CIS367 - Computer Graphics

Interaction / Animation

Erik Fredericks - frederer@gvsu.edu

Overview

Handling input from user

Animating your scene

But first

Something cool I saw on Twitter!

https://github.com/michal-z/zig-gamedev/tree/main/samples/rasterization

HIIIIIISTOORRRRRYYYYYY

Project Sketchpad (1963) -- Ivan Sutherland establishes user interaction paradigm

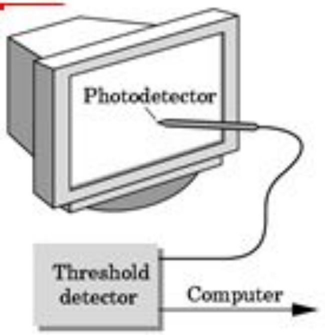
- User sees an **object** on display
- User points to (**picks**) object with input device
  - Light pen / mouse / trackball
- Object **changes**
  - Moves / rotates / morphs
- Rinse and repeat

The University of New Mexico

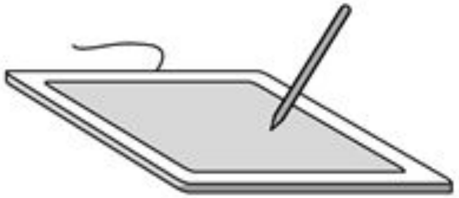
Physical Devices

mouse

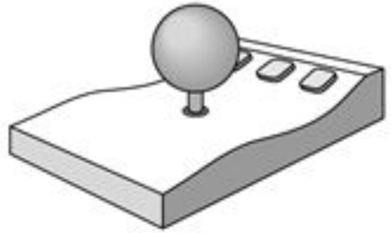
trackball



light pen


data tablet

joy stick


space ball

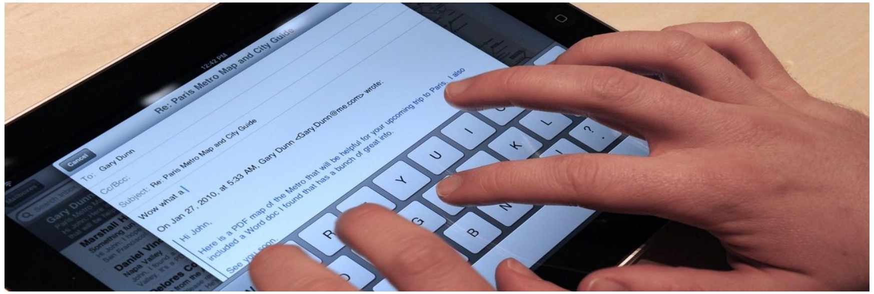
Devices

Described by either **physical properties**

- Mouse / keyboard / trackball / VR handset

or **logical properties**

- Data returned to program via API
  - Position
  - Object identifier
  - etc.



and its **mode**

- How / when input is obtained
  - Request-driven?
  - Event-driven?

Incremental (relative) devices

Devices such as a tablet return position **directly**

Devices such as a mouse/trackball/joystick return **incremental** inputs (velocity)

- **Velocities** must be handled to determine the *absolute position*
  - Rotation of cylinders in ball-mouse
  - Trackball roll
  - Variable sensitivities based on device precision

Logical devices

Consider the following C and Python code:

C++ / C

```
cin >> x;
//
scanf("%d", &x);
```

Python

```
x= input("Enter the value for X")
```

What is the input device?

- Can't tell from this code!
- Keyboard ... file ... other?

Logical input!

- Number is returned **regardless** of physical device!

Graphical logical devices

Graphical input much more complicated than other inputs

- Those are usually numbers, bits, bytes, characters, etc.

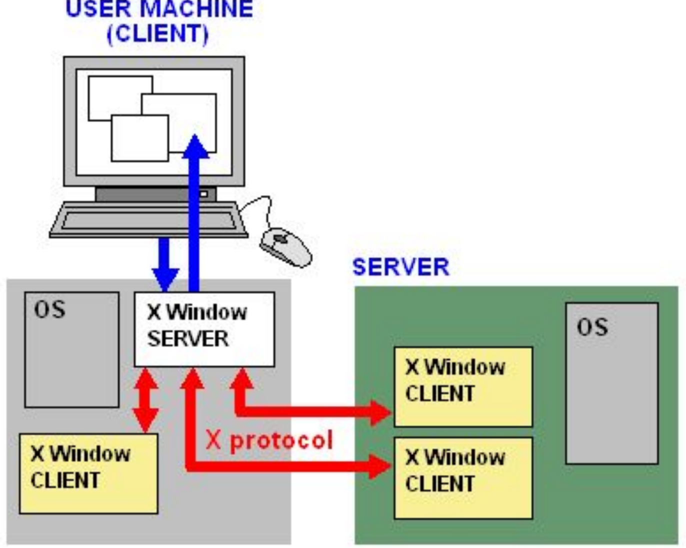
Six types of input were defined by older APIs (GKS, PHIGS)

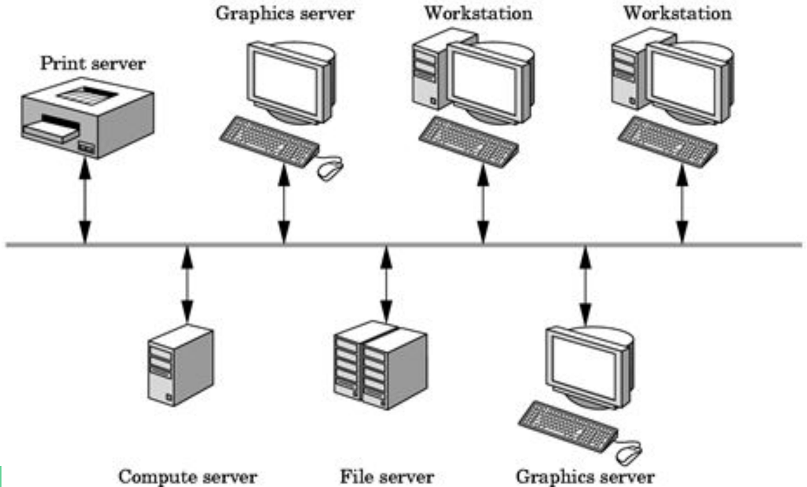
Locator:	return a **position**
Pick:	return **ID** of an object
Keyboard:	return **strings of characters**
Stroke:	return **array of positions**
Valuator:	return **floating point number**
Choice:	return **one of n items**

X Window Input

Introduced client/server model for network of workstations

Client: OpenGL program
Server: Bitmap display with pointing device and keyboard





Input modes

Input devices contain a trigger

- Mouse button
- Key press/release
- Etc.

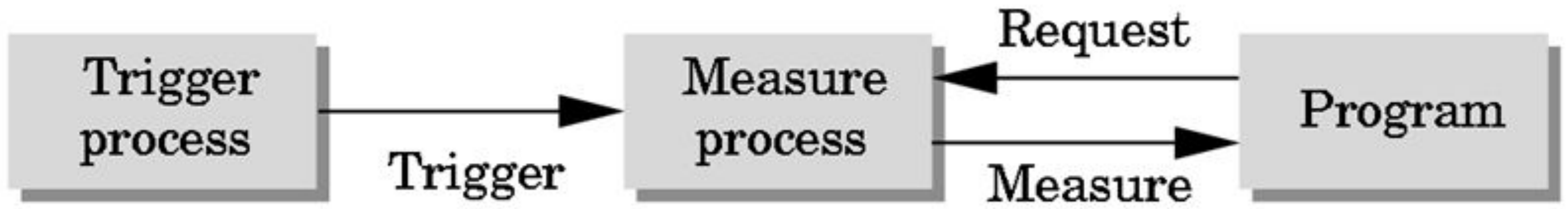
that sends a signal to the operating system

On trigger, devices return information (measure) to operating system

- Mouse:		positional information
- Keyboard:	ASCII key code
- Etc.

Input modes

Input sent **only** when user triggers action



Event modes

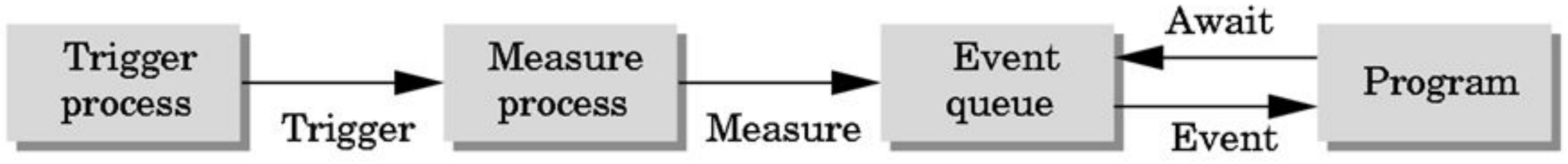
Do we only have one method of input?

Of course not!

- They can also be triggered whenever the user wants!

When triggered, an **event** is generated and placed in the **event queue**

- Consumed by user program



Event types

Window:

- Resize, expose, iconify

Mouse:

- Click one or more buttons

Motion:

- Move mouse

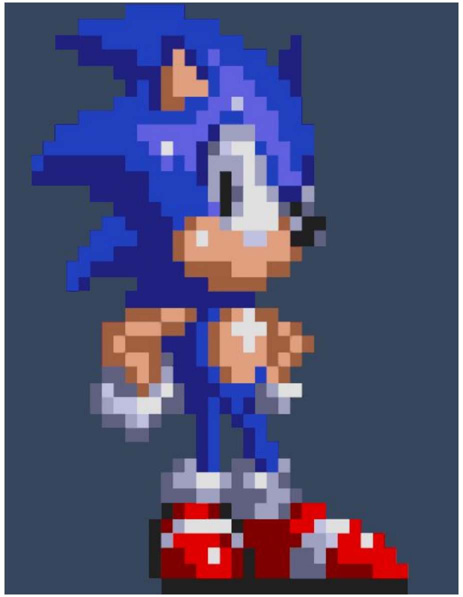
Keyboard:

- Press or release a key

Idle:

- Non-event

Define what should be done if no other event is in queue



Handling user input

We must process the queue!

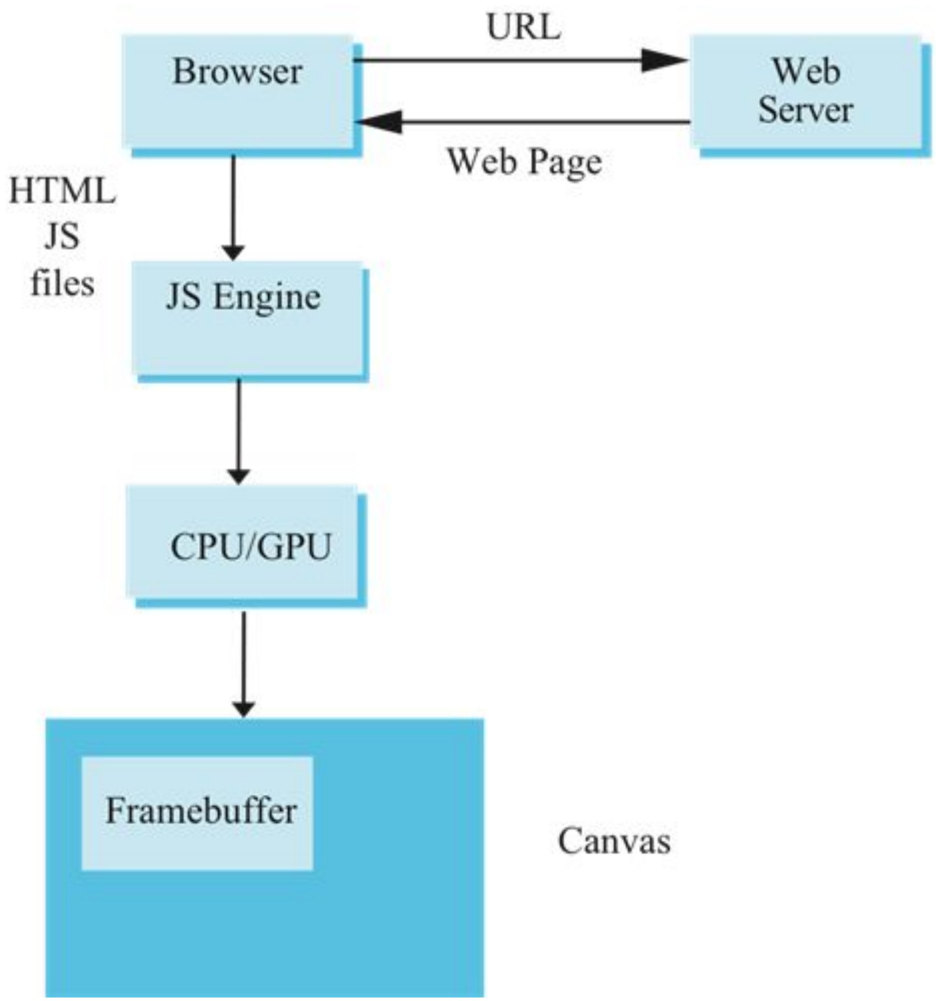
Pic related: it's a user

Callbacks

Pretty much everything these days is callback-driven, get used to it!

Event-driven input: **callback functions** or **event listeners**

1) Define a callback for each event to be recognized (by gfx system)
2) Browser enters event loop and responds to events for registered callback events
3) Callback function executed when event occurs


Browser
URL
Web Server
Web Page
HTML JS files
JS Engine
CPU/GPU
Framebuffer
Canvas

Browser execution

Websites start with an HTML file

- Which … do nothing except contain presentation information!
- May contain shaders
- Loads all other files

However…

- Files are then loaded (asynchronously) and JavaScript files loaded in and executed

Browser then in an **event loop** waiting…

`onload` event

In JavaScript file...

- Everything happens within functions like `init()` and `render()`

- BUT
    - They're not executed...
    - Nothing happens!

To kick things off, use the `onload` window event to start things off

- Think of it as JavaScript's `main` function
- But, it is executed *only after* all files are loaded in

```
window.onload = init;
```

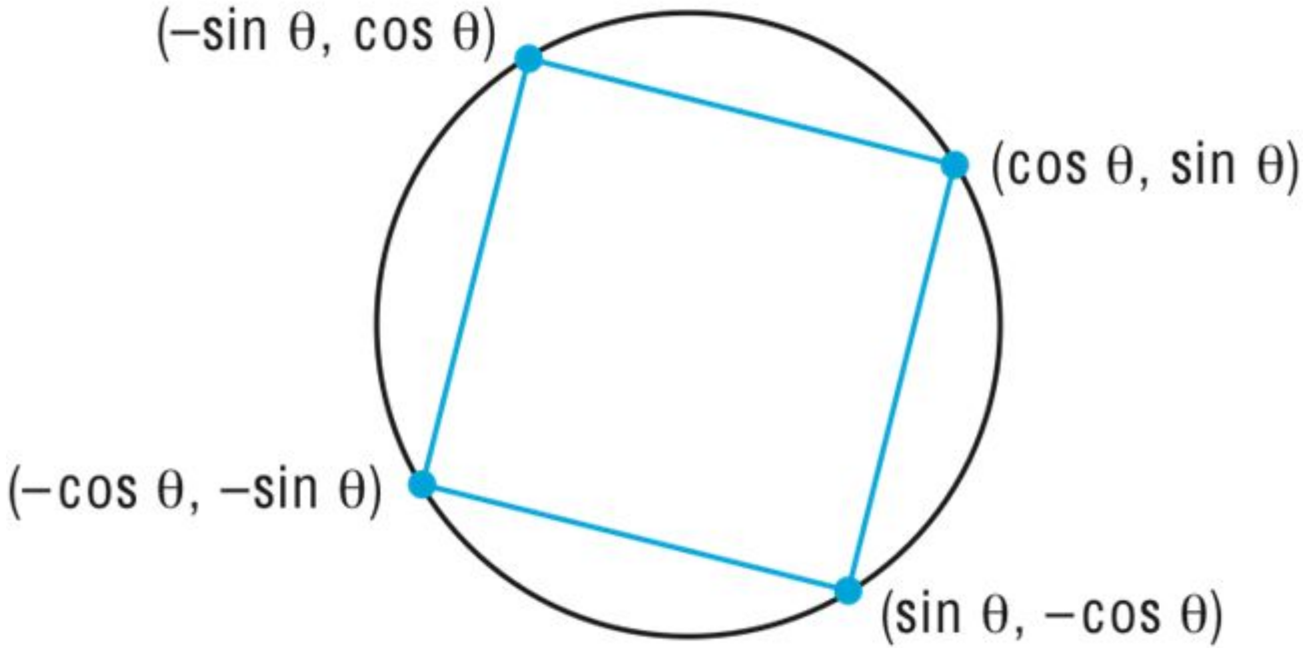
AND NOW TO BLEND THINGS UP A BIT

What better way to learn about **user input** than to spice it up with some **animation**?

Let's get rotatey

4 vertices defined by functions!

Animate by **re-rendering** and **updating** theta!


$(-\sin\theta, \cos\theta)$
$(\cos\theta, \sin\theta)$
$(-\cos\theta, -\sin\theta)$
$(\sin\theta, -\cos\theta)$

"The simple way"

```
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();

}
```

What was wrong with that?

"The better way"

Let's leverage our GPU!

- Send vertices to vertex shader
- Send theta to shader as a **uniform variable**
- **Compute** vertices in the vertex shader
- Render recursively!

Updated 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

Square is being rendered, but to the non-display buffer!

Browser uses **double buffering**

- Front buffer ➔ displayed
- Back buffer ➔ rendered to
- Buffer swap (flip) required to show!

This helps with showing the full render

- Hiding a partial render

Buffer swap

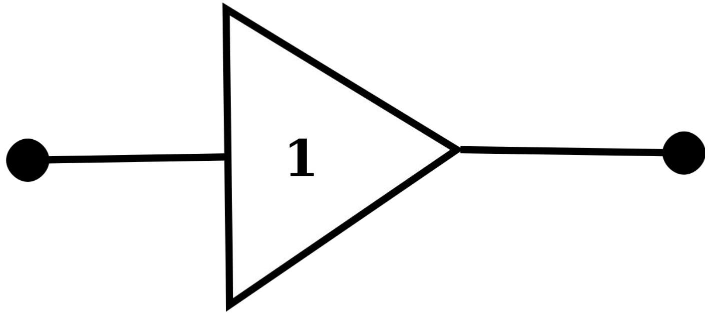
Browser refresh display at approximately 60Hz

- This is not the swap!
- Only refreshes **front buffer**

Buffer swap must occur via an **event**

Two options for this example:

- `Interval` timer
- `requestAnimFrame`



Pic related: a digital buffer

Interval timer

Executes function after set amount of time (milliseconds)

- Also, forces buffer swap!

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

Interval time of 0 will perform swap *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);
}
```

You should call this method whenever you're ready to update your animation onscreen. This will request that your animation function be called before the browser performs the next repaint. The number of callbacks is usually 60 times per second, but will generally match the display refresh rate in most web browsers as per W3C recommendation

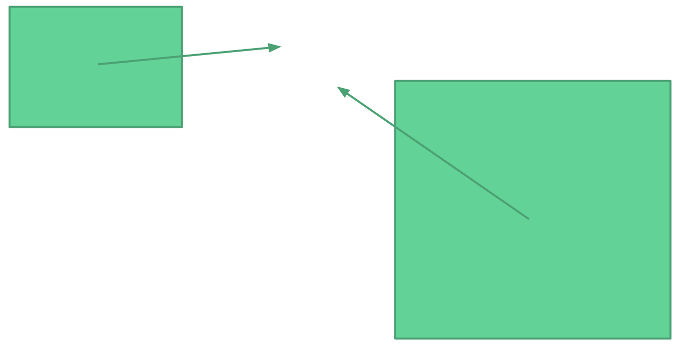
https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

Adding an interval

```
function render()
{
  setTimeout( function() {
    requestAnimFrame(render);                   // set animation

    gl.clear(gl.COLOR_BUFFER_BIT);              // clear viewport
   theta += 0.1;                                // update pos
   gl.uniform1f(thetaLoc, theta);               // update theta
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);      // draw triangle strip
  }, 100);                                      // every 100ms
}
```

In-Class Things!

Let's say you have two rectangles of varying size



Take a couple of minutes and consider how you would check if they had collided!

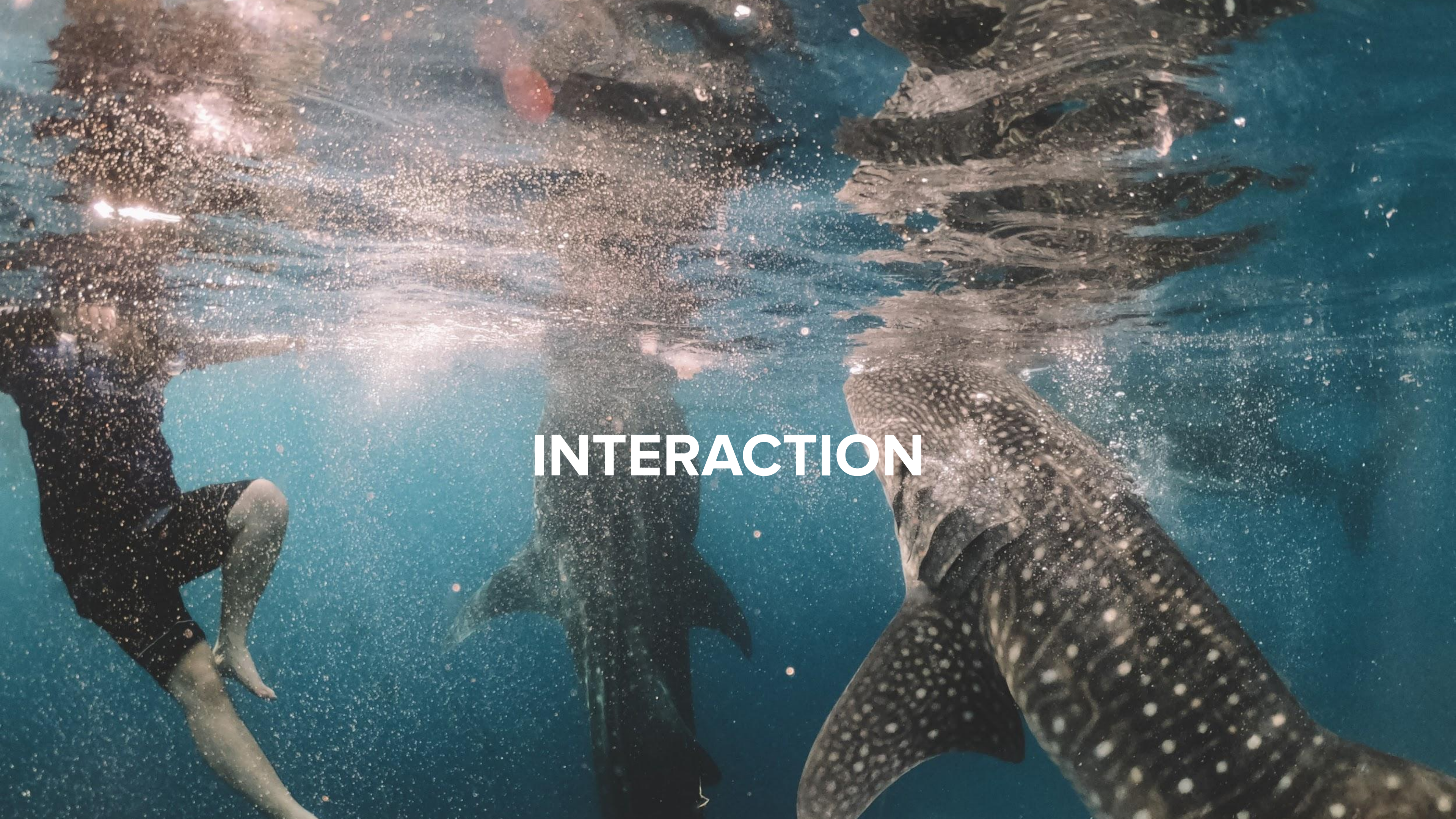
(or if they collided with a wall?)

Couple of interesting things!

"My favorite description was that a CPU is like having someone with a PhD per core. A gpu is like having an army of millions of kindergarteners. Want to do complex math on a lot of data? Hand it to the 8 PhDs. Want to fill in a bunch of tiny spots with a different color? Pitch it to the kindergarteners." -- Reddit

https://youtu.be/frLwRLS_ZR0

A deep rabbit hole: https://www.youtube.com/user/DisneyResearchHub/videos



INTERACTION

What can we use to interact with the program?

Buttons
Menus
Mouse
Keyboard
Reshape (window resize)

How???

- Callbacks / event listeners!

HTML button

Clicking the button will reverse the rotation of the square

- Update render function
- Add button
- Add listener

```
var direction = true; // initialize variable

    ###

// in render function
if (direction) theta += 0.1;
else theta -= 0.1;
```

HTML

HTML5 button tag
`id` ➔ pull in via JavaScript
Clicking generates an `onClick` event

```
<button id="btnDirection">Change rotation direction</button>
```

Any idea how to get a 'prettier' button?
- for instance, to match your game?

Event listener

If we skip the event listener, the event occurs, but nothing happens!
Two possibilities!

```
var btn = document.getElementById('btnDirection');

btn.addEventListener('click', function() {
  direction = !direction;
});
```

```
document.getElementById('btnDirection').onclick = function() {
  direction = !direction;
};
```

onClick variants

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

```
btn.addEventListener('click', function() {
  if (event.shiftKey == 1) { direction = !direction; }
});
```

```
<button onclick='direction = !direction;'>Reverse</button>
```

Updating render

```
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);
}
```

Also: https://stackoverflow.com/questions/47421760/how-to-record-and-display-fps-on-webgl

Menu (HTML `select`)

Each entry is an `option` with a return value specified by `value`

```
<select id='spin_menu' size='3'>
  <option value='0'>Toggle rotation direction</option>
  <option value='1'>Faster</option>
  <option value='2'>Slower</option>
</select>
```

```
var m = document.getElementById("spin_menu");

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

keydown

```
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;
    }
});
```

What if we 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

Requires:

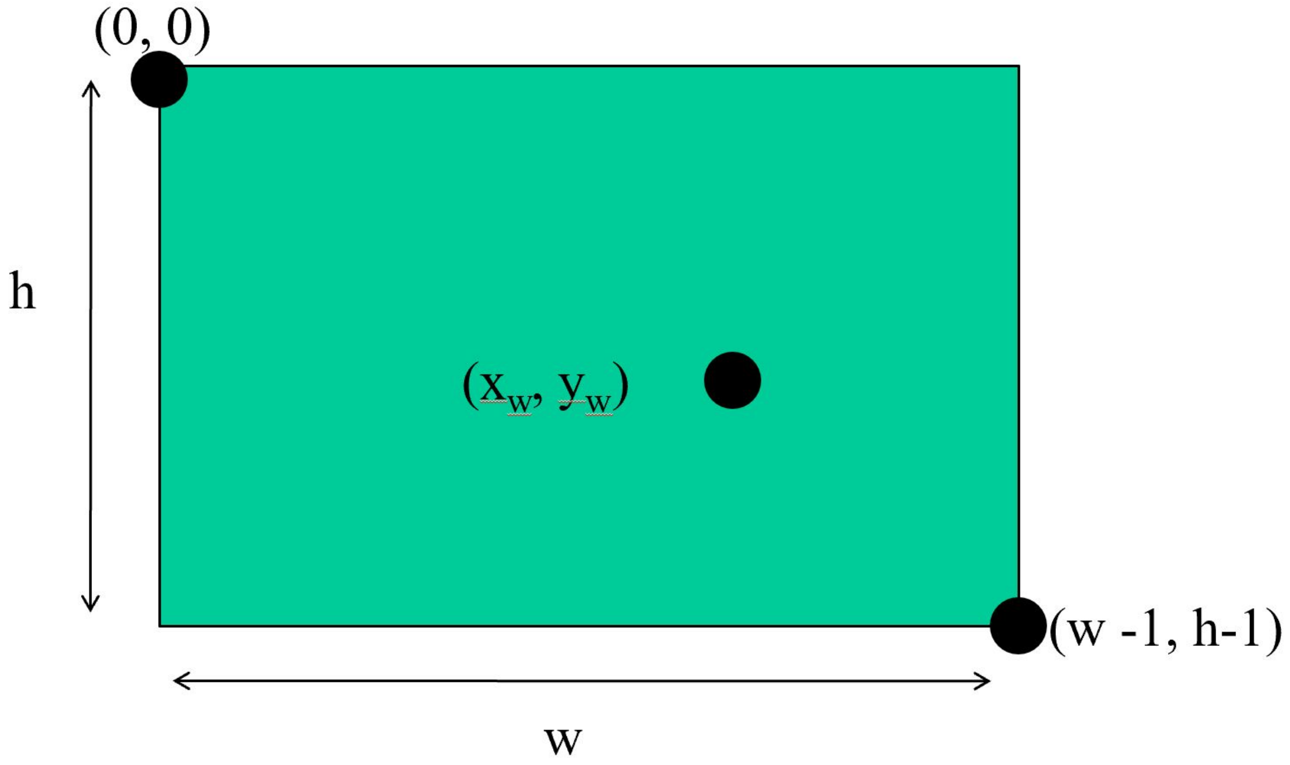
id, minimum / maximum values, step size, initial value

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

###

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

What do we need to talk about before we talk about this little guy?

Browser position != Viewport position

Conversion required!



The following is from the book…

Window ➔ Clip coords

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

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

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

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

Let's get a position from a mouse `click` event

Click position relative to **canvas** width and height!

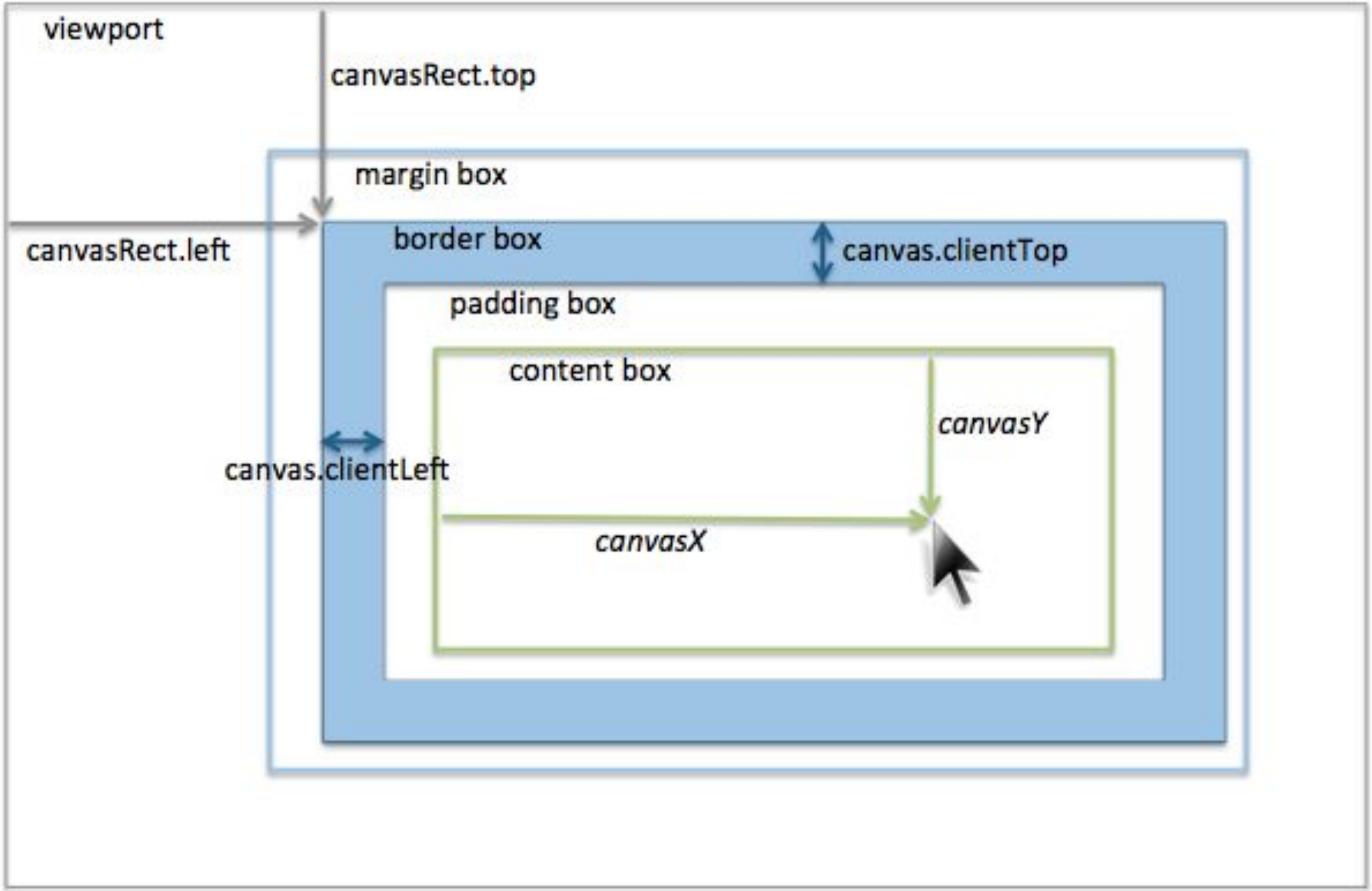
`canvas.width, canvas.height` → `event.clientX, event.clientY`

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

Now let's do it correctly

The formula from the book is slightly off, and not really conducive to concerns like **browser updates**

We need to consider a **bounding rectangle**


viewport
canvasRect.top
margin box
canvasRect.left
border box
canvas.clientTop
padding box
content box
canvasY
canvas.clientLeft
canvasX

```
var rect = gl.canvas.getBoundingClientRect();

// Scale to bounding rectangle of the canvas, as defined by CSS
// and the browser!!!
var newx = ((event.clientX - rect.left) / canvas.width) * 2 - 1;
var newy = ((event.clientY - rect.top) / canvas.height) * -2 + 1;
```

Examples (see GitHub page)

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

More in-class dealies!

Quick chat about term project!

Each person:

1) What is one idea you have?
2) What is one (of many) thing you need to learn to accomplish it?

Window events

Events can be generated by anything affecting the window/canvas

- Moving/exposing window
- Resizing
- Opening new window
- Iconifying/deiconifying window (~generally minimize to icon -- suspend drawing)

Other events from other applications can trigger events as well!

Reshape

Use mouse to change canvas size

- Contents must be redrawn to fit (within reason)

Options:

- Display same objects but change size
- Display more/fewer objects of same size
- Clip

Generally want to keep proportions same

`onresize` event

Returns size of new canvas on `window.innerHeight, window.innerWidth` variables

- Change `canvas.width, canvas.height` in relation!

Maintaining a square

```
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);
   }
};
```

And now, let's do this:

Mouse picking

- Selection
- Off-screen buffer / color
- Bounding boxes

Picking can be difficult!

Why?

- How do you map a canvas point to an object?
- Non-uniqueness
- Pipeline has a 'forward-nature'
- "Exact" position with pointing device

Selecting

Actually supported in OpenGL pipeline

- Each primitive assigned an **id** in OpenGL
    - Which object it belongs to

- During rendering, **id** of primitives rendered near mouse are put in a **hit list**

- **Check hit list after rendering!**

Picking: selection

Create a small window around mouse

- Track if primitive renders to this window
- **Do not display this window!**
    - Render **off-screen** to either an extra color buffer or a back buffer and don't swap
- Requires depth into hit record (does it cross multiple buffers)
- Can do in WebGL as well!

Picking: color

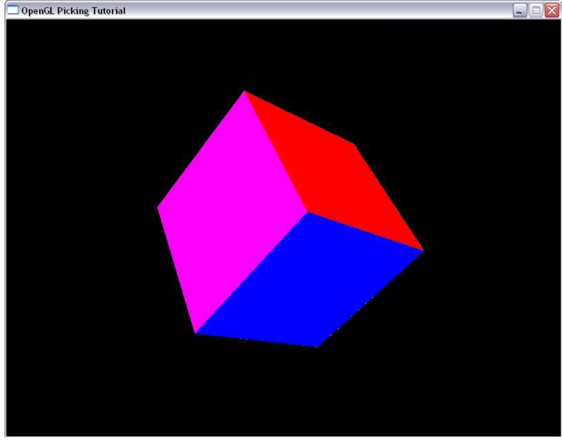
Use `gl.readPixels` to get color in window

Use color to identify object, however

- Multiple objects can have same color
- Shading / texturing objects will result in many colors

Assign unique color **per object** and render off-screen

- `gl.readPixels` to get color at mouse location
- Lookup table to map color to object

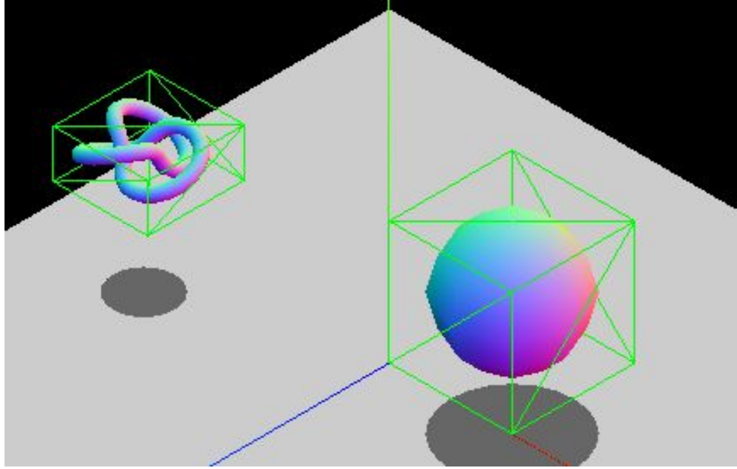


Picking: bounding box

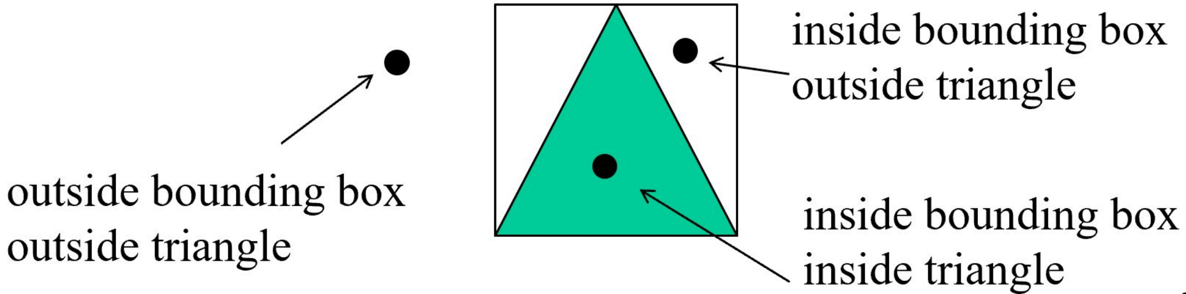
Extra render required with prior methods!

Use table of axis-aligned bounding boxes (AABB)

- Map mouse location through table



```
function isPointInsideAABB(point, box) {
  return (point.x >= box.minX && point.x <= box.maxX) &&
         (point.y >= box.minY && point.y <= box.maxY) &&
         (point.z >= box.minZ && point.z <= box.maxZ);
}
```



Back to CAD

Let's build a model up from mouse clicks

- Essentially, we're *buffering* vertices
- (cad1/cad2 for reference)

This is a very common practice ... we want to build *something*

First, the shaders

If we are building up a model vertex by vertex, what do we need to change?

Anything in the vertex shader?

Anything in the fragment shader?

NOPE!

Same deal!

Pass-through vertex + color variable

```
    gl_Position = vPosition;
    fColor = vColor;
```

Pass-through fragment

```
    gl_FragColor = fColor;
```

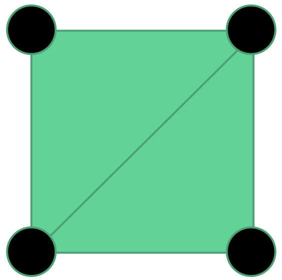
The application will be different though

We need to understand the concepts of *indexing* into our buffer

Enter the `bufferSubData` function

First we define our buffers as usual:

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

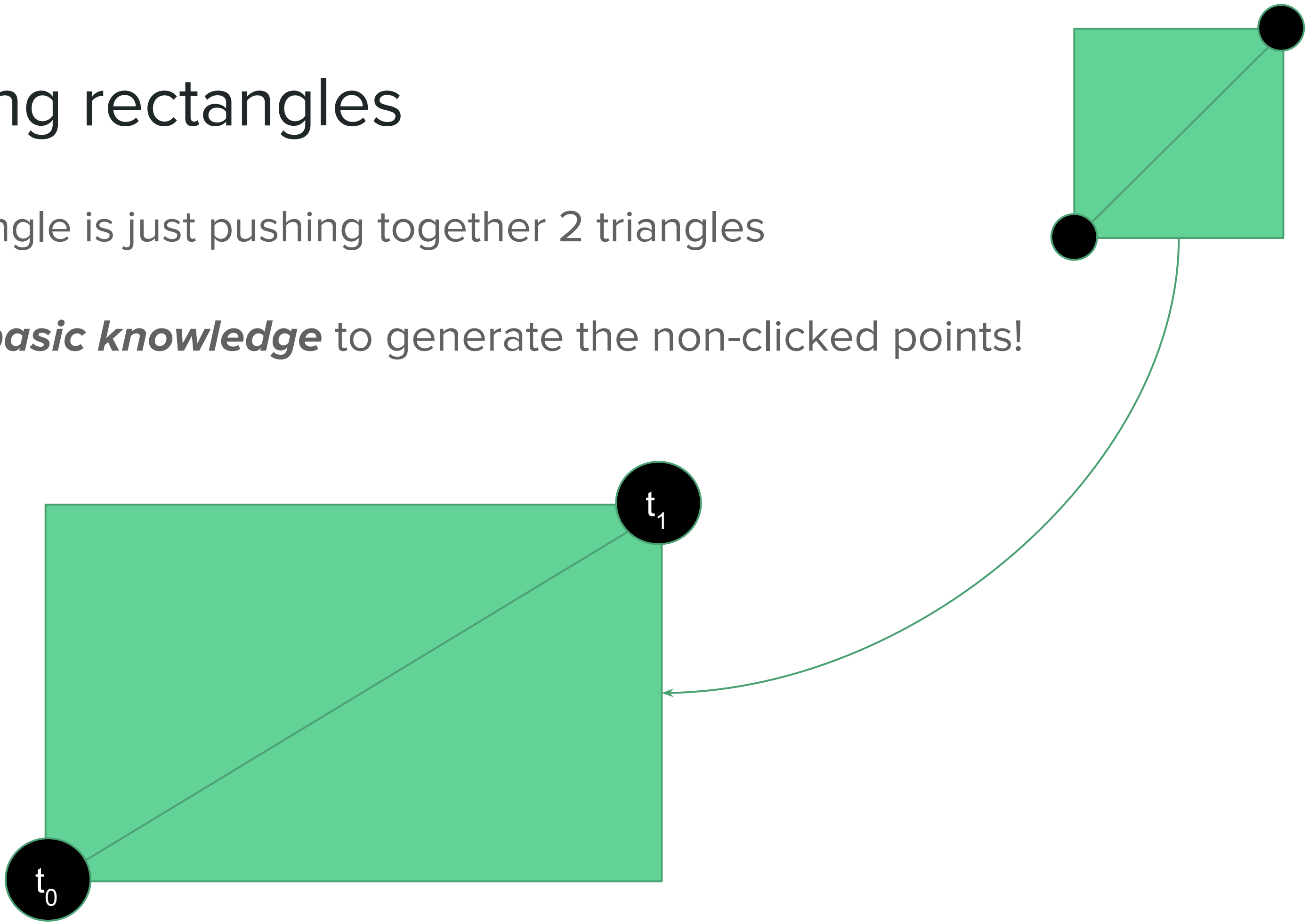


And index into it later (after mouse clicks)

CAD - drawing rectangles

So drawing a rectangle is just pushing together 2 triangles

We can use some ***basic knowledge*** to generate the non-clicked points!



The algorithm

Get a click

- Record the location

Get a second click

- Record the location
- Generate the other points via vertex mixing

Render!

- Triangle fan appropriate here

`bufferSubData?`

Bind our ARRAY_BUFFER to our whims

Use `bufferSubData` to write our new vertices into the pre-defined array

- Make sure to track the index into the array, coupled with how many vertices we needed

`render?`

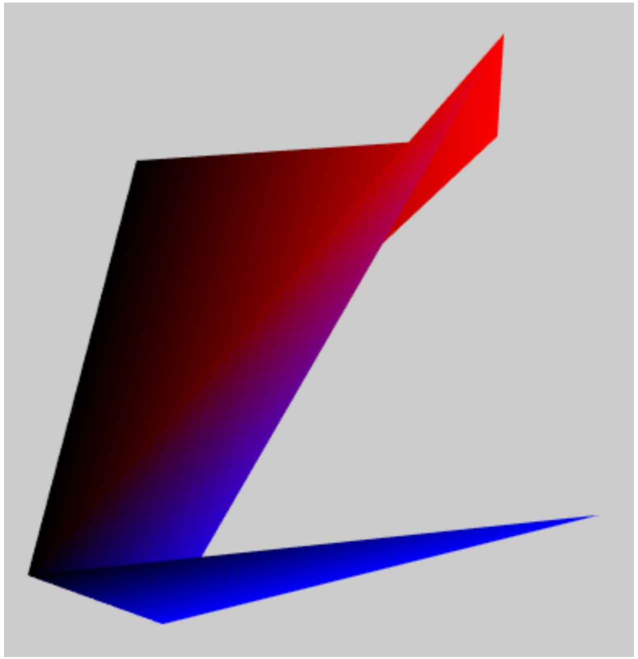
Same, but different!

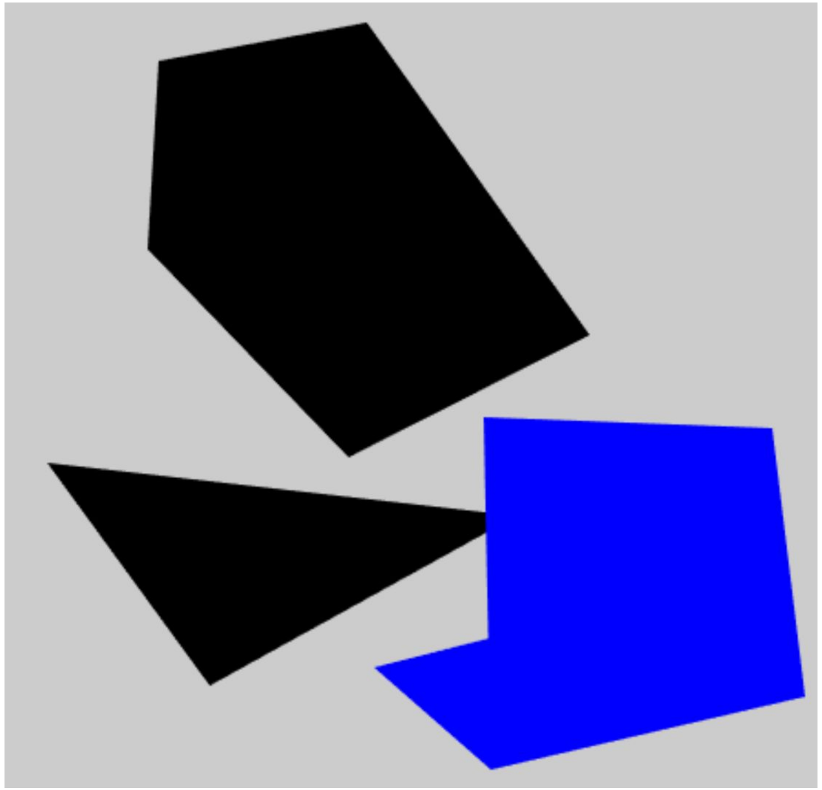
Call `drawArrays` for each square (TRIANGLE_FAN) we have in our buffer

Why not just all at once?

Ok squares were easy....

Let's take a look at **polygons** now





Shaders?

No change

`render?`

No change

Application?

Change, of course

BUT, not a major change!

- Instead of stopping every 2nd click, we will keep adding vertices until the **button** is clicked

- Then, we'll keep track of the # of polygons with a counter
    - Necessary for indexing

- And, we'll push down the color with each vertex instead of tracking it object-by-object

```
gl.bufferSubData(gl.ARRAY_BUFFER, 8*index, flatten(t));

gl.bufferSubData(gl.ARRAY_BUFFER, 16*index, flatten(t));
```

`index`	→	current vertex we're looking at
`numPolygons`	→	current polygon we're looking at

~~Homework~~!

Make a copy of your triangle rotation script from the last lab

1) Add a slider to change the value of **theta**
    a) Up to you to set the bounds and step
    b) Don't forget that a slider value comes in as a **string!**

1) Add a button to **start** or **stop** its rotation

~~Due **Wednesday night @ 11:59pm**~~
TBD - depends on when we get here!