Lecture 3
Graphics 101

Why are things the way they are?

Review

How are we going to learn graphics?

- Class Organization
- Web Programming Basics

Today

Some basic background (Graphics 101)

Some web graphics basics (for the workbook)

More web graphics stuff Thursday (for the workbook)

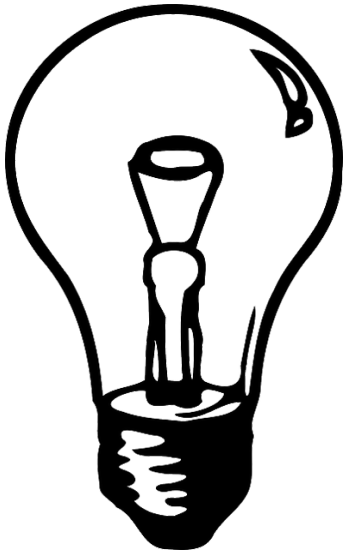
Computer graphics (the field) is the study of

How computers create things we see

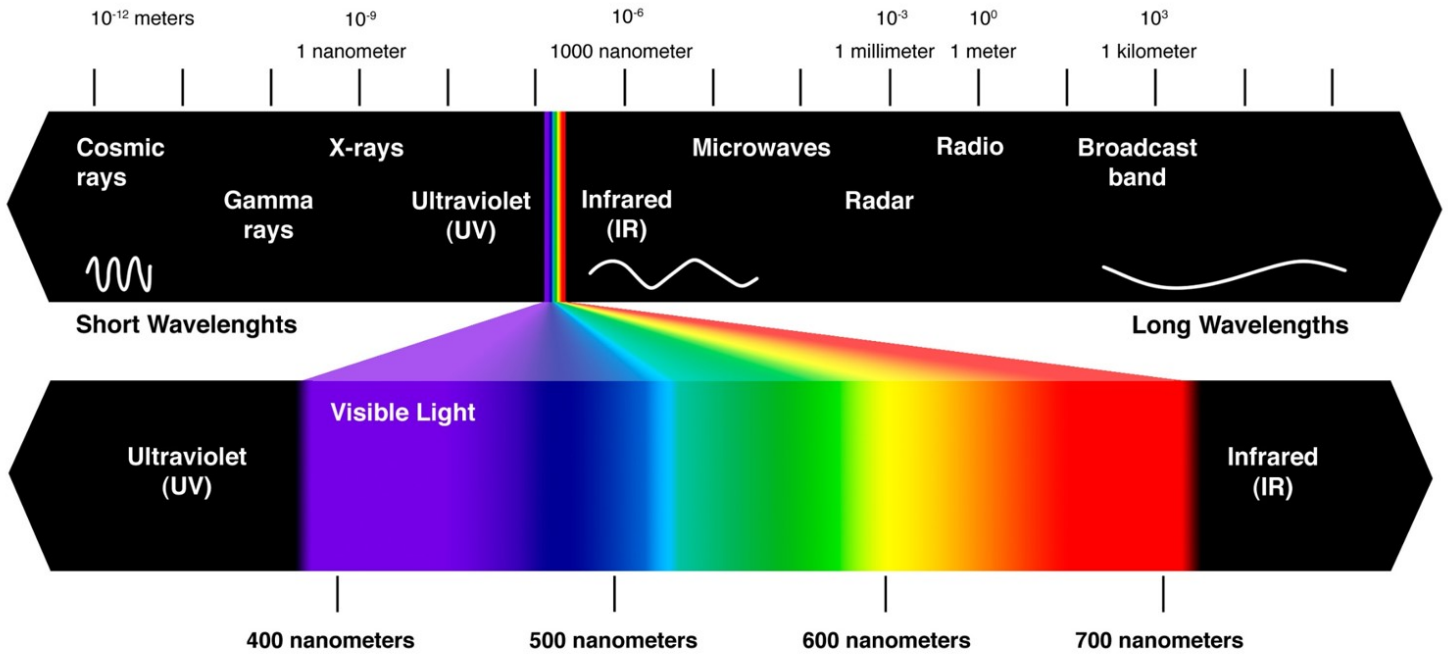
How do we see?

How do we see?

How do we see? (What do we see?)

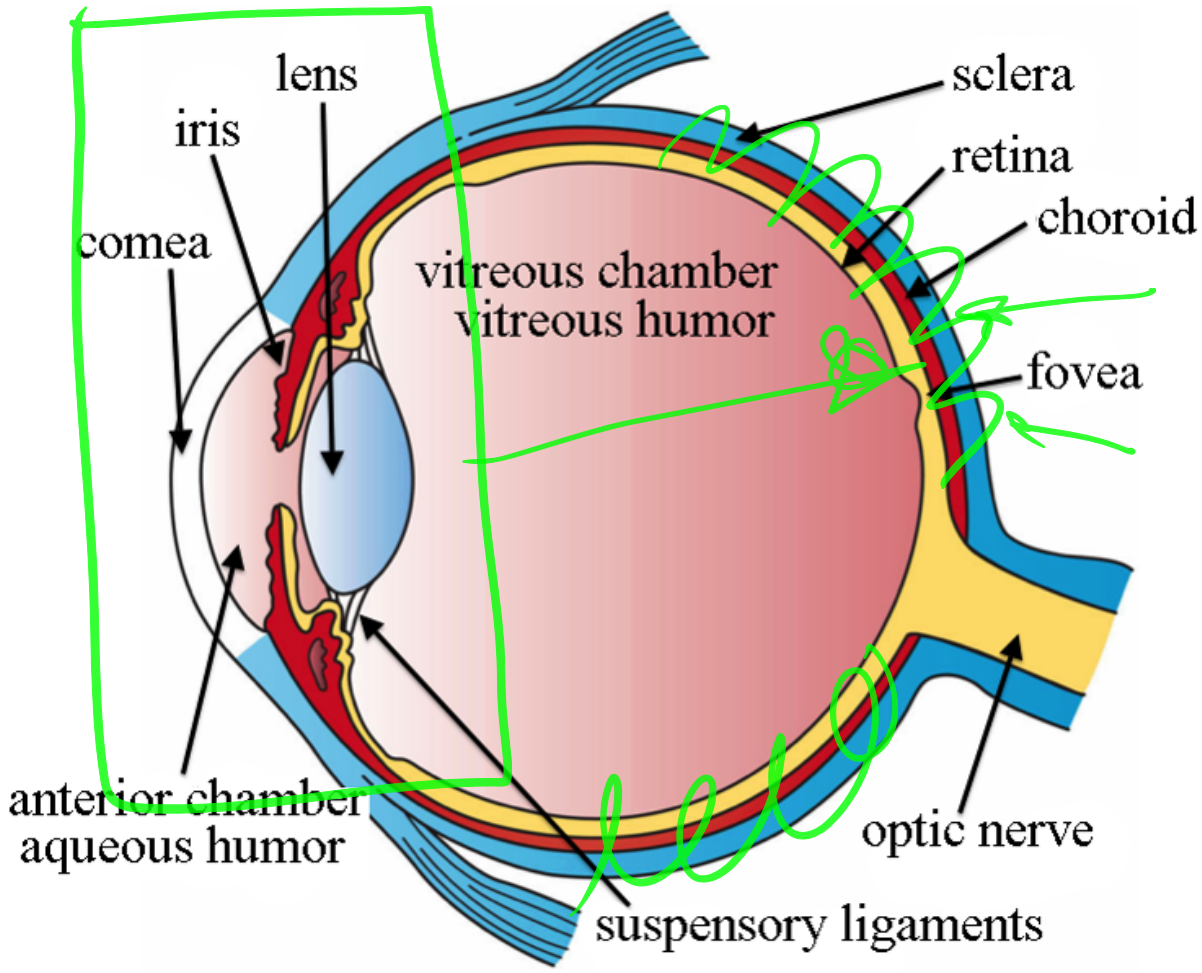


A little about light

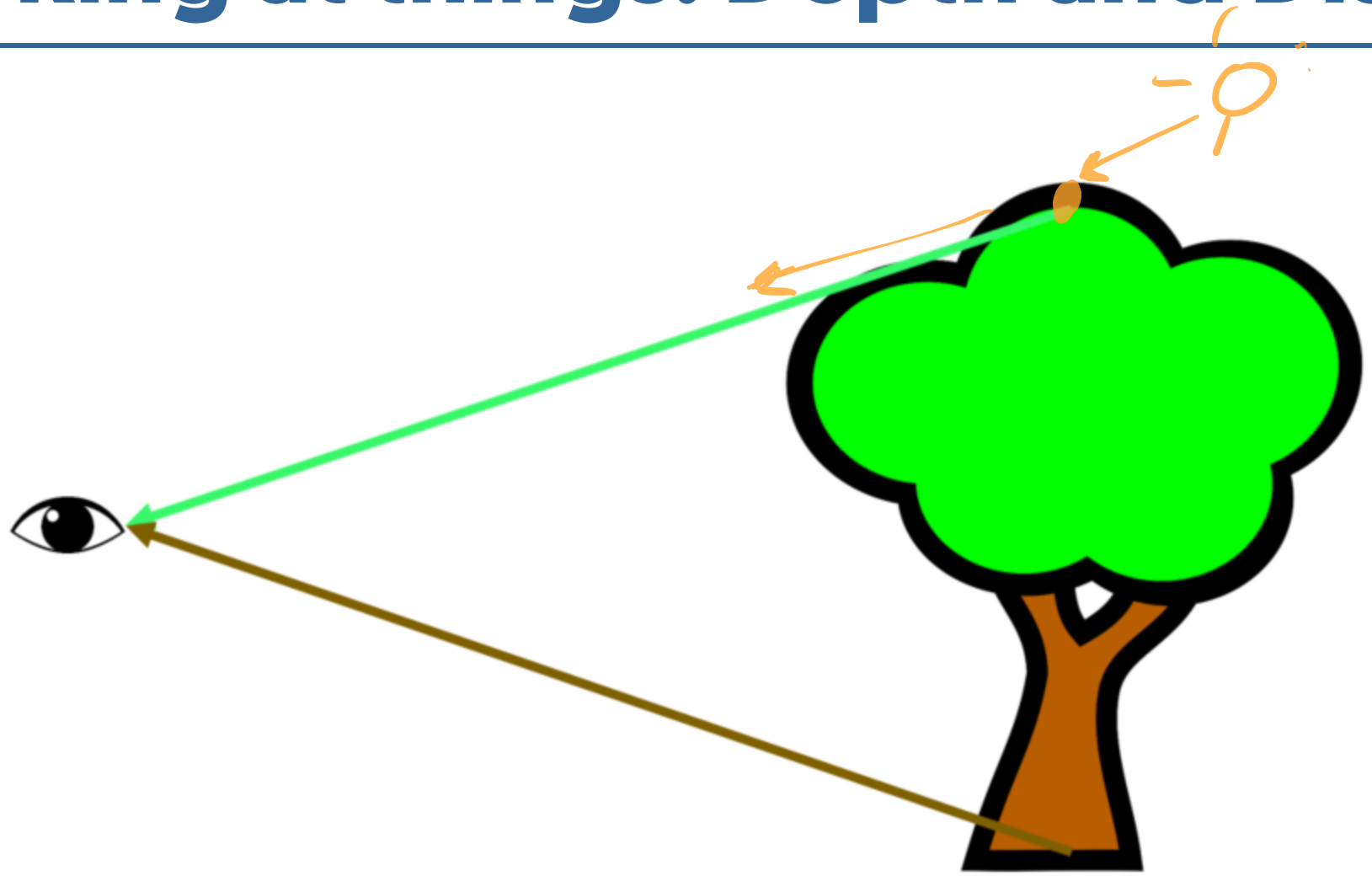


- travels in straight lines
- hits things
  - absorbed
  - bounces
- has color (wavelengths)
  - Why 3 numbers?

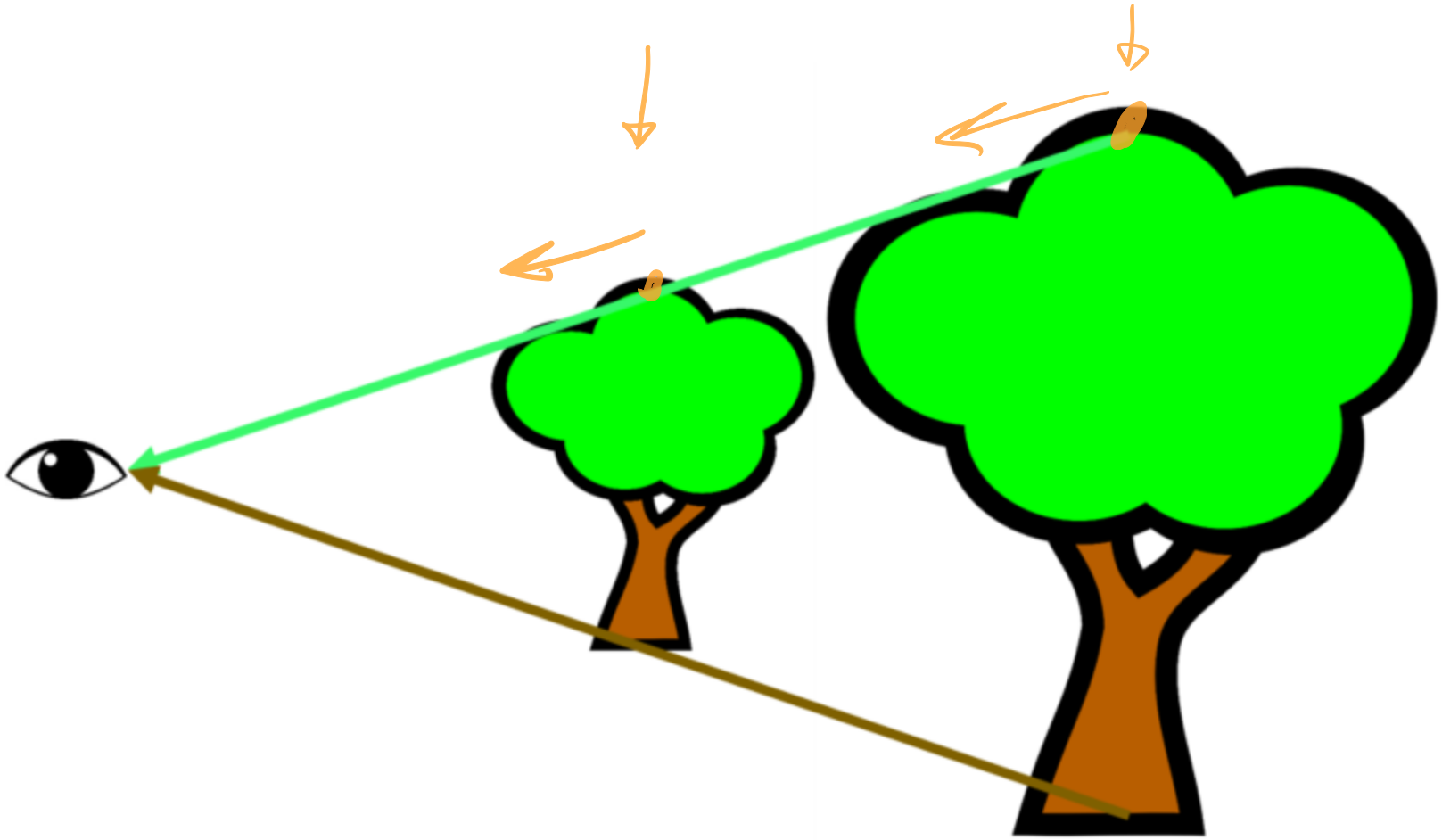
Where (some) light ends up



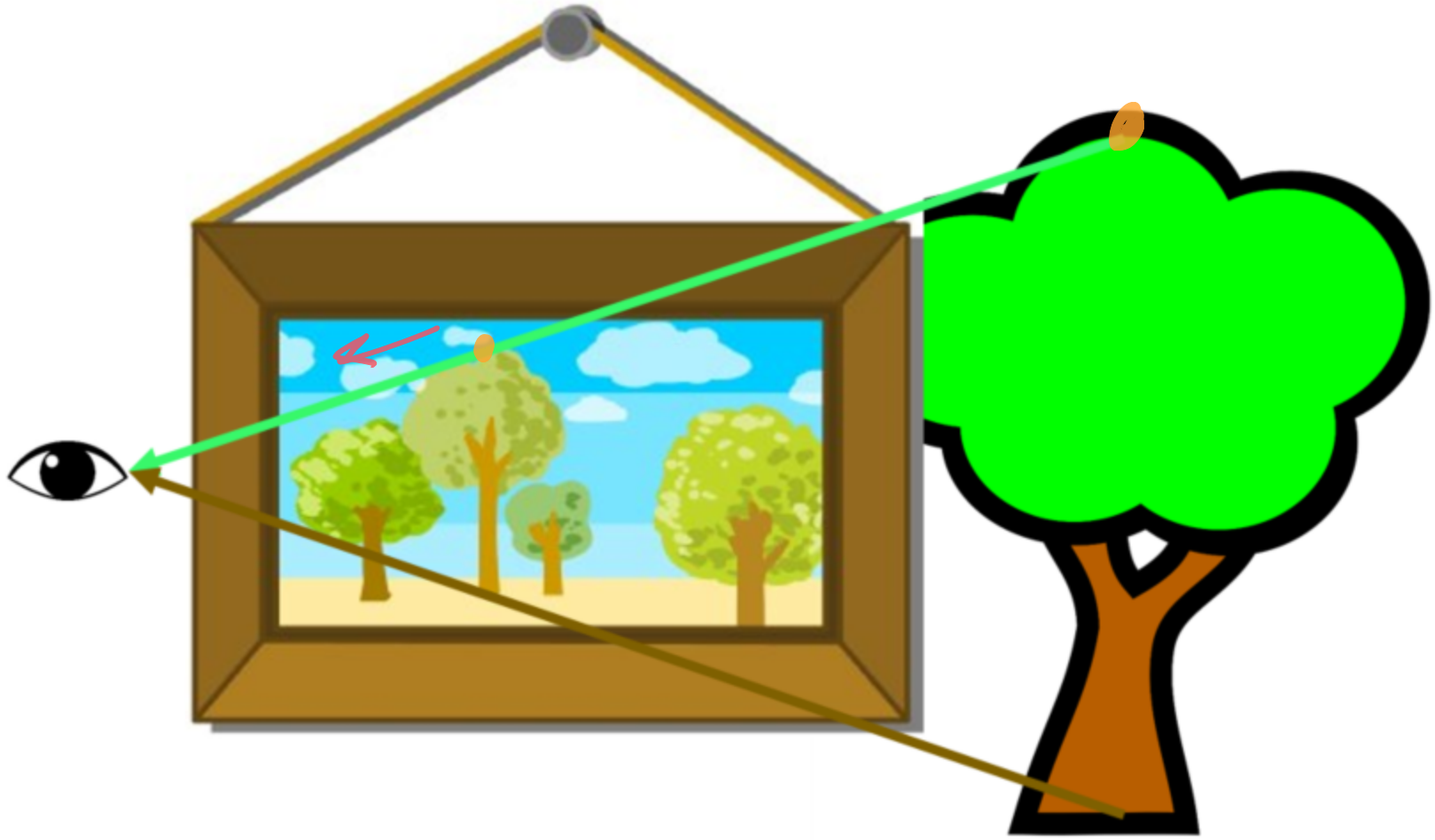
Looking at things: Depth and Distance



Looking at things: Depth and Distance



Looking at things: Depth and Distance



Can a Picture Fake Us Out?


https://www.turneadv.it/wp-content/uploads/2017/10/3D-pedestrian-crossing-island-2-59f03455342f2__880.jpg
GUSTI


Horn


Hótel
SUZUKI


PURE

The artist is Julien Beever - you can look him up on the web

We sense 2D

(actually, a little more than that)

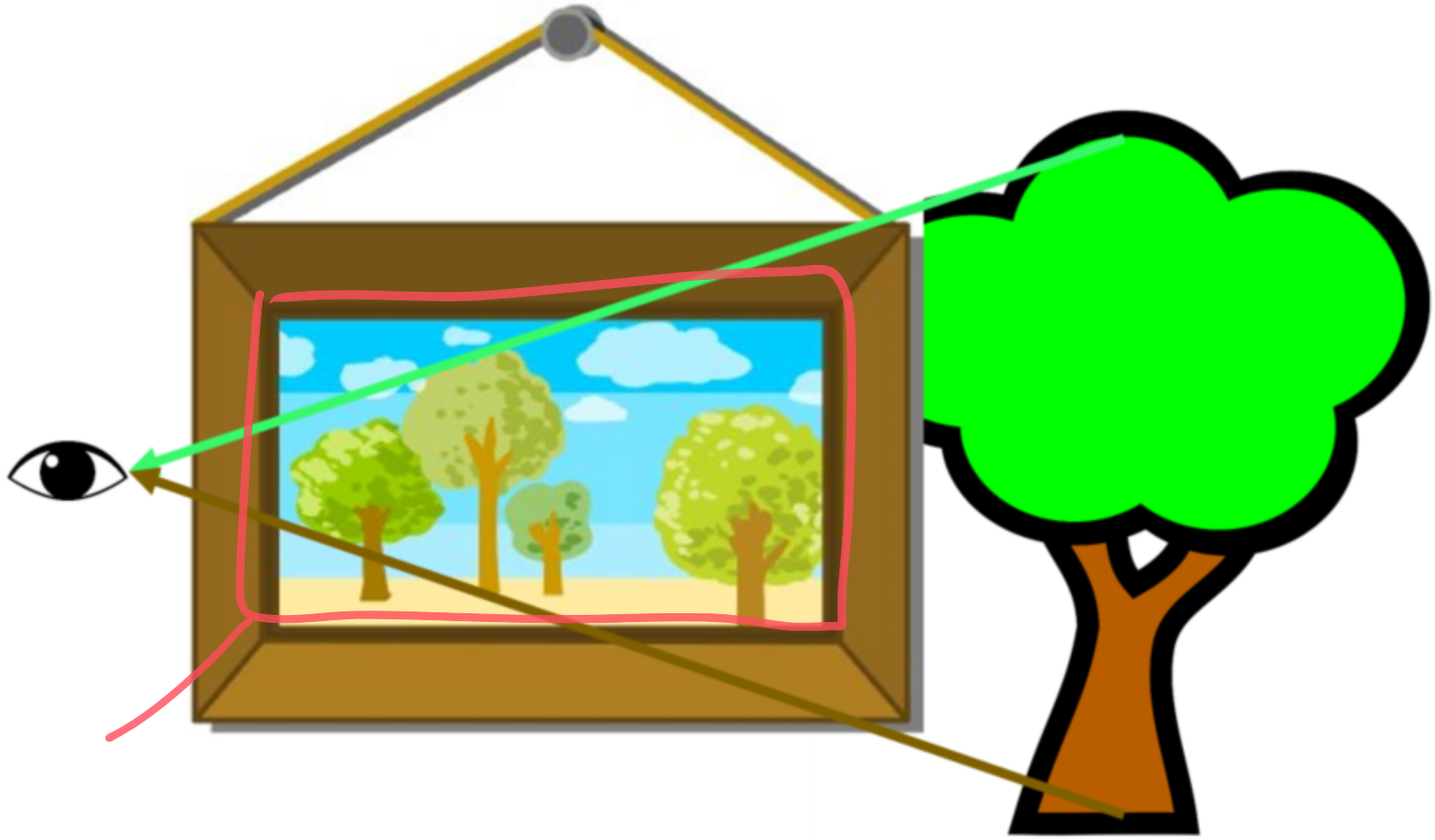
There are other cues...

but single image cues are very strong!

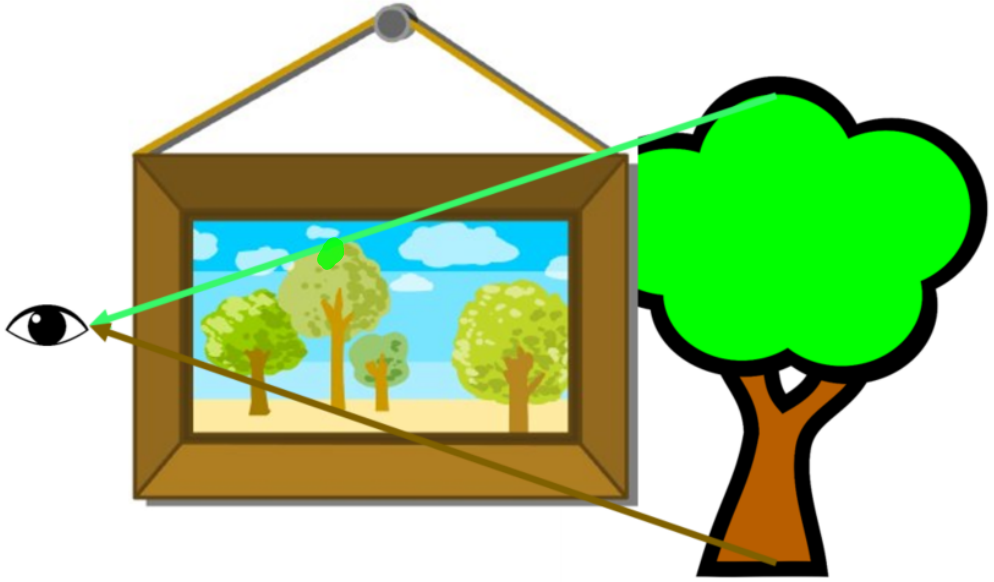
More on this later in the course

We infer 3D

Images



Creating Images



- simulate **photons**
- simulate **painting**
- just draw in 2D

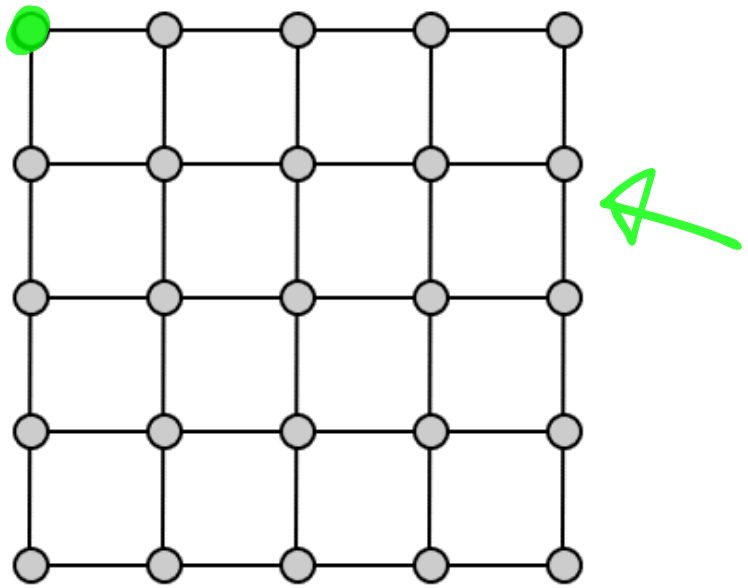
Physically-Based

vs.

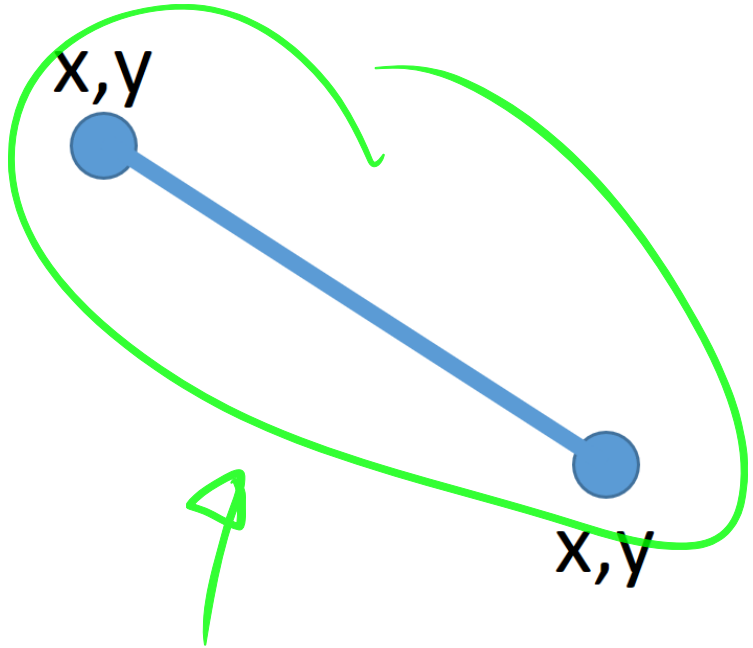
Primitive-Based

Representing Images

Sampled (Raster)



Geometric (Primitives)



Displays

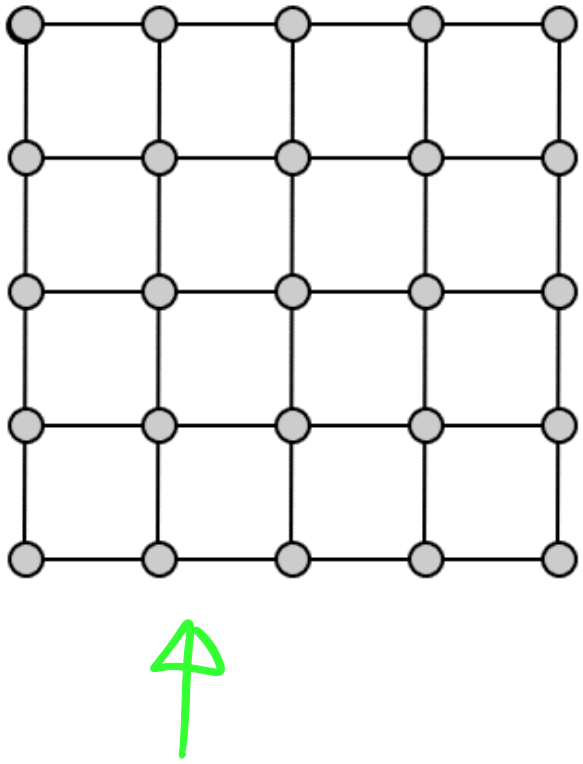
How we **show** images

Sometimes the output is 3D (e.g. a 3D printer)

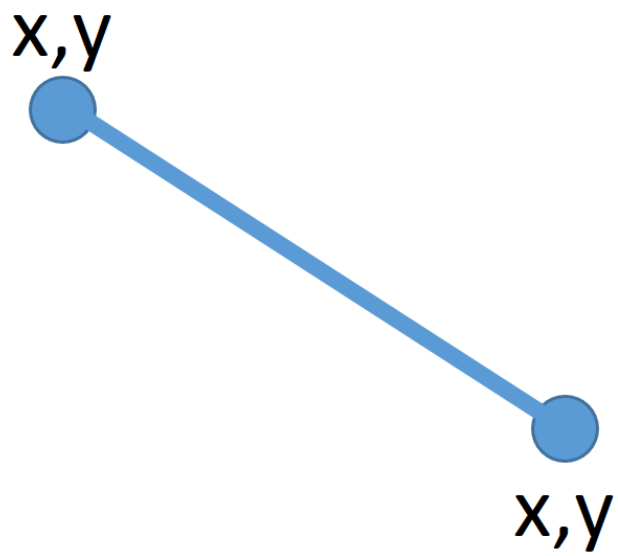
- we need to represent **shapes**
- similar problem to making pictures

Types of Displays

Sampled (Raster)



Geometric (Primitives)



Examples of Displays

Sampled (Raster)

- LCD/LED/CRT
- Laser printer, inkjet printer, …
- 3D printer (most)
- Projectors
- Film (irregular grid of crystals)

(just about anything you encounter)

Geometric (Primitives)

- Pen plotters
- Laser light shows
- Old fashioned vector displays

(nothing that is common today)

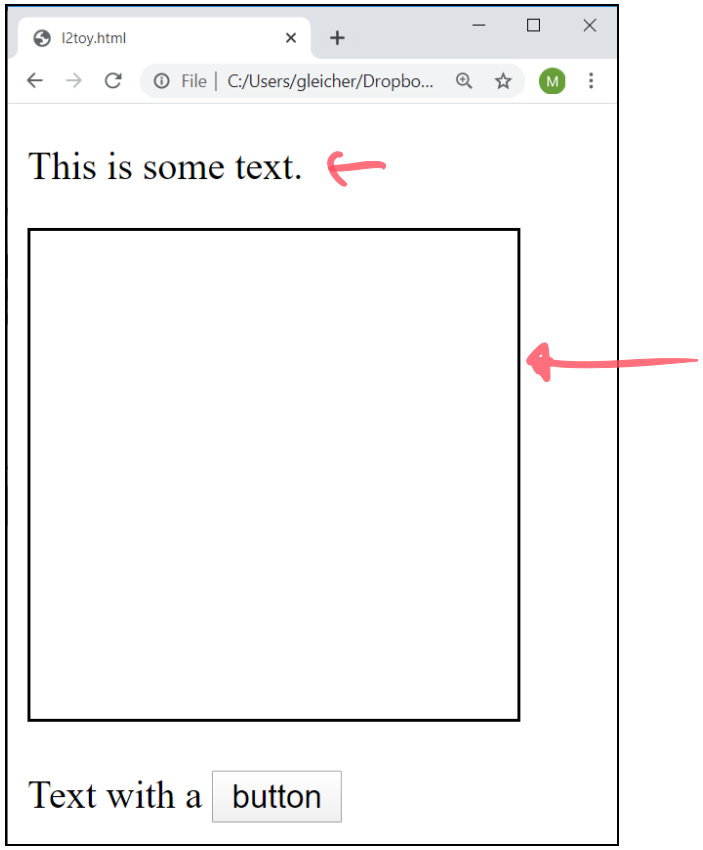
Rasterization:
Convert primitives to Pixels

We'll let the APIs/libraries/hardware take care of it (for most of the class)

We are going to work with **primitives**

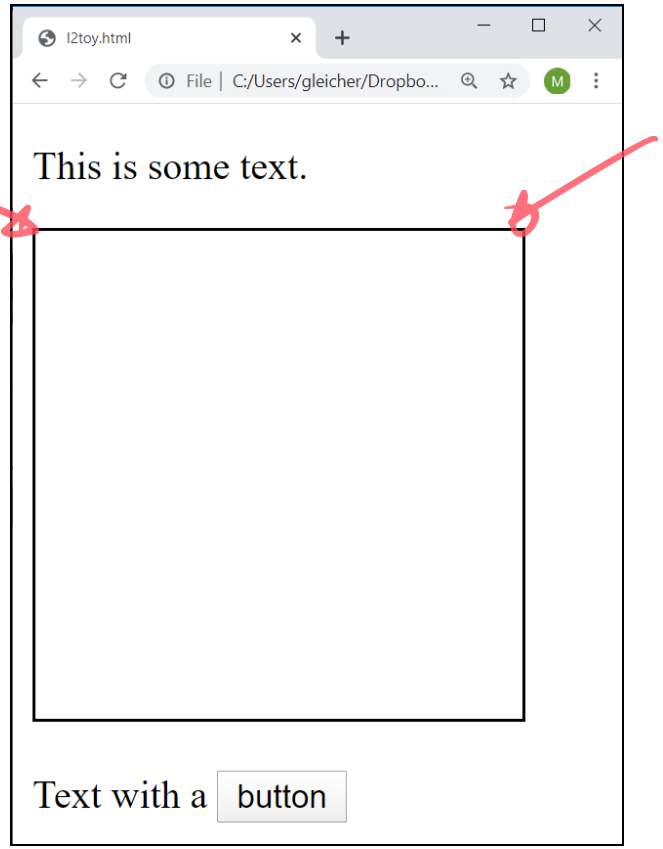
How do we draw primitives?
On a web page

We can make web pages



Now, Let's use this for **Graphics!**

How can we put stuff in this box*?



Web Browser Graphics APIs

- Canvas (HTML5 2D Canvas API)
- SVG (scalable vector graphics)
- WebGL (technically, a Canvas)
- libaries on top of these
  - THREE.JS (a layer over WebGL)

*The "Box" can be the whole window/screen

Web Graphics APIs (built in)

Canvas 2D

- an *immediate mode* 2D drawing library

SVG (Scalable Vector Graphics)

- a display-list (object based) graphics library / file format
- graphics objects are DOM elements

WebGL (a JavaScript version of OpenGL ES)

- direct access to the graphics hardware
- **requires** low-level control - you must program the hardware

Often we will use layers on these

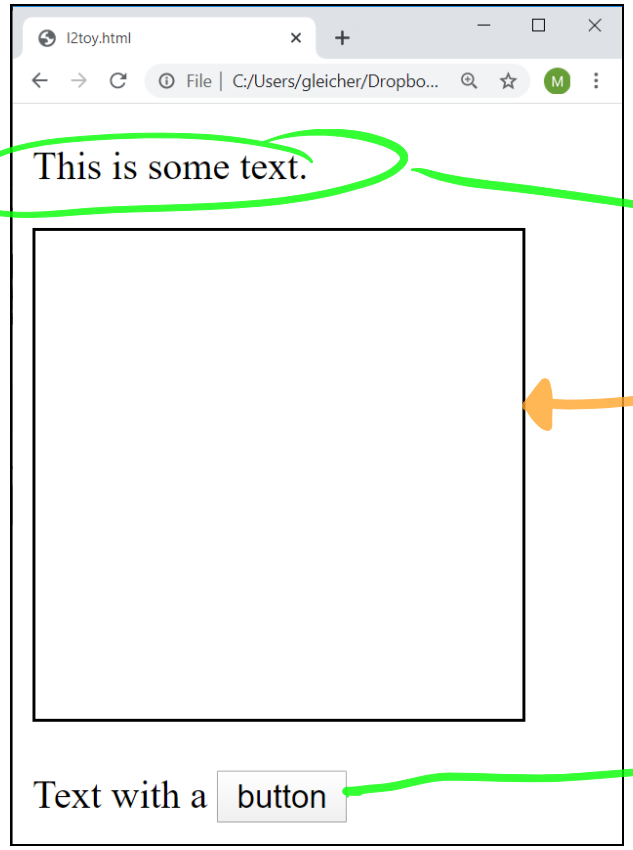
Three.js (or just Three)

- A display list API built on top of WebGL
- Takes care of details for you

D3 (not used in class)

- A tool that makes it easy to manipulate DOM elements
- Very useful for SVG, especially for doing visualization

Web page with a Canvas element




```
<!DOCTYPE html>
<html>
<body>
    <p>This is some text.</p>
    <canvas id="myc" width="200px" height="200px"
            style="border:1px solid black">
    </canvas>
    <p>Text with a <button>button</button></p>
</body>
</html>
```

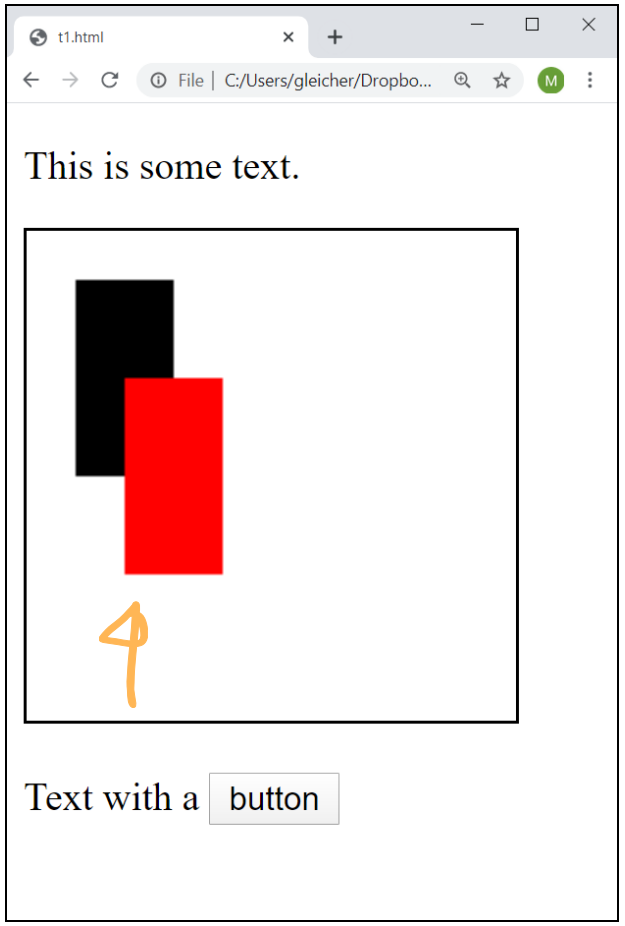
Canvas vs. SVG

Workbook mentions both:

- Canvas = immediate mode API
- SVG = retained mode API

We will mainly use Canvas - and look at SVG more later in the class.

Web page with a Canvas element




```
<!DOCTYPE html>
<html><body>
    <p>This is some text.</p>
    <canvas id="myc" width="200px" height="200px"
            style="border:1px solid black">
    </canvas>
    <p>Text with a <button>button</button></p>
</body>
<script>
    let canvas = document.getElementById("myc");
    let context = canvas.getContext("2d");

    context.clearRect(0,0, canvas.width, canvas.height);

    context.fillRect(20,20, 40, 80);

    context.fillStyle = "red";
    context.fillRect (40,60,40,80);
</script></html>
```

Immediate vs. Retained APIs

The workbook discusses this

Today, we focus on canvas which isn an **immediate** API

When we draw a primitive (rectangle)

- it "immediately" gets "converted"
- we have no access to the rectangle after the command
    - we have to keep track of it!
- it may not appear immediately (buffering)
- it may stay around (e.g., on the screen)

Things to notice about Canvas

Canvas is the **element**

Context is the **API**

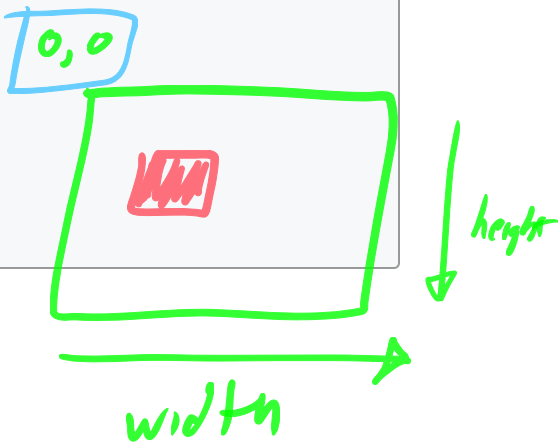
Need to clear frame

Coordinate System

Measurement Units

Stateful Drawing

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

context.clearRect(0,0, canvas.width, canvas.height);

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



Three Questions...

When do I draw?

when it's your turn!

Where do I draw?

In the Canvas coordinate system

What do I draw?

Primitives!

When do I draw?

Once

when the page Loads

Over and Over

in an animation loop

When an event happens

that causes us to need to change the picture

Not too Fast!

Make sure the viewer sees it

requestAnimationFrame

Drawing and Redrawing

General assumptions:

- it's empty (background color) before we start
- no one else cares to draw in our canvas (but they could)

We can:

- Add to the existing drawing
- Draw a rectangle to "erase" a region (draw background color)
- Erase the whole thing and redraw

We **cannot remove an object** (immediate mode) - just draw over it

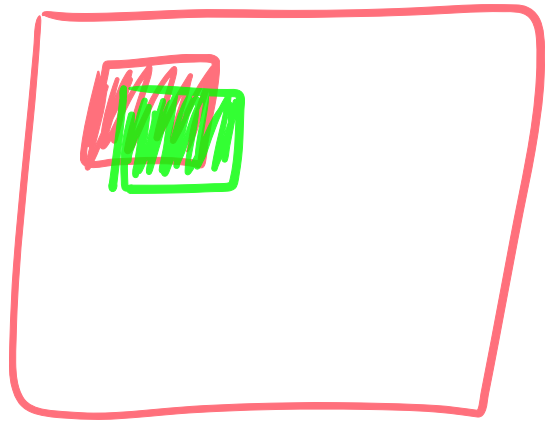
About Redrawing (Animation)

To make things appear to move:

- clear the screen
- draw something
- wait until the viewer has seen it

- clear the screen (?)
- draw something (slightly different)
- wait until the viewer has seen it

(and so on...)



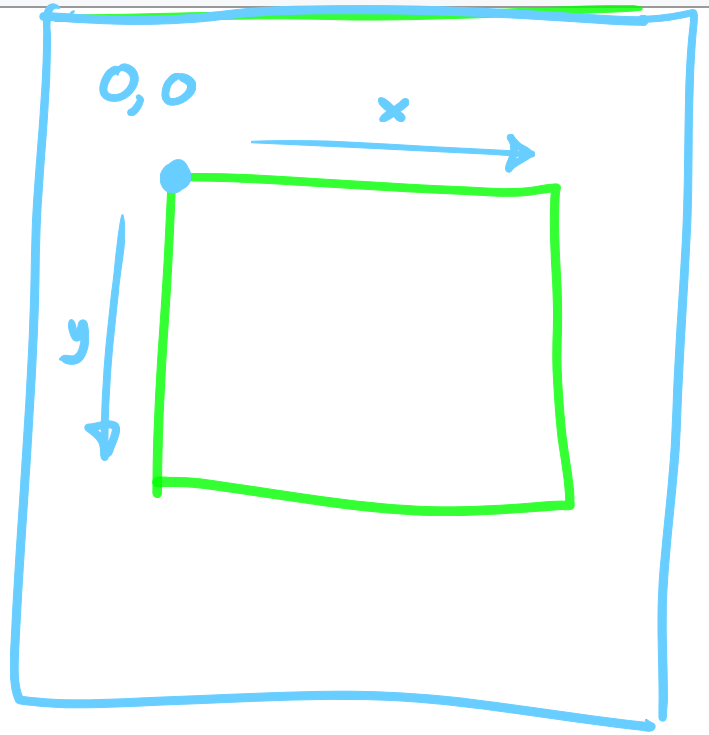
Where do I draw?

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

```
context.fillRect(40,60,80,50);
```

Canvas coordinates:

- origin at top left
- x to the right in "html pixels"
- y down in "html pixels"



Canvas Coordinates

```
<canvas width="400px" "height=200px"></canvas>
```

(0,0) is top left

`canvas.width,canvas.height` is bottom right

When we **draw**, this is relative to the **canvas**

What do I draw: Rectangles

```
context.fillStyle = "red";
context.fillRect (40,60,40,80);
```

Rectangles are: x,y,w,h - w,h are sizes, not positions

Separate commands for **stroke** and **fill**

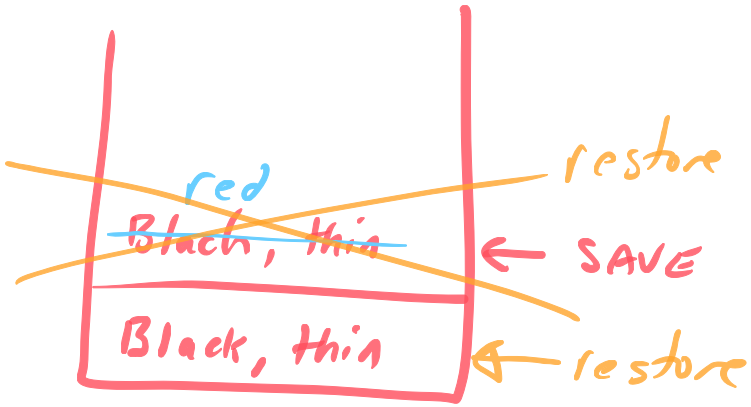
Styles as **state**

- fill color, stroke color
- fill patterns
- stroke patterns (dashed), line width, ...

Save and Restore

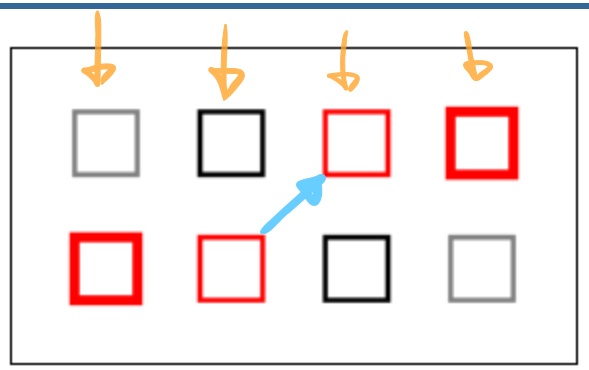
```
context.save();
context.fillStyle="red";
context.fillRect(40,40,20,20);
context.restore();
context.fillRect(50,50,20,20);
```

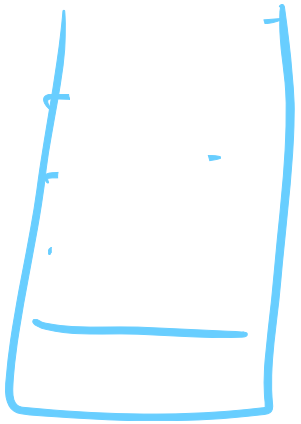
`save` and `restore` capture most (all?) context information



Save and restore is a stack

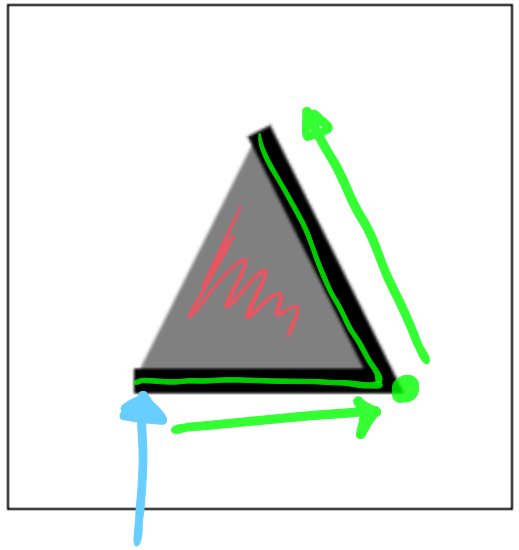
```
context.strokeStyle = "black";
context.strokeRect(25,25,25,25);
context.save();
context.lineWidth = 2;
context.strokeRect(75,25,25,25);
context.save();
context.strokeStyle = "red";
context.strokeRect(125,25,25,25);
context.save();
context.lineWidth = 4;
context.strokeRect(175,25,25,25);


context.strokeRect(25,75,25,25);
context.restore();
context.strokeRect(75,75,25,25);
context.restore();
context.strokeRect(125,75,25,25);
context.restore();
context.strokeRect(175,75,25,25);
```

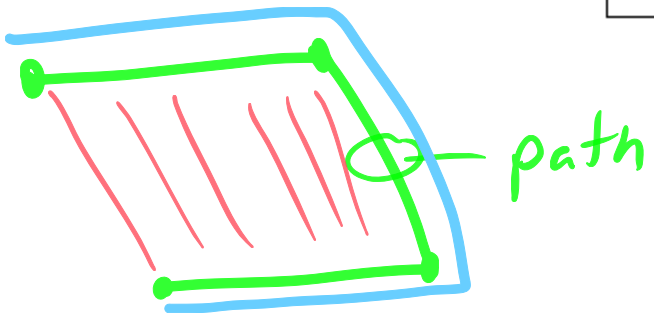




Beyond Rectangles: Paths

```
context.beginPath();
context.moveTo(x,y);
context.lineTo(x2,y2);
context.lineTo(x3,y3);
context.fill();
context.stroke();
```

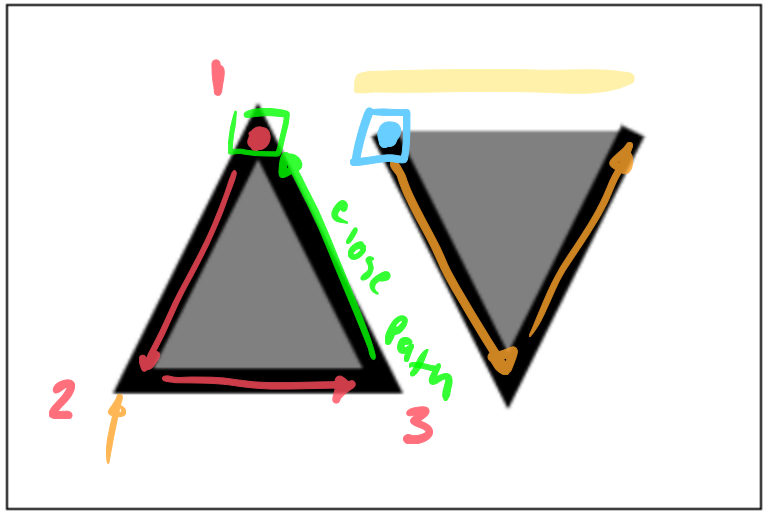




- beginPath
- moveTo
- lineTo
- fill
- stroke

Open, Closed, Disconnected ...

```
context.beginPath();
context.moveTo(100,100);
context.lineTo(110,120);
context.lineTo(120,100);
context.closePath();
context.moveTo(150,100);
context.lineTo(160,120);
context.lineTo(170,100);
context.fill();
context.stroke();
```



numbers are not precise
(triangles are upside-down)

Filling "complex" paths can be tricky...

We'll talk about the rules next time

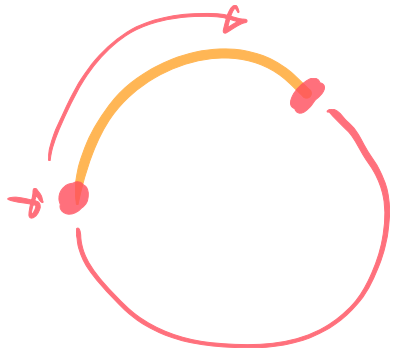
The Pen Model

Methods use the **current pen position**

Methods add to the **current path**

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

- `arc`, `arcTo`, `curveTo`, …

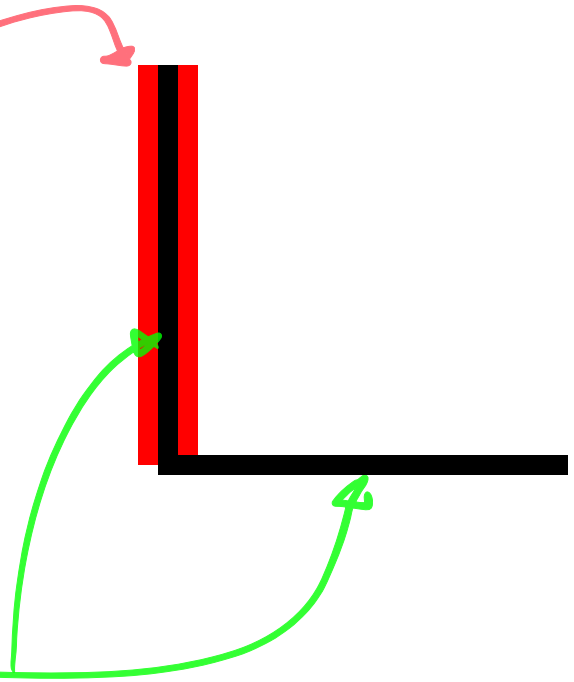


moveTo (x,y)
lineTo (x,y)
absolute
relative

Stroke/Fill the entire path!

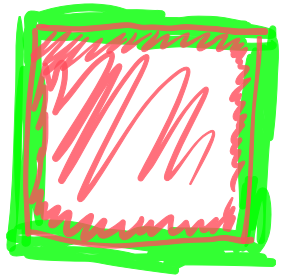
The entire path is redrawn with the current pen!

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

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



And there's more...

But this is enough to make pictures! (go make some!)



Key concepts for today

- importance of 2D
- primitives (geometric vs. raster graphics)
- immediate mode (vs. retained mode)
- basic drawing with canvas (when, where, what)