



Animation

Geometric Transformations

CS 415: Game Development

Professor Eric Shaffer

Squash & Stretch

From: John Lasseter: "Principles of Traditional Animation Applied to 3-D Computer Animation"
Proc. SIGGRAPH 87

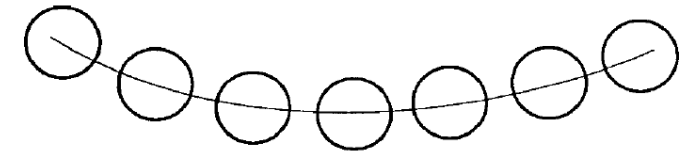
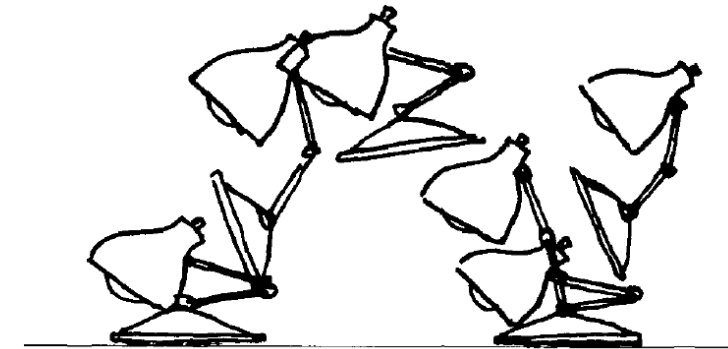
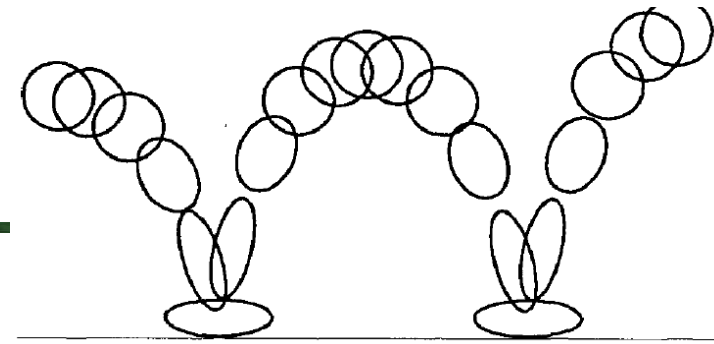
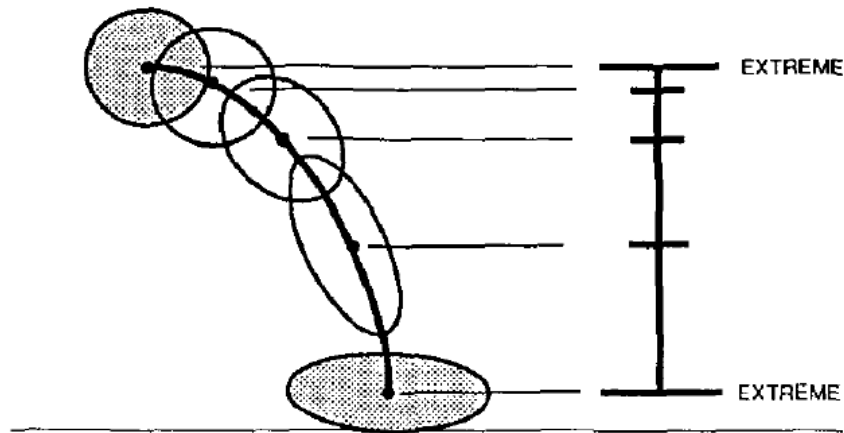


FIGURE 4b. Strobing occurs in a faster action when the object's positions do not overlap and the eye perceives separate images.

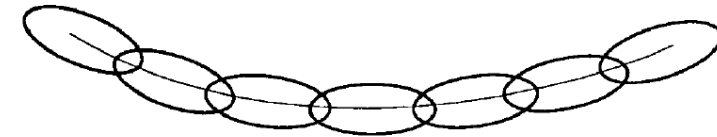


FIGURE 4c. Stretching the object so that its positions overlap again will relieve the strobing effect.

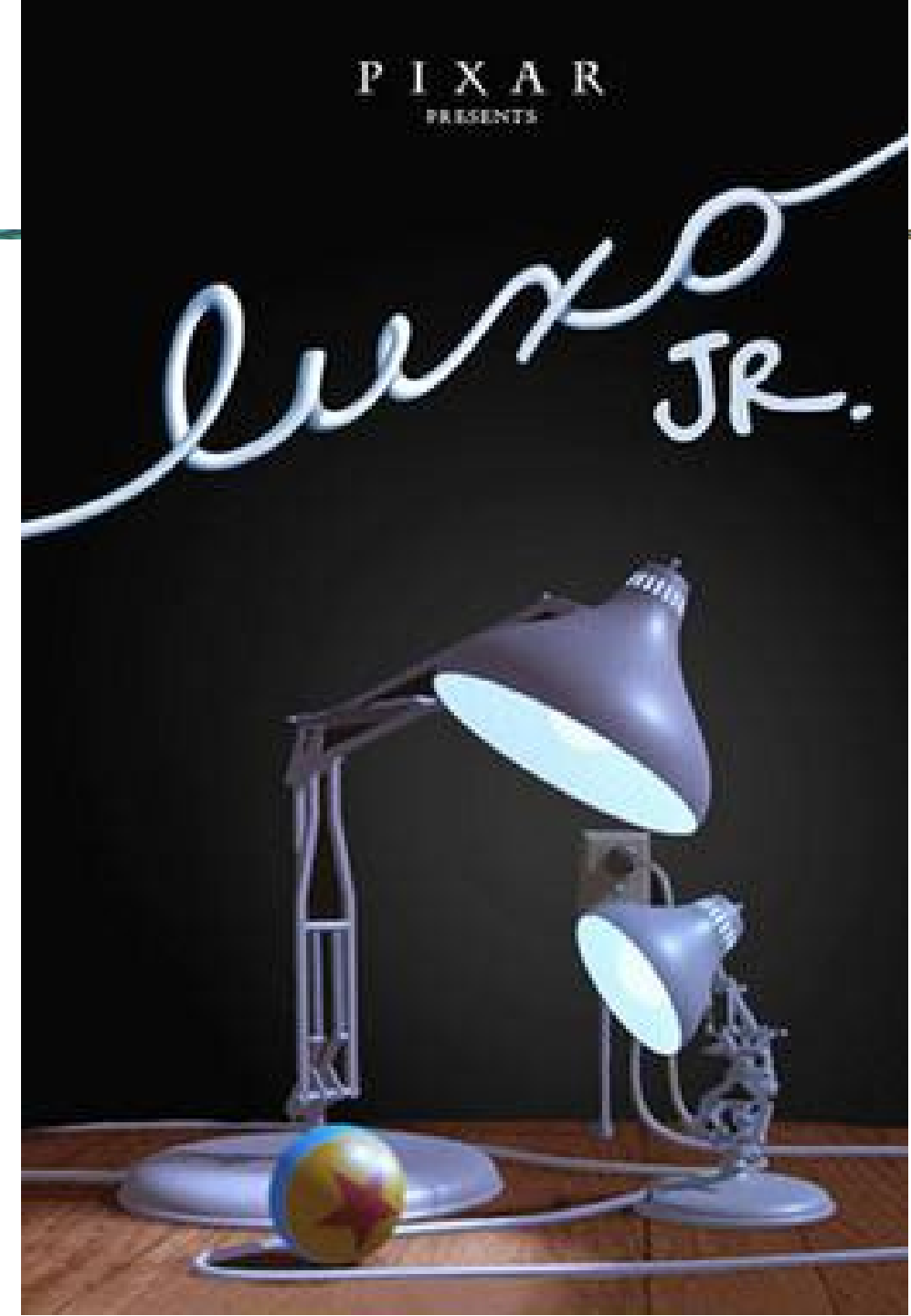
Luxo Jr.

Directed by	John Lasseter
Written by	John Lasseter
Production company	Pixar
Release date	Aug. 17, 1986 (SIGGRAPH)



Luxo Jr.

- Breakthrough demonstration of computer animation in 1987
- Demonstrates shadow maps and Renderman surface shaders
 - Lamp and ball behavior implemented using affine transformations
- Took about 5 months of work from Pixar's (then) tiny animation team
- It was the first [CGI](#) film nominated for an Academy Award.
- In 2014, *Luxo Jr.* was deemed "culturally, historically, or aesthetically significant" by the [Library of Congress](#) and selected for preservation in the [National Film Registry](#).



Basic Idea...

$$\begin{bmatrix} d & e & f & a \\ g & h & i & b \\ j & k & l & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- The geometry of a mesh is defined by the vertex set
- Applying a transformation to a mesh means transforming each vertex location
- Simple transformations can be accomplished by multiplying a matrix times the vertex location
 - Rotation
 - Scale
 - Translation
- 3 dimensional transformations are implemented as a 4x4 matrix
 - We express the 3D position of a point using homogeneous coordinates...add a fourth w coordinate
 - Each 3D position (x,y,z) corresponds to a homogeneous point (x,y,z,1)



3-D Affine Transformations

General Form (with homogeneous coordinates)

$$\begin{bmatrix} d & e & f & a \\ g & h & i & b \\ j & k & l & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} dx + ey + fz + a \\ gx + hy + iz + b \\ jx + ky + lz + c \\ 1 \end{bmatrix}$$

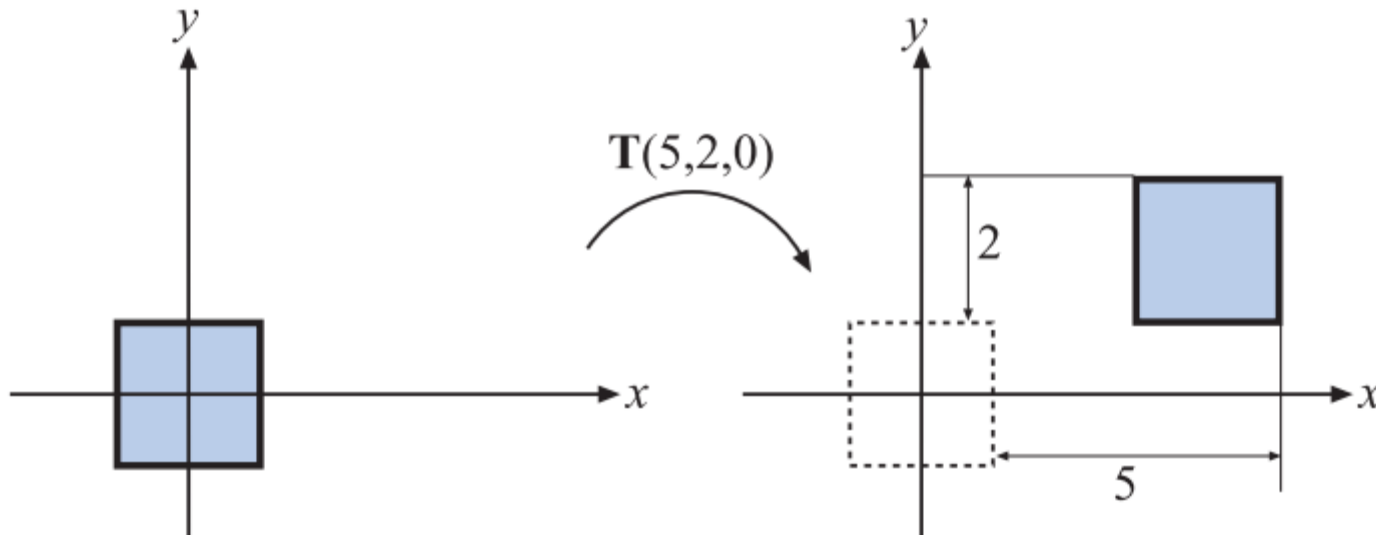
Translation

$$\begin{bmatrix} 1 & & & a \\ & 1 & & b \\ & & 1 & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \\ z + c \\ 1 \end{bmatrix}$$

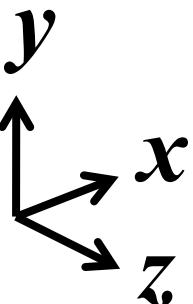
Translation

$$\mathbf{T}(\mathbf{t}) = \mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

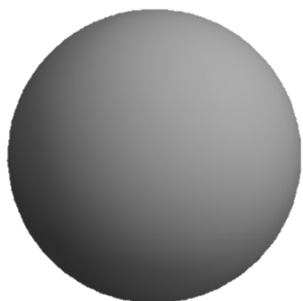
The inverse of a translation matrix is $\mathbf{T}^{-1}(\mathbf{t}) = \mathbf{T}(-\mathbf{t})$, that is, the vector \mathbf{t} is negated.



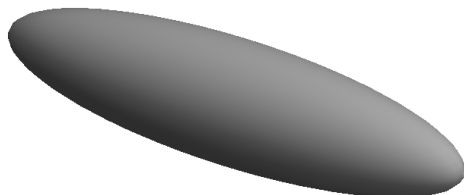
Scale



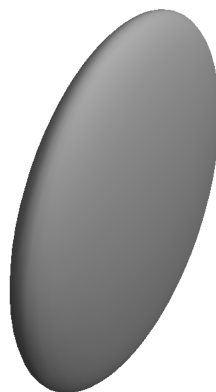
$$\begin{bmatrix} a & & \\ & b & \\ & & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ by \\ cz \\ 1 \end{bmatrix}$$



Uniform Scale
 $a = b = c = \frac{1}{4}$



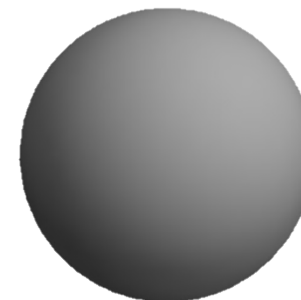
Stretch
 $a = b = 1, c = 4$



Squash
 $a = b = 1, c = \frac{1}{4}$

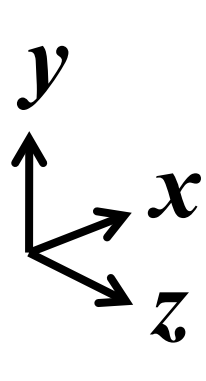


Project
 $a = b = 1, c = 0$



Invert
 $a = b = 1, c = -1$

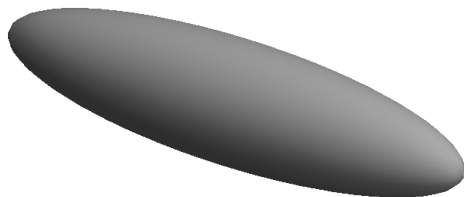
Scale


$$\begin{bmatrix} a & & \\ & b & \\ & & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ by \\ cz \\ 1 \end{bmatrix}$$

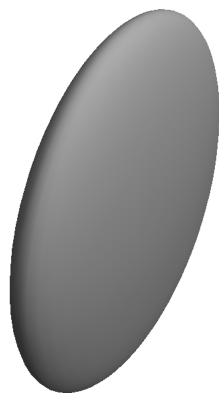


Uniform Scale
 $a = b = c = \frac{1}{4}$

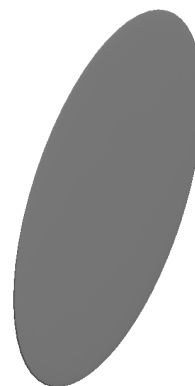
What is the inverse transformation?



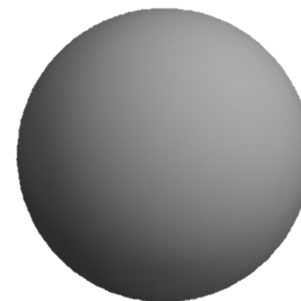
Stretch
 $a = b = 1, c = 4$



Squash
 $a = b = 1, c = \frac{1}{4}$



Project
 $a = b = 1, c = 0$



Invert
 $a = b = 1, c = -1$

2-D Rotations: It's Not Magic

In two dimensions, the rotation matrix is simple to derive. Assume that we have a vector, $\mathbf{v} = (v_x, v_y)$, which we parameterize as $\mathbf{v} = (v_x, v_y) = (r \cos \theta, r \sin \theta)$. If we were to rotate that vector by ϕ radians (counterclockwise), then we would get $\mathbf{u} = (r \cos(\theta + \phi), r \sin(\theta + \phi))$. This can be rewritten as

$$\begin{aligned}\mathbf{u} &= \begin{pmatrix} r \cos(\theta + \phi) \\ r \sin(\theta + \phi) \end{pmatrix} = \begin{pmatrix} r(\cos \theta \cos \phi - \sin \theta \sin \phi) \\ r(\sin \theta \cos \phi + \cos \theta \sin \phi) \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix}}_{\mathbf{R}(\phi)} \underbrace{\begin{pmatrix} r \cos \theta \\ r \sin \theta \end{pmatrix}}_{\mathbf{v}} = \mathbf{R}(\phi) \mathbf{v},\end{aligned}\tag{4.4}$$

3-D Rotations

- About x-axis
 - rotates $y \rightarrow z$

$$\mathbf{R}_x(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi & 0 \\ 0 & \sin \phi & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

- About y-axis
 - rotates $z \rightarrow x$

$$\mathbf{R}_y(\phi) = \begin{pmatrix} \cos \phi & 0 & \sin \phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \phi & 0 & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

- About z-axis
 - rotates $x \rightarrow y$

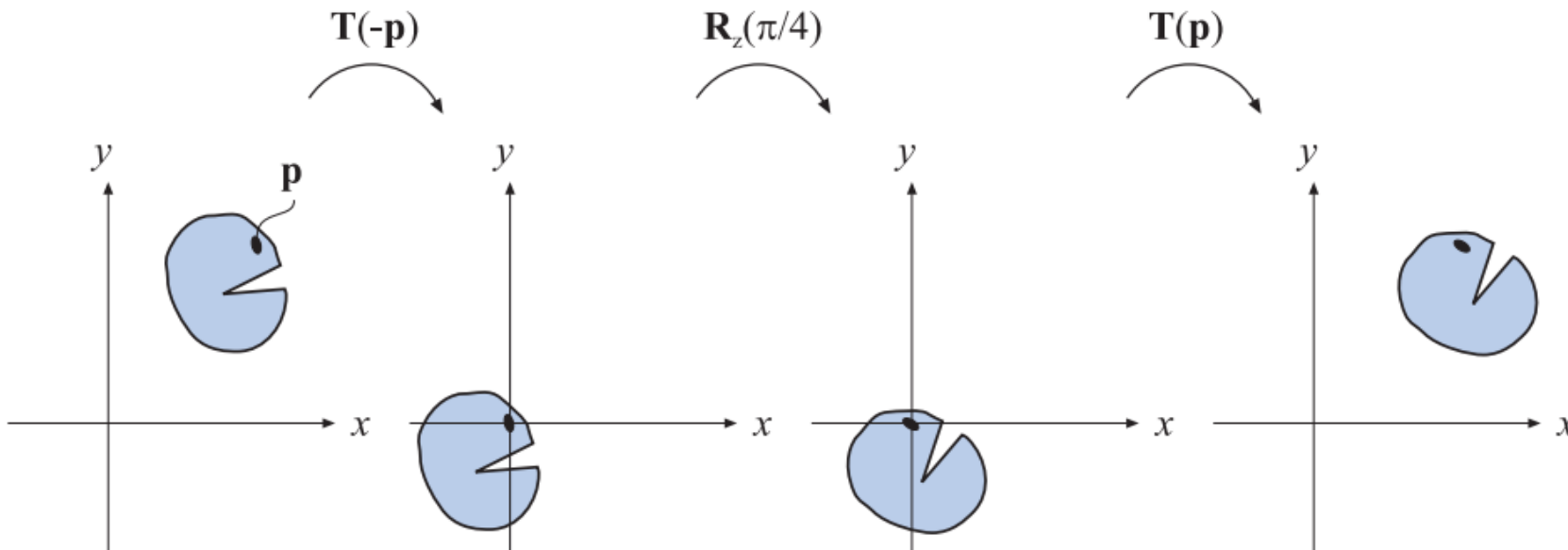
$$\mathbf{R}_z(\phi) = \begin{pmatrix} \cos \phi & -\sin \phi & 0 & 0 \\ \sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- Rotations do not commute!

What is the inverse transformation?

3-D Rotations

- Rotation is around the origin
- To rotate around a point p we have to:
translate p to origin, rotate, translate back to p
- Example: $\mathbf{X} = \mathbf{T}(\mathbf{p})\mathbf{R}_z(\phi)\mathbf{T}(-\mathbf{p})$.



Concatenating Transformations

- You can execute a sequence of transformations using a single matrix

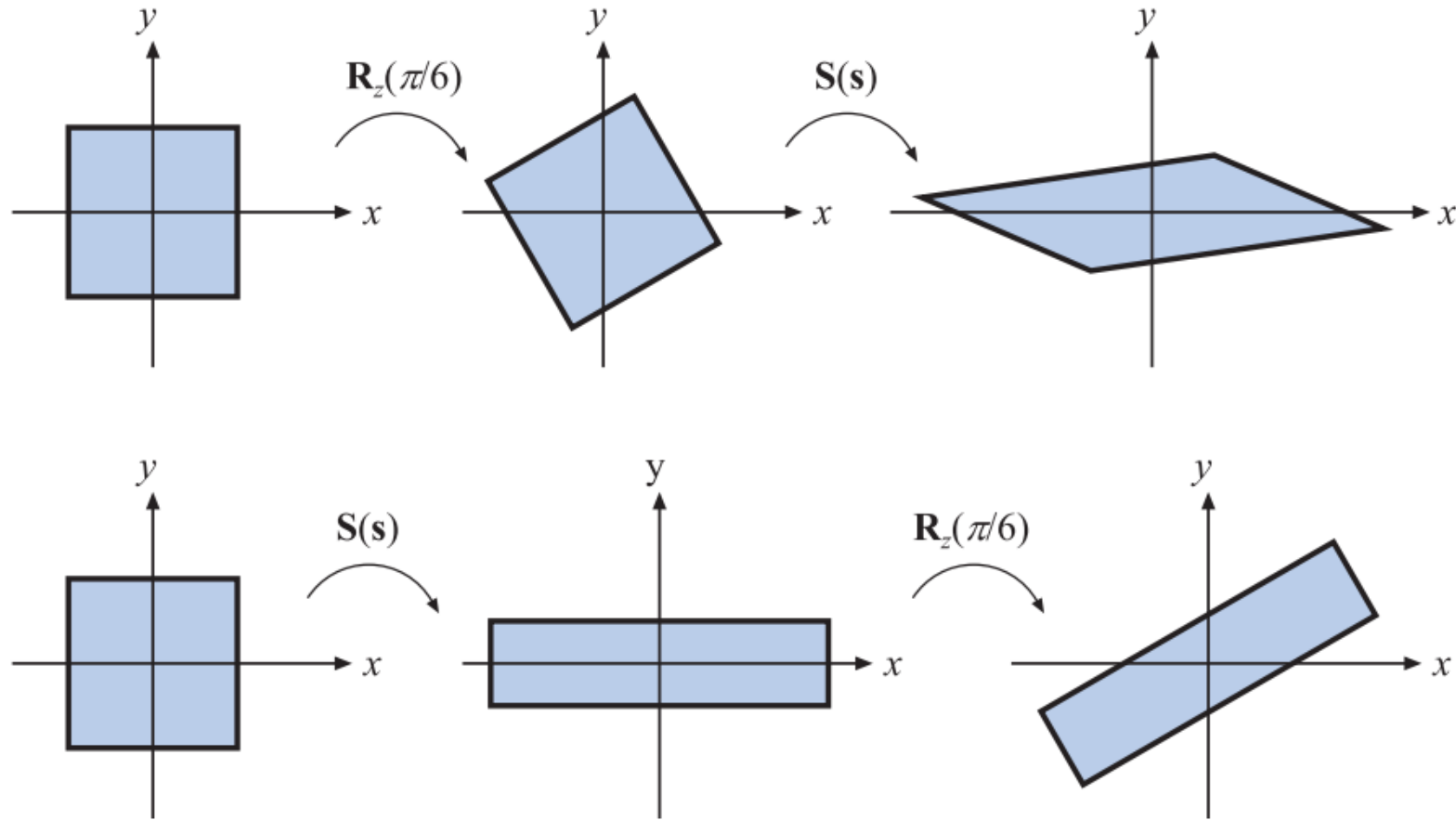
$$M_n M_{n-1} \dots M_2 M_1 p = Mp$$

- Just compute the product of the transformation matrices
- Important M_1 is applied first then M_2 and so on
- ***Modeling transformation:***
model is sized, placed and oriented in the virtual world

$$C = TRS.$$

$$TRSp = (T(R(Sp))),$$

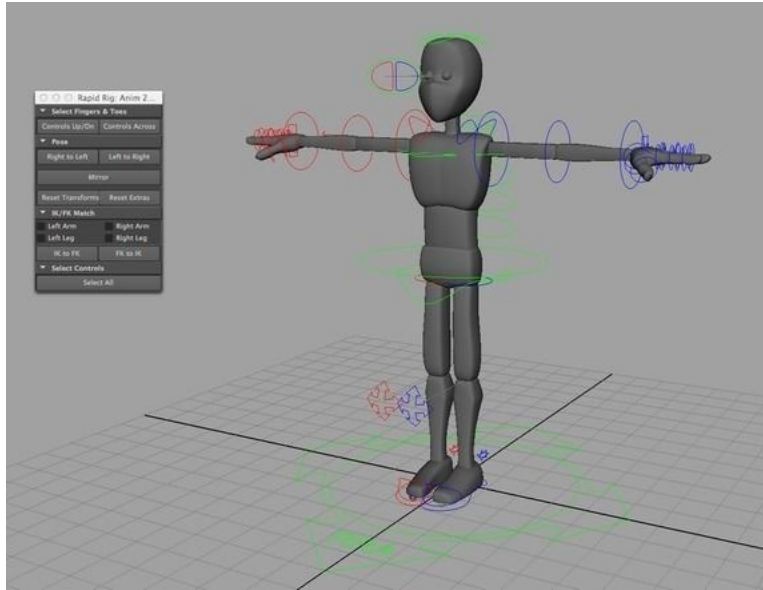
Non-Commutativity Example



It generally holds that $MN \neq NM$, for arbitrary matrices M and N .

But sometimes... $MN=NM$...examples?

Use in Rigging



"3D rigging is **the process of creating a skeleton for a 3D model so it can move**. Most commonly, characters are rigged before they are animated because if a character model doesn't have a rig, they can't be deformed and moved around."

Each joint has a transformation associated with it

The rigging skeleton generates a tree of transformations

So, fingers are transformed a concatenated transform
Product of matrices from the chest to shoulder to elbow to wrist, etc.

