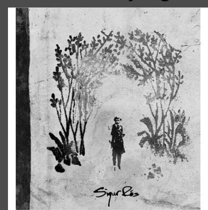
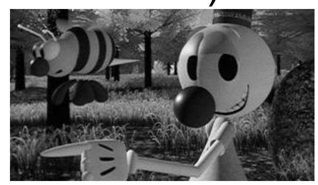
Now Playing:



Gong
Sigur Rós
From *Takk...*
Released September 13, 2005

Andre and Wally B (Lucasfilm CG Project [later Pixar], 1984)



2D Transforms

Rick Skarbez, Instructor
COMP 575
September 4, 2007

Last Time

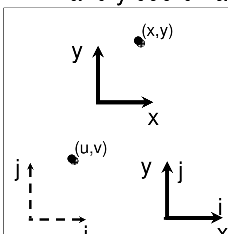
- Reviewed the math we're going to use in this course
  - Points
  - Vectors
  - Matrices
  - Linear interpolation
  - Rays, planes, etc.

Today

- Vector spaces and coordinate frames
- Transforms in 2D
- Composing Transforms

Vector Spaces

- Let's think for a minute about what x and y coordinates really mean



$$\mathbf{i} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

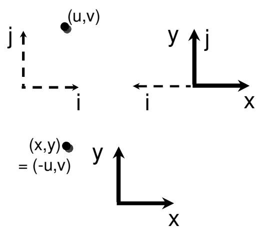
$$\mathbf{j} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Vector Spaces

- For illustration, let's flip things around



$$\mathbf{i} = \begin{bmatrix} -1 & 0 \end{bmatrix}$$

$$\mathbf{j} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Vector Spaces

- Any pair of non-parallel, non-anti-parallel vectors can define a vector space in 2D
  - We're used to thinking about spaces defined by orthogonal, normalized, axis-aligned vectors
  - But there's no reason this is the only way to do things

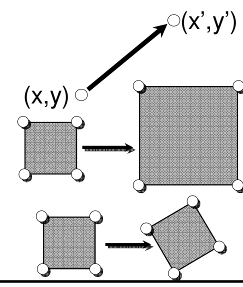
Vector Space Terms

- <u>Linear Vector Space</u>: Any space made up of vectors and scalars
- <u>Euclidean Space</u>: Vector space with a distance metric
- <u>Affine Space</u>: Vector space with an origin

- The Cartesian plane is both *affine* and *Euclidean*
  - We call this type of space a frame

2D Transforms

- What am I talking about when I say "transforms"?
  - Translation
  - Scaling
  - Rotation



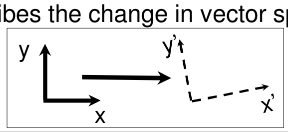
General Form

- The transformations we consider are of the following form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Transformation Matrix

- Remember that the transformation matrix describes the change in vector space:

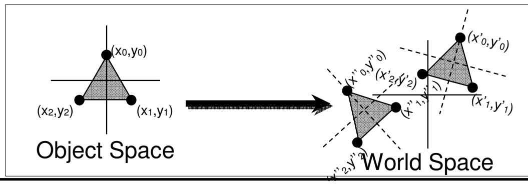


Object vs. World Space

- Let's stop and think about why we're doing this...
- We can define the points that make up an object in "object space"
  - Whatever is most convenient, often centered around the origin
- Then, at run time, we can put the objects where we want them in "world space"

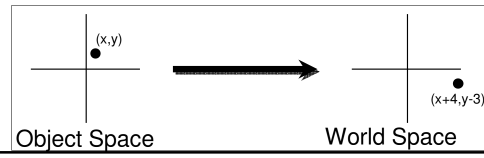
Object vs. World Space

- Makes building large models easier
- Example:



Translation

- Basically, just moving points
  - In 2D, up, down, left, or right
  - All points move in the same way
- For example, we may want to move all points 4 pixels to the right and 3 down:



So How Do We Do It?

- What transformation matrix will add 4 to x and subtract 3 from y?
  - That is, what are the values of a, b, c, and d needed for this transformation?

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Transformation Matrix

- Actually, this is impossible to do with a 2x2 matrix and 2-vectors

How Do We Do It?

- Option 1: Implement translation as a 2-step process

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

e is the x-offset
f is the y-offset

- What are the values for our example?

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 4 \\ -3 \end{bmatrix}$$

So How Do We Do It?

- Option 2: Use bigger matrices

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- If we set w = 1, then

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

c is the x-offset
f is the y-offset

How We Do It

- This is the way we'll normally do it
- However, in computer science, we really like square matrices, so it'll be written as:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

So what does this w stand for?

Homogeneous Coordinates

- We refer to this as a homogeneous coordinate: $\begin{bmatrix} x \\ y \\ w \end{bmatrix}$
- This mathematical construct allows us to
- Represent affine transforms with a single matrix
- Do calculations in projective space (vectors are unique only up to scaling)

Homogeneous Coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- For points, w must be non-zero
  - If w=1, the point is "normalized"
  - If w!=1, can normalize by

$$\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{w}{w} \end{bmatrix}$$

Vectors in Homogeneous Coordinates

- Remember last time, I mentioned that it would be useful that we could represent points and vectors the same way?
  - Here's the payoff
- Can use homogeneous coordinates to represent vectors, too
  - What is w?
    - Remember, vectors don't have a "position"

Vectors in Homogeneous Coordinates

- Since vectors don't have a position, they should not be affected by translation
  - What about rotation/scaling?
- Set w=0 for vectors:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

These have no effect (the column c, f, i)

Summing up Translation

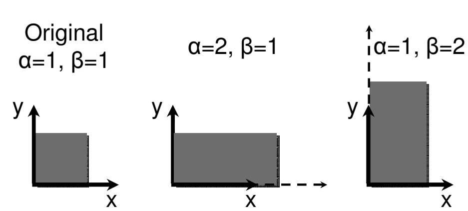
- We will represent translation with a matrix of the following form:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & u \\ 0 & 1 & v \\ 0 & 0 & 1 \end{bmatrix}$$

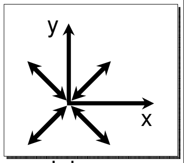
u is the x-offset
v is the y-offset

Scaling

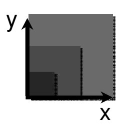
- Want to stretch or shrink the entire space in one or more dimensions
- α = stretch in x, β = stretch in y

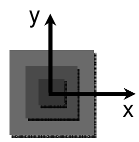


Scaling



- Scaling is centered around the origin
  - Points either get pulled toward the origin or pushed away from it





Scaling

- So, given what we know, how would we construct a scaling matrix?
  - Assume we have an object centered around the origin, and want to scale it by α in x, and β in y
    - x' = αx
    - y' = βy

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

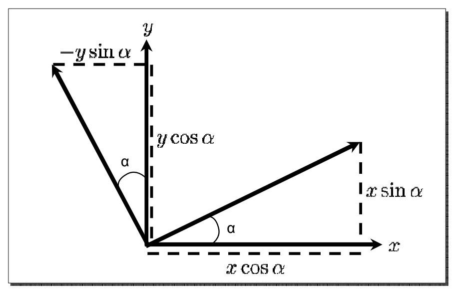
Summing up Scaling

- We will represent scaling with a matrix of the following form:

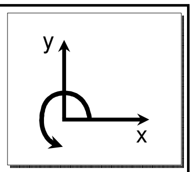
$$\mathbf{M} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

α is the scale factor in the x-direction
β is the scale factor in the y-direction

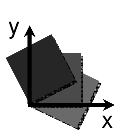
Rotation

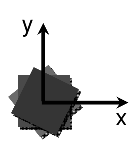


Rotation



- Like scaling, rotations are centered about the origin





Rotation

$$i = \begin{cases} x' = x \cos \alpha \\ y' = x \sin \alpha \end{cases}$$

$$j = \begin{cases} x' = -y \sin \alpha \\ y' = y \cos \alpha \end{cases}$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Summing up Rotation

- We will represent rotation with a matrix of the following form:

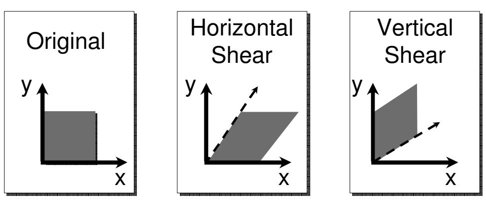
$$\mathbf{M} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

α is the angle of rotation (counter-clockwise)

Anything else?

- Translation, scaling, and rotation are the three most common transforms
- What do they have in common?
  - They maintain the orthogonality of the coordinate frame
- What happens if we relax this constraint?

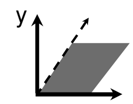
Shearing



Shearing

Horizontal Shear

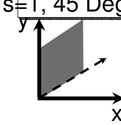
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$



s=0, No Shear
s=1, 45 Degree Shea

Vertical Shear

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ s & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$



s is -tan(θ), where θ is the desired shear angle

Transforms Summary

- Discussed how to do 4 common transforms in 2D
  - Translation
  - Scaling
  - Rotation
  - Shearing
- Also took a detour to discuss homogeneous coordinates

Composing Transforms

- These are the basics
- However, single transforms aren't really very interesting
- The real power comes from using multiple transforms simultaneously
  - That's what we're going to do now

Composing Transforms

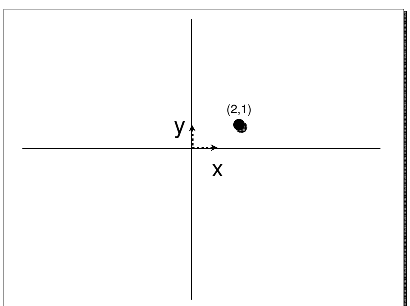
- So why did we go through the trouble to use homogeneous coordinates for our points, and do our transforms using square transformation matrices?
  - To make composing transforms easy!
- Composing 2 transforms is just multiplying the 2 transform matrices together

WARNING: The order in which matrix multiplications are performed may (and usually does) change the result! (i.e. they are not commutative)

Let's do an example

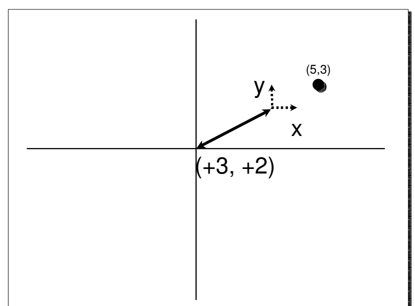
- Two transforms:
  - Scale x and y by a factor of 2
  - Translate points (+3, +2)
- Let's pick a single point in object space
  - (1, 2)

Two Transform Paths



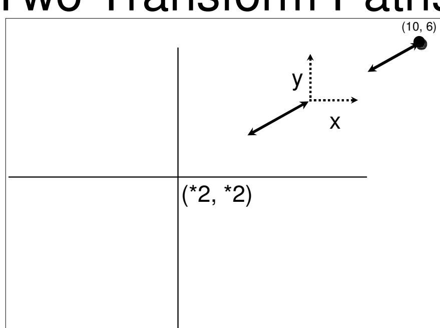
Translate Then Scale

Two Transform Paths



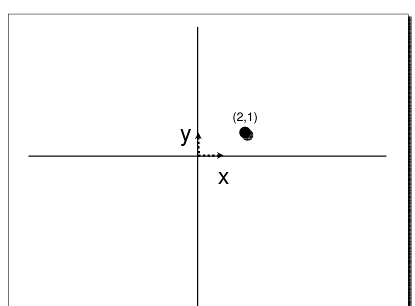
Translate Then Scale

Two Transform Paths



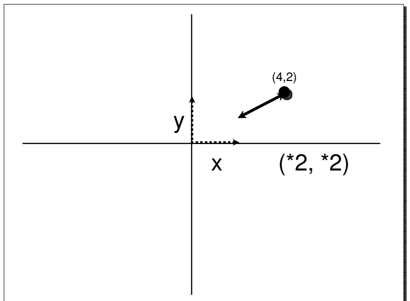
Translate Then Scale

Two Transform Paths



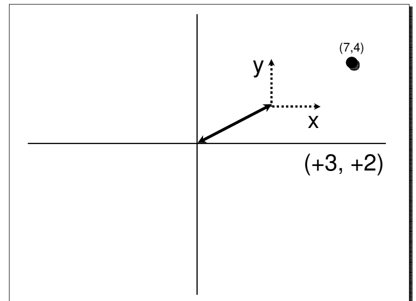
Scale Then Translate

Two Transform Paths



Scale Then Translate

Two Transform Paths



Scale Then Translate

Composing Transforms

- Translate then scale
  - (x', y') = (10, 6)
- Scale then translate
  - (x', y') = (7, 4)

- Need a standard way to order transforms!

Transform Matrix Multiplications

$$\mathbf{P}' = \mathbf{M_3 M_2 M_1 P}$$

- Always compose from right to left
  - Here, transform M_1 is applied first
  - Transform M_3 is applied last

$$\mathbf{P}' = \mathbf{M_3}\left(\mathbf{M_2}\left(\mathbf{M_1}\left(\mathbf{P}\right)\right)\right)$$

Two Transform Paths

Translate then Scale

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 2 & 0 & 6 \\ 0 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Two Transform Paths

Scale then Translate

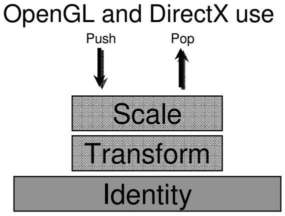
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

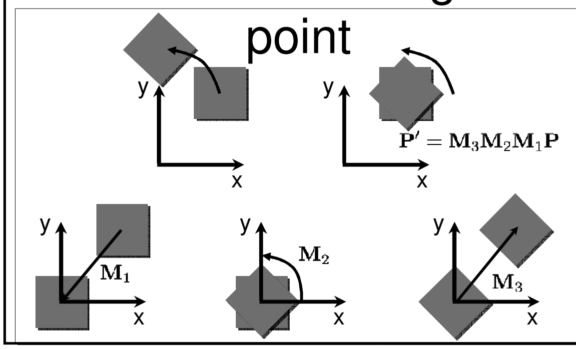
Composing Transforms

- We can go one step further than just multiplying matrices
- Specifically, we can use a matrix stack
  - This is what OpenGL and DirectX use

You can build it in either order (world to object or object to world), then multiply out when needed

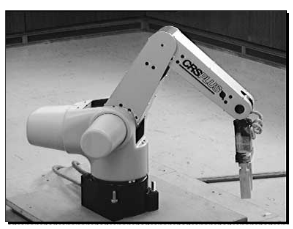


Application: Rotation about a non-origin point



Robotic Arm Example

- Fingers first
- Then wrist
- Then elbow
- Finally, shoulder



Next Time

- Going to talk a bit more about transforms, and in particular, 3D transforms
- Might talk a little bit about animation