Today: 2-D Transformations

Transformations are functions applied to points in space

$$\mathbf{p}' = f(\mathbf{p})$$

Provide a mechanism for manipulating geometric models

Transformations are *essential* pieces of graphics systems

- OpenGL and PostScript, for instance, use them extensively

Why Do We Need Transformations?

Makes modeling more convenient

- for example, often easier to generate models around origin
  - gluSphere() draws a sphere of radius r about the origin
- can then move them to final position with transformations

Model viewing process via transformations

- projecting 3-D to 2-D will be done this way

Animation

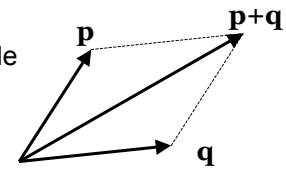
- transformations as a function of time creates motion

Linear Algebra in 30 Seconds

We represent points as vectors $\mathbf{p} = [x\ y\ z]$

- vectors add according to parallelogram rule
- a linear combination of two vectors is

$$\alpha\mathbf{p} + \beta\mathbf{q}$$



- set of vectors is linearly independent if none is a linear combination of the others
- a basis for a space is a linearly independent set of vectors whose linear combinations include all vectors in the space

standard basis for 2-D plane: $\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

- but there are infinitely many possible bases

Linear & Affine Transformations

We'll be specifically interested in linear transformations

$$f(\alpha\mathbf{p} + \beta\mathbf{q}) = \alpha f(\mathbf{p}) + \beta f(\mathbf{q})$$

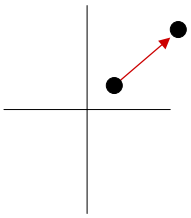
- transformation of shape determined by effect on vertices
- a crucial property that allows for efficient implementation

And the related class of affine transformations

$$f(\mathbf{p} + \alpha(\mathbf{q} - \mathbf{p})) = f(\mathbf{p}) + \alpha f(\mathbf{q} - \mathbf{p})$$

- preserves affine combinations (e.g., they map lines to lines)
- another view: a linear transformation + a translation
- this is a more general class of functions

Translation



Offset all points by constant amount

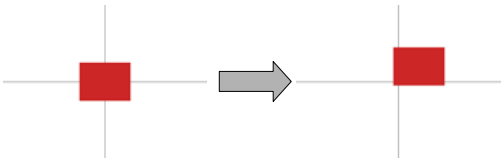
$$x' = x + \Delta x$$
$$y' = y + \Delta y$$

Written as more concise vector equation

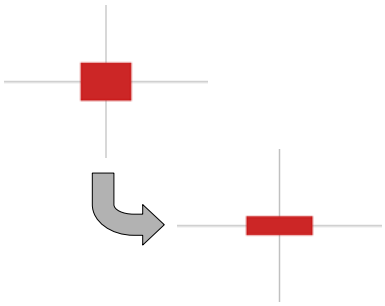
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

or

$$\mathbf{p}' = \mathbf{p} + \mathbf{d}$$



Scaling



Scale all points by constant amount

$$x' = sx$$
$$y' = ty$$

And written as a vector equation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

or

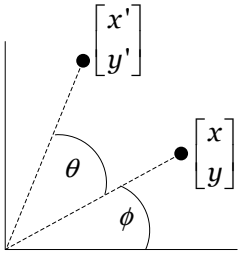
$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$

Squash/stretch along x & y axes

Rotation of Points About Origin

First, write points in polar coordinates

$$x = \rho\cos\phi \qquad y = \rho\sin\phi$$
$$x' = \rho\cos(\phi+\theta) \qquad y' = \rho\sin(\phi+\theta)$$



And solve for the new positions

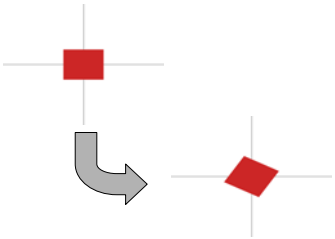
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

Can write this as a vector equation as well

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

or

$$\mathbf{p}' = \mathbf{R}\mathbf{p}$$



The Three Fundamental Transformations

Translation:	$\mathbf{p}' = \mathbf{p} + \mathbf{d}$
Scaling:	$\mathbf{p}' = \mathbf{S}\mathbf{p}$
Rotation:	$\mathbf{p}' = \mathbf{R}\mathbf{p}$

We can represent any affine transformation as a sequence of these 3

Translation is only one not represented as matrix multiplication

- because it's not a linear transformation
- wouldn't it be nice if we could come up with a matrix formulation!

Homogeneous Coordinates

Let's add an extra dimension to our vectors

position: $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ direction: $\begin{bmatrix} a \\ b \end{bmatrix} \rightarrow \begin{bmatrix} a \\ b \\ 0 \end{bmatrix}$

- this added dimension is the homogeneous coordinate
- in general, we'll have coordinates $[x\ y\ w]$
- the resulting 3-D space is a projective space

To convert back, divide by w and drop the last coordinate

- all vectors $[\alpha x\ \alpha y\ \alpha w]$ represent the same point
- if w=0, it represents a "point at infinity"

So what does this do to our transformation equations?

Transforms in Homogeneous Coordinates

Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or

$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$

Rotation

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

or

$$\mathbf{p}' = \mathbf{R}\mathbf{p}$$

Translation

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

or

$$\mathbf{p}' = \mathbf{T}\mathbf{p}$$

Fundamental Homogeneous Transforms

Translation: $\mathbf{p}' = \mathbf{Tp}$

Scaling: $\mathbf{p}' = \mathbf{Sp}$

Rotation: $\mathbf{p}' = \mathbf{Rp}$

Now we can write all three transforms as matrix multiplications

In general, we'll be using some sequence of transformations

$$\mathbf{M}_1(\mathbf{M}_2(\cdots\mathbf{M}_n(\mathbf{v})\cdots))$$

Composing Transformations

Matrix multiplication is associative (but *not* commutative)

- can collapse sequences of transformations into single matrix
- but must *not* reorder any of them

$$\mathbf{A}(\mathbf{B}(\mathbf{C}(\mathbf{D}(\mathbf{v})))) = \mathbf{ABCDv}$$
$$= (\mathbf{ABCD})\mathbf{v}$$

Can choose to multiply pairs at either end

- add new matrices at beginning — pre-multiply

$$\mathbf{ABCD} = \mathbf{A}(\mathbf{B}(\mathbf{CD}))$$

- add new matrices at the end — post-multiply

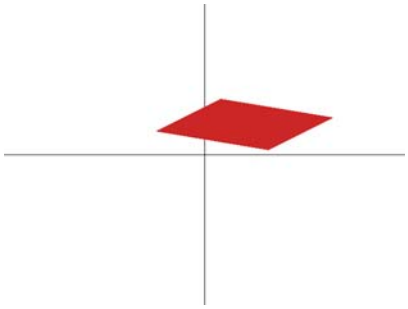
$$\mathbf{ABCD} = (((\mathbf{AB})\mathbf{C})\mathbf{D})$$

Exercise: Composing Transformations

original | **R** = rotate(60°) | **S** = scale(1.3, 0.5) | **T** = trans(0.2, 0.2)

What order of **R**, **S**, **T** will produce this figure?

(a) **TRSv**
(b) **RSTv**
(c) **TSRv**
(d) **RTSv**



Writing Transformations in OpenGL

OpenGL maintains a current transformation matrix M

- issue commands to post-multiply matrices into **M**
- so, commands are listed in *reverse order of application*
- glLoadIdentity() — set **M** to identity matrix $(M \leftarrow I)$
- glTranslatef(Δx, Δy, Δz) — translation $(M \leftarrow MT)$
- glRotatef(θ, x, y, z) — rotate about given axis $(M \leftarrow MR)$
- glScalef(r, s, t) — scale by given factors $(M \leftarrow MS)$

Example: to rotate about an arbitrary point p = [*x y*]

```
glTranslatef(x, y, 0);     // (3) move p back
glRotatef(theta, 0, 0, 1); // (2) rotate around z axis
glTranslatef(-x, -y, 0);   // (1) move p to origin
DrawSomething();
```

A Word of Caution on Notation

We've consistently written points as column vectors

- virtually everyone does this
- typically represent matrices in row-major order
  [[a b] [c d]]

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

Some in graphics have traditionally used row vectors

- convert by transposing everything:
  $\mathbf{ABv} \rightarrow (\mathbf{ABv})^T = \mathbf{v}^T \mathbf{B}^T \mathbf{A}^T$
- note that order is reversed as well
- typically represent matrices in column-major order
  [[a c] [b d]]
- OpenGL actually does it this way
- but you'll probably never notice

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

Next Time: 3-D Transformations

We'll extend all this to 3-D

The mechanisms are largely the same

But that extra dimension complicates things

And just to remind you:

- Solid understanding of transforms is *essential* for a lot material