Lecture 3: Thursday 16 September 1999

Administrative:

- Asst 1 due tomorrow by 5pm
  see `turnin` instructions on course page
- Asst 2 due next Friday by 5pm. For Bresenham, remember: must use integer variables only !
- Charles (TA, `chocobo@graphics.lcs.mit.edu`)
  Office Hours W 5-7pm, R 10-12**pm** in W20-575
- Damian (TA, `naimad@graphics.lcs.mit.edu`)
  Office Hours T 5-7pm in NE43-2; R 5-7pm in 4-035
- Kari Anne (TA, `karianne@graphics.lcs.mit.edu`)
  Office Hours MT 7-9pm in 4-035
- Prof. T. (`teller@lcs.mit.edu`)
  Office Hours T 4-5pm in 4-370/NE43-252
- (All start 9/20, or by appointment this week.)

Today:

- Overview of Rendering Pipeline
- Modeling Transformations

Rendering

Generation of a *synthetic image*, given

Scene description:
Geometric model of objects, surface and volumetric qualities, light sources, etc.

Lighting model:
Computational description of object and light properties, interaction (reflection, etc.)

Synthetic camera:
Eye position and view *frustum* (or *volume*)

Raster Viewport:
Pixel grid onto which image plane is mapped

Output:
Intensity values suitable for display (e.g., 24-bit RGB framebuffer)

Classical Rendering Pipeline

Model Traversal 
Graphics *primitives* issued for display

Modeling Transformations 
Object-space → World-space

Lighting (Shading) 
Vertices *lit* (*shaded*) according to local lighting model

Clipping Transformation 
World-space → Eye-space → Clip-space

Clipping 
Portions of object outside 3D *view volume* are removed

Perspective Projection 
3D scene *projected* onto 2D *image plane*
Falloff of apparent size with distance is simulated

Transformation to Screen Space
Clip-space → Screen-space
Portion of image plane mapped to raster *viewport*

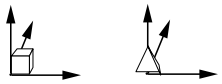
Scan Conversion: Rasterization 
Object diced into horizontal *spans*, one per raster
Depth, color, other attributes interpolated along object boundaries

Scan Conversion: Span Filling 
Depth, color, etc. interpolated across each span, yielding pixels

Visibility Resolution 
Established *per-pixel*, (typically) in hardware

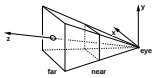
Note: many other organizations possible!

Coordinate systems

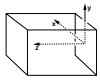
Object space

coordinate system local to each object primitive

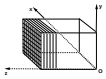
World space

common coordinate system into which all primitives resolved

Eye space (camera space)


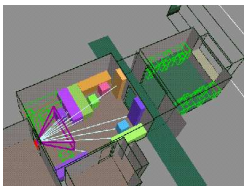
viewer-centered coordinate system derived from view frustum

Clip space (or NDC)

parallelepiped spanning $[-1,+1] \times [-1,+1] \times [-1,+1]$

Screen space (3D and 2D)

discretized parallelepiped, indexed according to hardware attributes
(floating-point, $x-y$ resolution; z resolution; etc.)

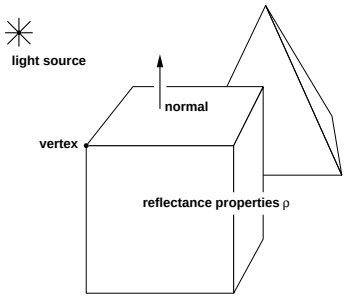
Modeling and Traversal

Host CPU processes each object in data set
One-shot or interactively



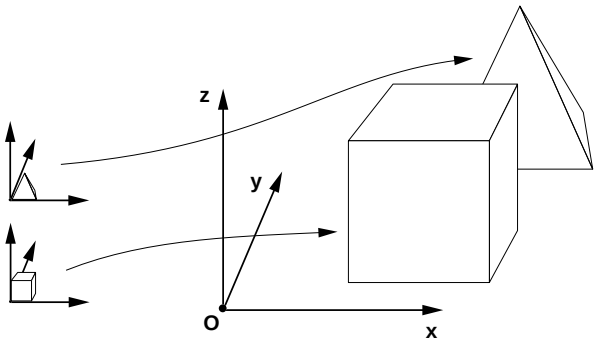
Point light sources specified
Local, Infinite sources

Scene geometry specified (today, as polygons)
Position: Vertices $\mathbf{V}_k$, with associated normals $\mathbf{N}_k$
Reflectance properties, abstractly denoted ρ
Color, shininess, transparency, texture, etc.



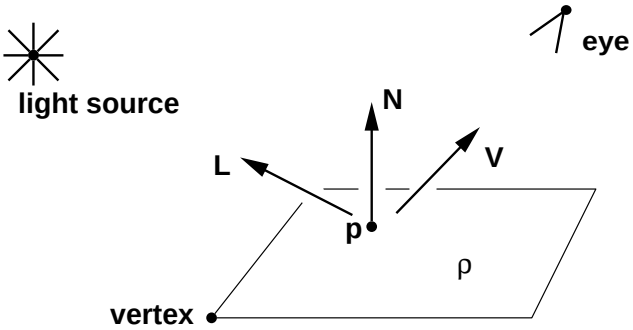
Modeling Xform (object → world space)

Hierarchical objects instantiated, mapped into single "world-space" coordinate system



Lighting (Shading)

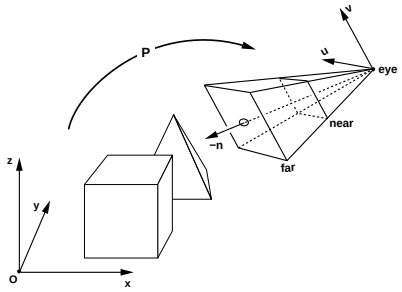
Color computed at each polygon vertex



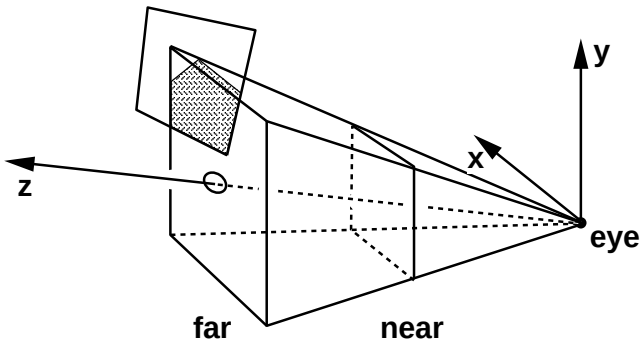
color = f (p, N, light L, eye V, ρ)

Clipping

Geometry re-expressed in eye-centered coordinates:

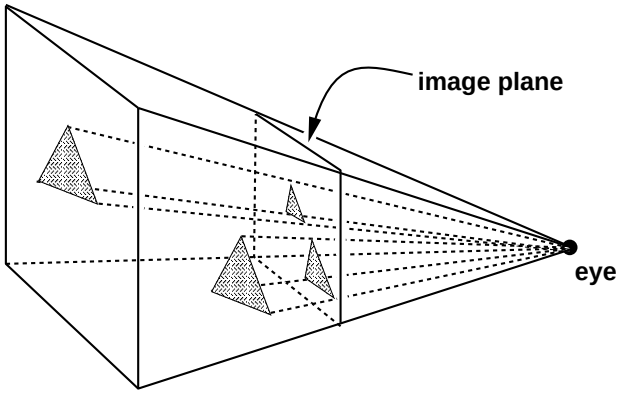


Portions of polygon outside view frustum are removed



Perspective Projection

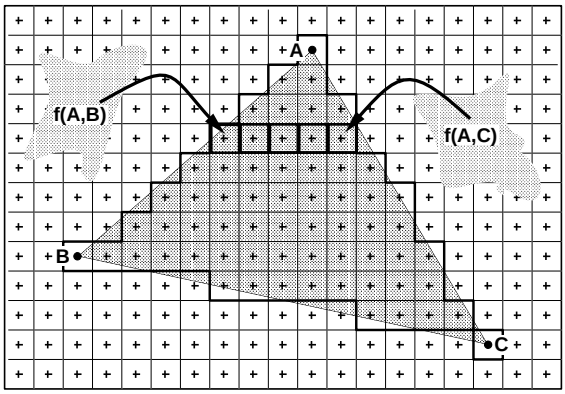
3D scene *projected* onto 2D *image plane*



Scan-conversion: Rasterization

Polygons *rasterized* into horizontal *spans*

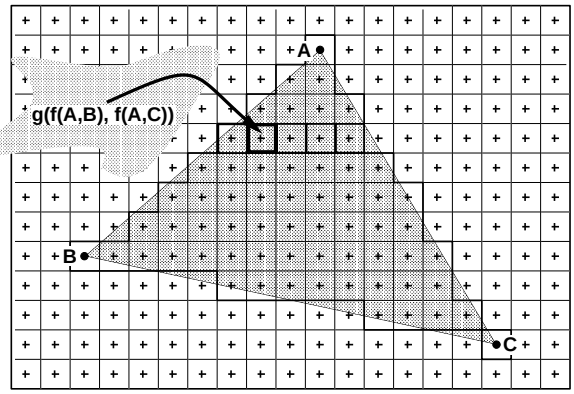
- Some *sampling rule* used for *discretization*
- Each span has associated color, depth, other information



Scan-conversion: Span Filling

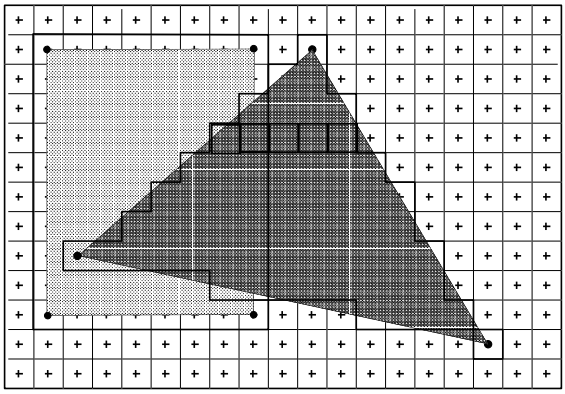
Each span is *filled* with a sequence of pixel values

- Associated attributes *interpolated* across spanline



Visibility Resolution

There exist many methods; we'll start with depth-bufferin
Every pixel has an integer quantity, its "depth"
At start of frame, all depths initialized to "far away"



Color value written *conditionally* at each pixel

If $z < z_{\text{stored}}$ then
update stored value
$z_{\text{stored}} = z$

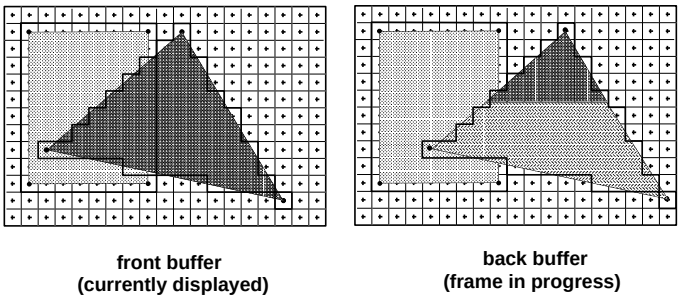
Output

Static rendering

2d array of color values *(image, frame, field)*

Dynamic rendering

Frame "swapped" forward by switching video memory bank pointers



Double-buffering eliminates ghosting, some temporal artifacts

Abstractions Examined

Modeling & Traversal
- Hierarchical shape representation methods
- Representation of surface reflectance/geometry
- Object-space *culling* methods for visibility

Coordinate Transformations
- Modeling transformations
- Viewing transformations
- Screen/Texture-space transformations

Lighting (also called Shading)
- Lighting Models (local, semi-local, global)
- Energy conservation, reciprocity, etc.

Clipping
- Transformation to Clip coordinates
- Segment, polygon clipping
- Dependence on input distribution

Abstractions Examined (cont.)

Perspective Projection
- Homogeneous coordinates
- Projection as a matrix operation

Scan Conversion
- Scan-line algorithms
- Flood-fill algorithms
- Convex-fill algorithms, parallelism
- Aliasing and Anti-Aliasing
- Screen- vs Object-space interpolation
- 2D and 3D Texture Mapping

Visibility determination
- Hidden surface elimination, history
- Depth buffering (z-buffering)
- Hierarchical depth-buffering algorithms
- Modern visible-surface algorithms

3D Coordinate Transformations

(H&B 12.1-12.5; OpenGL PG Appendix G/F)

CAUTION: conventions differ among texts!

I'll almost always use right-multiplication ($\mathbf{p'} = \mathbf{Mp}$)

Transformation: operator defined on geometric object

Abstractly:

$\mathbf{M}$ maps points $\mathbf{p}$ to points $\mathbf{M}(\mathbf{p}) = \mathbf{p'} = (x', y', z')$

Types of transformations

Linear: rotation, scaling, reflection, shear, or any combination

$\mathbf{M}(\mathbf{p} + \mathbf{q}) = \mathbf{M}(\mathbf{p}) + \mathbf{M}(\mathbf{q}); \mathbf{M}(\alpha\mathbf{p}) = \alpha\mathbf{M}(\mathbf{p})$

Rigid-Body: translation, rotation (or combination)

Preserves [ ]

Affine: translation, rotation, scaling, reflection, shear (or any combination)

(linear mappings plus a translation)

Preserves [ ]

Projective: perspective projection

Aka *homogeneous* or *rational linear* mapping

Preserves lines

(most general line-preserving transformation)

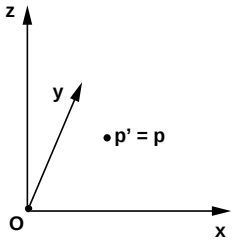
Homogeneous Coordinates

Convenient to describe points as 4-vectors (x, y, z, w)

w is "homogeneous coordinate," to be explained later

For now, we will use $w = 1$ everywhere

Trivial example: identity transformation



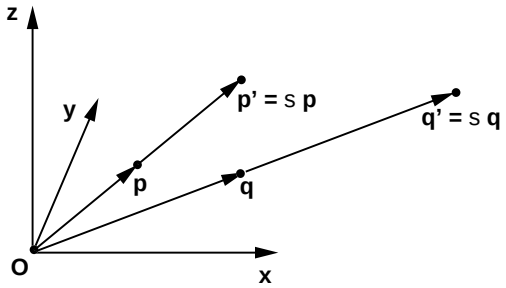
$$\begin{aligned} x' &= x \\ y' &= y \\ z' &= z \end{aligned}$$

In matrix form

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

Isotropic Scaling

Scale all points about origin by factor s:



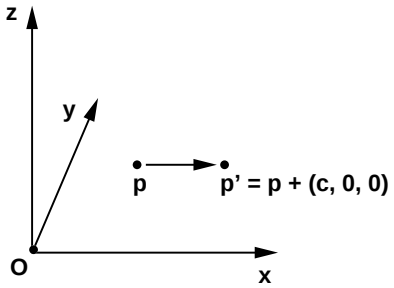
$$\begin{aligned} x' &= sx \\ y' &= sy \\ z' &= sz \end{aligned}$$

In matrix form

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \square & 0 & 0 & 0 \\ 0 & \square & 0 & 0 \\ 0 & 0 & \square & 0 \\ 0 & 0 & 0 & \square \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Translation

Translation along $+x$ axis by a constant c:



$$\begin{aligned} x' &= x + c \\ y' &= y \\ z' &= z \end{aligned}$$

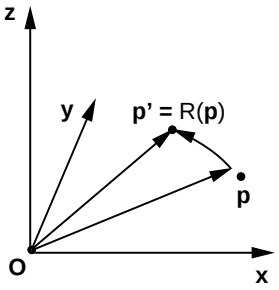
In matrix form,

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & c \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Note: [ ] ! Why?

Rotation

Rotation about $+z$ axis by angle θ (note convention)



$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \\ z' &= z \\ 1 &= 1 \end{aligned}$$

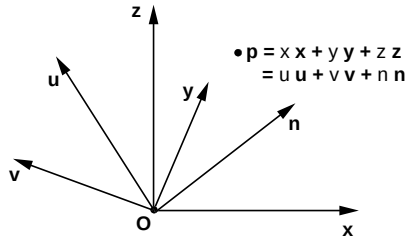
In matrix form

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

Rotation about x, y axes analogous

Change of orthobasis (xyz to uvn)

"Express **xyz** (i.e., world-space) points in **uvn** basis."



We know $\mathbf{p} = x\hat{\mathbf{x}} + y\hat{\mathbf{y}} + z\hat{\mathbf{z}} = u\hat{\mathbf{u}} + v\hat{\mathbf{v}} + n\hat{\mathbf{n}}$
Find transformation $\mathbf{M}$ that, given $x, y, z, 1$, produces $u, v, n, 1$

$$\begin{aligned} \hat{\mathbf{x}} &= (\hat{\mathbf{x}}\cdot\hat{\mathbf{u}})\hat{\mathbf{u}} + (\hat{\mathbf{x}}\cdot\hat{\mathbf{v}})\hat{\mathbf{v}} + (\hat{\mathbf{x}}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}} \\ \hat{\mathbf{y}} &= (\hat{\mathbf{y}}\cdot\hat{\mathbf{u}})\hat{\mathbf{u}} + (\hat{\mathbf{y}}\cdot\hat{\mathbf{v}})\hat{\mathbf{v}} + (\hat{\mathbf{y}}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}} \\ \hat{\mathbf{z}} &= (\hat{\mathbf{z}}\cdot\hat{\mathbf{u}})\hat{\mathbf{u}} + (\hat{\mathbf{z}}\cdot\hat{\mathbf{v}})\hat{\mathbf{v}} + (\hat{\mathbf{z}}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}} \end{aligned}$$

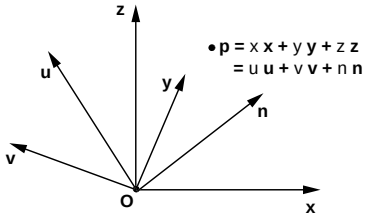
Rewrite $\mathbf{p}$, then collect terms in u, v, n:

$$\begin{aligned} u &= x(\hat{\mathbf{x}}\cdot\hat{\mathbf{u}}) + y(\hat{\mathbf{y}}\cdot\hat{\mathbf{u}}) + z(\hat{\mathbf{z}}\cdot\hat{\mathbf{u}}) \\ v &= x(\hat{\mathbf{x}}\cdot\hat{\mathbf{v}}) + y(\hat{\mathbf{y}}\cdot\hat{\mathbf{v}}) + z(\hat{\mathbf{z}}\cdot\hat{\mathbf{v}}) \\ n &= x(\hat{\mathbf{x}}\cdot\hat{\mathbf{n}}) + y(\hat{\mathbf{y}}\cdot\hat{\mathbf{n}}) + z(\hat{\mathbf{z}}\cdot\hat{\mathbf{n}}) \end{aligned}$$

In matrix form

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Two interpretations of orthobasis change



$$\mathbf{M} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \qquad \text{Check:} \qquad \mathbf{M}\begin{pmatrix} u_x \\ u_y \\ u_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

What is $\mathbf{M}^{-1}$? (What takes **uvn** to **xyz** coords?)

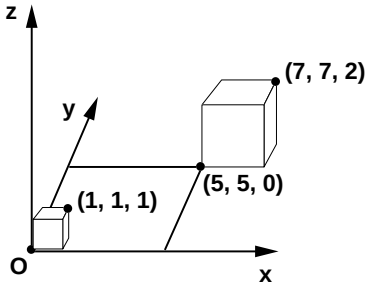
Simply $\mathbf{M}^T$, since $\mathbf{M}^T \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} u_x \\ u_y \\ u_z \\ 1 \end{pmatrix}$, etc.

Interpretations:

1: $\mathbf{M}$ "takes" **uvn** basis into **xyz** basis
2: $\mathbf{M}$ "reexpresses" **xyz** points in **uvn** coords!

Composition, Non-Commutativity

Given: a unit cube at the origin
Transform to: a side-2 cube at (5, 5, 0) from origin

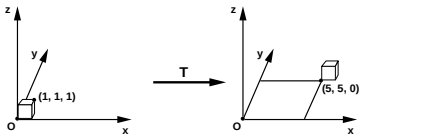


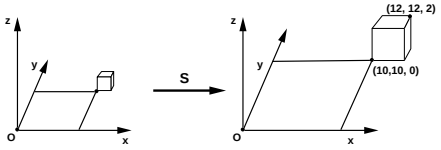
We know that
T(0,0,0) = (5,5,0) and
T(1,1,1) = (7,7,2)
What do we do?
Translate, then scale? Or...
Scale, then translate?

Translate, then Scale:

Compute individual matrices, then compose

$$\mathbf{T}_{(5,5,0)} = \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \qquad \mathbf{S}_2 = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



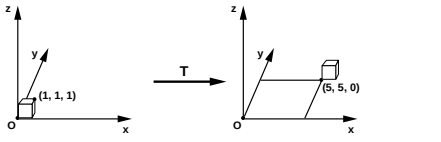


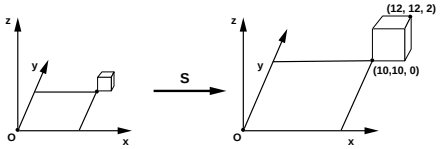
$$\mathbf{ST} = \begin{pmatrix} 2 & 0 & 0 & 10 \\ 0 & 2 & 0 & 10 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

What happened?

$$\mathbf{STp} = \mathbf{ST}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ 10 \\ 0 \\ 1 \end{pmatrix}$$

$$\mathbf{STp} = \mathbf{ST}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 12 \\ 12 \\ 2 \\ 1 \end{pmatrix}$$



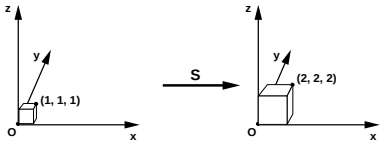


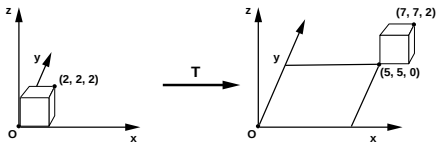
$\mathbf{S}\ \mathbf{T}\ \mathbf{p} = \mathbf{S}\ (\mathbf{T}\ \mathbf{p})$; $\mathbf{S}$ acting on translated $\mathbf{p}$

Scale, then Translate

Apply $\mathbf{S}$ first:

$$\mathbf{TS} = \begin{pmatrix} 2 & 0 & 0 & 5 \\ 0 & 2 & 0 & 5 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

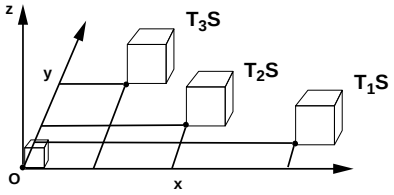




$$\mathbf{TSp} = \mathbf{TS}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 7 \\ 7 \\ 2 \\ 1 \end{pmatrix}$$

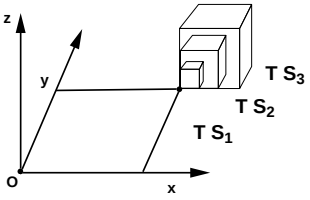
World Space Transformation

$\mathbf{T}\ \mathbf{S}\ \mathbf{p} = \mathbf{T}\ (\mathbf{S}\ \mathbf{p})$; $\mathbf{T}$ acting on scaled $\mathbf{p}$
Think of $\mathbf{M}$ as a *string* of transformations, $\mathbf{p}' = \mathbf{Mp}$
Now apply $\mathbf{N}$ to the *left* of $\mathbf{M}$: $\mathbf{p}'' = \mathbf{NMp} = \mathbf{N}(\mathbf{Mp})$
We say that $\mathbf{N}$ is applied in *world space*.



Object-Space transformation

Apply $\mathbf{N}$ to the *right* of $\mathbf{M}$: $\mathbf{p}'' = \mathbf{MNp} = \mathbf{M}(\mathbf{Np})$
We say that $\mathbf{N}$ is applied in *object space*.
This makes sense – it's the first thing to "act upon" $\mathbf{p}$

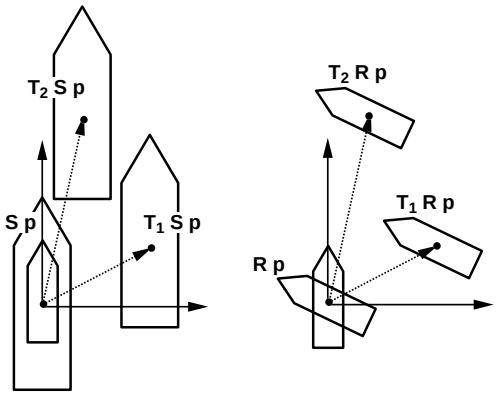


World-Space Transformation

$\mathbf{T}\ \mathbf{R}\ \mathbf{p} = \mathbf{T}\ (\mathbf{R}\ \mathbf{p})$; $\mathbf{T}$ acting on rotated $\mathbf{p}$

Again, apply $\mathbf{N}$ to the *left* of $\mathbf{M}$: $\mathbf{p}'' = \mathbf{NMp} = \mathbf{N}(\mathbf{Mp})$

Again, say that $\mathbf{N}$ is applied in "world space."



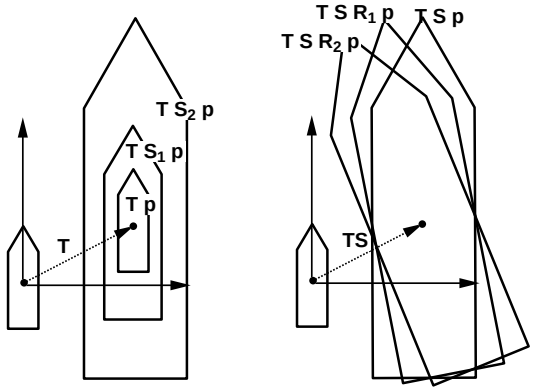
Object Space Transformation

Apply $\mathbf{N}$ to the *right* of $\mathbf{M}$: $\mathbf{p}'' = \mathbf{MNp} = M(\mathbf{Np})$

Aagin, $\mathbf{N}$ is applied in "object space;"

It's the first thing to "act upon" $\mathbf{p}$

So it must happen in $\mathbf{p}$'s local coordinate system



Composite Transformations

Scale about fixed point F:

Translate F to origin

Apply scaling

Translate origin back to F

M = Trans · Scale · Trans

Reflect about line $y = x$:

Rotate counter-clockwise 45°

Reflect about x axis

Rotate clockwise 45°

M = Rot · Refl · Rot

Reflect about line $Ax + By + C = 0$:

Translate line to origin (how?)

Rotate down to x axis

Reflect about x axis

Rotate x axis back to line

Translate line back (how?)

M = Trans · Rot · Refl · Rot · Trans