Lecture 13:
More 3D

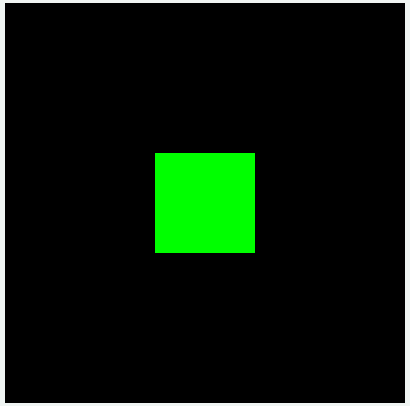
What were we really doing last time?

Get to THREE Transformations

(practice for exam)

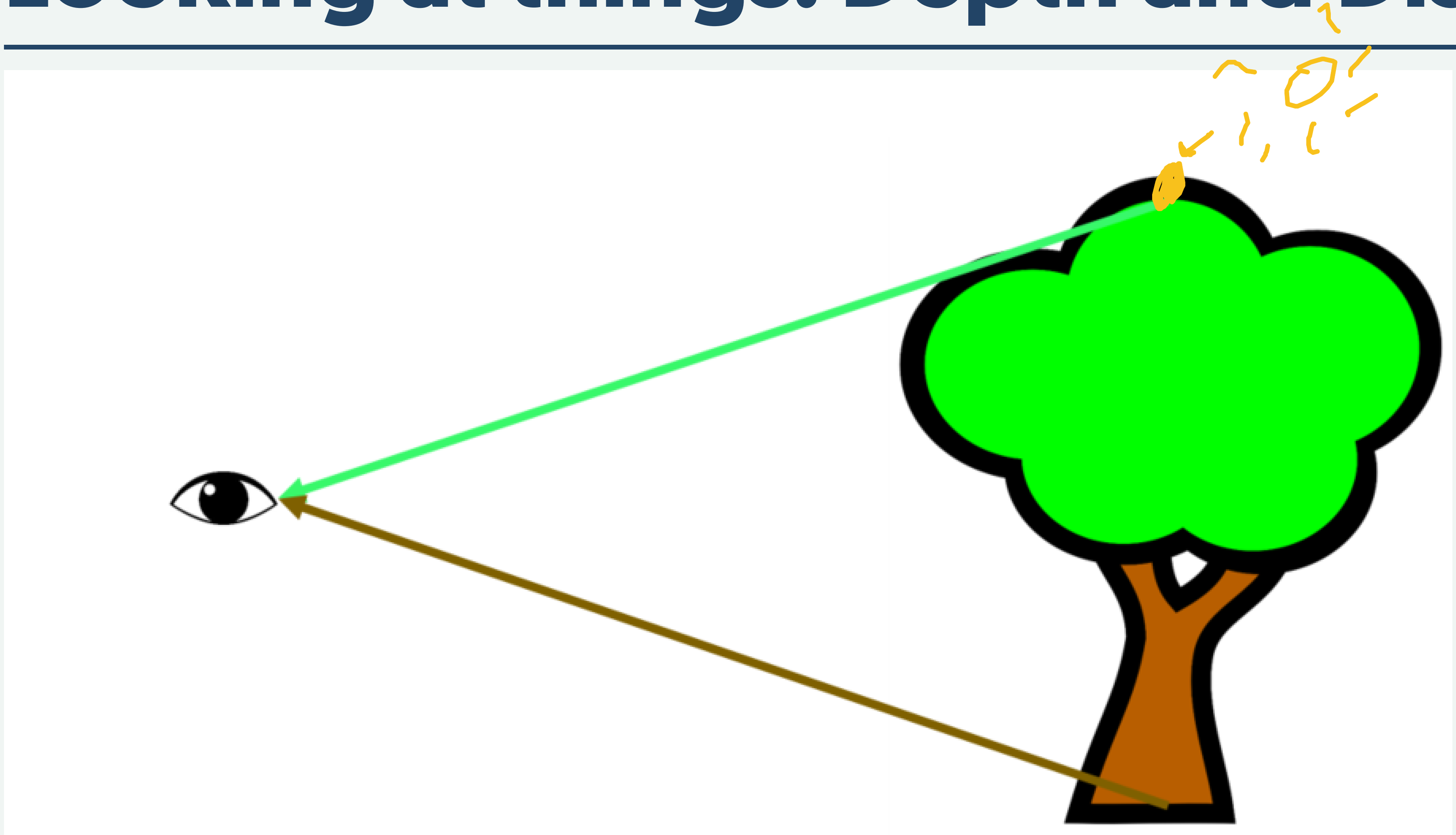
All Together

```
let renderer = new T.WebGLRenderer(); /*1*/
renderer.setSize(200,200);
document.body.appendChild(renderer.domElement);
let scene = new T.Scene(); /*2*/
let geometry = new T.BoxGeometry(1,1,1); /*3*/
var material = new T.MeshBasicMaterial( { color: 0x00ff00 } ); /*4*/
let mesh = new T.Mesh(geometry, material); /*5*/
scene.add(mesh); /*5*/
let camera = new T.OrthographicCamera(-2,2, -2,2, -2,2); /*6*/
renderer.render( scene, camera ); /*7*/
```

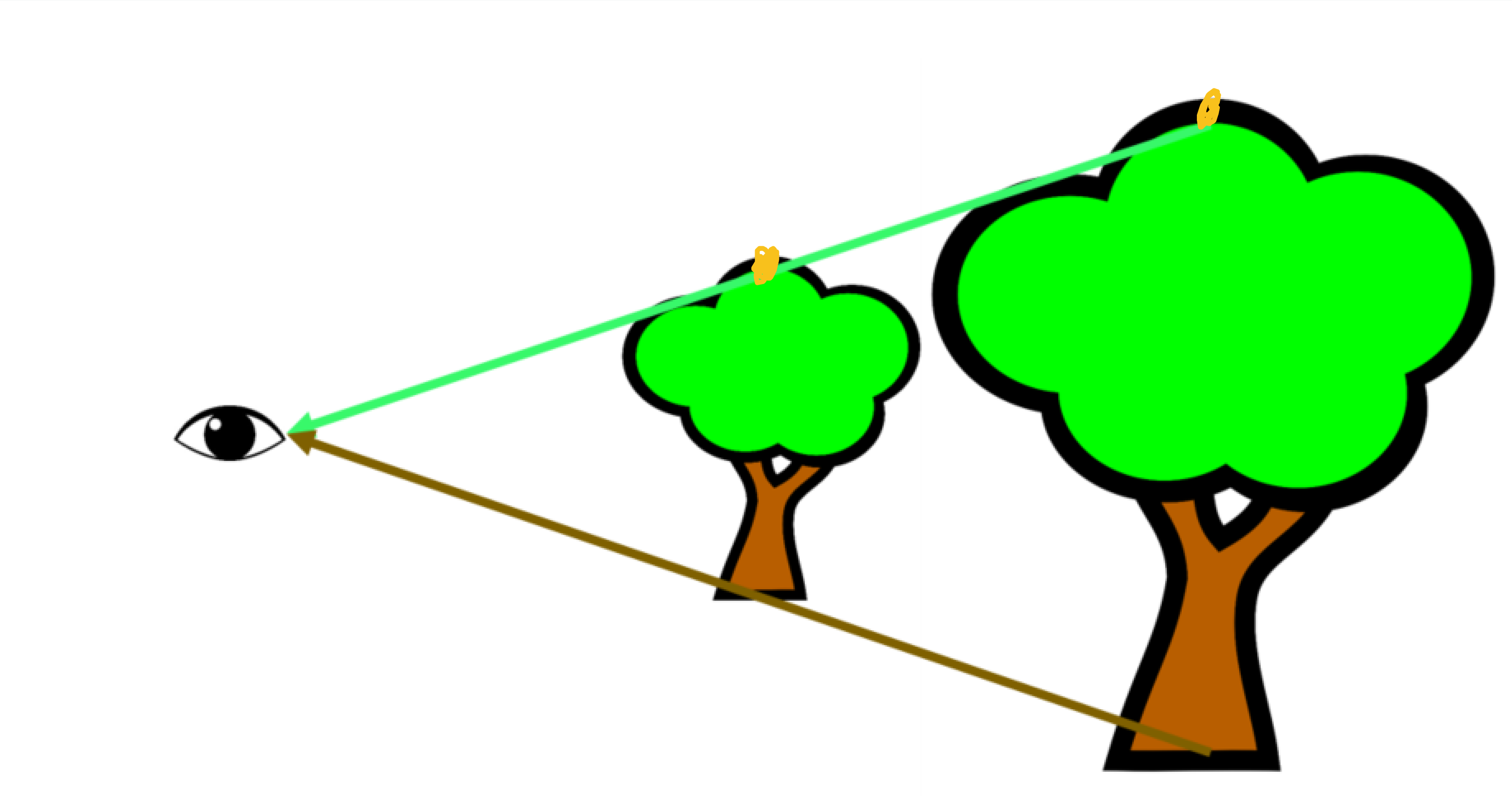


1. Create the Canvas and Set up
2. Create the World
3. Create the Cube
4. Give it a Material (how it should look)
5. Put the Cube into the World
6. Make a Camera (transform 3D to 2D)
7. Draw

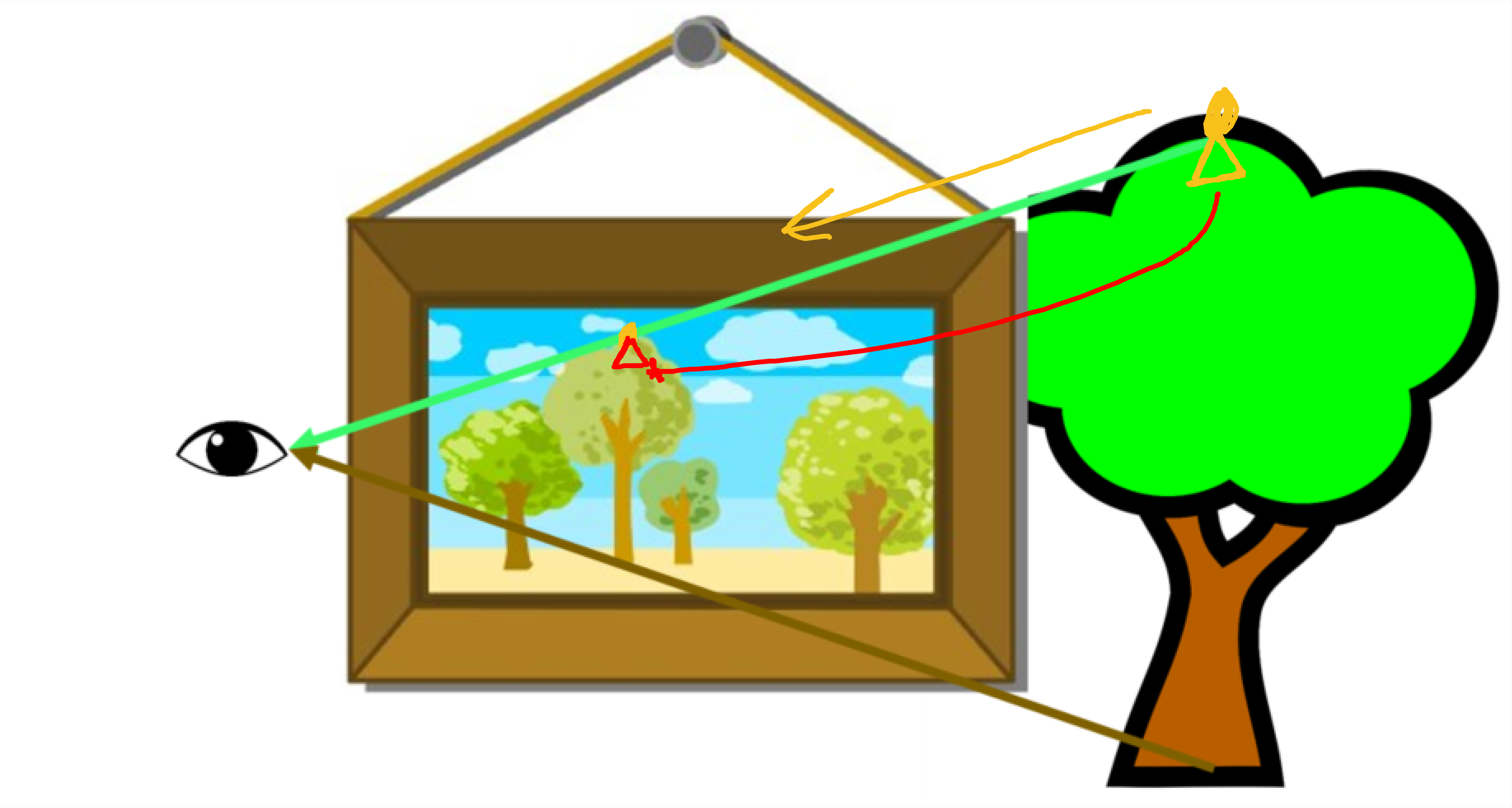
Looking at things: Depth and Distance



Looking at things: Depth and Distance



Looking at things: Depth and Distance



We sense 2D

(actually, a little more than that)

We infer 3D

Sensing 3D (we'll come back to this)

One eye:

- Accomodation

Two eyes:

- Vergence
- Disparity

Many eyes: (multiple times)

- Parallax
- Depth from Motion

3D Cues from One image

Occlusion

Perspective

Familiar Size

Relative Size

Lighting (shading)

Lighting (reflections/shadows)

Texture/Pattern

Horizon Elevation

Long Distance Shifts

SNCF

What makes an image look 3D?

Occlusion

Perspective

Familiar Size

Relative Size

Lighting (shading)

Lighting (reflections/shadows)

Texture/Pattern

Horizon Elevation

Long Distance Shifts

OK - so how do we do that?

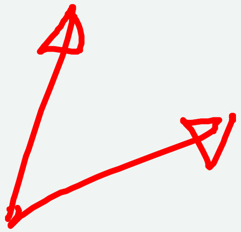
Some 3D Math

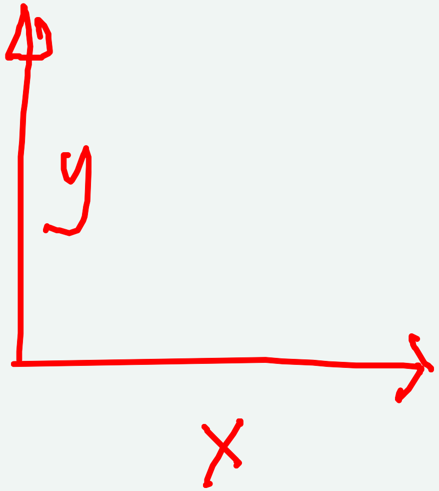
Coordinate systems

Right hand rule

Cross-Product

X Y Z

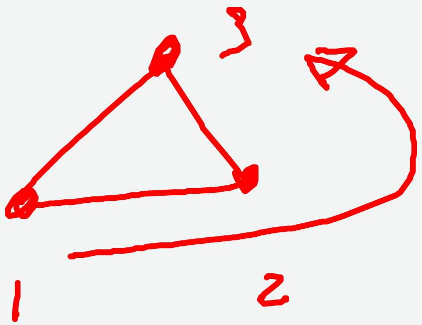


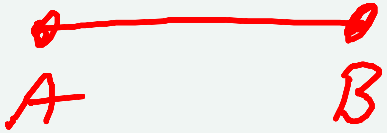


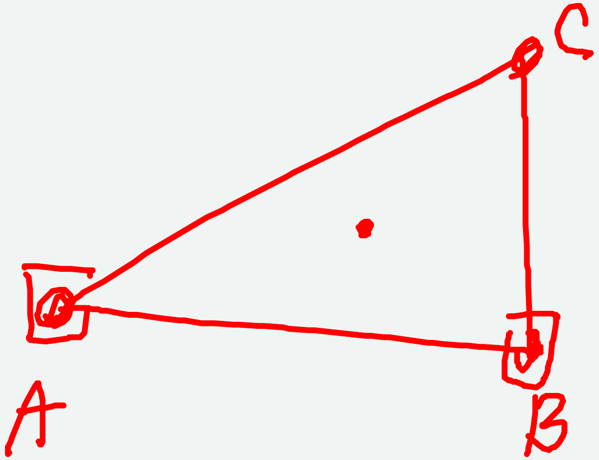
Triangles

Normals

Barycentric Interpolation







$$\underset{\uparrow}{\alpha} A + \underset{\uparrow}{\beta} B + \underset{\uparrow}{\gamma} C =$$

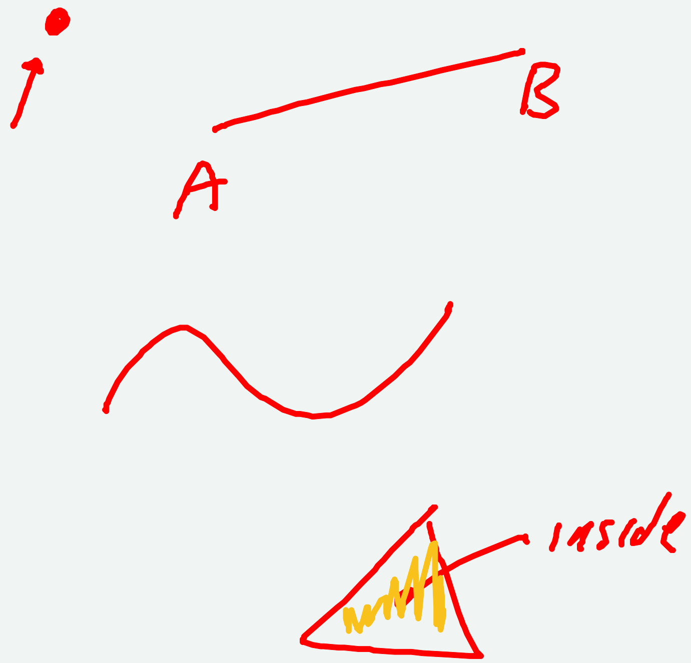
Curves vs. Surfaces vs. Volumes

A Point is 0D (just a point) - can be 2D, 3D, ...

A Curve is 1D (length) - can be 2D, 3D, ...

A Surface is 2D (area) - can be 2D, 3D, ...

A Volume is 3D (solid) - can be 3D, ...



Not all curves are the boundaries of areas

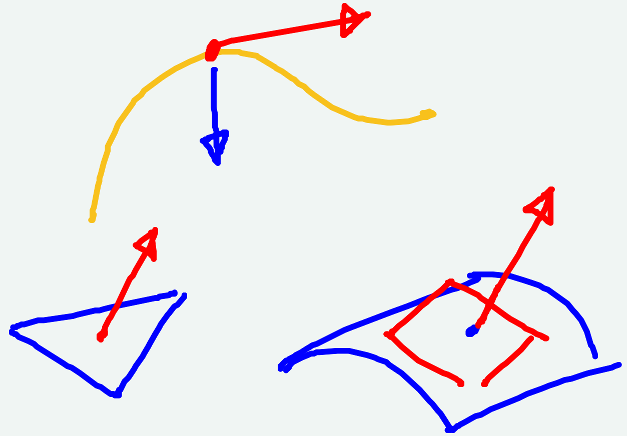
Not all surfaces are the boundaries of solids

Normals and Tangents

In 3D, the tangent to a **surface** (at a point) is a plane

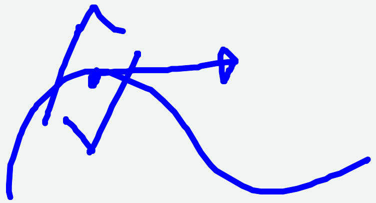
In 3D, the normal to a **surface** (at a point) is a vector

For a triangle, it is constant over the whole shape



In 3D, the tangent to a **curve** is a vector

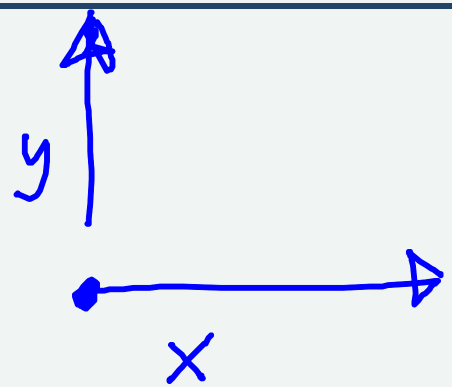
In 3D, the normal to a **curve** is a plane (defined by normal vectors)



Coordinates in 3D

X, Y and Z axes

- right hand rule
- conventions on how we use them
- I prefer "Y is up" (direction of gravity)
- exceptions for what makes sense
    - book example



Coordinate Systems in 3D

- Book
- Table
- Room
- House
- City
- GPS (world)

- Screen
- Camera

Need some coordinate system with object and camera

Cover on Book

Book on Table

Table in Room

Camera in Room

Camera's picture on Screen

"Scene" Coordinates

A Coordinate System that has the objects and camera

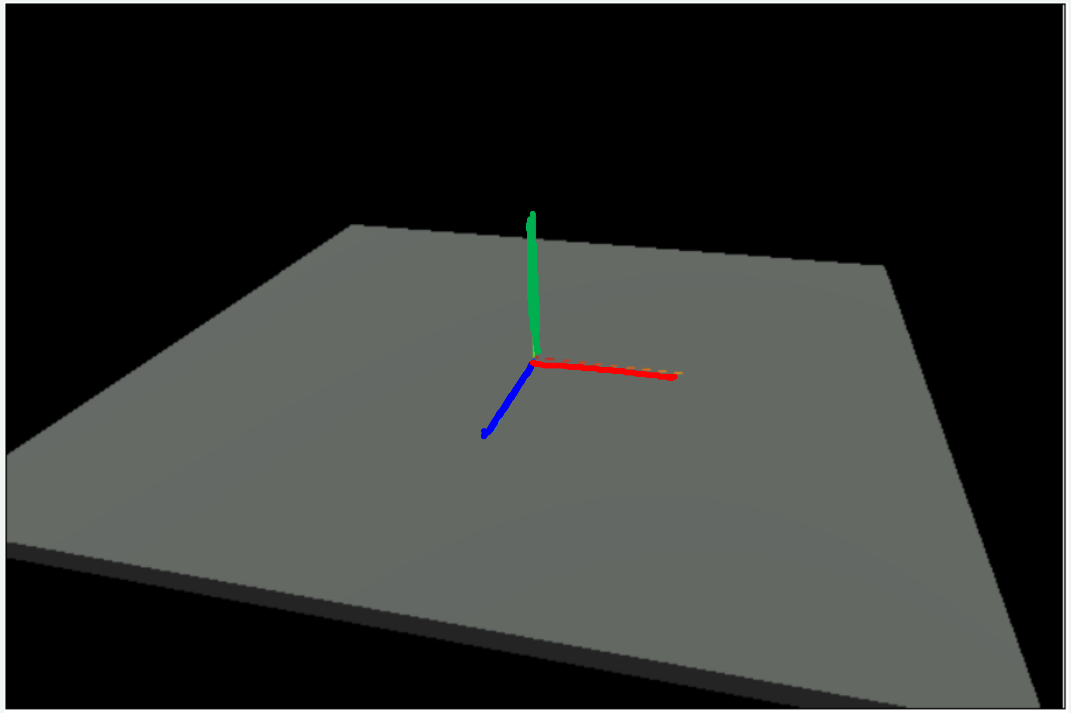
A Simple Example

Warning: the code is "fake" - because the actual code is a mix of THREE and "class Framework code"

A Simple Example

Scene

The class framework (just wait) makes a "ground plane" - a big flat object centered at the origin of the scene.



Cube in Scene

Cube in Scene

The center of the cube's coordinate system is the center of the cube

This is THREE's default

```
let cube = new T.Cube();
cube.translateY(1);
scene.add(cube);
/* cube.position.set(0,1,0); */
```

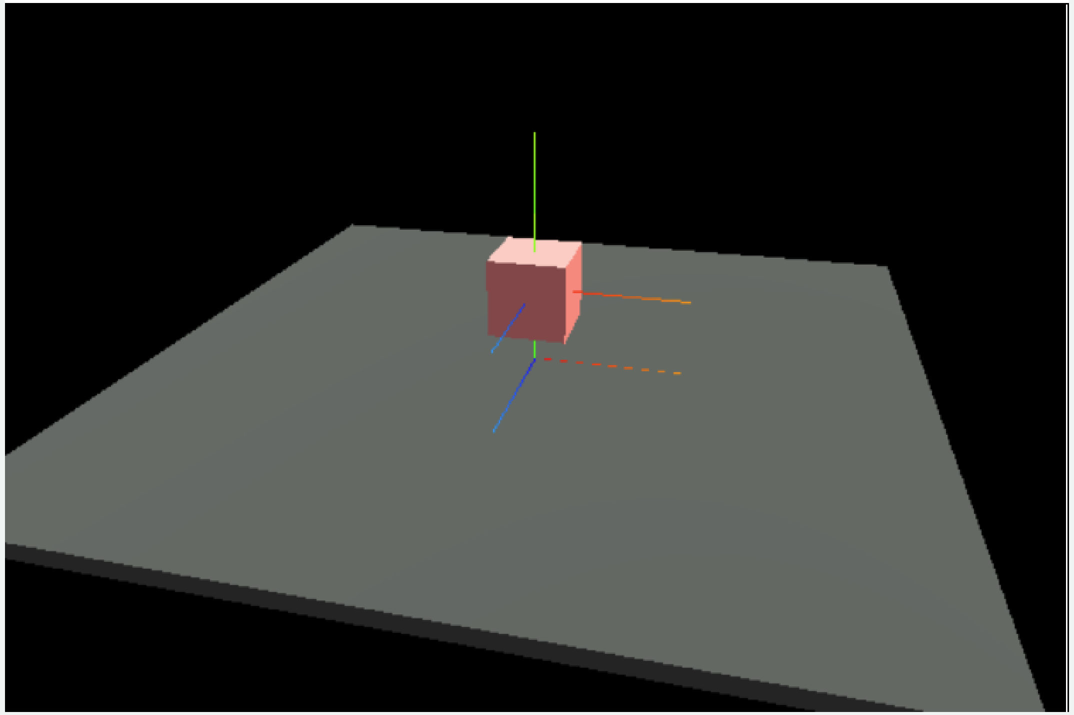


Table in Scene

Table in Scene

Note: I made the table

I defined the table to have its origin at the corner of the table top.

```
let table = new Table();
scene.add(table);
table.position.set(0,3,0);
```

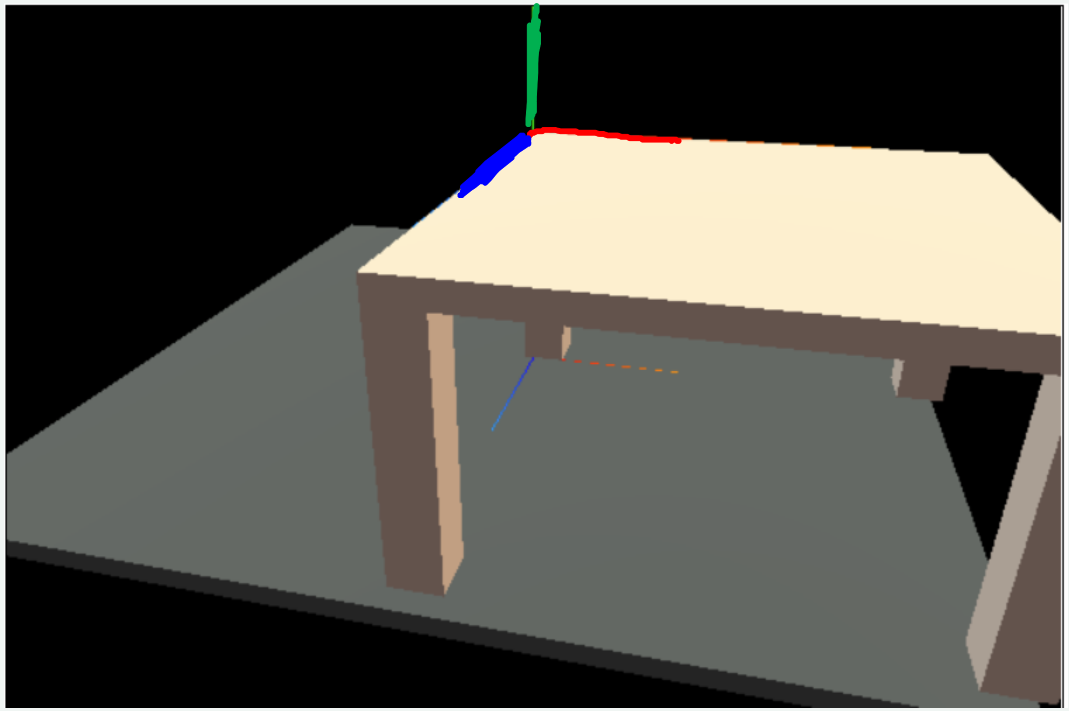


Table in Scene

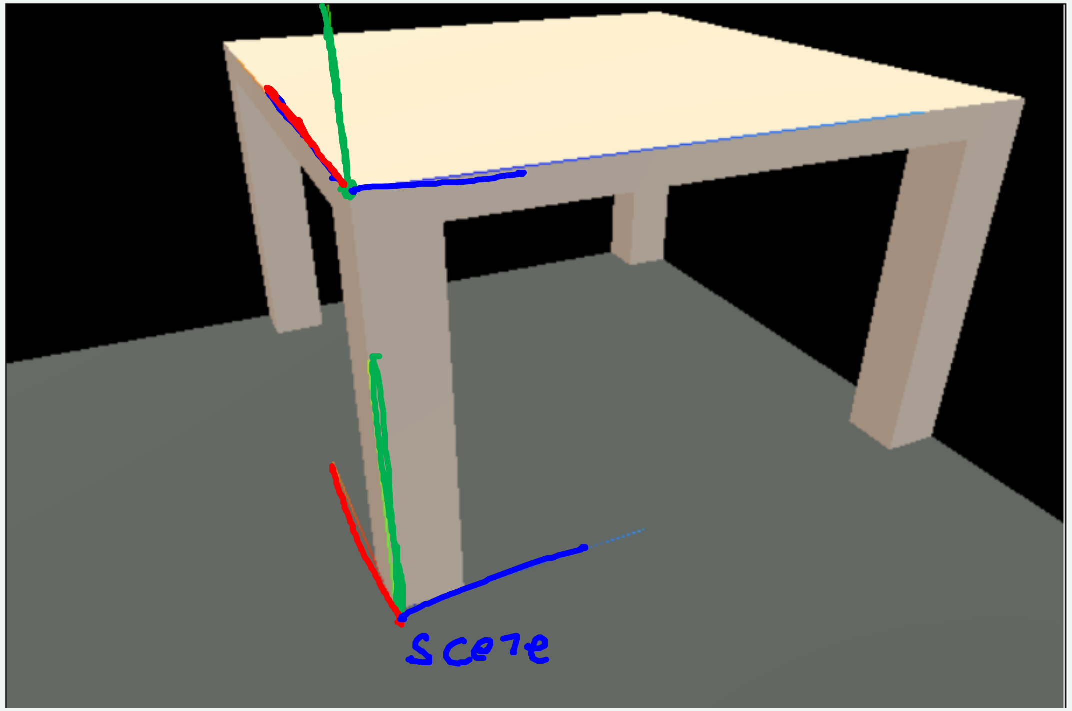
Table in Scene

I defined the table to have its origin at the corner of the table top.

I need to position the table above the floor.

(transform to change its coordinates)

```
table.position.set(0,3,0);
```



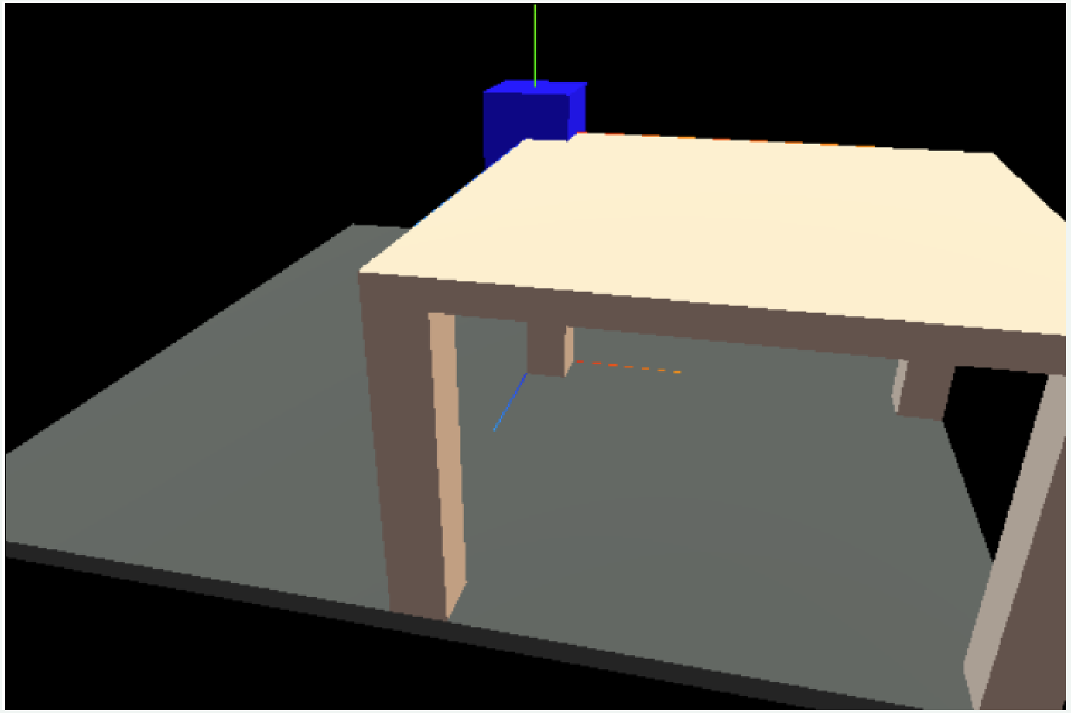
Cube on Table in Scene

Cube on Table in Scene

The cube's origin is at it's center.

If I place it "on the table" (position 0), it is actually inside the table.

```
let cube = new T.Cube();
table.add(cube);
```



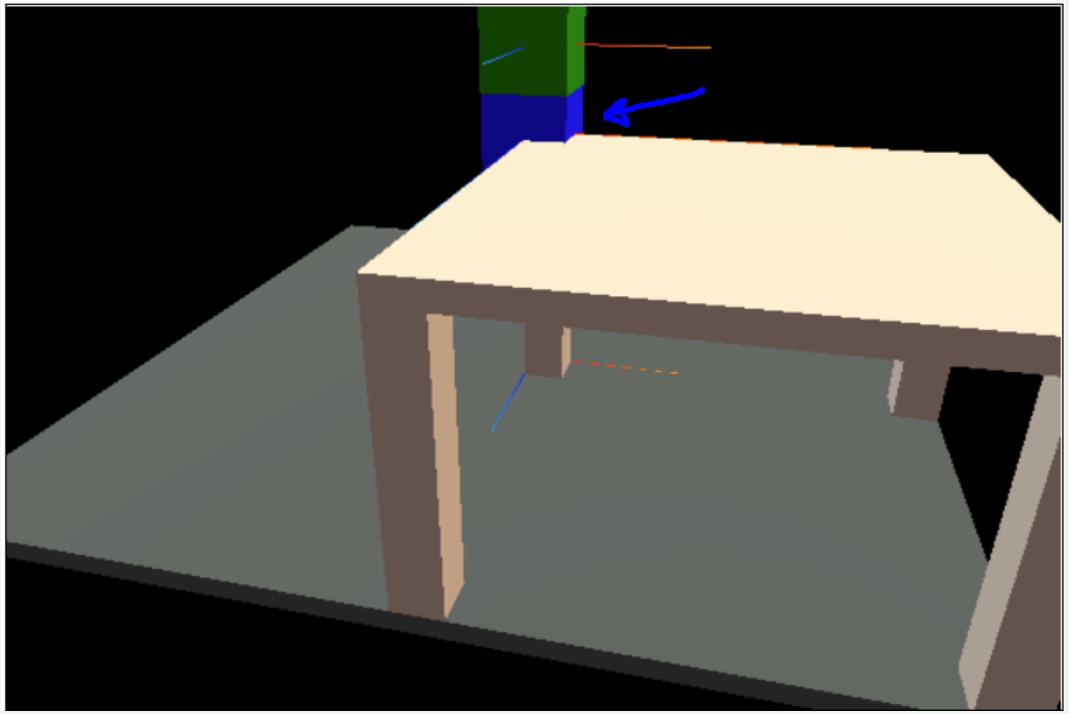
Cube on Cube on Table in Scene

Cube on Table in Scene

I transformed the second cube to be 1 unit upwards.

```
let cube1 = new T.Cube();
table.add(cube1);

let cube2 = new T.Cube();
cube1.add(cube2);
cube2.translateY(1);
/* cube2.position.set(0,1,0); */
```

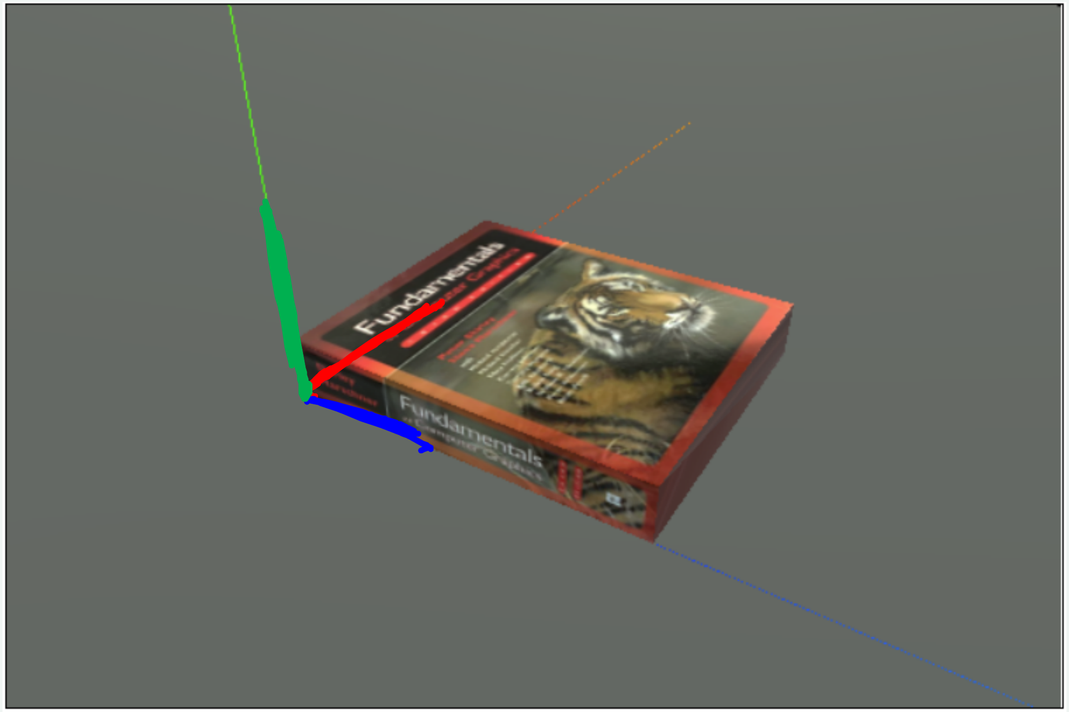


Book

Book (in Scene)

I made the book to have its origin at the bottom corner (at the "top" of the book)

```
let book = new Book();
scene.add(book);
```



Book on Table

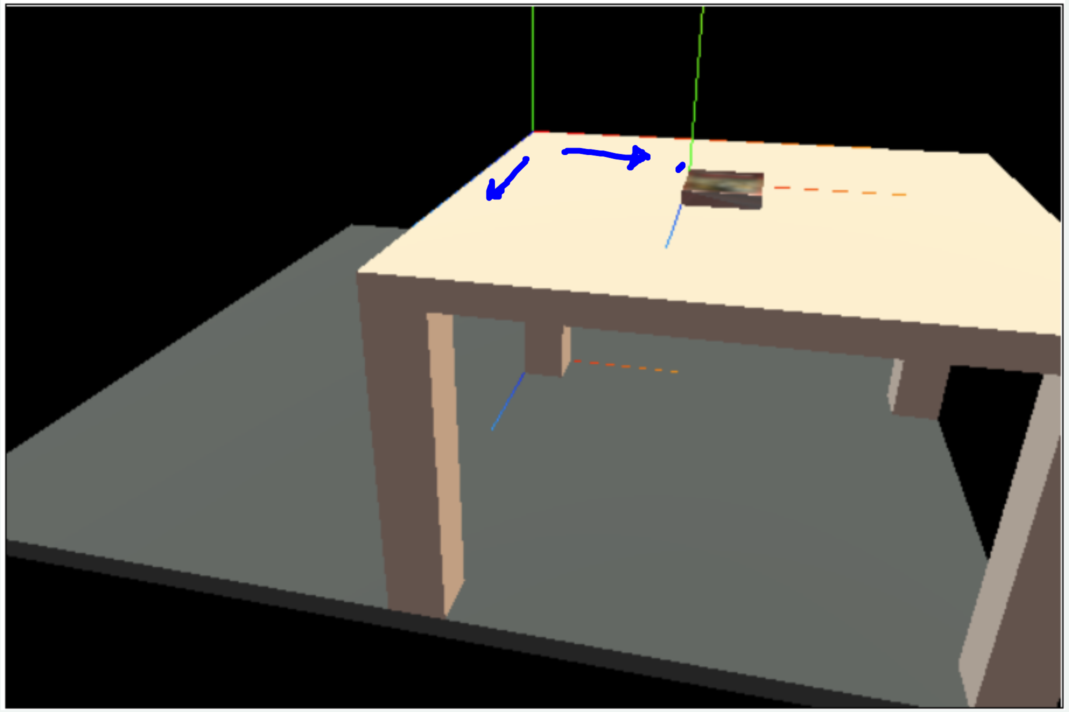
Book on Table (in Scene)

The book is at the origin.

The book is parented to the table

```
let table = new Table();
scene.add(table);
table.position.set(0,3,0);
let book = new Book();
table.add(book);
book.position.set(2,0,2);
```

x z



Camera Looks at Book on Table …

Camera in Scene

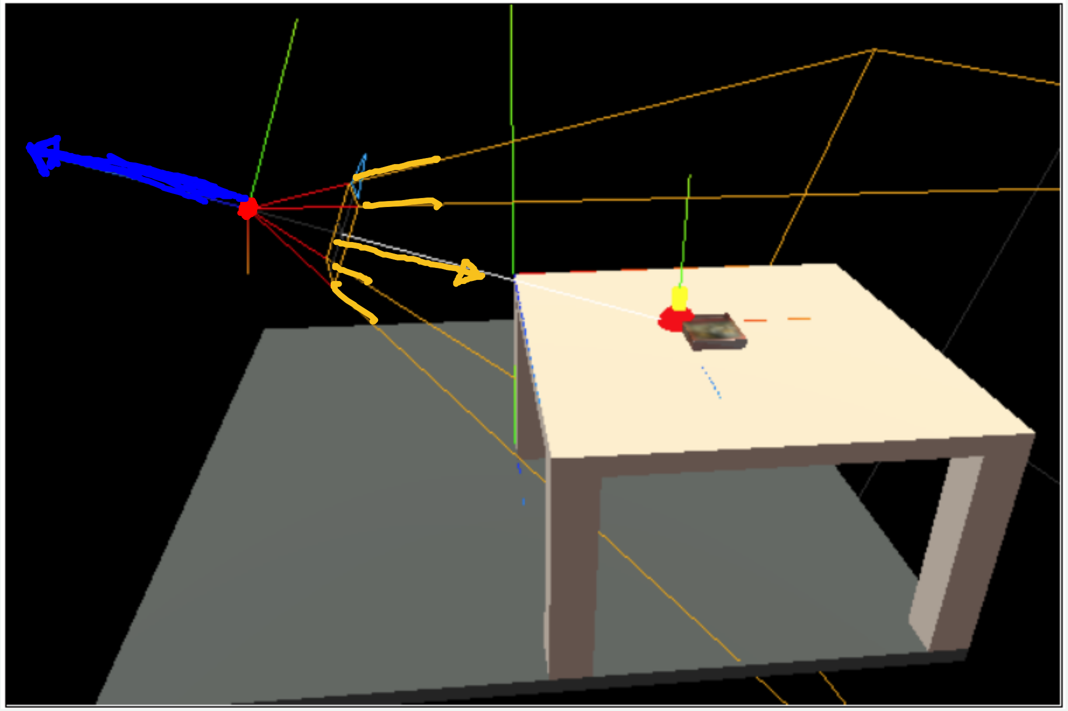
Camera looks at Book on Table

(in Scene)

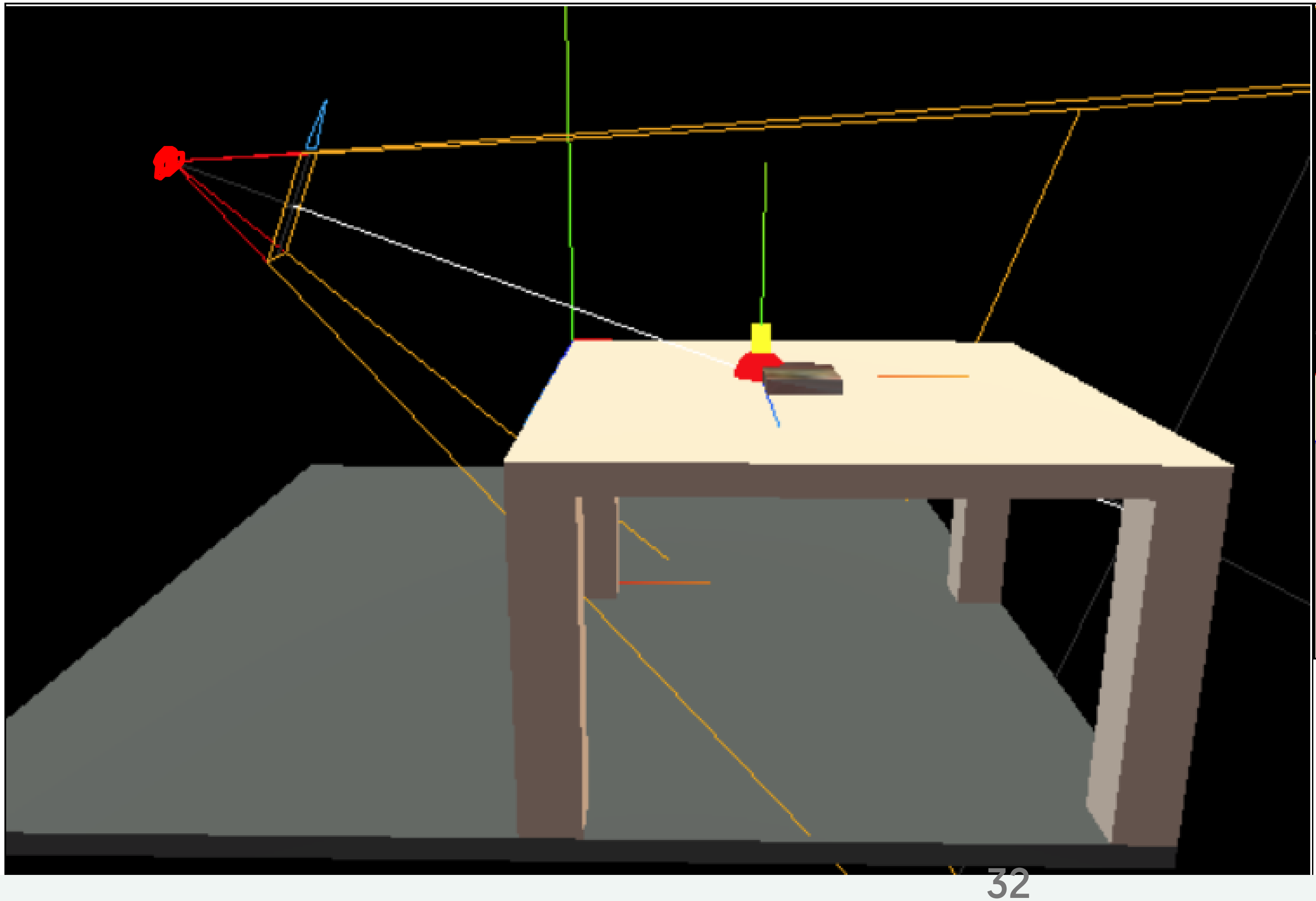
The camera is just an object

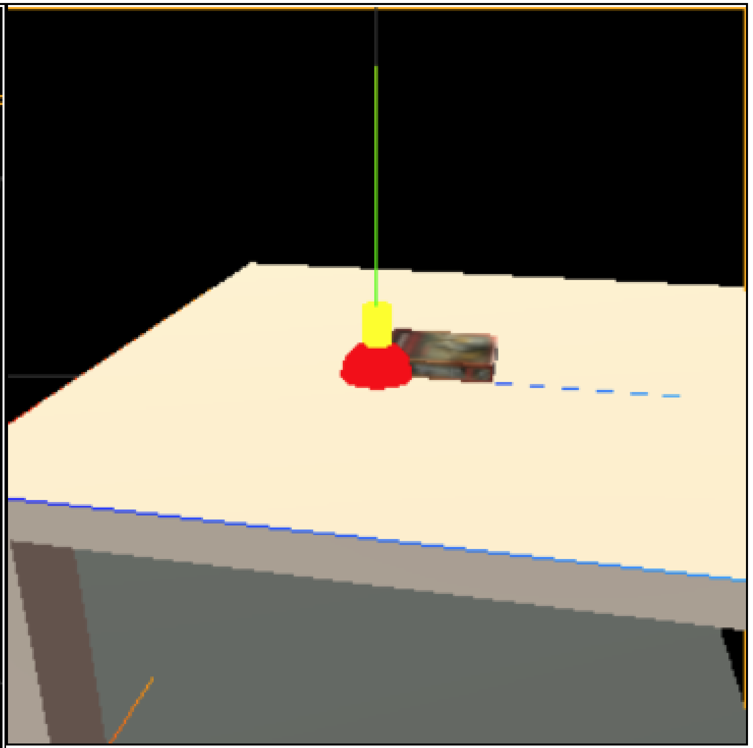
It has a coordinate system

It can be positioned and oriented



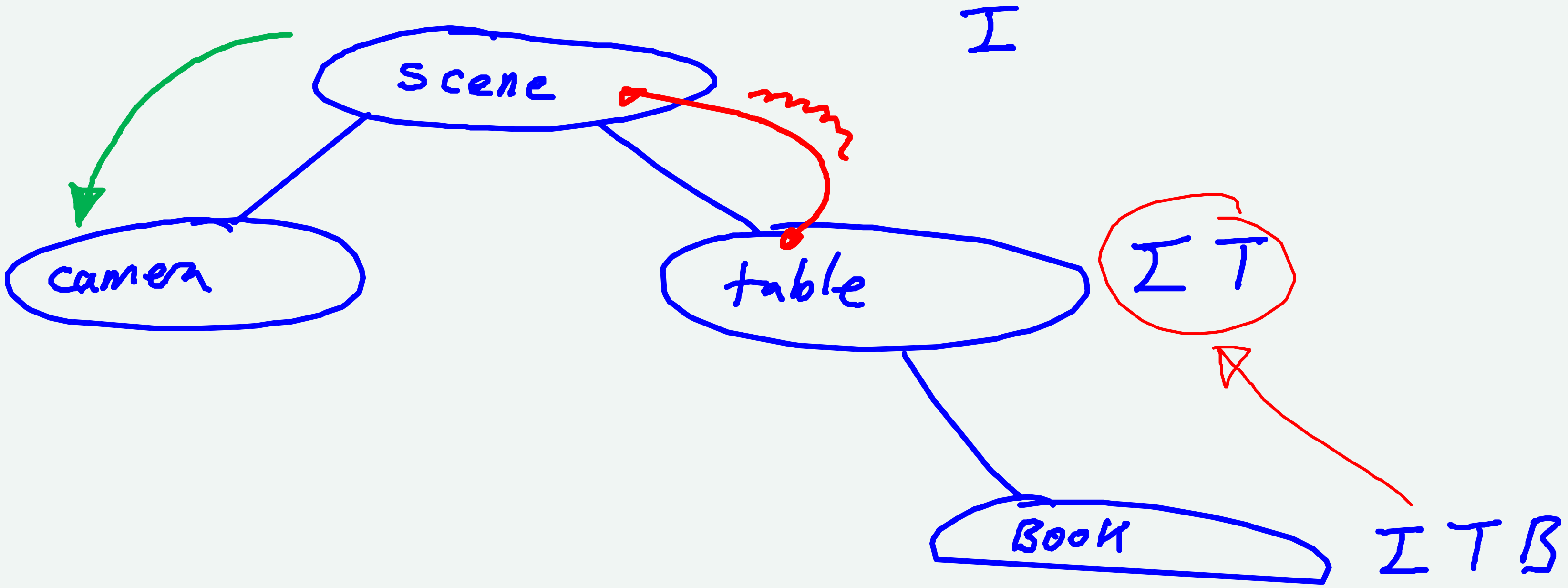
What does the camera See?





4

Demo



Coordinate Systems

Local Coordinates - how the object is defined

- vertices of the triangles
- sub-parts

Group Coordinates - any object's coordinate system

- child objects relative to the parent

Scene Coordinates - the scene is like an object

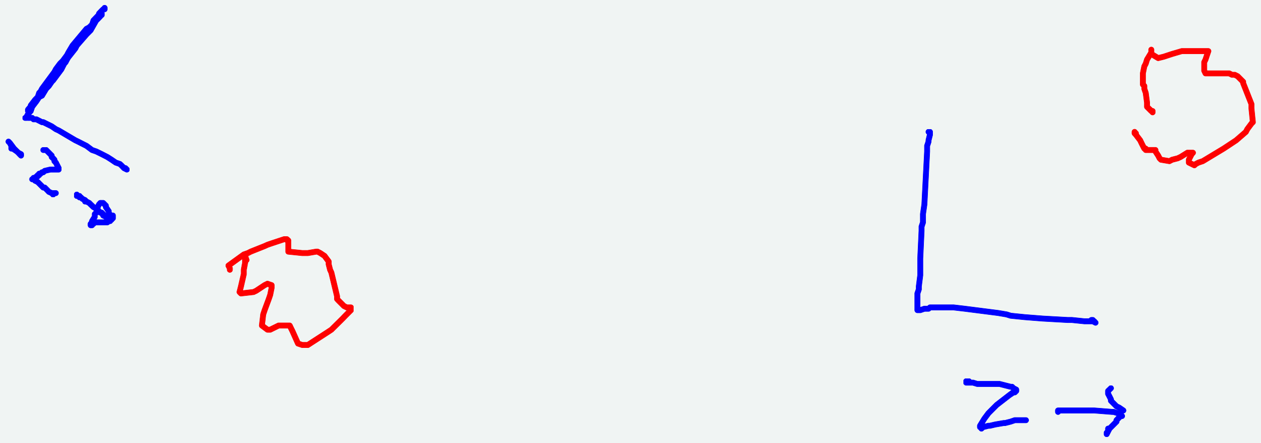
World Coordinates - doesn't matter, scene is OK

Camera Coordinates

The Camera is just another object (in the scene)

It has a coordinate system

We can transform objects into the camera's coordinate system

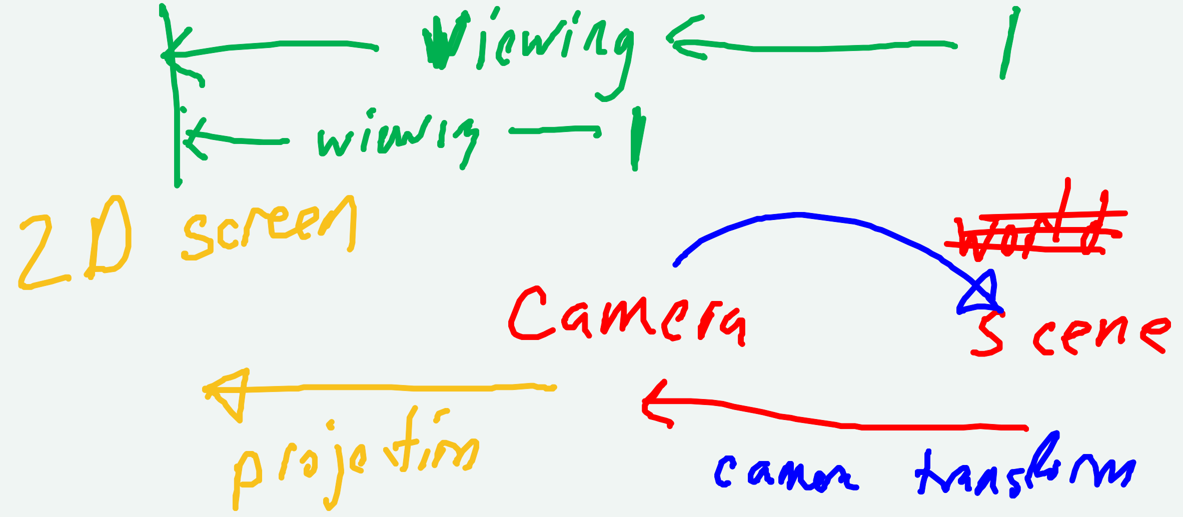


Viewing

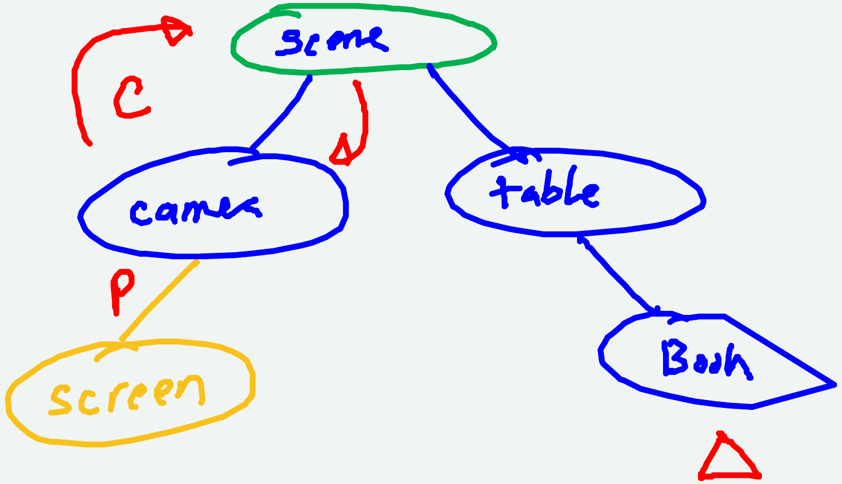
We transform from the camera's coordinate system into "screen coordintates"

We'll discuss in detail

But... it's just another homogeneous transformation



From object to screen...



$$P \quad C^{-1} \quad \square \quad T_{ABLE} \quad B_{ook} \quad \Delta$$

camera coords

scene coords

Transformations in 3D

4x4 Homogeneous Transformations

1. Affine transformations to position objects in world
2. Camera transformations to position relative to camera
3. Projection (Viewing) transformation to position on screen

Multiply matrices to combine!

THREE as an API

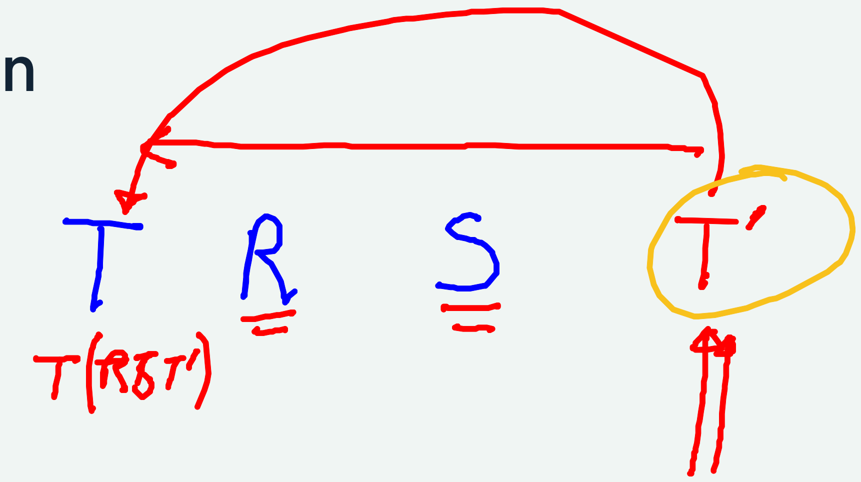
It is a **scene graph** API

We do need to explicitly render (immediate)

It is like SVG in some ways

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

Objects have state that can be set directly screen ← $P_{(0)} \; C^{-1} \; T \; B$ pt

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