Drawing in 3D
(viewing, projection, and the rest of the pipeline)

CS559 – Spring 2016
Lecture 6
February 11, 2016
The first 4 Key Ideas...

1. Work in convenient **coordinate systems**. Use **transformations** to get from where you want to be to where you need to be. Hierarchical modeling lets us build things out of pieces.

2. Use **homogeneous coordinates** and transformations to make common operations easy. Translation, projections, coordinate system shift all become simple matrix multiplies.

3. Create **viewing transformations** with **projection**. The geometry of imaging (pinhole camera model) leads to linear transformations in homogeneous coordinates.

4. Implement primitive-based rendering (interactive graphics) with a **pipeline**. The abstractions map nicely onto hardware, and let you do things like visibility computations easily. Be aware that there are other paradigms for drawing.
The first 4 Key Ideas...

1. Work in convenient **coordinate systems**. Use **transformations** to get from where you want to be to where you need to be. Hierarchical modeling lets us build things out of pieces.

2. Use **homogeneous coordinates** and **transformations** to make common operations easy. Translation, projections, coordinate system shift all become simple matrix multiplies.

3. Create **viewing transformations with projection**. The geometry of imaging (pinhole camera model) leads to linear transformations in homogeneous coordinates.

4. Implement primitive-based rendering (interactive graphics) with a **pipeline**. The abstractions map nicely onto hardware, and let you do things like visibility computations easily. Be aware that there are other paradigms for drawing.
ReCap: Basic Idea

To draw we need a coordinate system.

Transformations can

Move between coordinate systems
Move objects around

Useful transformations as matrices
ReCap: Math

Points, Vectors, Matrices
Transformations as linear operators
Matrices (homogeneous coordinates)

3D Coordinate Systems
cross product, right hand rule
ReCap: Code

Matrix stack
• Save
• Transform (concat)
• Draw (current trans)
• Restore

```javascript
this.context.save();
this.context.translate(50,50);
this.context.rotate(this.frontPropAngle);
this.drawProp();
this.context.restore();
```
What about 3D?
Primitive-based Rendering

Draw 2D Objects in the image
   Paint strokes on canvas
   Lines / Triangles on screen

Map (Transform)
3D primitives (world) to 2D primitives (screen)
What does it take to do this?

1. Put a 3D **primitive** in the **World**
2. Figure out what **color** it should be
3. Position relative to the **Eye**
4. Get rid of stuff behind you/offscreen
5. Figure out where it goes on **screen**
6. Figure out if something else blocks it
7. Draw the **2D primitive**
1. Put a 3D primitive in the World
   **Modeling**
2. Figure out what color it should be
   **Shading**
3. Position relative to the Eye
   **Viewing** / Camera Transformation
4. Get rid of stuff behind you/offscreen
   **Clipping**
5. Figure out where it goes on screen
   **Projection** (sometimes called Viewing)
6. Figure out if something else blocks it
   **Visibility** / Occlusion
7. Draw the 2D primitive
   **Rasterization** (convert to Pixels)
1. Put a 3D primitive in the World Modeling
2. Figure out what color it should be Shading
3. Position relative to the Eye Viewing / Camera Transformation
4. Get rid of stuff behind you/offscreen Clipping
5. Figure out where it goes on screen Projection (sometimes called Viewing)
6. Figure out if something else blocks it Visibility / Occlusion
7. Draw the 2D primitive Rasterization (convert to Pixels)

Did some, will do more
A little for P4, lots later
Today (FCG 7)
Not much to say (FCG 8)
Today (FCG 7)
This week (FCG 8)
Not much to say (FCG 8)
In case you’re wondering...
(We’ll come back to this. It’s a detail)

For the kinds of projection we will use...

3D Points map to 2D Points
3D Lines map to 2D Lines
3D Triangles map to 2D Triangles

Doesn’t work for curves (even ellipses)
Viewing

How to get from the object to the screen?

A transformation between coord systems

A little weird…
3D to 2D

Do we lose a dimension?

No – we actually need to keep it
Yes – but we’ll just ignore Z

The screen as a fishtank
A (simple) bird

```javascript
function birdGeometry(ct) {
  "use strict";
  ct.fillStyle = "#88C";
  ct.beginPath();
  ct.arc(0,0,20, 0, 2 * Math.PI, false);
  ct.arc(15,15,10, 0, 2 * Math.PI, false);
  ct.fill();
  ct.fillStyle = "#CC0";
  ct.beginPath();
  ct.moveTo(24,10);
  ct.lineTo(24,20);
  ct.lineTo(32,15);
  ct.closePath();
  ct.fill();
}
```
A Tree for the Bird
In it’s coordinate system

```javascript
function treeGeometry(ct) {
    "use strict";
    ct.strokeStyle = "#8B4513";
    ct.lineWidth = 5;
    ct.beginPath();
    ct.moveTo(30, 0);
    ct.bezierCurveTo(20, 50, 20, 50, 20, 100);
    ct.lineTo(-20, 100);
    ct.bezierCurveTo(-20, 50, -20, 50, -30, 0);
    ct.stroke();
    ct.strokeStyle = "#0000";
    ct.fillStyle = "#DDE";
    ct.beginPath();
    ct.arc(0, 175, 75, 0, 2 * Math.PI, false);
    ct.fill();
    ct.stroke();
}
```
Bird in Tree

```javascript
function drawTree(ct) {
  treeGeometry(ct);
  ct.save();
  birdInTree.apply(ct); // transform
  drawBird(ct);
  ct.restore();
}
```
Bird in Tree, Tree in Park
And a person to look at the bird...
The Eye Coordinate System (or camera)
Look At (the bird)
Perspective (just wait)
Look At (the bird)
Where is everything in the Park?

Points in Park Coords

Tree in Park

Points in Tree Coords

Person in Park

Eye in Person

Points in Person Coords

Eye Point 0

Points in Bird Coords

Bird in Tree

Points in Park Coords
Bird in Eye (Camera) Coordinates

- Tree in Park
- Bird in Tree
- Points in Bird Coords
- Points in Eye Coordinates
- Points in Park Coords
- Person in Park
- Eye in Person
- Points in Person Coords
\[ e = M_e^{-1} M_p^{-1} M_t M_b p \]
\[ e = M_e^{-1} M_p^{-1} M_t M_b p \]
From object to eye: ModelView

Modeling matrix: object to world

Viewing matrix: world to eye / camera
  Rigid Transformation (rotate/translate)

Invert the camera’s model matrix
Build a “LookAt / LookFrom” matrix
Next Problem: Projection

Convert 3D (eye coordinates) to 2D (screen)
A transformation

Types:
  Orthographic
  Perspective
  some others we won’t talk much about
Orthographic Projection

Scale X and Y to fit things on screen

Note: we can look in any direction we are already in camera coordinates!
Orthographic Projections

Simple
Preserves Distances

Objects far away same size as close
Looks weird
Perspective Projections

Objects that are far away look smaller
Perspective Imaging

Eye Point  
a.k.a.  
Focal Point  

Image Plane
Perspective Assumptions

There is a single focal point

Simplifying Assumptions: (not required)
image plane orthogonal to view direction
image plane centered on view direction
Image plane in front of eye
Image plane behind eye

d is the focal length
Pinhole Camera
Perspective math

\[ e \quad g \quad \text{view plane} \quad \downarrow \quad y_s \quad \downarrow \quad y \\
\]

\[ e \quad g \quad \rightarrow \quad d \quad \rightarrow \quad z \]
Perspective math

\[ y_s = \frac{d}{z} y \]