Learning Objectives

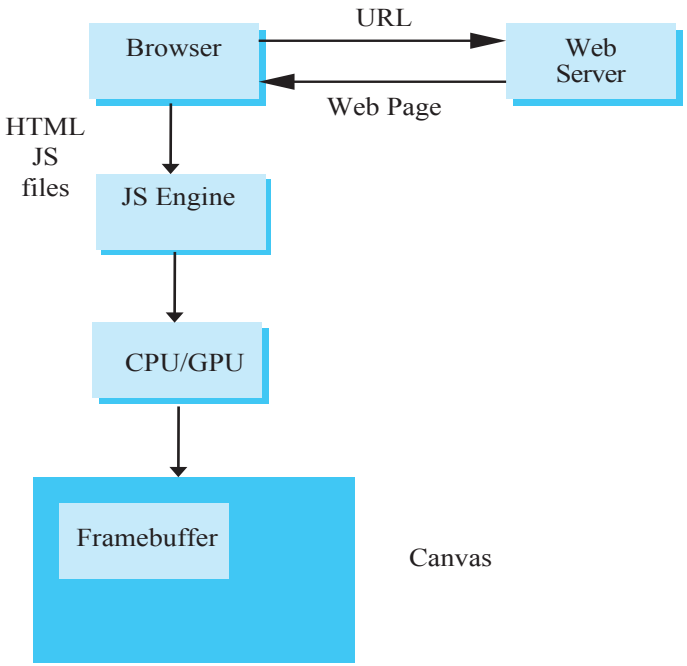
- Students completing this lecture will be able to
  - Derive the relation between window coordinates and clip coordinates
  - Explain the following terms: event-driven programming, double buffering
  - Explain how to partially update data in the GPU `gl.bufferSubData()`
  - Write WebGL code with animation and various interaction controls

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



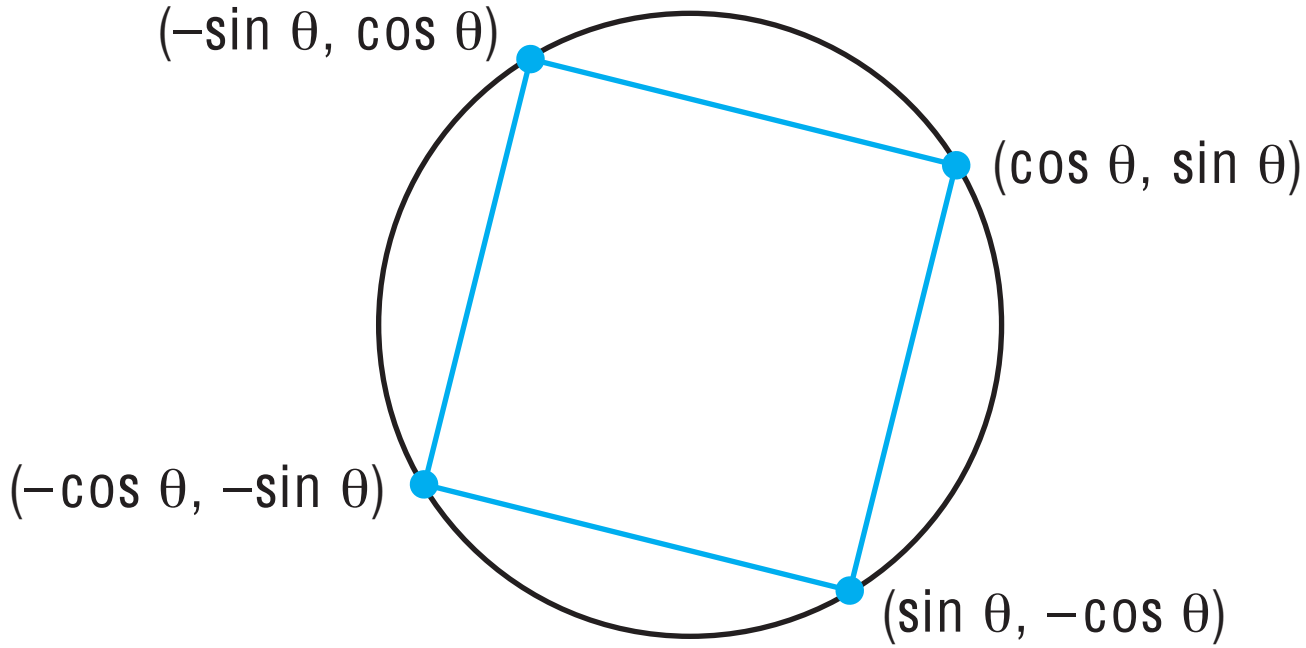
- Start with HTML file
  - Describes the page
  - May contain the shaders
  - Loads files
- Files are loaded asynchronously and JS code is executed
- Then what?
  - Browser is in an event loop and waits for an event

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

       gl.bufferData(…);

       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(); // you will get maximum call stack size
exceeded error
}
```

Vertex Shader

```
attribute vec4 vPosition;
uniform float theta;

void main()
{
   gl_Position.x = cos(theta) * vPosition.x -
sin(theta) * vPosition.y;
   gl_Position.y = sin(theta) * vPosition.x +
cos(theta) * vPosition.y;
   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
- Could lead to performance problems: this JS method was designed before browsers started to support multiple tabs, it executes regardless of which tab is active

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);
}
```

- Only called when the tab in which it was defined is activated
- Not yet standardized
- Defined in the library supplied by Google, `webgl-utils.js`

requestAnimFrame

- Either one of these will work

```
requestAnimFrame(render);
requestAnimationFrame(render);
window.requestAnimFrame(render);
window.requestAnimationFrame(render);
```

Add an Interval

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

setTimeout(
    function(){requestAnimFrame(render);},
    1000/60); // 60 frames per second
```

Yet Another Way

```
var g_last = Date.now(); // last time this function was called
var mspf = 1000/30.0;  // ms per frame, 30 frames per second
var g_elapsed = 0;

function render () {
    // calculate the elapsed time
    var g_now = Date.now(); // time in ms
    g_elapsed += g_now - g_last;
    g_last = g_now;
    if (g_elapsed >= mspf) {
        gl.clear(gl.COLOR_BUFFER_BIT);
        theta += 0.1;
        gl.uniform1f(thetaLoc, theta);
        g_elapsed = 0;
    }
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    requestAnimFrame(render);
};
```

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()
{
    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);
}
```

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; };
```

Radio Buttons

```
<form>
    <input type="radio" name="function"
id="rotationon" checked>Rotation On
    <input type="radio" name="function"
id="rotationoff">Rotation Off
</form>
```

- Define an form object, only one radio button in the form could be true
- The default true one is marked with “checked”

Turn on/off the Animation

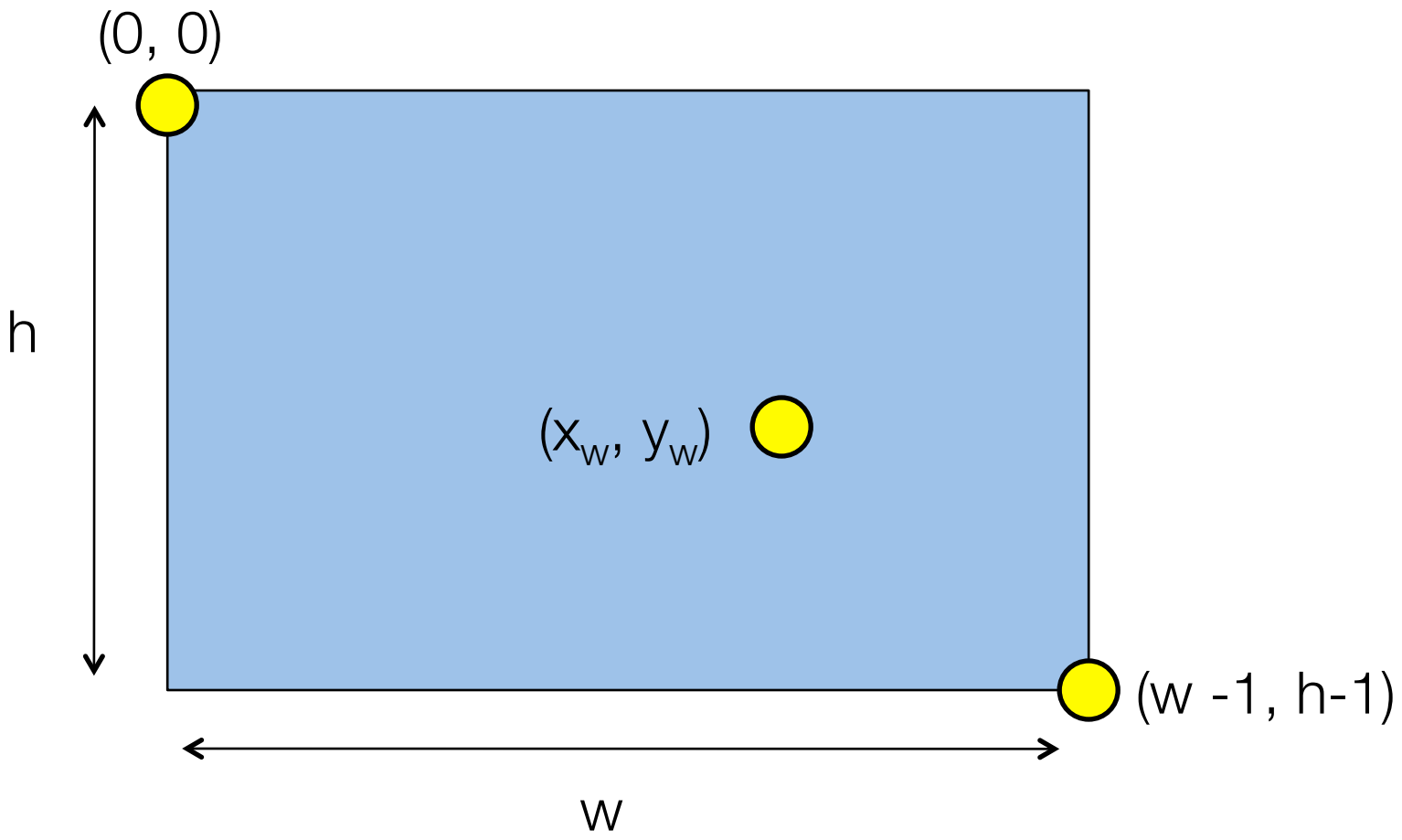
```
function render() {
  gl.clear( gl.COLOR_BUFFER_BIT );
  if (document.getElementById("rotationon").checked
     == true)
    {
      theta += 0.02;
      gl.uniform1f( thetaLoc, theta );
    }
  gl.drawArrays( gl.TRIANGLE_STRIP, 0, 4 );
  window.requestAnimFrame(render);
}
```

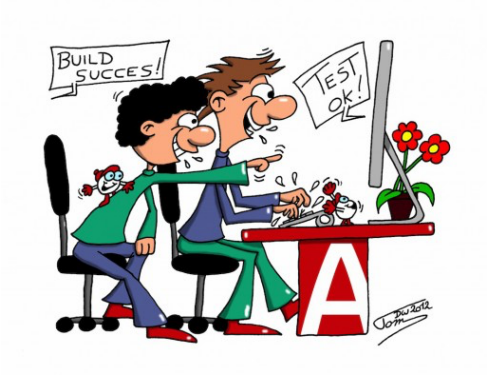
Position Input

Objectives

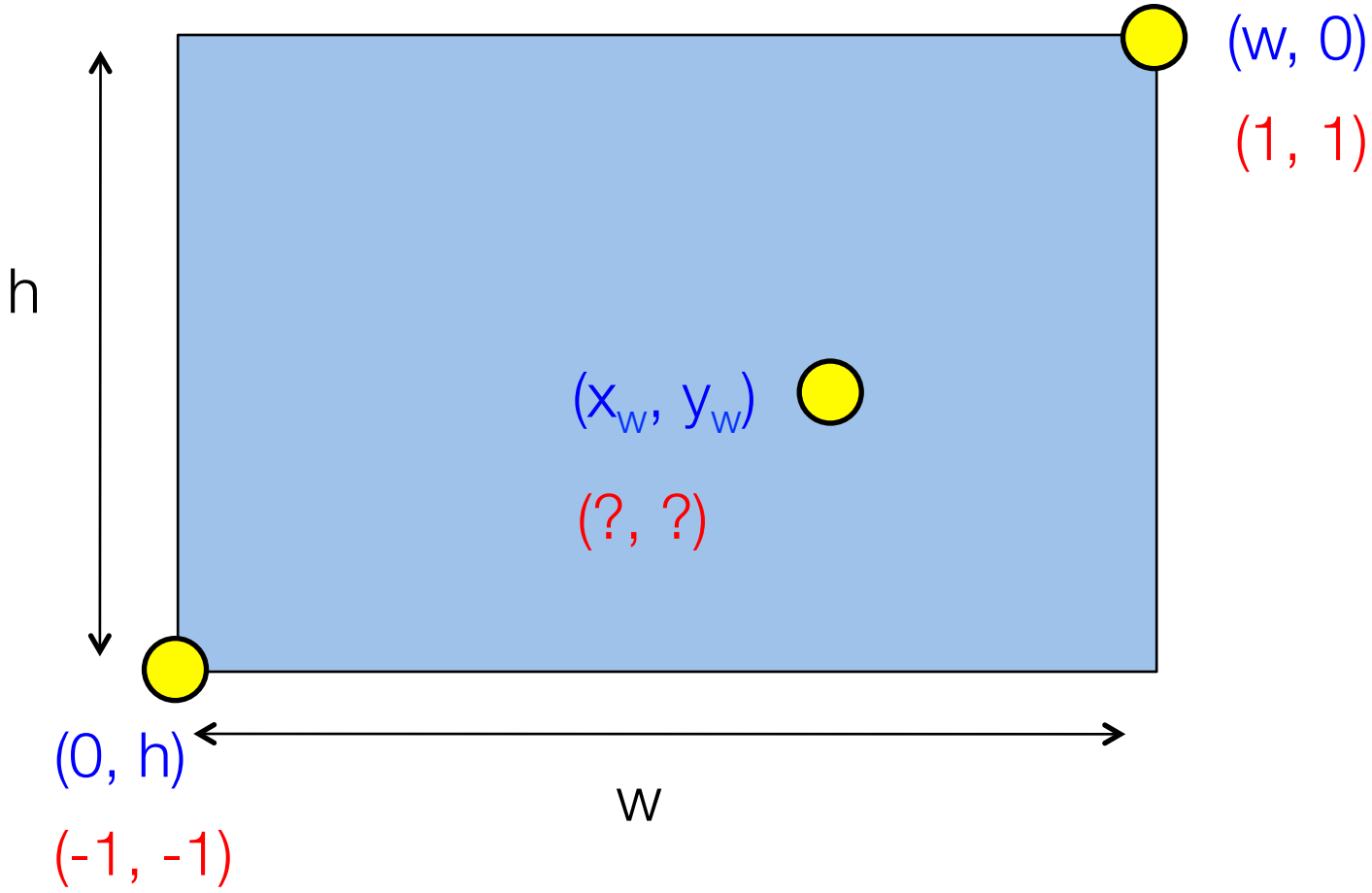
- Learn to use the mouse to give locations
  - Must convert from position on canvas to position in application
- Respond to window events such as reshapes triggered by the mouse

Window Coordinates





Window to Clip Coordinates



Window to Clip Coordinates

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

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

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

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

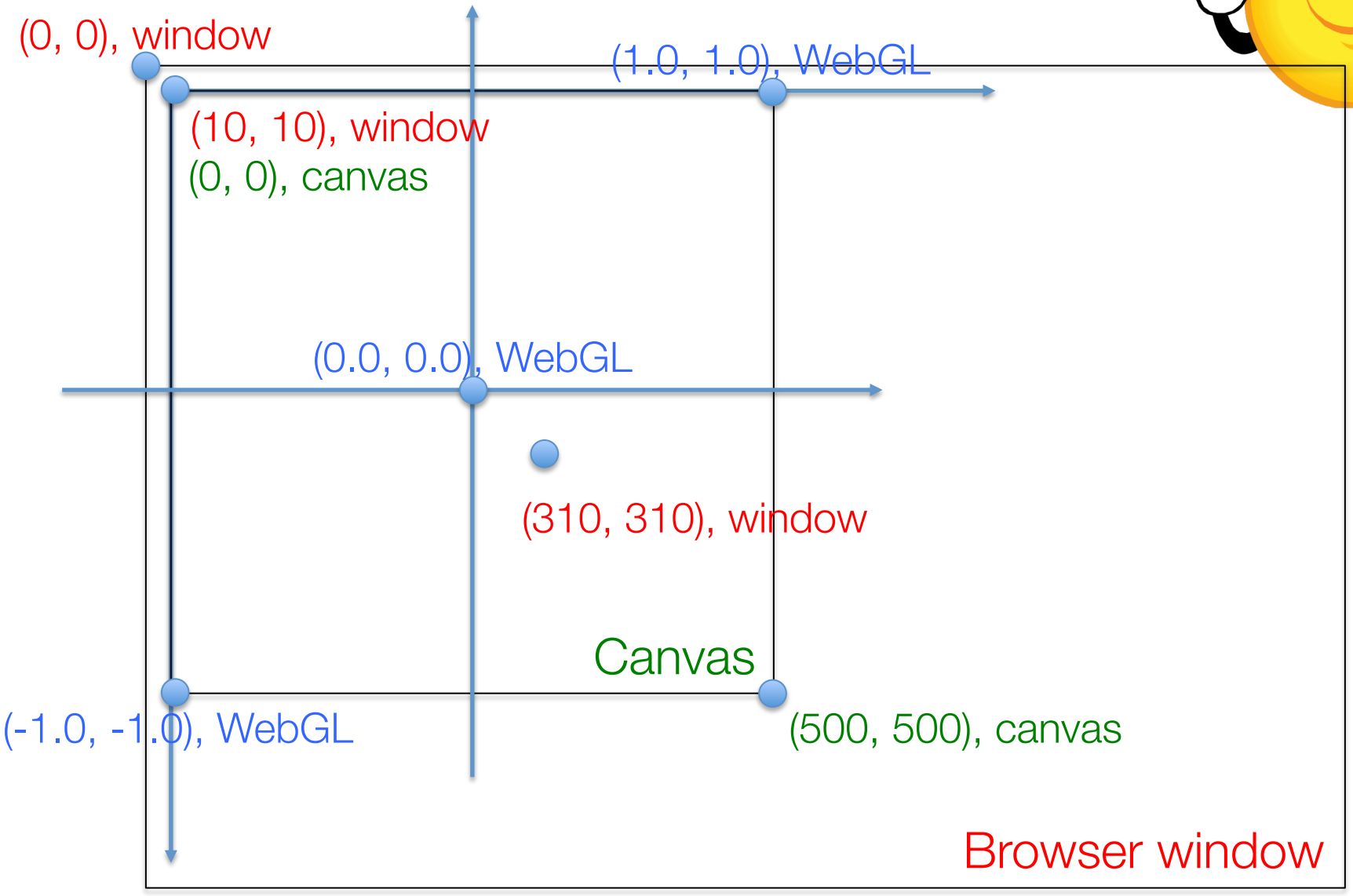
- Scale the window coordinates x_w and y_w to (0, 1) by dividing its ranges w and h
- Note that y should be flipped to (h-y_w)
- Rescale (0, 1) to (-1, 1)

Returning Position from Click Event

- Canvas specified in HTML file of size `canvas.width` and `canvas.height`
- Returned window coordinates are `event.clientX` and `event.clientY`

```
// add a vertex to GPU for each click
canvas.addEventListener("click", function() {
  var t = vec2(-1 + 2*event.clientX/canvas.width,
     -1 + 2*(canvas.height-event.clientY)/canvas.height);
  gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
  // sizeof['vec2'] = 8
  gl.bufferSubData(gl.ARRAY_BUFFER, sizeof['vec2']*index,
     flatten(t));
   index++;
});
```

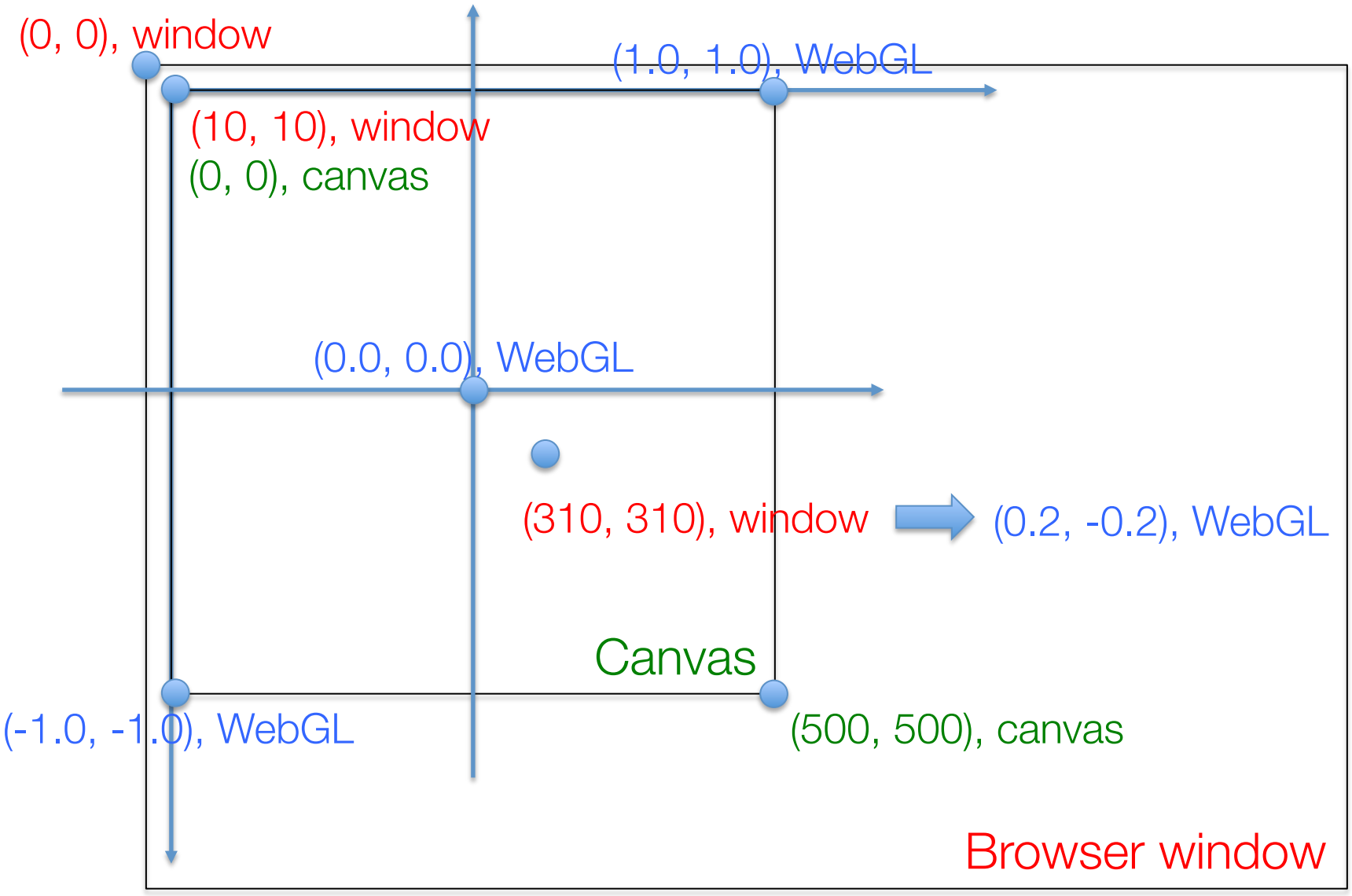
The Actual Truth



The Accurate Calculation

```
// add a vertex to GPU for each click
canvas.addEventListener("click", function() {
    // in our example:
    // offsetX = 10, offsetY = 10
    offsetX = event.target.getBoundingClientRect().left;
    offsetY = event.target.getBoundingClientRect().top;
    // in our example:
    // event.clientX = 310, event.clientY = 310
    // canvas.width = 500, canvas.height = 500
    var t = vec2(-1 + 2*(event.clientX-offsetX)/canvas.width,
      -1 + 2*(canvas.height-(event.clientY-offsetY))/
     canvas.height);
    gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
    gl.bufferSubData(gl.ARRAY_BUFFER,
      sizeof['vec2']*index, flatten(t));
    index++;
});
```

The Actual Truth



CAD-like Examples

- `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 the 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 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

`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;
    }
    // min is the smaller of innerWidth and innerHeight
    if (min < canvas.width || min < canvas.height) {
       gl.viewport(0, canvas.height-min, min, min);
    }
};
```