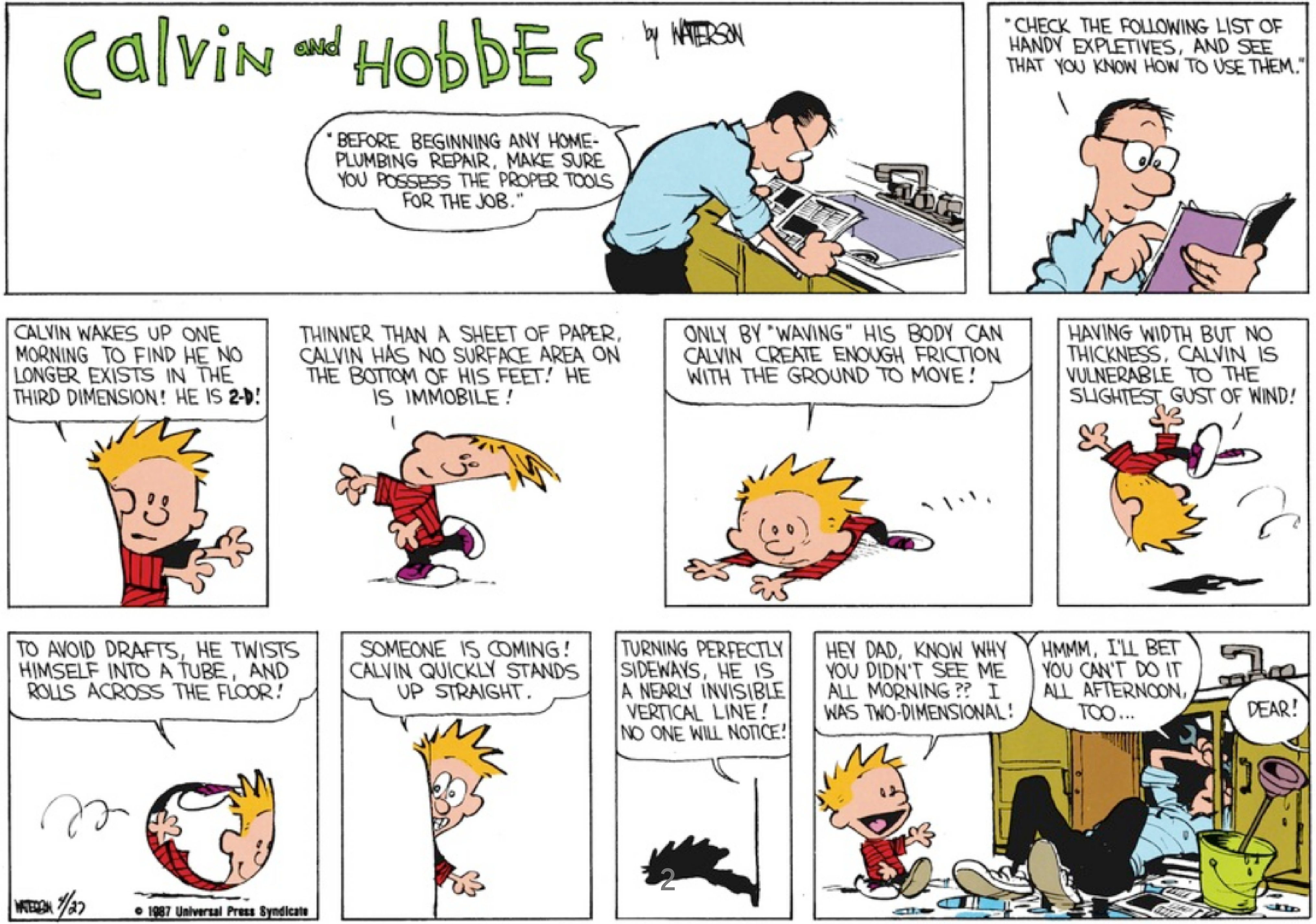
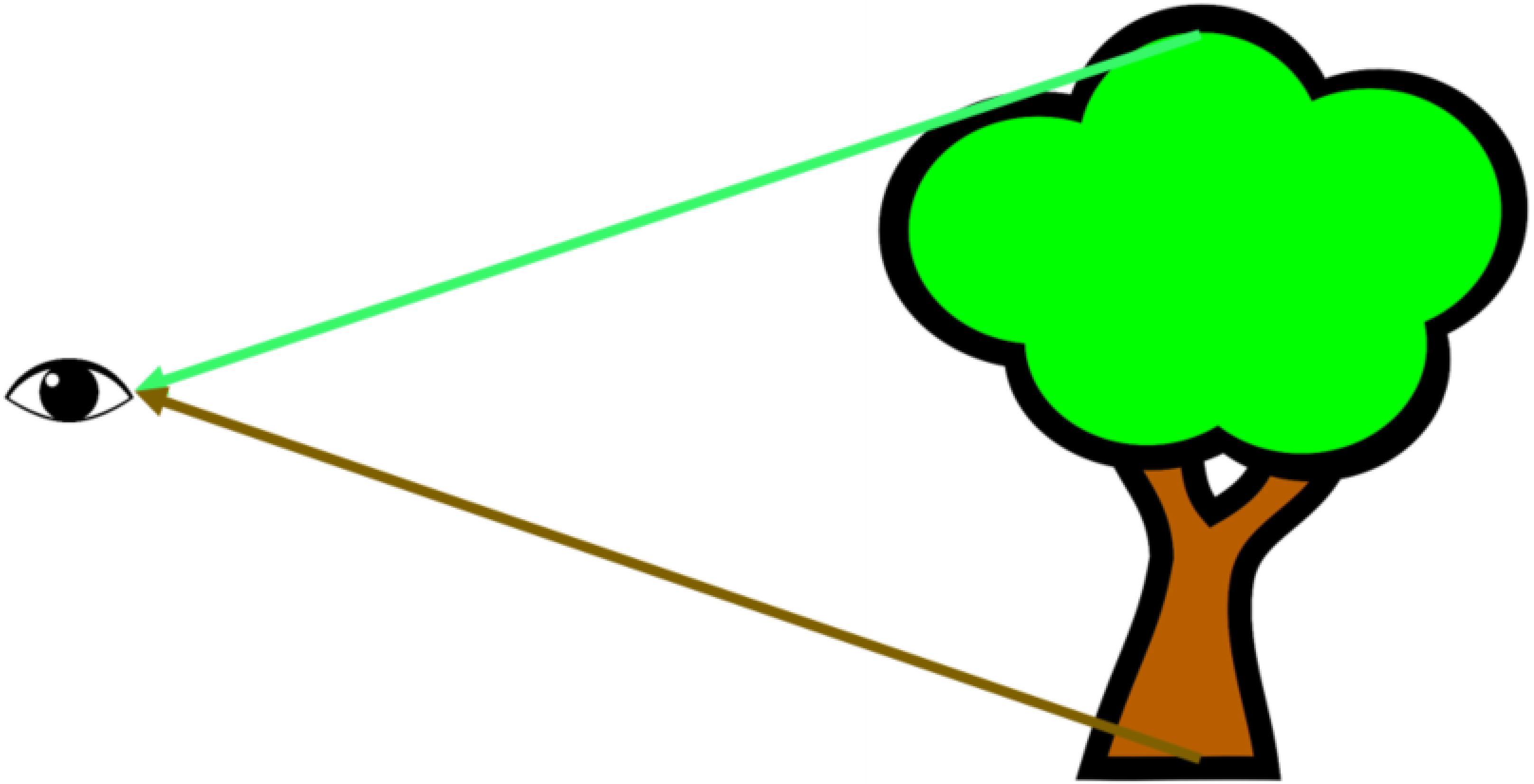
Lecture 12: 3D

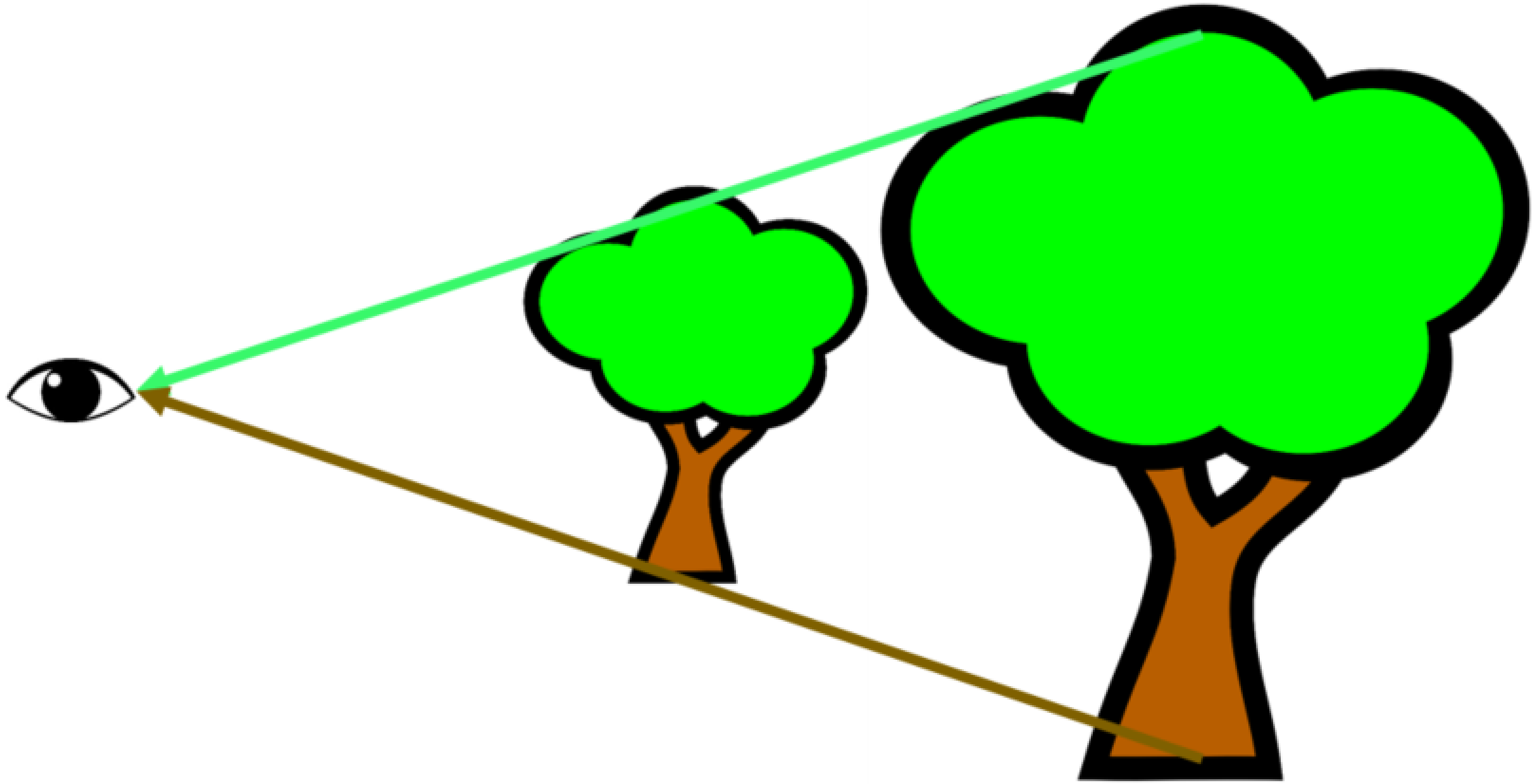

Calvin and HobbEs
by WATERSON
"BEFORE BEGINNING ANY HOME-PLUMBING REPAIR, MAKE SURE YOU POSSESS THE PROPER TOOLS FOR THE JOB."
"CHECK THE FOLLOWING LIST OF HANDY EXPLETIVES, AND SEE THAT YOU KNOW HOW TO USE THEM."
CALVIN WAKES UP ONE MORNING TO FIND HE NO LONGER EXISTS IN THE THIRD DIMENSION! HE IS 2-D!
THINNER THAN A SHEET OF PAPER, CALVIN HAS NO SURFACE AREA ON THE BOTTOM OF HIS FEET! HE IS IMMOBILE!
ONLY BY "WAVING" HIS BODY CAN CALVIN CREATE ENOUGH FRICTION WITH THE GROUND TO MOVE!
HAVING WIDTH BUT NO THICKNESS, CALVIN IS VULNERABLE TO THE SLIGHTEST GUST OF WIND!
TO AVOID DRAFTS, HE TWISTS HIMSELF INTO A TUBE, AND ROLLS ACROSS THE FLOOR!
SOMEONE IS COMING! CALVIN QUICKLY STANDS UP STRAIGHT.
TURNING PERFECTLY SIDEWAYS, HE IS A NEARLY INVISIBLE VERTICAL LINE! NO ONE WILL NOTICE!
HEY DAD, KNOW WHY YOU DIDN'T SEE ME ALL MORNING?? I WAS TWO-DIMENSIONAL!
HMMM, I'LL BET YOU CAN'T DO IT ALL AFTERNOON, TOO...
DEAR!
WATERSON 9/27
© 1987 Universal Press Syndicate

Drawing in 3D

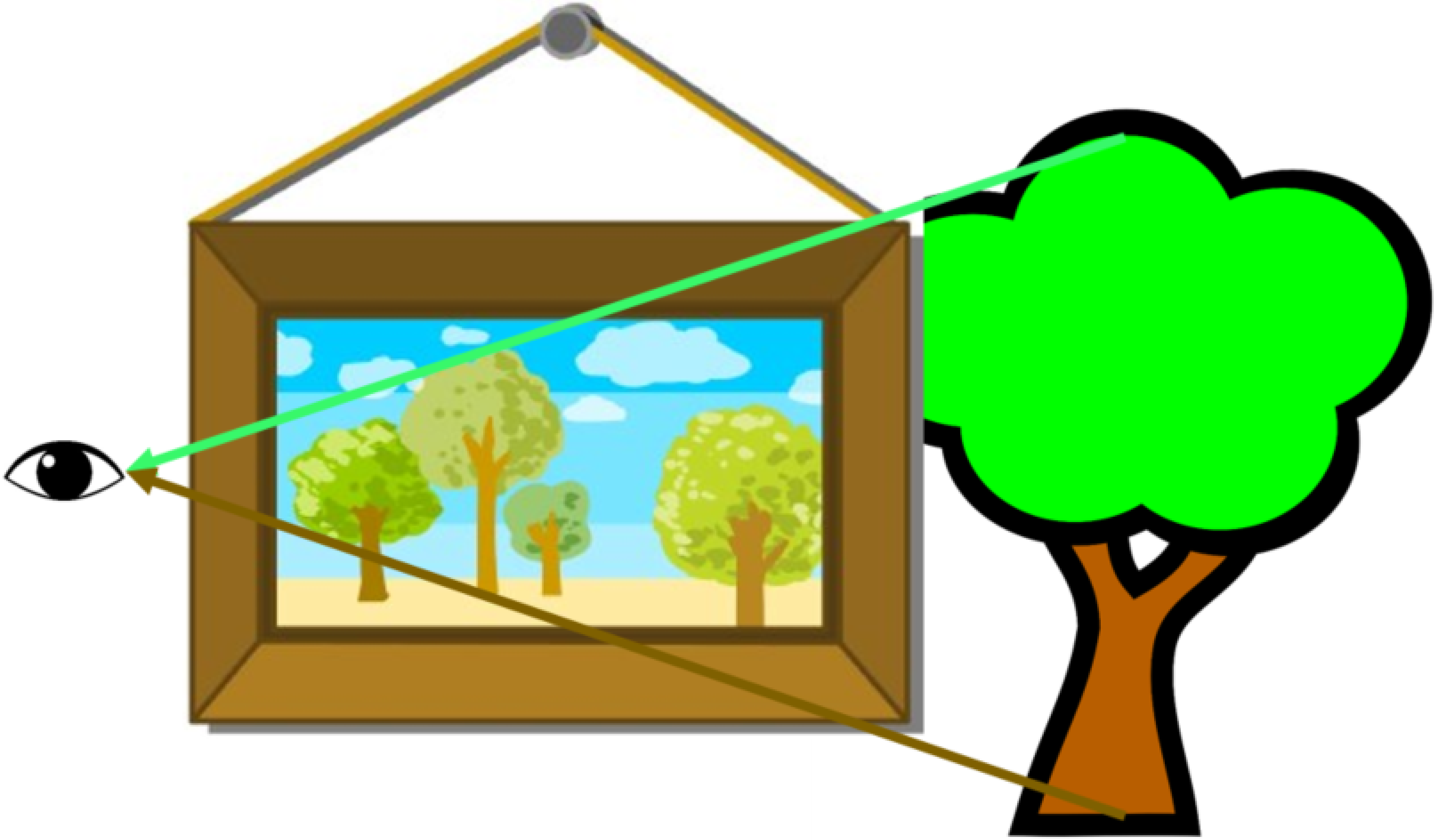
Looking at things: Depth and Distance



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

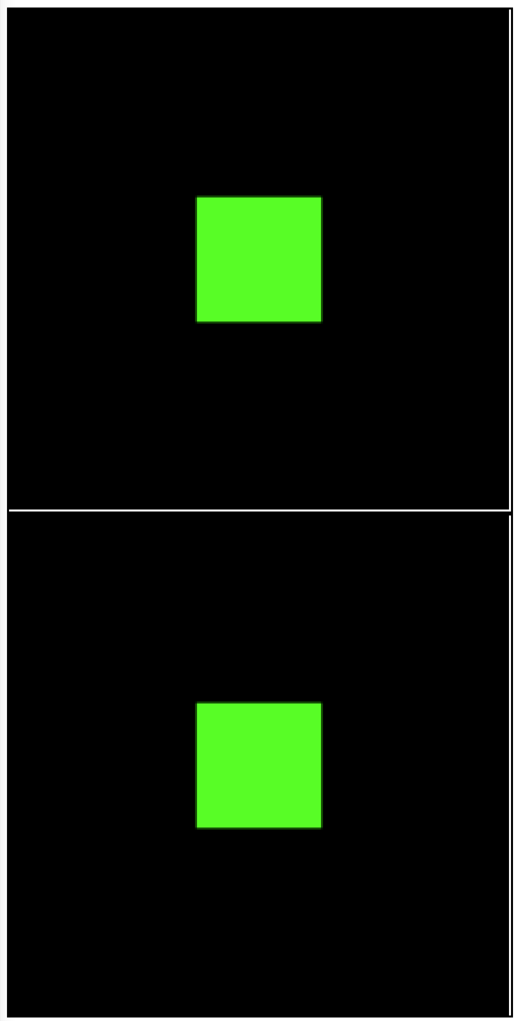
Horizon Elevation

Long Distance Shifts

OK - so how do we do that?

3D Graphics

3D?



Which one is 3D? (they are identical)

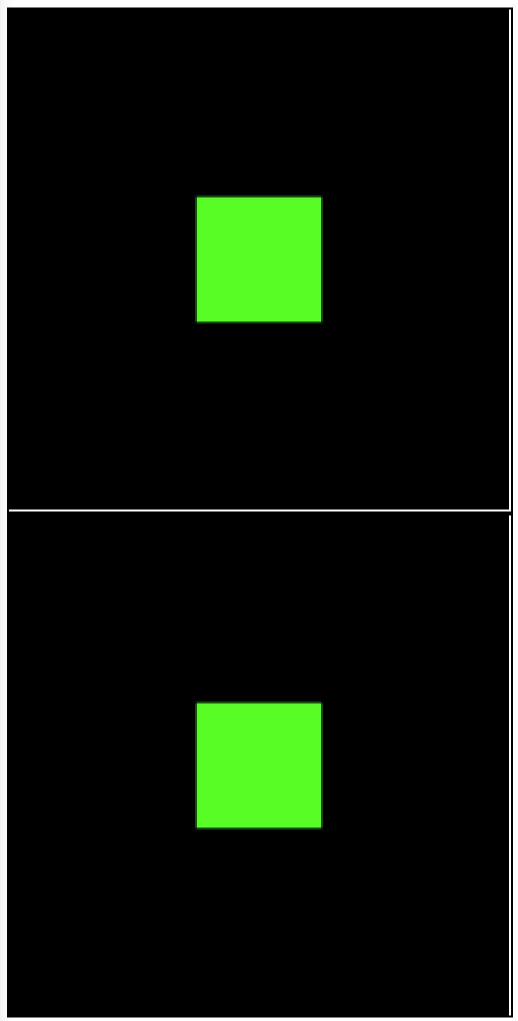
One is just a square (2D)

One is a cube (3D) viewed from the side

We see the same pixels

I made the background black

2D

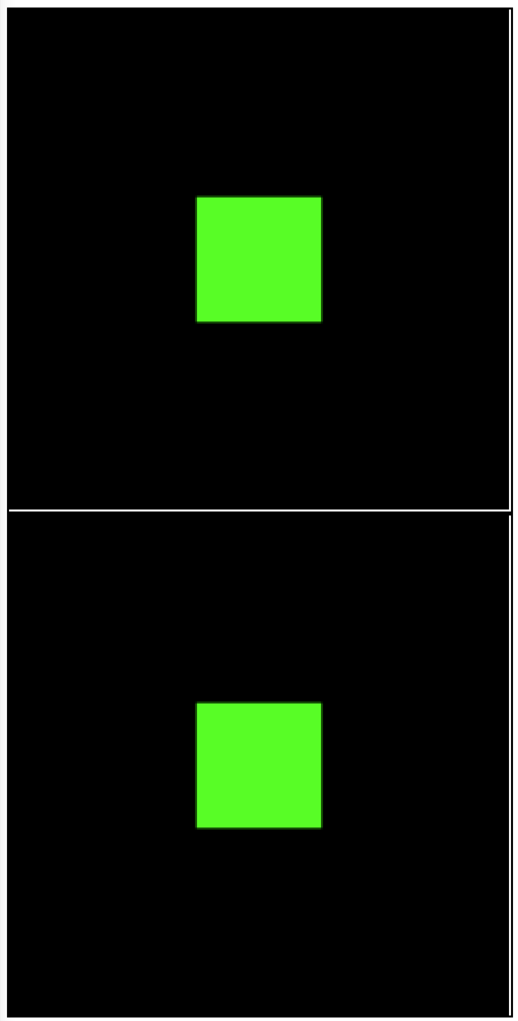


```
let canvas = document.createElement("canvas");
canvas.height = 200;
canvas.width = 200;
document.getElementById("div2").append(canvas);

let context=canvas.getContext("2d");
context.fillStyle = "black";
context.fillRect(0,0,canvas.height,canvas.width);

context.fillStyle = "#00FF00";
context.fillRect(75,75,50,50);
```

3D



```
import * as T from "../libs/CS559-Three/build/three.module.js";

let renderer = new T.WebGLRenderer();
renderer.setSize(200, 200);
document.getElementById("div1").appendChild(renderer.domElement);

let scene = new T.Scene();
let camera = new T.OrthographicCamera(-2, 2, -2, 2, -2, 2);

let material = new T.MeshStandardMaterial({ color: 0x00ff00 });
let geometry = new T.BoxGeometry(1, 1, 1);
let mesh = new T.Mesh(geometry, material);
scene.add(mesh);

let light = new T.AmbientLight( 0xFFFFFF);
scene.add(light);

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

Drawing in 3D

Three new things (not in 2D):

Where on the screen do things appear?

- Viewing (transform world to screen)

Do I see the object? (occlusions)

- Visibility

What color should it be?

- Shading (needs to consider lighting, surfaces, ...)

Other things are more complicated (shape, ...)

Thinking in terms of a World

Make a world

Take a picture of it

(rather than the act of drawing)

What does it take to "draw" in 3D?

1. Have a world
2. Make objects from primitives
3. Place objects in world
4. Figure out what color/style
5. Transform to screen
6. Figure out what is visible
7. Color the pixels

THREE.js

A mid-level graphics API

Easy (relatively) to get started

Takes care of messy bits

Allows us to work with scenes/objects - and not hardware

Graphics Abstrations

1. Have a world
2. Make objects from primitives
3. Place objects in world
4. Figure out what color/style
5. Transform to screen
6. Figure out what is visible
7. Color the pixels

Graphics Abstrations

1. Have a world
2. Make objects from primitives
3. Place objects in world
4. Figure out what color/style
5. Transform to screen
6. Figure out what is visible
7. Color the pixels

THREE.JS Concepts

1. Scenes
2. Geometries, Meshes
3. Transformations, Hierarchy
4. Materials
5. Cameras
6. Renderer
7. Renderer

What about the built in Canvas API?

We've been doing:

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

Why not:

```
let context = canvas.getContext("3d");
```

or

```
let context = canvas.getContext("webgl");
```

Abstractions of the "3D Canvas"

Frame buffer (pixel storage)

4D Vectors

Triangles

Shader program management

Memory blocks and buffers

Options for various hardware operations

How to manage the graphics hardware

Hello Triangle!

Just draw a yellow triangle

Really in 2D (since everything is positioned on screen)

http://graphics.cs.wisc.edu/WP/cs559-fall2015/2015/10/05/webgl-and-glsl-examples-fromfor-class/
https://jsbin.com/popiqi/edit?html,js,output

(the simple version is more than 90 lines, 43 lines without comments)
(plus shaders written in GLSL)

1. Vertex Shader (in GLSL - not JavaScript)
2. Fragment Shader (in GLSL - not JavaScript)
3. Load and Compile Shaders (JS runs GLSL compiler)
4. Setup to use shaders
5. Create vertex buffers (with triangle information)
6. Bind buffers (send memory to hardware)
7. Clear Screen
8. Enable Drawing Modes
9. Setup to draw
10. Excute buffers

We'll be able to do all that ... later

In THREE we can choose to control everything, but...

- Reasonable defaults for most things
- Appropriate abstractions that match what we learn
- General purpose implementations of key things
- Nice implementations of lots of fancy features
- Convenient scene-graph API

Hello Cube

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

Hello Square/Box (Workbook 6:2-1)

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

Where did the "graphics" go?

1. Hardware Setup (prepare "window")
    - Automatically creates canvas with settings
2. Triangle Drawing
    - Done by the hardware (or low-level drivers)
3. Shader Programs (program hardware for colors)
    - Standard shaders for common (fancy) lighting
    - Easy to write your own
4. Procedural drawing (specifying to draw the triangles)
    - sends the "world" to the hardware

Where did the "graphics" go?

5. Viewing Transformation
   - Camera Object
   - Matrices inside
6. Modeling Transformations
   - Matrices inside of objects
   - Can specify in many different ways
7. Making Triangles
   - High-level "Graphics Objects"
   - Meshes (collections of triangles)
   - Automatically organizes for the hardware

OK, Back to that program...

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

Before we get started...

We need to load THREE as a module

```
import * as T from "https://cdnjs.cloudflare.com/ajax/libs/three.js/110/three.module.js";
```

or, in a workbook (when you have your own copy of THREE)

```
import * as T from "../libs/THREE/src/Three.js";
```

Type Information and THREE

The workbook is set up so that you get all the typing information for THREE.

You can see the typing information in the library directory.

If you don't like using typing...

You can remove the comment that turns on type checking

```
// @ts-check
```

Hello Cube

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

Hello Square/Box (Workbook 6:2-1)

```
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: Renderer (domElement)

```
let renderer = new T.WebGLRenderer();
renderer.setSize(200,200);
document.body.appendChild(renderer.domElement);
```

Let THREE make the Canvas for you! (deals with drawing params)

You can do...

```
let canvas = /** @type {HTMLCanvasElement} */ ( document.getElementById("canvas1"));
let renderer = new T.WebGLRenderer({ canvas: canvas });
```

What's going on

Lots of parameters need to be set correctly

- Defaults are OK
- It would be a lot of boilerplate code
- You can set the various parameters if you need to

In the Future...

- This sets up a lot of machinery that we will learn about

2: World/Scene (Scene)

```
let scene = new T.Scene();
```

This is the root of the display list Tree - a container for objects

What's going on

THREE is a scene graph API

- It will keep a tree of objects
- The scene is the root of the tree
- It doesn't do much other than collect objects

3. Geometry: make the cube

```
let geometry = new T.BoxGeometry(1,1,1);
```

Geometry is a collection of triangles

What's going on

Geometry as an abstraction:

- make everything out of triangles
- store triangles using special data structures
- standard shapes provided

In the future

- Understand why we need these special data structures
- Make some of our own

4: Material

```
var material = new T.MeshBasicMaterial( { color: 0x00ff00 } );
```

We need the "stuff" the object is made out of before the object

What's going on

Abstract the appearance of objects

- Surface properties, how it interacts with light, ...
- Will allow us to do fancy things later
- THREE provides many options - easy to do fancy things

In the future:

- Learn to use many of the materials
- Learn the math to make our own materials

What's hidden?

Materials take care of a lot for us

- Shaders (per-pixel programs to do appearance)
- Lighting
- Texturing / Coloring (fancy appearances)

5.A Mesh: make it an object

```
let mesh = new T.Mesh(geometry, material);
```

A **Mesh** is a THREE graphics Object

Unfortunate Naming

A Mesh (in graphics) is a collection of triangles

Three's Geometry is more like a Mesh

Three's Mesh "has a" Geometry (and a Material)

Three's Mesh "has a" Transform, hierarchy, ...

Three's Mesh is an **Object3D**

5.B Add it to the world

```
scene.add(mesh);
```

Object3D: The objects in the world

Objects in the world (that we add to Scenes)

Has a lot of functionality

- Can be a Node in the Scene Tree (any object can have children)
- Has a transformation (relative to parent)
- Provides useful functions for working with transformations
- Transformations are hidden inside

Transformations

We left the object at the origin!

Generally, transformations are inside of objects

Future:

- Learn to use many of the tools Three provides
- Understand how the transformations work

Note: hierarchical model means no explicit stack

Hierarchy in THREE

`Scene` is like a special object (no parent)

- Objects have parents and children
- Objects have transformations relative to their parent
- Most (All?) Objects can have children

Transformations in THREE

- Object3D has a matrix
- Object3D has position / rotation / scale
- Object3D has "transform" methods

Three builds the matrix from position, rotation, scale

- always in that order

You can take care of the matrix yourself

- `object.matrixAutoUpdate = false;`

Transformations in THREE

- Object3D has a matrix
- Object3D has position / rotation / scale
- Object3D has "transform" methods

It tries to make things easy for you.

This can be confusing

- `object.position.x = 10;`
- `object.translateX(10);`

Where do we put things?

Anywhere we want!

3D space is up to us

We define the **camera** that looks at things (and places them in 2D)

6. Camera

```
let camera = new T.OrthographicCamera(-2,2, -2,2, -2,2);
```

A **Camera** projects the world onto the screen

The Camera as a Transformation

From the World to the Screen

World Coordinates

Where our objects live

Can be anything we want

Right-handed

Y-up is convention

Screen Coordinates

Actual coordinates on the screen

2D + depth

We don't need the details (yet)

The Viewing Transformation

Let Three take care of it for us

- Provides a flexible camera (multiple kinds)
- Provides good functions for manipulating cameras

Future:

- Understanding this transformation is a key to 3D graphics
- Understanding perspective is part of this

7. Draw!

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

The render command causes things to get drawn

Actual Drawing

`render` is what causes the image to be created

It takes a picture of the world in its current state

Animation Loop:

- update world (based on time/input)
- draw (render)
- repeat

Drawing takes care of things

In the future, we'll want to understand the details

- Visibility (near blocks far)
- The actual process of transformation from 3D->2D
- The actual process of determining appearance

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

More?

1. Understand how things work
2. Learn more pieces to use to make better pictures