Lecture 14:
Lights!
Camera!
Action!

(not in that order)

Last Time...

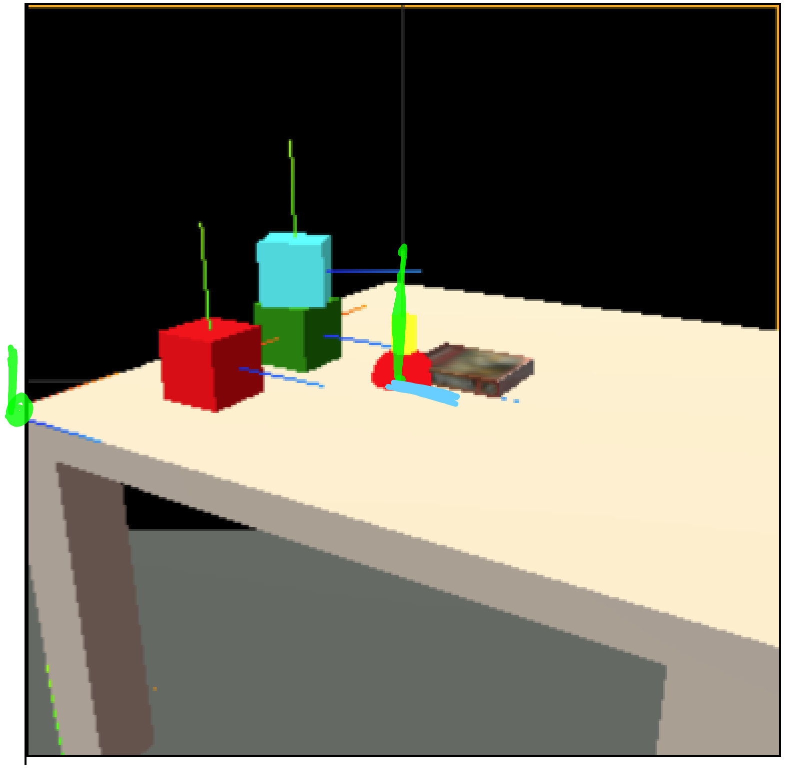
A 3D World

A Three World

Transformations and Hierarchy in THREE

Making a Scene

```
scene.add(new Table());
table.translateY(3);

let book = new Book();
table.add(book);
book.translate(2,0,2);

// I have function that makes cubes
let c2 = cube("green");
c2.translate(2,.25,1);
table.add(c2);
let c3 = cube("cyan");
c3.translate(0,1,0);
c3.rotateY(.5);
c2.add(c3);
```



State vs. Transformation

In **THREE.JS** (not all APIs have this)

```
cube.position.x = 5;
```

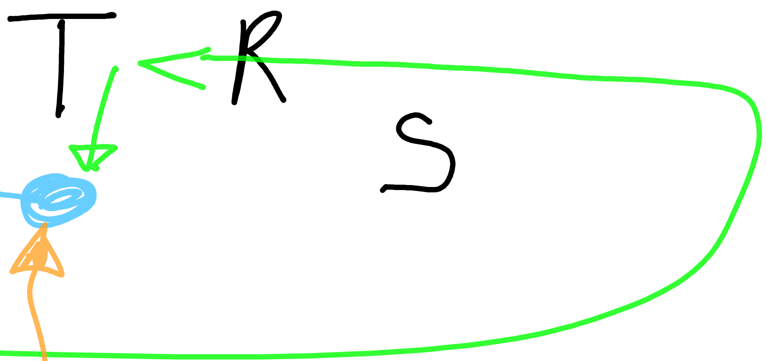
vs.

```
cube.translateX(5);
```

motion in coordinate system

vs.

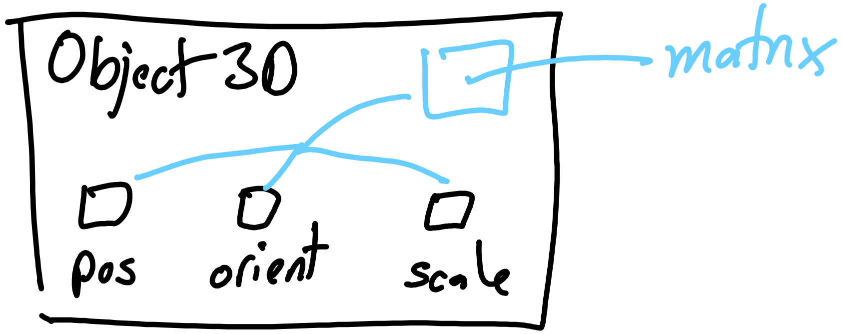
```
cube.position.x += 5;
```



How THREE works inside

Store state in "factored form" (Trans Rot Scale)

Move transformations through existing transformations



Scale

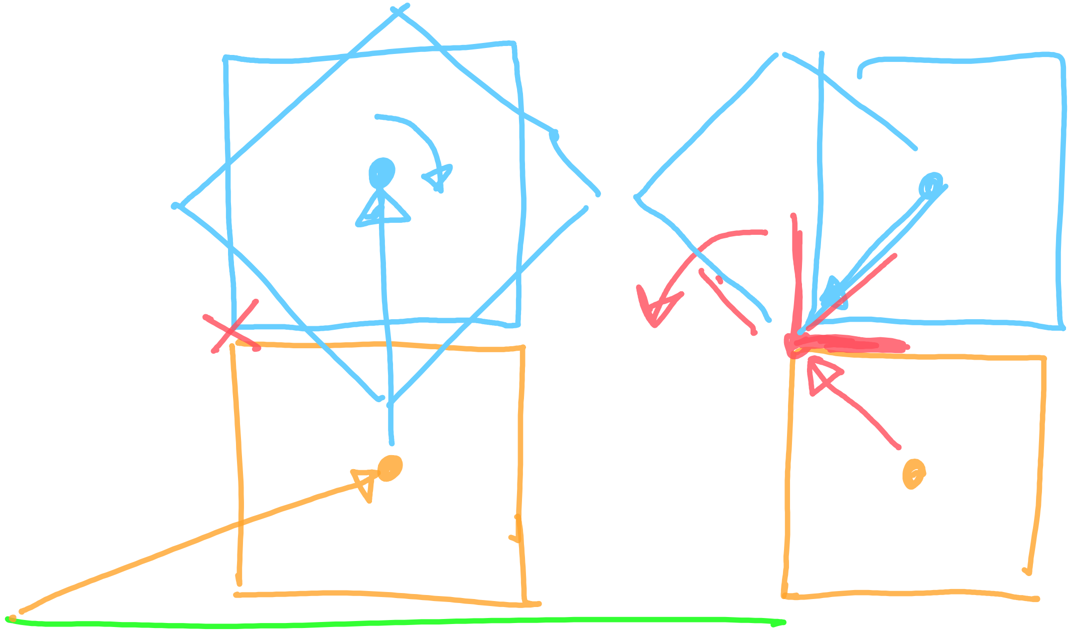
Three.js handles scale in a special way

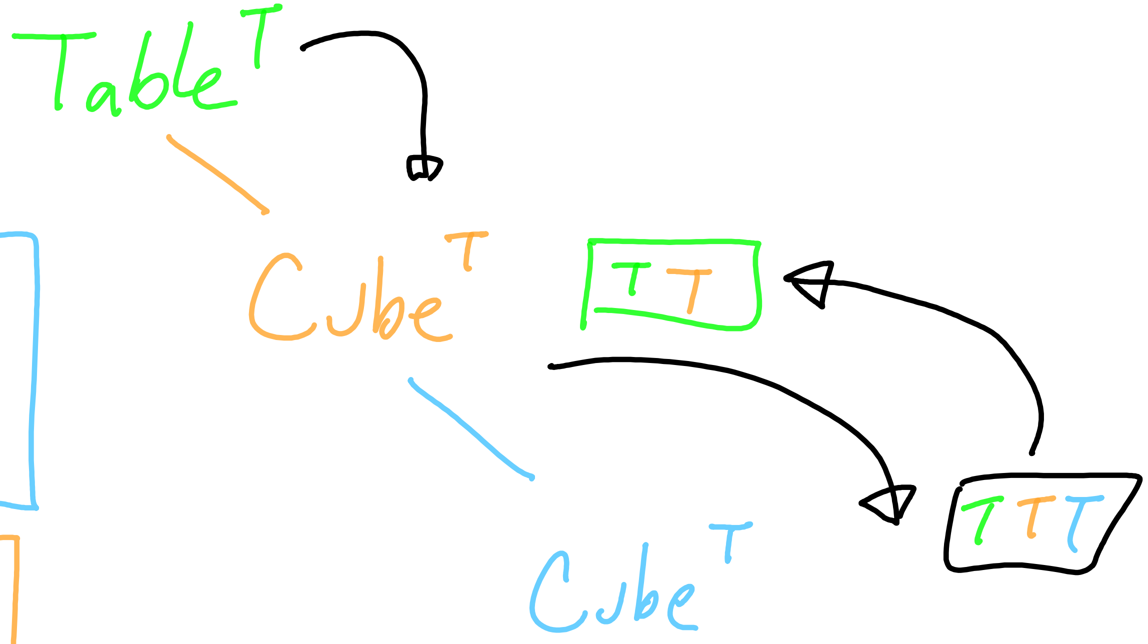
Confusing for graphics students

Convenient much of the time

Center of Rotation Example

Use groups and hierarchy!





What's Next

Workbook 6: Try out THREE

Workbook 7: do more with transformation and hierarchy

- Understand cameras and viewing
- Basics of lighting and shading
- Animation in THREE

- Rotations (which are tricky in 3D)
- More details of shape and lighting
- Texture

The Camera

The Viewing Transformation

From world (scene) coordinates to screen - via the camera

- the camera is in the world/scene
- we see things relative to the camera

Two parts to what we see:

1. Positioning the camera
2. Projecting from 3D to 2D

Positioning the Camera

1. It's a rigid Body (translate rotate)
2. Describe by what we see

(and there's the lens "zoom" - more on that in a bit)

Describing Cameras (or anything)

Position "eye point"

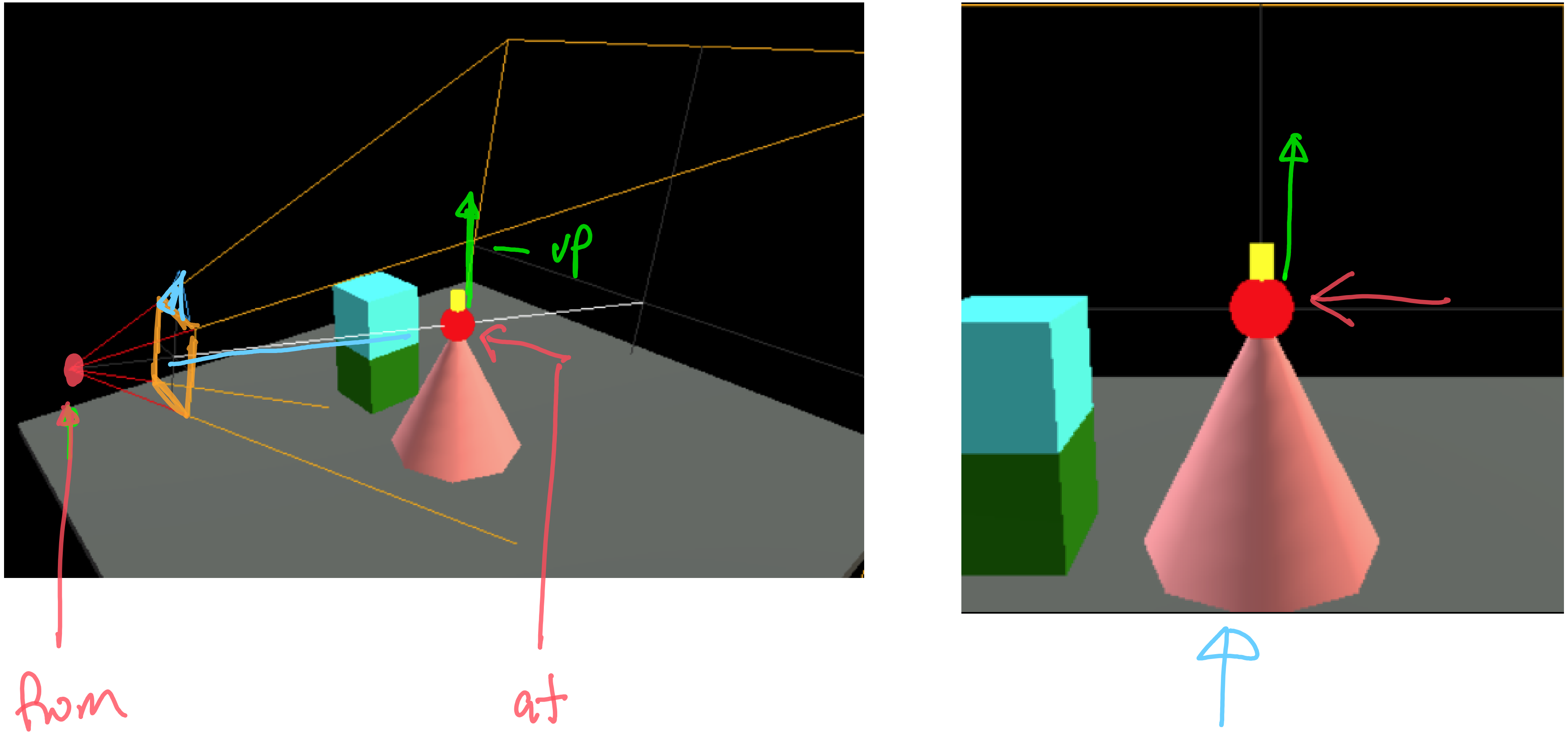
Rotate to "look at" something

- LookFrom (where to put the eye)
- LookAt (point the camera towards a point)
- Up (extra degree of freedom)

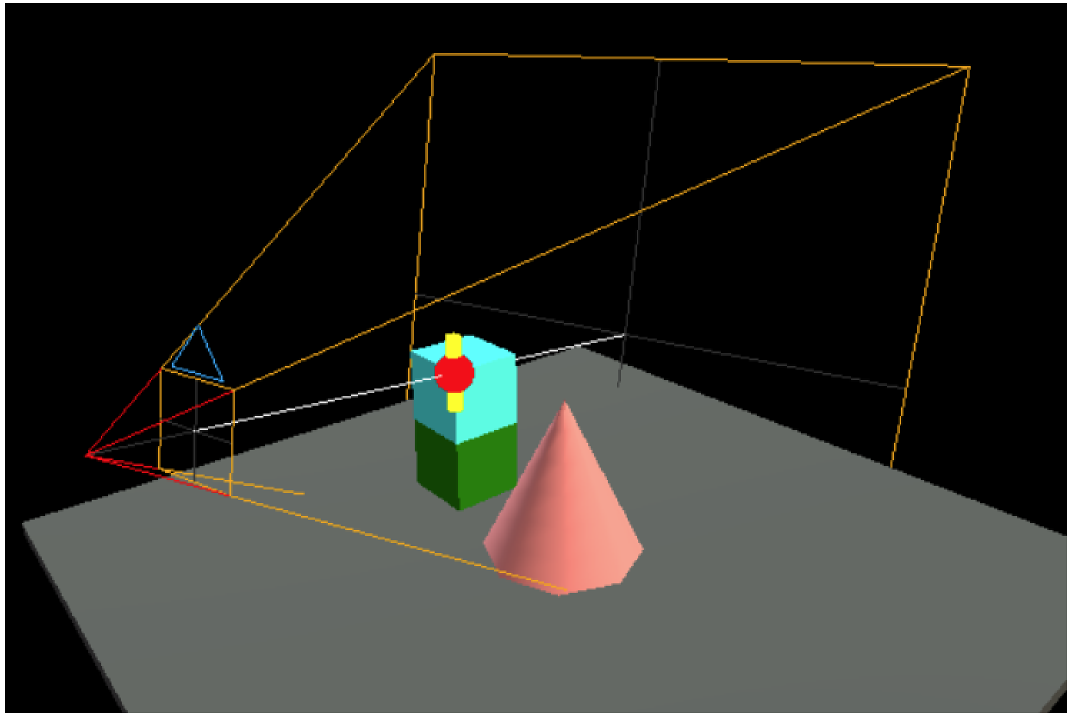
Lookfrom/Lookat/VUp

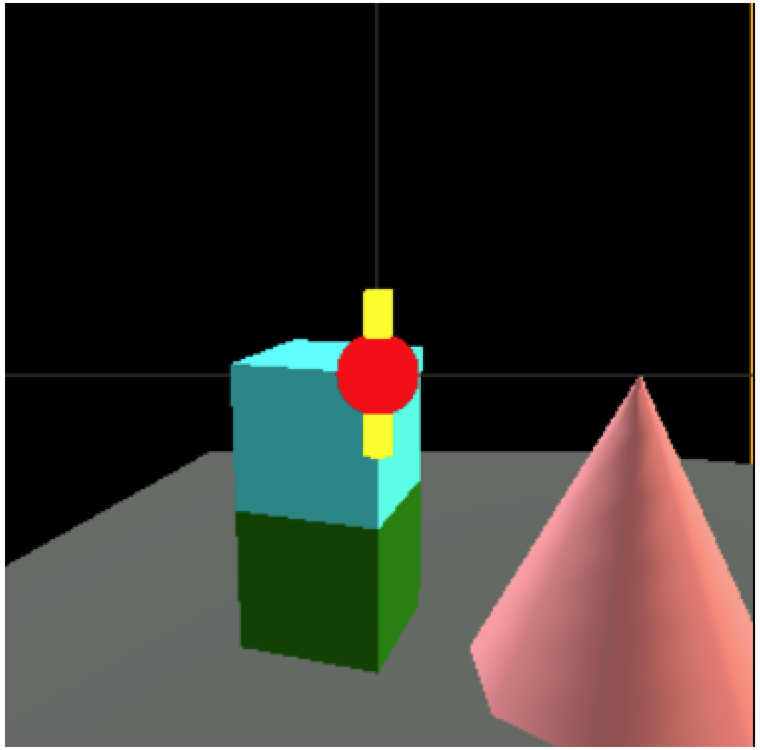
- implementing this is interesting (but not for today because...)

From the Demo

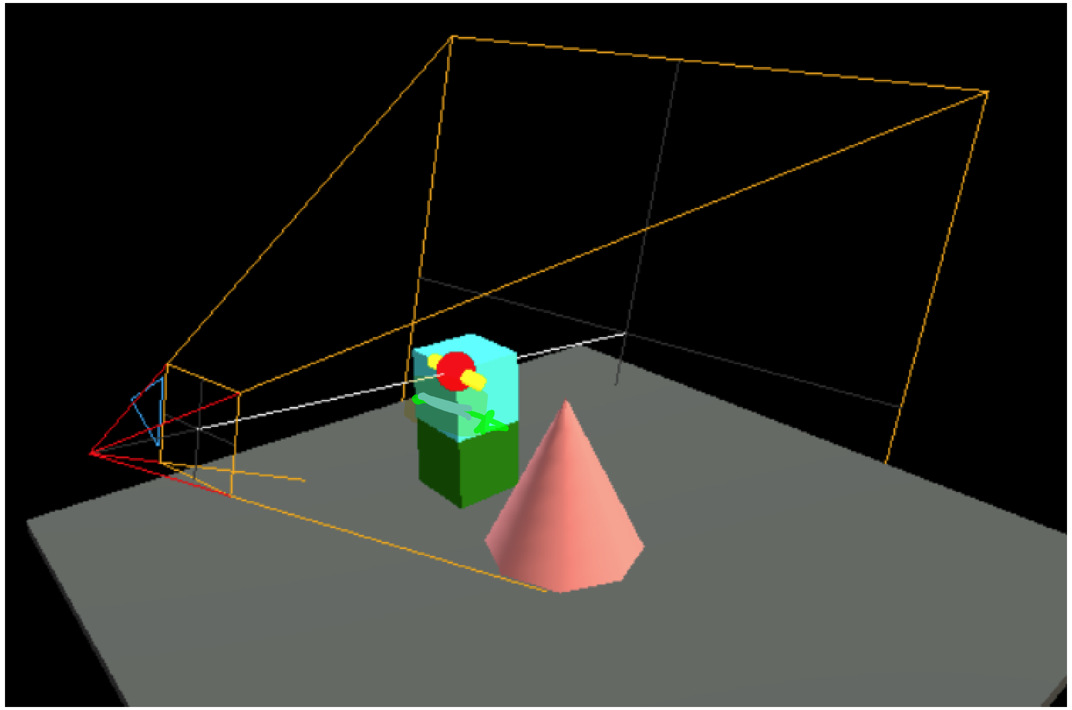


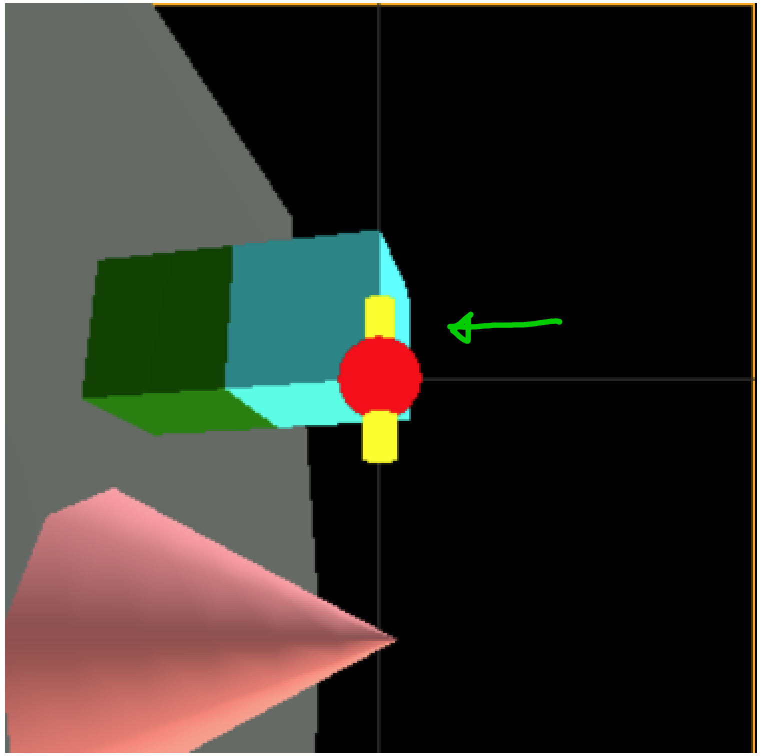
From the Demo (change LookAt)



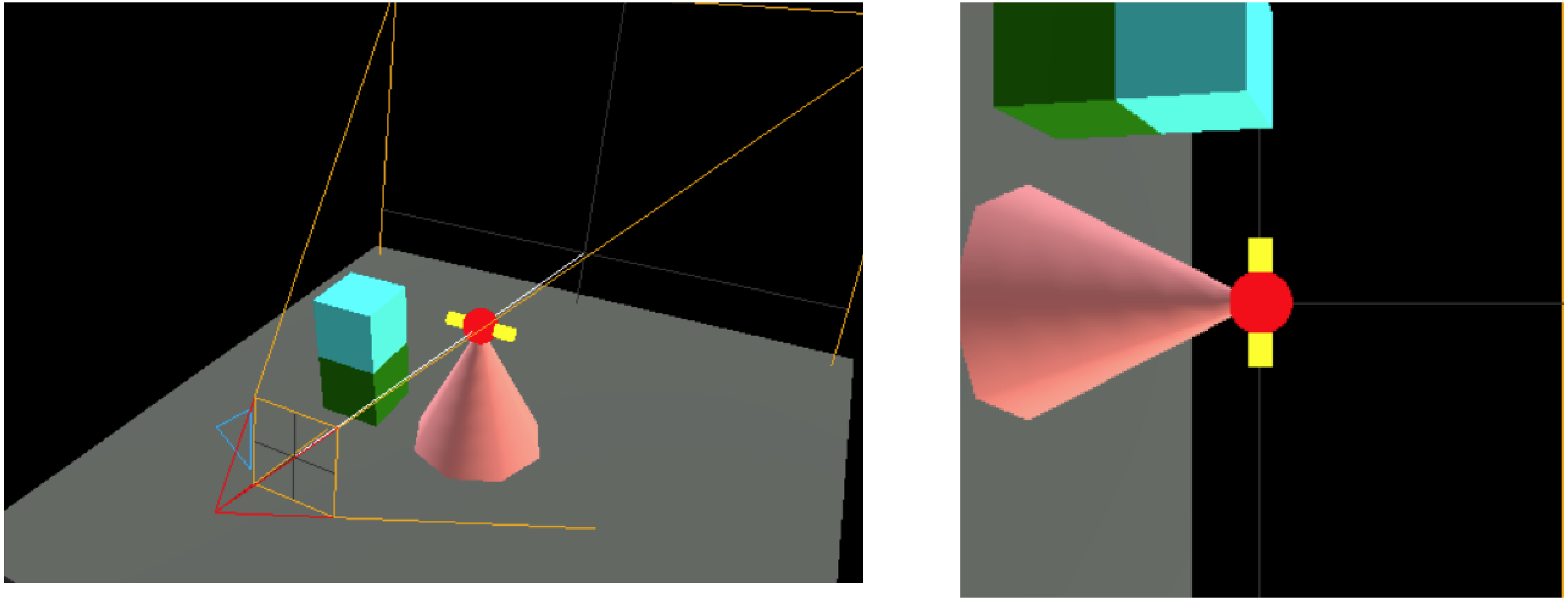


From the Demo (change Up)

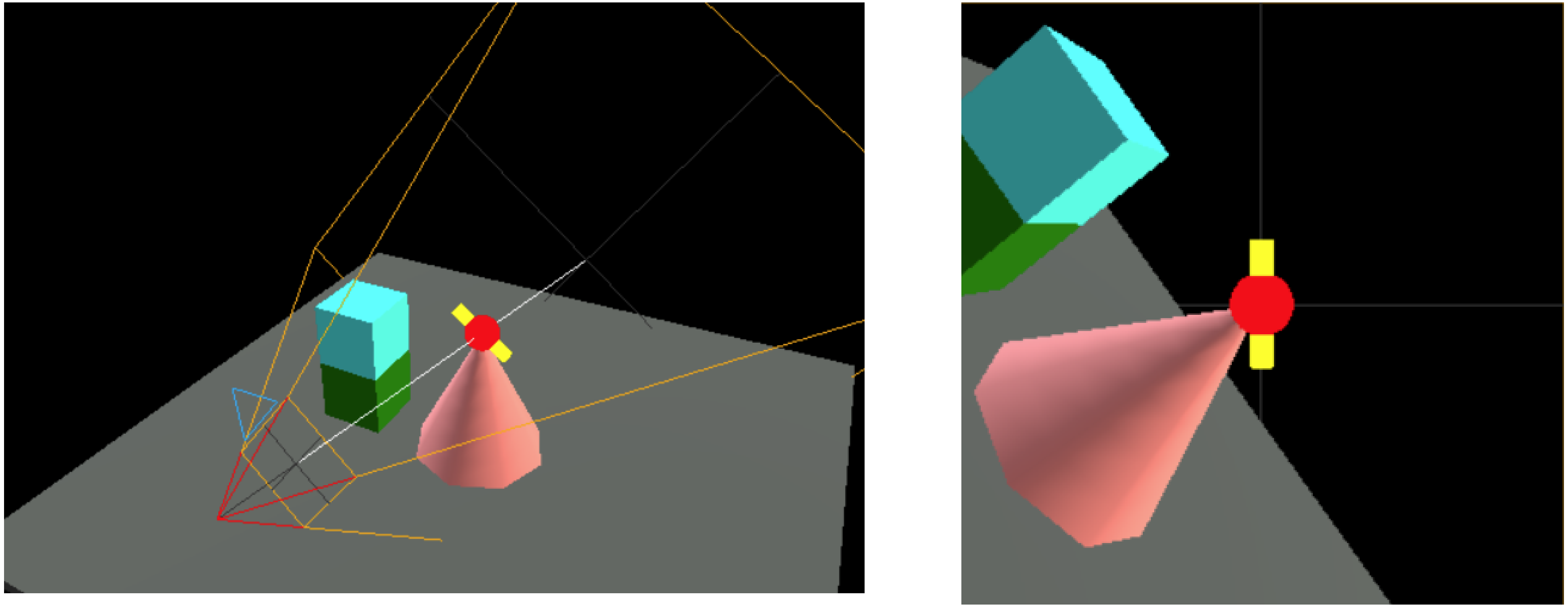




From the Demo (change LookAt)



From the Demo (change LookAt)



Demo Notes

- The red dot is something I am drawing
- The camera "frustum" is something I am drawing
- The yellow cylinder is something I am drawing

- Up can be any vector - I am controlling it via an angle (so 1 slider)

Describing Cameras (or any object)

Position "eye point" (center)

Rotate to "look at" something

- LookFrom (where to put the eye)
- LookAt (point the camera towards a point)
- Up (extra degree of freedom)

Lookfrom/Lookat/VUp

- implementing this is interesting (but not for today because...)

LookFrom/LookAt/VUp in THREE

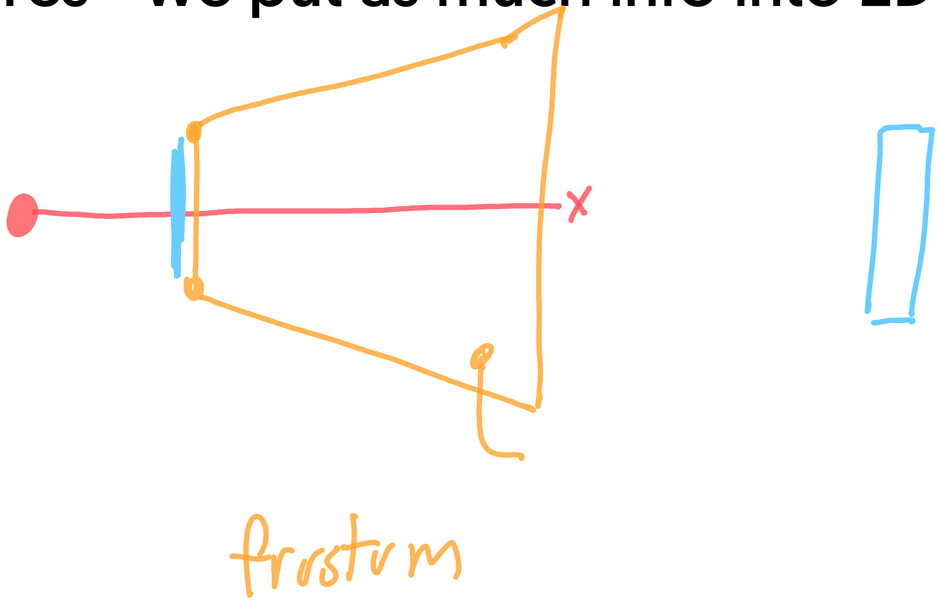
```
camera.position.set(fromX,fromY,fromZ); // normal translation/position
camera.up.set(upX,upY,upZ);  // this is a member variable
camera.lookat(atX,atY,atZ);  // uses the above two things

camera.fov = angle;      // another variable
camera.updateProjectionMatrix(); // need to recompute
```

- lookat works for any object3D
- note what is state vs. method
- recompute when variables change

Projection 3D to 2D

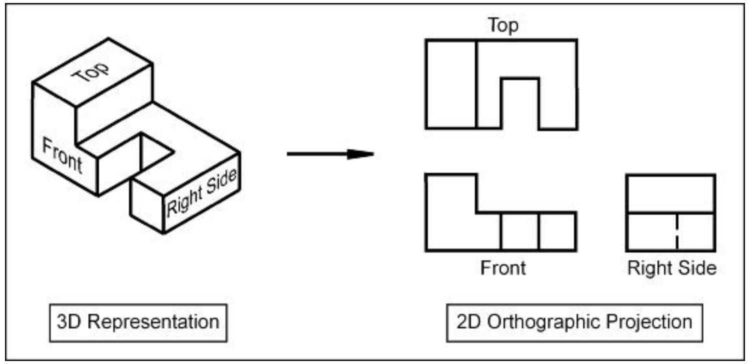
We lose a dimension

- No - we actually keep it (screen as a fishtank)
- Yes - we put as much info into 2D as possible

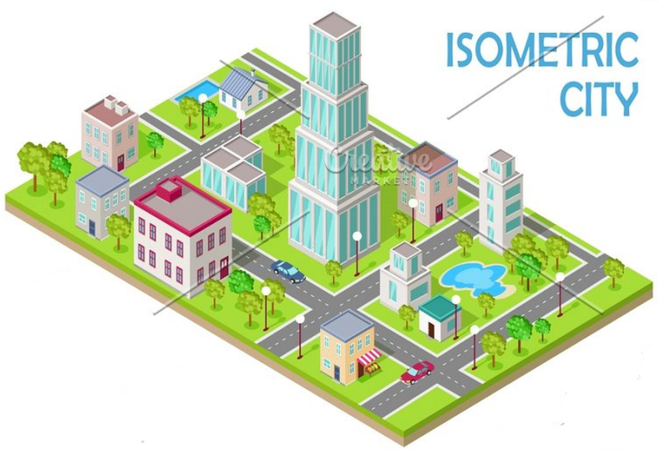


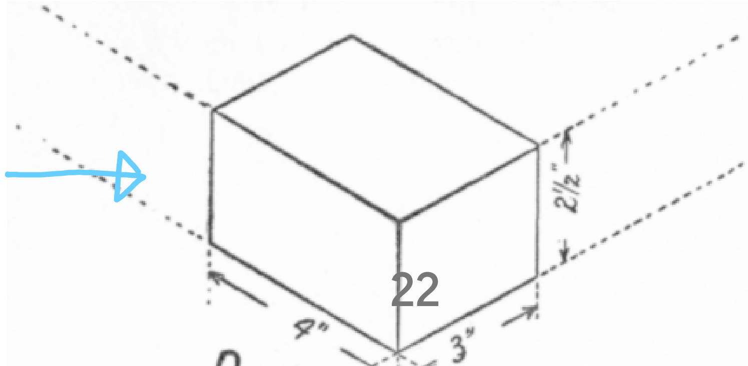
Types of Projections

Orthographic

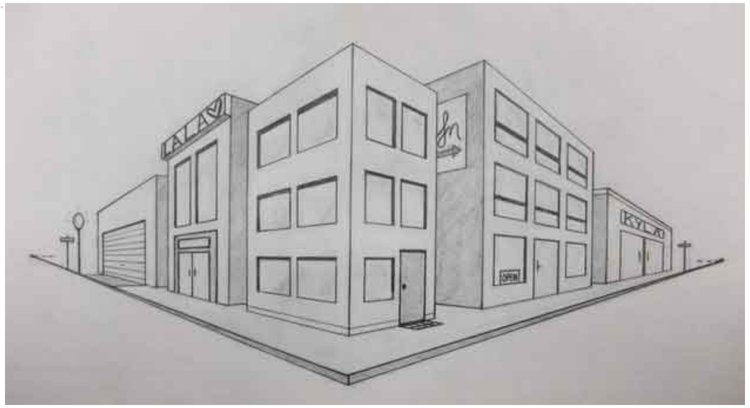


Isometric

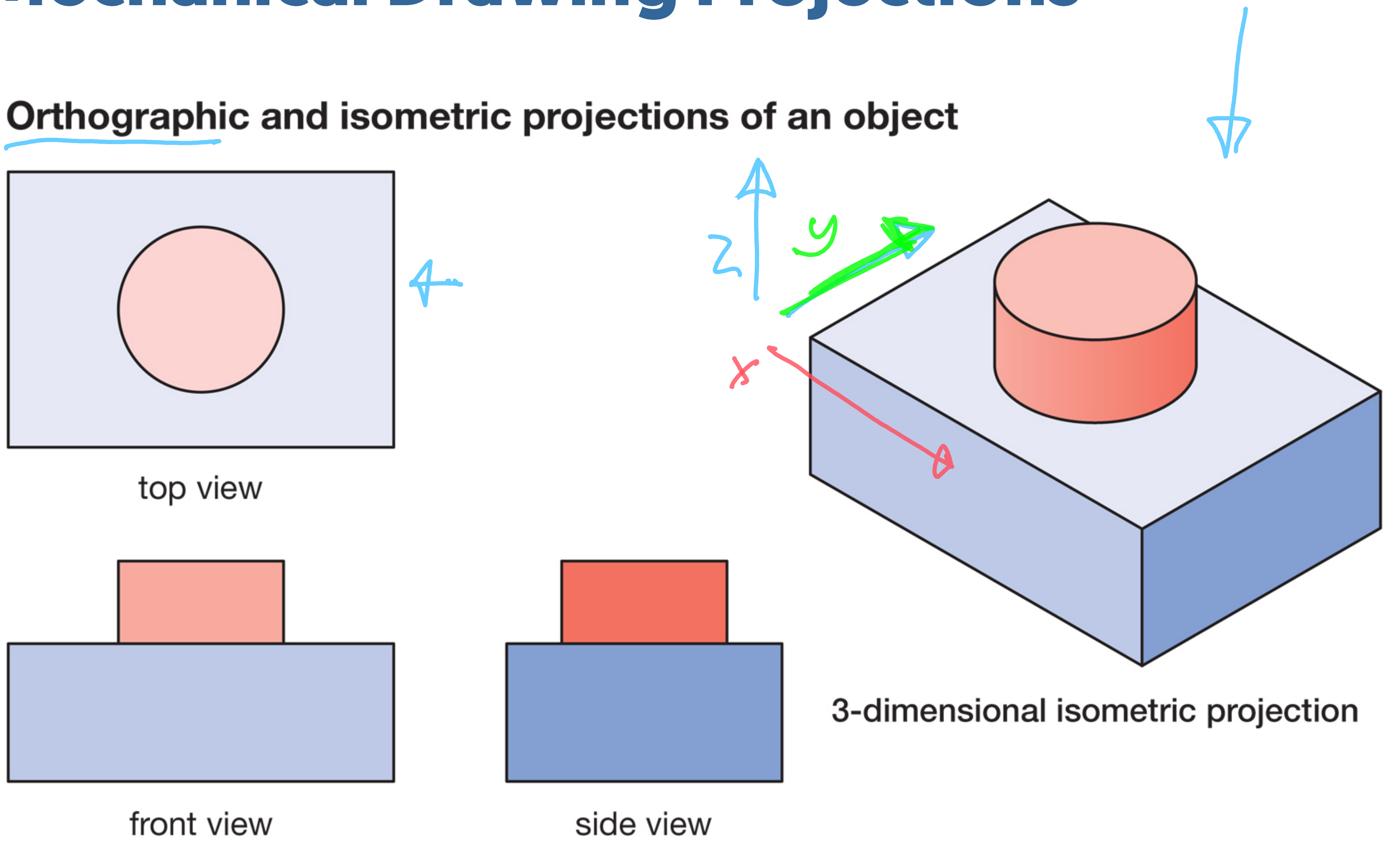




Perspective



Mechanical Drawing Projections



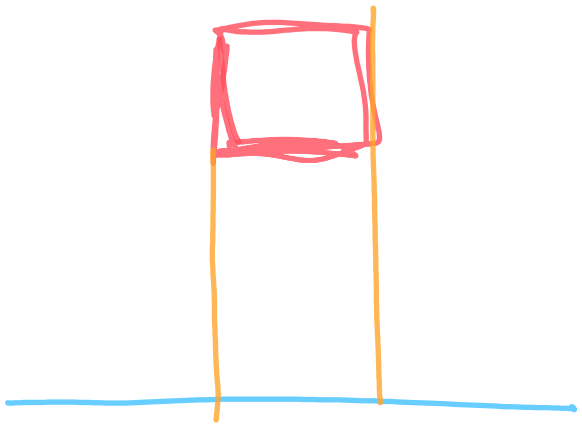
Orthographic and isometric projections of an object

3-dimensional isometric projection

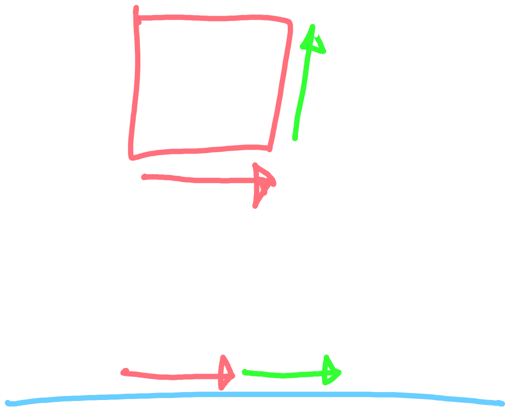
2-dimensional orthographic projection

Types of Projections

Orthographic



Isometric

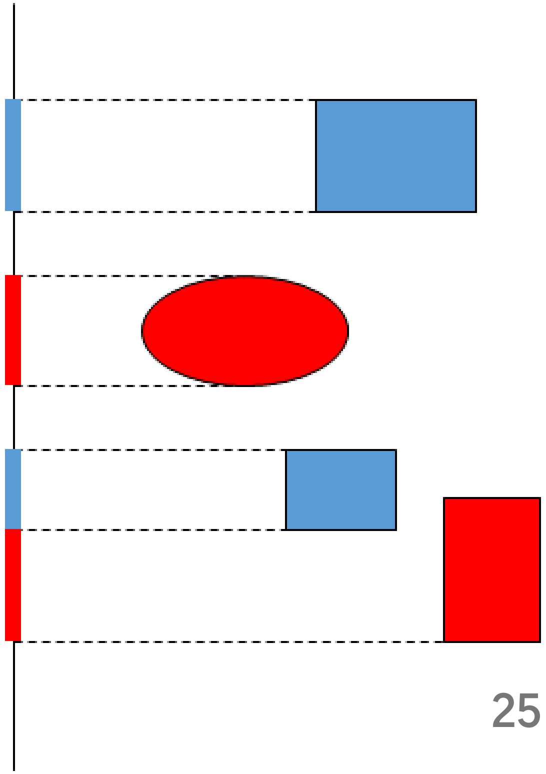


Perspective

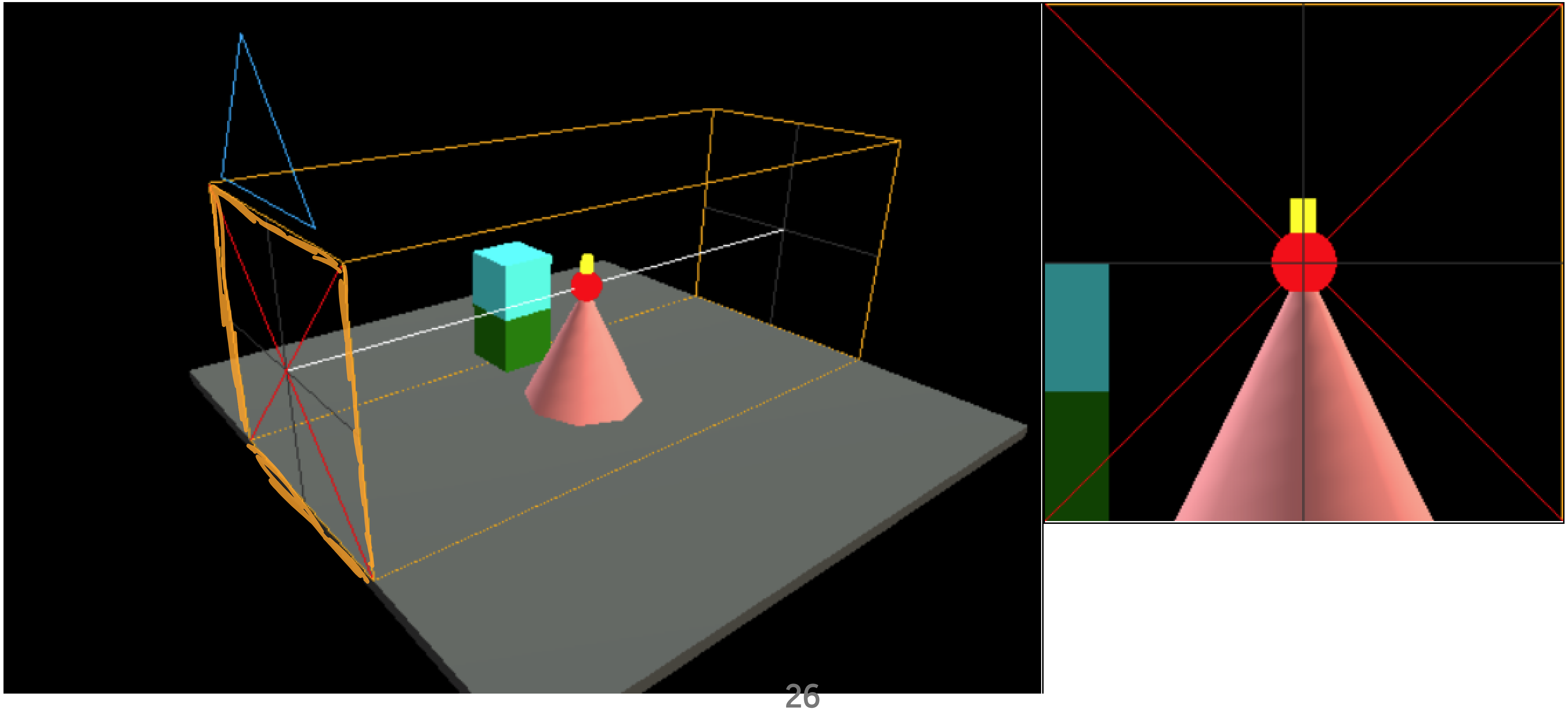
Orthographic Projection

Projection = transformation that reduces dimension

Orthographic = flatten the world onto the film plane



The Orthographic "Box"

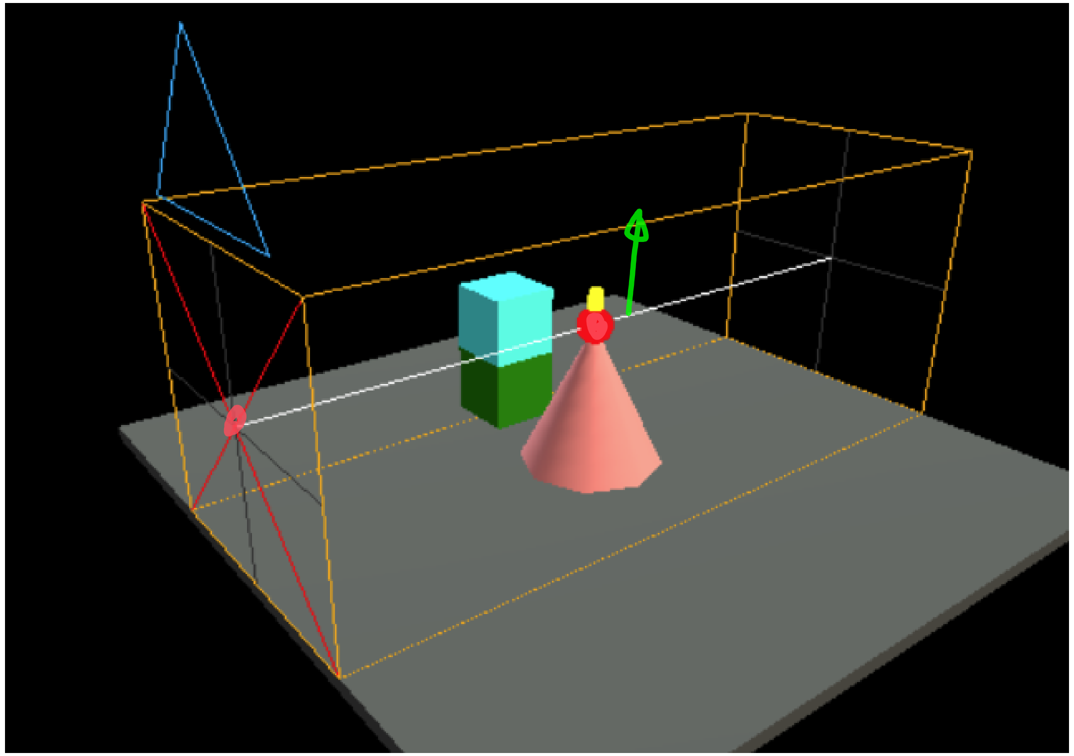


The Orthographic "Box"

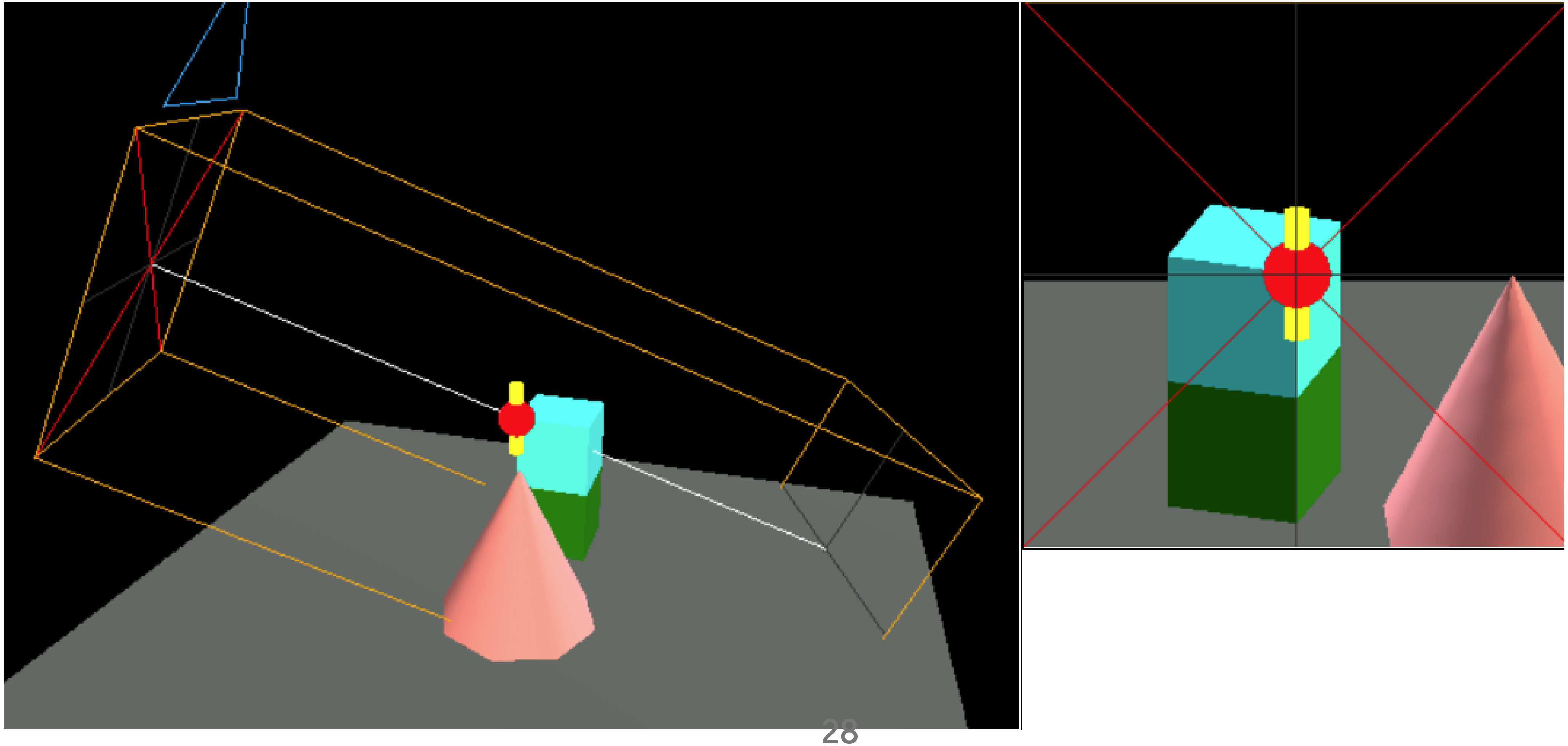
It is a "Camera Object"

It is a Box in the World

- position (eye point)
- forward direction (neg Z)
- up direction (Y)
- size (left/right/top/bottom)
- front/back



You can orient the Box (rotate)



Orthographic

```
new T.OrthographicCamera(-2,2, -2,2, -2,2);
```

The screen (x,y,z)

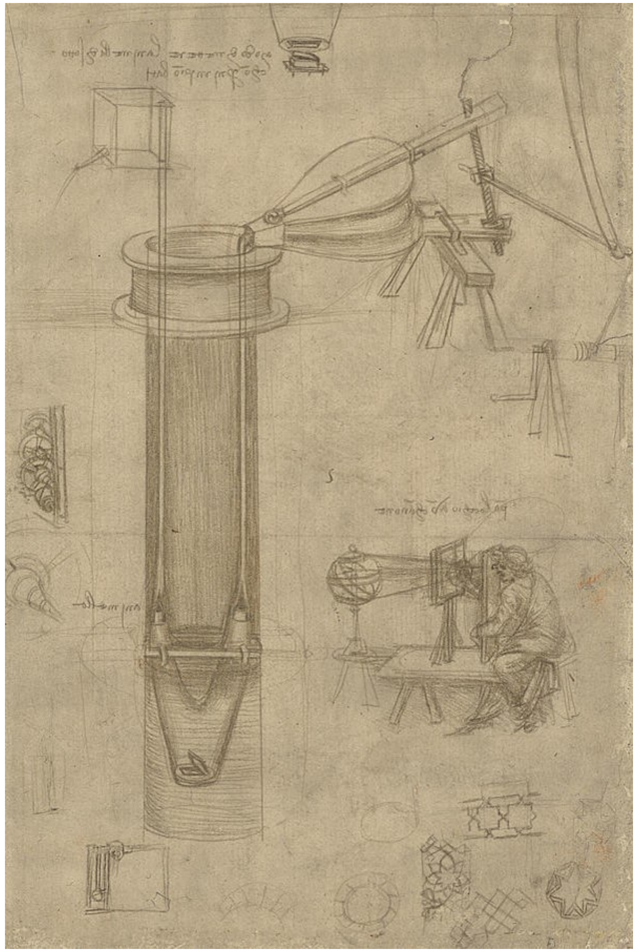
Shift and scale to fit

Rotations to get top, side, front

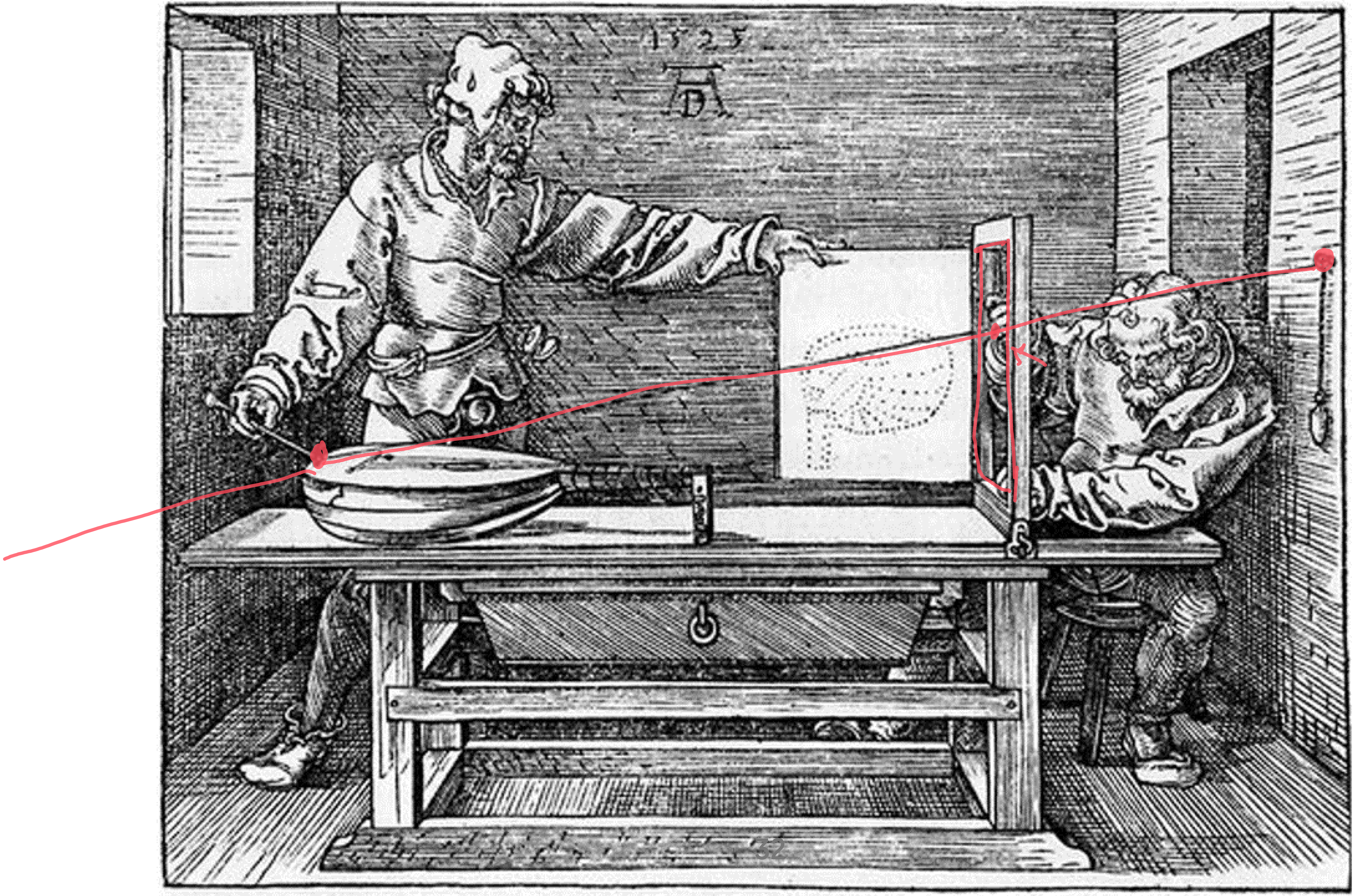
The need to scale in Z

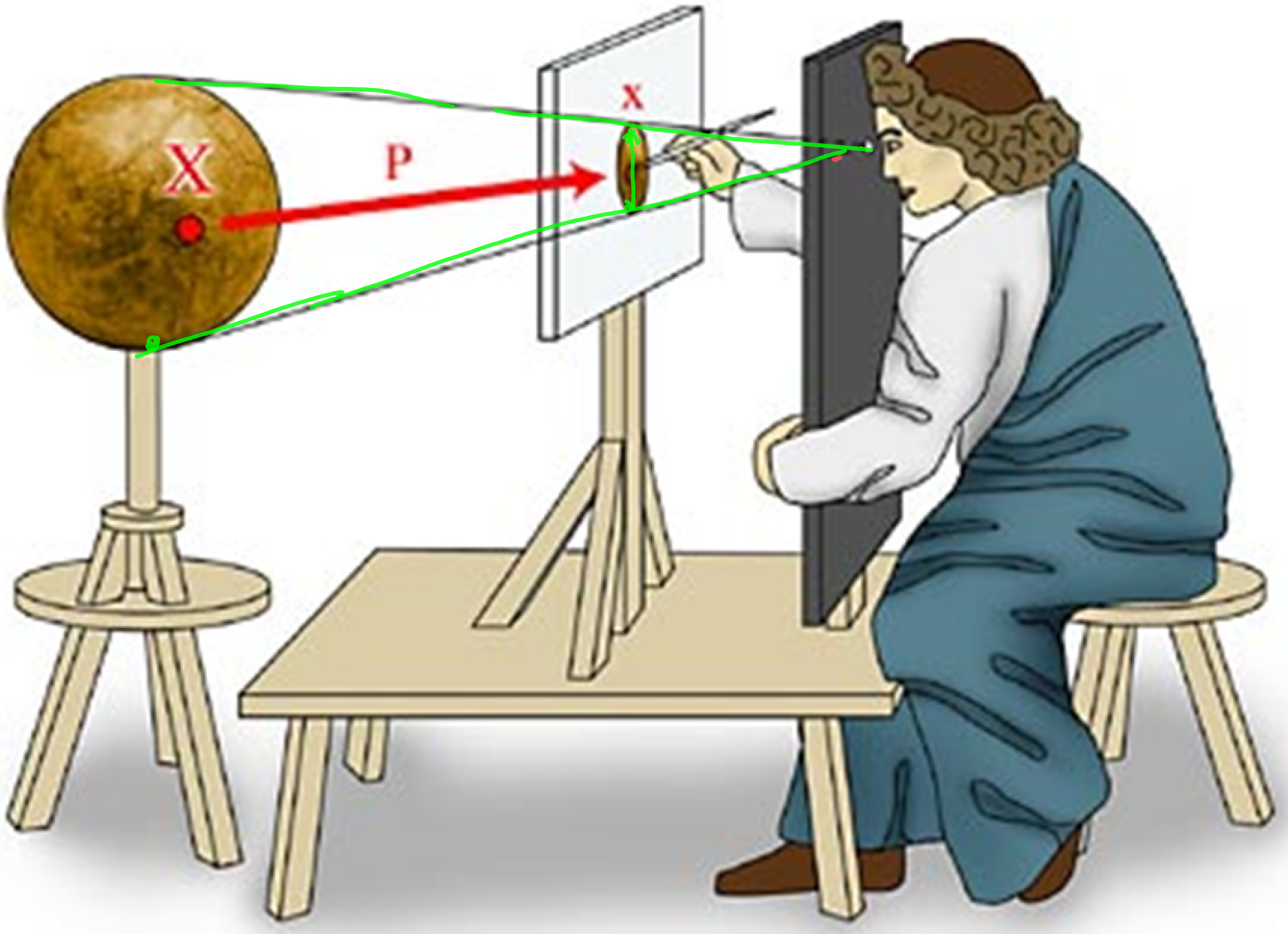
Perspective

Do it like Da Vinci!



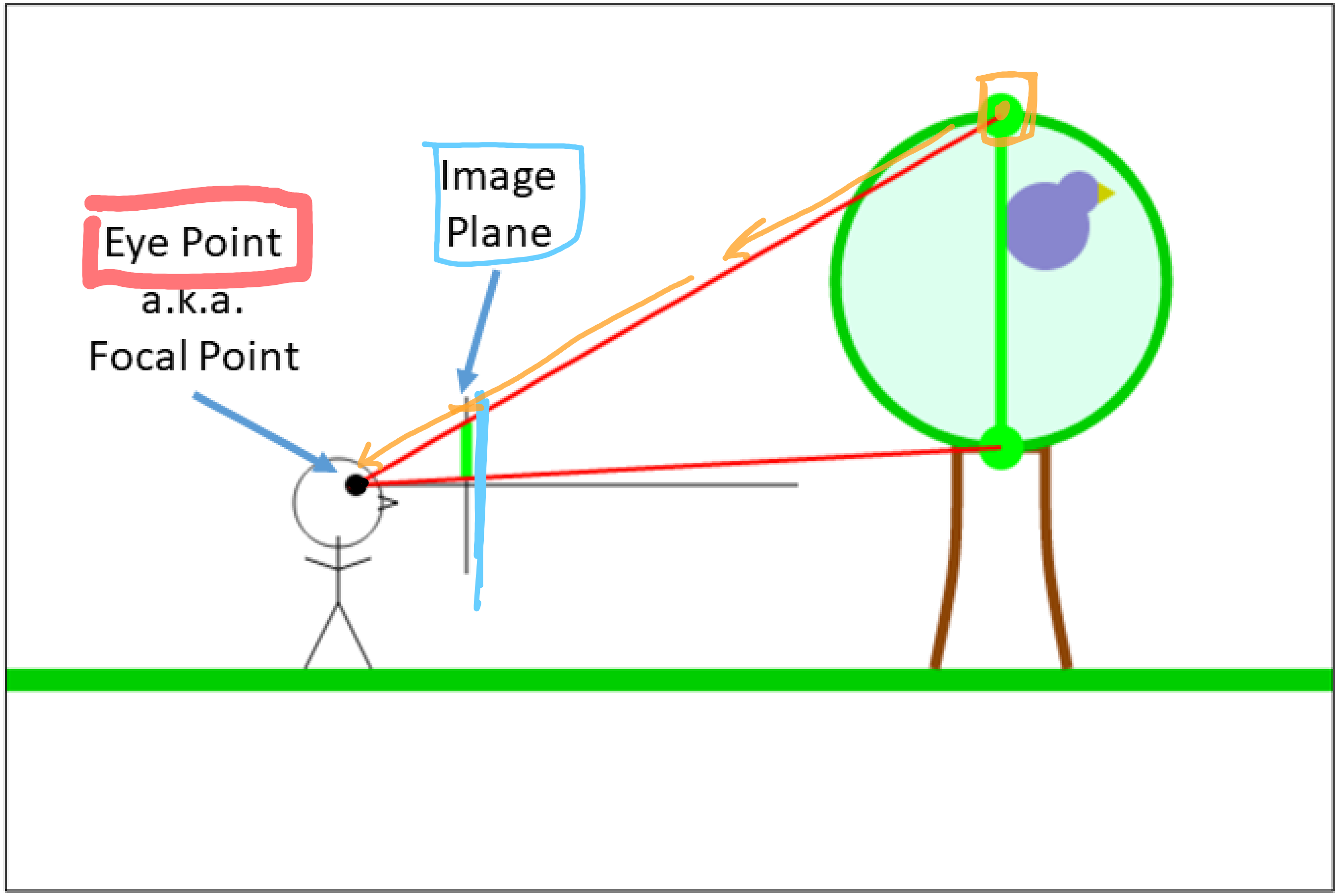
http://fineartamerica.com/featured/bellows-perspectograph-with-man-examining-inside-from-atlantic-codex-leonardo-da-


1525
AD



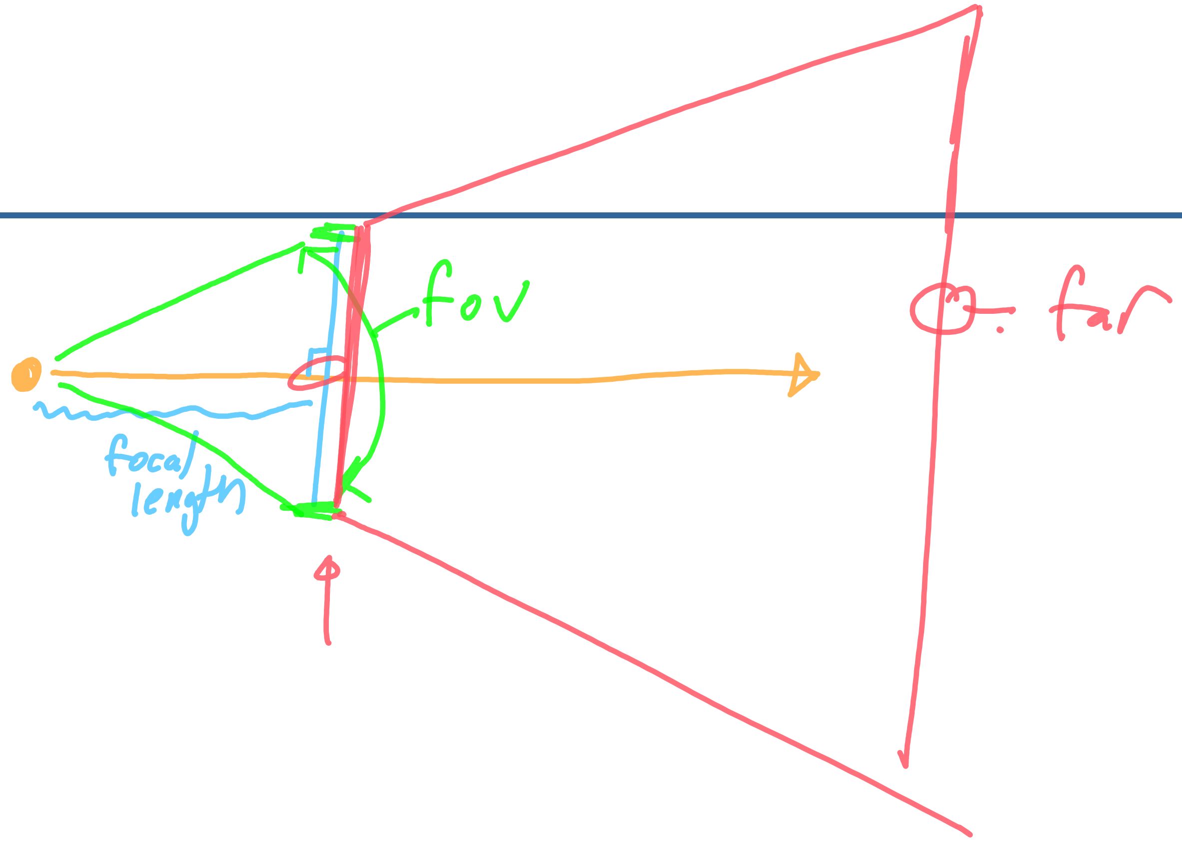
http://hans.wyrdweb.eu/about-perspective/

Perspective Imaging



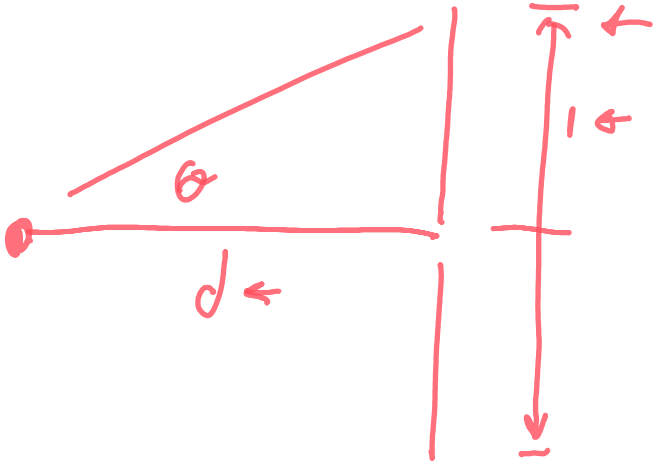
The intuitions

- focal point
- line of sight
- image plane
- focal length
- field of view — fov
- frustum



Field of View vs. Focal Length

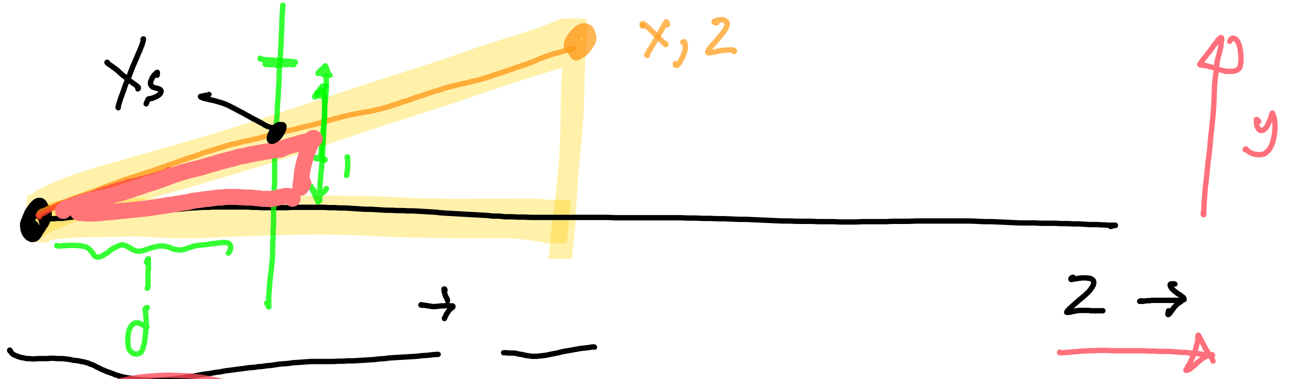
- angle
- distance (film size)



The Math

$$x_s = \frac{d}{z}x \qquad y_s = \frac{d}{z}y$$

This assumes that we are looking down the z axis



$$d \frac{x}{z} = \frac{x_s}{d} \qquad \cancel{\frac{z}{x} = \frac{d}{x_s}}$$

Linear? – in homogeneous coordinates

$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & \textcircled{1} \\ 0 & 0 & \textcircled{1} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad or \quad \begin{aligned} x_p &= d\,x \\ y_p &= d\,y \\ \underline{\underline{z_p}} &= \underline{\underline{1}} \\ \underline{\underline{w_p}} &= \underset{\uparrow}{z} \end{aligned}$$

Don't forget the divide by w!

Note what happens to z

$$X_s = \frac{x_p}{w_p} = \frac{\underline{\underline{d}}x}{z}$$

Is it really that simple?

Almost

A couple of catches:

- we need to scale z appropriately
- we need to scale x/y appropriately
- we're sighting down the positive/negative z
- the book discusses this well

The Matrix in the Book

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & \enclose{circle}{1} & 0 \end{bmatrix}$$

n - near plane distance

f - far plane distance

It's just a transformation!

Just like any other linear transformation

$$P \qquad C^{-1} \qquad p \qquad T \qquad T \qquad T$$

↑ perspective

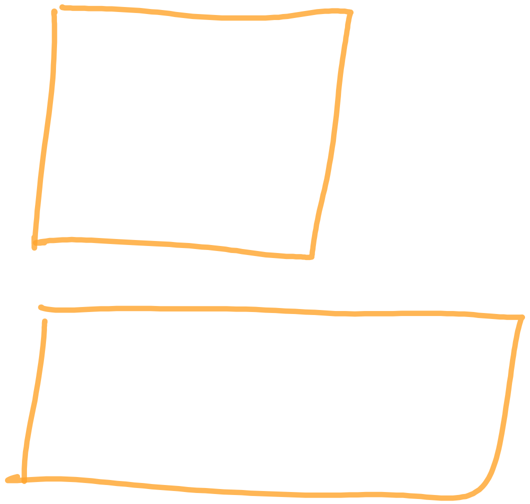
↑ camera

scene coord

In THREE

```
let cam = new T.PerspectiveCamera(fov,aspect,near,far);
```

- `fov` is angle in degrees
- `aspect` is width/height (needs to match canvas)
- `near` - anything closer is not seen
- `far` - anything farther is not seen



This is an Object3D.

It isn't visible, but it has all the transformations.

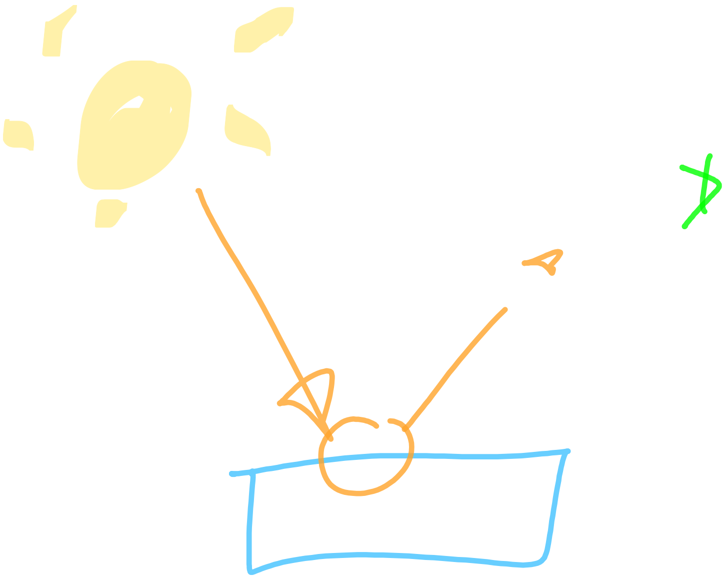
Lighting and Materials

Lighting and Shading
A brief intro

A topic we will return to later in the class

What color is something?

- specify pixel value (2D)



real world

- material
- geometry
- light

standard 3D programming

- compute color from material, geometry, light

Material and Lighting

The material responds to lights

How a point (pixel) appears depends on:

- the surface properties
- the surface orientation
- the color/intensity of the light
- the direction of the light

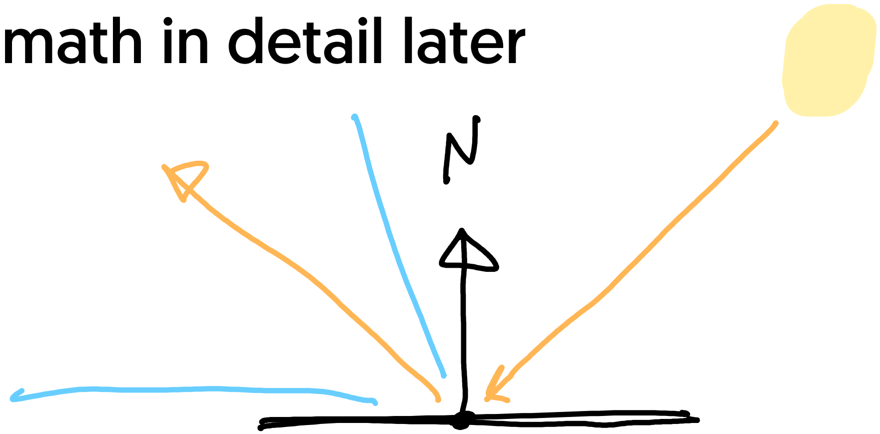
For now, light travels direct from source to point

- Local Lighting - no shadows / reflections / spill

Shading Intuitions

What does direction, shininess, normals, have to do with it?

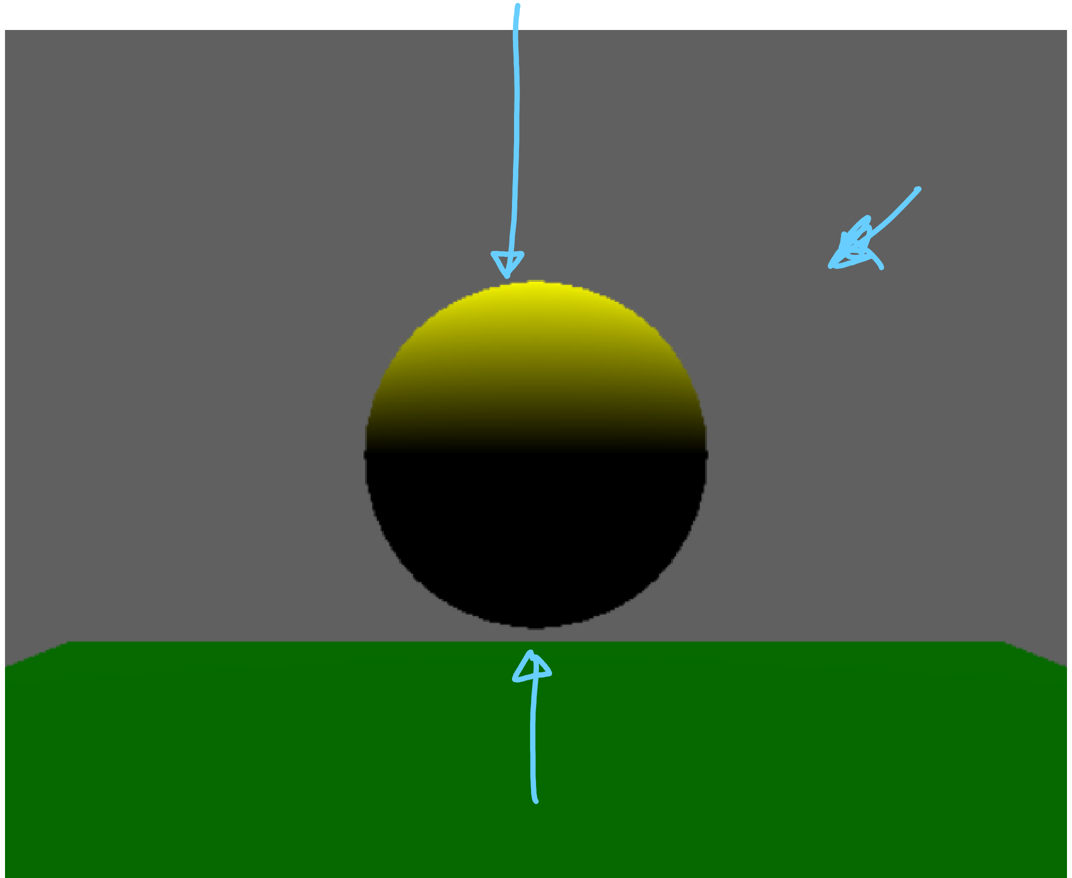
We'll look at the math in detail later



Simple Surface Model

Diffuse

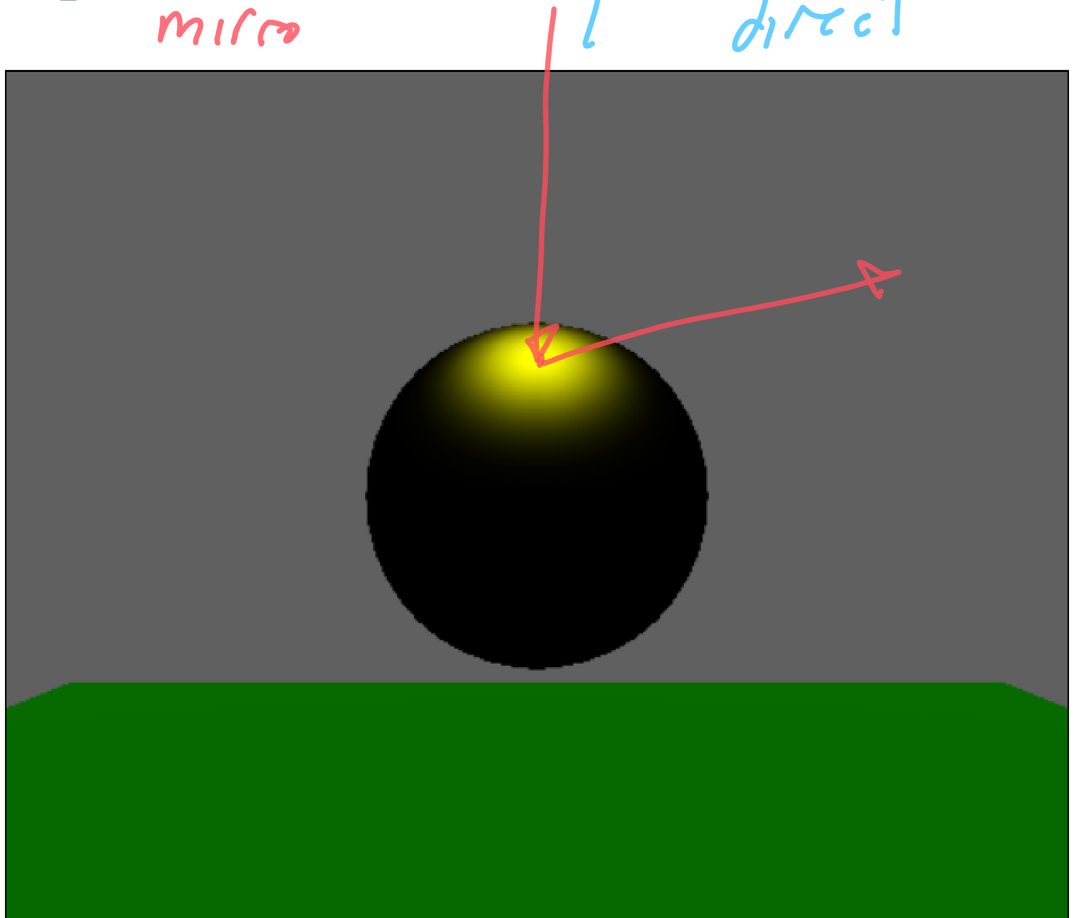
- light in all directions



Specular

mirro

- light bounces in 1 direct



Colors

- Surfaces have colors
    - per material
    - per vertex (triangle?)
    - more colors later
- Lights have color

- Red light on white object = red
- White light on red object = red
- Red light on blue object? - nothing

Add lights

```
let ambientLight = new T.AmbientLight ("white", 0.5);
scene.add( ambientLight );
let pointLight = new T.PointLight( "white", 1 );
pointLight.position.set( 25, 50, 25 );
scene.add( pointLight );
```

The lights are objects in the world

We control their *transformation* to place them

Types of Lights

Ambient Light

Types of Lights

Point **Directional** **Spot**

Lights in THREE

- They are just Objects!
- You can position and orient them

- THREE's materials know to look for them

Summary

1. Use Materials and Lights to create appearance
2. Color depends on geometry, material, and lighting
3. Specular and Diffuse material properties
4. Local lighting
5. Lights with different geometries

Animation in THREE

Some details to know

The Animation Loop

```
let lastTimestamp; // undefined to start
function animate(timestamp) {
  let timeDelta = 0.001 * (lastTimestamp ? timestamp - lastTimestamp : 0);
  lastTimestamp = timestamp;

  cube.rotation.x += 0.5 * timeDelta;
  cube.rotation.y += 0.5 * timeDelta;

  renderer.render(scene, camera);
  window.requestAnimationFrame(animate);
}
window.requestAnimationFrame(animate);
```

The new pieces

Update objects: (change their transformations)

```
cube.rotation.x += 0.5 * timeDelta;
cube.rotation.y += 0.5 * timeDelta;
```

Redraw:

```
renderer.render(scene, camera);
```

Animation in THREE

- it's a scene graph
- we update the objects
- we ask three to redrawn the world

Warning:

- not everything is easy to change
- hard to understand unless we know what is happening inside
- We are not talking about THREE's animation system

What is easy to animate?

Easy

Change a transformation

Change a material property (*)

Change a light property

Properties designed to be animated

- small number of numbers
- specialized mesh operations

Hard

Change points in a Mesh

Change a material

Change a light type

- Anything that requires sending large data to the **hardware**
- Anything that requites recompiling a **shader**

Transformations

Put objects in places

Make objects move by transforming them

```
    cube.rotation.x += 0.5*timeDelta;
    cube.rotation.y += 0.5*timeDelta;
```

Do not move objects by modifying vertices!

- too many vertices to change
- need to rebuild data structures
- need to send data to graphics card

Summary

1. Use animation loops with THREE
2. Update the scene and re-render
3. Only change what is easy to change
    - move objects by transformation!