render()
{
   setTimeout(function() {
      requestAnimFrame(render);
      gl.clear(gl.COLOR_BUFFER_BIT);
      theta += (direction ? 0.1 : -0.1);
      gl.uniform1f(thetaLoc, theta);
      gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   }, delay);
}
```

Menus

- Use the HTML select element
- Each entry in the menu is an option element with an integer value returned by click event

```
<select id="mymenu" size="3">
<option value="0">Toggle Rotation Direction</option>
<option value="1">Spin Faster</option>
<option value="2">Spin Slower</option>
</select>
```

Menu Listener

```
var m = document.getElementById("mymenu");
m.addEventListener("click", function() {
   switch (m.selectedIndex) {
      case 0:
         direction = !direction;
         break;
      case 1:
         delay /= 2.0;
         break;
      case 2:
         delay *= 2.0;
         break;
       }
});
```

Using keydown Event

```
window.addEventListener("keydown", function() {
   switch (event.keyCode) {
      case 49: // ’1’ key
         direction = !direction;
         break;
      case 50: // ’2’ key
         delay /= 2.0;
         break;
      case 51: // ’3’ key
         delay *= 2.0;
         break;
   }
});
```

Don’t Know Unicode

```
window.onkeydown = function(event) {
  var key = String.fromCharCode(event.keyCode);
  switch (key) {
    case ’1’:
      direction = !direction;
      break;
    case ’2’:
      delay /= 2.0;
      break;
    case ’3’:
      delay *= 2.0;
      break;
  }
};
```

Slider Element

- Puts slider on page
  - Give it an identifier
  - Give it minimum and maximum values
  - Give it a step size needed to generate an event
  - Give it an initial value
- Use div tag to put below canvas

```
<div>
speed 0 <input id="slide" type="range"
   min="0" max="100" step="10" value="50" />
100 </div>
```

onchange Event Listener

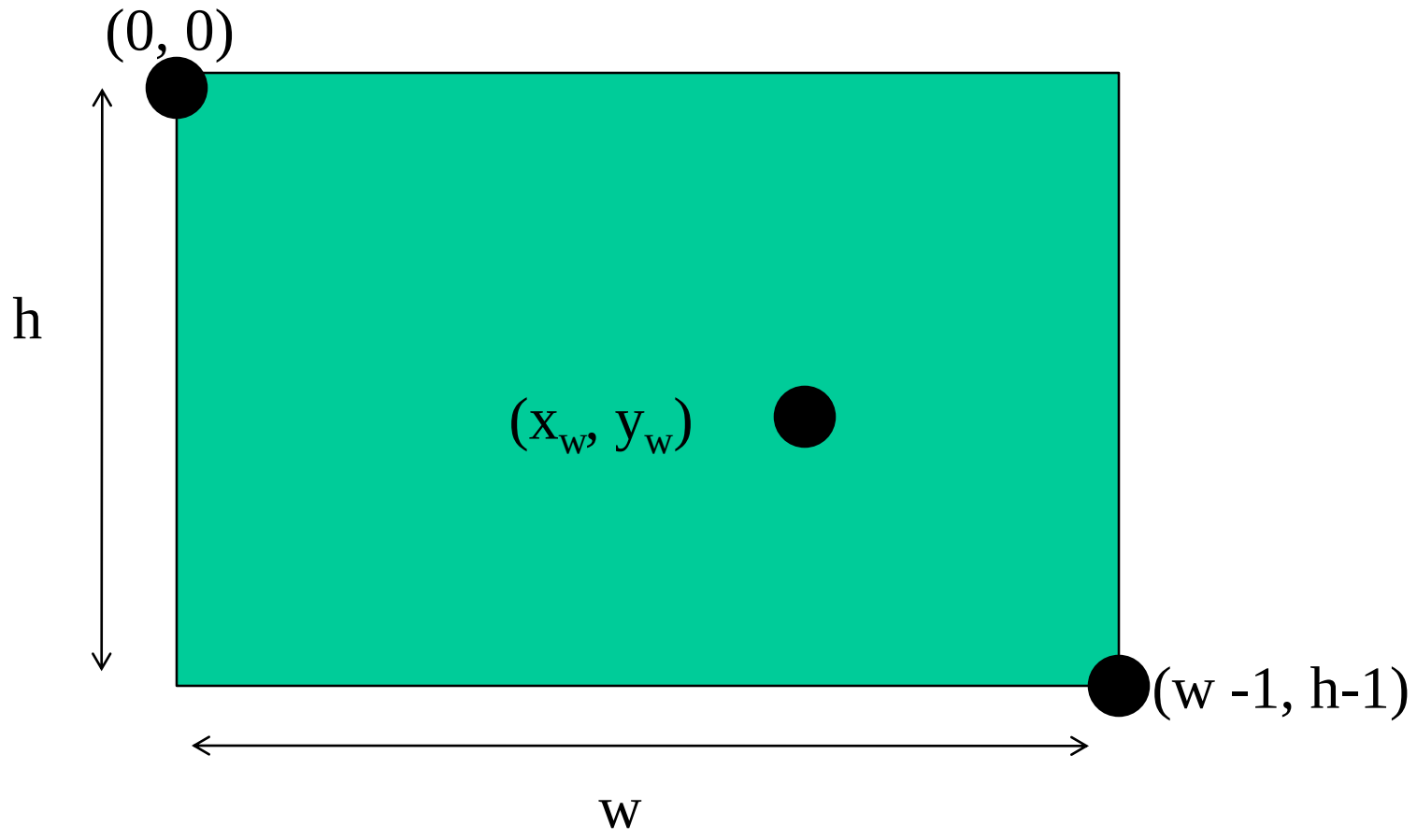
```
document.getElementById("slide").onchange =
  function() { delay = event.srcElement.value; };
```

Position Input

Objectives

- Learn to use the mouse to give locations
  - Must convert from position on canvas to position in application
- Respond to window events such as reshapes triggered by the mouse

Window Coordinates

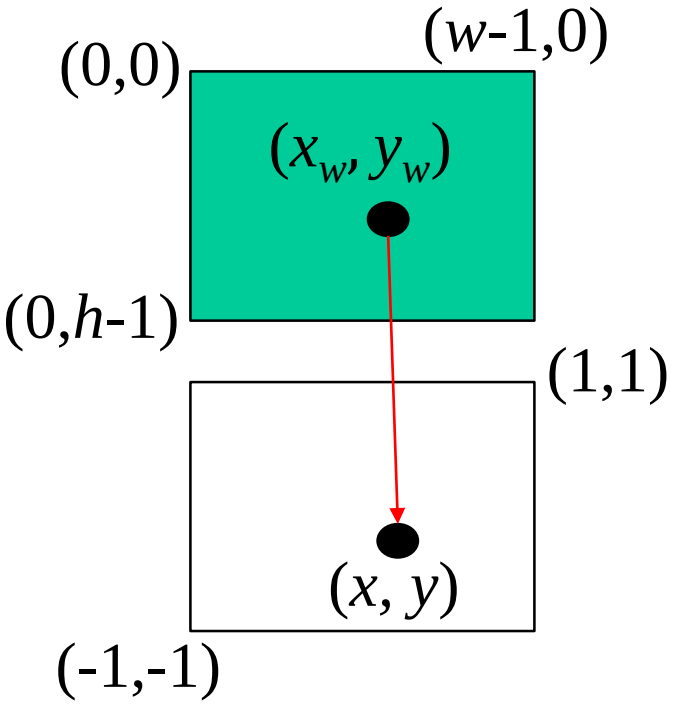


Window to Clip Coordinates

$$(0, h-1) \rightarrow (-1,-1)$$

$$(w-1, 0) \rightarrow (1,1)$$

$$(x_w, y_w) \rightarrow (x, y)$$



where: $x = -1 + \dfrac{2 * x_w}{w}$

$$y = -1 + \frac{2 * (h - y_w)}{h}$$

Returning Position from Click Event

Canvas specified in HTML file of size
canvas.width x canvas.height

Returned window coordinates are event.clientX and event.clientY

Screen coordinates World coordinates

$(x_w, y_w) \rightarrow (x, y)$

$$x = -1 + \frac{2 * x_w}{w}$$

$$y = -1 + \frac{2 * (h - y_w)}{h}$$

```
// add a vertex to GPU for each click
canvas.addEventListener("click", function() {
   gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
   var t = vec2(-1 + 2*event.clientX/canvas.width,
                -1 + 2*(canvas.height-event.clientY)/canvas.height);
   gl.bufferSubData(gl.ARRAY_BUFFER, sizeof['vec2']*index, t);
   index++;
});
```

CAD-like Examples

www.cs.unm.edu/~angel/WebGL/7E/03

square.html: puts a colored square at location of each mouse click

triangle.html: first three mouse clicks define first triangle of triangle strip. Each succeeding mouse clicks adds a new triangle at end of strip

cad1.html: draw a rectangle for each two successive mouse clicks

cad2.html: draws arbitrary polygons

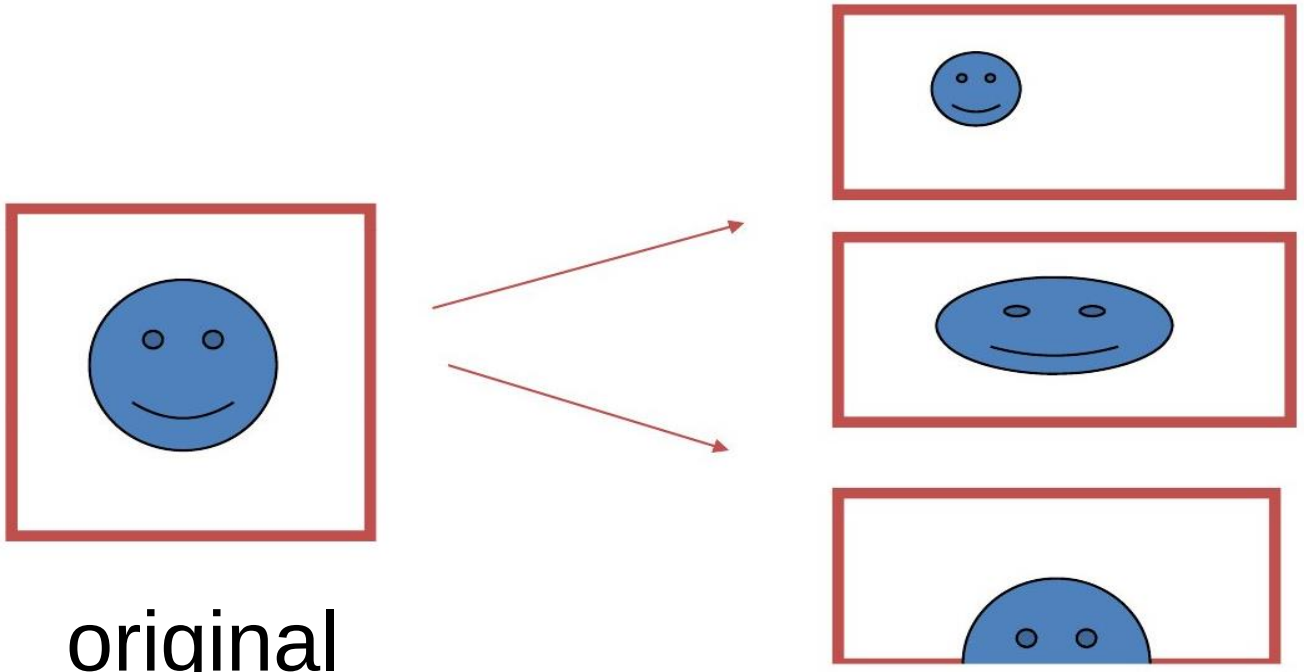
Window Events

- Events can be generated by actions that affect the canvas window
  - moving or exposing a window
  - resizing a window
  - opening a window
  - iconifying/deiconifying a window
- Note that events generated by other application that use the canvas can affect the WebGL canvas
  - There are default callbacks for some of these events

Reshape Events

- Suppose we use the mouse to change the size of our canvas
- Must redraw the contents
- Options
  - Display the same objects but change size
  - Display more or fewer objects at the same size
- Almost always want to keep proportions

Reshape Possibilities



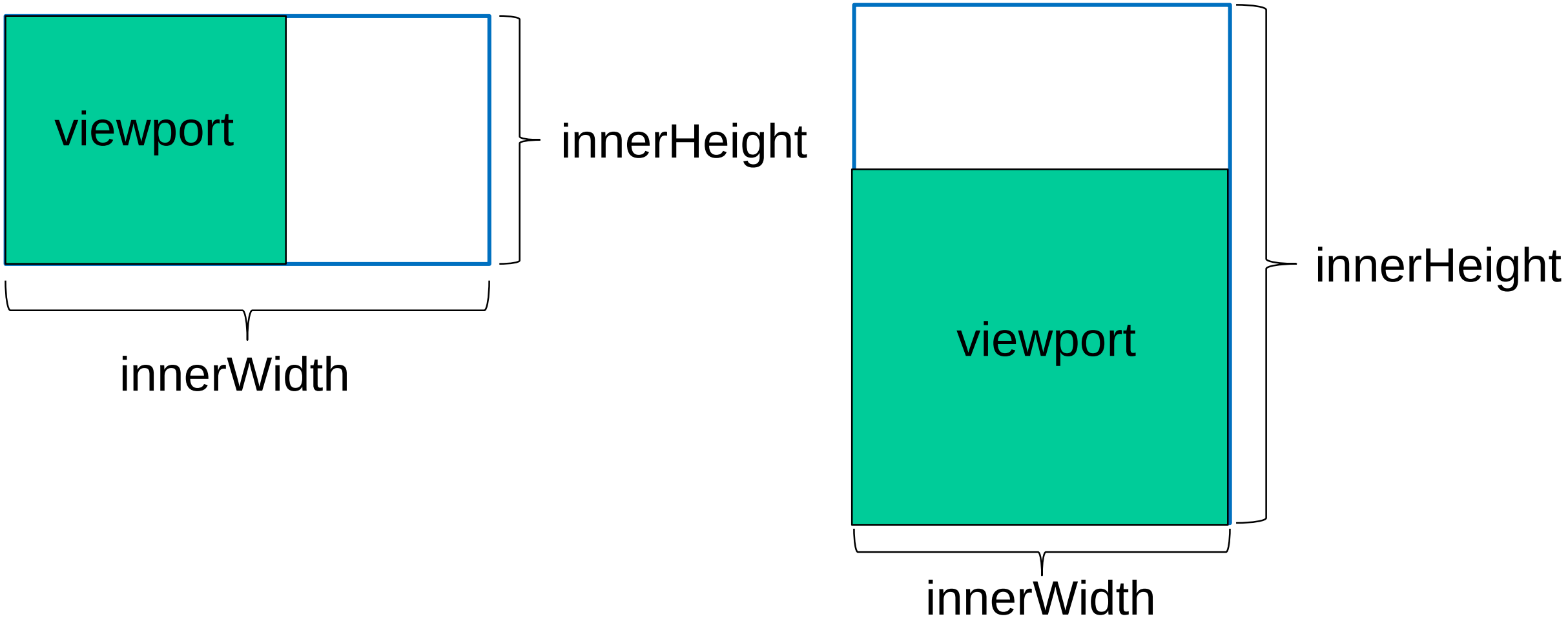
onresize Event

- Returns size of new canvas is available through window.innerHeight and window. innerWidth
- Use innerHeight and innerWidth to change canvas.height and canvas.width
- Example (next slide): maintaining a square display

Keeping Square Proportions

```
window.onresize = function() {
   var min = innerWidth;
   if (innerHeight < min) {
     min = innerHeight;
   }
   if (min < canvas.width || min < canvas.height) {
      gl.viewport(0, canvas.height-min, min, min);
   }
};
```

Keeping Square Proportions



Picking

Objectives

- How do we identify objects on the display
- Overview three methods
  - selection
  - using an off-screen buffer and color
  - bounding boxes

Why is Picking Difficult?

- Given a point in the canvas how do map this point back to an object?
- Lack of uniqueness
- Forward nature of pipeline
- Take into account difficulty of getting an exact position with a pointing device

Selection

- Supported by fixed function OpenGL pipeline
- Each primitive is given an id by the application indicating to which object it belongs
- As the scene is rendered, the id's of primitives that render near the mouse are put in a hit list
- Examine the hit list after the rendering

Selection

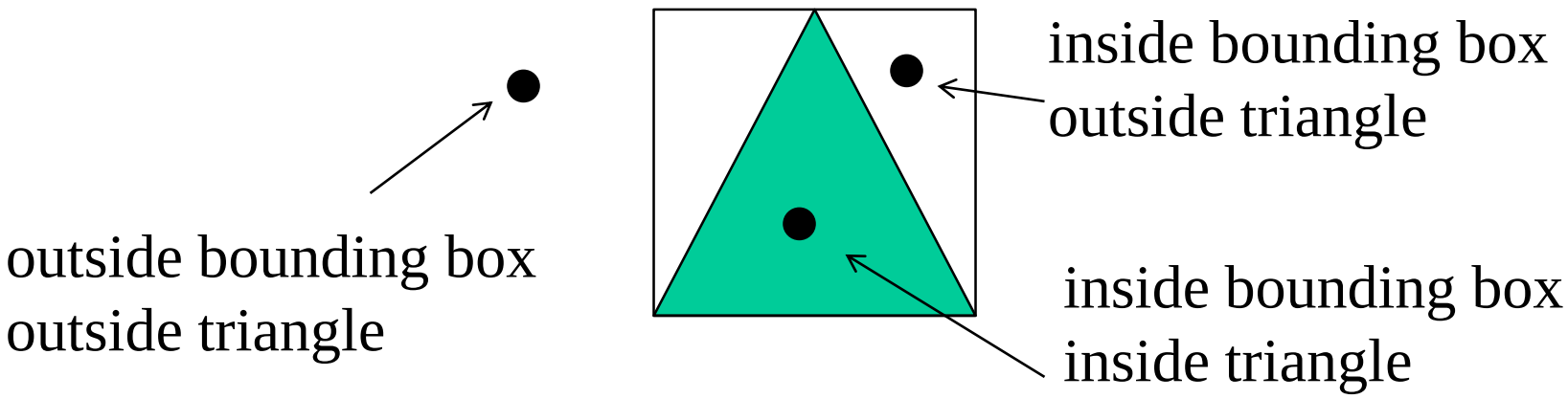
- Implement by creating a window that corresponds to small area around mouse
  - We can track whether or not a primitive renders to this window
  - Do not want to display this rendering
  - Render off-screen to an extra color buffer or user back buffer and don’t do a swap
- Requires a rendering which puts depths into hit record
- Possible to implement with WebGL

Picking with Color

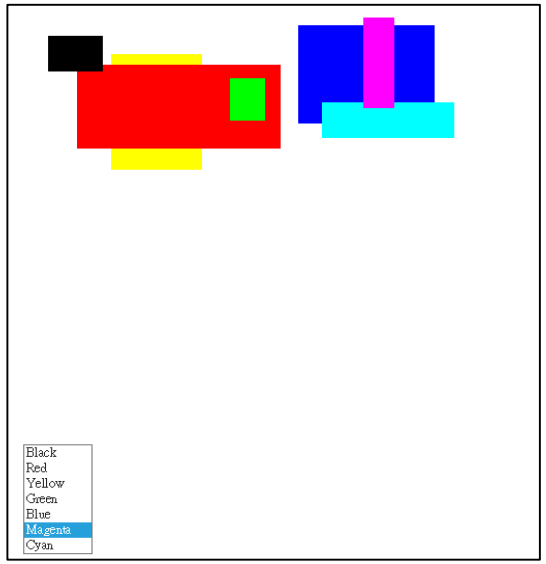
- We can use gl.readPixels to get the color at any location in window
- Idea is to use color to identify object but
  - Multiple objects can have the same color
  - A shaded object will display many colors
- Solution: assign a unique color to each object and render off-screen
  - Use gl.readPixels to get color at mouse location
  - Use a table to map this color to an object

Picking with Bounding Boxes

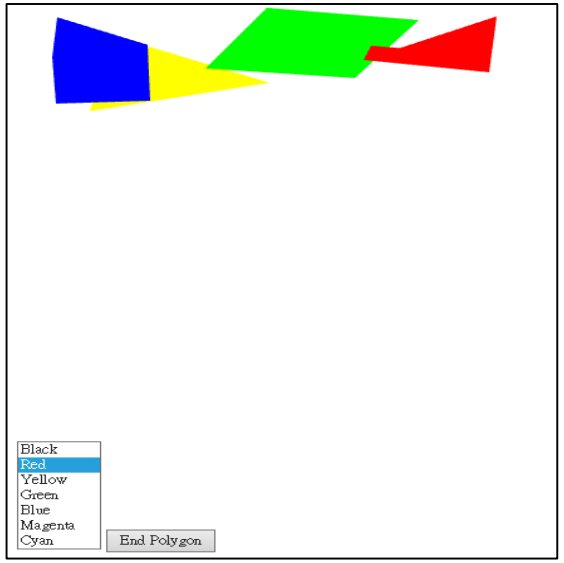
- Both previous methods require an extra rendering each time we do a pick
- Alternative is to use a table of (axis-aligned) bounding boxes
- Map mouse location to object through table



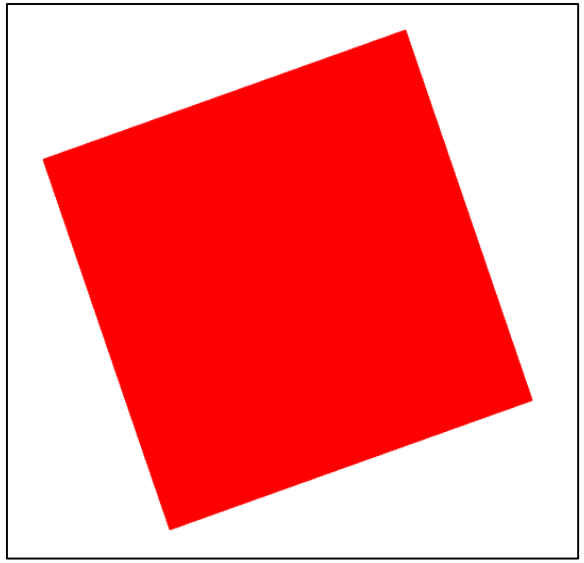
Sample Programs



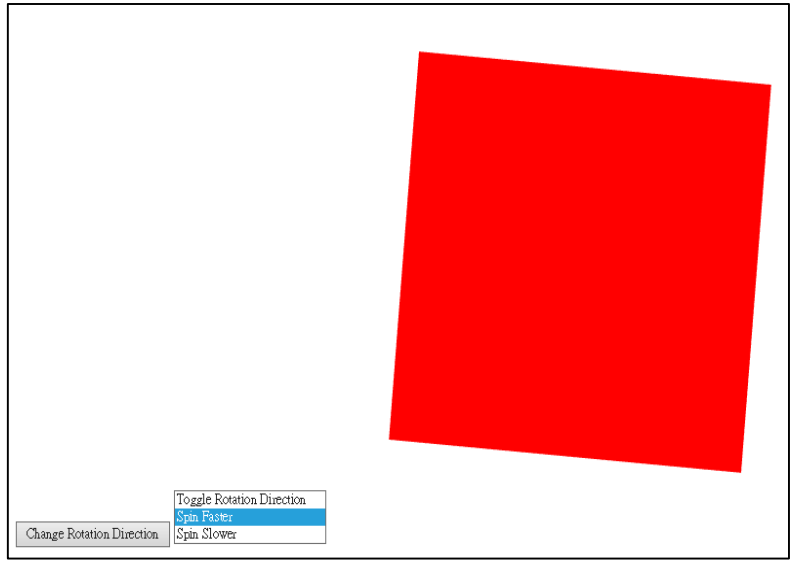
cad1



cad2

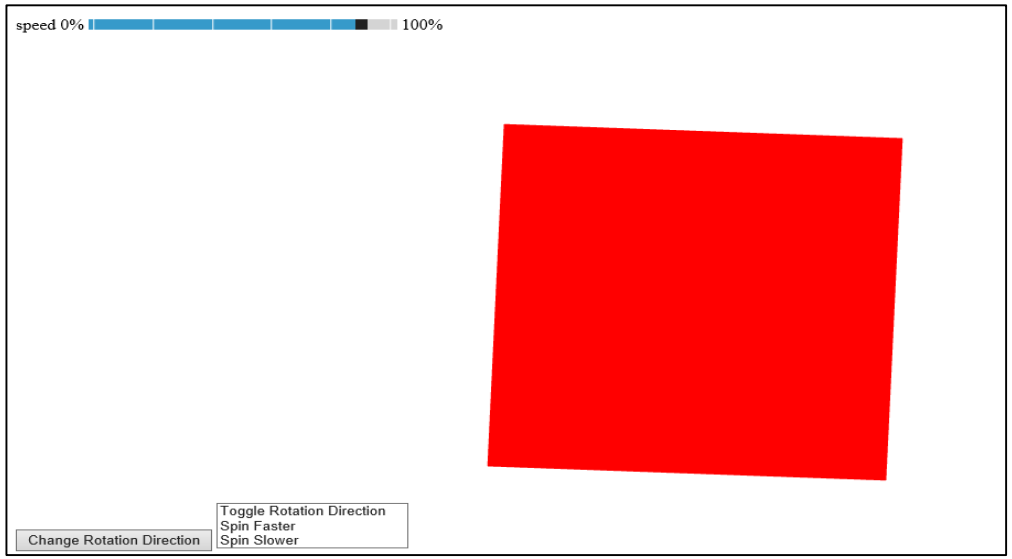


rotatingSquare1

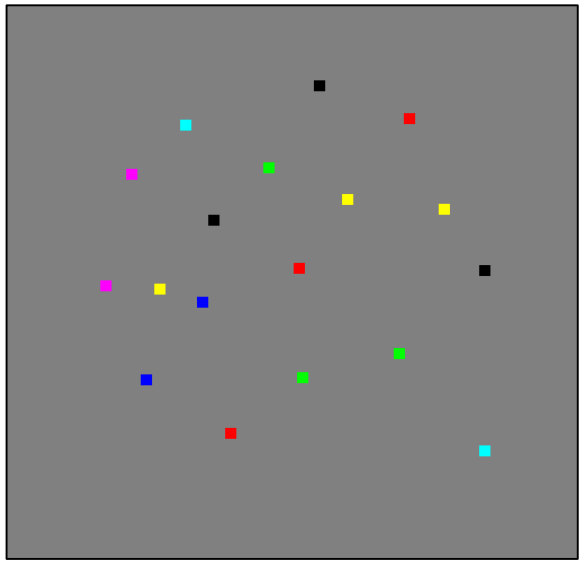


rotatingSquare2

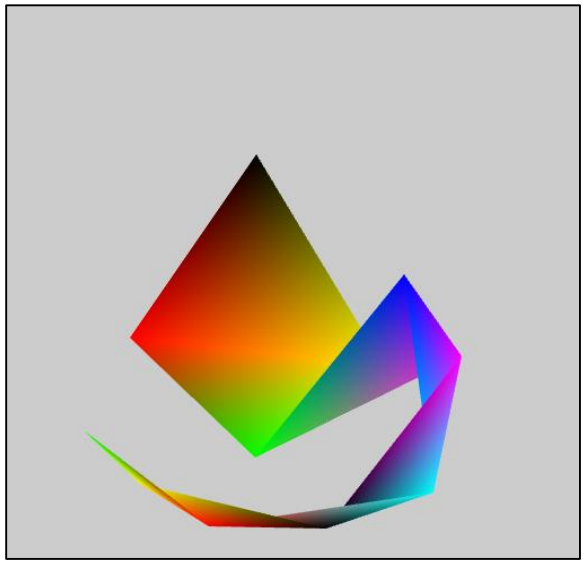
Sample Programs



rotatingSquare3

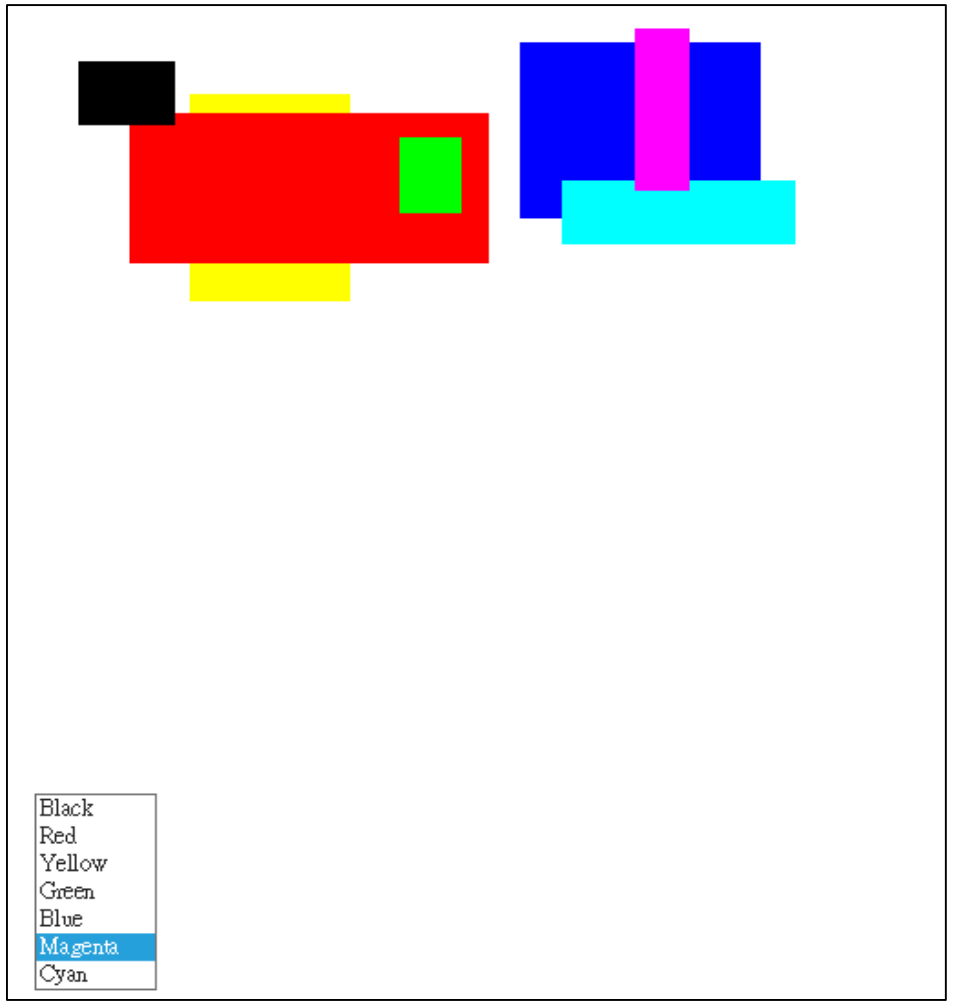


square



triangle

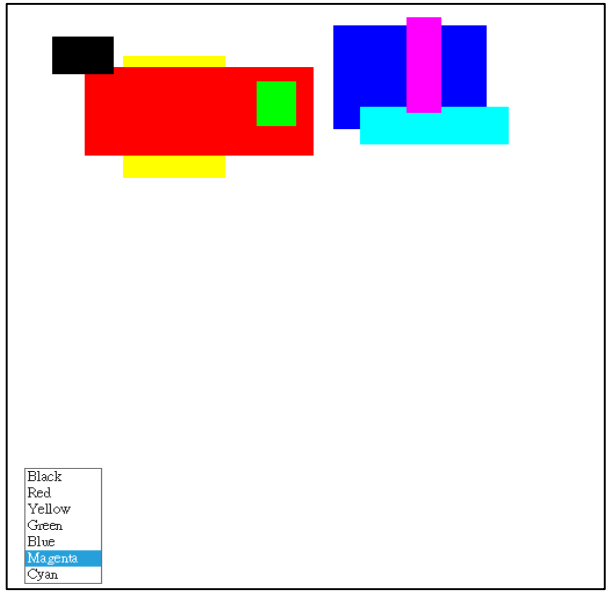
Sample Programs: cad1.html, cad1.js



Rectangle drawing. Each pair of mouse clicks adds a new rectangle

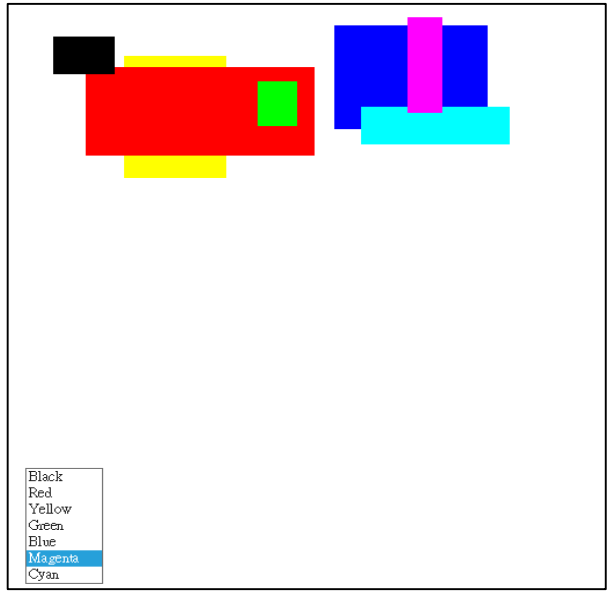
cad1.html (1/3)

```
<html>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute vec4 vColor;

varying vec4 fColor;

void main()
{
    gl_Position = vPosition;
    fColor = vColor;
}
</script>
```



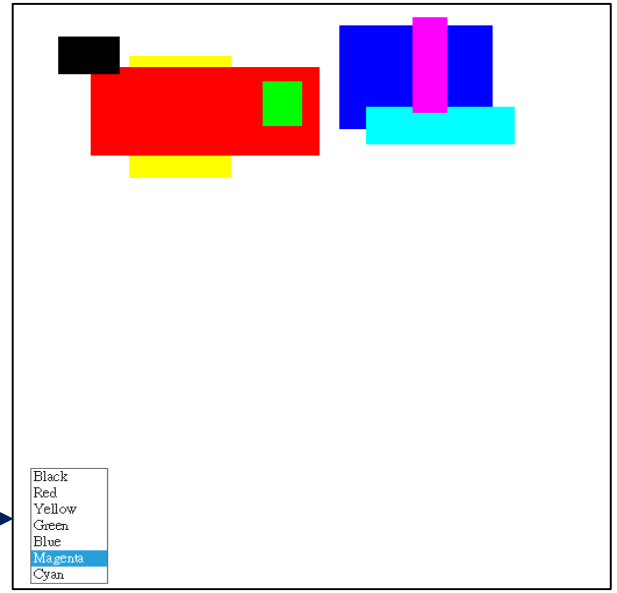
cad1.html (2/3)

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

precision mediump float;

varying vec4 fColor;
void main()
{
    gl_FragColor = fColor;
}
</script>

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="cad1.js"></script>
```



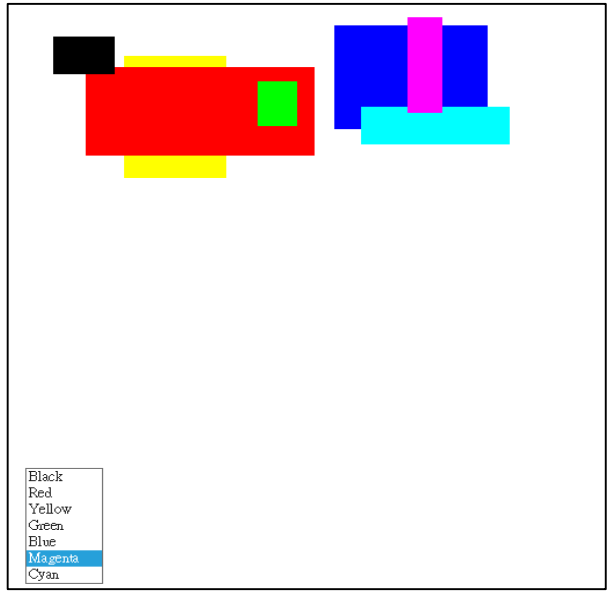
cad1.html (3/3)

```
<body>
<div>
<canvas id="gl-canvas" width="512"" height="512"
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</div>
<div>
<select id = "mymenu" size = "7">
    <option value = "0">Black</option>
    <option value = "1">Red</option>
    <option value = "2">Yellow</option>
    <option value = "3">Green</option>
    <option value = "4">Blue</option>
    <option value = "5">Magenta</option>
    <option value = "6">Cyan</option>
</select>
</div>
</body>
</html>
```



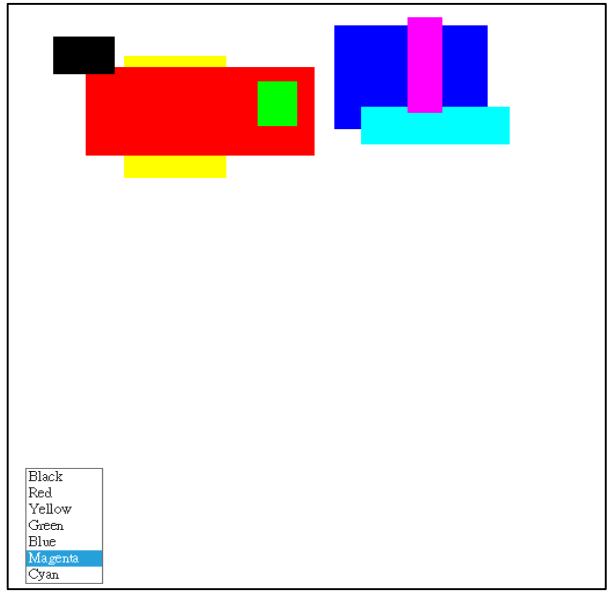
cad1.js (1/8)

```
var canvas;
var gl;

var maxNumTriangles = 200;
var maxNumVertices  = 3 * maxNumTriangles;
var index = 0;
var first = true;

var t1, t2, t3, t4;

var cIndex = 0;
```



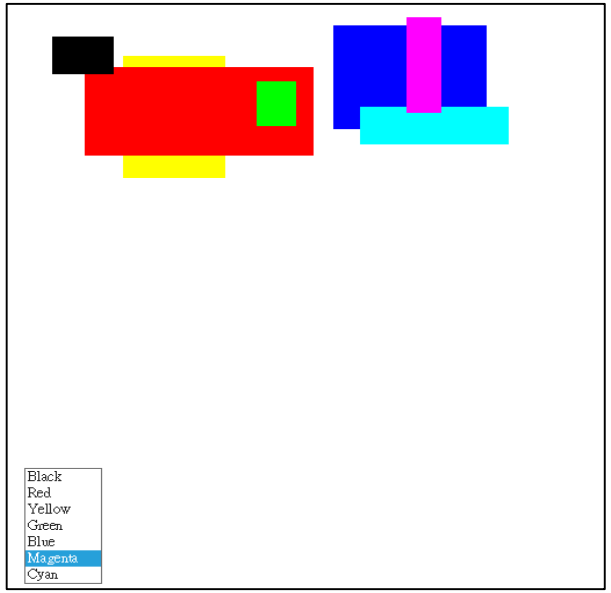
cad1.js (2/8)

```
var colors = [
    vec4( 0.0, 0.0, 0.0, 1.0 ),  // black
    vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
    vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
    vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    vec4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    vec4( 0.0, 1.0, 1.0, 1.0 )   // cyan
];
```



cad1.js (3/8)

```
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( 0.8, 0.8, 0.8, 1.0 );
    gl.clear( gl.COLOR_BUFFER_BIT );
```



cad1.js (4/8)

```
//  Load shaders and initialize attribute buffers

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

var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, 8*maxNumVertices, gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program, "vPosition");
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```

8 bytes (storing (x,y) coordinates) / 2D vertex

buffer containing vertex attributes, such as vertex coordinates, texture coordinate data, or vertex color data.

setting the size of the buffer object's data store

contents of the buffer are likely to be used often and not change often

Fixed point values are accessed

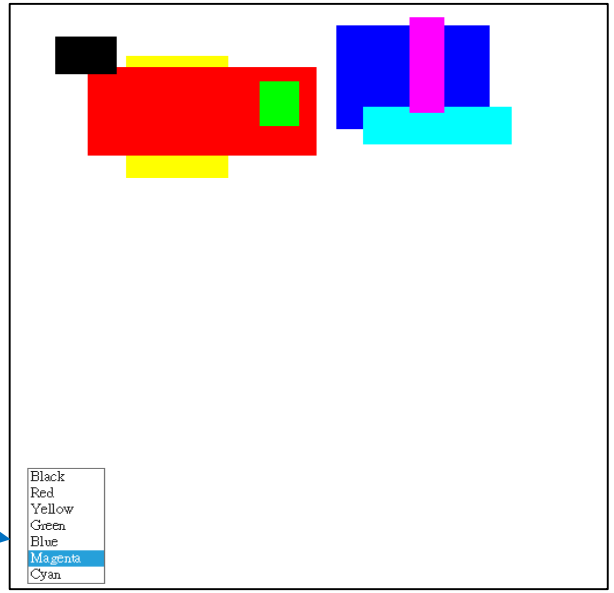
the offset in bytes between the beginning of consecutive vertex attributes

an offset in bytes of the first component in the vertex attribute array

cad1.js (5/8)

16 bytes (storing RGBA values/4 floats)/vertex color

```
var cBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cBuffer);
gl.bufferData(gl.ARRAY_BUFFER, 16*maxNumVertices, gl.STATIC_DRAW );

var vColor = gl.getAttribLocation( program, "vColor");
gl.vertexAttribPointer(vColor, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vColor);

var m = document.getElementById("mymenu");

m.addEventListener("click", function() { cIndex = m.selectedIndex; });
```



cad1.js (6/8)

$(x_w, y_w) \rightarrow (x, y)$

Screen coordinates World coordinates

$$x = -1 + \frac{2 * x_w}{w}$$

$$y = -1 + \frac{2 * (h - y_w)}{h}$$

```
canvas.addEventListener("mousedown", function(){
    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer);
    if(first) {
      first = false;
      gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer)
      t1 = vec2(2*event.clientX/canvas.width-1,
                2*(canvas.height-event.clientY)/canvas.height-1);
    }
    else {
      first = true;
      t2 = vec2(2*event.clientX/canvas.width-1,
                2*(canvas.height-event.clientY)/canvas.height-1);
      t3 = vec2(t1[0], t2[1]);
      t4 = vec2(t2[0], t1[1]);
      gl.bufferSubData(gl.ARRAY_BUFFER, 8*index,      flatten(t1));
      gl.bufferSubData(gl.ARRAY_BUFFER, 8*(index+1), flatten(t3));
      gl.bufferSubData(gl.ARRAY_BUFFER, 8*(index+2), flatten(t2));
      gl.bufferSubData(gl.ARRAY_BUFFER, 8*(index+3), flatten(t4));
```

mouse

Buffer containing vertex attributes, such as vertex coordinates, texture coordinate data, or vertex color data

an offset in bytes where the data replacement will start

data that will be copied into the data store.

t1 t4

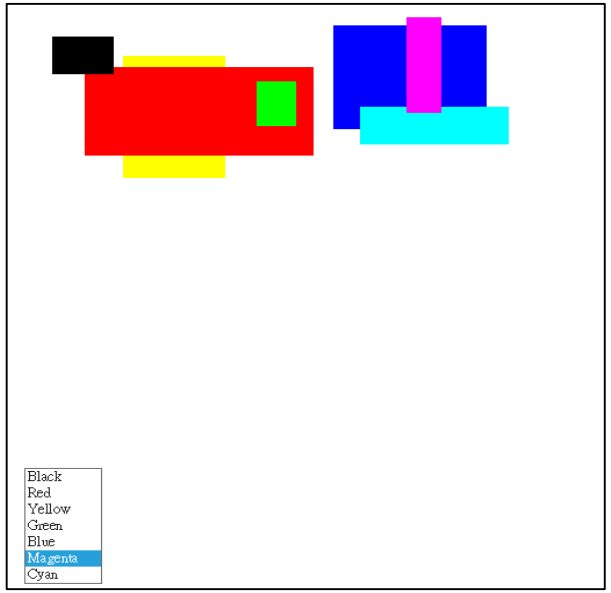
gl.TRIANGLE_FAN

t3 t2

cad1.js (7/8)

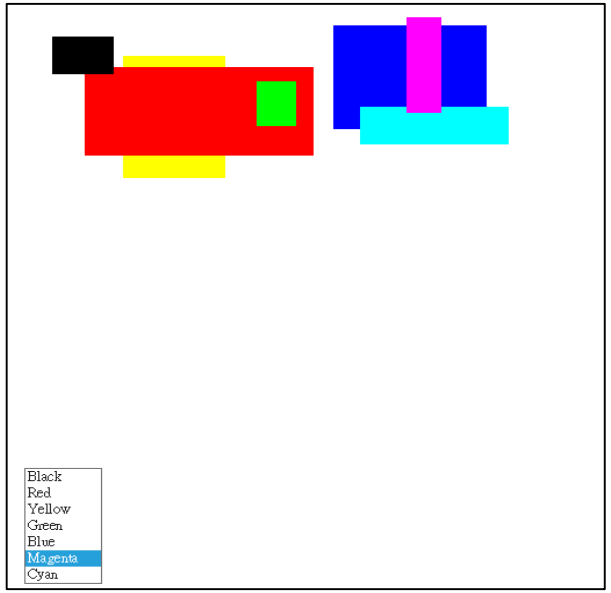
16 bytes (storing RGBA values/vertex color)

```
        gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer);
        t = vec4(colors[cIndex]);

        gl.bufferSubData(gl.ARRAY_BUFFER, 16*(index),     flatten(t));
        gl.bufferSubData(gl.ARRAY_BUFFER, 16*(index+1), flatten(t));
        gl.bufferSubData(gl.ARRAY_BUFFER, 16*(index+2), flatten(t));
        gl.bufferSubData(gl.ARRAY_BUFFER, 16*(index+3), flatten(t));
      }
        index += 4;
    } );

    render();

}  // end of window.onload
```

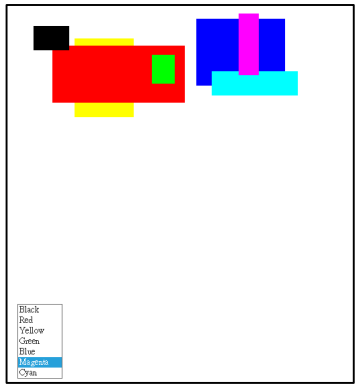


cad1.js (8/8)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT );

    for(var i = 0; i<index; i+=4)
        gl.drawArrays( gl.TRIANGLE_FAN, i, 4 );

    window.requestAnimFrame(render);

}
```



cad1.js : Data Structures



one rectangle one rectangle one rectangle index=12

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
vBuffer	0 8 bytes	8	16	24	32	40	48	56	64	72	80	88	96	
cBuffer	0 16 bytes	16	32	48	64	80	96	112	128	144	160	176	192	

(x,y) coordinates RGBA values

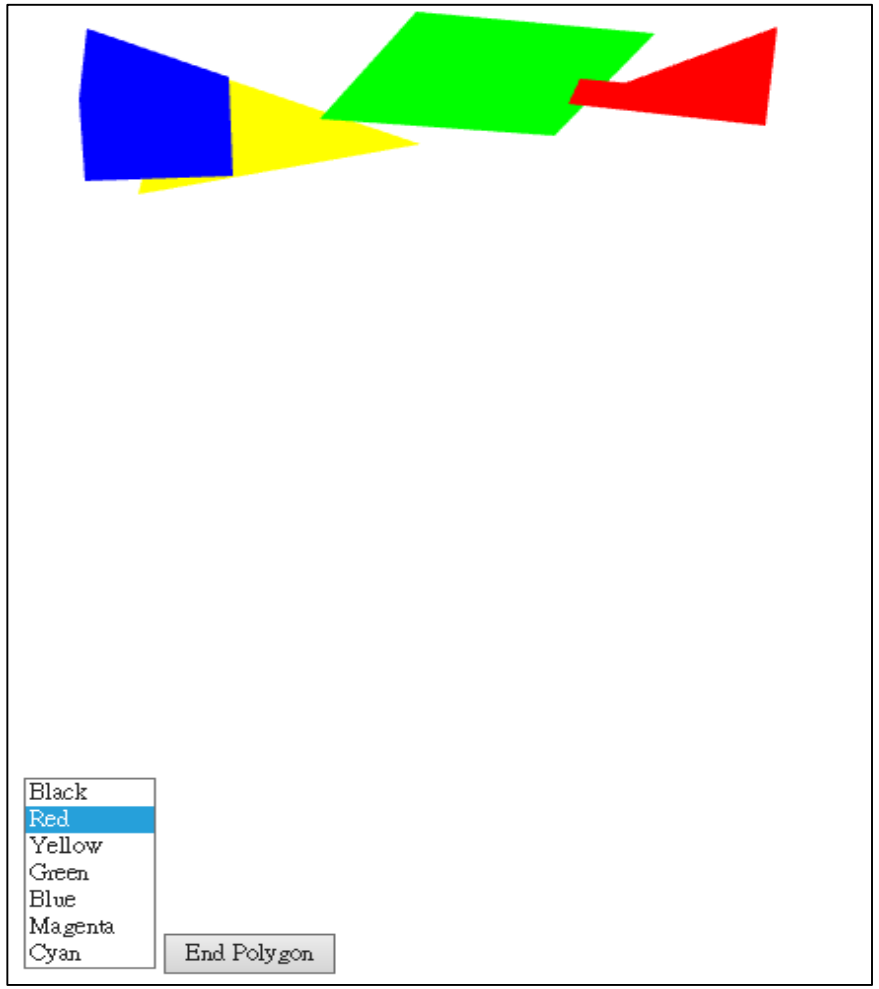
Each rectangle

Vertices: t1, t3, t2, t4

Colors: c, c, c, c


```
for(var i = 0; i<index; i+=4)
    gl.drawArrays( gl.TRIANGLE_FAN, i, 4 );
```

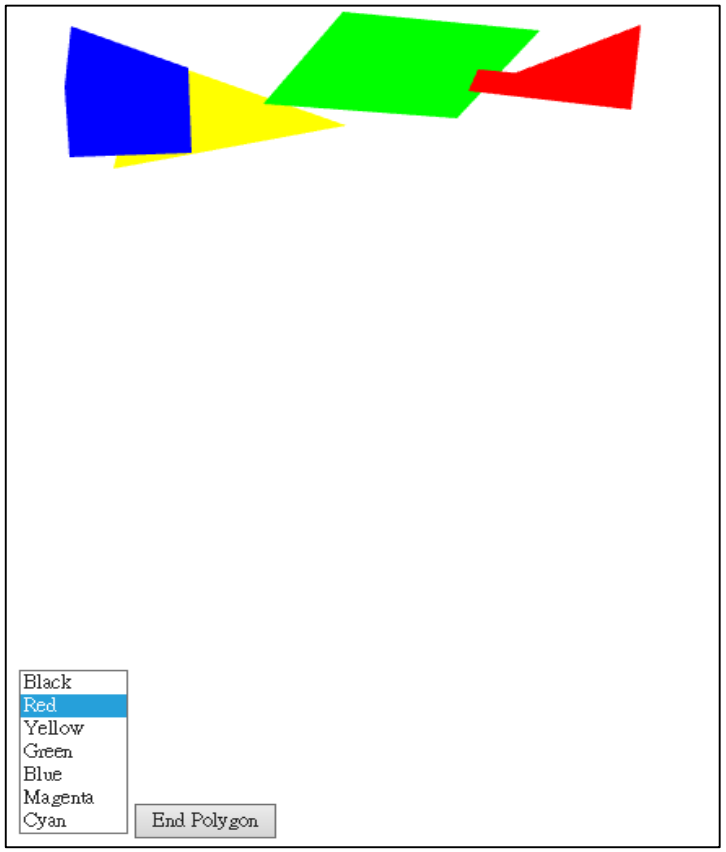
Sample Programs: cad2.html, cad2.js



Polygon drawing. Each mouse click adds a vertex. End a polygon with button click.

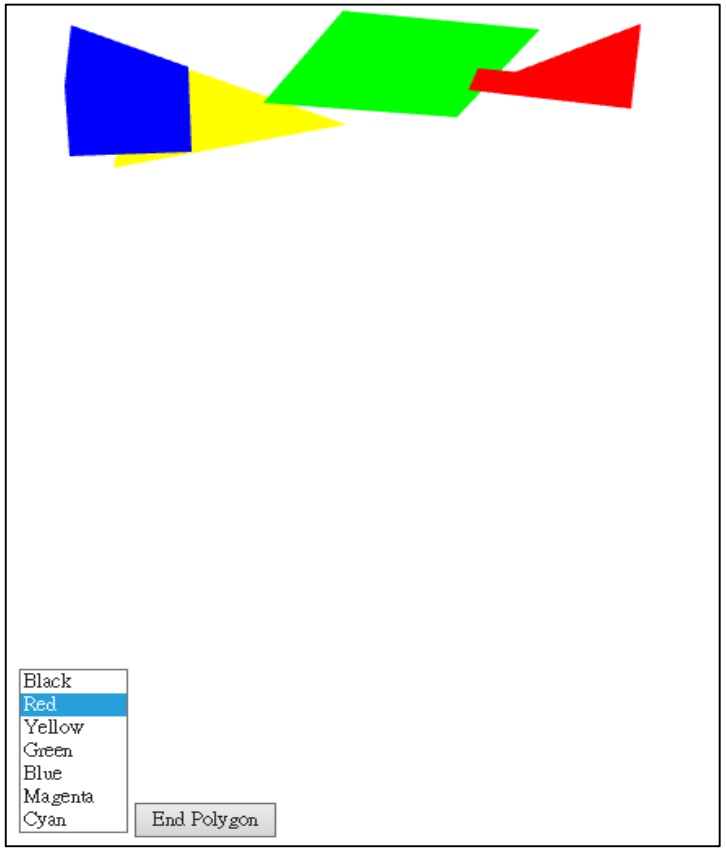
cad2.html (1/4)

```
<html>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute vec4 vColor;

varying vec4 fColor;

void
main()
{
    gl_Position = vPosition;
    fColor = vColor;
}
</script>
```



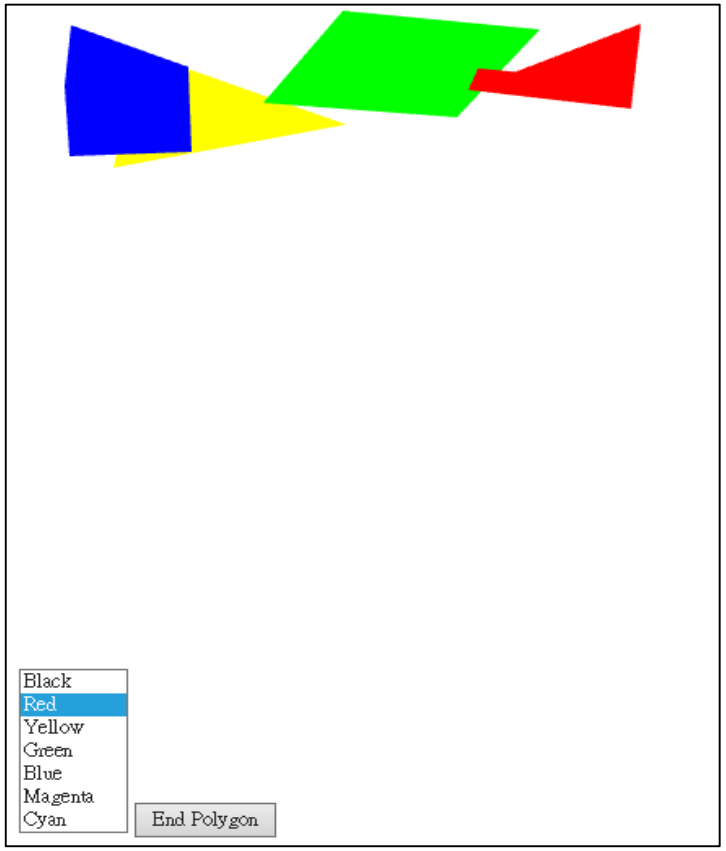
cad2.html (2/4)

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

precision mediump float;

varying vec4 fColor;

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

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="cad2.js"></script>
```



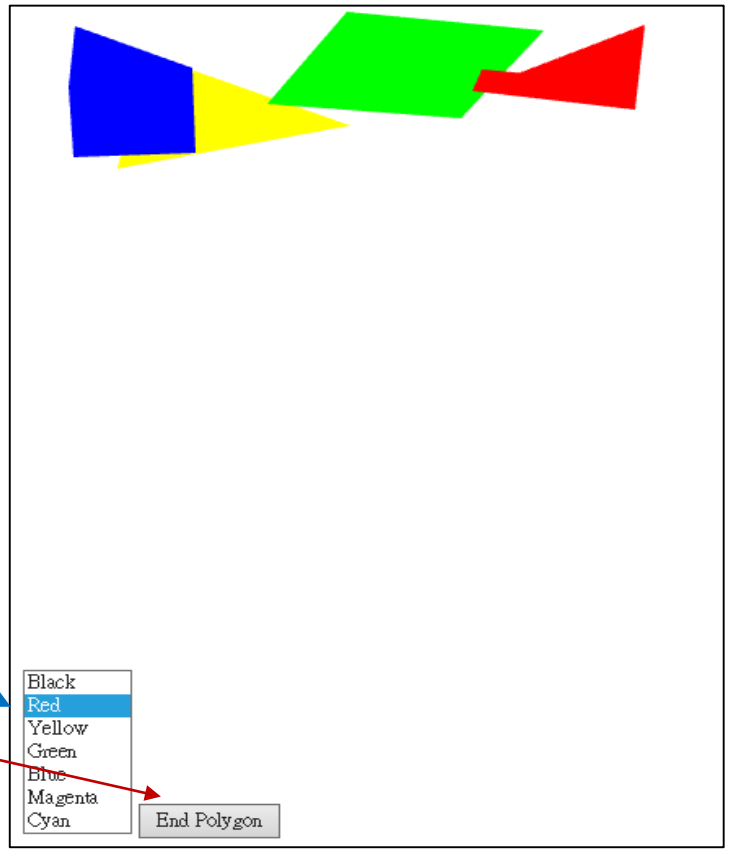
cad2.html (3/4)

```
<body>
<div>
<canvas id="gl-canvas" width="512"" height="512"
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</div>
```



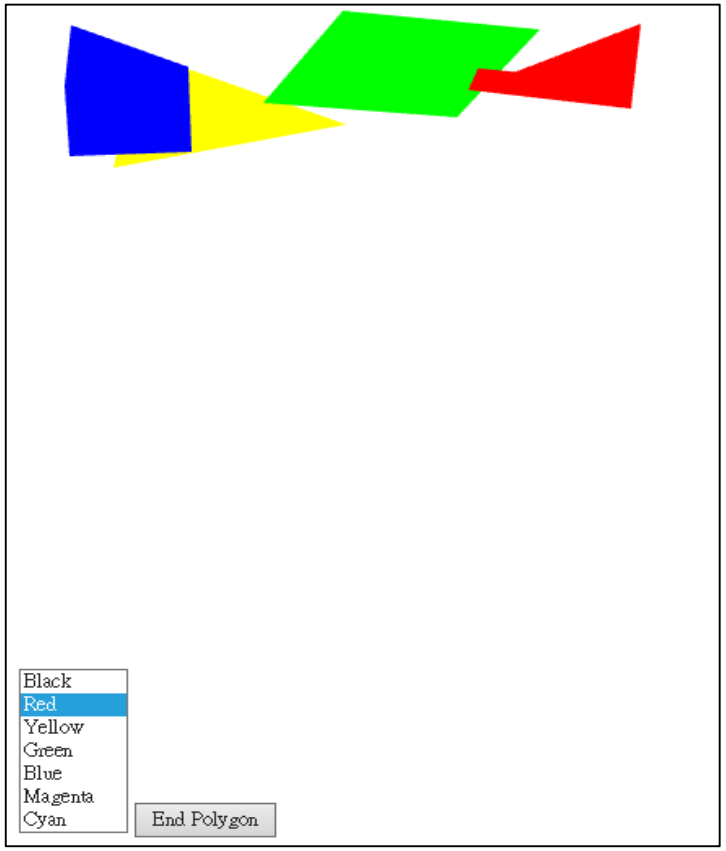
cad2.html (4/4)

```
<div>
<select id = "mymenu" size = "7">
    <option value = "0">Black</option>
    <option value = "1">Red</option>
    <option value = "2">Yellow</option>
    <option value = "3">Green</option>
    <option value = "4">Blue</option>
    <option value = "5">Magenta</option>
    <option value = "6">Cyan</option>
</select>
<button id = "Button1">End Polygon</button>
</div>
</body>
</html>
```



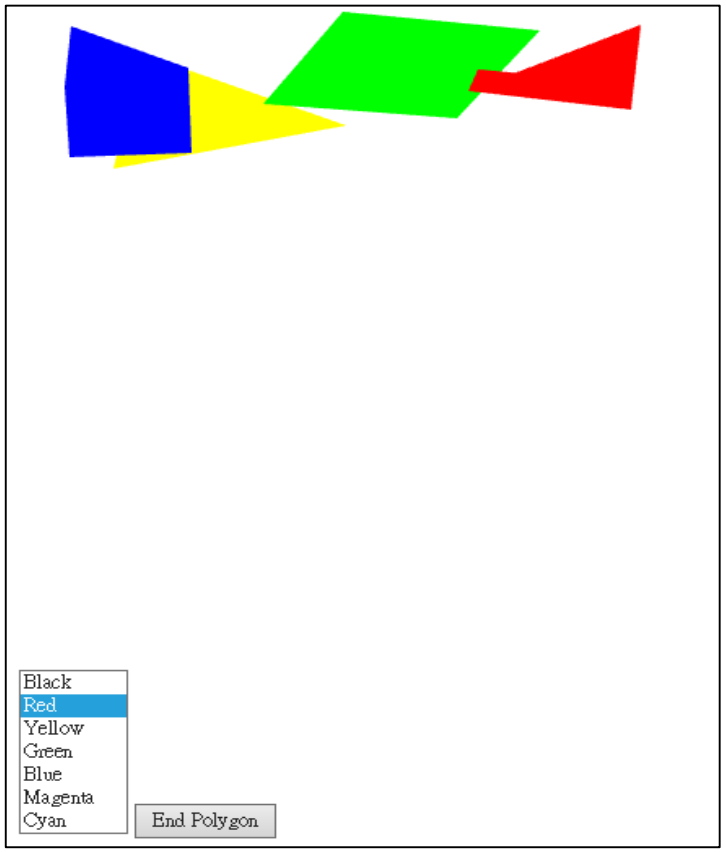
cad2.js (1/6)

```
var canvas;
var gl;

var maxNumVertices  = 200;
var index = 0;

var cindex = 0;

var colors = [
    vec4( 0.0, 0.0, 0.0, 1.0 ),  // black
    vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
    vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
    vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    vec4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    vec4( 0.0, 1.0, 1.0, 1.0 )   // cyan
];
```



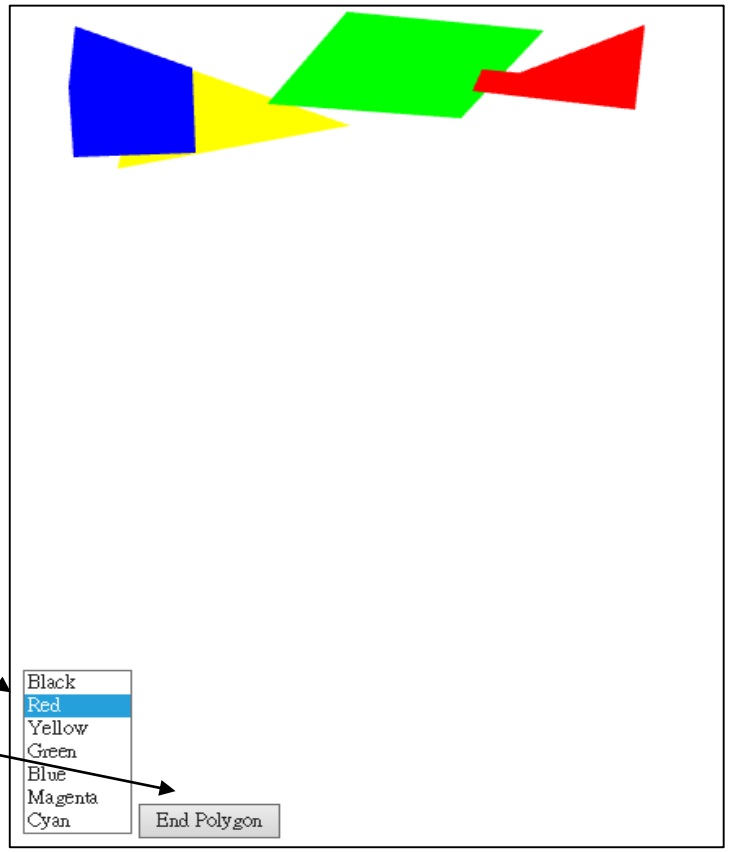
cad2.js (2/6)

```
var t;
var numPolygons = 0;
var numIndices = [];
numIndices[0] = 0;
var start = [0];
```



cad2.js (3/6)

```
window.onload = function init() {
    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    var m = document.getElementById("mymenu");
    m.addEventListener("click", function() { cindex = m.selectedIndex; });

   var a = document.getElementById("Button1")
    a.addEventListener("click", function() {
    numPolygons++;
    numIndices[numPolygons] = 0;
    start[numPolygons] = index;
    render();
    });  // end of a.addEventListener (Button1)
```



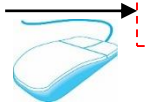
cad2.js (4/6)

$(x_w, y_w) \rightarrow (x, y)$

Screen coordinates World coordinates

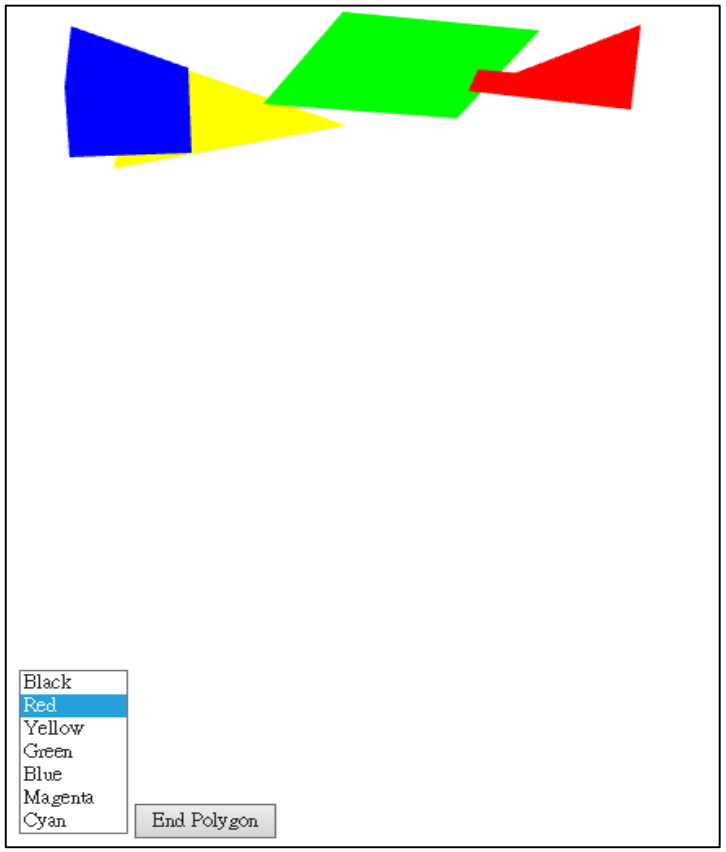
$$x = -1 + \frac{2 * x_w}{w}$$

$$y = -1 + \frac{2 * (h - y_w)}{h}$$



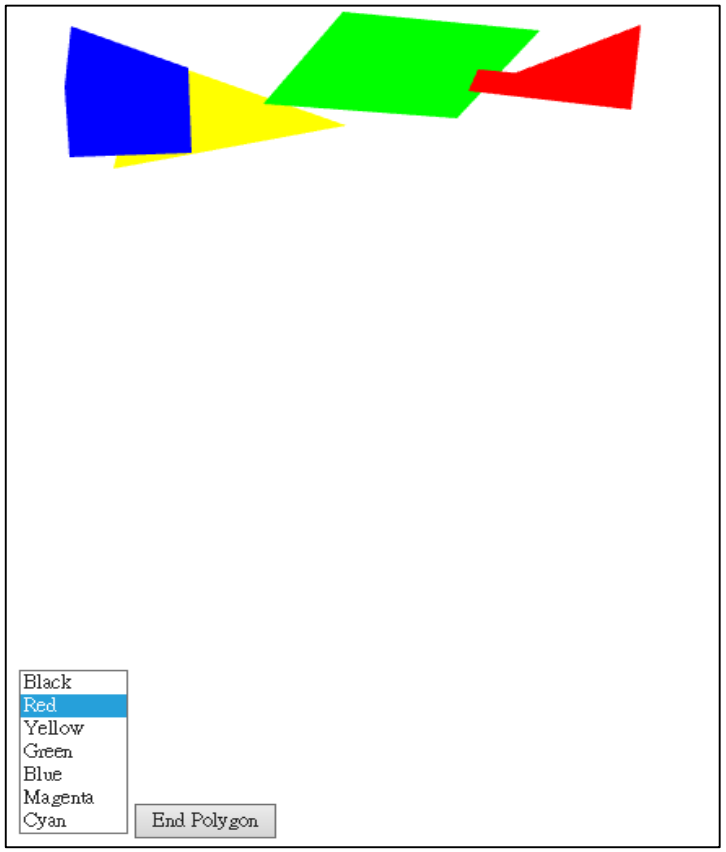
```
canvas.addEventListener("mousedown", function(){
      t  = vec2(2*event.clientX/canvas.width-1,
                2*(canvas.height-event.clientY)/canvas.height-1);
      gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
      gl.bufferSubData(gl.ARRAY_BUFFER, 8*index, flatten(t));

      t = vec4(colors[cindex]);
      gl.bindBuffer( gl.ARRAY_BUFFER, cBufferId );
      gl.bufferSubData(gl.ARRAY_BUFFER, 16*index, flatten(t));

      numIndices[numPolygons]++;
      index++;
  } ); // end of canvas.addEventListener (Mouse)

  gl.viewport( 0, 0, canvas.width, canvas.height );
  gl.clearColor( 0.8, 0.8, 0.8, 1.0 );
  gl.clear( gl.COLOR_BUFFER_BIT );
```

mouse



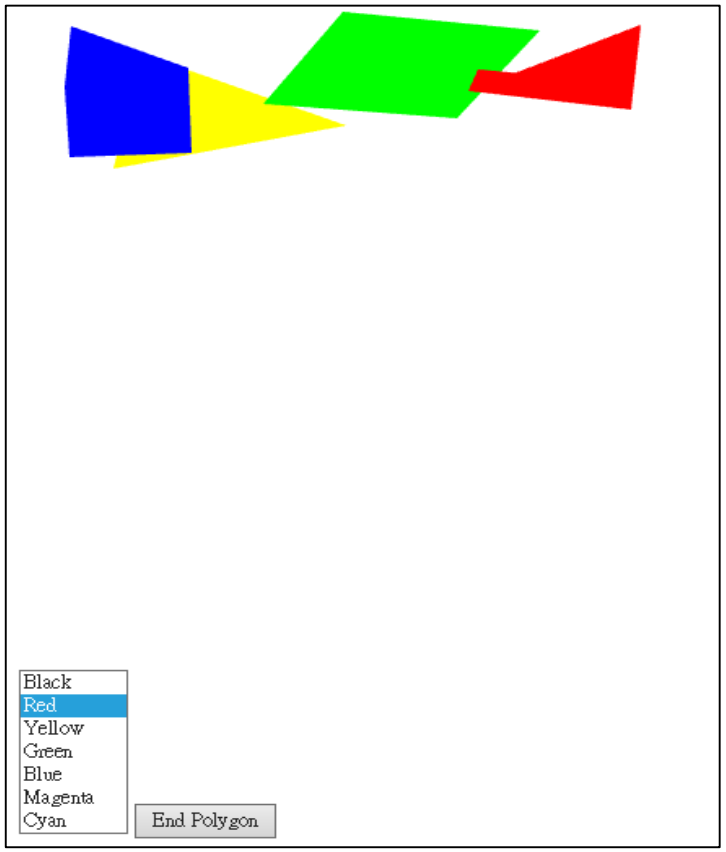
cad2.js (5/6)

```
//  Load shaders and initialize attribute buffers
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, 8*maxNumVertices, gl.STATIC_DRAW );
var vPos = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPos, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPos );

var cBufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBufferId );
gl.bufferData( gl.ARRAY_BUFFER, 16*maxNumVertices, gl.STATIC_DRAW );
var vColor = gl.getAttribLocation( program, "vColor" );
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vColor );
}  // end of window.onload
```

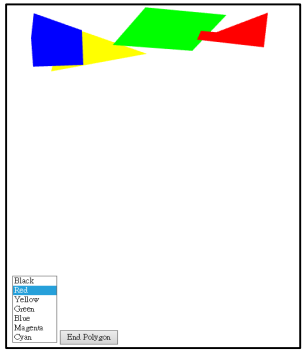


cad2.js (6/6)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT );

    for(var i=0; i<numPolygons; i++) {
        gl.drawArrays( gl.TRIANGLE_FAN, start[i], numIndices[i] );
    }
}
```



cad2.js : Data Structures



numPolygons=3

```
for(var i=0; i<numPolygons; i++) {
    gl.drawArrays( gl.TRIANGLE_FAN, start[i], numIndices[i] );
```

polygonId	0	1	2	3	4	5	6
numIndices	3	4	5				
start	0	3	7				

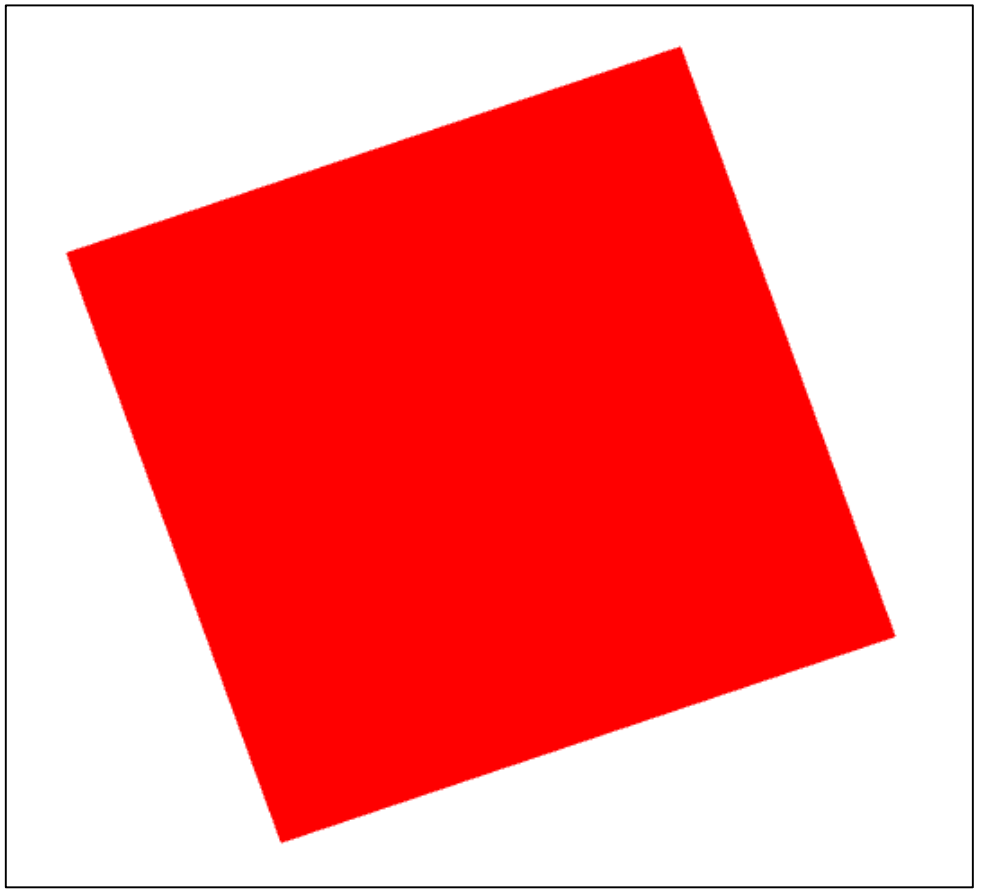
index=12

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
bufferId	0 8 bytes	8	16	24	32	40	48	56	64	72	80	88	96	
cBufferId	0 16 bytes	16	32	48	64	80	96	112	128	144	160	176	192	

(x,y) coordinates

RGBA values

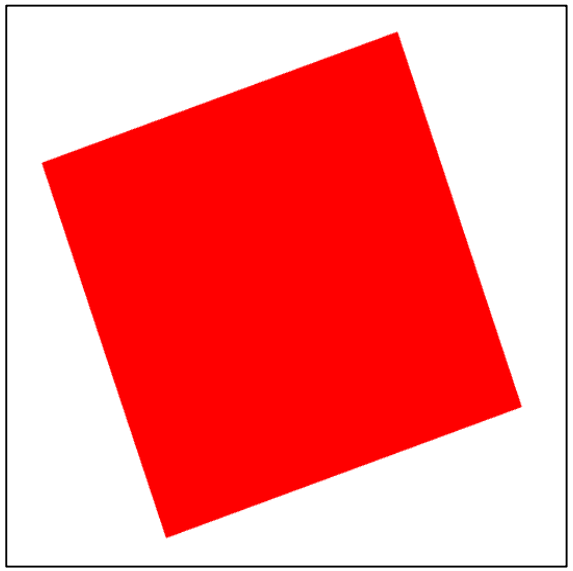
Sample Programs: rotatingSquare1.html, rotatingSquare1.js



Rotating square with no interaction

rotatingSquare1.html (1/3)

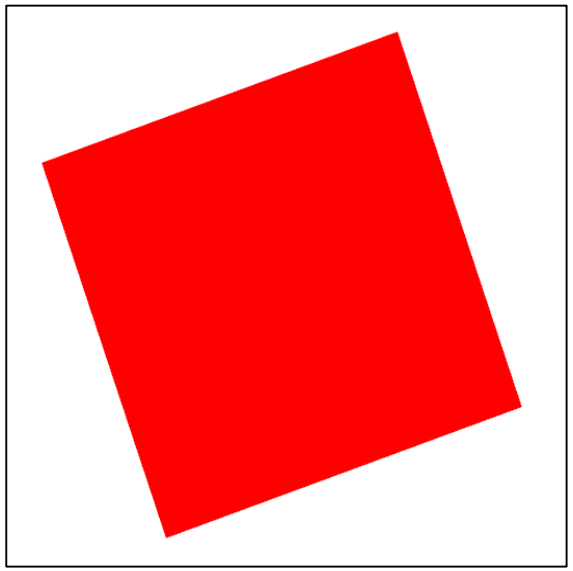
```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<title>Rotating Square</title>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
uniform float theta;

void main()
{   float s = sin( theta );
    float c = cos( theta );
    gl_Position.x = c * vPosition.x  -  s * vPosition.y;
    gl_Position.y = c * vPosition.y  + s * vPosition.x;
    gl_Position.z = 0.0;
    gl_Position.w = 1.0;
}
</script>
```



$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} vPosition.x \\ vPosition.y \end{bmatrix} = \begin{bmatrix} gl\_Position.x & gl\_Position.y \end{bmatrix}$$

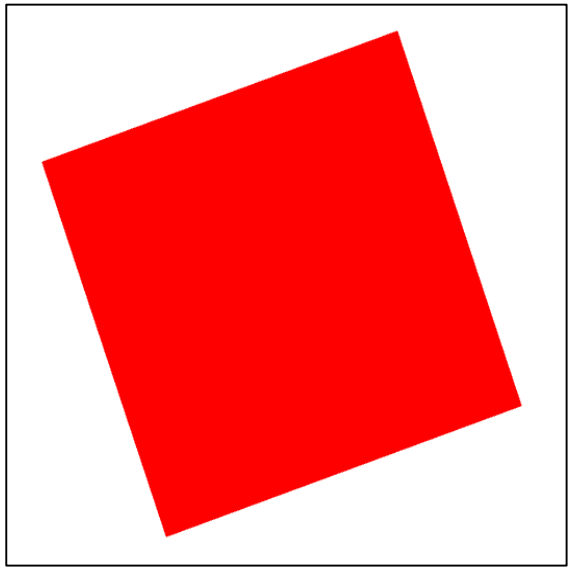
rotatingSquare1.html (2/3)

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

precision mediump float;

void main()
{
    gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
</script>

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="rotatingSquare1.js"></script>
</head>
```



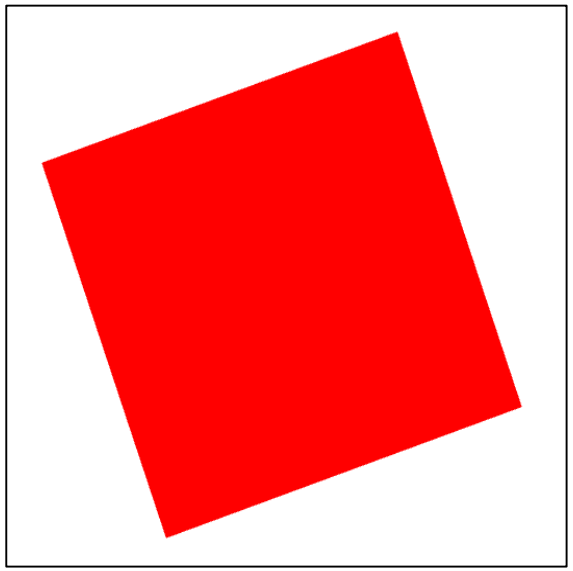
rotatingSquare1.html (3/3)

```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



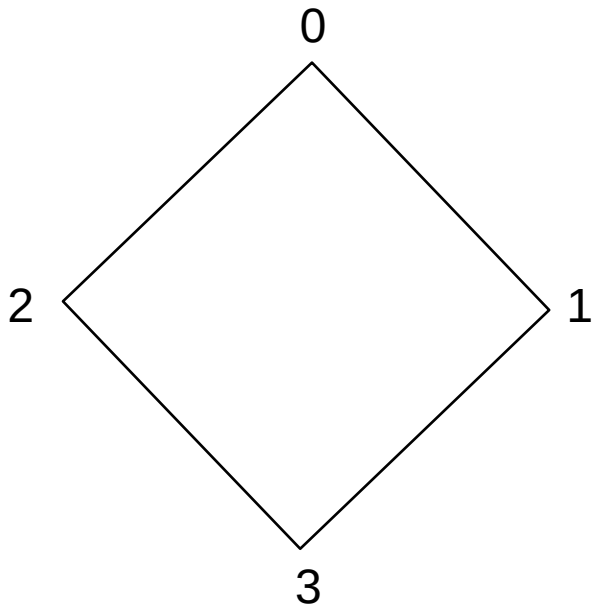
rotatingSquare1.js (1/4)

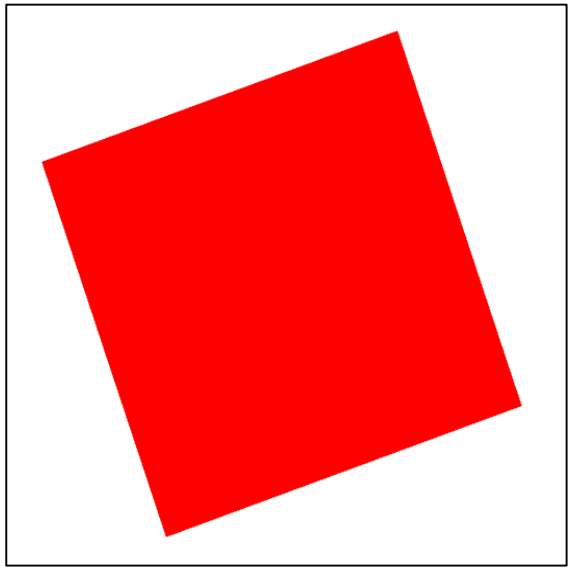
```
var canvas;
var gl;

var theta = 0.0;
var thetaLoc;

window.onload = function init()
{
    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    //  Configure WebGL
    //
    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
```



rotatingSquare1.js (2/4)

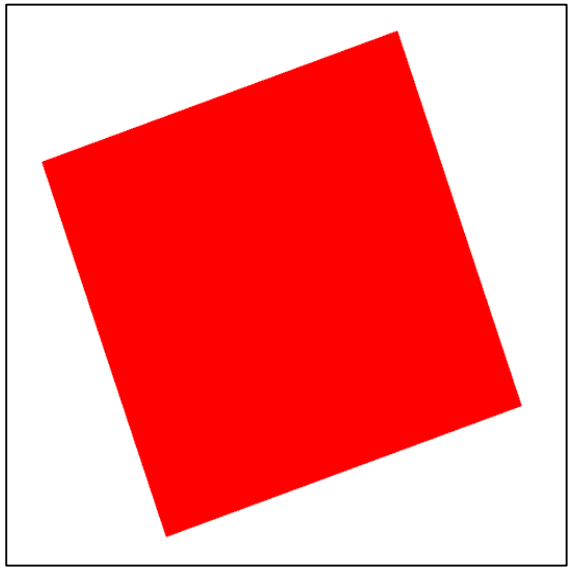
```
//  Load shaders and initialize attribute buffers
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );

    var vertices = [
        vec2(  0,  1 ),
        vec2(  1,  0 ),
        vec2( -1,  0 ),
        vec2(  0, -1 )
    ];
```





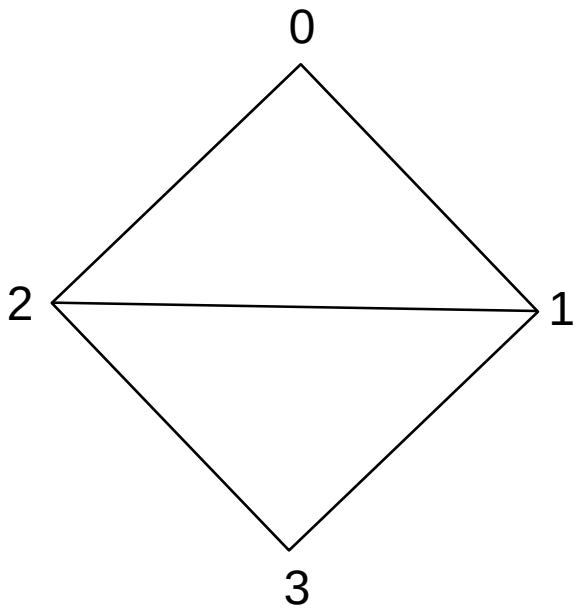
rotatingSquare1.js (3/4)

```
    // Load the data into the GPU
    var bufferId = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
    gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

    // Associate out shader variables with our data buffer
    var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );

    thetaLoc = gl.getUniformLocation( program, "theta" );

    render();
};   // end of window.onload
```



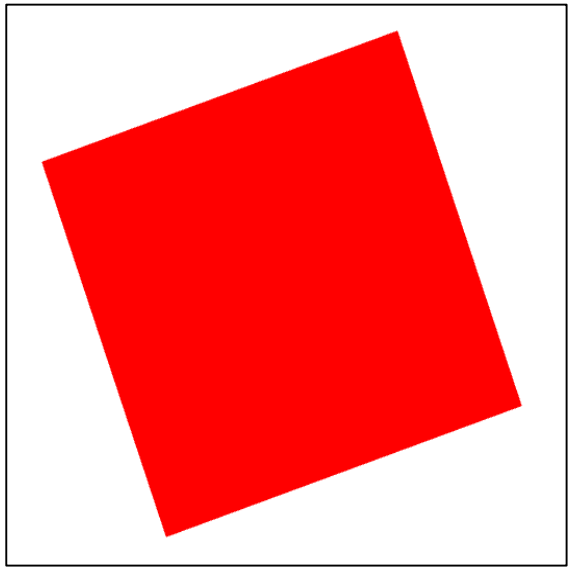
rotatingSquare1.js (4/4)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT );

    theta += 0.1;
    gl.uniform1f( thetaLoc, theta );

    gl.drawArrays( gl.TRIANGLE_STRIP, 0, 4 );

    window.requestAnimFrame(render);
}
```

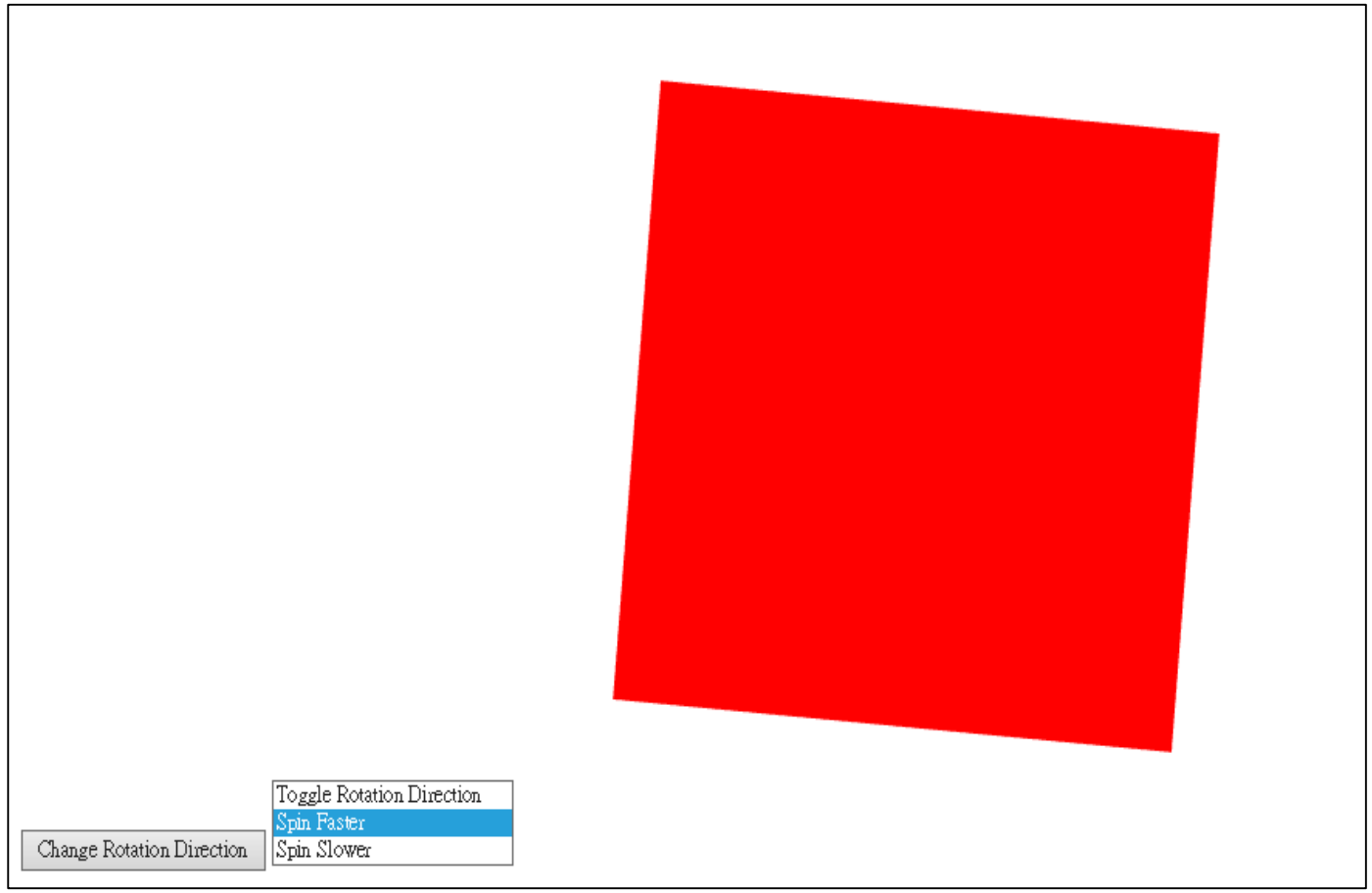


vPosition: 0, 1, 2, 3

Gl.TRIANGLE_STRIP



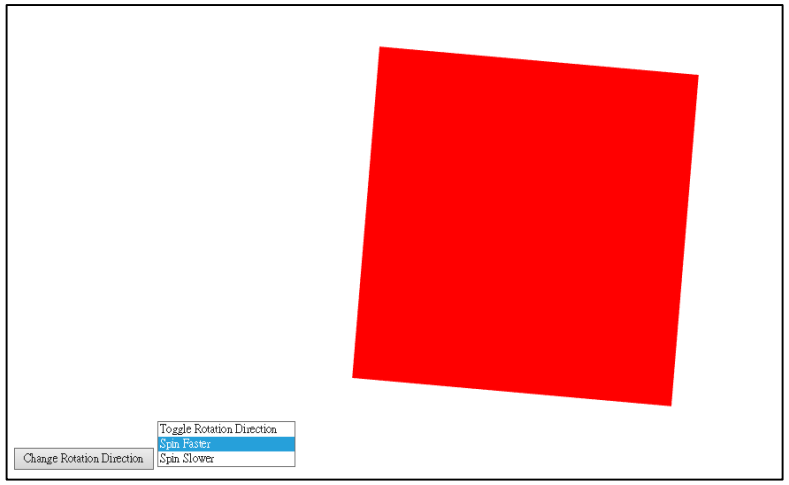
Sample Programs: rotatingSquare2.html, rotatingSquare2.js



Rotating Square with speed and direction of rotation controlled with buttons and menus

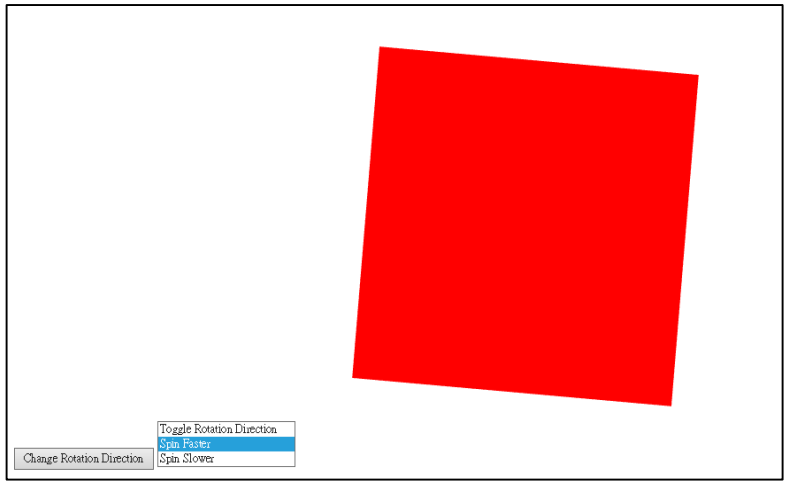
rotatingSquare2.html (1/3)

```
<!DOCTYPE html>
<html>
<title>Rotating Square</title>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
uniform float theta;

void main()
{   float s = sin( theta );
    float c = cos( theta );

    gl_Position.x = -s * vPosition.x + c * vPosition.y;
    gl_Position.y =  s * vPosition.y + c * vPosition.x;
    gl_Position.z = 0.0;
    gl_Position.w = 1.0;
}
</script>
```



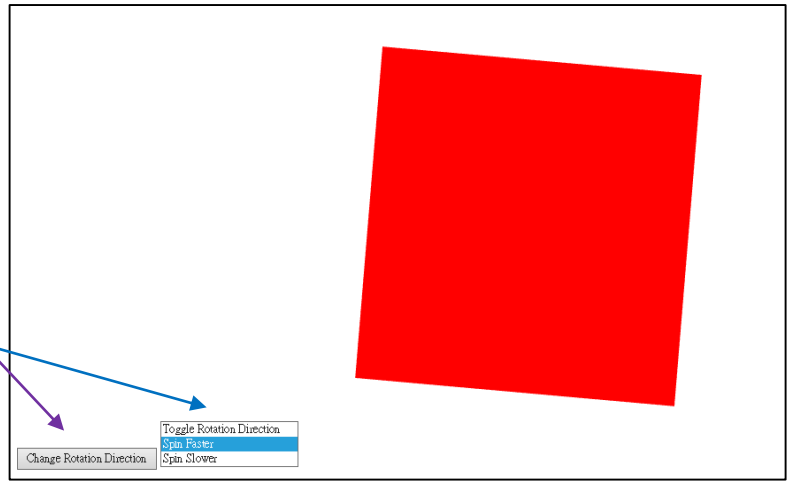
rotatingSquare2.html (2/3)

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

precision mediump float;

void main()
{
    gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
</script>

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="rotatingSquare2.js"></script>
</head>
```



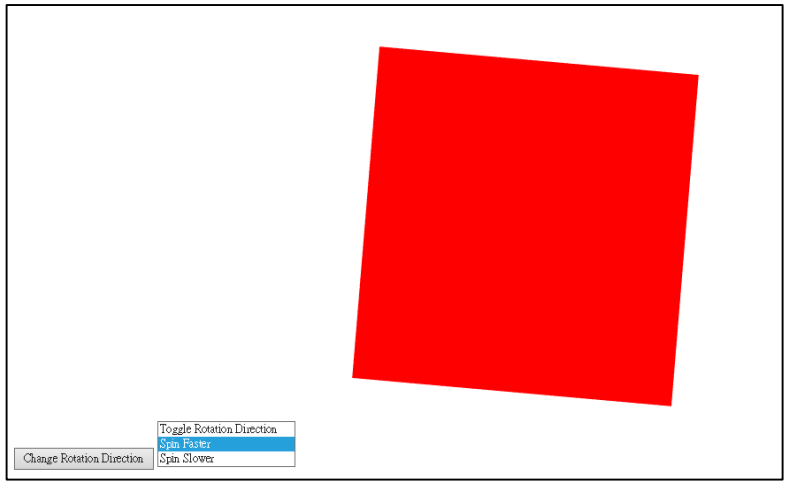
rotatingSquare2.html (3/3)

```
<body>
<button id="Direction">Change Rotation Direction</button>

<select id="Controls" size="3">
    <option value="0">Toggle Rotation Direction</option>
    <option value="1">Spin Faster</option>
    <option value="2">Spin Slower</option>
</select>

<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



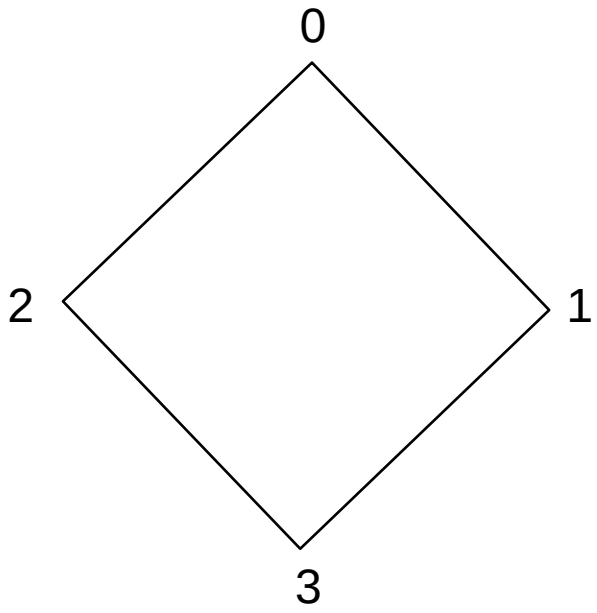
rotatingSquare2.js (1/6)

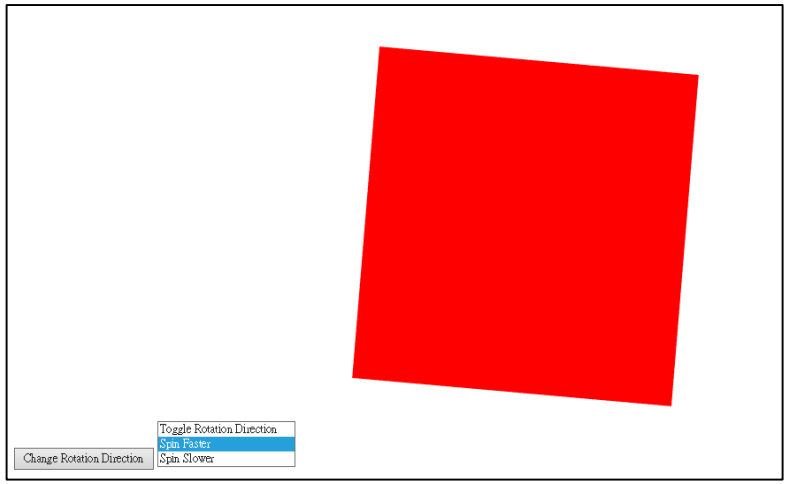
```
var gl;

var theta = 0.0;
var thetaLoc;

var delay = 100;
var direction = true;

window.onload = function init()
{
    var canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    //  Configure WebGL
    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
```



rotatingSquare2.js (2/6)

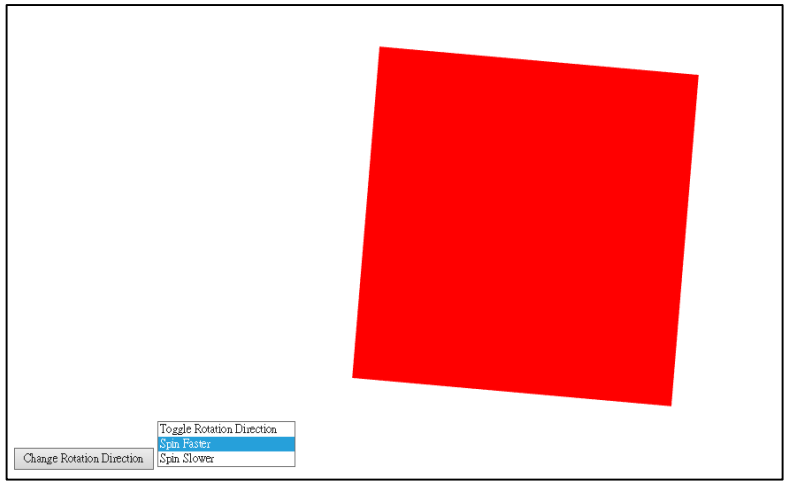
```
//  Load shaders and initialize attribute buffers

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

var vertices = [
    vec2(  0,  1 ),
    vec2(  1,  0 ),
    vec2( -1,  0 ),
    vec2(  0, -1 )
];
```





rotatingSquare2.js (3/6)

```
// Load the data into the GPU

var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);

// Associate out shader variables with our data buffer

var vPosition = gl.getAttribLocation( program, "vPosition");
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);

thetaLoc = gl.getUniformLocation( program, "theta" );
```



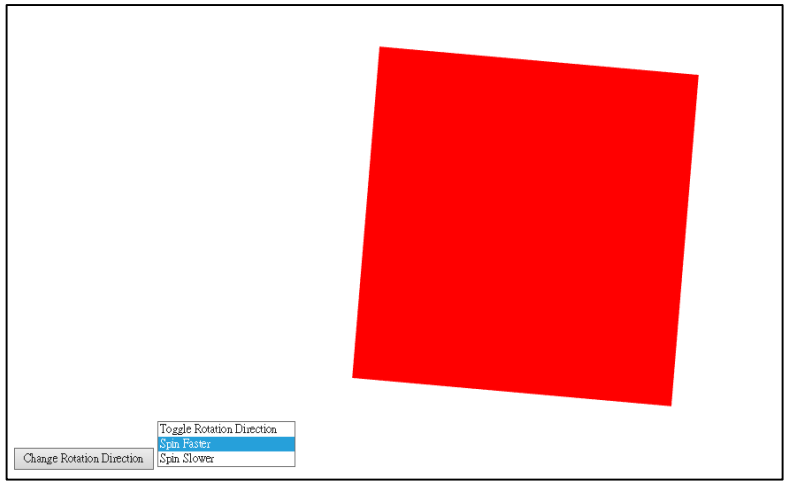
rotatingSquare2.js (4/6)

```
// Initialize event handlers
document.getElementById("Direction").onclick = function () {
    direction = !direction;
};

document.getElementById("Controls" ).onclick = function(event) {
    switch( event.srcElement.index ) {
      case 0:
        direction = !direction;
        break;
     case 1:
        delay /= 2.0;
        break;
     case 2:
        delay *= 2.0;
        break;
   }
};
```

rotatingSquare2.js (5/6)

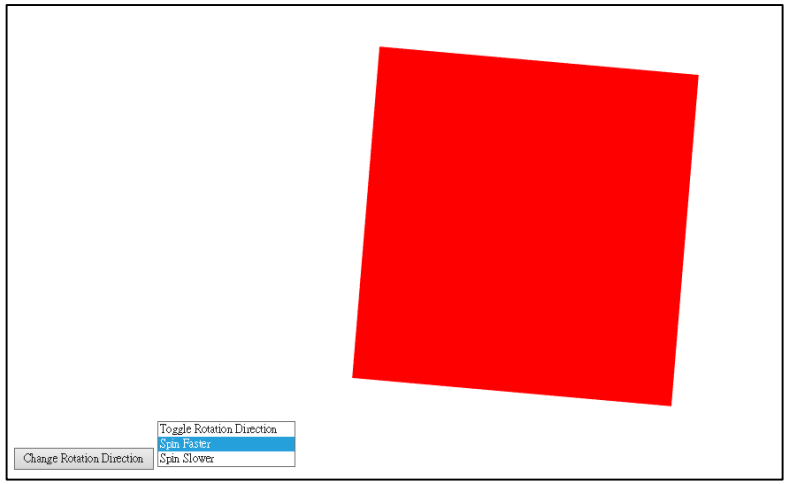
```
window.onkeydown = function(event) {
        var key = String.fromCharCode(event.keyCode);
        switch(key) {
          case '1':
            direction = !direction;
            break;

          case '2':
            delay /= 2.0;
            break;

          case '3':
            delay *= 2.0;
            break;
        }
    };
    render();
};   // end of window.onload
```

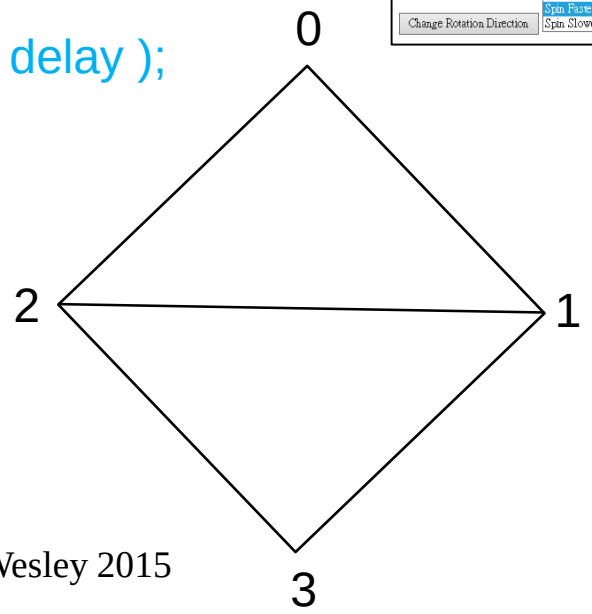
keyboard



rotatingSquare2.js (6/6)

```
function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT );

    theta += (direction ? 0.1 : -0.1);
    gl.uniform1f(thetaLoc, theta);

    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

    setTimeout( function (){requestAnimFrame(render);}, delay );
}
```

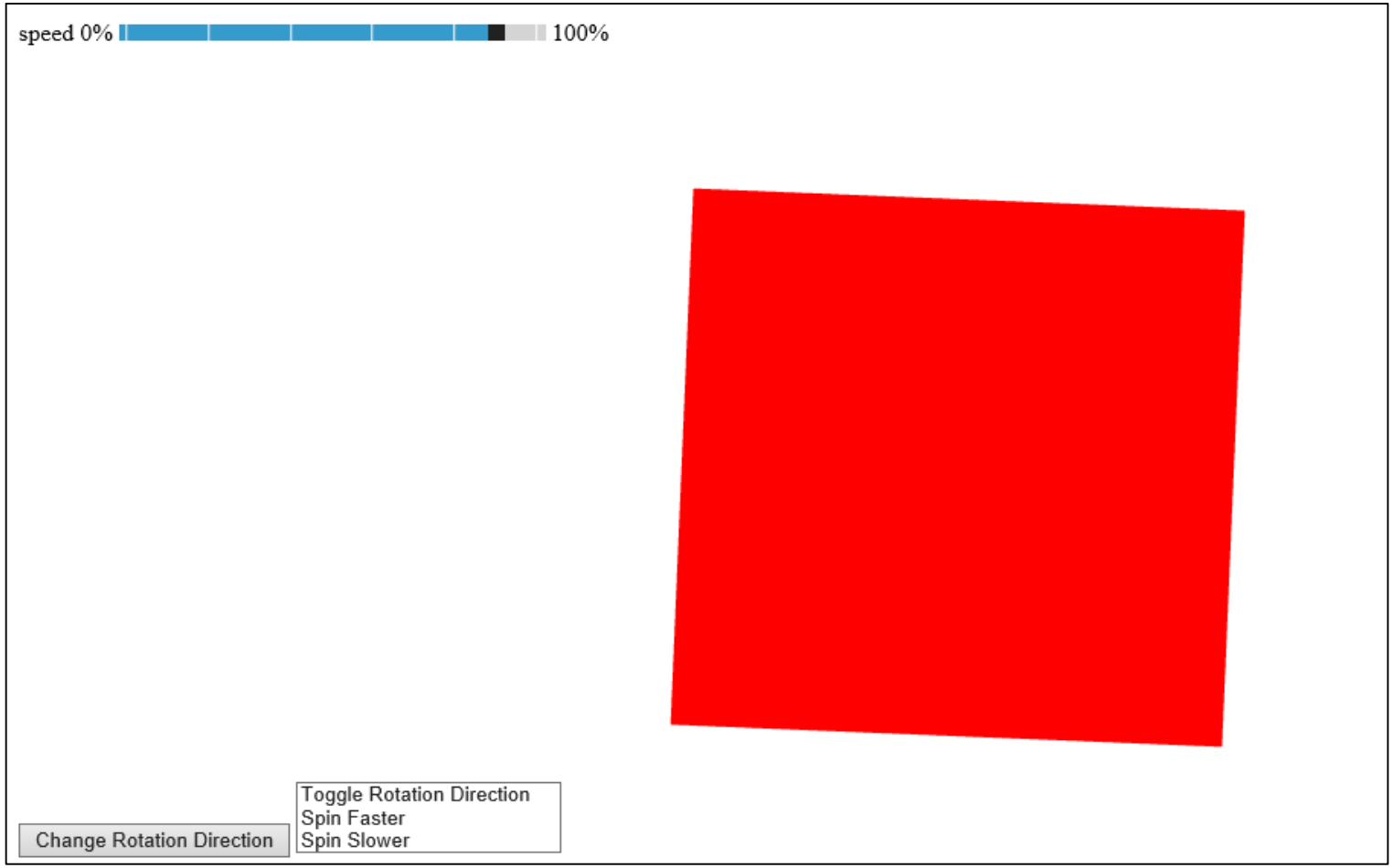




vPosition: 0, 1, 2, 3

Gl.TRIANGLE_STRIP

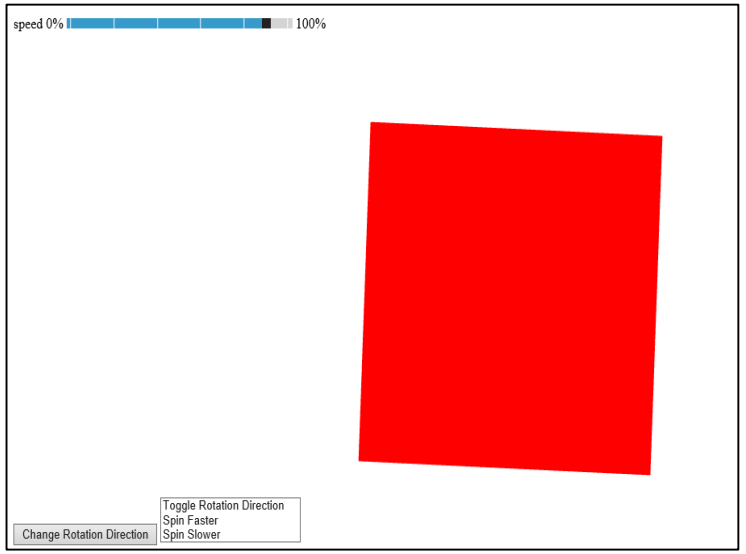
Sample Programs: rotatingSquare3.html, rotatingSquare3.js



Same as rotatingSquare2 with slider

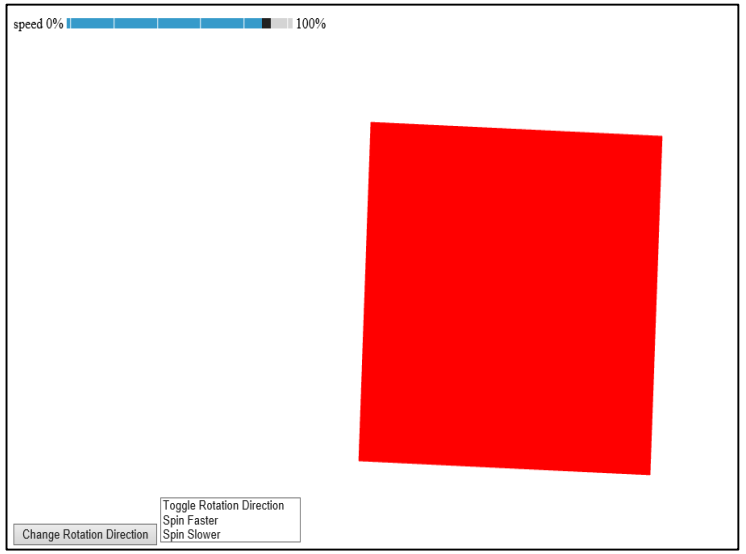
rotatingSquare3.html (1/4)

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<title>Rotating Square</title>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
uniform float theta;

void main()
{   float s = sin( theta );
    float c = cos( theta );

    gl_Position.x = -s * vPosition.x + c * vPosition.y;
    gl_Position.y =  s * vPosition.y + c * vPosition.x;
    gl_Position.z = 0.0;
    gl_Position.w = 1.0;
}
</script>
```



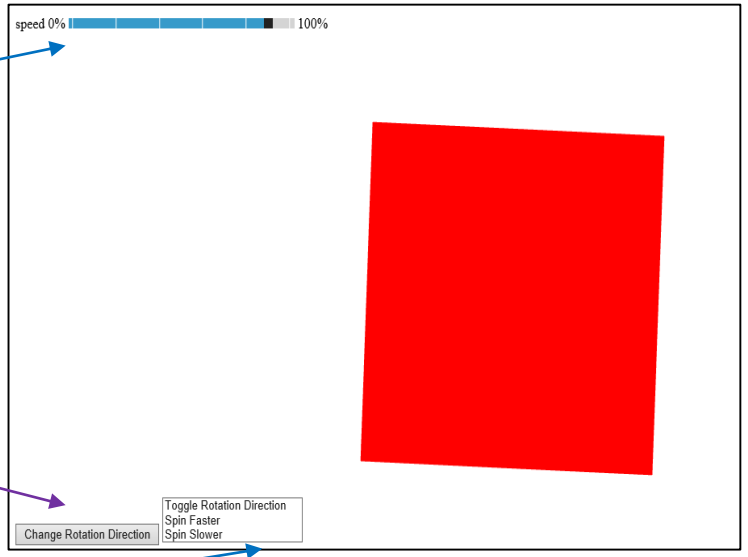
rotatingSquare3.html (2/4)

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

precision mediump float;

void main()
{
    gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
</script>

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="rotatingSquare3.js"></script>
</head>
```



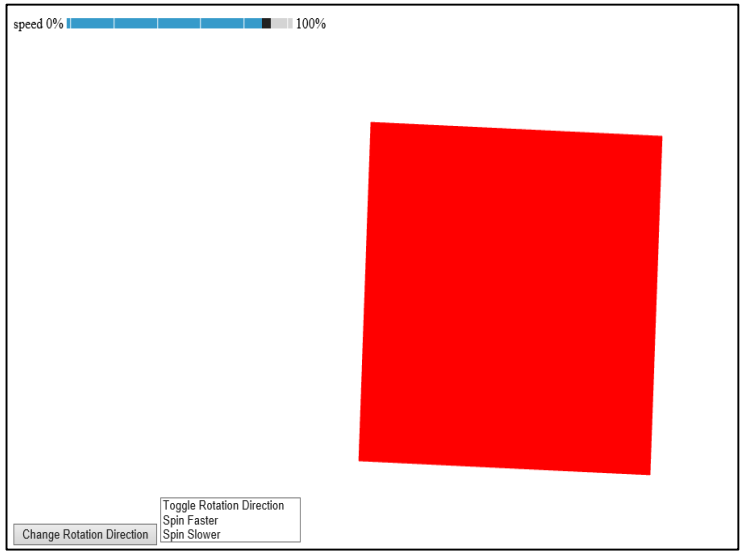
rotatingSquare3.html (3/4)

```
<body>
<div>
speed 0% <input id="slider" type="range"
 min="0" max="100" step="10" value="50" />
 100%
</div>

<button id="Direction">Change Rotation Direction</button>

<select id="Controls" size="3">
    <option value="0">Toggle Rotation Direction</option>
    <option value="1">Spin Faster</option>
    <option value="2">Spin Slower</option>
</select>
```



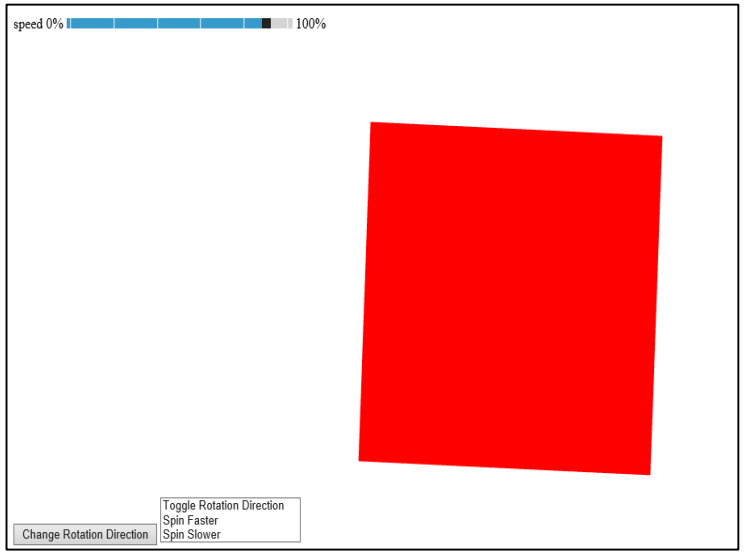
rotatingSquare3.html (4/4)

```
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



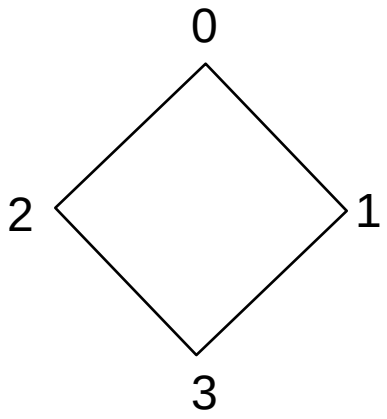
rotatingSquare3.js (1/7)

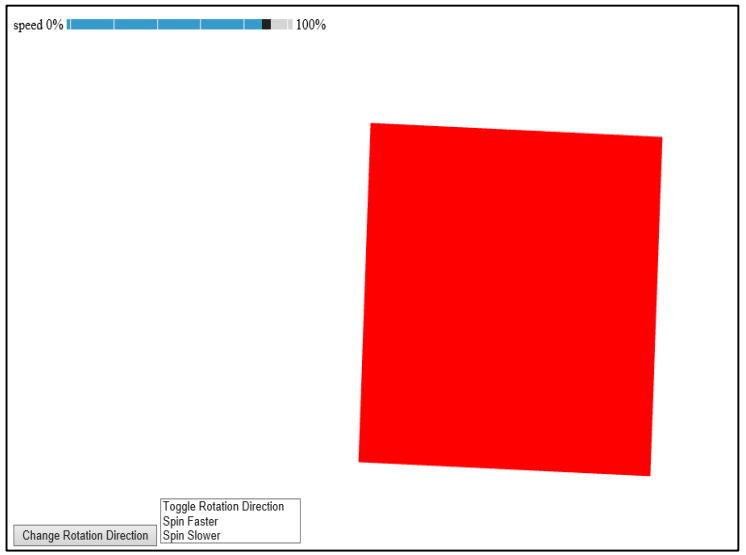
```
var gl;

var theta = 0.0;
var thetaLoc;

var speed = 100;
var direction = true;

window.onload = function init()
{
    var canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }
```



rotatingSquare3.js (2/7)

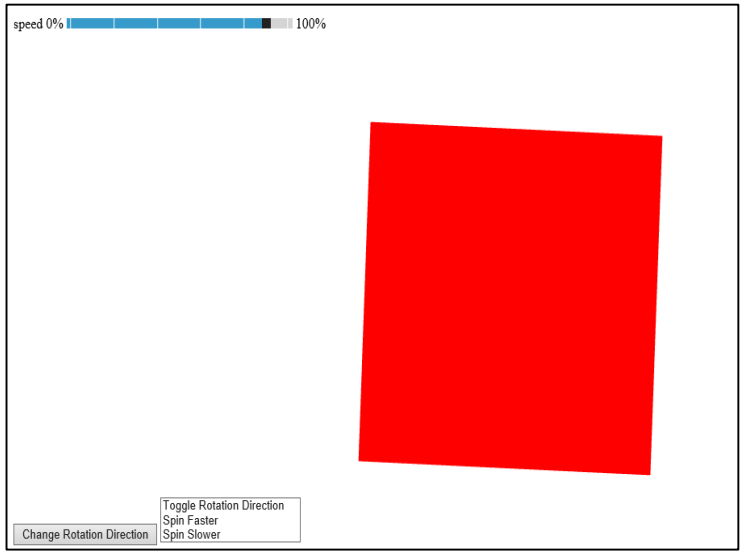
```
//
//  Configure WebGL
//
gl.viewport( 0, 0, canvas.width, canvas.height );
gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

//  Load shaders and initialize attribute buffers

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

var vertices = [
    vec2( 0,  1),
    vec2( 1,  0),
    vec2(-1,  0),
    vec2( 0, -1)
];
```





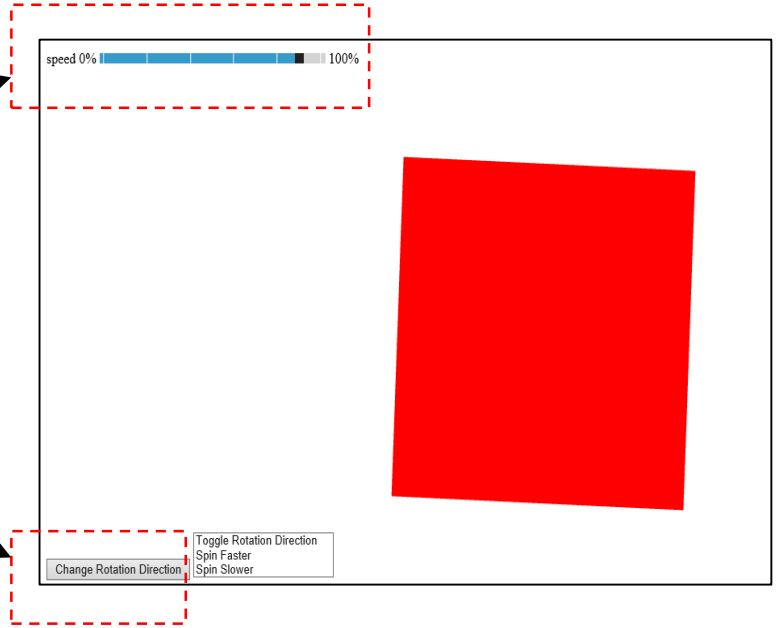
rotatingSquare3.js (3/7)

```
// Load the data into the GPU

var bufferId = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, bufferId);
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);

// Associate out shader variables with our data buffer

var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray(vPosition);

thetaLoc = gl.getUniformLocation(program, "theta");
```



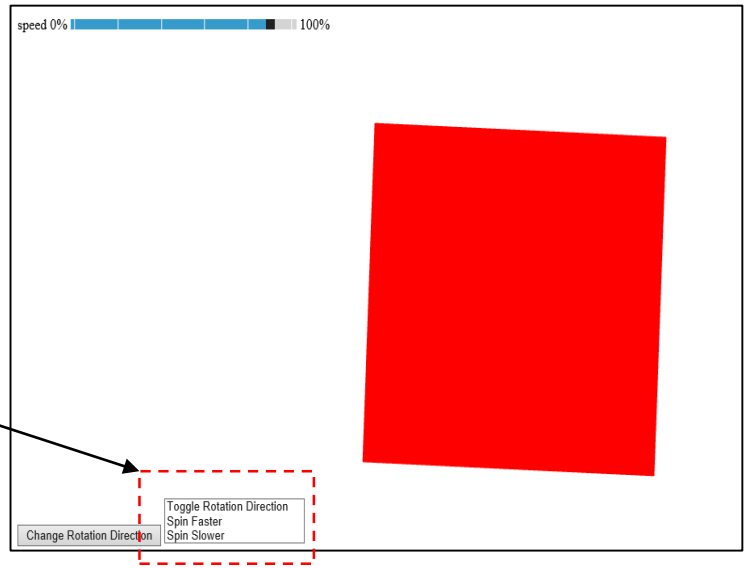
rotatingSquare3.js (4/7)

```
// Initialize event handlers

document.getElementById("slider").onchange = function() {
    speed = 100 - event.srcElement.value;
};

document.getElementById("Direction").onclick = function () {
    direction = !direction;
};
```



rotatingSquare3.js (5/7)

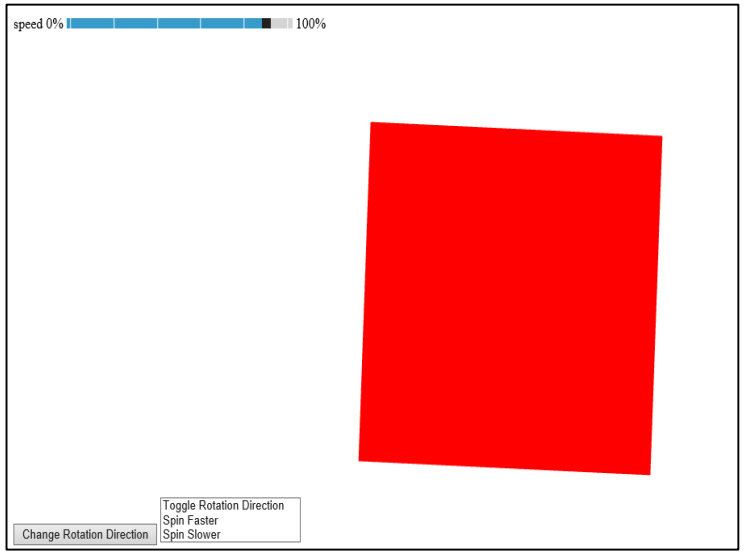
```
document.getElementById("Controls").onclick = function( event) {
   switch(event.srcElement.index) {
     case 0:
       direction = !direction;
       break;

     case 1:
       speed /= 2.0;
       break;

     case 2:
       speed *= 2.0;
       break;
   }
};
```



rotatingSquare3.js (6/7)

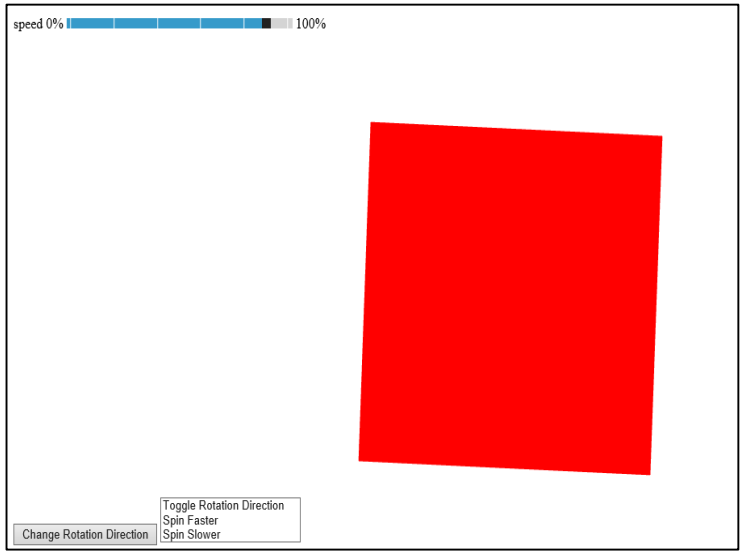
```
window.onkeydown = function( event ) {
    var key = String.fromCharCode(event.keyCode);
    switch( key ) {
      case '1':
        direction = !direction;
        break;

      case '2':
        speed /= 2.0;
        break;

      case '3':
        speed *= 2.0;
        break;
    }
};
render();
};  // end of window.onload
```

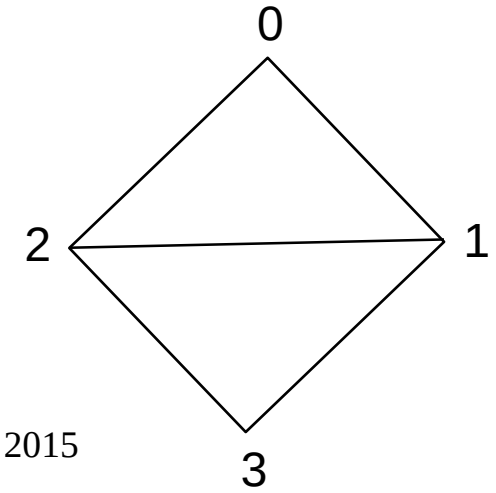
keyboard



rotatingSquare3.js (7/7)

```
function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT );

    theta += (direction ? 0.1 : -0.1);
    gl.uniform1f(thetaLoc, theta);

    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

    setTimeout( function () { requestAnimFrame( render ); }, speed );
}
```

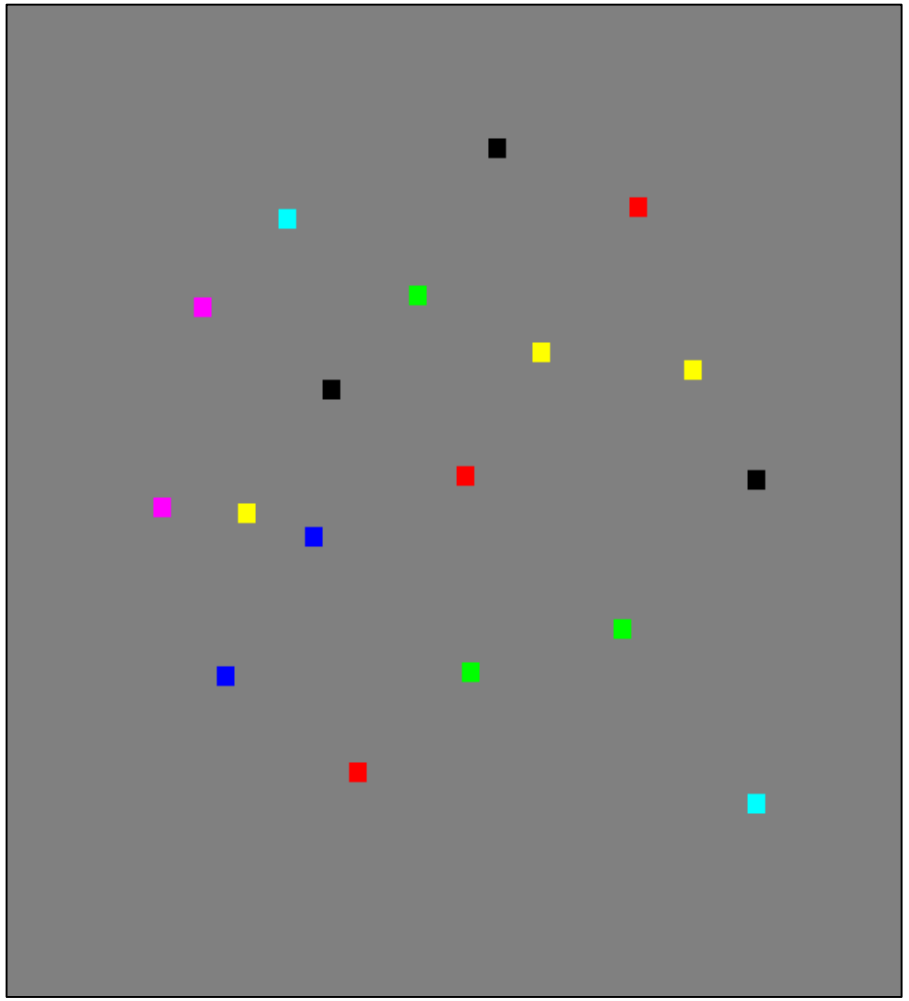




vPosition: 0, 1, 2, 3

Gl.TRIANGLE_STRIP

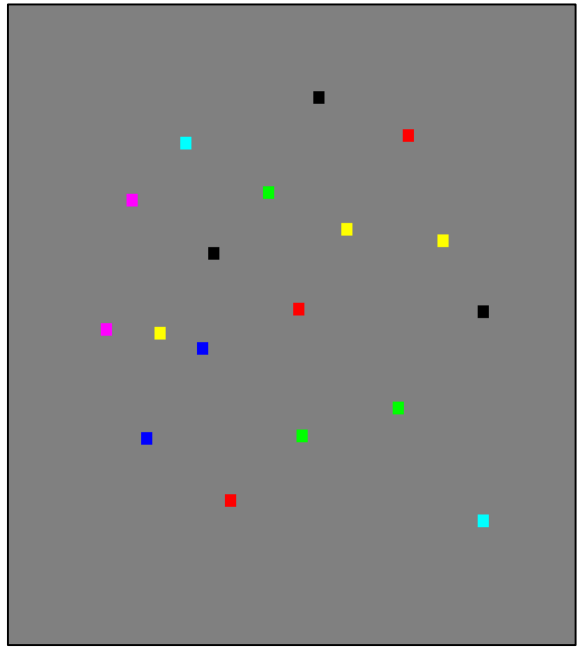
Sample Programs: square.html, square.js



Demo position input by drawing a small colored square at each point on the display where the mouse is clicked

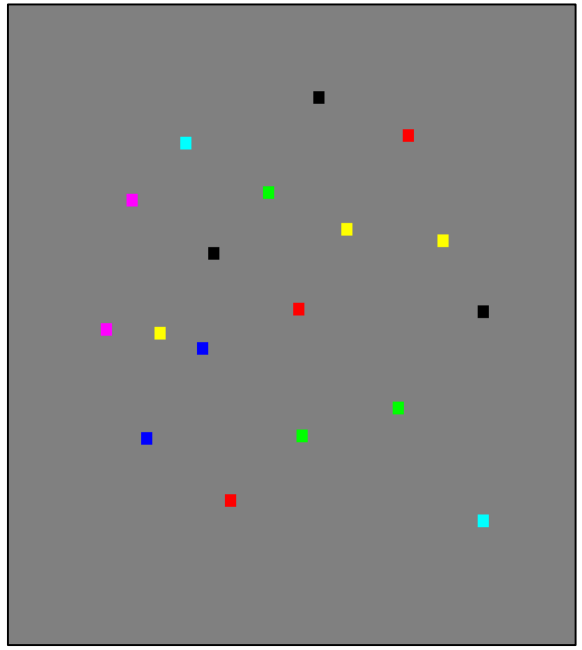
square.html (1/3)

```
<!DOCTYPE html>
<html>

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

attribute vec4 vPosition;
attribute vec4 vColor;

varying vec4 fColor;

void main()
{
    gl_Position = vPosition;
    fColor = vColor;
    gl_PointSize = 10.0;
}
</script>
```



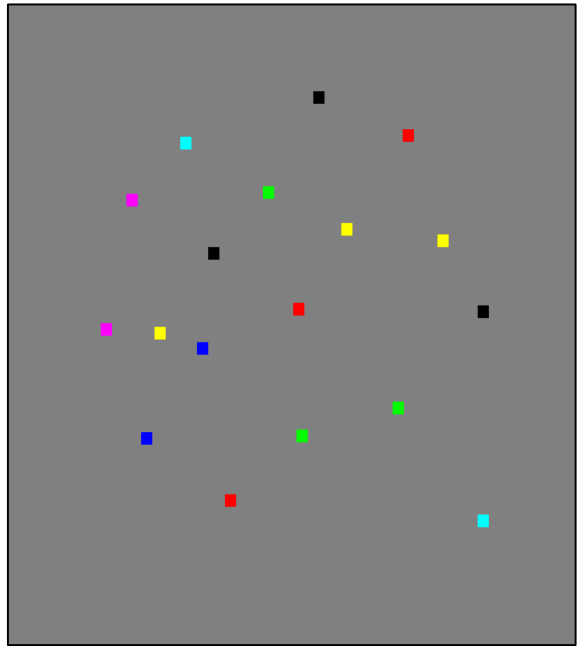
square.html (2/3)

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

precision mediump float;

varying vec4 fColor;

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

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="square.js"></script>
```



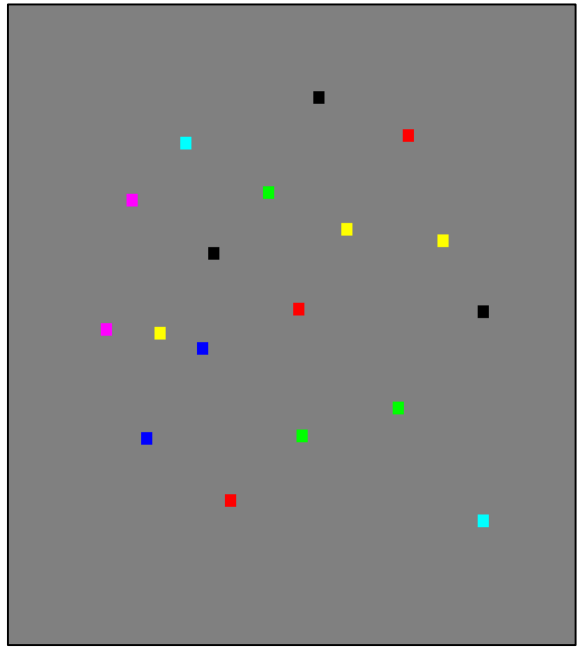
square.html (3/3)

```
<body>
<canvas id="gl-canvas" width="512" height="512">>
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



square.js (1/5)

```
var canvas;
var gl;


var maxNumTriangles = 200;
var maxNumVertices  = 3 * maxNumTriangles;
var index = 0;

var colors = [
    vec4( 0.0, 0.0, 0.0, 1.0 ),  // black
    vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
    vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
    vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    vec4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    vec4( 0.0, 1.0, 1.0, 1.0 )   // cyan
];
```



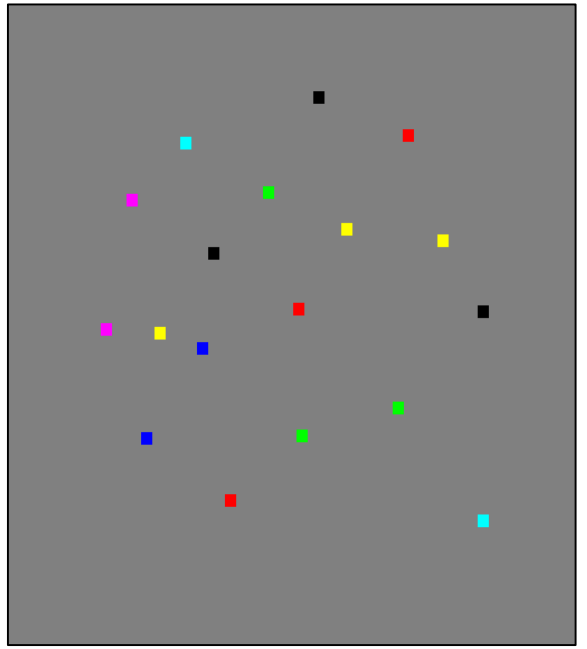
square.js (2/5)

$(x_w, y_w) \rightarrow (x, y)$

Screen coordinates World coordinates

$$x = -1 + \frac{2 * x_w}{w}$$

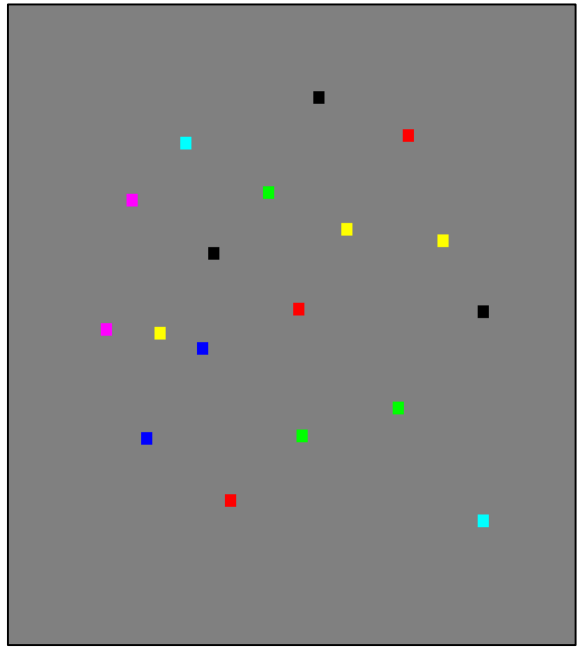
$$y = -1 + \frac{2 * (h - y_w)}{h}$$



```
window.onload = function init() {
    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    canvas.addEventListener("mousedown", function() {
        gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
        var t = vec2(2*event.clientX/canvas.width-1,
                     2*(canvas.height-event.clientY)/canvas.height-1);
        gl.bufferSubData(gl.ARRAY_BUFFER, 8*index, flatten(t));

        gl.bindBuffer(gl.ARRAY_BUFFER, cBuffer);
        t = vec4(colors[(index)%7]);
        gl.bufferSubData(gl.ARRAY_BUFFER, 16*index, flatten(t));
        index++;
    } );
```

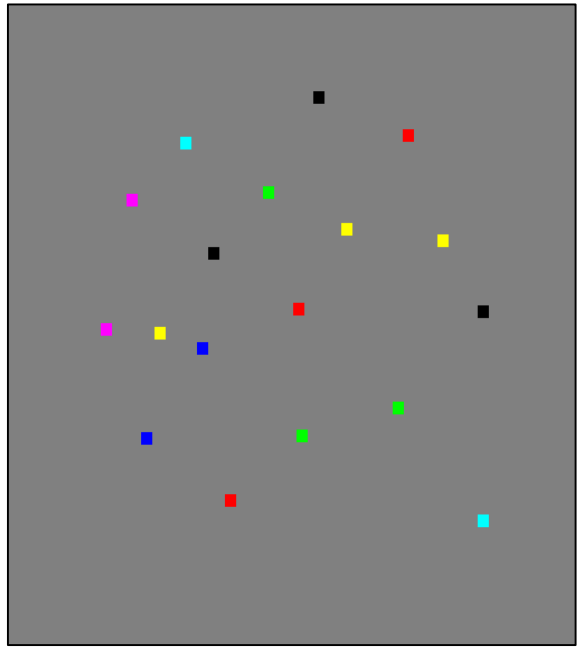
square.js (3/5)

```
gl.viewport( 0, 0, canvas.width, canvas.height );
gl.clearColor( 0.5, 0.5, 0.5, 1.0 );

//  Load shaders and initialize attribute buffers

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


var vBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer);
gl.bufferData( gl.ARRAY_BUFFER, 8*maxNumVertices, gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```



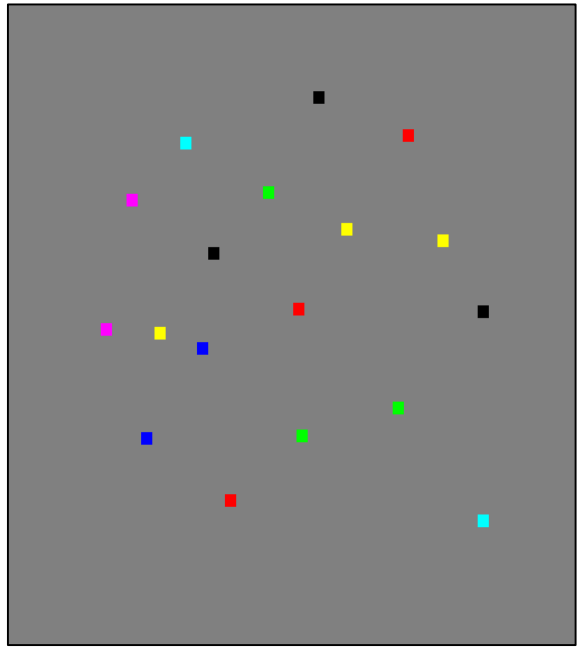
square.js (4/5)

```
    var cBuffer = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
    gl.bufferData( gl.ARRAY_BUFFER, 16*maxNumVertices, gl.STATIC_DRAW );

    var vColor = gl.getAttribLocation( program, "vColor" );
    gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vColor );

    render();

} // end of window.onload
```

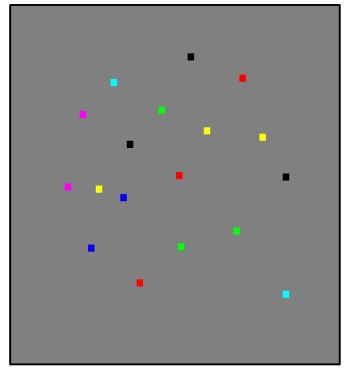


square.js (5/5)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.POINTS, 0, index );

    window.requestAnimFrame(render);

}
```



square.js : Data Structures



12 vertices

index=12

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
vBuffer	0 8 bytes	8	16	24	32	40	48	56	64	72	80	88	96	
cBuffer	0 16 bytes	16	32	48	64	80	96	112	128	144	160	176	192	

(x,y) coordinates

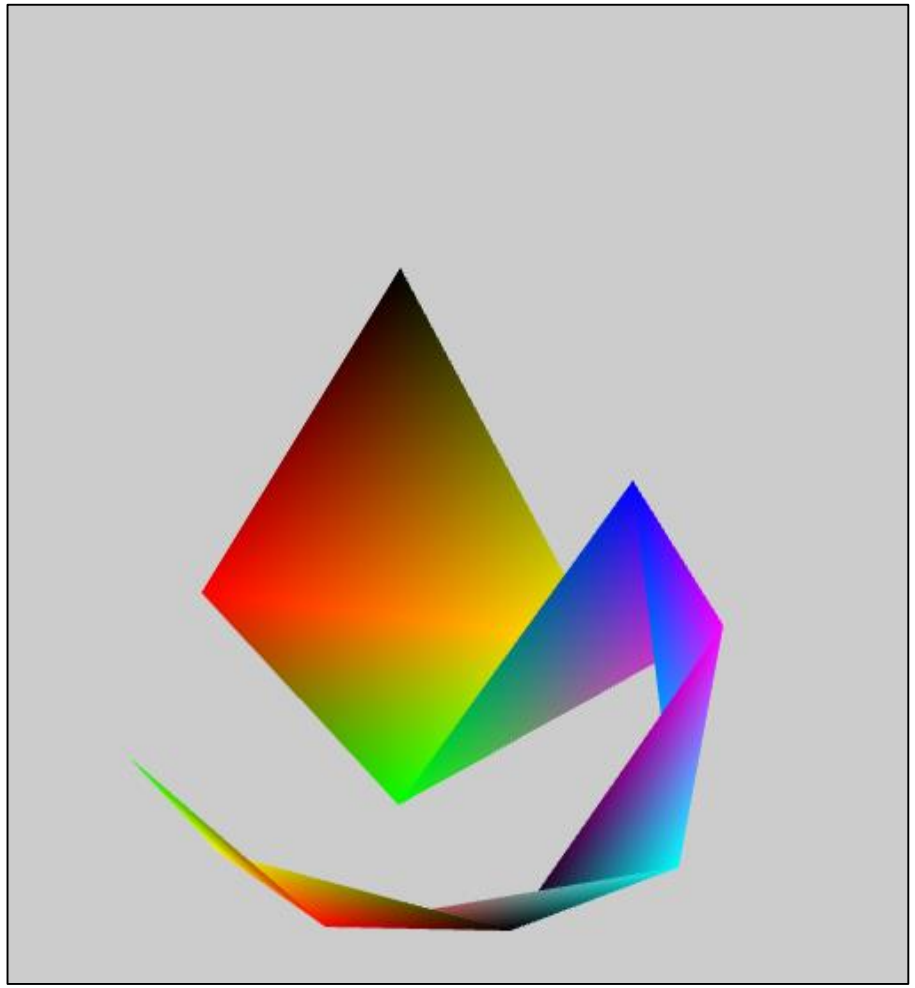
RGBA values

gl.drawArrays(gl.POINTS, 0, index)

Each vertex

(x,y) coordinates: 2 floats (8 bytes)
Colors: RGBA values(4 floats, 16 bytes)

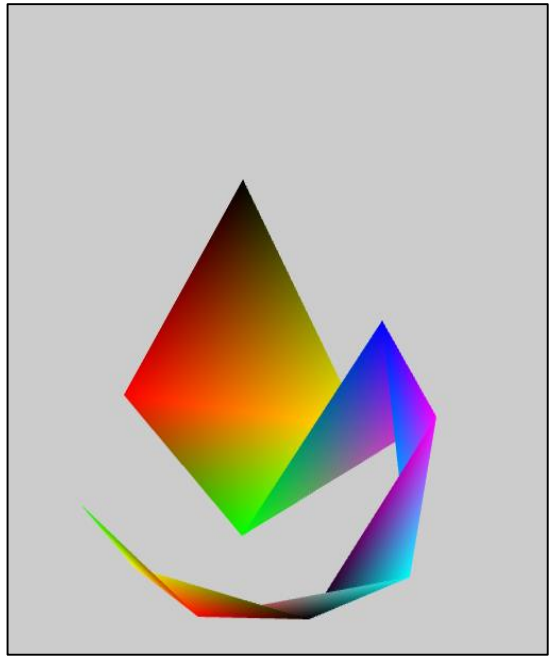
Sample Programs: triangle.html, triangle.js



Each mouse click adds another point to a triangle strip at the location of the mouse. Show color interpolation across each triangle.

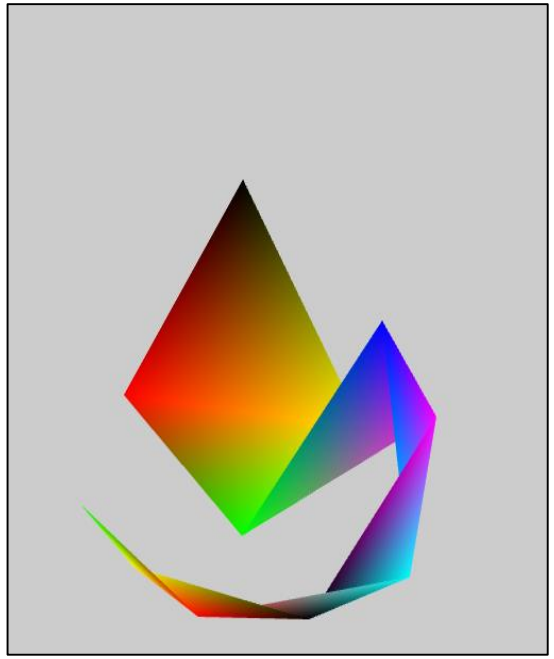
triangle.html (1/3)

```
<!DOCTYPE html>
<html>

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

attribute vec4 vPosition;
attribute vec4 vColor;

varying vec4 fColor;

void
main()
{
    gl_Position = vPosition;
    fColor = vColor;
}
</script>
```



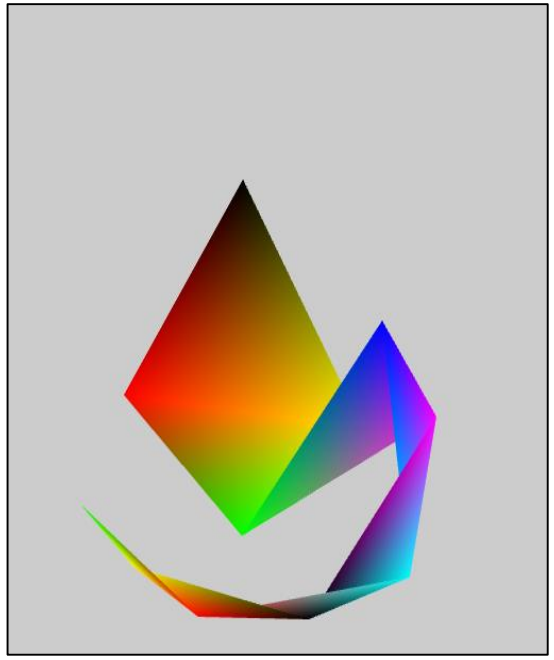
triangle.html (2/3)

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

precision mediump float;

varying vec4 fColor;

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

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="triangle.js"></script>
```



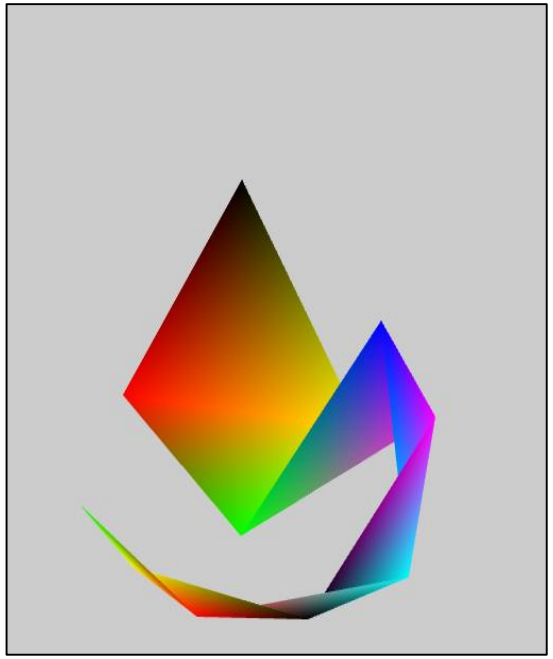
triangle.html (3/3)

```
<body>
<canvas id="gl-canvas" width="512" height="512">>
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



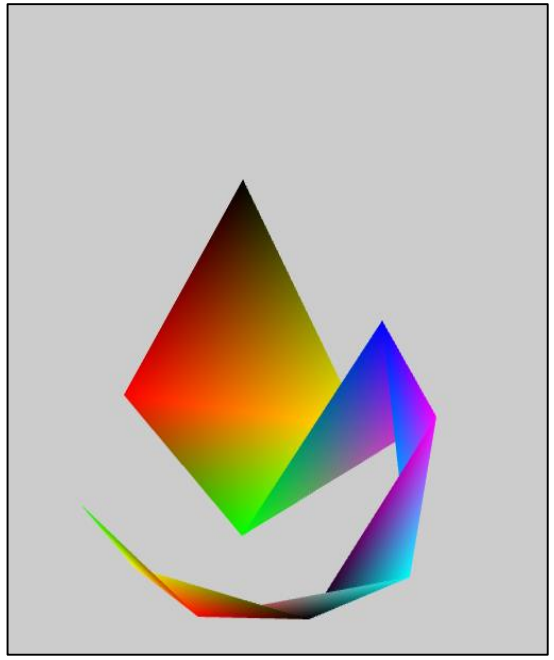
triangle.js (1/6)

```
var canvas;
var gl;

var maxNumTriangles = 200;
var maxNumVertices  = 3 * maxNumTriangles;
var index = 0;

var colors = [
    vec4( 0.0, 0.0, 0.0, 1.0 ),  // black
    vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
    vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
    vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    vec4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    vec4( 0.0, 1.0, 1.0, 1.0)    // cyan
];
```



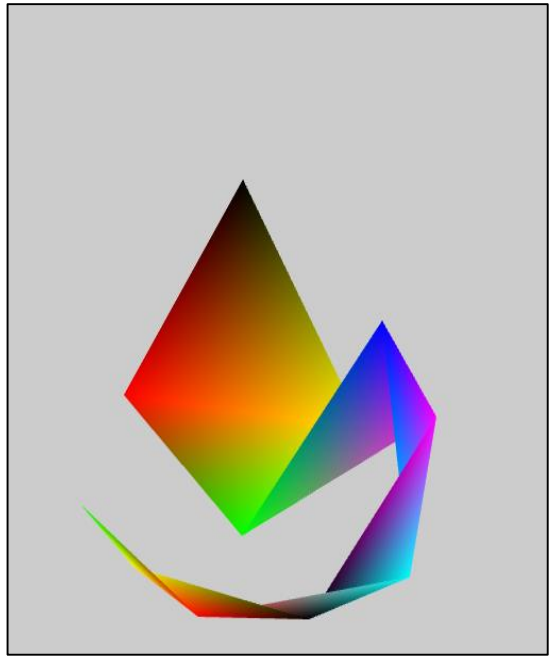
triangle.js (2/6)

```
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( 0.8, 0.8, 0.8, 1.0 );
```



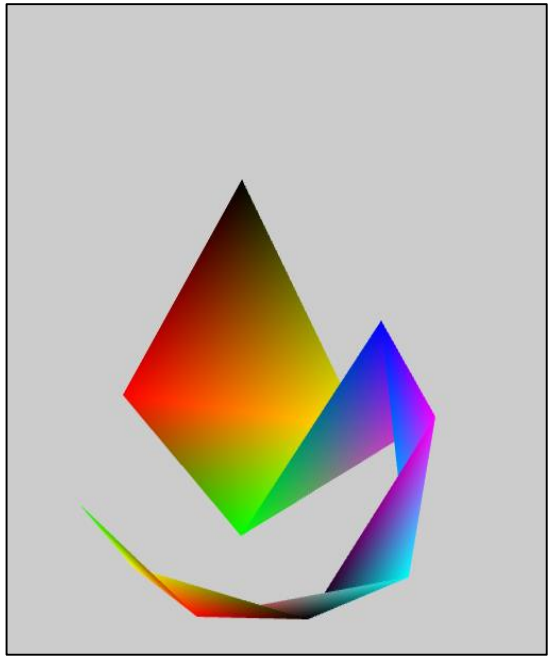
triangle.js (3/6)

```
//
//  Load shaders and initialize attribute buffers
//
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, 8*maxNumVertices, gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```



triangle.js (4/6)

```
var cBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cBuffer);
gl.bufferData(gl.ARRAY_BUFFER, 16*maxNumVertices, gl.STATIC_DRAW);

var vColor = gl.getAttribLocation( program, "vColor");
gl.vertexAttribPointer(vColor, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vColor);
```



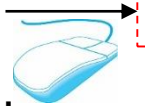
triangle.js (5/6)

$(x_w, y_w) \rightarrow (x, y)$

Screen coordinates World coordinates

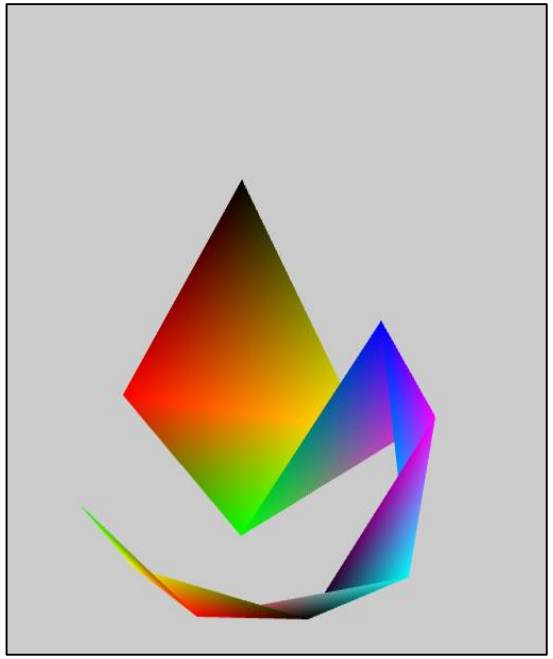
$$x = -1 + \frac{2 * x_w}{w}$$

$$y = -1 + \frac{2 * (h - y_w)}{h}$$



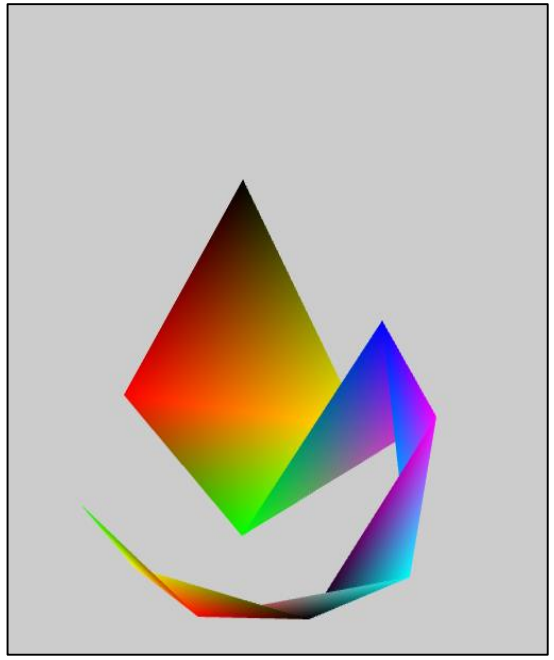
mouse

```
canvas.addEventListener("click", function() {

    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer);
    var t = vec2(2*event.clientX/canvas.width-1,
                 2*(canvas.height-event.clientY)/canvas.height-1);
    gl.bufferSubData(gl.ARRAY_BUFFER, 8*index, flatten(t));

    gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer);
    t = vec4(colors[index%7]);
    gl.bufferSubData(gl.ARRAY_BUFFER, 16*index, flatten(t));
    index++;
  } );

  render();
}  // end of window.onload
```

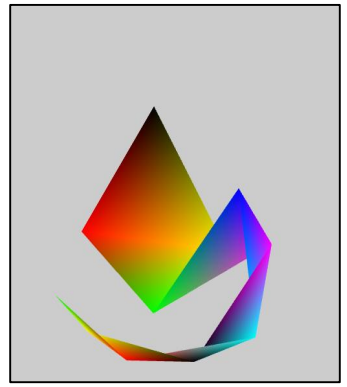


triangle.js (6/6)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLE_STRIP, 0, index );

    window.requestAnimFrame(render);
}
```



triangle.js : Data Structures



12 vertices

index=12

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
vBuffer	0 8 bytes	8	16	24	32	40	48	56	64	72	80	88	96	
cBuffer	0 16 bytes	16	32	48	64	80	96	112	128	144	160	176	192	

(x,y) coordinates

RGBA values

gl.drawArrays(gl.TRIANGLE_STRIP, 0, index)

Each vertex

(x,y) coordinates: 2 floats (8 bytes)
Colors: RGBA values(4 floats, 16 bytes)