Lecture ~~3~~ 4
More Graphics 101
More Web Graphics

February 2, 2023

A grab bag of topics...

- Buffering and Frame Rate
- More on the pen model
- Complex Polygons
- Events and Canvas
- Coordinate Systems and Transformations

Three Questions...

When do I draw?

when it's your turn!

What do I draw?

Primitives!

Where do I draw?

In the Canvas coordinate system

Buffers

Frame Buffer / Color Buffer ←

(and many more to come)

Memory used to story an image as pixels

Another Important Distinction in Displays

Continuous vs. Flicker/Strobe

Appearing Continuous

Flicker Fusion ←

~~**not** persistence of vision~~

Important Issues in Flicker Fusion

Frame Rate $\leftarrow$

Consistency $\leftarrow$

How a movie projector works

Flash (shutter opens)

Flash (shutter opens)

Advance Film

Flash (shutter opens)

24fps

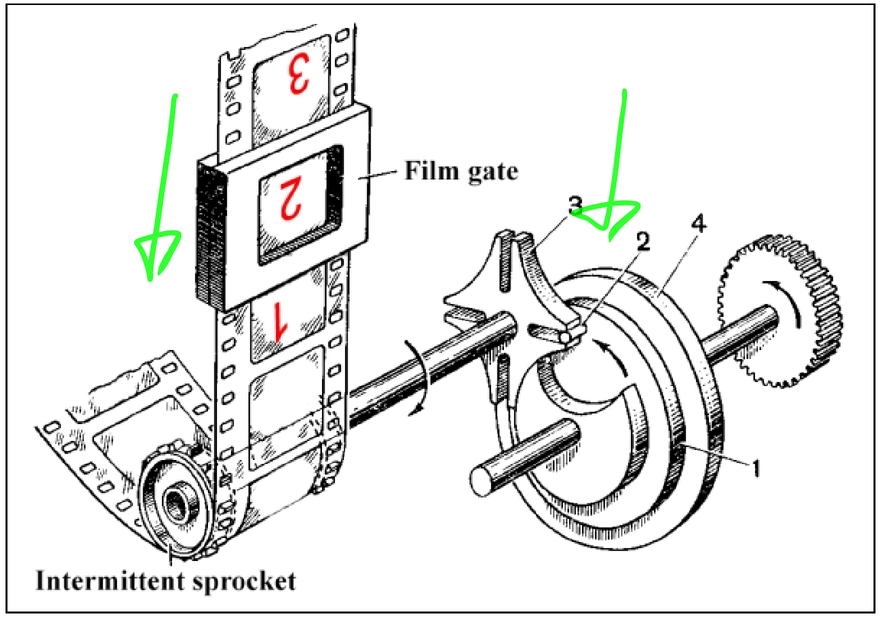
Flash (shutter opens)

Advance Film

Flash (shutter opens)

and so on ...

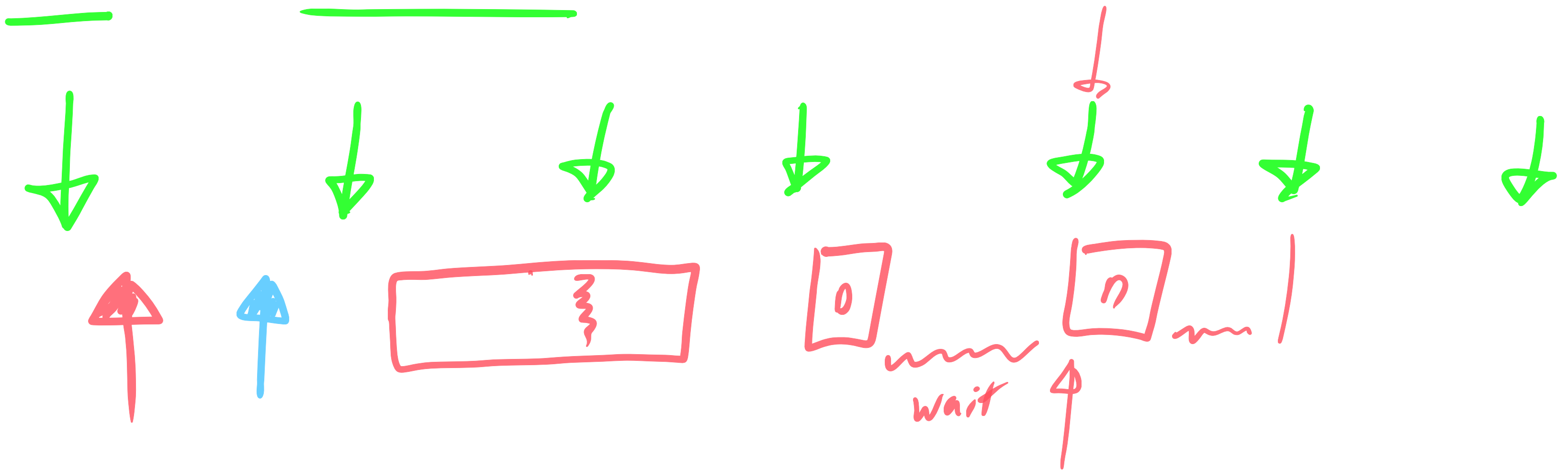
Lumiere brothers, 1894 (not Edison!)



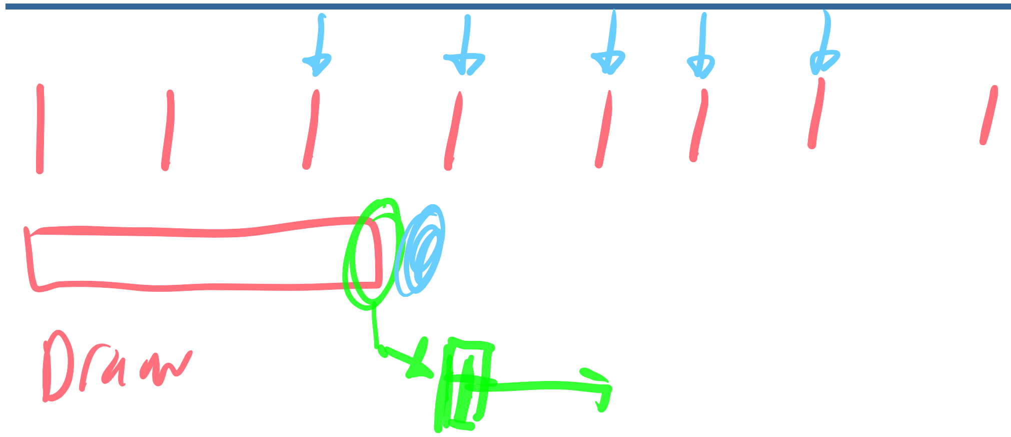
Most computer displays are Flicker-Based

Animation and Redraw

Erase and start over

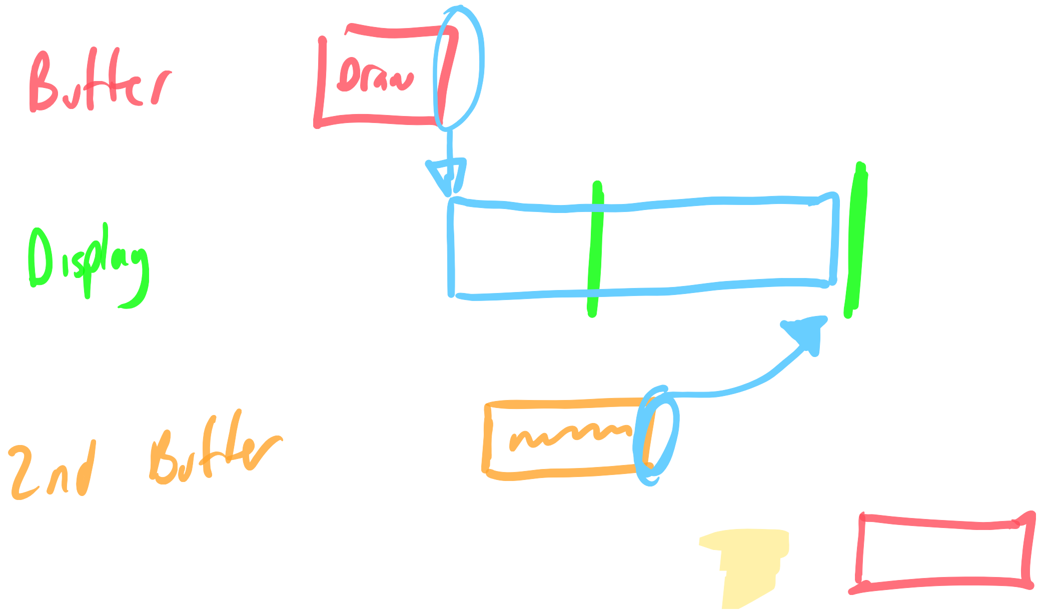


Display Synchronization (Buffering)

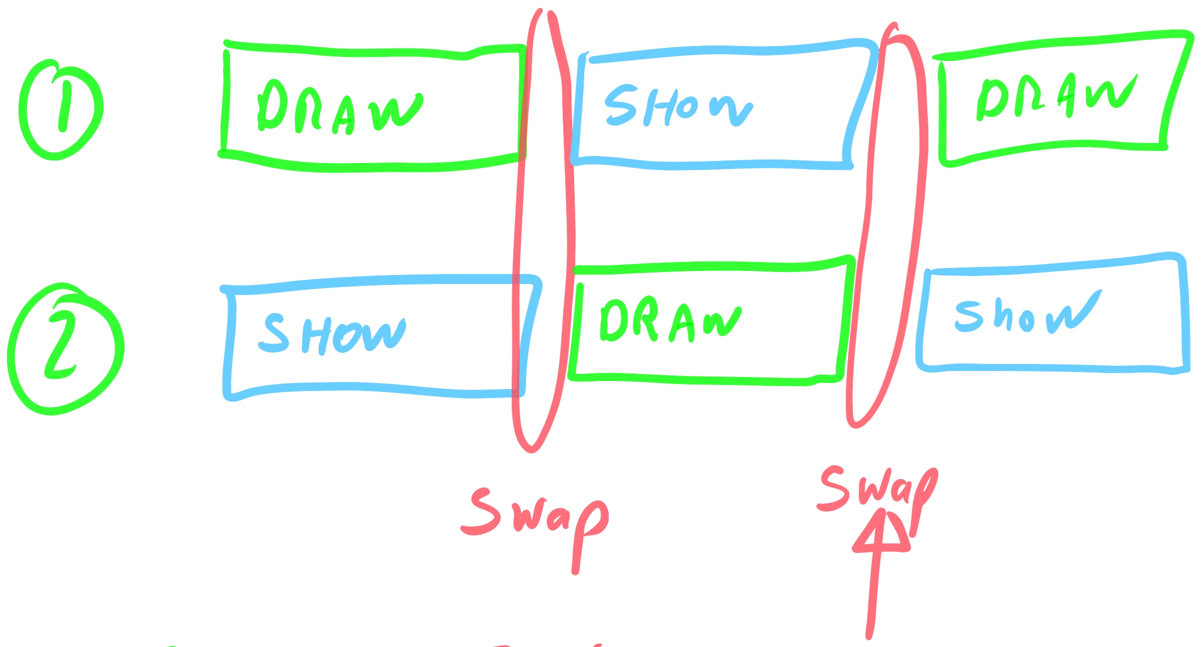


Buffering

What if you draw too slowly? or too fast?



Double Buffering



Draw - Back

Show - Front

Why double buffer?

- only show finished images
- frame rate constancy $\leftarrow$ helps

Buffering and Web Graphics?

The web browser takes care of this

(we lose control)

`window.requestAnimationFrame` waits until after a buffer swap

(in simplified theory)

Three Questions...

When do I draw?

when it's your turn!

What do I draw?

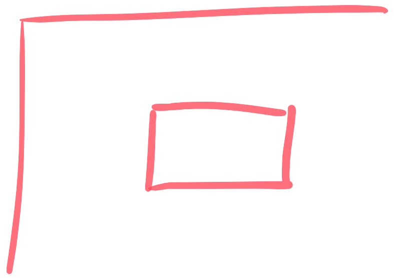
Primitives!

Where do I draw?

In the Canvas coordinate system

Canvas Primitives

- Axis aligned rectangles
- All other shapes (paths)
- Images
- Text



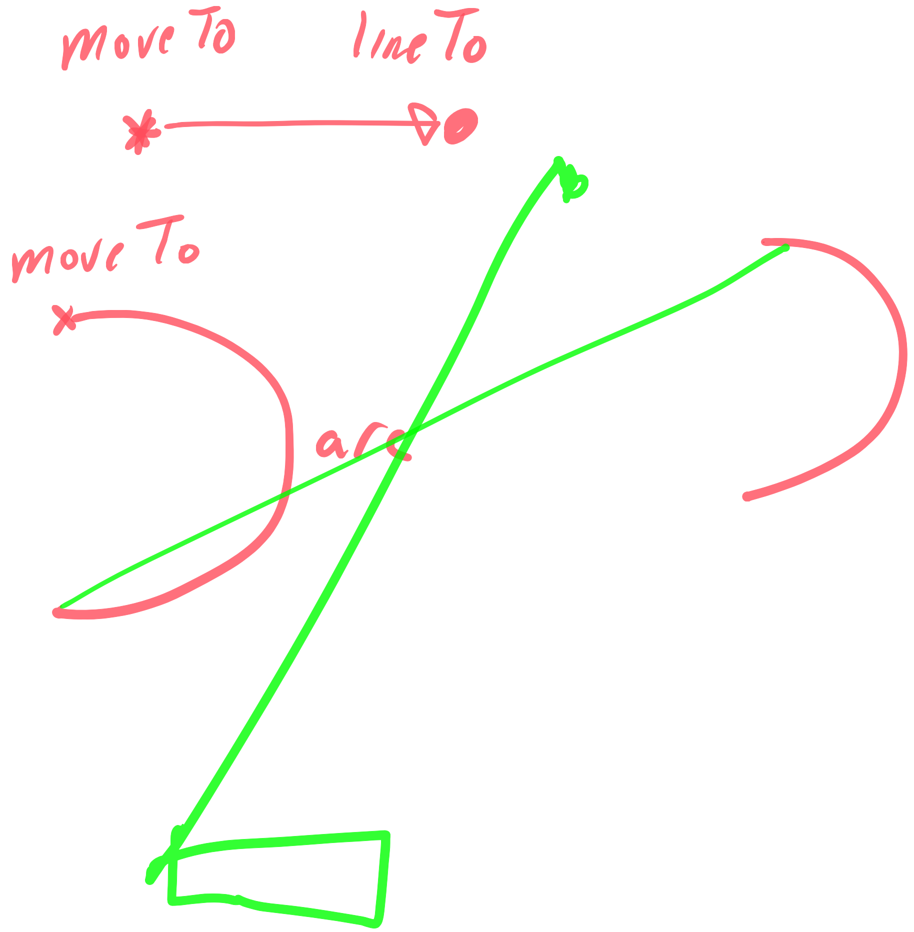
The Pen Model

Methods use the **current pen position**

Methods add to the **current path**

- `moveTo`
- `lineTo`
- `closepath`

- `arc`, `arcTo`, `curveTo`, ...



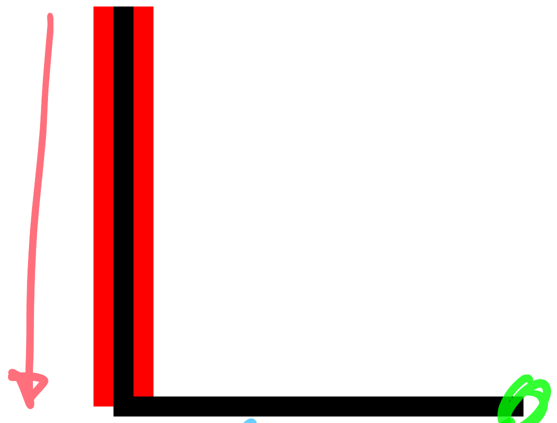
Stroke/Fill the entire path!

The entire path is redrawn with the current pen!

```
    context.beginPath();
    context.strokeStyle = "red";
    context.lineWidth = 12;
1   context.moveTo(20,20);
2   context.lineTo(20,100);
    context.stroke();

    context.strokeStyle = "black";
    context.lineWidth = 4;
3   context.lineTo(100,100);
    context.stroke();
```

3



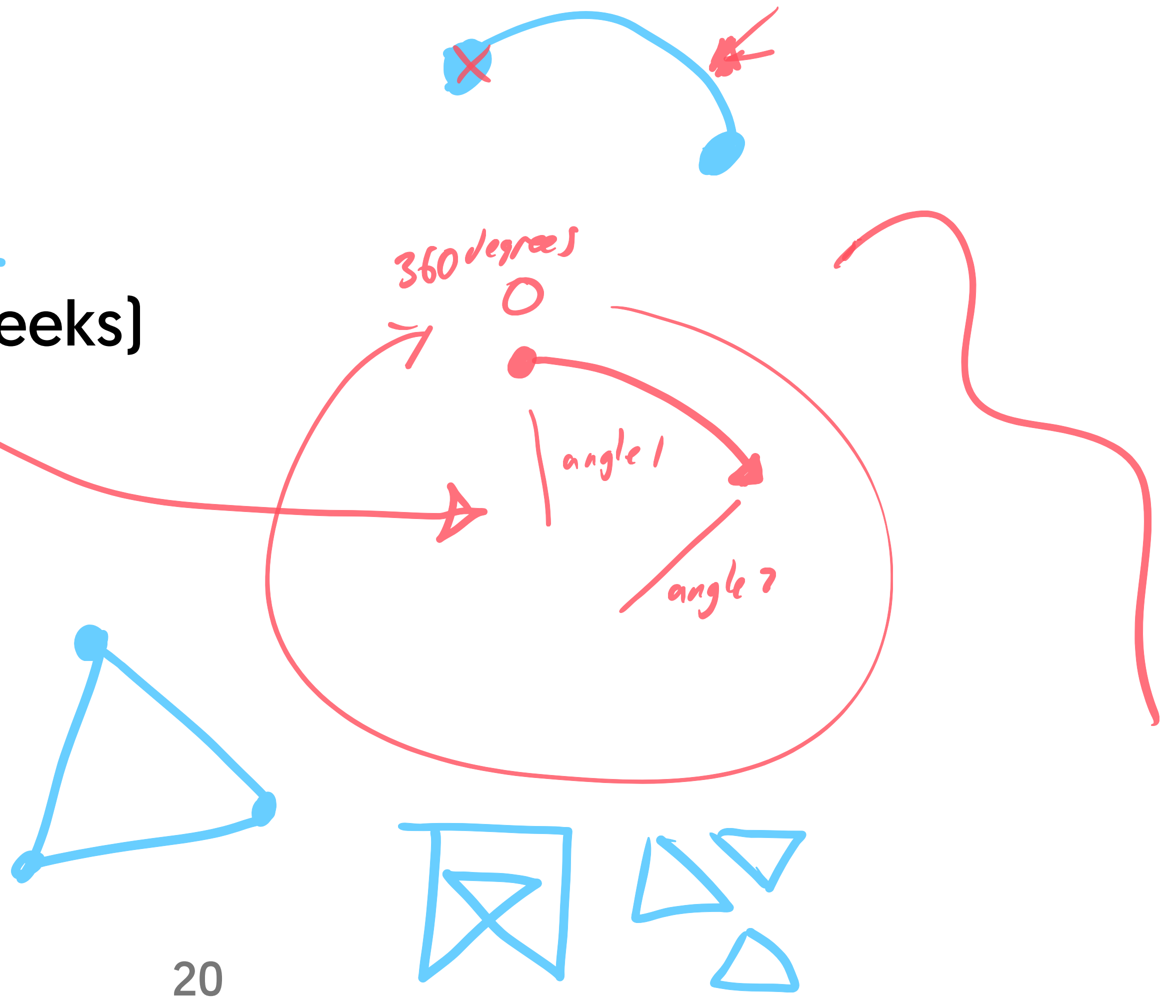
Other Shapes

More Path Operators

- arcs (circles) `arc` vs. `arcTo`
- curves (Bézier - wait a few weeks)

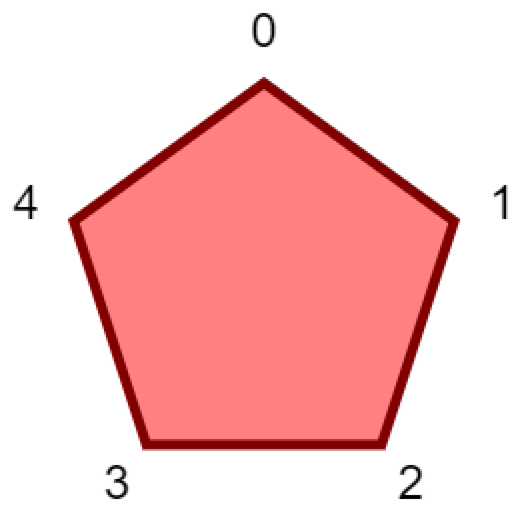
Polygon filling rules

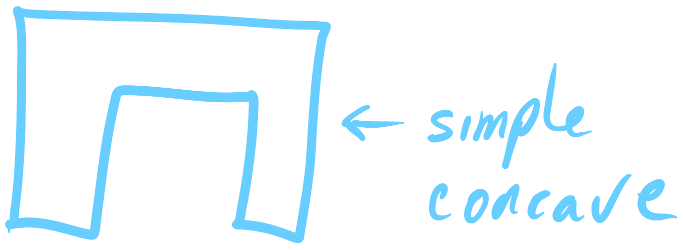
- non-convex shapes
- non-simple (crossings)
- disconnected (holes)



Convex = simple

```
context.beginPath();
context.closePath();
context.moveTo(...pent[0]);
context.lineTo(...pent[1]);
context.lineTo(...pent[2]);
context.lineTo(...pent[3]);
context.lineTo(...pent[4]);
context.closePath();
context.fill();
context.stroke();
```





JavaScript Tip of the Day:
Spread Syntax

`pent[0]` is an array `[200,100]`

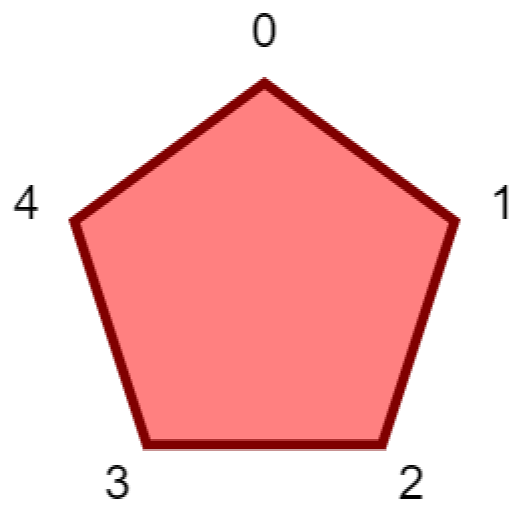
`context.moveTo()` takes 2 parameters x, and y

`context.moveTo(pent[0][0],pent[0][1])` is clunky

`context.moveTo(...pent[0])` uses the **spread operator**

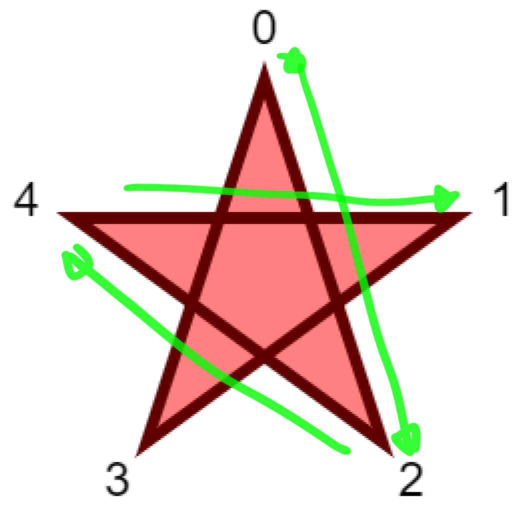
Convex

```
context.beginPath();
context.closePath();
context.moveTo(...pent[0]);
context.lineTo(...pent[1]);
context.lineTo(...pent[2]);
context.lineTo(...pent[3]);
context.lineTo(...pent[4]);
context.closePath();
context.fill();
context.stroke();
```

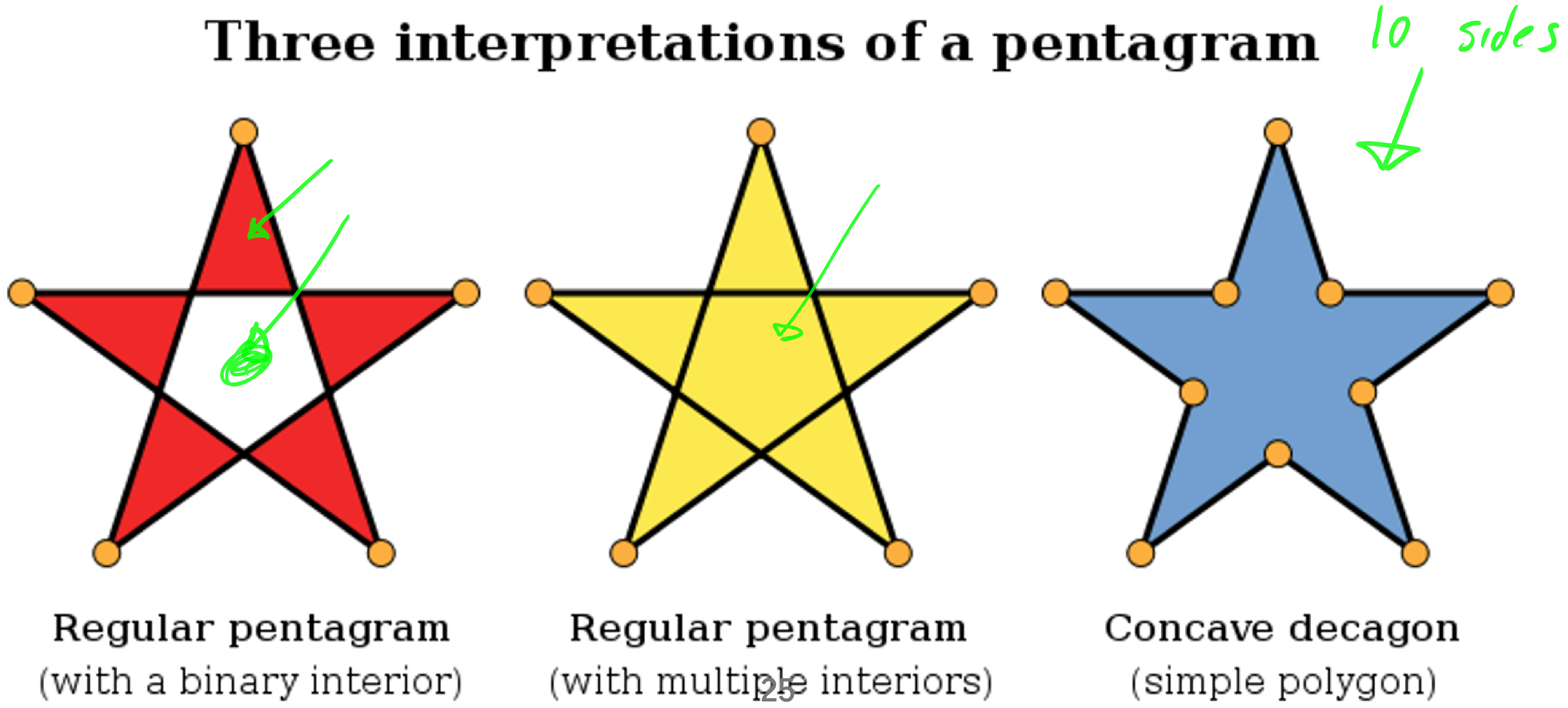


Re-order vertices (lines cross)

```
context.beginPath();
context.closePath();
context.moveTo(...pent[0]);
context.lineTo(...pent[2]);
context.lineTo(...pent[4]);
context.lineTo(...pent[1]);
context.lineTo(...pent[3]);
context.closePath();
context.fill();
context.stroke();
```

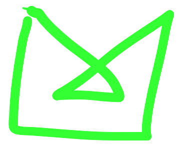


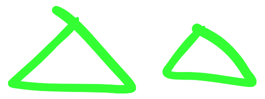
5 sides vs. 10 sides?

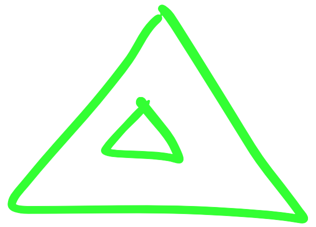


Non-Simple Polygons

- Edges Cross
- Edges are disconnected (multiple loops)
- Not simple to define inside and outside
- We'll use different rules



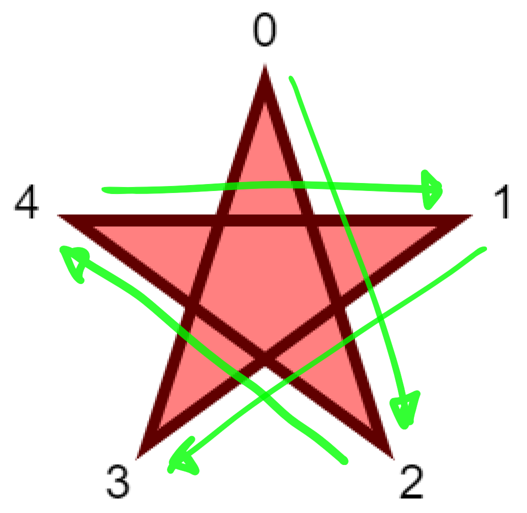




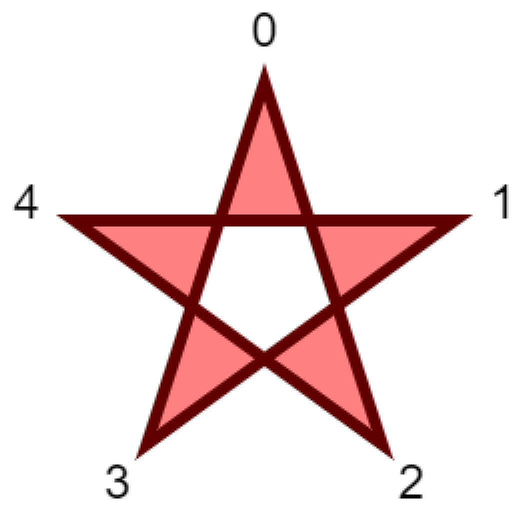
- Canvas lets you make non-simple polygons
- Canvas gives you different rules to interpret them

Two Different Rules

Non-Zero Winding



Even-Odd



Even / Odd

```
context.fill("evenodd");
```

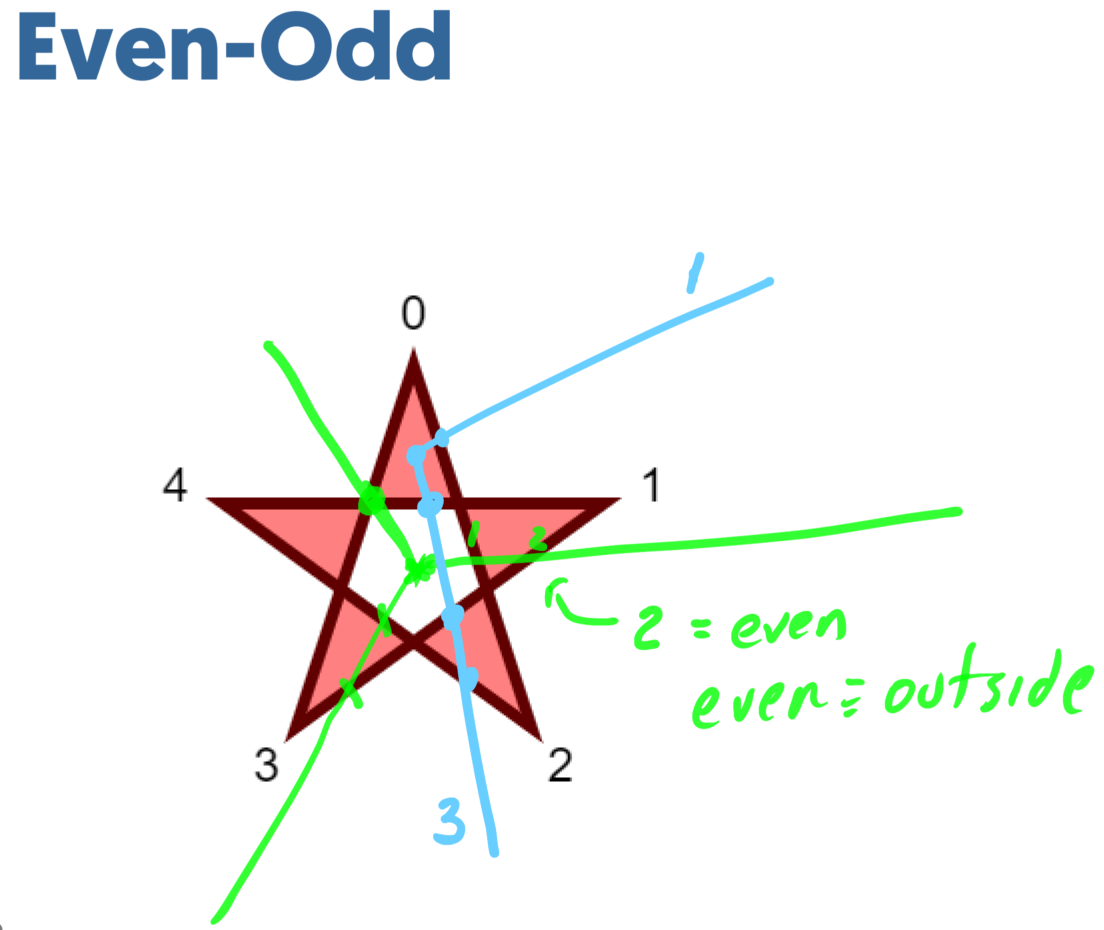
Pick any point

Go to infinity in any direction

Count the number of crossings

Even (includes 0) = outside

Odd = inside



Winding (non-zero)

```
context.fill();
```

Count the "loops" around a point

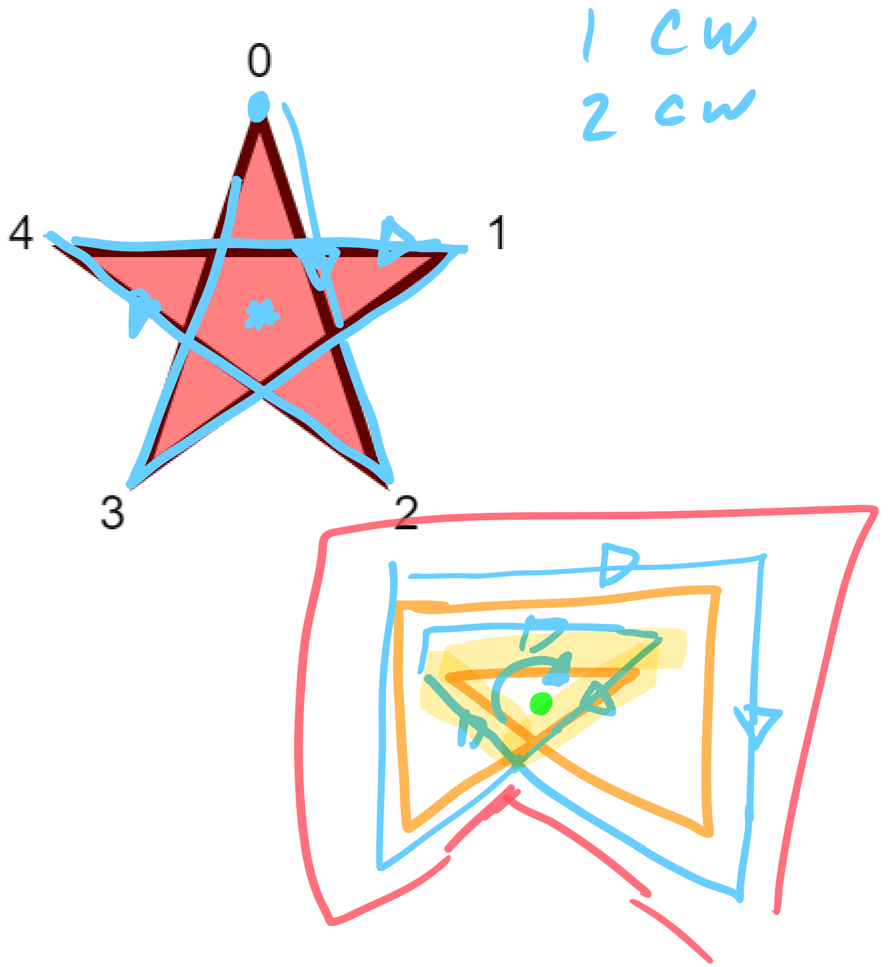
+1 for clockwise

-1 for counter-clockwise

order matters

inside if total is not zero

~~(inside if odd - Adobe, not Canvas)~~

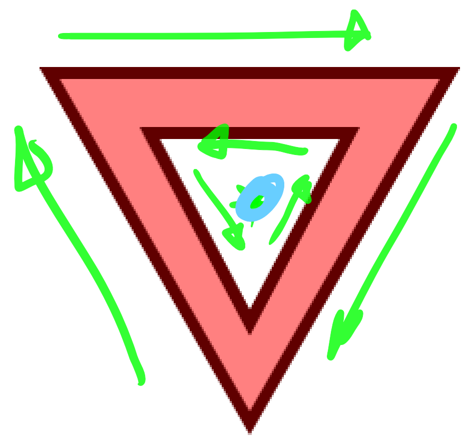


Why use winding rules?

```
context.beginPath();     // clockwise
context.moveTo(100,100);
context.lineTo(300,100);
context.lineTo(200,275);
context.closePath();

context.moveTo(150,130); // counter
context.lineTo(200,225);
context.lineTo(250,130);
context.closePath();

context.fill();
context.stroke();
```

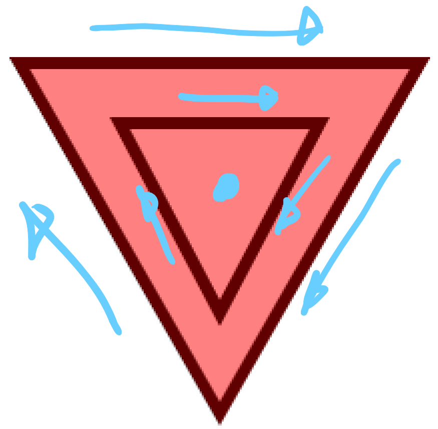


Use direction to control insides

```
context.beginPath();
context.moveTo(100,100); // clockwise
context.lineTo(300,100);
context.lineTo(200,275);
context.closePath();

context.moveTo(150,130); // clockwise
context.lineTo(250,130);
context.lineTo(200,225);
context.closePath();

context.fill();
context.stroke();
```

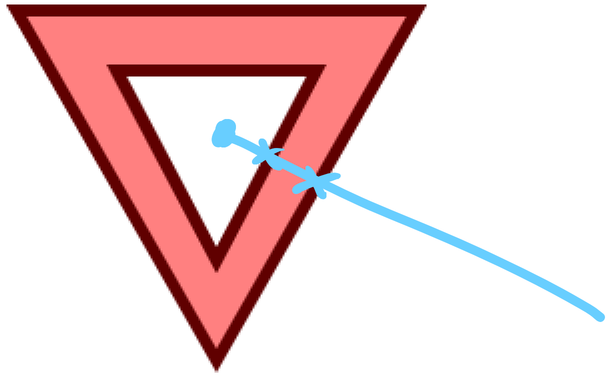


Even Odd is Easier (?)

```
context.beginPath();
context.moveTo(100,100); // clockwise
context.lineTo(300,100);
context.lineTo(200,275);
context.closePath();

context.moveTo(150,130); // clockwise
context.lineTo(250,130);
context.lineTo(200,225);
context.closePath();

context.fill("evenodd");
context.stroke();
```



Example

In Practice...

Non-Simple Polygons are rare

Most APIs only give you simple polygons

OpenGL only gives you **triangles**

A less esoteric point...

What do the vertex positions mean?

Where do I draw?

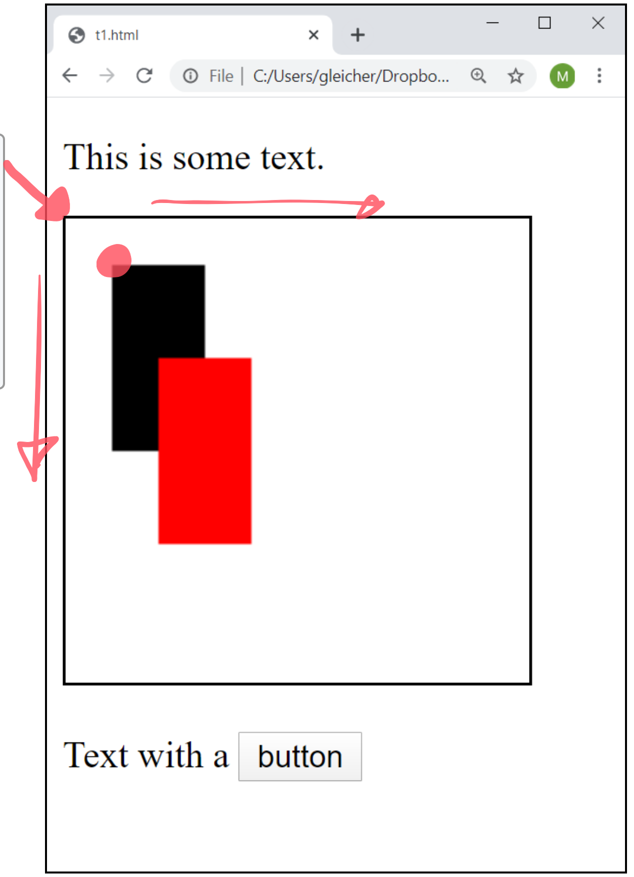
Points (x,y) in the **current coordinate system**

```
            x  y    w    h
context.fillRect(20,20, 40, 80);
context.fillStyle = "red";
context.fillRect (40,60,40,80);
```

Default coordinates:

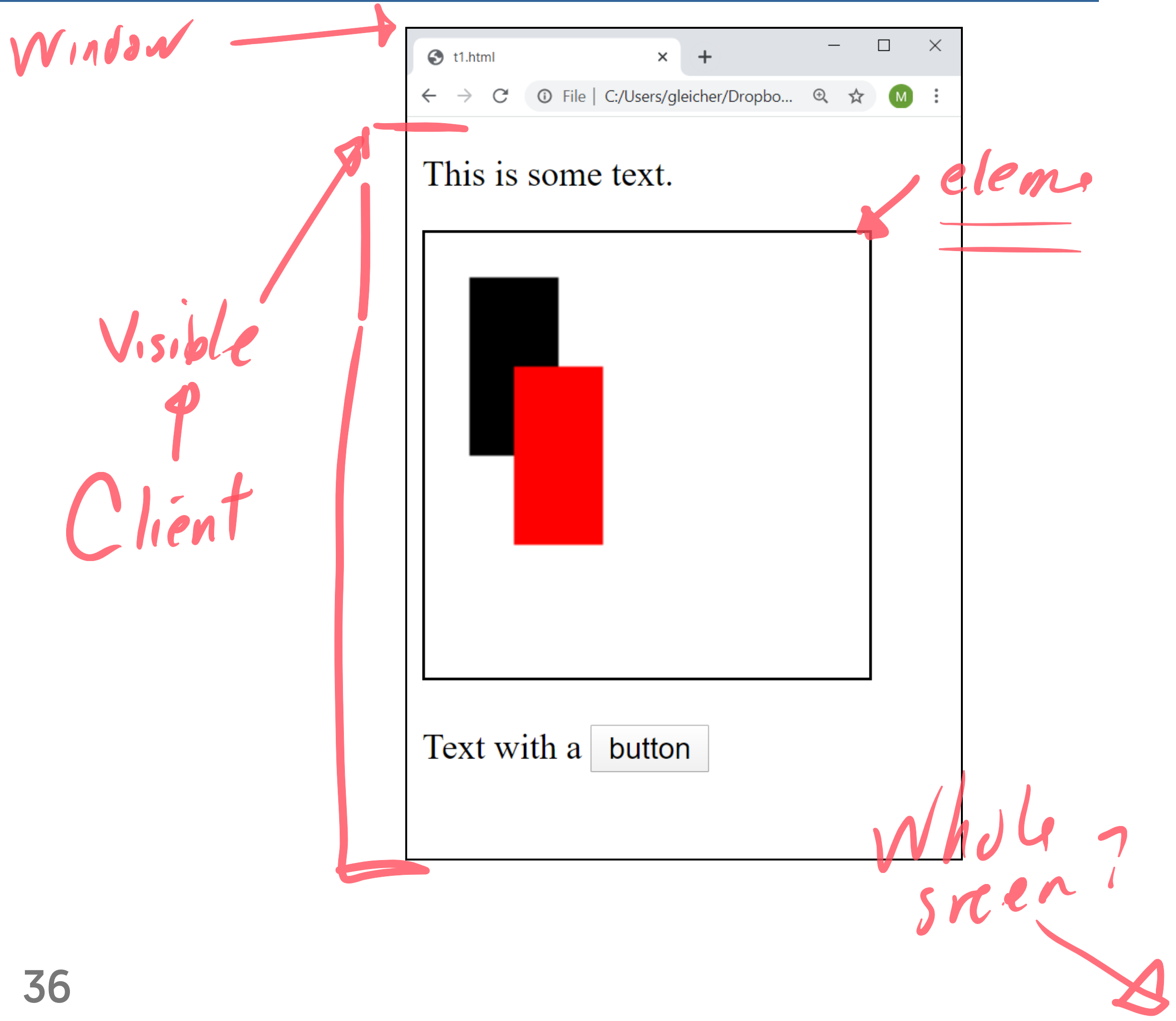
- origin at top left (of canvas)
- x to the right in "html pixels"
- y down in "html pixels"

Convenient (for the Canvas)



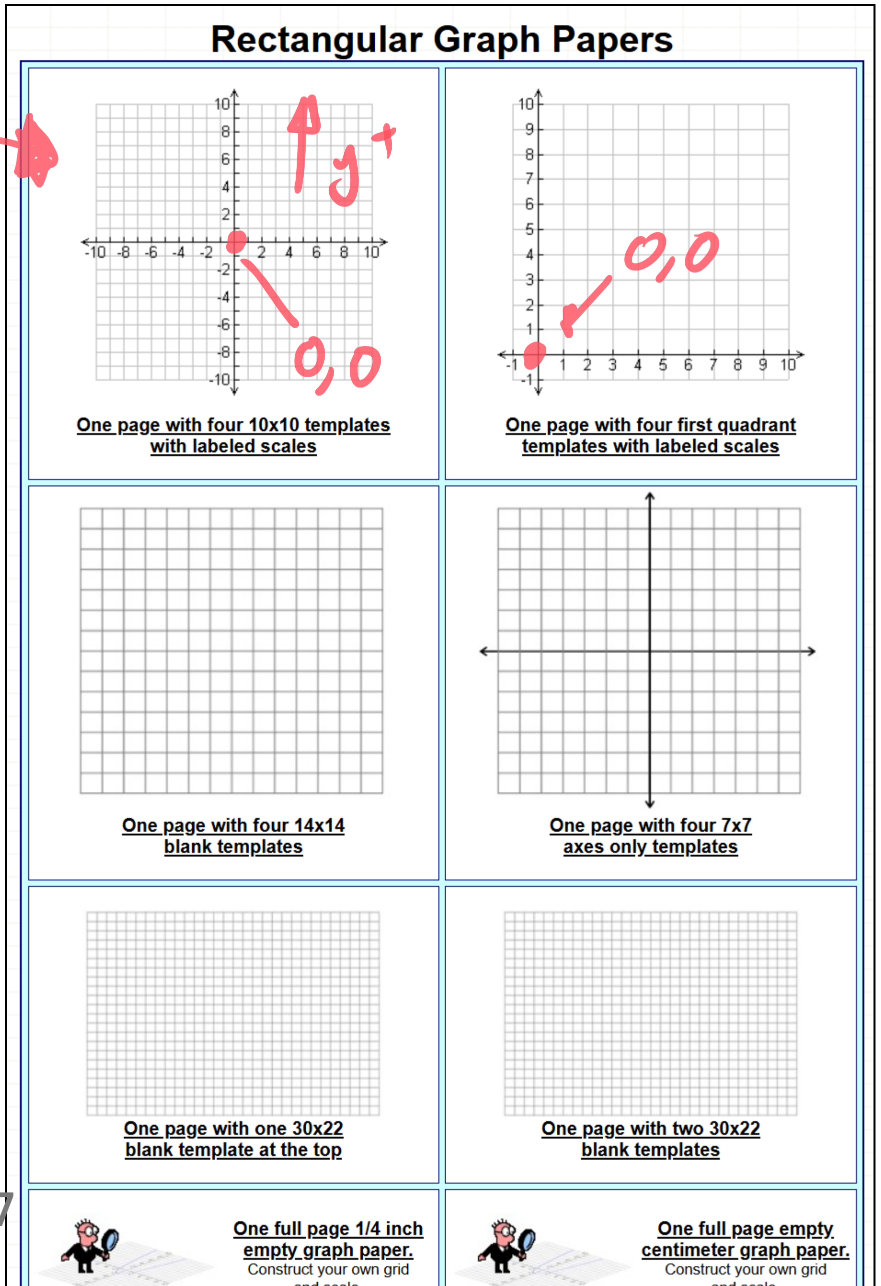
Other Coordinate Systems

- Canvas Coordinates
- Page (document coordinates)
- Window Coordinates
- Screen Coordinates
- And others…



Math Class Coordinates

- Y axis goes up
- Origin at Center
- Origin at Lower Left (1st Quadrant)



Rectangular Graph Papers

One page with four 10x10 templates with labeled scales

One page with four first quadrant templates with labeled scales

One page with four 14x14 blank templates

One page with four 7x7 axes only templates

One page with one 30x22 blank template at the top

One page with two 30x22 blank templates

One full page 1/4 inch empty graph paper. Construct your own grid

One full page empty centimeter graph paper. Construct your own grid

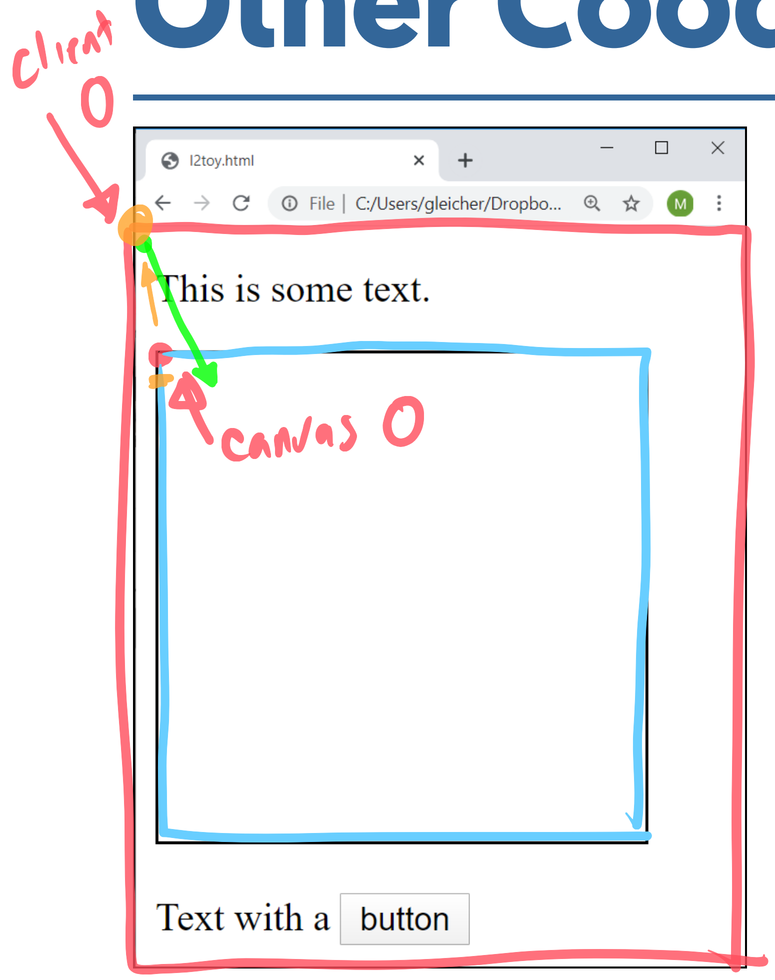
Handling Events

The **canvas** is the HTML element

The **canvas** receives events

- mouse enter / leave
- mouse move (inside)
- click

Other Coodinates?



Mouse position is in "client" coordinates

```
let box = event.target.getBoundingClientRect();
let x = event.clientX - box.left;
let y = event.clientY - box.top;
```

Need to convert from window to Canvas

It is **convenient** to draw in Canvas Coordinates

Where is the mouse?

```
let canvas = document.getElementById("myc");
let context = canvas.getContext("2d");

canvas.onmousemove = function(event) {
    let box = event.target.getBoundingClientRect();
    let x = event.clientX - box.left;
    let y = event.clientY - box.top;
    context.fillStyle = "#80800080";
    context.fillRect(x-5, y-5,10,10);
}
canvas.onclick = function() {
    context.clearRect(0,0,canvas.height,canvas.width);
}
```

Canvas "Events"

Only the “canvas” is an HTML element

Only the “canvas” gets events

The graphics are represented in code

There is no object to get an event

Immediate mode: primtives "immediately" turned to pixels

Click in a rectangle

```
canvas.fillRect(20,20, 60,60);

canvas.onclick = function(event) {
    let mouseX = getXposition(event);
    let mouseY = getYposition(event);
    // check if event is inside of rectangle
    if ( (x>=20) and (x<=(20+60) ) and (y>=20) and (y<=(20+60))) {
        console.log("rectangle was clicked")
    }
}
```

Warning: the event must be converted to canvas coordinates!

Remember the rectangle?

let

```
rects = [];

canvas.fillRect(20,20, 60,60);
rects.push( { x:20, y:20, w:60, h:60} );
```

In immediate mode, the shapes are in the code - not data structures.

If you want to remember them, you need to make your own data structures.

drawRect (obj)

canvas.fillrect (obj.x . . . .)

Coordinate System

You need to know how to interpret coordinates!

- Where is the origin?
- How do I interpret the X Axis?
- How do I interpret the Y Axis
- (in 3D, we will have a 3rd axis)

We'll come back to this

Changing Coordinate Systems

```
context.translate(x,y)
```

1. Move all future drawing points by x,y
2. Move the **coordinate system** by x,y

For translation, there isn't much difference

Immediate mode

Once something is drawn, we can't move it

`translate` moves **future** drawing commands

It is drawing state - just like the pen (save/restore works)

Demo

https://cs559.github.io/2DTransformDemos/

Some things to note

- we change the coordinate system
  for **future** drawing!
- translate in the current coordinate system
  translations **add up**
- need to "clean up" to get back to start
  save and **restore** are handy

Why is this a big deal

Coming Attractions

- Define groups of objects that go together
- Place groups appropriately
- Re-use groups

- Other types of changes to coordinates systems
  - rotate
  - scale
  - and other **transformations**

Hints for Fireworks (WB2)

- Read the page 6 examples (02-06-05b.js)

- Keep a list of objects
    - store position, velocity, color, ...

- Events
    - Mouse click - create objects
    - Animation loop - move objects

Summary

- Buffer to help with timing
- Use rules for complex polygons
- Events for Canvas, not Primitives
- Coordinate Systems