CS559 – Lecture 11
Polygons, Transformations

These are course notes (not used as slides)
Written by Mike Gleicher, Oct. 2005
With some slides adapted from the notes of Stephen Chenney

Final version (after class)

Today

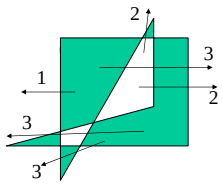
- Polygon rules (since I got them wrong last time)
- Homogeneous coordinates
- Working with transformations / composition
- Hierarchies / Matrix Stacks
- Transformations in 3D
  - Coordinate systems in 3D
  - Rotations
  - Projections (3D->2D)

General Polygons?

- Inside / Outside not obvious for general polygons
- Usually require simple polygons
  - Convex (easy to break into triangles)
- For general case, three common rules:
  - Non-exterior rule: A point is inside if every ray to infinity intersects the polygon
  - Non-zero winding number rule: trace around the polygon, count the number of times the point is circled (+1 for clockwise, -1 for counter clockwise). Odd winding counts = inside (note: I got this wrong in class)
  - Parity rule: Draw a ray to infinity and count the number or edges that cross it. If even, the point is outside, if odd, it's inside (ray can't go through a vertex)

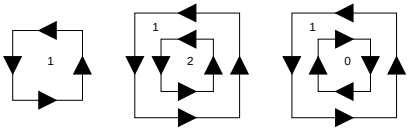
Parity

- Any point, take any ray (that doesn't go through a vertex)
- Odd number of crossings = inside
- Even number of crossings = outside



Power Point uses this rule!

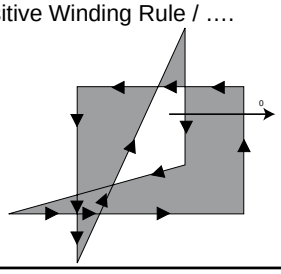
Winding Numbers



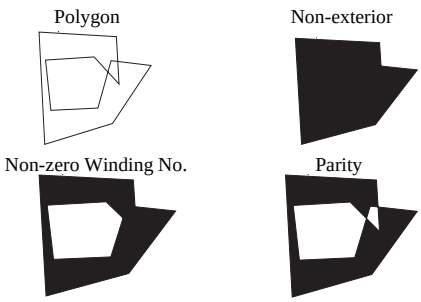
- Count the number of times a point is circled counter clockwise
  - Clockwise counts negative
- Can pick any ray from point and count left/right
  - Right (relative to away direction) = CCW = +1
  - Left = CW = -1

Non-Zero Winding Rule

- Any non-zero winding is "inside"
- What Adobe Illustrator does
- Odd Winding Rule / Positive Winding Rule / ….



Inside/Outside Rules

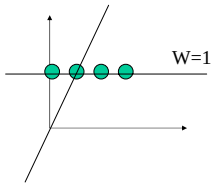


Homogeneous Coordinates

- Big idea for graphics – really important
  - Will be used for several things – translation is just 1
- Basic idea: add an extra coordinate
  - 2D becomes 3D (3x3 matrices)
  - 3D becomes 4D (4x4 matrices)
- Convert "back" from homogeneous coordinates by division
  - (x,y) -> (x,y,1)
  - (x,y,w) -> (x/w, y/w)
- Projection
  - Many points in higher dim space = 1 point in lower dim space
- For now, just make w=1

Homogeneous Coordinates

- "Normal" space is a subspace
  - W = 1
- Think about 1D case (so embed into 2D x,w)
- Many equivalent points (projection)

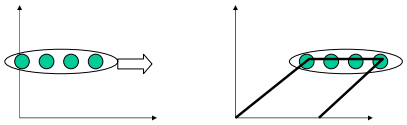


Only 1D Linear operation is scale

(about origin)

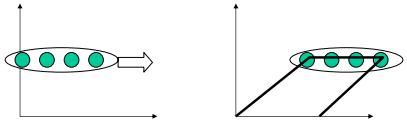
Translation in Homogeneous Coords

- 1D Translation = 2D Skew



Translation in Homogeneous Coords

- Translate in 2D = Skew in 3D
  - Deck of cards



$$trans(x,y) = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

What about other linear ops

- Just add an extra coordinate
- Don't change w (unless you know what you're doing)

$$scale(s) = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$rotate(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Coordinates

- Makes translation (affine transforms) linear
- Need to work in higher dimensional space

- Useful for lots of other things
  - Viewing (perspective)

Matrices as Coordinate Systems

- Where does X axis go?
- Where does Y axis go?
- Where does origin go?

- Assumes that bottom row is [0 0 1]

- Can you scale by changing w?
  - Yes, but often we prefer to renormalize so bottom right number is 1

Composing Transformations

- Order matters!
  - Scale / rotate vs. rotate/scale

- Can implement by multiplying matrices
  - $T_1\ T_2\ T_3\ \mathbf{x} = (T_1\ T_2\ T_3)\ \mathbf{x}$

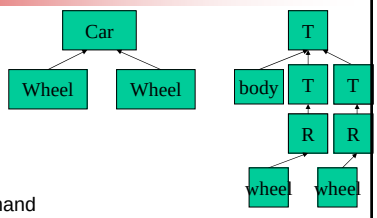
Why Compose?

- Rotate about a point
  - $T_c\ R\ T_{-c}\ x$
- Scale along an axis
  - Move point to origin
  - Align axis w/major axis
  - Scale
  - Put things back
  - $T_c\ R_\theta\ S\ R_{-\theta}\ T_{-c}\ x$

Hierarchical coordinate Systems

- Car
  - Wheel
  - Wheel

- Person
  - Head / Neck
  - Arm / forearm / hand



Matrix Stack

- Multiply things onto the top
- Top is “current” coordinate system
- Push (copy the top) if you’ll come back
- Pop to go back

- Think about it as moving the coordinate system
- Top of stack is “current coordinate system”
  - Where we will draw
- Transformations change current coord system
  - Or change the objects that we are going to draw

Matrix Stack Example

- Draw Car = …. Push trans wheel pop …
- Push trans – draw car – pop push trans – draw car

3D

- 3D coordinate system & handedness
- Prefer right-handed coordinate systems
- Right-hand rule

What happens in 3D?

- 4D Homogeneous Points
  - 4x4 matrices
- Basic transforms are the same
  - Translate
  - Scale
  - Skew
- Rotation is different
  - Rotation in 3D is more complicated?

What is a rotation?

- A transformation that:
  - Preserves distances between points
  - Preserves the zero
  - Preserves "handedness" (in 2D clockwiseness)
- A subset of linear transformations
- Some things that come out of these:
  - Axes remain perpindicular
  - Axes remain unit length
  - Cross product holds

Parameterizing Rotations

- Rotations are Linear Transformations
  - 2x2 matrix in 2D
  - 3x3 matrix in 3D
- The set of rotations = set of OrthoNormal Matrices
- Inconvenient way to deal with them
  - Can't work with them directly
  - Not stable (small change makes it not a rotation)
- Is there an easier way to parameterize the set?

Measuring rotation in 2D

- Pick 1 point (1,0)
- Any rotation must put this on a circle
- If you know where this point goes, can figure out any other point
  - Distances (w/point & origin) + handedness says where things go
- Parameterize rotations by distance around circle
  - Angle
- Issues with wrap around
  - Many different angles = same rotation

Much harder in 3D

- Any point can go to a sphere
- That one point doesn't uniquely determine things

- No vector in R^n can compactly represent rotations
  - Singularities
    - nearby rotations / far away numbers
    - Nearby numbers / far away rotations
- Hairy-Ball Theorem
  - Any parameterization of 3D rotations in R^n will have singularities

Representation of 3D Rotations

- Two Theorems of Euler
  - Any rotation can be represented by a single rotation about an arbitrary axis (axis-angle form)
  - Any rotation can be represented by 3 rotations about fixed axes (Euler Angle form)
    - XYZ, XZX, any non-repeating set works
    - Each set is different (gets different singularities)

- Building rotations
  - Pick a vector (for an axis)
  - Pick another perpindicular vector (or make one w/cross product)
  - Get third vector by cross product

Euler Angles

- Pick convention
  - Are axes local or global?
  - Local: roll, pitch, yaw
  - What order?
- Apply 3 rotations
- Good news: 3 numbers
- Bad news:
  - Can't add, can't compose
  - Many representations for any rotation
  - Singularities