CS559 Midterm 2

Shapes Basics – lec9, 12

curves vs. areas vs. surfaces vs. volumes

Not all curves are the boundaries of areas

Not all surfaces are the boundaries of solids

- 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, ...

implicit vs. parametric vs. subdivision forms

- Implicit: f(x,y) = 0, geometric test, harder for drawing
- Parametric: (x,y) = f(t), generate points, free parameter (t) control mapping
  - Curve: a set of points
  - parameterization: mapping
- Subdivision: initial points, converges rule to add new points, repeat infinitely
  - limit curve

free parameters (parametric representations)

control mapping, can always scale to 0–1

convention:

- u for unit parameterization [0,1]
- t for more general cases, include units

Parametric Curves – lec9

tangents – lec9, 10

x' = f'(x), vector, function of the free paremeter

The 2nd derivative can be seen as 2 parts:

- component in the direction of the tangent
- component perpindicular to the direction of the tangent

-> Normalize the "speed" (unit tangent), 2nd derivative is perpindicular

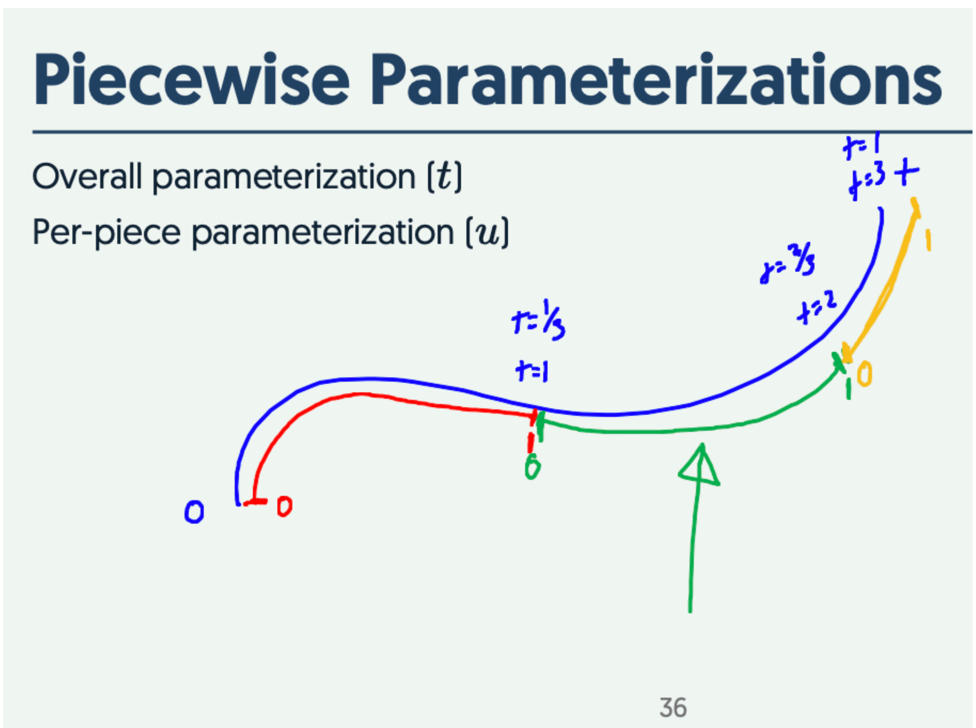
piecewise polynomials and parameters – lec9 p33

Chains of low-degree polynomials

line segment chains (1st degree), chains of 2nd or 3rd degree (or more)

Given n points, you can make an $n-1$ degree polynomial

hard to compute, hard to control, unwanted wiggles



cubic segments – lec9 p47

- specify position and 1st derivative at the ends C(1)
- interpolation, local control
- 4x4 matrices (just like 3D transformations)

$$f(u) = a_0 + a_1 u + a_2 u^2 + a_3 u^3$$

Blending (basis) function forms

Define in terms of basis functions

$$f(u) = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p'_0 + b_3(u)p'_1$$

More: lec11 p28

Piecewise Parametric curves – lec9

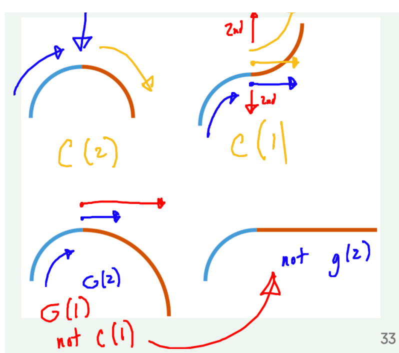
continuity conditions – lec9

C(n)

- c(0), no gaps –> positions
- c(1), no corners –> positions, tangents (1st derivatives)
- c(2), looks smooth –> positions, tangents (1st derivatives), 2nd derivatives
- important for airflow (airplane, car, boat design) and reflections

C vs. G continuity – lec9

C(n) continuity – all derivatives up to *n* match G(n) continuity – the directions of the derivatives match, ignore speed



- ## Hermite forms – lec9 p49

can make c(1) easily, end point and derivative at end point to be the same

specify position and 1st derivative at ends: Po, p1 and p0', p1'

need to compute a_i from these

$f(u) = p_0 u^0 + p_0' u^1 + (-3p_0 - 2p_0' + 3p_1 - p_1')u^2 + (2p_0 + p_0' - 2p_1 - p'1)u^3$

$f(u) = (1 - 3u^2 + 2u^3)p_0 + (u - 2u^2 + 1)p_0' + (3u^2 - 2u^3)p_1 + (-u^2 + u^3)p_1'$

or $f(u) = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p_0' + b_3(u)p_1'$

basic functions b(u), e.g. $b_0(u) = 1 - 3u^2 + 2u^3$

matrix form

Basis Matrix

Matrix Form

$$\begin{bmatrix} u^0 & u^1 & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} p_0 \\ p_0' \\ p_1 \\ p_1' \end{bmatrix}$$

The matrix is for **Hermite** curves - different matrix, different control points

Interpolating Curves – lec10, workbook5

interpolation: specify the value of curve at a particular site

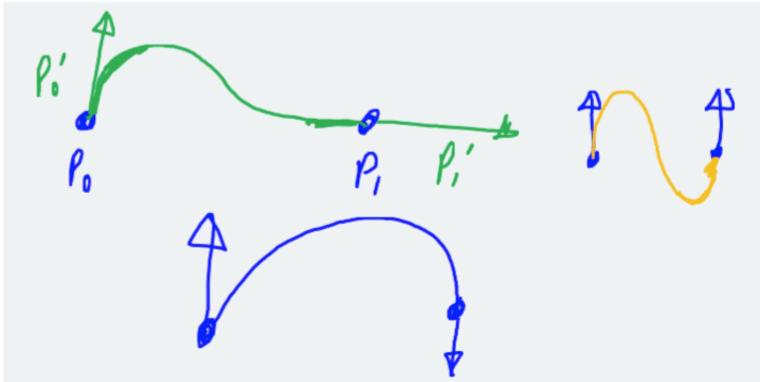
f(ui) = pi, f'(ui) = pi'

- ### Hermite interpolation

see above for hermite equations

$f(u) = p_0 u^0 + p_0' u^1 + (-3p_0 - 2p_0' + 3p_1 - p_1')u^2 + (2p_0 + p_0' - 2p_1 - p'1)u^3$

- sketching and designing with Hermites – lec11 p 8



- Cardinal Splines, Catmull–Rom splines – lec11 p10

https://cs559-spring21.github.io/wb05-YoungWu559/docs/5/

avoid specifying derivatives, compute derivatives based on neighbor points

e.g. p1' = p2 – p0

not interpolate the endpoints

n points –> n–1 degree polynomial,

hard to compute, control, unwanted wiggles

C(1)

- sketching, drawing, converting to other forms, ...

- locality (interpolating high–order polynomials)

changing a point only influences a limited amount of the curve

Bezier Curves

Bezier curve principles and properties – lec 10 p 21

Some points interpolate, others influence

2 points: connect the dots (line) – or anything else!

If we are not interpolating the third point...

- Interpolate the end points
- Stay inside the triangle
- Not "wiggle too much"
- Symmetry (forward/backwards)
- Locality (only these points)
- Control tangents (2* vector)
- Generalize to higher degree (more points)

n points –> n–1 degree polynomial

Advantages

- Efficient algorithms
- Common UIs
- Supported in most APIs
- Nice mathematical properties
- Affine Invariance: transform points –> transform curve
- Elegant derivations

max crossing = degree of polynomial –> 4 points: 3 crossing

Quadratic Bezier Curves –lec10, p 27, lec11

3 points

Three points will give quadratic polynomials $d = n - 1$

$f(u) = B_1(u)p_1 + B_2(u)p_2 + B_3(u)p_3$

$B_1(u) = (1-u)^2$

$B_2(u) = 2u(1-u)$

$B_3(u) = u^2$

quadratic to cubic:

C0 = Q0

C1 = Q0 + (2/3) (Q1 – Q0)

C2 = Q2 + (2/3) (Q1 – Q2)

C3 = Q2

Cubic Bezier Curves (relationship to Hermite)

4 points

$f(u) = B_1(u)p_1 + B_2(u)p_2 + B_3(u)p_3 + B_4(u)p_4$

$B_1(u) = (1-u)^3$

$B_2(u) = 3u(1-u)^2$

$B_3(u) = 3u^2(1-u)$

$B_3(u) = u^3$

Geometric Algorithms (DeCastlejau) – lec10

Repeated Linear Interpolation to evaluate curve

workbook5: https://cs559-spring21.github.io/wb05-YoungWu559/docs/7/

Basis Functions (Bernstein forms) – lec 10

General blending(basic) function:

- $f(u) = \sum_{i=0}^{d} b_{i,d} p_i$
- $b_{k,n}(u) = C(n,k) u^k (1-u)^{n-k}$

limits of Bezier curves (why rational curves) – lec11

- Sometimes we want interpolation
- They are polynomials
  - Some shapes cannot be represented exactly(e.g., circles)
- The are "only" affine invariant: transform points: transformed curve
  - not projective invariant
- Hard to get smoothness better than C/G(1)

Rational polynomial curve

- a curve as the ratio of two polynomials
  - $f(u) = \frac{\sum a_i u^i}{\sum b_i u^i}$
- allows for projective invariance
- more shapes
- complicated
- mechanical design –> exact shape

- ### Bezier curves in APIs – lec10

```
context.quadraticCurveTo(cx,cy, x,y);
context.bezierCurveTo(c1x,c1y, c2x,x2y, x,y);
```

works like lineTo, extend path

Advanced Curve Topics – lec11, workbook5

- ### Arc length and arc-length parameterization

https://cs559-spring21.github.io/wb05-YoungWu559/docs/10/

Compute the distance along the curve from 0 to u,

require integral -> usually must be approximated by breaking into small segments

parameterization:

- use the length as parameter
- s = distance along the curve, not time
- can normalize (total distance 1 or 100%)
- s changes at constant rate -> velocity along the curve will be constant
- equal steps in s give equal amounts of distancd
- give s -> compute t

- ### Approximating curves with segments

https://cs559-spring21.github.io/wb05-YoungWu559/docs/10/

- ### B-Splines

- motivations

- A very general formulation of curves
- Many different types of B-Splines
- Interesting History
- Geometric and Algebraic Derivations

advantages

- Curves of any degree d
- Locality: each control point influences $d + 1$ segments

- Continuity: curve is $C(d-1)$
- Stays with the Convex Hull
- Symmetric
- Affine Invariant
- Variation Diminishing
- not interpolating

Geomatic approach: chakin corner cutting

subdivide each line segment, piecewise quadratic, converges to a B-Spline

Approximating curve: it does not interpolate its (initial) points

blending, basis functions

Degree d has $d+1$ segments, meet with $C(d-1)$ continuity

3D Basics – lec12

how we see in 3D

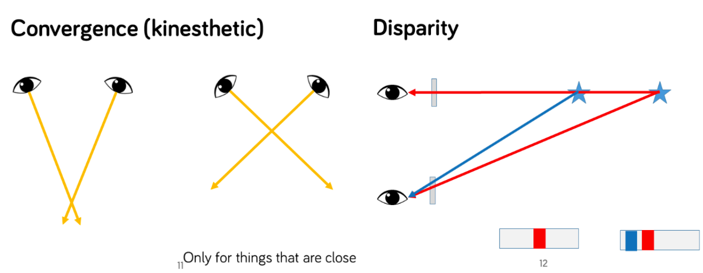
depts and distance, sense 2D, infer 3D

depth cues (1 eye, 2 eye, image based)

One eye: accomdation

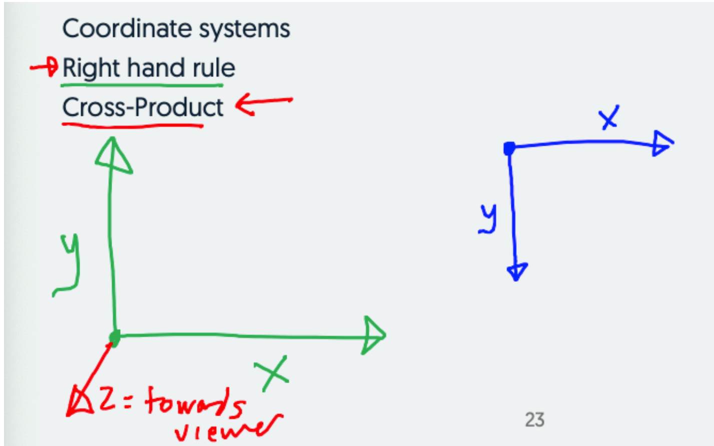
- depth from focus: cameras
- weak cue
- Everything more than a meter is far

two eyes: convergence, disparity



many eyes (multiple time): parallax, depth from motion

- ## 3D coordinate systems, right hand rule



- ## Normals and Tangents

In 3D,

- the tangent to a surface (at a point) is a plane
- the normal to a surface (at a point) is a vector
- the tangent to a curve is a vector
- the normal to a curve is a plane (defined by normal vectors)

3D APIs

- ## comparison of WebGL and THREE

Three

- 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
- concepts
  - Scenes, Geometries, Meshes,Transformations, Hierarchy . Materials, Cameras, Renderer

- ## main abstractions of 3D Graphics

- 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

Graphic abstractions

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 Basics – lec13, 14

- ## hello cube abstractions – lec13

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

- ## meshes vs. geometries – lec13

Geometry: make the object, collection of triangles in three

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

Mesh: three graphics object, geometry and material, transform, hierarchy

materials – lec13

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

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

Standard Graphics Models: Lambert's model Phong's model

Fancier Models provided in Three: MeshStandardMaterial, MeshPhysicalMaterial

And Fancier ones (and programmable ones)

transformations – lec13

Object3D has a matrix

Object3D has position / rotation / scale

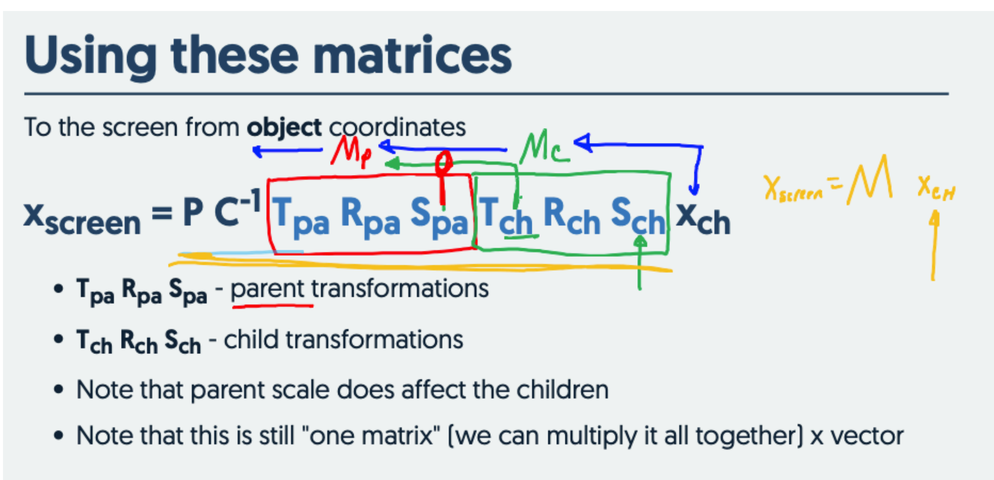
Object3D has "transform" methods

Three builds the matrix from position, rotation, scale in order

hierarchy – lec15

Group are Object3D with no geometry or material

Every object can have one parent, Might be the scene (not really a parent) or Trees (not more complex DAGs)



- ### deferred loading – lec 14

```
import { OBJLoader } from "../libs/CS559-Three/examples/jsm/loaders/OBJLoader.js";
let loader = new OBJLoader();

loader.load("./objects/07-astronaut.obj", function(astronaut) {
  astronaut.position.set(1.5, 4, 0);
  astronaut.scale.set(0.5, 0.5, 0.5);
  scene.add(astronaut);
  renderer.render(scene, camera);
});
```

asynchronous programming, function was called after loading the object

- ### lighting and shading basics – lec13, 14

Ambient Light

Point Light

Directional Light

SpotLight

Area Lights (for fancy materials)

T.MeshStandardMaterial reflect light

The lights are objects in the world

We control their transformation to place them

- ### state vs. transformation commands

state: set the state to...

transform commands: translate/rotate.....

Objects have methods to perform transformations

Objects have state that can be set directly

translate affected by rotation, set position += ... not afffected by rotation

Transformations in 3D – lec14

- ### use of homogeneous coordinates

4x4 Homogeneous Transformations

- Affine transformations to position objects in world
- Camera transformations to position relative to camera
- Projection (Viewing) transformation to position on screen

Multiply matrices to combine!

Objects each have their own transformation

built from trans, rot, scale each time

keep pieces spearate for convience

- ### rotate, translate, and scale in THREE

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)

Rotation:

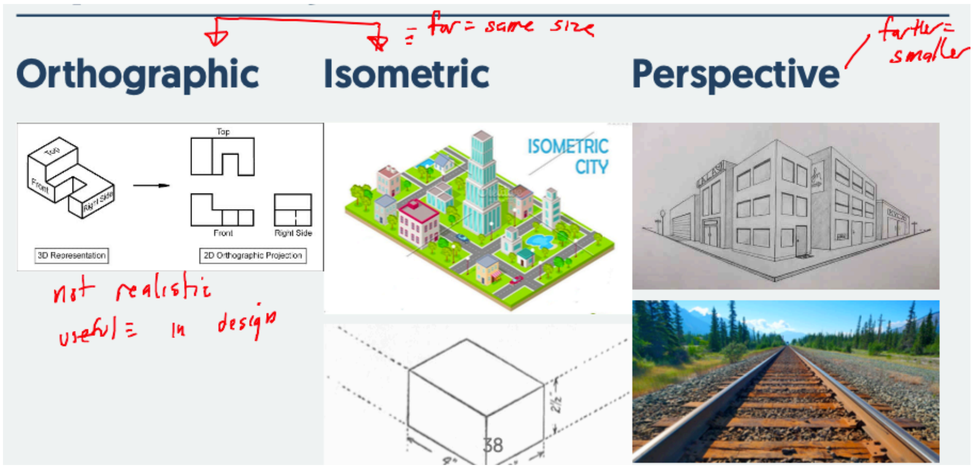
Viewing (Projection) transformations – lec14, 15

- ### Viewing vs. Camera Transformations

- camera.lookat(x,y,z) – orients the camera
    - changes the object's rotation
    - works for any object
    - rotates around the "position" so Z points toward the point
    - takes care of twist around the point

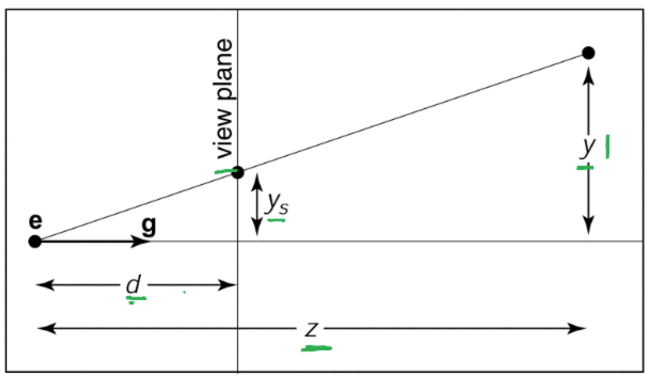
- ### type of projections

Orthographic, isometric, perspective



- projection math

Perspective math $\quad y_s = \dfrac{d}{z} y$



25

Linear?

linear in homogeneous coordinates

$$\begin{array}{cccc} x & & z & \\ \end{array}$$

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

Don't forget the divide by w!

Note what happens to z

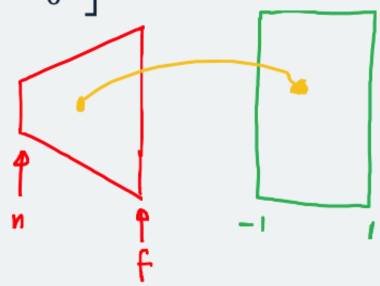
$$x' = \frac{x_p}{w_p} = \frac{dx}{z}$$

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



29

- clipping and frustum

Clip things that are too close or too far

- too close can be weird (or behind you)
- too far can be too small (or numerical issues)
- map n/f to the range +/– 1

Map the frustum to the "screen box"

Transformations position objects in the screen box Clipped in all directions

Clipping:

- You don't see things out of the "window" "

- Canonical" Coordinates:

  - origin at center
  - +/– 1 in all directions
  - X left to right (1=right edge)
  - Y bottom to top (1=top)
  - Z back to front* (1=front), Note: we look down the negative z axis – but THREE flips this
- Important internally – hidden inside the transformations in THREE

Rotations – lec16

- basic facts

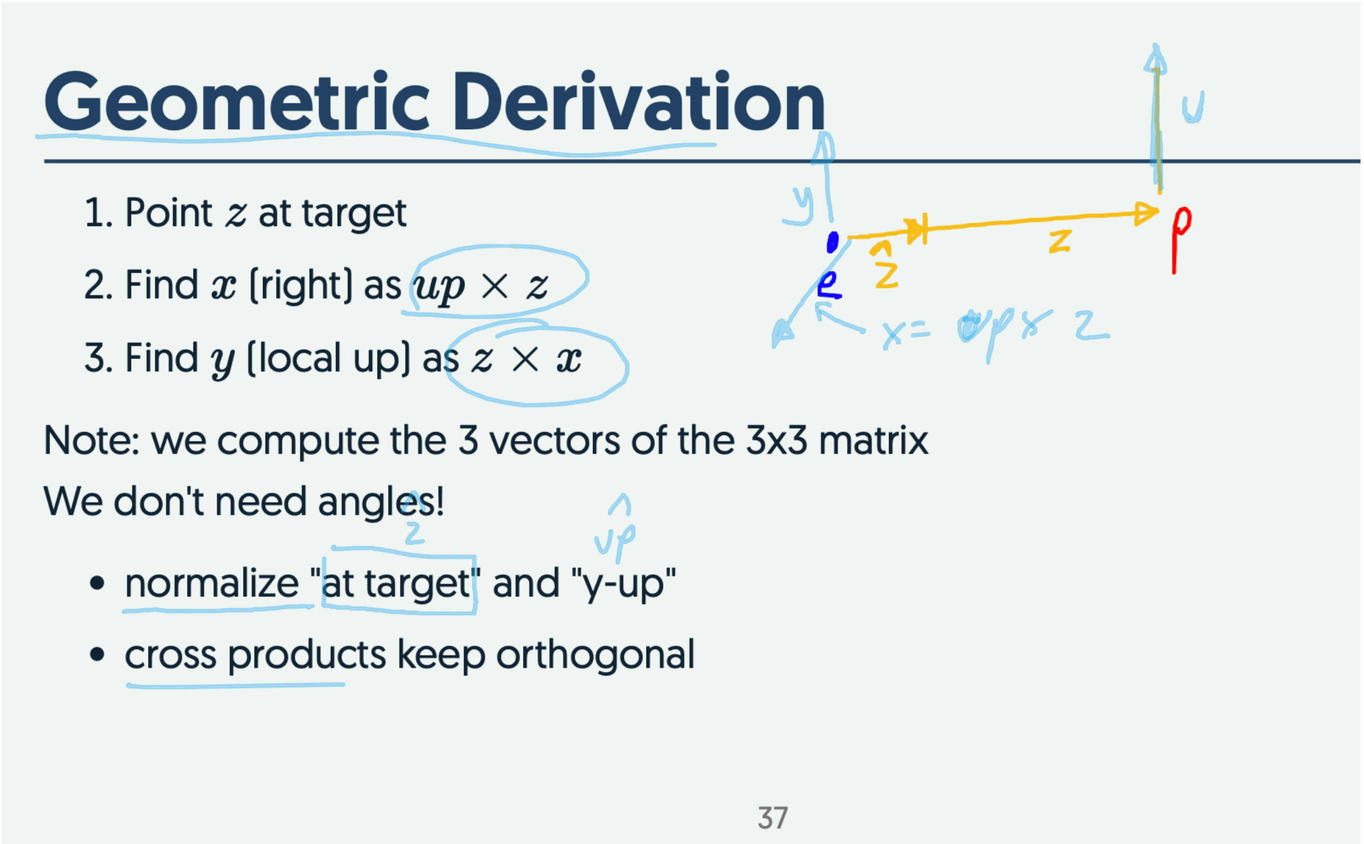
  - Orthonormal Matrices (linear transformations)
    - all rows (columns) have unit length
    - all rows (columns) mutually orthogonal
  - Positive determinant (preserve handedness)
  - Have an inverse
  - The inverse is the transpose (only rotations)
  - Closed set (composition yields another rotation)
  - Associates (do operation in any order)
  - Does not commute (in general)
  - It is a rigid transformation –> preserves distances and handedness
  - 3D: Center of rotation is an axis
  - normal 2D: about Z axis

- single axis rotations

- lookat transformations – lec15

Only specifies 2 dergees of freedom

Z axis is the direction the camera looks in



- rotations about multiple axes

- Euler Angles

Any rotation can be represented as a sequence of three rotations about a fixed axis

Rotation around 3 fixed axes

Could be any order (XYZ, ZYX, ZXY)

Can repeat (ZXZ)

Can be local or global

- Axis Angle representations

Any rotation can be represented as a single rotation about some axis

rotate around a single axis

Bad: hard to figure out what axis, hard to compose

Rotation vector: Store the angle as the magnitude of the axis

- ## Gimbal Lock and other Euler Angle problems

Gimbal lock

- No matter what X is, Y=90 aligns Z with it
- There is no way to get the Y axis out of the X=0 plane We lost a degree of freedom

Rotate about X then Y == Rotate about Y then Z same! (but different path)

Euler angle good: Easy for 1 axis, Easy for simple combinations

bad:

- Hard to get what you want (unintuitive combinations)
- Can't interpolate
- Gimbal lock (can't get there from here)

- ## Motivations for Quaternions

4-dimensional complex number

State of 3dobject

easy to compose and interpolate

multiplication preserves unit-ness

multiplication composes transformations

JavaScript Tips:

- ## ES6 modules - lec9

```
<script src="1-1.js" type="module" defer></script>
```

```
import { functionGallery } from "./1-curves.js";
export function functionGallery(context, t, tangentScale) {
```

- ## Casts and Type Checks – lec11

```
// Unsafe casts, document what you know
const canvas = ( /** @type {HTMLCanvasElement} */document.getElementById("mycanvas"));

// Explicit Type Checks
let canvas = document.getElementById("mycanvas");
if (!(canvas instanceof HTMLCanvasElement))
  throw new Error("Canvas is not an HTML Canvas Element"); // handle error
```

- ## inheritance and subclasses–lec16

```
class Parent {
  constructor(a,b) {
    this.a = a;
    this.b = b;
    this.c = 10;
  }
  method() {
  console.log(this.a,this.c);
  }
};

class Child extends Parent {
  constructor(b) {
    super(3,b);
    this.c = 20;  // this doesn't exist until super()
  }
}

let thing1 = new Parent(1,2);
thing1.method(); // prints 1,10

let thing2 = new Child(5);
// Child class uses parent methods (unless it overrides them)
thing1.method(); // prints 3,20
```

- ## methods and this – lec15

```
// methods: this does correct
class MyClass {
  constructor(a) {
this.a = a;
}
a2() { console.log(this.a);} }
let inst = new MyClass(5); inst.a1();
```

```
//inner function,
class MyClass {
  constructor(a) {
  this.a = a;
  this.a1 = function() {console.log(this.a);} }
a2() { console.log(this.a);}
}
let inst = new MyClass(5);
inst.a1(); inst.a2();

// in any function as member, defined at call time, not lexical
class MyClass {
  constructor(a) {
  this.a = a;
  this.a1 = function() {console.log(this.a);} }
a2() { console.log(this.a);}
}
let inst = new MyClass(5);
inst.a3 = function() {console.log(this.a);} inst.a1();
inst.a2();
inst.a3();
```

- asynchronous programming (callbacks, promises, await) – lec14

```
import { OBJLoader } from "../libs/CS559-Three/examples/jsm/loaders/OBJLoader.js";
let loader = new OBJLoader();
// function here is callback, to be called when things are ready
loader.load("./objects/07-astronaut.obj", function(astronaut) {
  astronaut.position.set(1.5, 4, 0);
  astronaut.scale.set(0.5, 0.5, 0.5);
  scene.add(astronaut);
  renderer.render(scene, camera);
});
```

```
//loadAsync return a promis, use then after, call when promise is filled
let obj = loader.loadAsync("./objects/07-astronaut.obj");
obj.then(function(astronaut) {
  astronaut.position.set(-2, 4, 0);
  astronaut.scale.set(0.5, 0.5, 0.5);
  scene.add(astronaut);
  renderer.render(scene, camera);
});
```

```
// await, works inside of asynchronous functions
let astro = await loader.loadAsync("./objects/07-astronaut.obj");
console.log(astro);
astro.position.set(-0, 4, -2);
astro.scale.set(0.5, 0.5, 0.5);
scene.add(astro);
renderer.render(scene, camera);
```

- parameter passing with dictionaries (WB7-1)