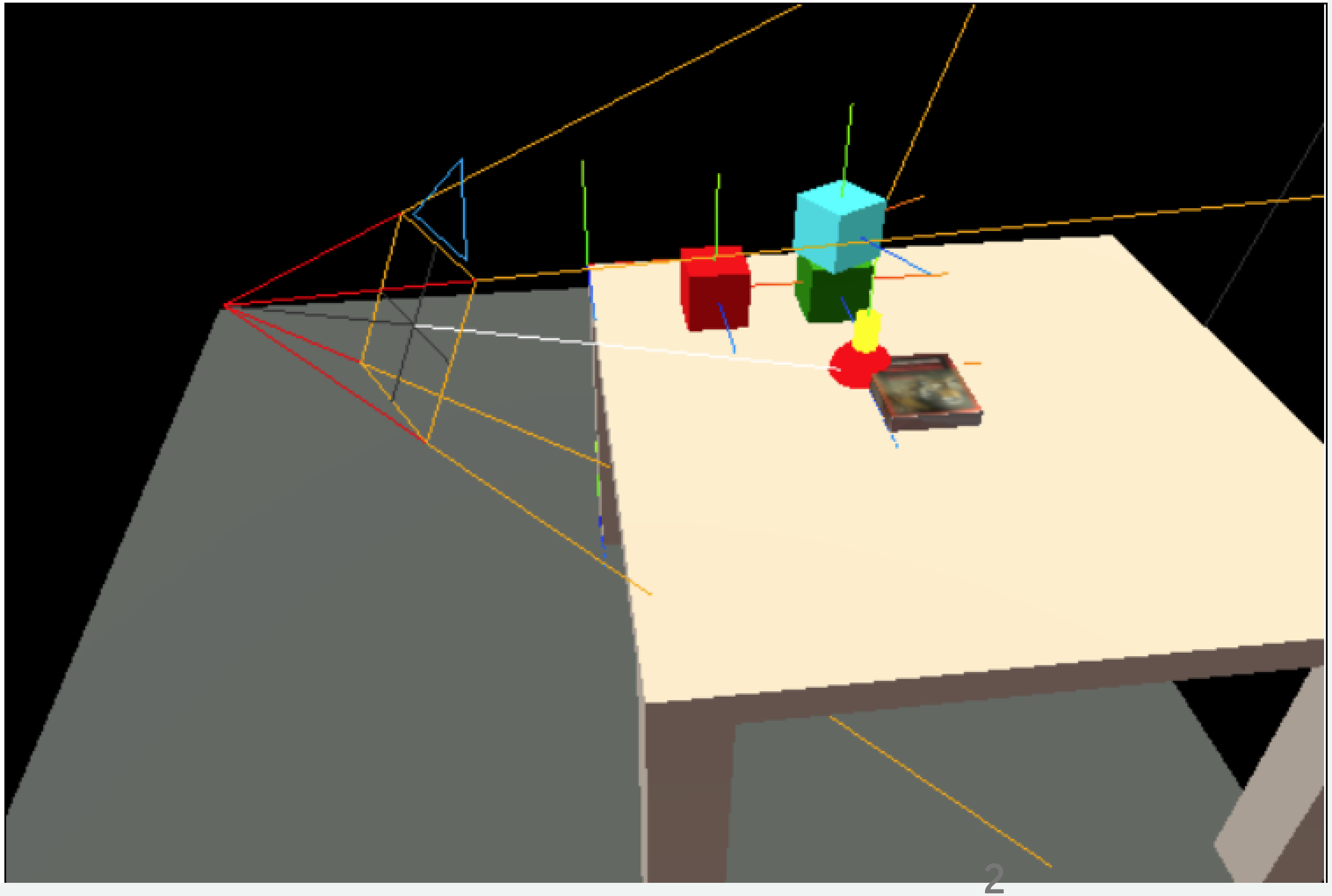
Lecture 14:
3D and THREE

A World...



Some notes...

I am drawing coordinate systems

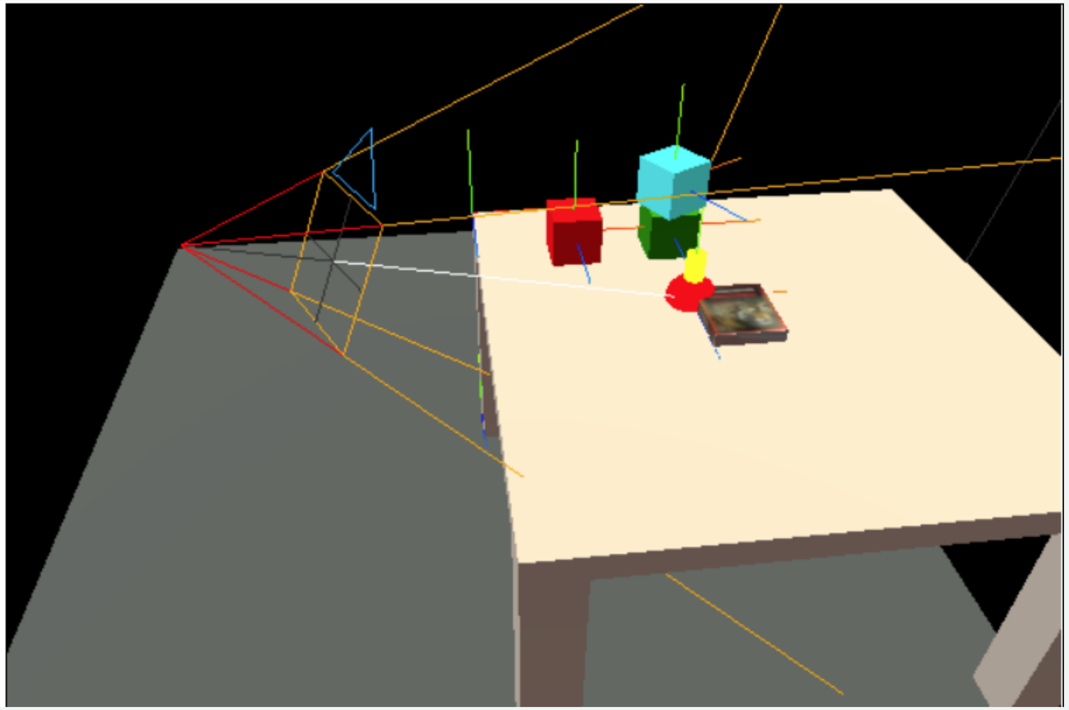
WebGL makes the lines thin

- Three.js uses WebGL

The Camera is an object

- normally it doesn't show up
- I am explicitly drawing the lines

Code mixes class framework and "direct" use of Three.js



What's Here

Table in Scene

Book on Table (in Scene)

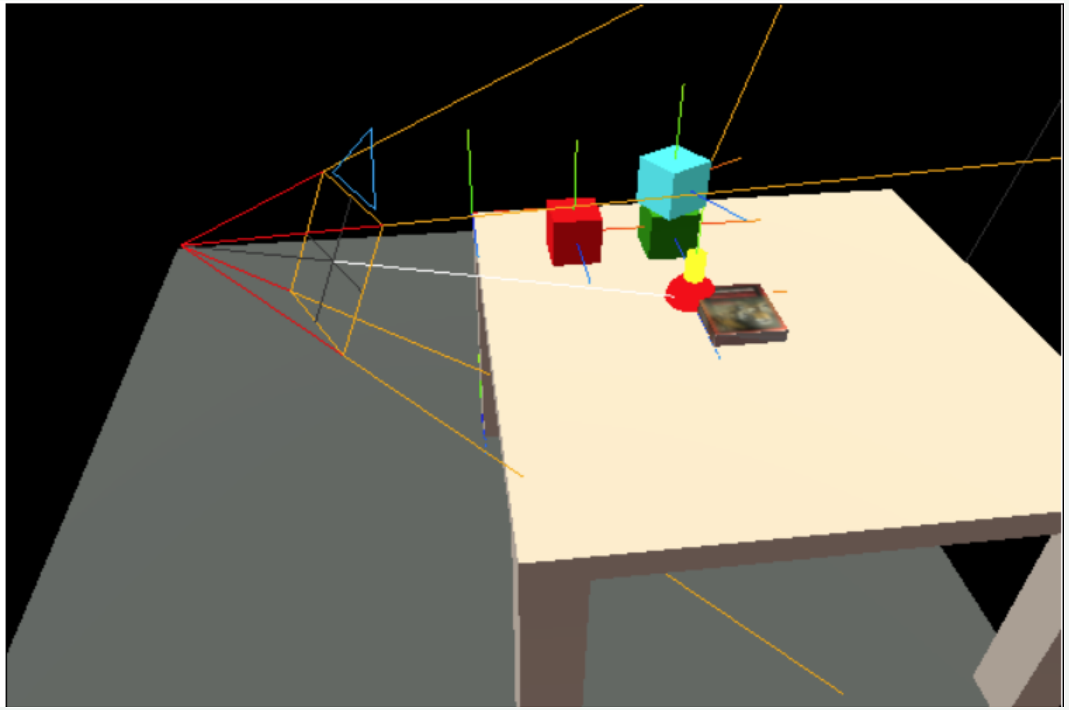
Red Cube on Table (in Scene)

Green Cube on Table (in Scene)

Cyan Cube on Green Cube (...)

An Extra Camera

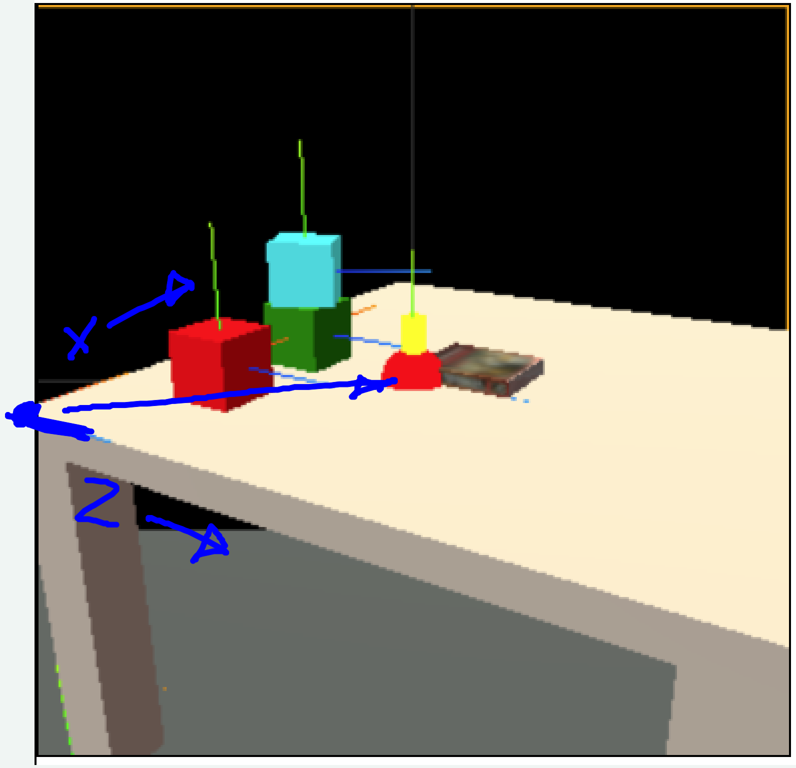
And the camera we're looking from



Making a Scene

```
scene.add(new Table());

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);
table.add(c3);
```

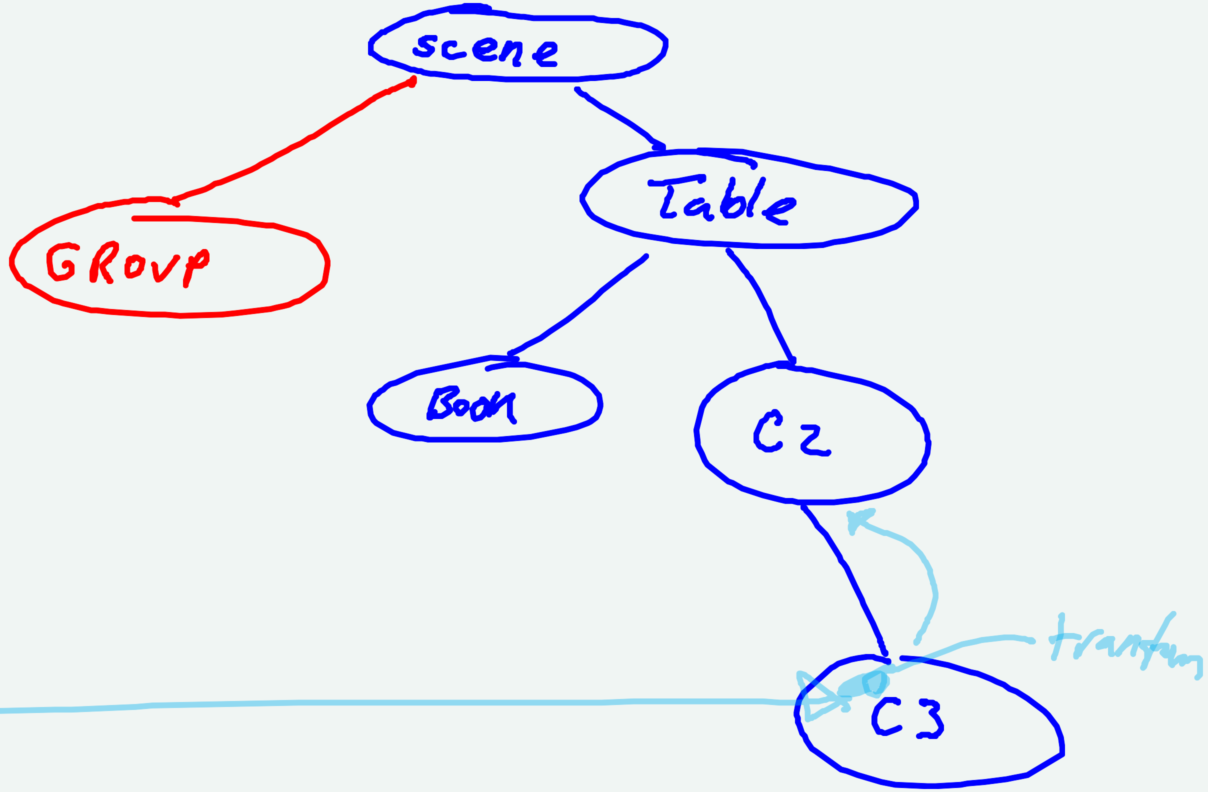
C2



Making a Scene (Graph)

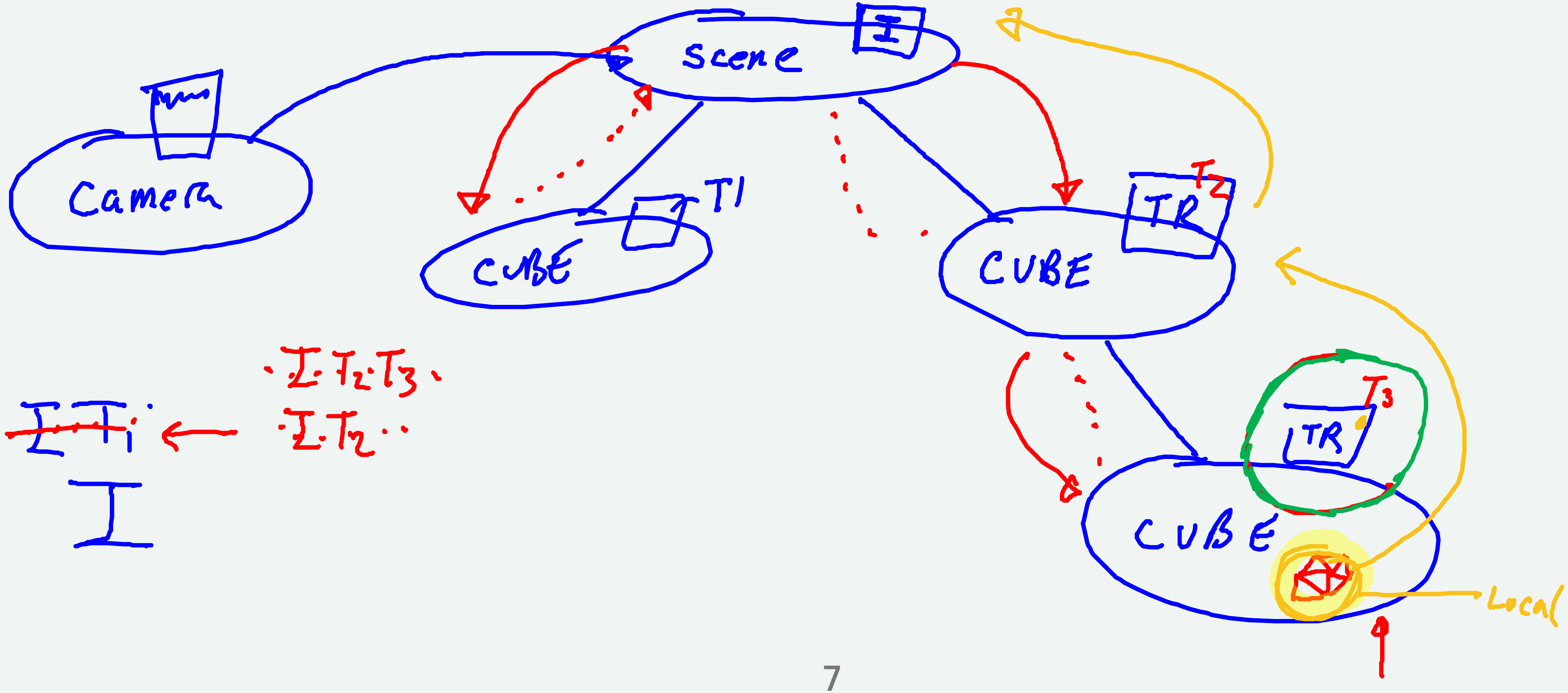
```
scene.add(new Table());

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);
table.add(c3);
```

C2



Where do the Matrices Live?

Tree traversals when we draw (convert to immediate)



THREE as an API

It is a **scene graph** API

We do need to explicitly render (immediate)

It is like SVG in some ways

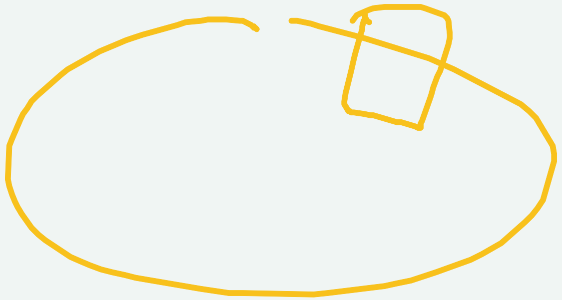
THREE Object3D

- Mesh (or not) → Geometry, Material

  Triangles   how it looks
- Transformation
- Children

Transformations in Three

Objects each have their own transformation



Objects "have" a matrix

- but it is built from pieces each time
- keep pieces separate for convenience (confusion?)

Objects have methods to perform transformations

move = translateX

Objects have state that can be set directly

position = set

In THREE.js

Internally, it builds the matrices for you

Provides many different ways to specify things

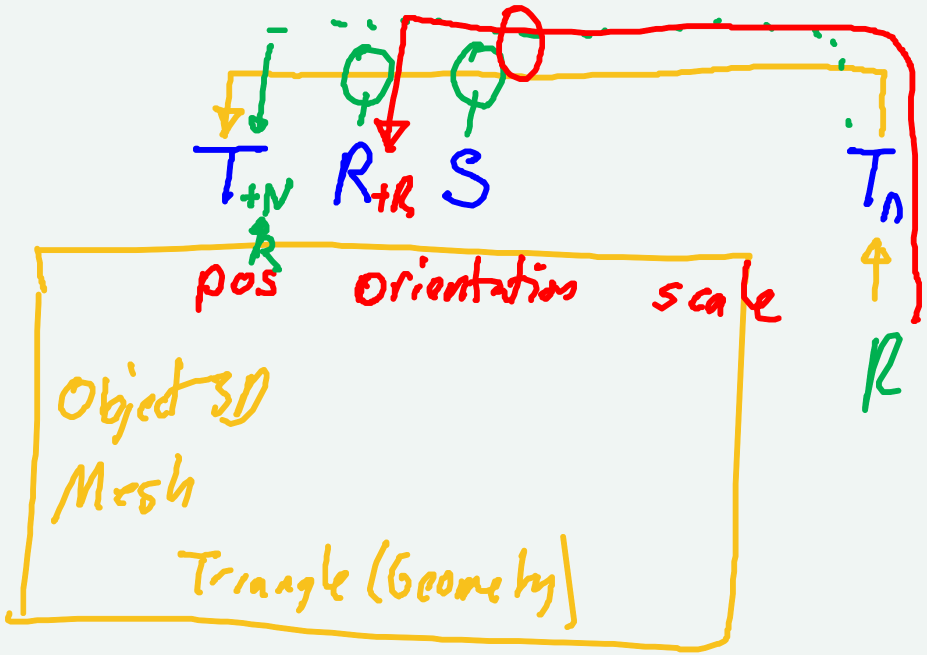
- rotations in several forms
- different ways to combine transformations
- hierarchies

You can control the transformations / matrices directly

- But you need to tell THREE not to over-write what you put in

Inside of a THREE object

- State: position, orientation, scale
- Orientation in many forms
- Matrix
- When when changes, others are updated
- Transformation commands



translate

rotate

State vs. Transformation

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

center of coordinate system

vs.

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

vs.

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

T R S

p

T R S T_N

OLD + RST_N

T R S

How THREE works inside

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

Move transformations through existing transformations

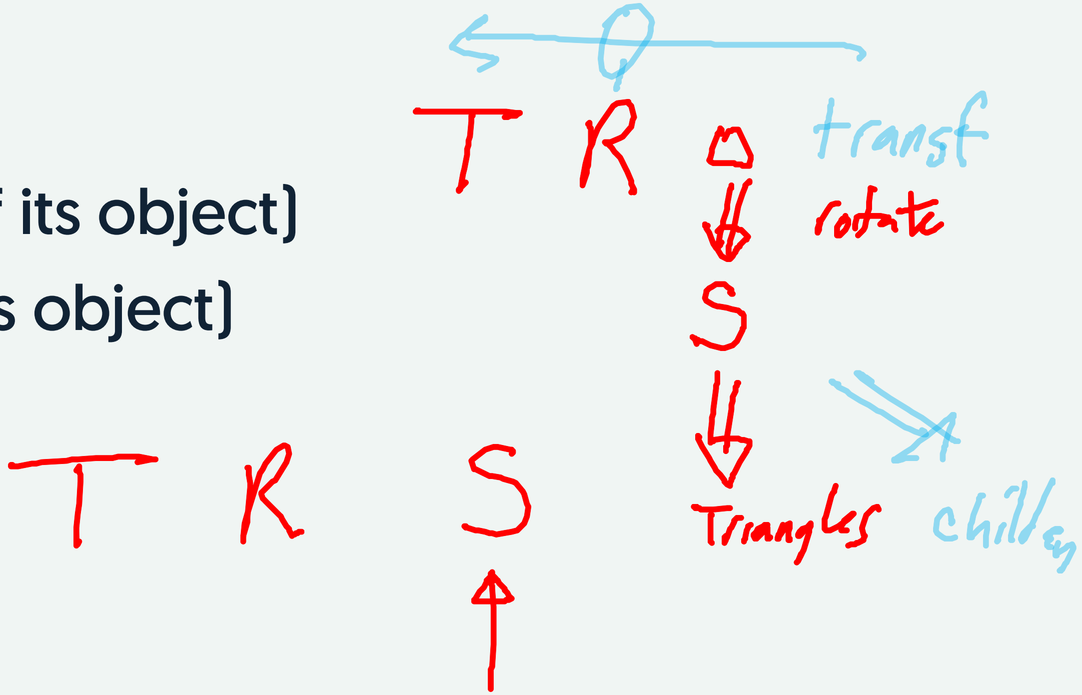
Scale

Is state (what is the scaling factor)

It is applied last (after translate/rotate)

It is not affected by other transformations (of its object)

It does not affect other transformations (of its object)



It does transform its children

It is the **local** scaling (according to the documentation)

Rotations

Warning: rotations in 3D are tricky!

Three gives us many different ways to do them

It always converts to a special format

We can do rotations or orientations

Summary

- We built scenes with hierarchy
- Cameras are (special) objects in the scene
- Transformations between objects
- THREE hides the transformations inside objects

What's Next

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

- More details of shape and lighting
- Texture

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:

orienting

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"

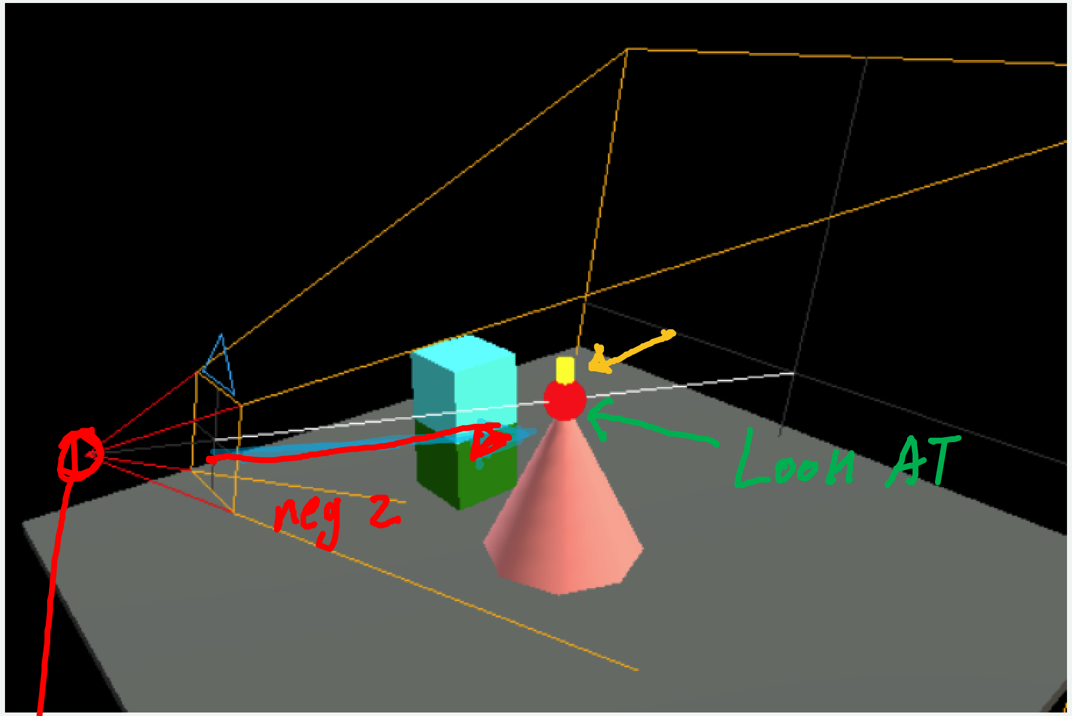
Rotate to "look at" something

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

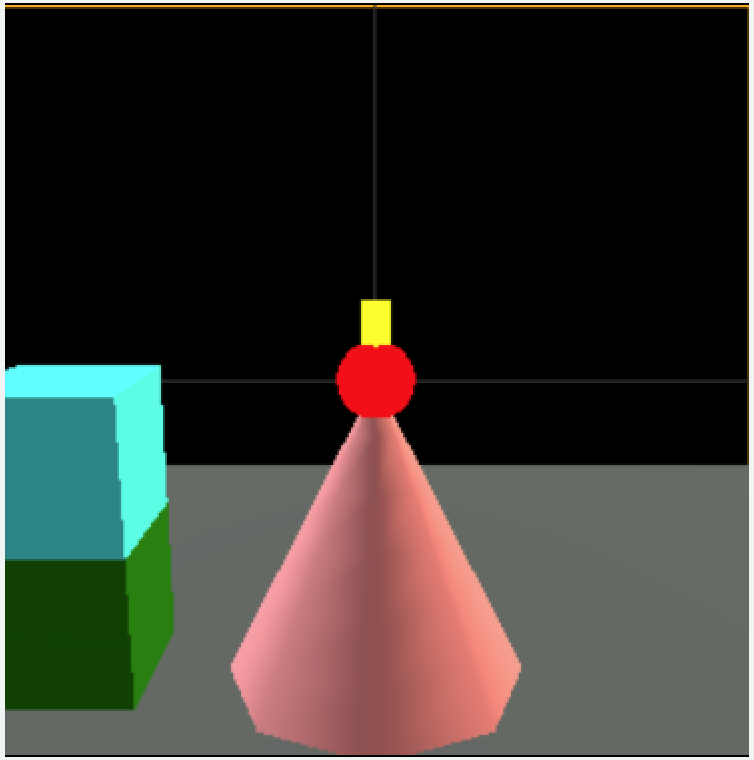
Lookfrom/Lookat/VUp

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

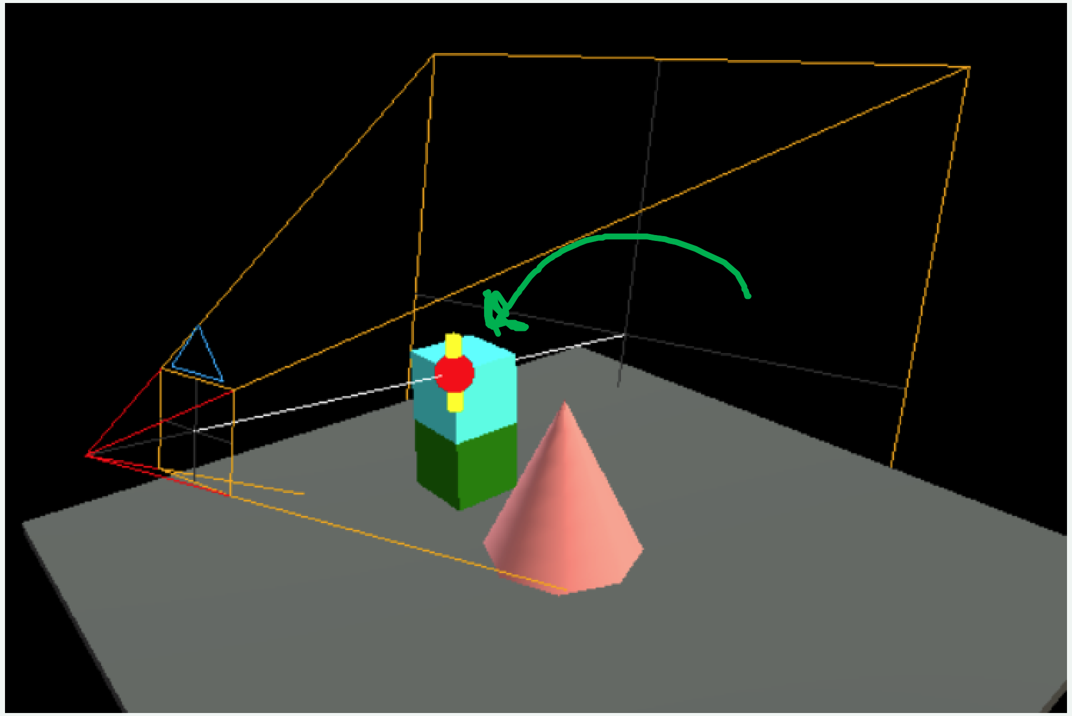
From the Demo

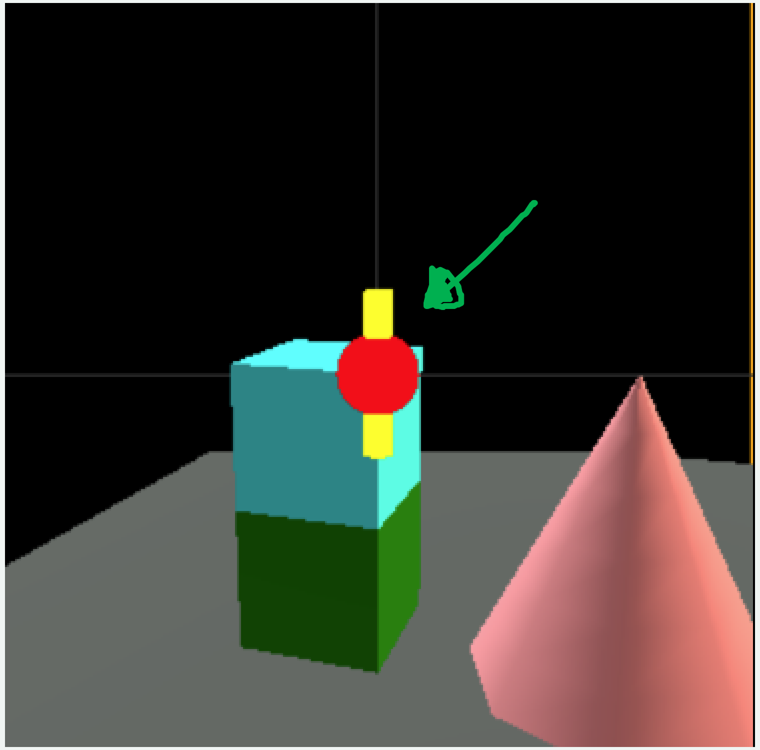


Lookfrom

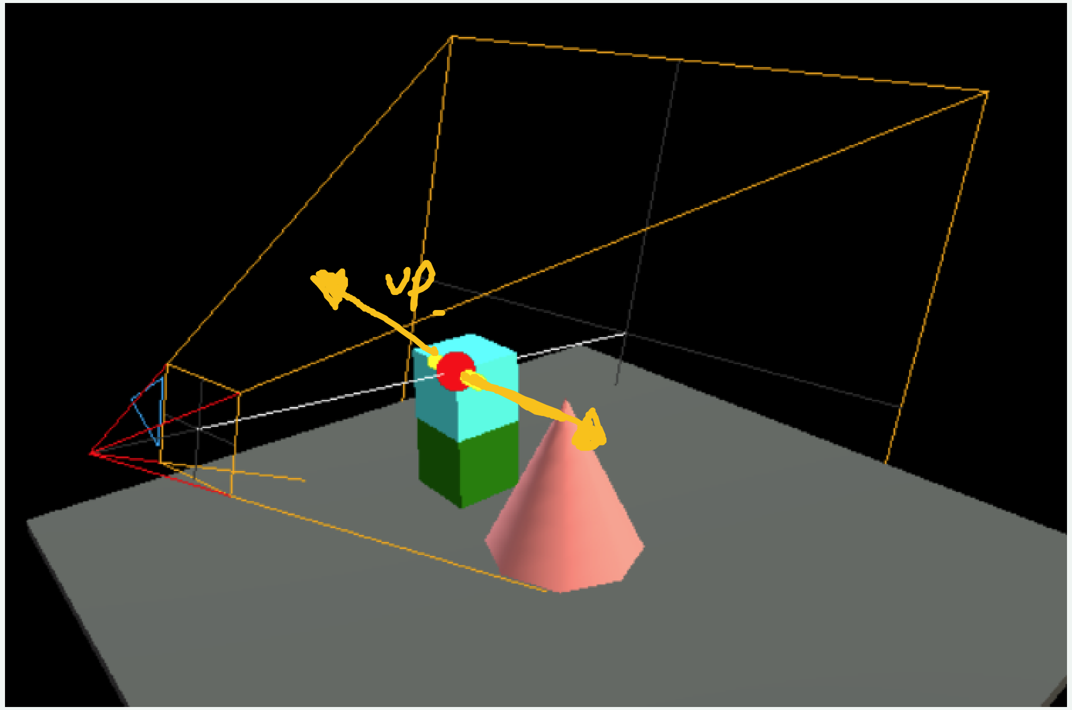


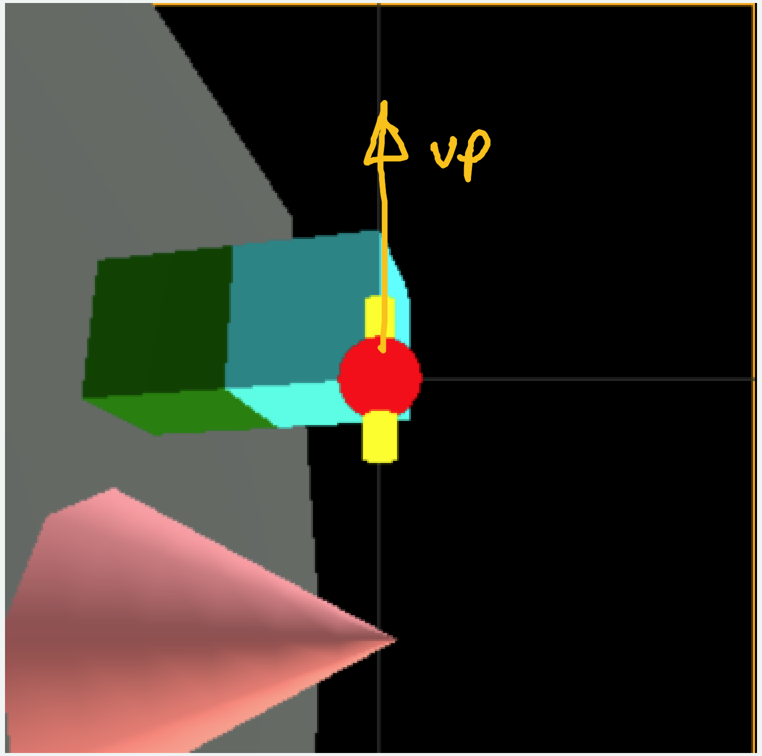
From the Demo (change LookAt)



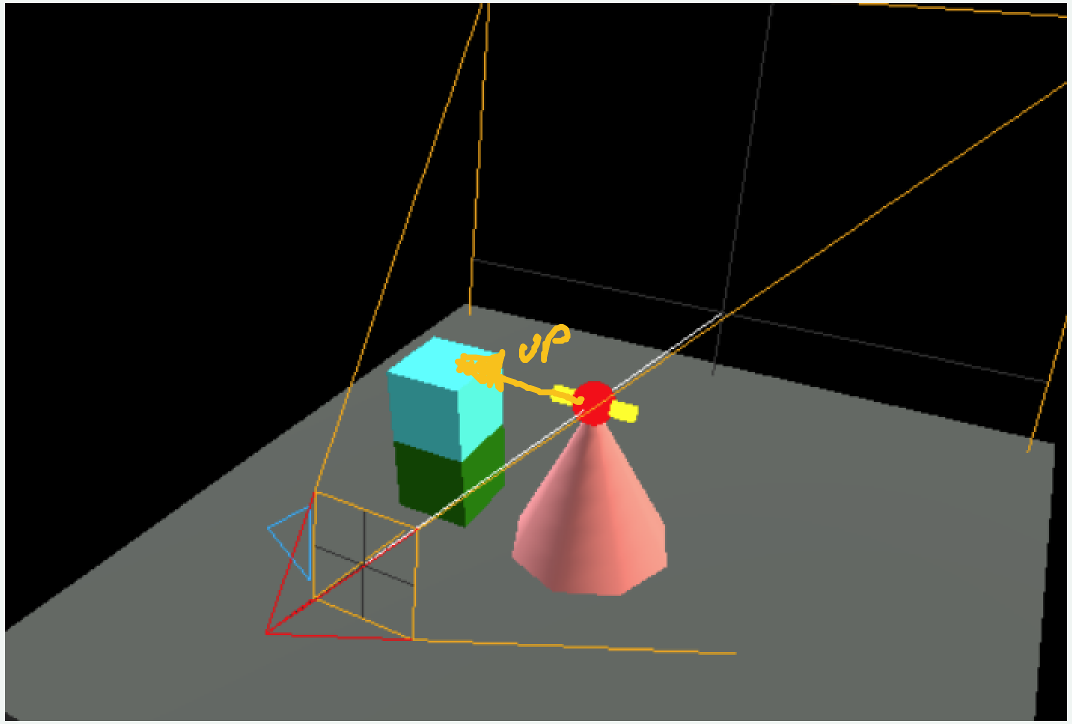


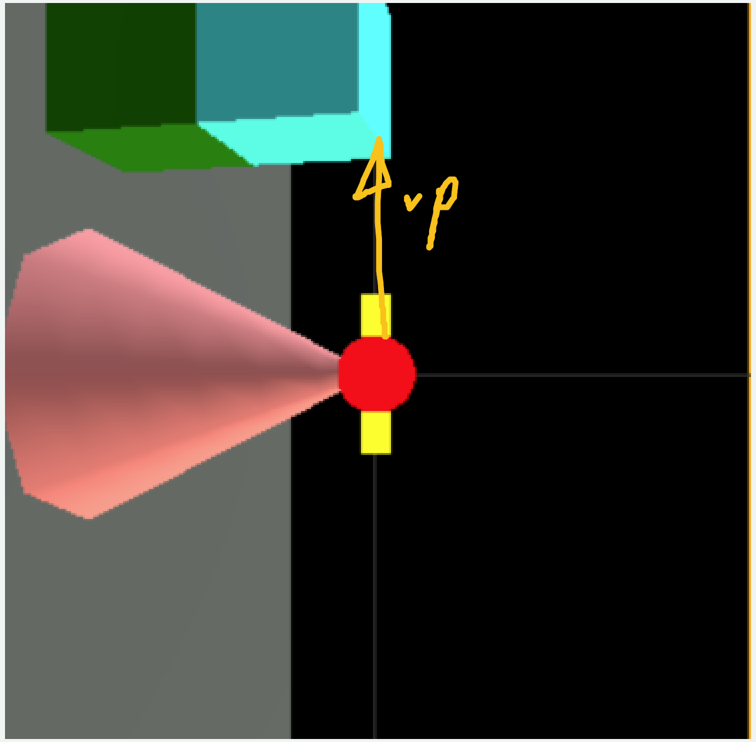
From the Demo (change Up)



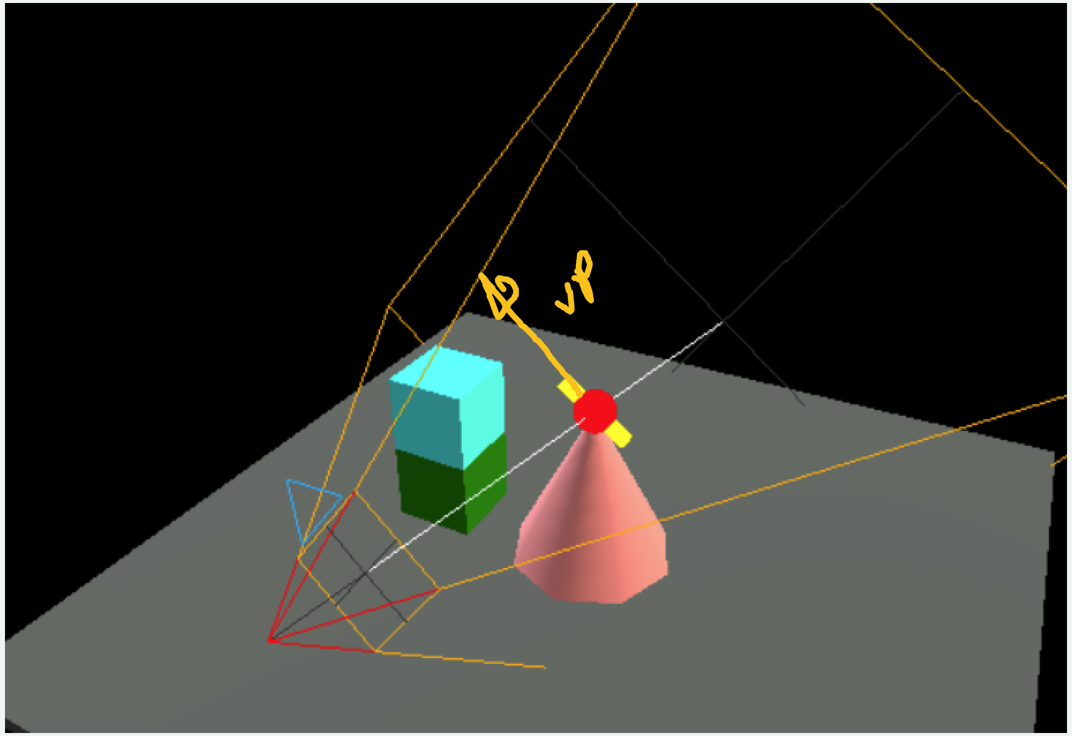


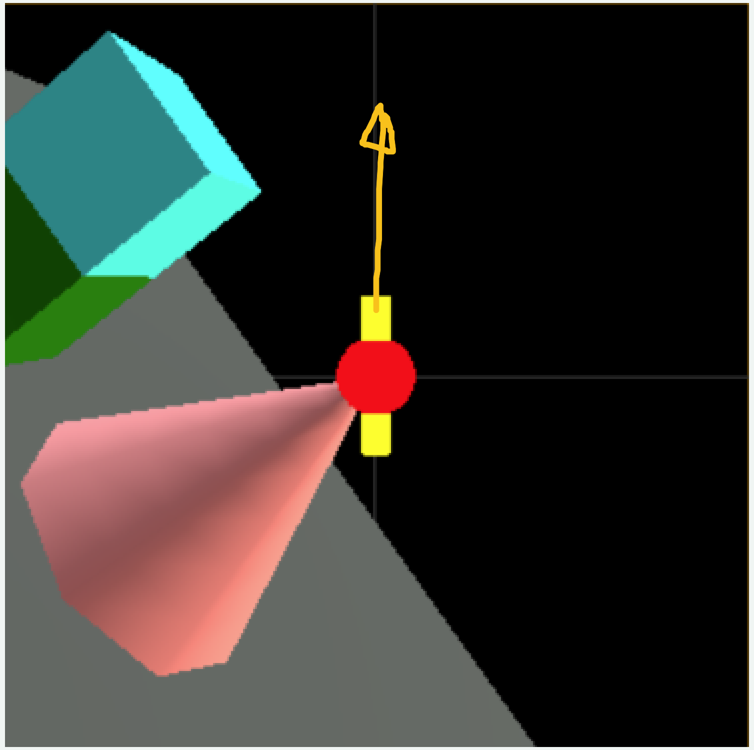
From the Demo (change LookAt)





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

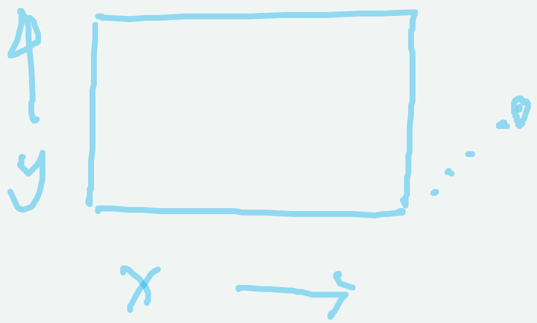
```
camera.position.set(fromX,fromY,fromZ); // normal translation/position
camera.up.set(upX,upY,upZ);   // this is a member variable
camera.lookat(atX,atY,atZ);   // uses the above two things

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

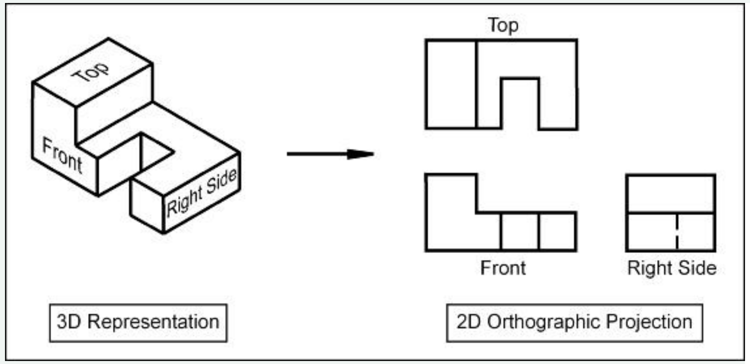
We lose a dimension

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

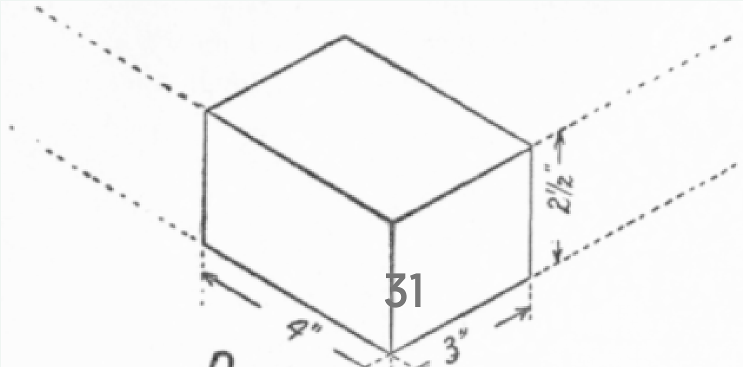


Types of Projections

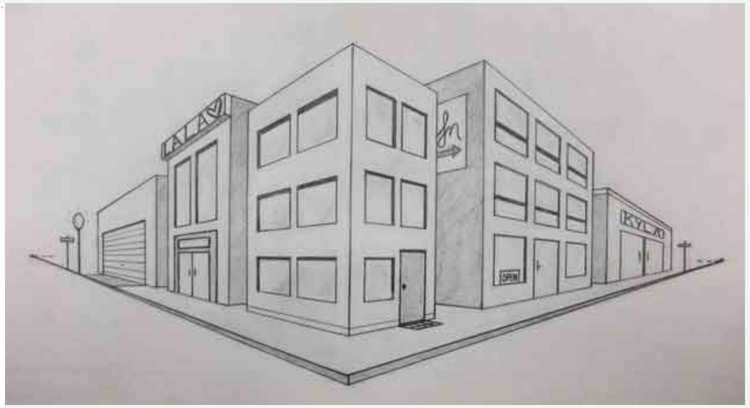
Orthographic



Isometric

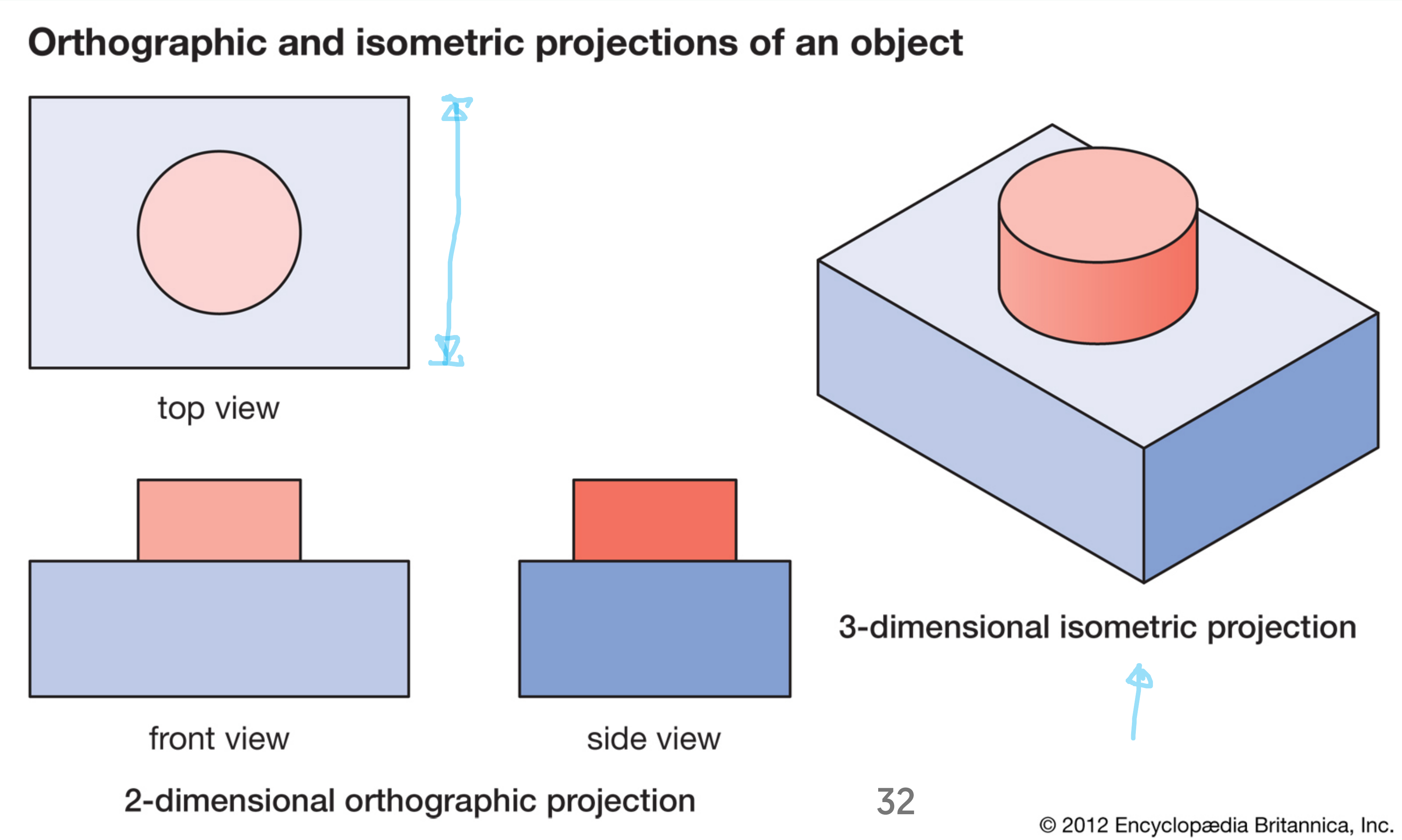


Perspective



Mechanical Drawing Projections

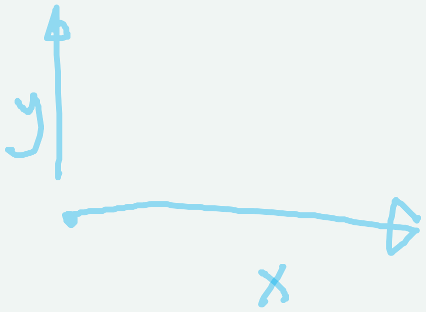
Don't get small when far



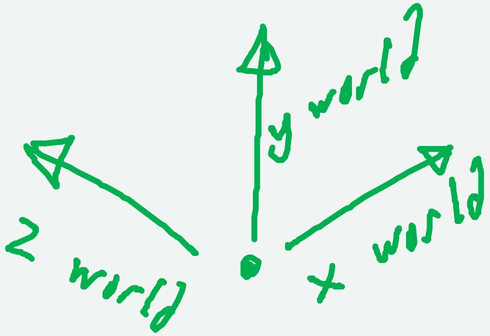
Types of Projections

Orthographic

3 axes ≡
stay perpendicular



Isometric



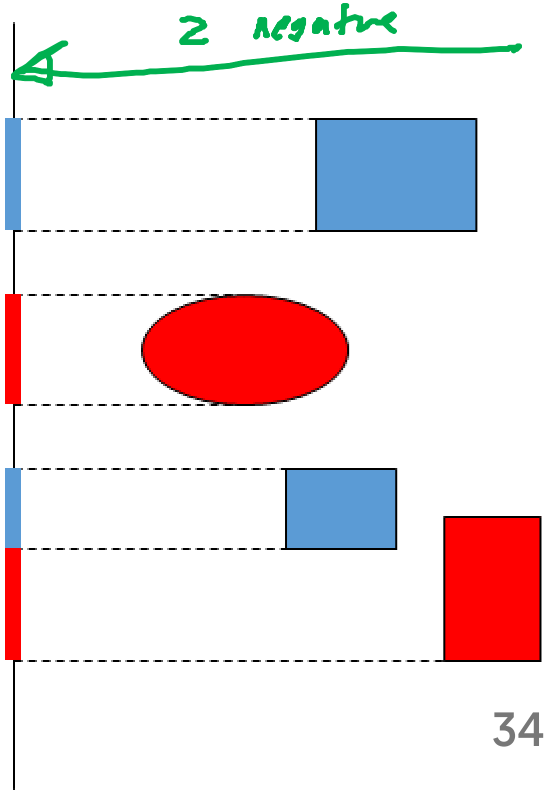
Perspective

things get smaller

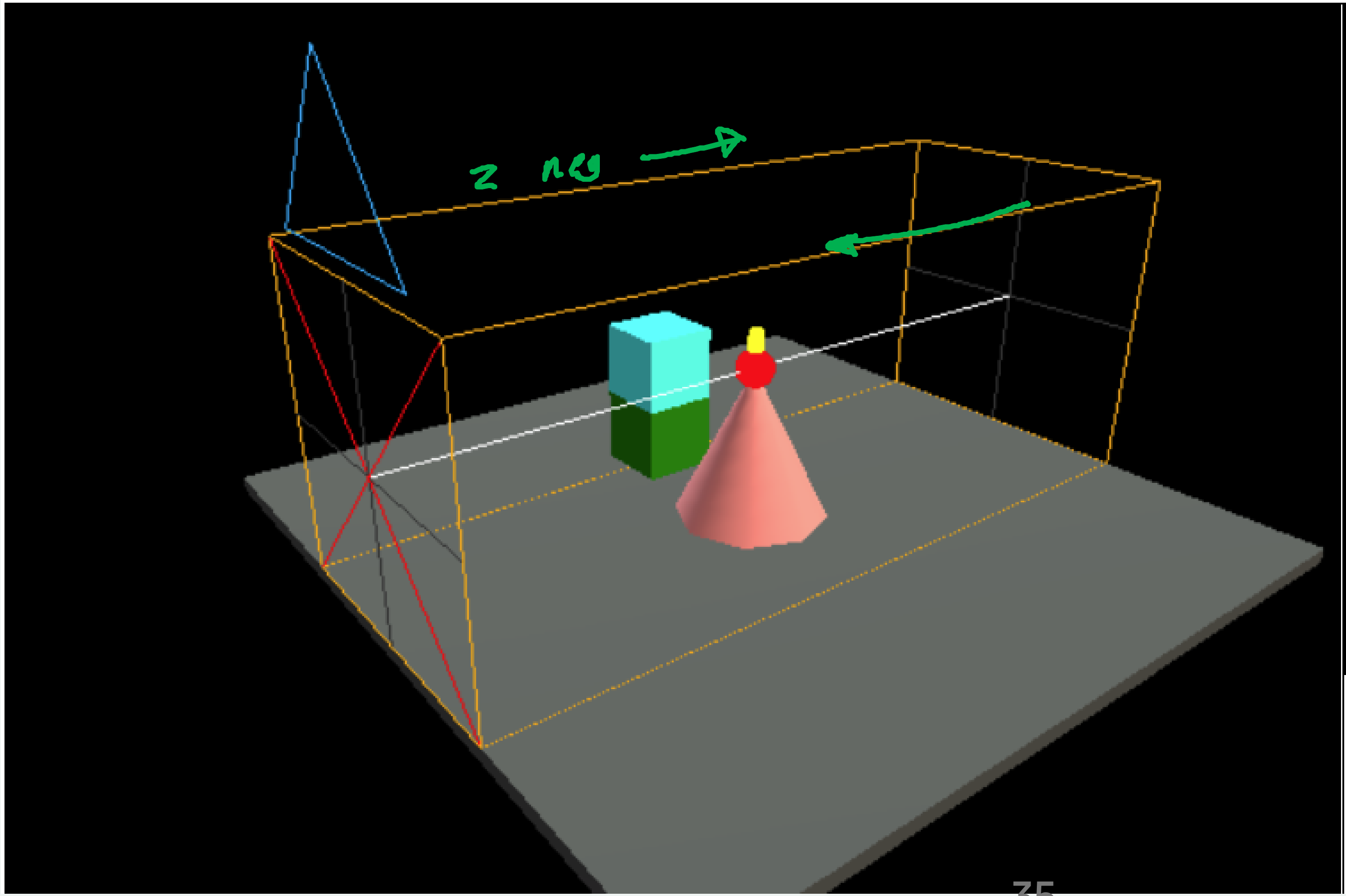
Orthographic Projection

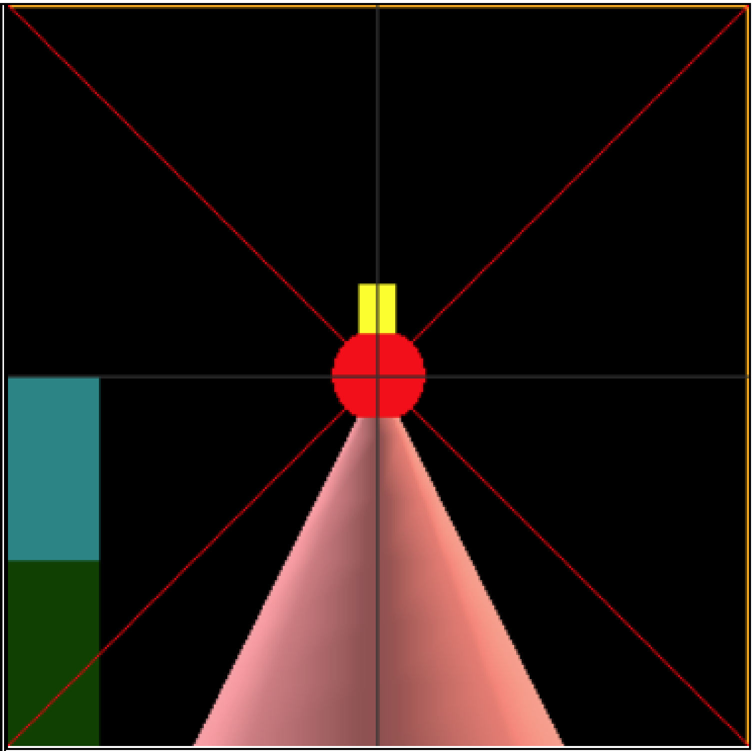
Projection = transformation that reduces dimension

Orthographic = flatten the world onto the film plane



The Orthographic "Box"



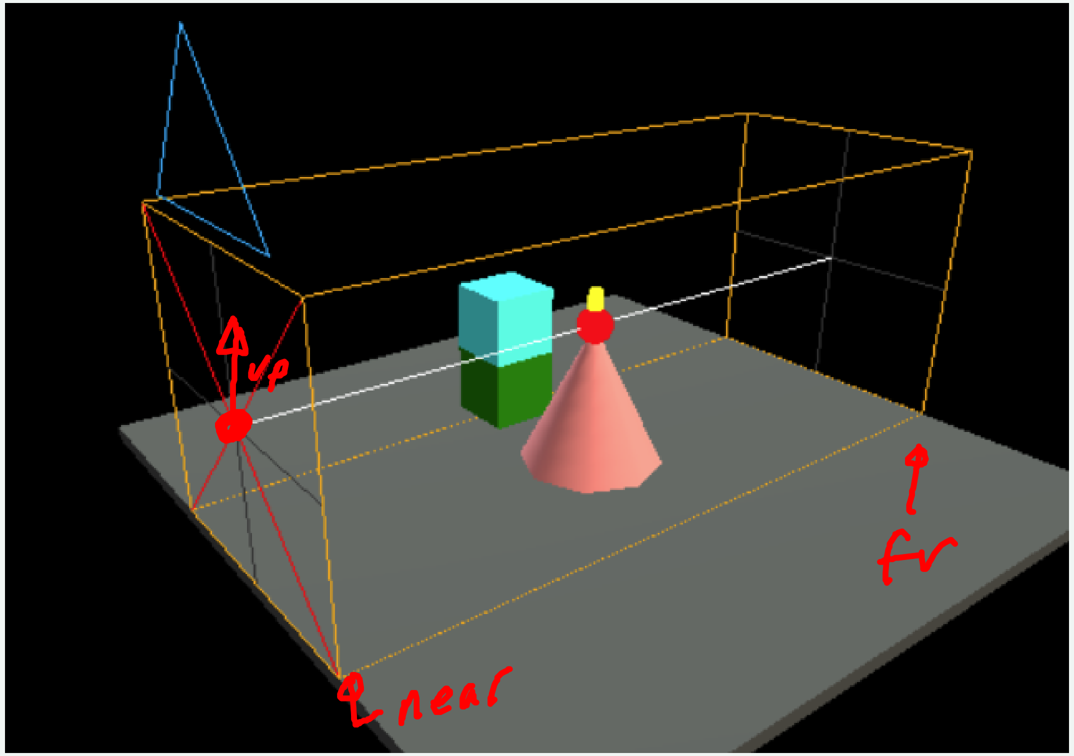


The Orthographic "Box"

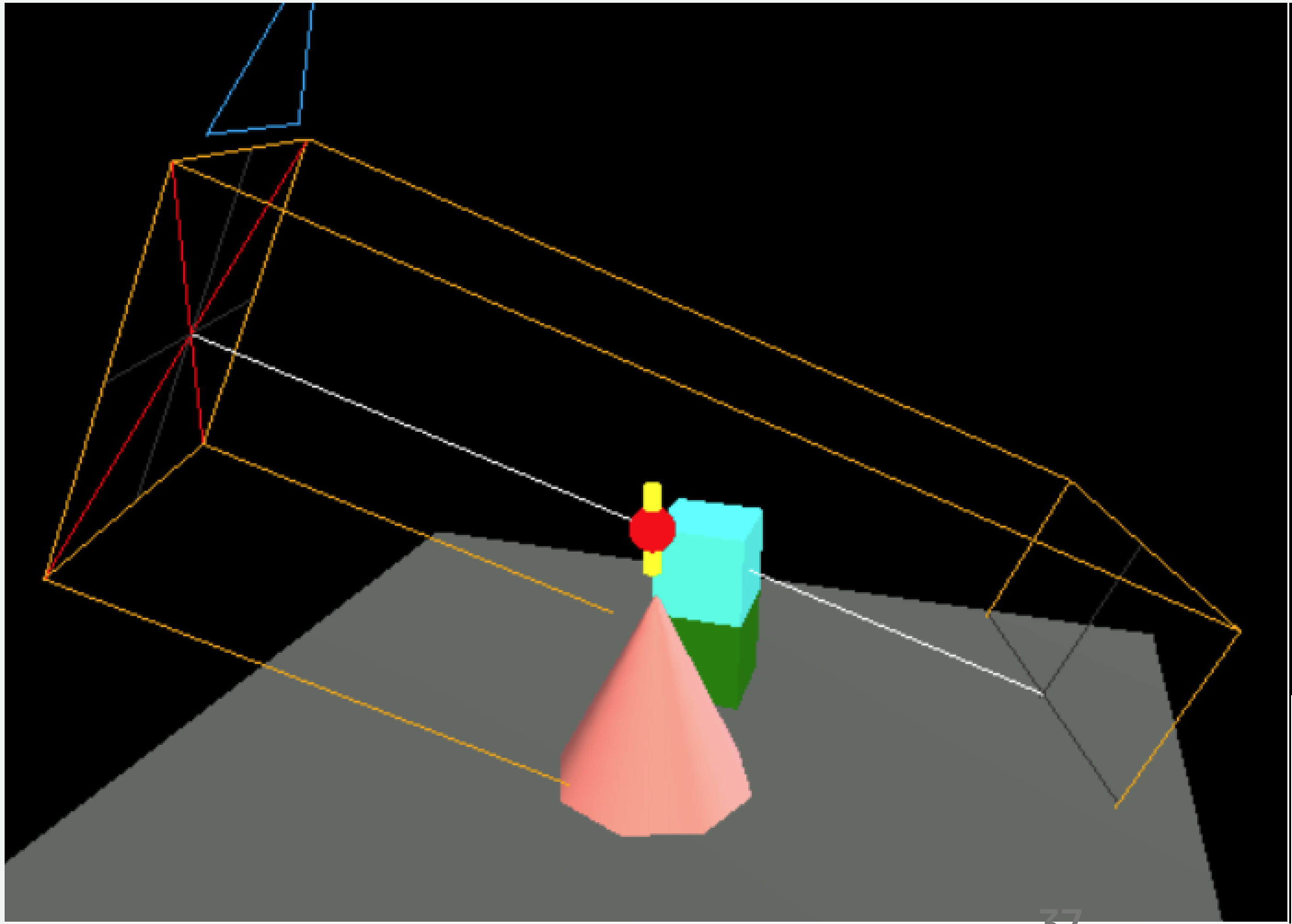
It is a "Camera Object"

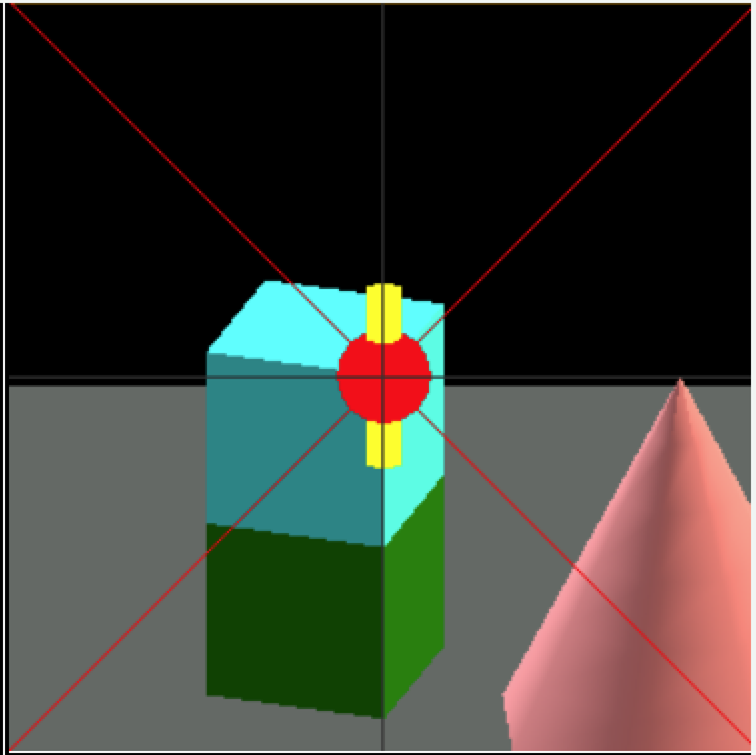
It is a Box in the World

- 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)

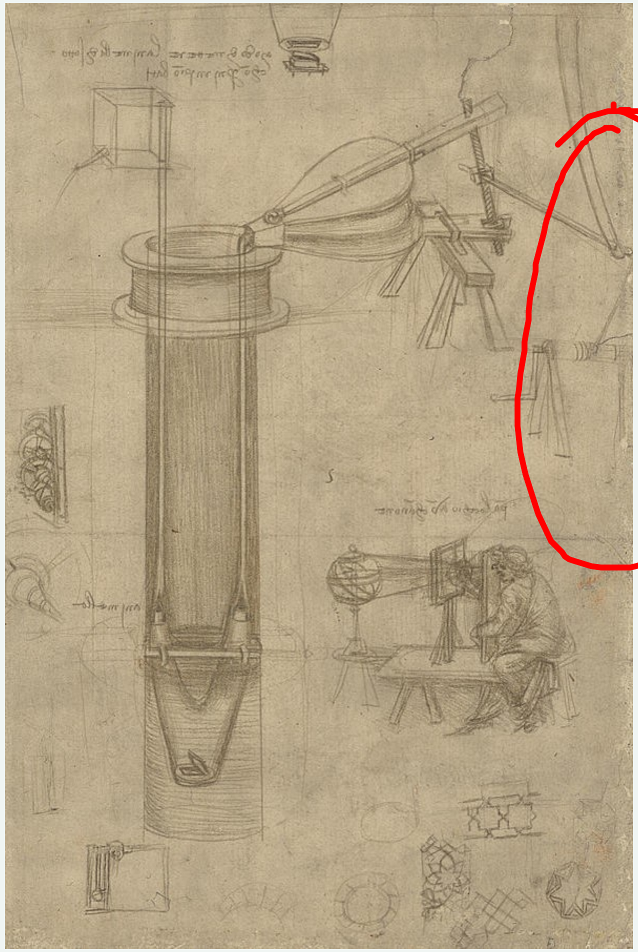
Shift and scale to fit

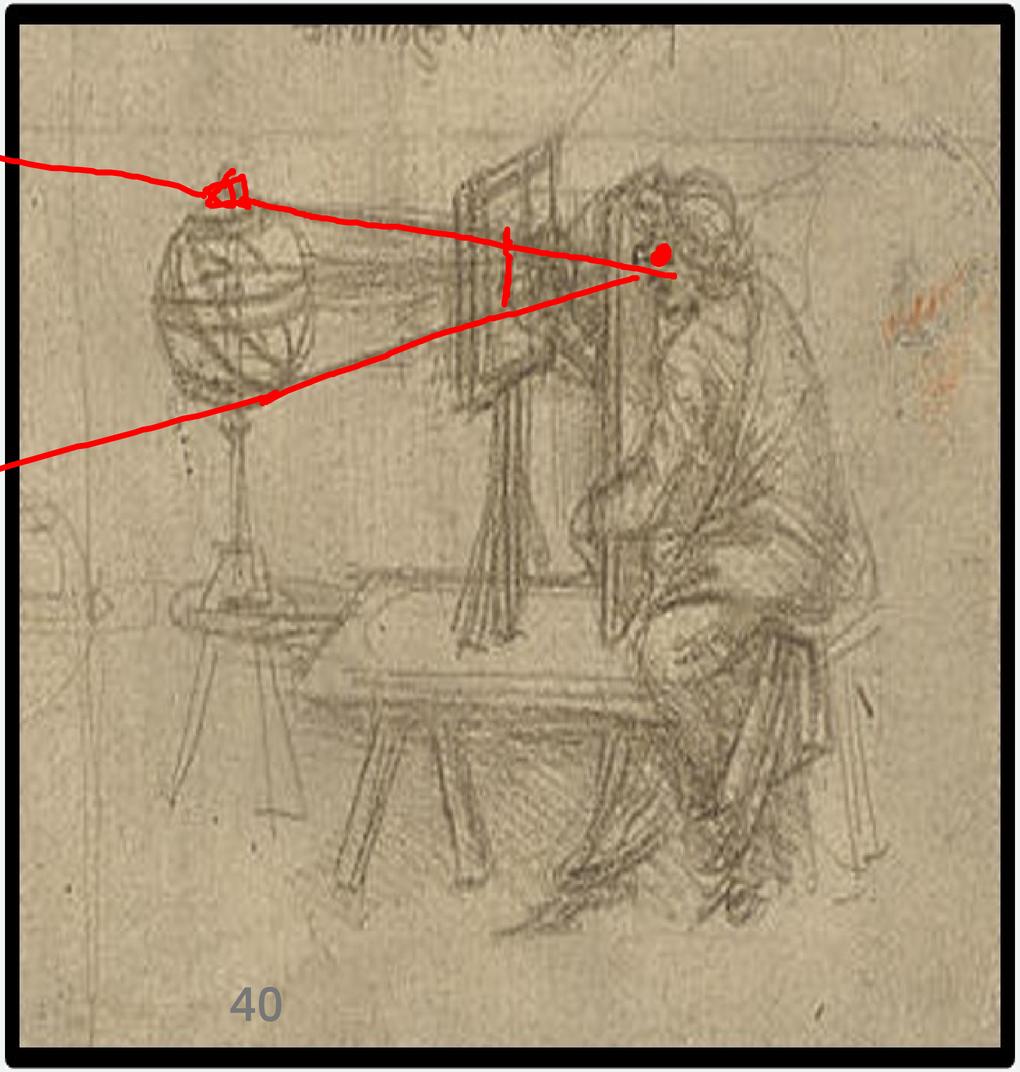
Rotations to get top, side, front

The need to scale in Z

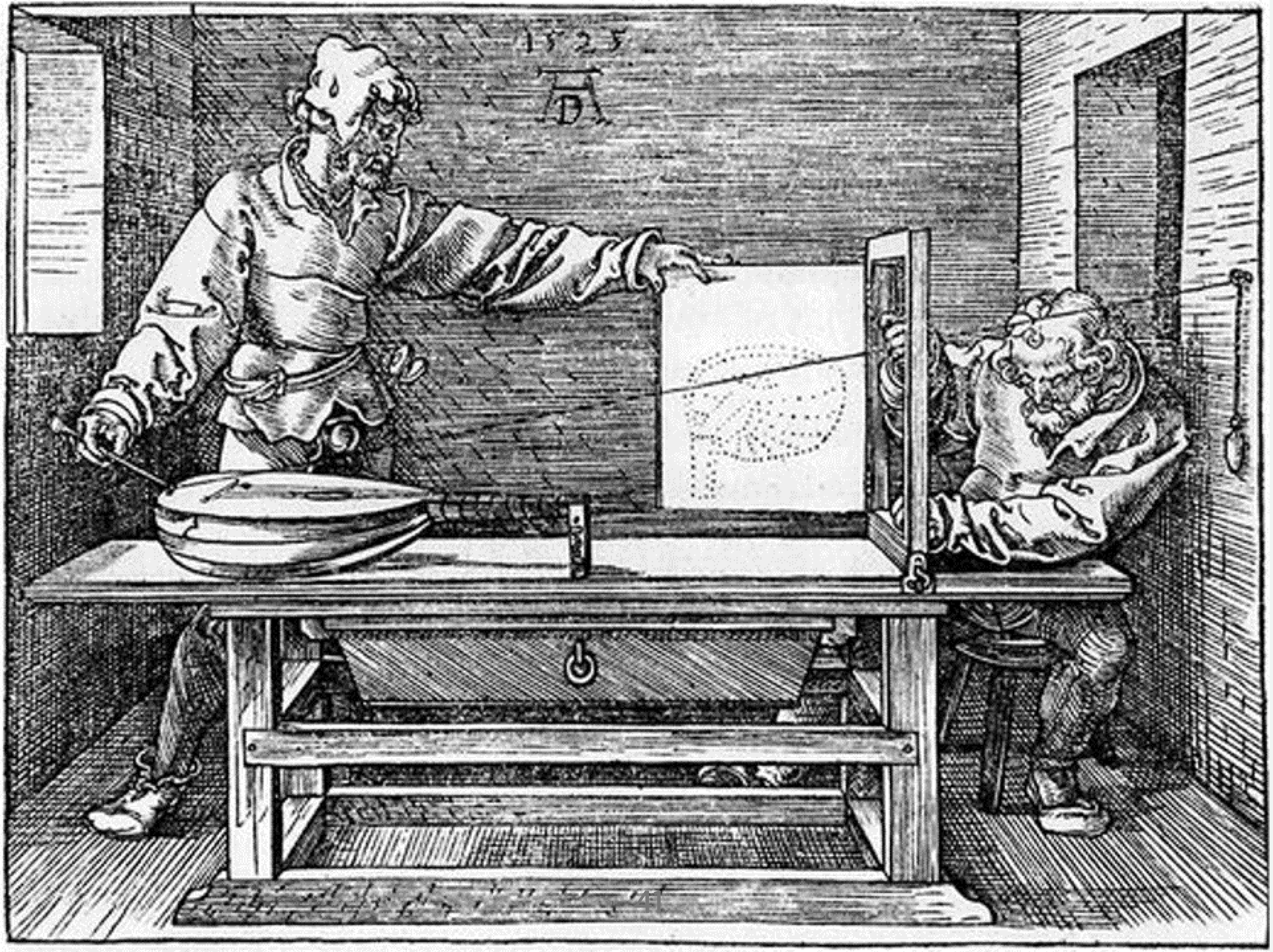
Perspective

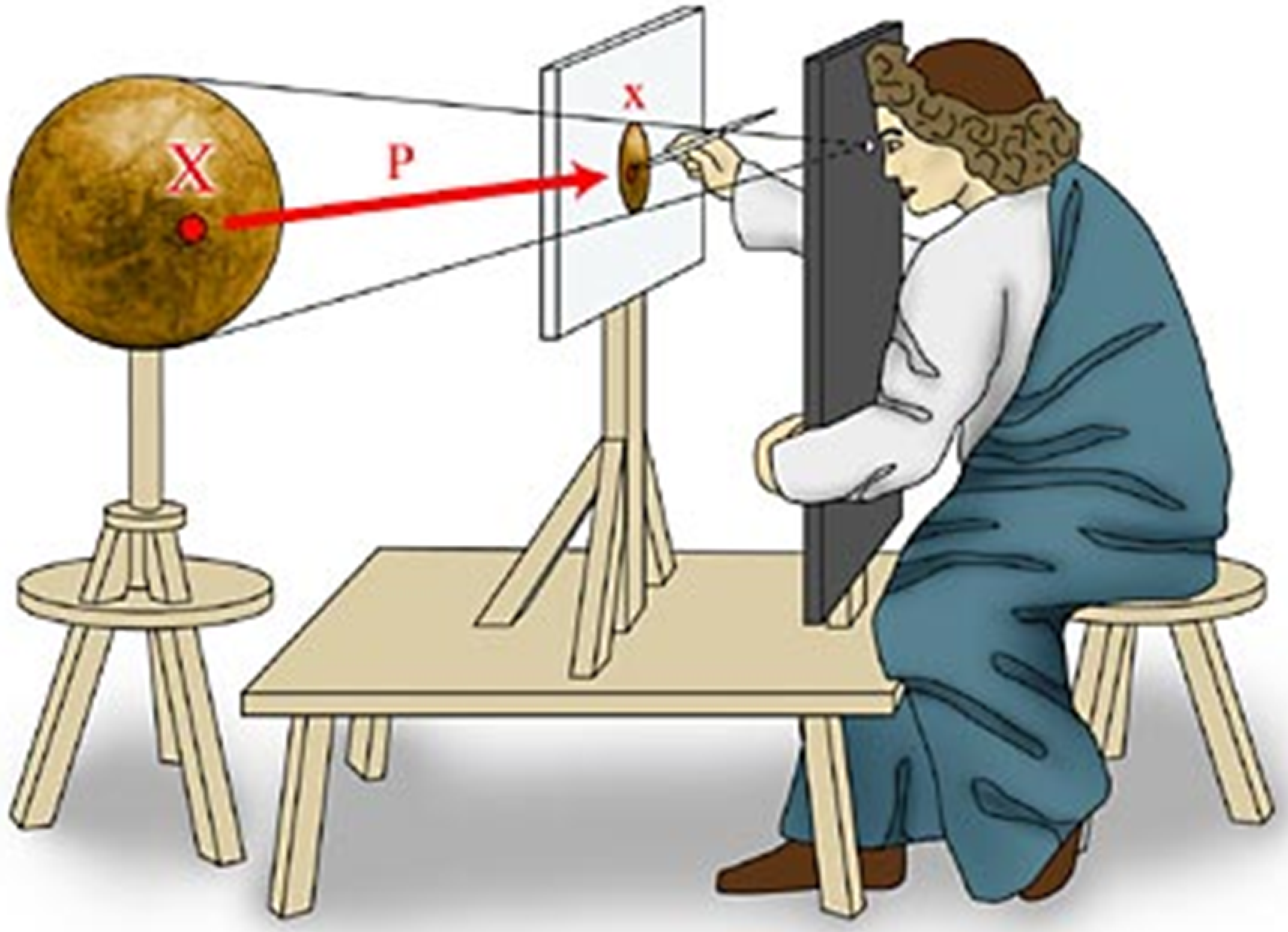
Do it like Da Vinci!





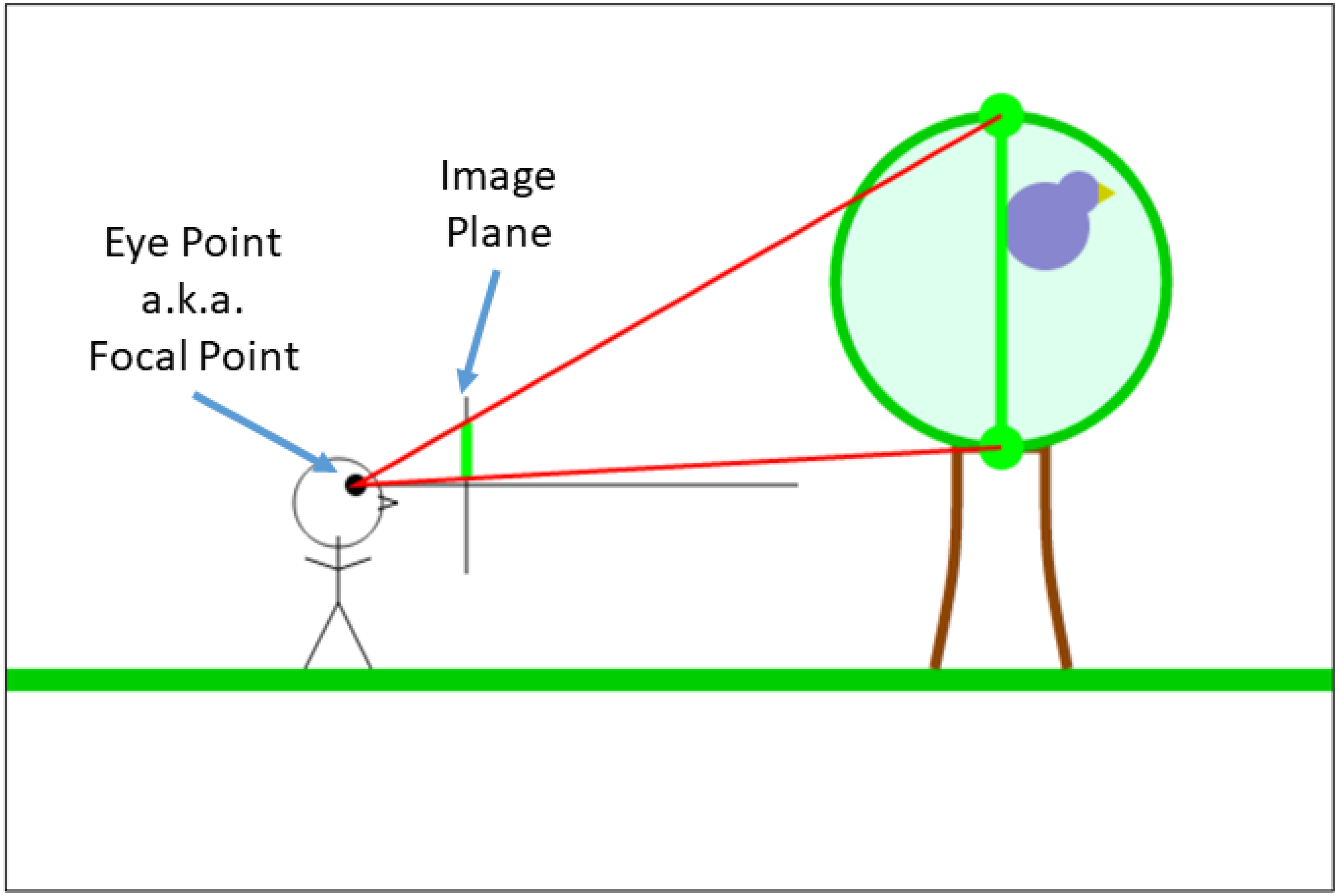
http://fineartamerica.com/featured/bellows-perspectograph-with-man-examining-inside-from-atlantic-codex-leonardo-da-


1525



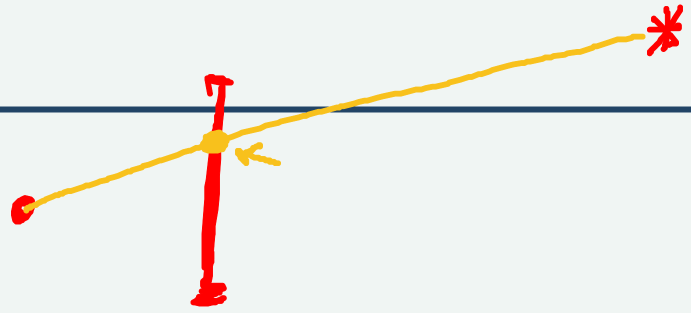
http://hans.wyrdweb.eu/about-perspective/

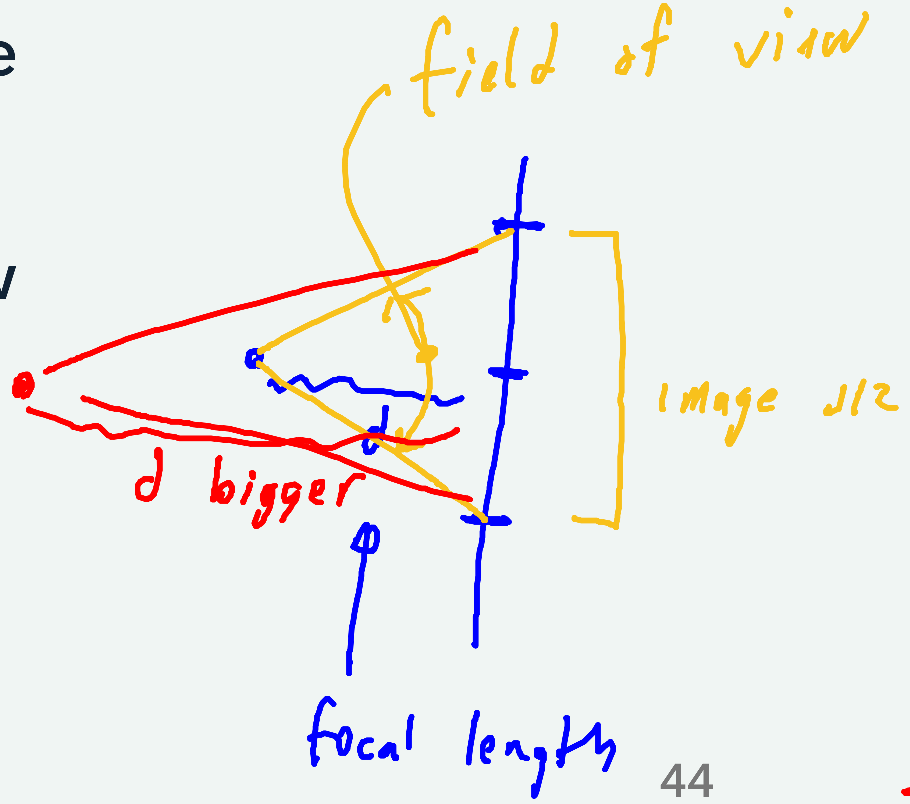
Perspective Imaging

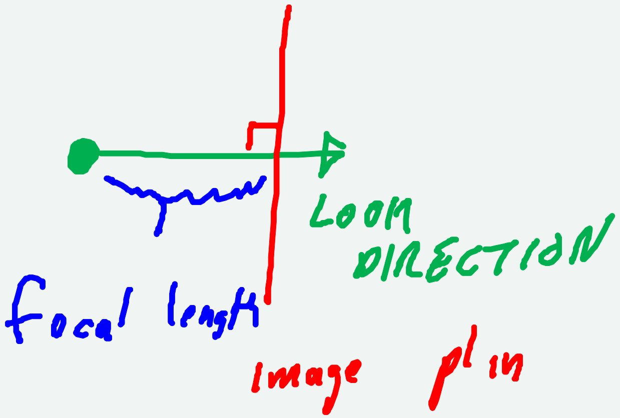


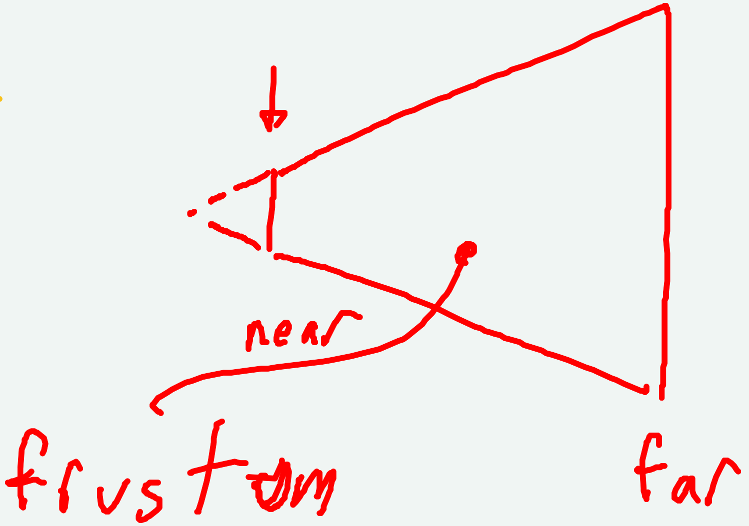
The intuitions

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









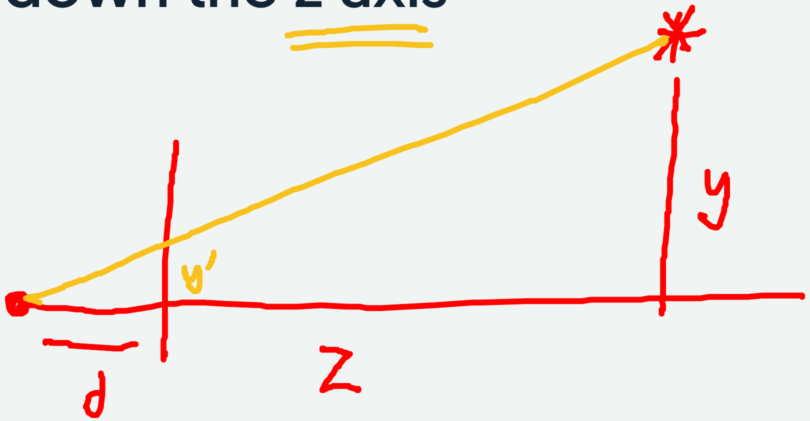
Field of View vs. Focal Length

- 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



$$\frac{y}{z} = \frac{y'}{d}$$

Linear?

$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

or

$$x_p = d\,x$$
$$y_p = d\,y$$
$$z_p = 1$$
$$w_p = z$$

Don't forget the divide by w!

Note what happens to z

$$x_s = \frac{x_p}{w_p} = \frac{dx}{z}$$

$$z_s = \frac{z_p}{w_p} = \frac{1}{z}$$

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

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

n - near plane distance

f - far plane distance

It's just a transformation!

Just like any other linear transformation

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.