



Graphics

Jungchan Cho

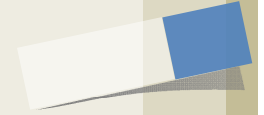
Dept. of Software, Gachon University

Many slides from Edward Angel and Dave Shreine

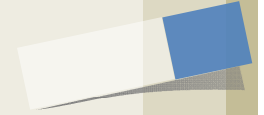
Many examples are from <https://webglfundamentals.org/>



Outline



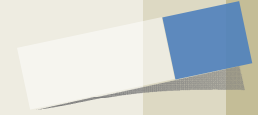
- ☐ Input and Interaction
- ☐ Animation
- ☐ Working with Callbacks
- ☐ Position Input
- ☐ Picking



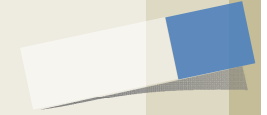
Input and Interaction



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

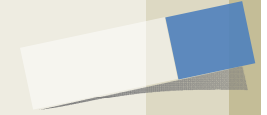


Project Sketchpad

- ❑ 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



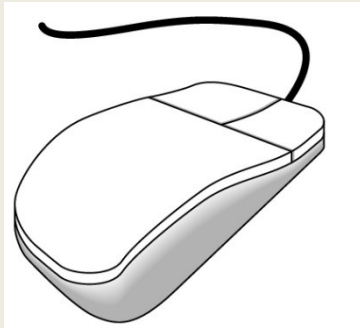
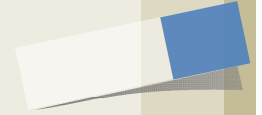
Graphical Input



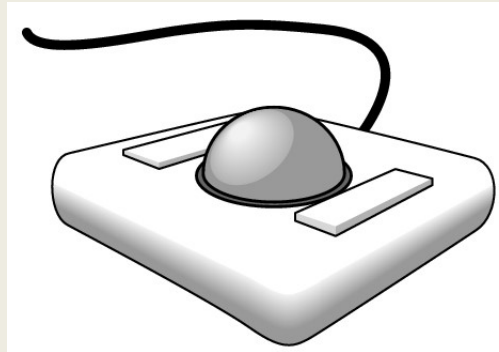
- ❑ Devices can be described either by
 - ❑ Physical properties
 - ❑ Mouse
 - ❑ Keyboard
 - ❑ Trackball
 - ❑ Logical Properties
 - ❑ What is returned to program via API
 - A position
 - An object identifier
- ❑ Modes
 - ❑ How and when input is obtained
 - ❑ Request or event



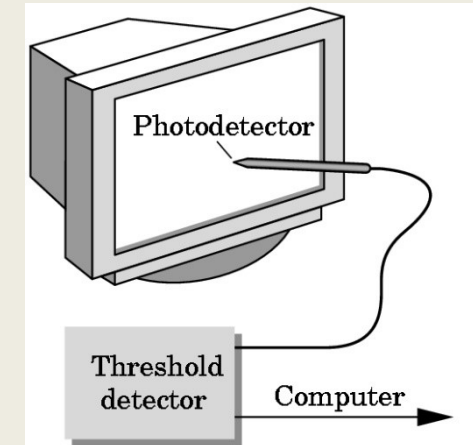
Physical Devices



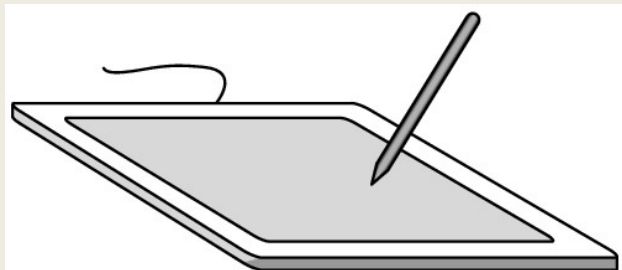
mouse



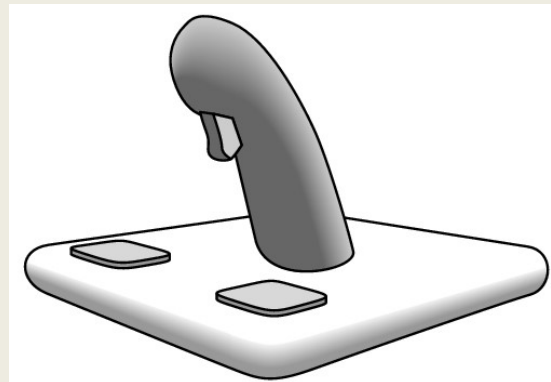
trackball



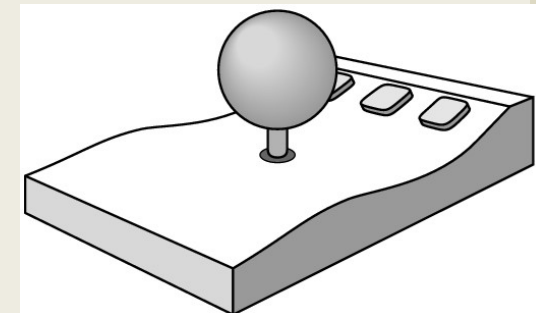
light pen



data tablet



joy stick



space ball

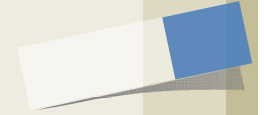


Incremental (Relative) Devices

- ❑ 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



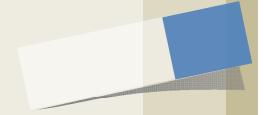
Logical Devices



- ❑ 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

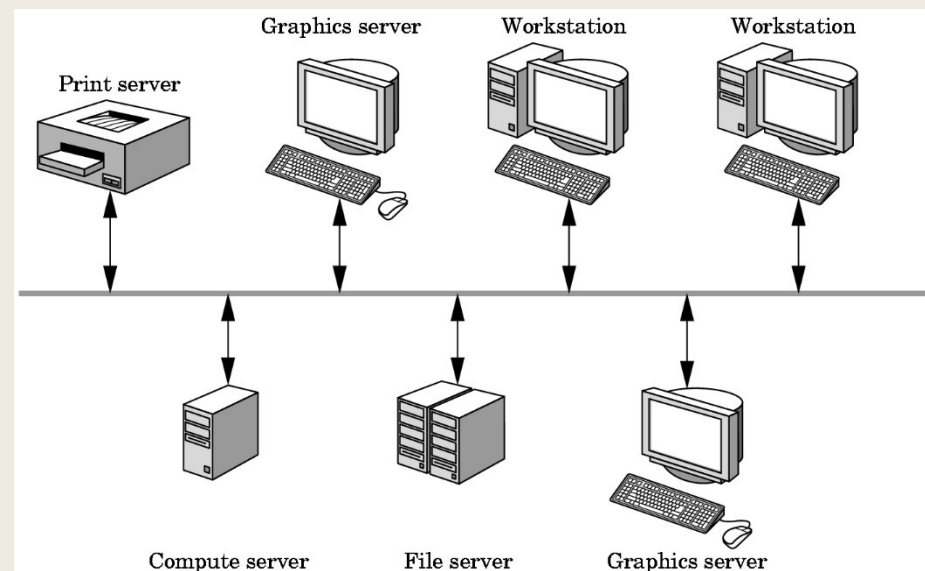


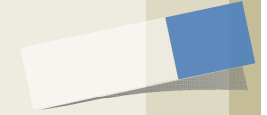
- ❑ Graphical input is more varied than input to standard programs which is usually numbers, characters, or bits
- ❑ Two older APIs (GKS, PHIGS) defined six types of logical input
 - ▣ **Locator**: return a position
 - ▣ **Pick**: return ID of an object
 - ▣ **Keyboard**: return strings of characters
 - ▣ **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





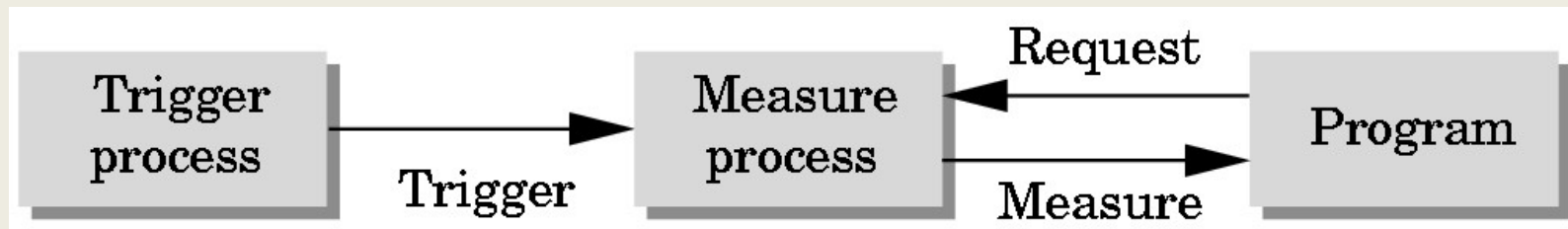
Input Modes

- ❑ 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

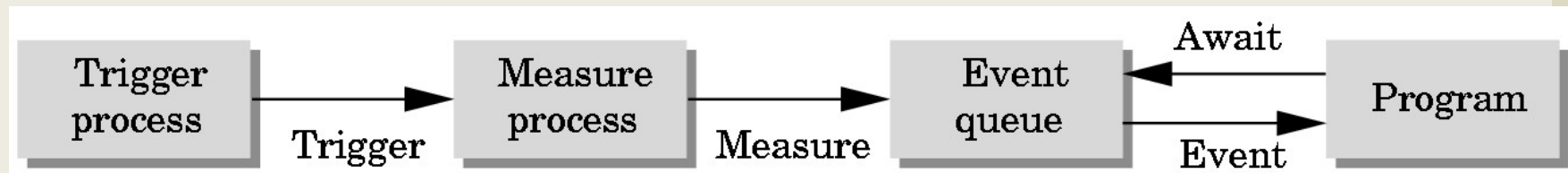
- ❑ Input provided to program only when user triggers the device
- ❑ Typical of keyboard input
 - ▣ Can erase (backspace), edit, correct until enter (return) key (the trigger) is depressed

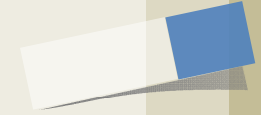




Event Mode

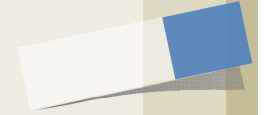
- ❑ Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user
- ❑ Each trigger generates an *event* whose measure is put in an *event queue* which can be examined by the user program





Event Types

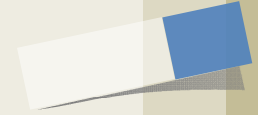
- ❑ Window: resize, expose, iconify
- ❑ Mouse: click one or more buttons
- ❑ Motion: move mouse
- ❑ Keyboard: press or release a key
- ❑ Idle: nonevent
 - ▣ Define what should be done if no other event is in queue



Animation



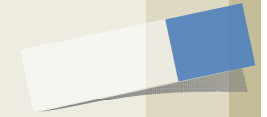
Callbacks



- ❑ 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



이벤트 리스너



- 이벤트 리스너는 말 그대로 해당 이벤트에 대해 대기중인 겁니다. 항상 리스닝 중이죠. 해당 이벤트가 발생했을 때 등록했던 이벤트 리스너가 실행됩니다.

```
1 window.onload = function () {  
2     alert('I\'m loaded');  
3 };
```

- window가 load될 때에 function 부분이 실행됨
- 브라우저가 load 되었다는 것을 알려줌
- window 말고도, 여러 태그에 각각 이벤트를 설정할 수 있음
- 모든 DOM들이 이벤트 리스너를 등록할 수 있는 속성들을 갖고 있음



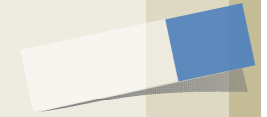
on + '이벤트명'

- ❑ **onload**(문서나 객체가 로딩되었을 때)
- ❑ **onclick**(객체를 클릭했을 때)
- ❑ **ondblclick**(더블클릭할 때)
- ❑ **onkeypress**(키를 누르고 있을 때)
- ❑ **onkeyup**(키를 눌렀다 떼었을 때)
- ❑ **onresize**(객체의 크기가 바뀌었을 때)
- ❑ ...

```
1 document.getElementById('clickMe').onclick = function () {  
2     alert('I\'m clicked!');  
3 };
```



이벤트를 붙이는 다른 방법: addEventListener




- ❑ 저는 on으로 붙이는 것보다 이 방식을 더 추천합니다. 여러 이벤트를 등록할 수 있고, 특정 이벤트를 제거(removeEventListener)할 수도 있거든요.

```
1 function onClick() {  
2     alert('I\'m clicked!');  
3 }  
4 function onClick2() {  
5     alert('또다른 이벤트');  
6 }  
7 document.getElementById('clickMe').addEventListener('click', onClick); // 이벤트 연결  
8 document.getElementById('clickMe').addEventListener('click', onClick2); // 또 하나의 이벤트 연결  
9 document.getElementById('clickMe').removeEventListener('click', onClick); // 연결할 이벤트 중 하나 제거
```



앞의 두 코드에서 function 부분을 부르는 게 콜백이라고 합니다.



- ❑ 영어로는 **callback**인데 call + back
- ❑ 즉, 다시 전화주는 거죠.
- ❑ 이벤트가 실행됐을 때, 사용자에게 다시 알려주는 겁니다.
- ❑ onload, onclick 같은 속성에 콜백을 등록하면 됩니다. 위처럼 대입할 수도 있고요.
- ❑ 꼭 이벤트가 아니더라도 콜백을 활용할 수 있습니다.

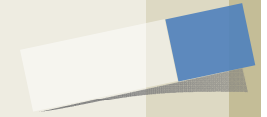
주어진 숫자부터 1까지 더하는 함수

```
1 var cbExample = function(number, cb) {
2   setTimeout(function() {
3     var sum = 0;
4     for (var i = number; i > 0; i--) {
5       sum += i;
6     }
7     cb(sum);
8   }, 0);
9 };
10 cbExample(10, function(result) {
11   console.log(result);
12 }); // 55
13 console.log('first');
```

- ❑ cbExample 함수의 매개변수 cb가 바로 콜백 함수를 받는 부분
- ❑ cbExample을 호출할 때 두 번째 인자로 function을
 - 그 함수가 콜백함수로 등록되어 계산이 끝난 후 실행됩니다. → 콜백만 등록해두면 계산이 끝난 후 알아서 알려주게 됨.
 - 위의 예시를 콘솔에 쳐보시면 first가 55보다 아래 줄에 있음에도 먼저 실행됨
 - setTimeout 비동기적으로 실행을 위한 것 (0초만에 실행하게 설정해놨는데도 비동기적으로 실행됩니다).
 - 이 부분을 빠뜨리면 55가 first보다 먼저 표시됩니다.
- ❑ 만약 cbExample의 일이 10초가 걸리는 일이라한다면?
 - cbExample이 끝나고 다음 코드를 실행할 때까지 10초나 기다려야 함.
 - 콜백을 등록하면 바로 다음 코드로 넘어갈 수 있음 → 알아서 일이 완료된 후에 알려주도록 만들어진 것임
 - 콜백은 언제 끝날 지 모르는 동작에 대해 그 결과를 전달받을 때 유용



콜백 함수(Callback function)



□ 일반적인 함수

- 파라미터를 받아 함수 내부로 진입
 - 결과값을 돌려줄 때는 return 키워드를 사용해 호출자에게 값을 돌려 주는 구조
- 또는 파라미터 자체를 Call by reference로 전달(포인터 전달)
 - 호출자가 참조하는 값 자체를 함수 내에서 변경하기도 합니다.
 - 이 경우 리턴값은 없거나 에러 여부를 전달하는 데 쓰임.

```
1 // 일반적인 함수
2 function sum(num1, num2){
3     return num1+num2;
4 }
5
6 // 함수 호출
7 sum(1, 2);
```

CS

□ 콜백 함수

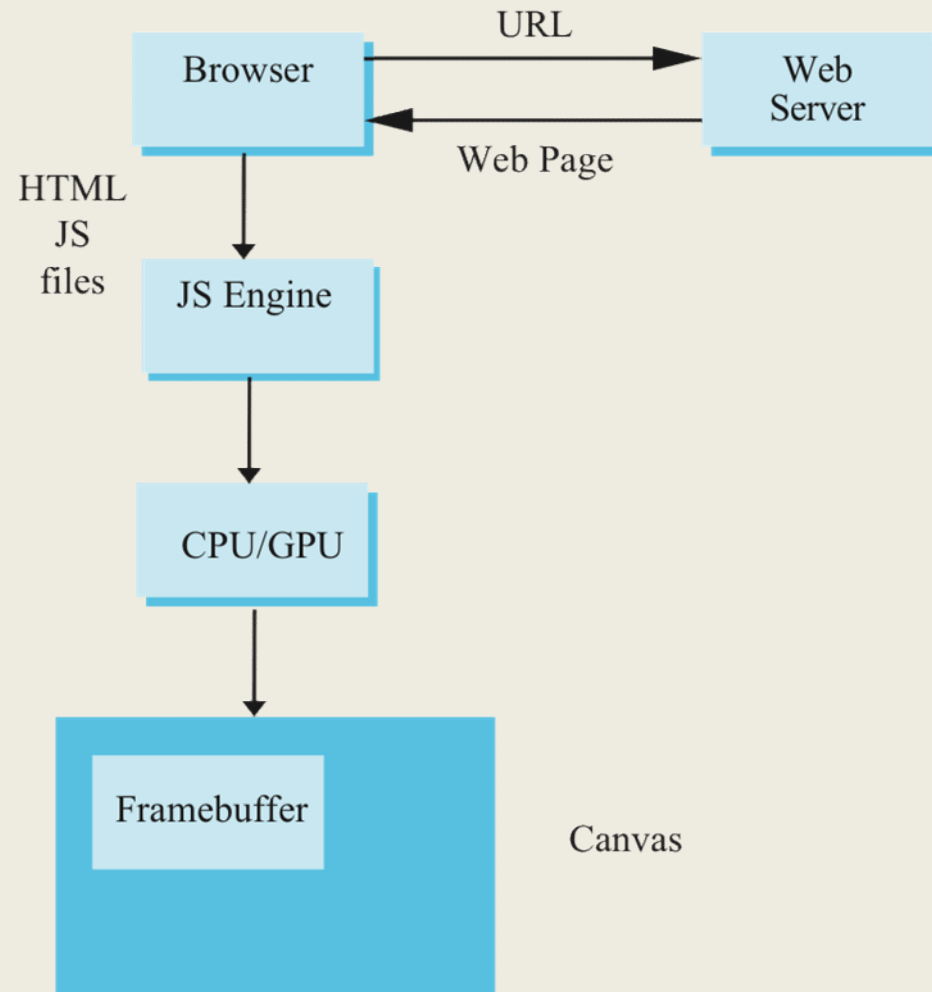
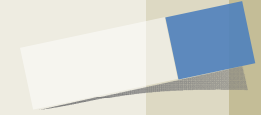
- 콜백 함수는 함수로써 다른 함수에 전달되며, 이는 외부 함수 내에서 일종의 루틴 또는 동작을 완성하기 위해 호출됩니다.

```
1 function greeting(name){
2     alert('반갑습니다. ' + name);
3 }
4
5 function processUserInput(callback){
6     var name = prompt('이름을 입력해 주세요. ');
7     callback(name);
8 }
9
10 processUserInput(greeting);
```

CS

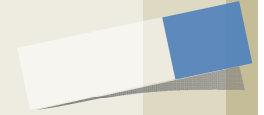


Execution in a Browser





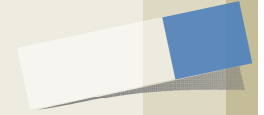
Execution in a Browser



- ❑ 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

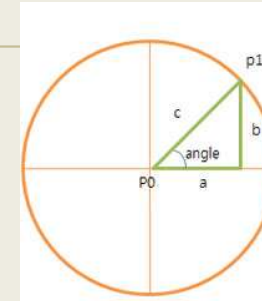
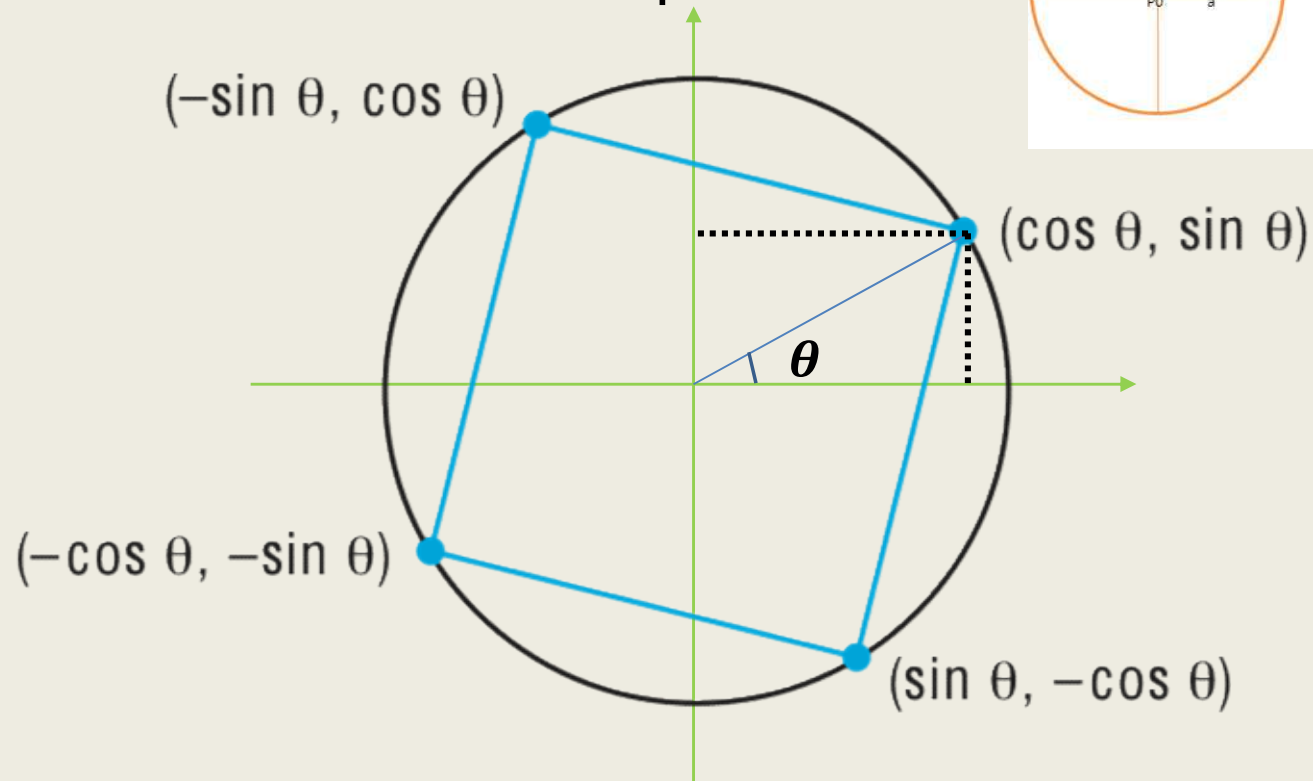


- ❑ What happens with our JS file containing the graphics part of our application?
 - ▣ All the “action” is within functions such as `init()` and `render()`
 - ▣ Consequently these functions are never executed and we see nothing
- ❑ Solution: use the onload window event to initiate execution of the init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



Rotating Square

❑ Consider the four points



각도 $\sin(\text{angle}) = \text{높이}(b)/\text{빗면}(c)$
각도 $\cos(\text{angle}) = \text{밑면}(a)/\text{빗면}(c)$

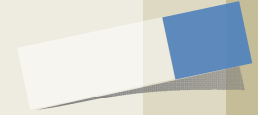
밑면 $a = \cos(\text{angle}) * \text{빗면}(c)$
높이 $b = \sin(\text{angle}) * \text{빗면}(c)$

(P1) $x = \cos(\text{angle}) * \text{빗면}(c)$
(P1) $y = \sin(\text{angle}) * \text{빗면}(c)$

Animate display by rerendering with different values of θ



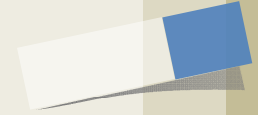
Simple but Slow Method



```
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();  
}
```



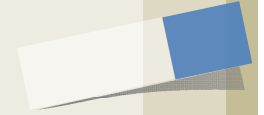
Better Way



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



Render Function

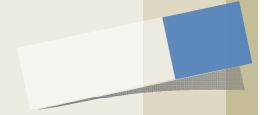


```
var thetaLoc = gl.getUniformLocation(program, "theta");
```

```
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

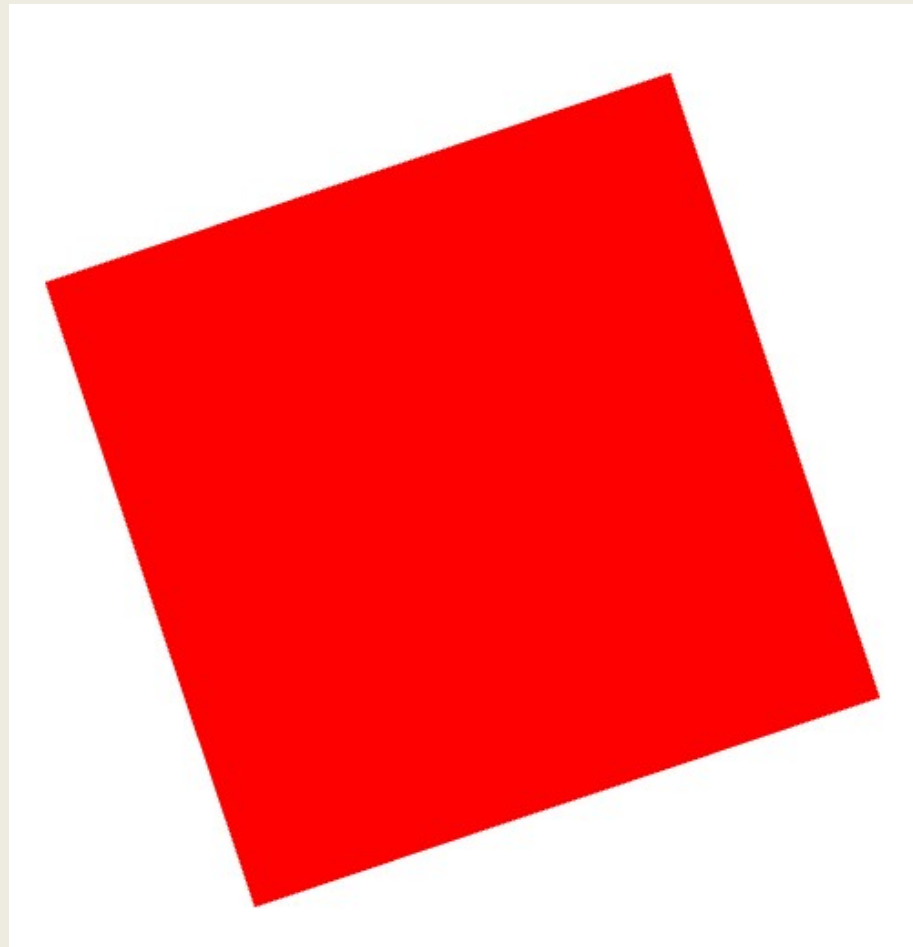
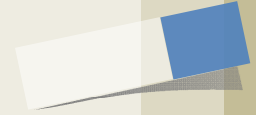


```
attribute vec4 vPosition;  
uniform float theta;
```

```
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;  
}
```



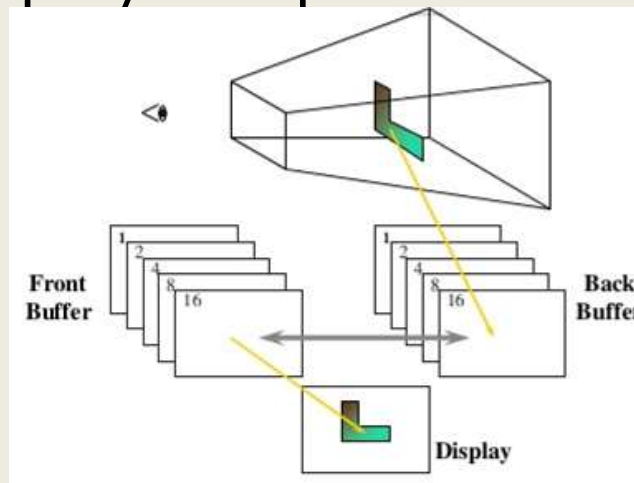
rotatingSquare1.html





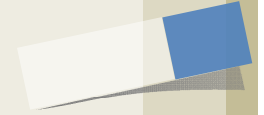
Double Buffering

- ❑ Although we are rendering the square, it's 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

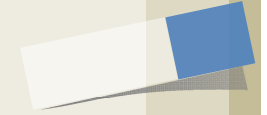




Triggering a Buffer Swap



- ❑ 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**
 - ❑ **requestAnimationFrame**



Interval Timer

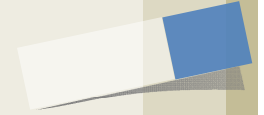
- ❑ Executes a function after a specified number of milliseconds
 - ▣ Also generates a buffer swap

`setInterval(render, interval);`

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



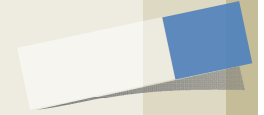
requestAnimationFrame



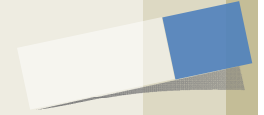
```
function render {  
    gl.clear(gl.COLOR_BUFFER_BIT);  
    theta += 0.1;  
    gl.uniform1f(thetaLoc, theta);  
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);  
    requestAnimationFrame(render);  
}
```



Add an Interval



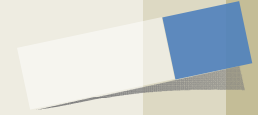
```
function render()
{
    setTimeout( function() {
        requestAnimationFrame(render);
        gl.clear(gl.COLOR_BUFFER_BIT);
        theta += 0.1;
        gl.uniform1f(thetaLoc, theta);
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    }, 100);
}
```



Working with Callbacks



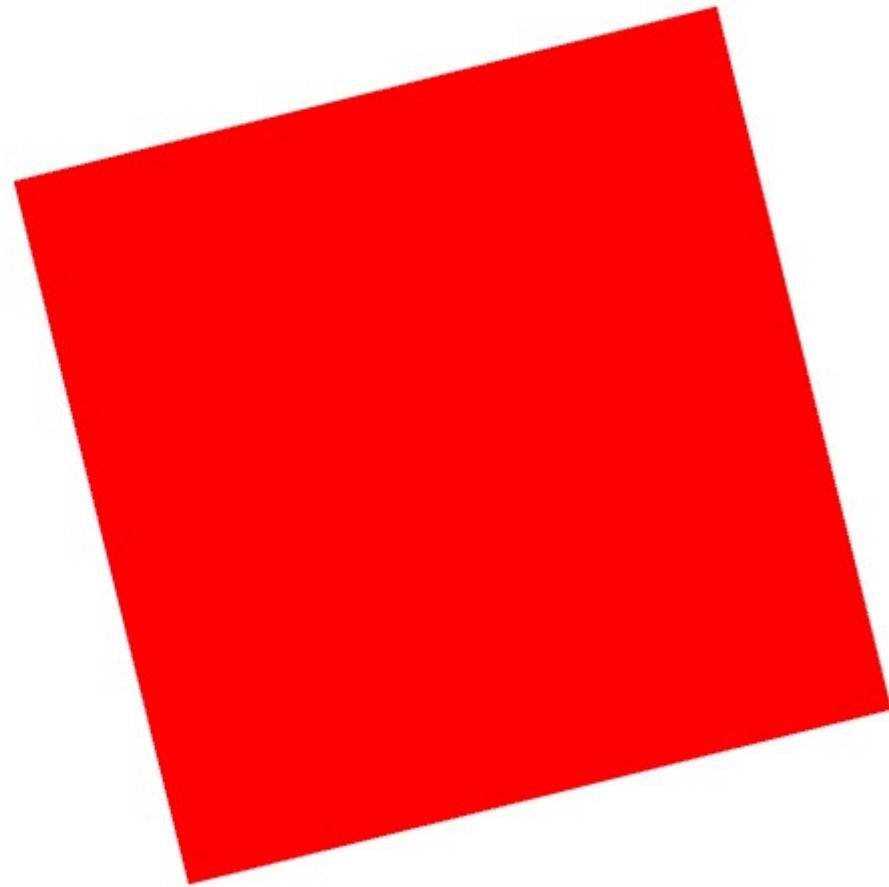
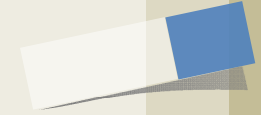
Objectives



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



rotatingSquare2.html

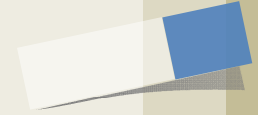


Change Rotation Direction

Toggle Rotation Direction
Spin Faster
Spin Slower



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

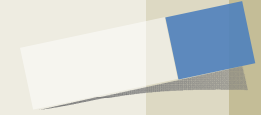
```
var direction = true; // global initialization
```

```
// in render()
```

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



The Button



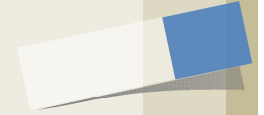
- ❑ In the HTML file

`<button id="Direction">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



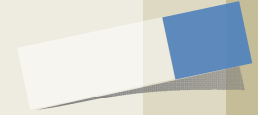
- ❑ We still need to define the listener
 - ❑ no listener and the event occurs but is ignored
- ❑ **Two forms** for event listener in JS file

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

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



onclick Variants



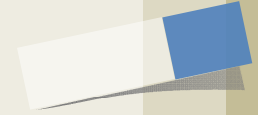
```
myButton.addEventListener("click", function() {  
    if (event.button == 0) { direction = !direction; }  
});
```

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

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



Controlling Rotation Speed

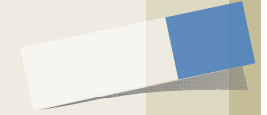


```
var delay = 100;

function render()
{
    setTimeout(function() {
        requestAnimationFrame(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

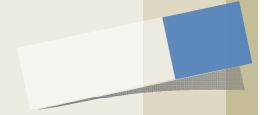


- ❑ Use the HTML `select` element
- ❑ Each entry in the menu is an `option` element with an integer `value` returned by click event

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



Menu Listener

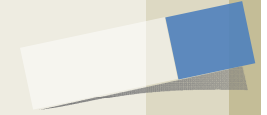


```
var m =
document.getElementById("Controls");
m.addEventListener("click", function()
{
    switch (m.selectedIndex) {
        case 0:
            direction = !direction;
            break;
        case 1:
            delay /= 2.0;
            break;
        case 2:
            delay *= 2.0;
     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

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



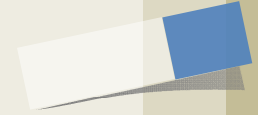
Using keydown Event



```
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;  
    }  
});
```



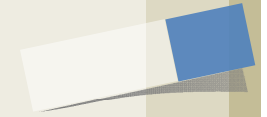

Don't Know Unicode



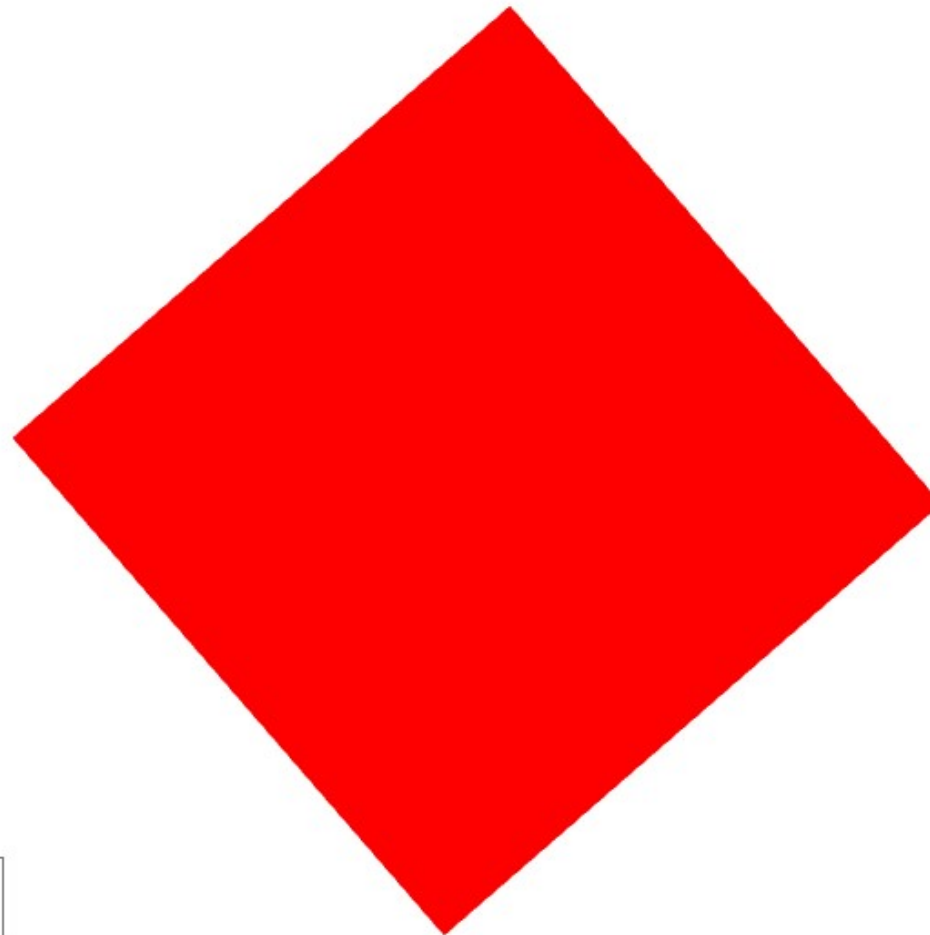
```
window.onkeydown = function(event) {  
    var key = String.fromCharCode(event.keyCode);  
    switch (key) {  
        case '1':  
            direction = !direction;  
            break;  
        case '2':  
            delay /= 2.0;  
            break;  
        case '3':  
            delay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



rotationSquare3.html



speed 0%  100%

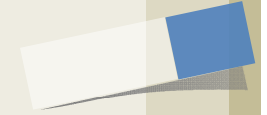


Change Rotation Direction

Toggle Rotation Direction
Spin Faster
Spin Slower



Slider Element

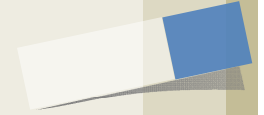


- ❑ Puts slider on page
 - ❑ Give it an **id**entifier
 - ❑ Give it **minimum** and **maximum** values
 - ❑ Give it a step **size** needed to generate an event
 - ❑ Give it an initial **value**
- ❑ Use **div** tag to put below canvas

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

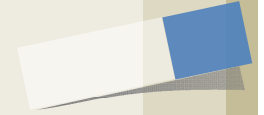


onchange Event Listener



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

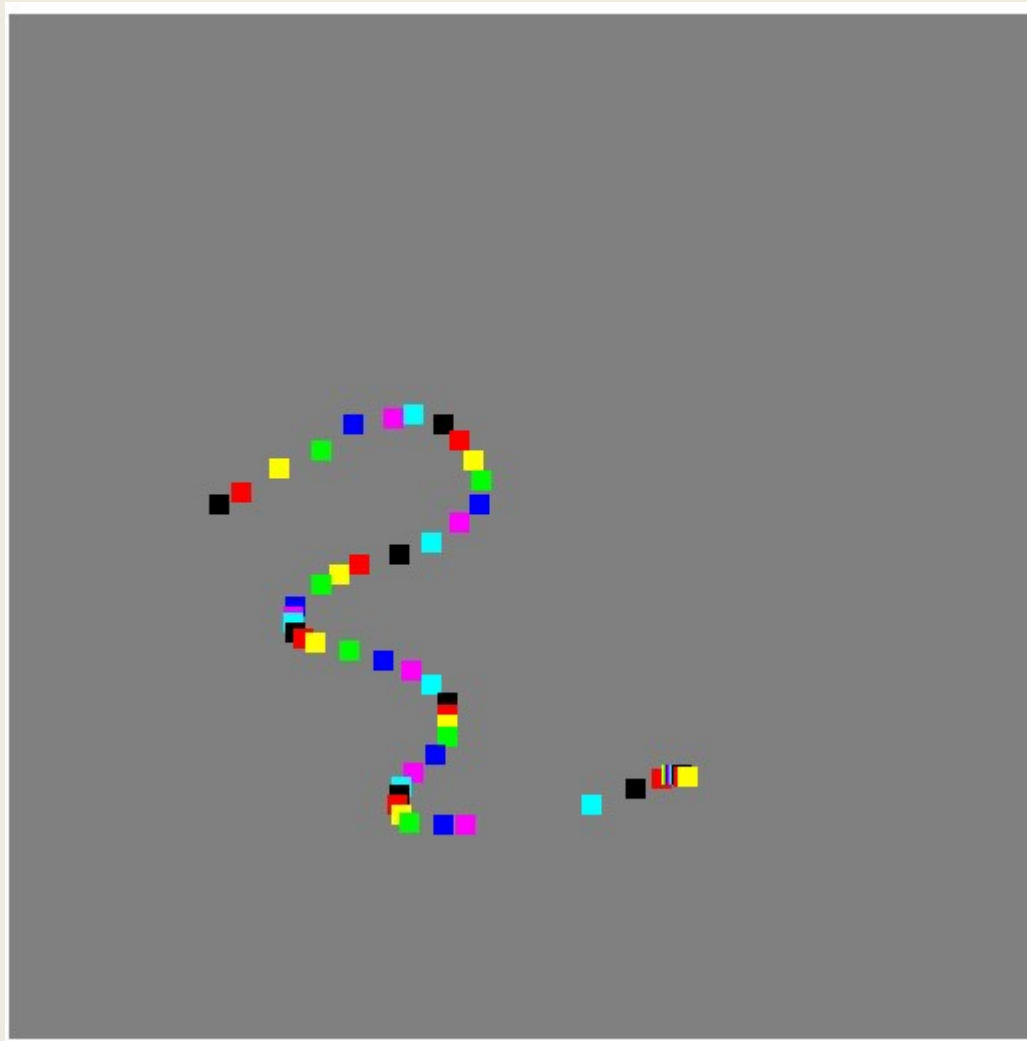
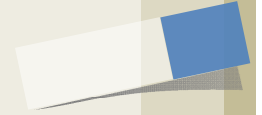
```
document.getElementById("slider").onchange =  
    function(event) {      speed = 100 - event.target.value;  };
```



Position Input

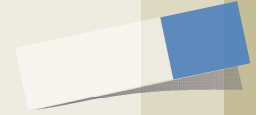


Squarem.html





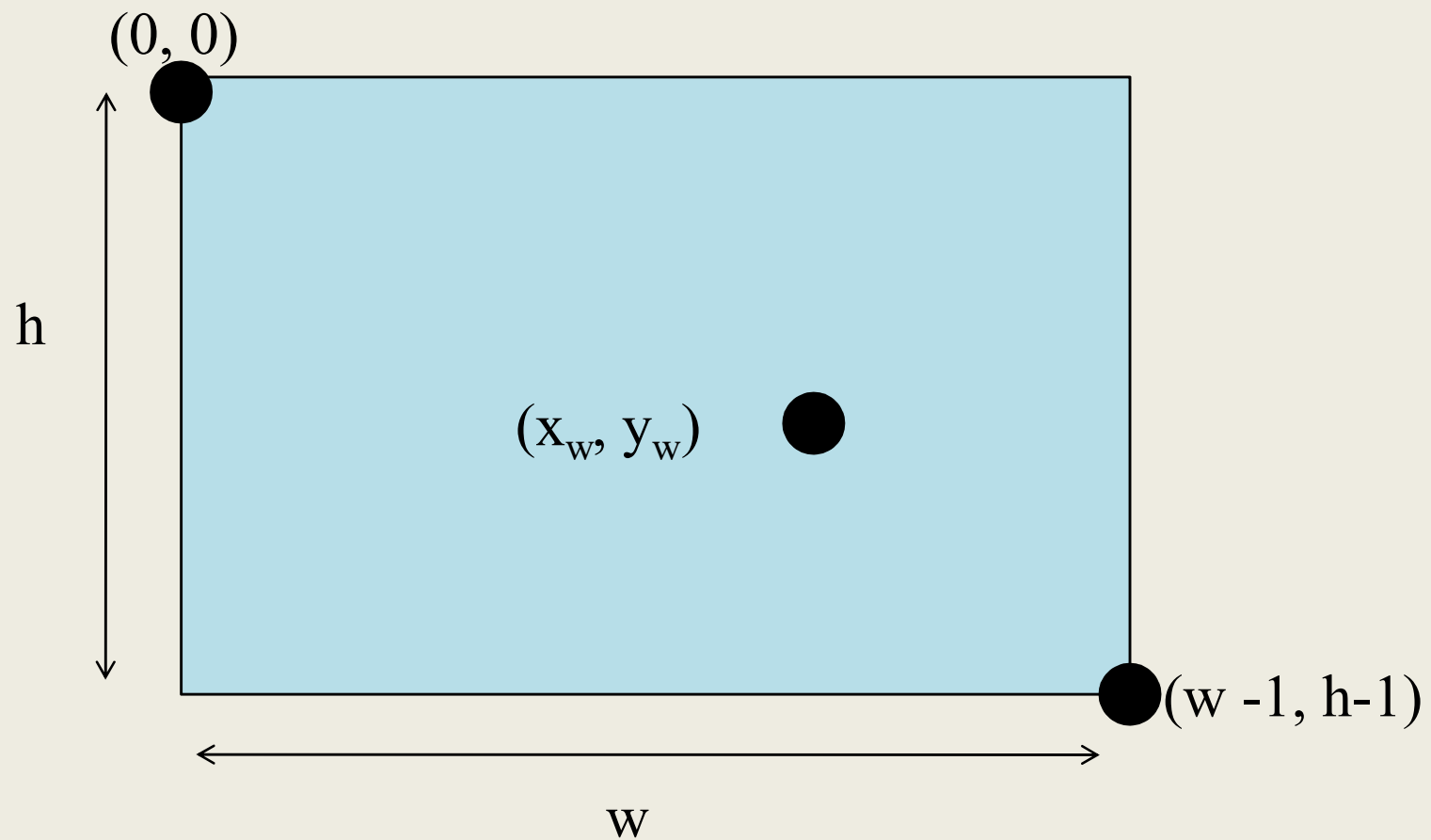
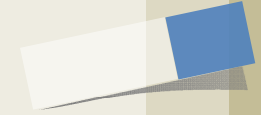
Objectives



- ❑ 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

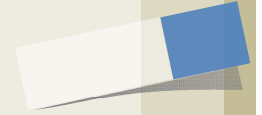


Window Coordinates





Window to Clip Coordinates



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

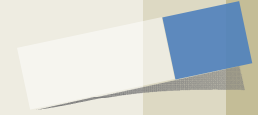
$$(w, 0) \rightarrow (1, 1)$$

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

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



Returning Position from Click Event



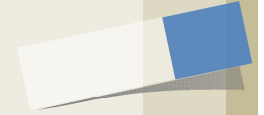
Canvas specified in HTML file of size
`canvas.width` x `canvas.height`

Returned window coordinates are
`event.clientX` and `event.clientY`

```
// 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++;
});
```



Window Events

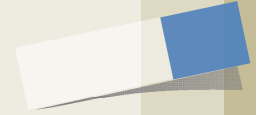


- ❑ Events can be generated by actions that affect the canvas window
 - ❑ moving or exposing a window
 - ❑ resizing a window
 - ❑ opening a window
 - ❑ iconifying/deiconifying a window a window
- ❑ Note that events generated by other application that use the canvas can affect the WebGL canvas
 - ❑ There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

http://voxelent.com/html/beginners-guide/chapter_1/ch1_Car.html



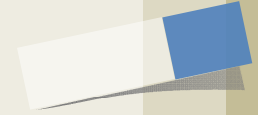
Reshape Events



- ❑ 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

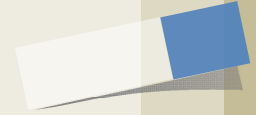


Keeping Square Proportions

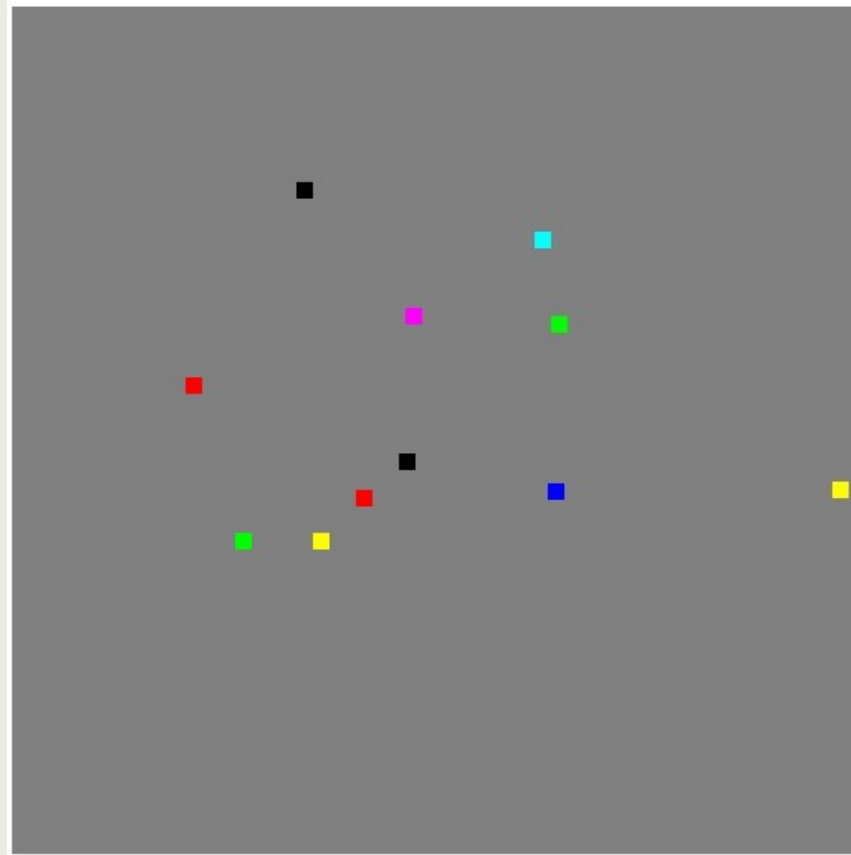
```
window.onresize = function() {  
    var min = innerWidth;  
    if (innerHeight < min) {  
        min = innerHeight;  
    }  
    if (min < canvas.width || min < canvas.height) {  
        gl.viewport(0, canvas.height-min, min, min);  
    }  
};
```



Exercise 8

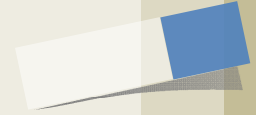


- ❑ Put a colored square at location of each mouse click

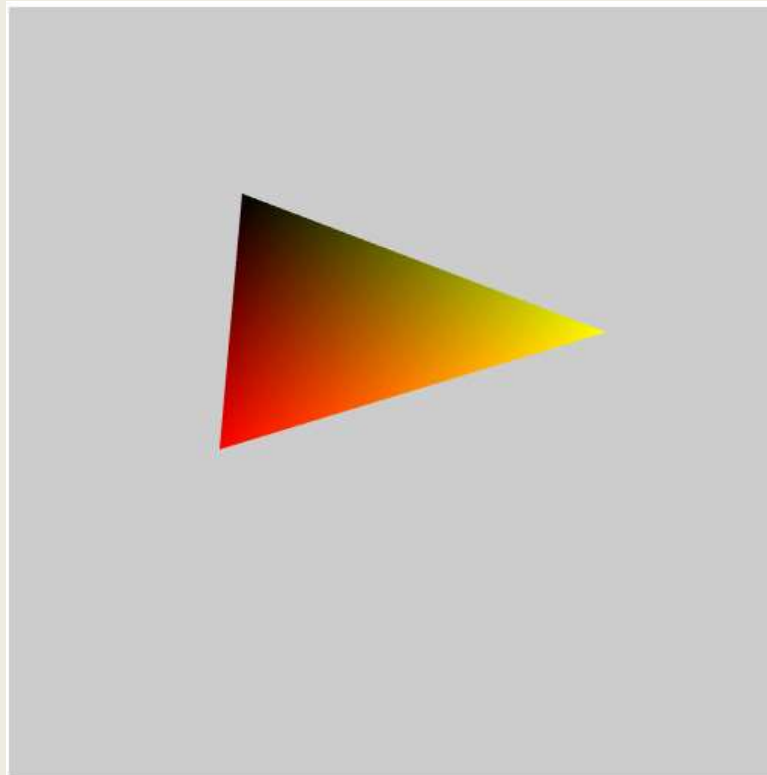




Exercise 9

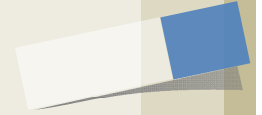


- ❑ First three mouse clicks define first triangle of triangle strip. Each succeeding mouse clicks adds a new triangle at end of strip

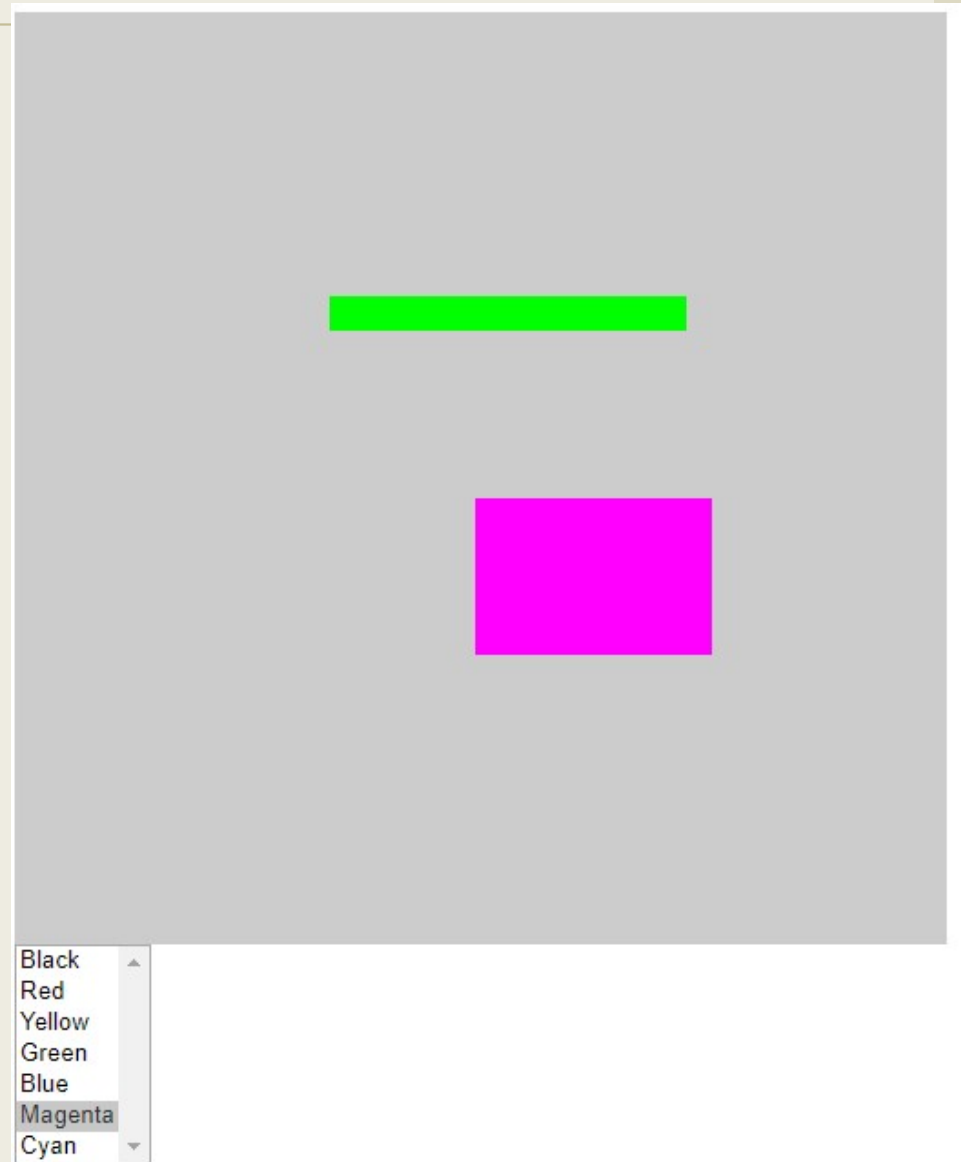




Exercise 10



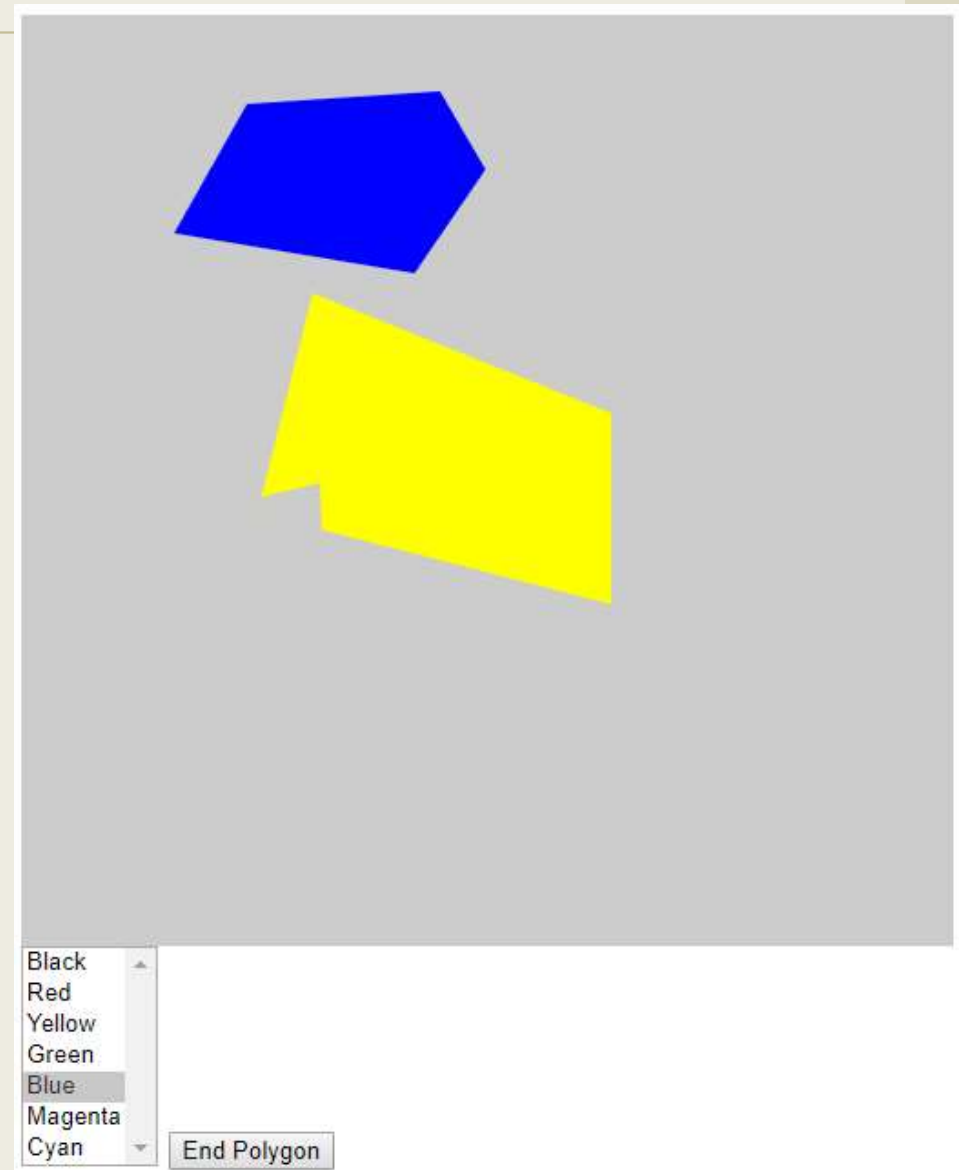
- ❑ Draw a rectangle for each two successive mouse clicks





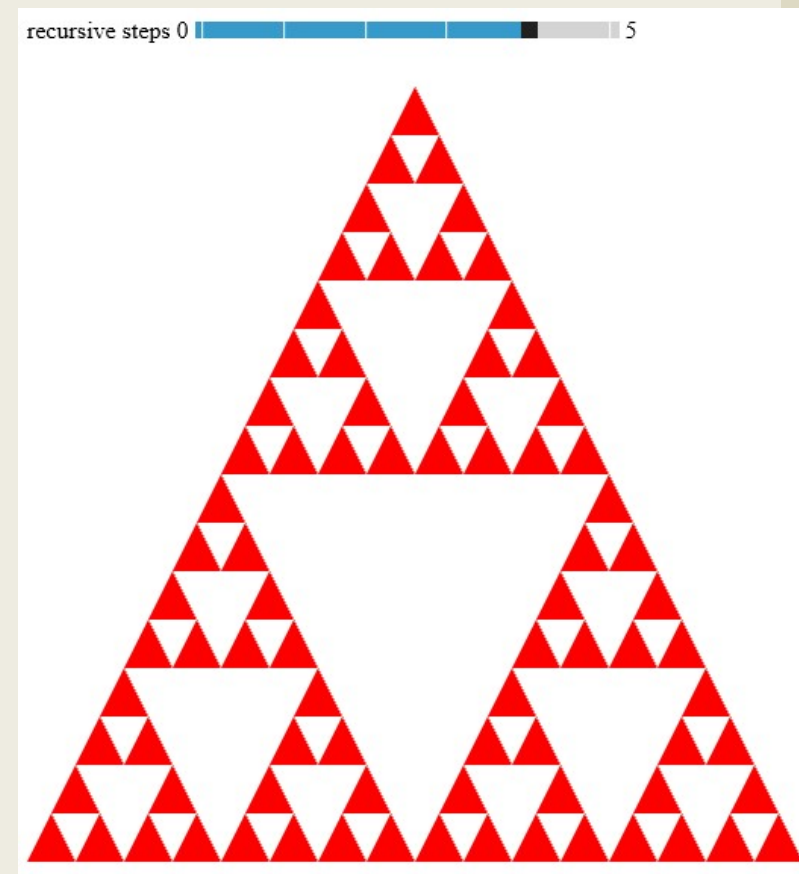
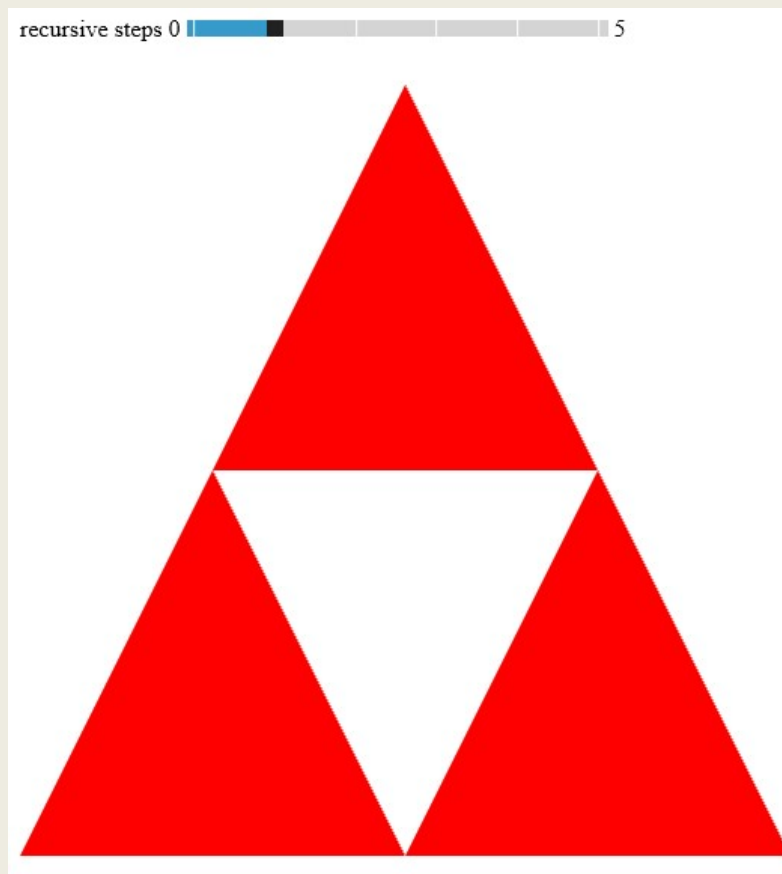
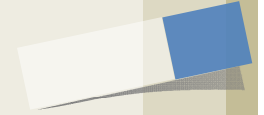
Exercise 11

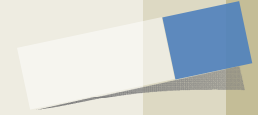
- ❑ Draws arbitrary polygons





Exercise 12



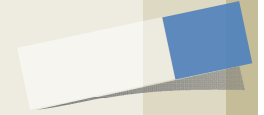


(Optional) Picking

http://voxelent.com/html/beginners-guide/chapter_8/ch8_Picking.html



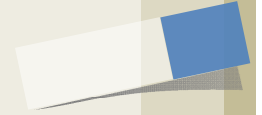
Objectives



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



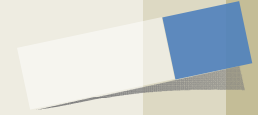
Why is Picking Difficult?



- ❑ 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



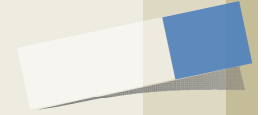
Selection



- ❑ 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



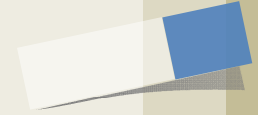
Selection



- ❑ 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

<https://andorsaga.wordpress.com/2010/03/19/compensating-for-webgl-readpixels-implementation-inconsistencie/>



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

