

Affine Transformations

CSE 457

Reading

Required:

- ♦ Angel 3.1, 3.7-3.11

Further reading:

- ♦ Angel, the rest of Chapter 3
- ♦ Foley, et al, Chapter 5.1-5.5.
- ♦ David F. Rogers and J. Alan Adams,
Mathematical Elements for Computer Graphics, 2nd Ed., McGraw-Hill, New York, 1990, Chapter 2.

Geometric transformations

Geometric transformations will map points in one space to points in another: $(x', y', z') = \mathbf{f}(x, y, z)$.

These transformations can be very simple, such as scaling each coordinate, or complex, such as non-linear twists and bends.

We'll focus on transformations that can be represented easily with matrix operations.

Vector representation

We can represent a **point**, $\mathbf{p} = (x,y)$, in the plane or $\mathbf{p}=(x,y,z)$ in 3D space

- ◆ as column vectors $\begin{bmatrix} x \\ y \end{bmatrix}$ $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$

- ◆ as row vectors $\begin{bmatrix} x & y \end{bmatrix}$
 $\begin{bmatrix} x & y & z \end{bmatrix}$

Canonical axes

Vector length and dot products

Vector cross products

Representation, cont.

We can represent a **2-D transformation** M by a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

If \mathbf{p} is a column vector, M goes on the left:

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

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

If \mathbf{p} is a row vector, M^T goes on the right:

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

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

We will use **column vectors**.

Two-dimensional transformations

Here's all you get with a 2 x 2 transformation matrix M :

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

So:

$$x' = ax + by$$

$$y' = cx + dy$$

We will develop some intimacy with the elements $a, b, c, d...$

Identity

Suppose we choose $a=d=1$, $b=c=0$:

- ◆ Gives the **identity** matrix:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- ◆ Doesn't move the points at all

Scaling

Suppose we set $b=c=0$, but let a and d take on any *positive* value:

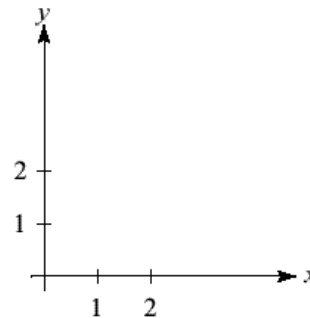
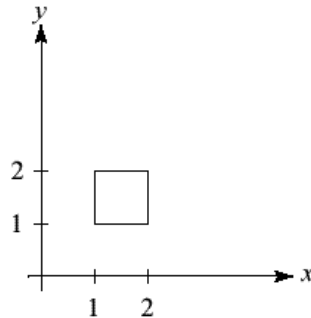
- ◆ Gives a **scaling** matrix:

$$\begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}$$

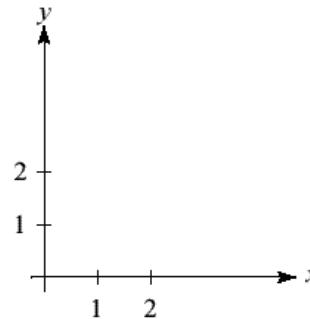
- ◆ Provides **differential (non-uniform) scaling** in x and y :

$$x' = ax$$

$$y' = dy$$



$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

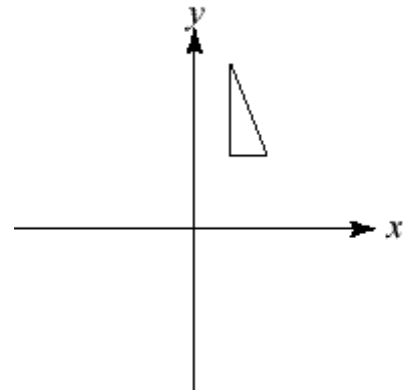


$$\begin{bmatrix} 1/2 & 0 \\ 0 & 2 \end{bmatrix}$$

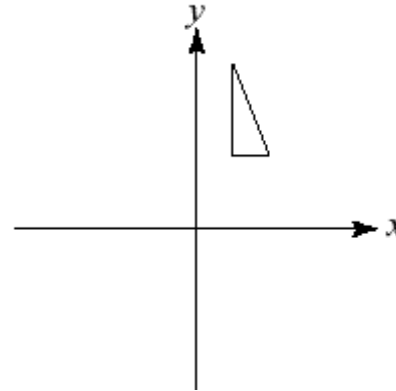
Suppose we keep $b=c=0$, but let either a or d go negative.

Examples:

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$



Now let's leave $a=d=1$ and experiment with b

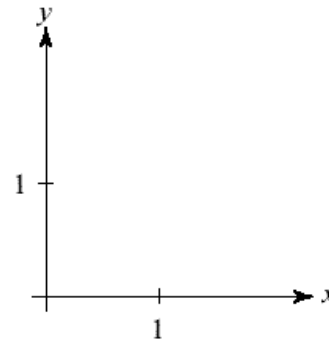
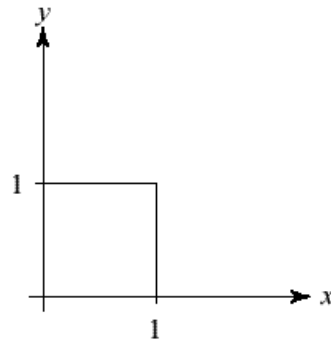
The matrix

$$\begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix}$$

gives:

$$x' = x + by$$

$$y' = y$$



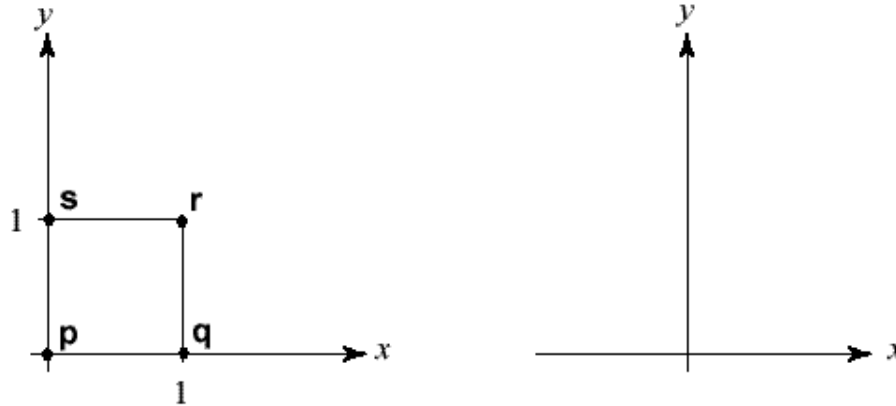
$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Effect on unit square

Let's see how a general 2 x 2 transformation M affects the unit square:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} p & q & r & s \end{bmatrix} = \begin{bmatrix} p' & q' & r' & s' \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & a & a+b & b \\ 0 & c & c+d & d \end{bmatrix}$$



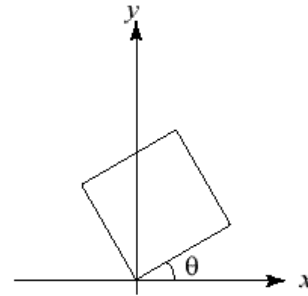
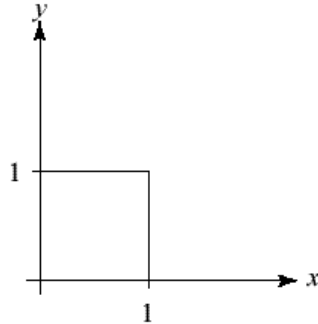
Effect on unit square, cont.

Observe:

- ◆ Origin invariant under M
- ◆ M can be determined just by knowing how the corners $(1,0)$ and $(0,1)$ are mapped
- ◆ a and d give x - and y -scaling
- ◆ b and c give x - and y -shearing

Rotation

From our observations of the effect on the unit square, it should be easy to write down a matrix for “rotation about the origin”:



$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow$$

Thus,

$$M = R(\theta) = \begin{bmatrix} & \\ & \end{bmatrix}$$

Limitations of the 2 x 2 matrix

A 2 x 2 linear transformation matrix allows

- ◆ Scaling
- ◆ Rotation
- ◆ Reflection
- ◆ Shearing

Q: What important operation does that leave out?

Homogeneous coordinates

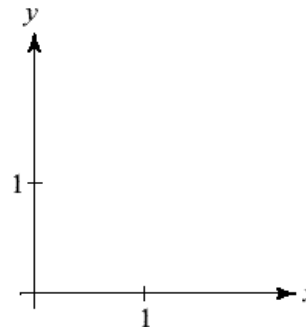
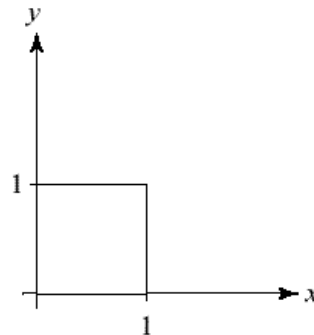
Idea is to loft the problem up into 3-space, adding a third component to every point:

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Adding the third “ w ” component puts us in **homogenous coordinates**.

And then transform with a 3 x 3 matrix:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = T(\mathbf{t}) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

... gives **translation!**

Anatomy of an affine matrix

The addition of translation to linear transformations gives us **affine transformations**.

In matrix form, 2D affine transformations always look like this:

$$M = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{cc|c} \mathbf{A} & & \mathbf{t} \\ \hline 0 & 0 & 1 \end{array} \right]$$

2D affine transformations always have a bottom row of [0 0 1].

An “affine point” is a “linear point” with an added w -coordinate which is always 1:

$$\mathbf{p}_{\text{aff}} = \begin{bmatrix} \mathbf{p}_{\text{lin}} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

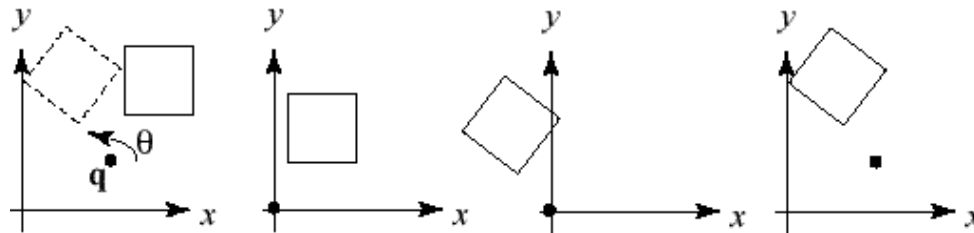
Applying an affine transformation gives another affine point:

$$M\mathbf{p}_{\text{aff}} = \begin{bmatrix} \mathbf{A}\mathbf{p}_{\text{lin}} + \mathbf{t} \\ 1 \end{bmatrix}$$

Rotation about arbitrary points

Until now, we have only considered rotation about the origin.

With homogeneous coordinates, you can specify a rotation, q , about any point $\mathbf{q} = [q_x \ q_y]^T$ with a matrix:



1. Translate \mathbf{q} to origin
2. Rotate
3. Translate back

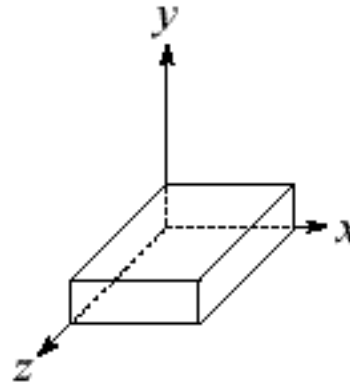
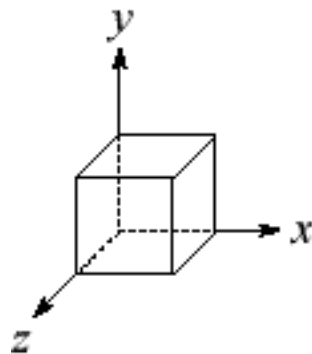
Note: Transformation order is important!!

Basic 3-D transformations: scaling

Some of the 3-D transformations are just like the 2-D ones.

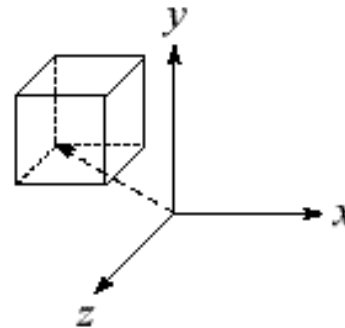
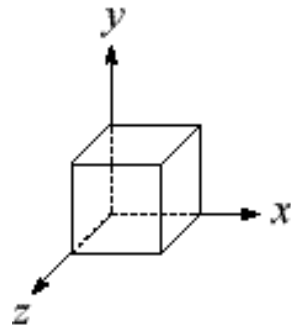
For example, scaling:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Translation in 3D

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



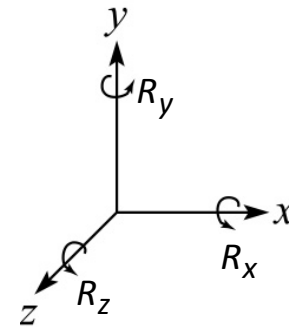
Rotation in 3D (cont'd)

These are the rotations about the canonical axes:

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

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

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



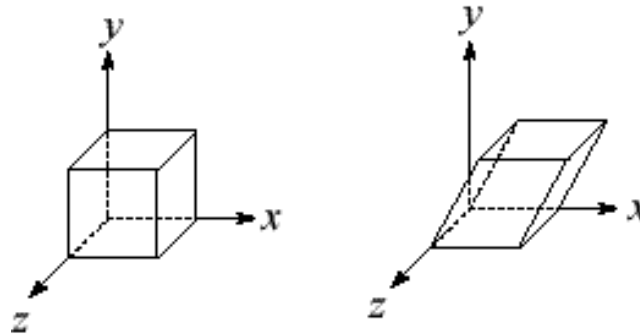
Use right hand rule

A general rotation can be specified in terms of a product of these three matrices. How else might you specify a rotation?

Shearing in 3D

Shearing is also more complicated. Here is one example:

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

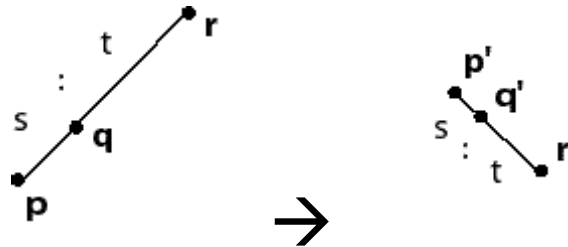


We call this a shear with respect to the x-z plane.

Properties of affine transformations

Here are some useful properties of affine transformations:

- ◆ Lines map to lines
- ◆ Parallel lines remain parallel
- ◆ Midpoints map to midpoints (in fact, ratios are always preserved)



$$\text{ratio} = \frac{\|pq\|}{\|qr\|} = \frac{s}{t} = \frac{\|p'q'\|}{\|q'r'\|}$$

Affine transformations in OpenGL

OpenGL maintains a “modelview” matrix that holds the current transformation **M**.

The modelview matrix is applied to points (usually vertices of polygons) before drawing.

It is modified by commands including:

- ♦ `glLoadIdentity()` **M** ← **I**
– set **M** to identity
- ♦ `glTranslatef(tx, ty, tz)` **M** ← **MT**
– translate by (*t_x*, *t_y*, *t_z*)
- ♦ `glRotatef(θ, x, y, z)` **M** ← **MR**
– rotate by angle *θ* about axis (*x*, *y*, *z*)
- ♦ `glScalef(sx, sy, sz)` **M** ← **MS**
– scale by (*s_x*, *s_y*, *s_z*)

Note that OpenGL adds transformations by *postmultiplication* of the modelview matrix.

Summary

What to take away from this lecture:

- ◆ All the names in boldface.
- ◆ How points and transformations are represented.
- ◆ How to compute lengths, dot products, and cross products of vectors, and what their geometrical meanings are.
- ◆ What all the elements of a 2×2 transformation matrix do and how these generalize to 3×3 transformations.
- ◆ What homogeneous coordinates are and how they work for affine transformations.
- ◆ How to concatenate transformations.
- ◆ The mathematical properties of affine transformations.