

Transformations & Coordinate Systems

EDAF80

Michael Doggett



Slides by Jacob Munkberg 2012-13

Labs

- Extra Lab Time : Friday 13-15
 - If you are in an earlier lab and could move to the Friday lab, you'll get more access to the TAs
- Start work on the lab before coming to the lab

Today

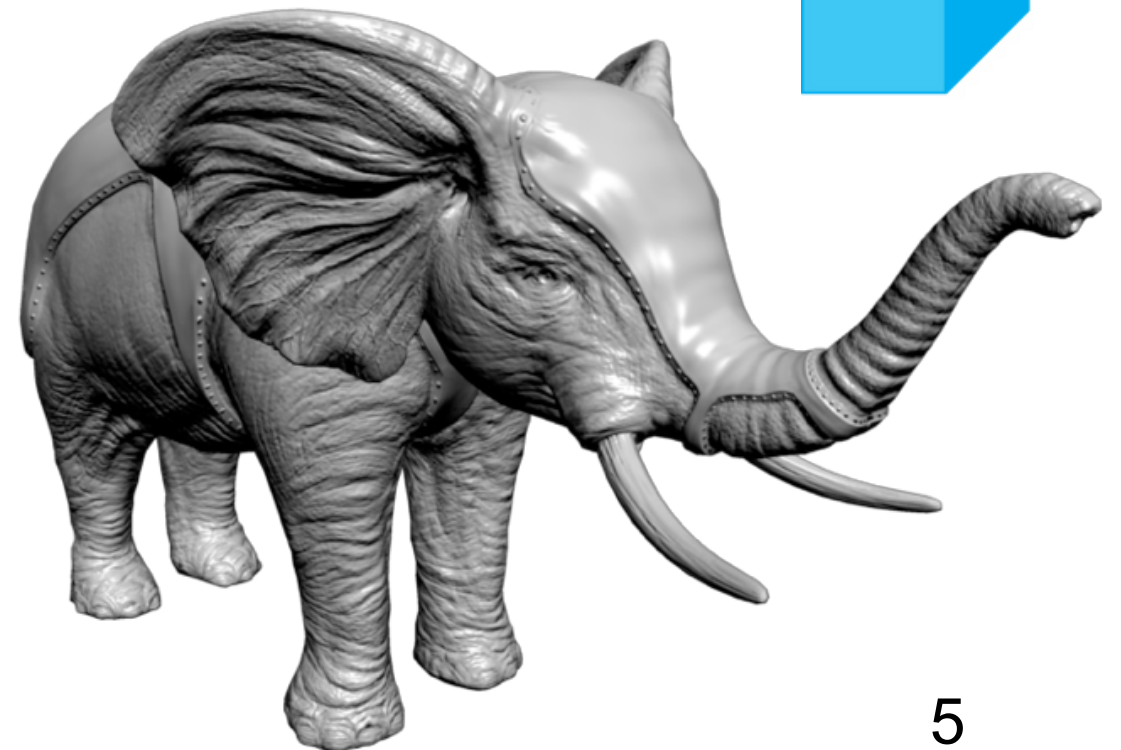
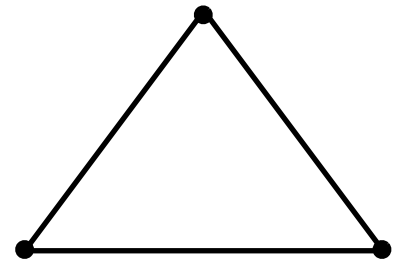
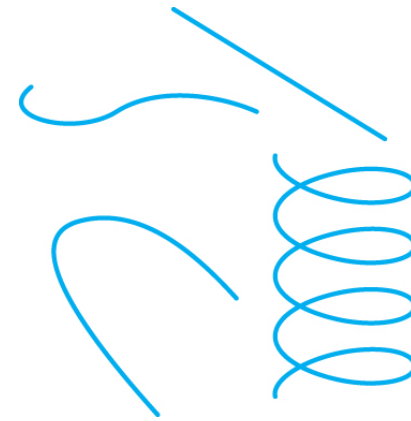
- Position objects in 3D
 - Transformations
 - Coordinate systems
- Reading
 - PBRT online : [Section 2.7 Transformations](#)

Last time

- Introduction to computer graphics
- Color, materials, real-time vs offline
- Linear algebra refresher

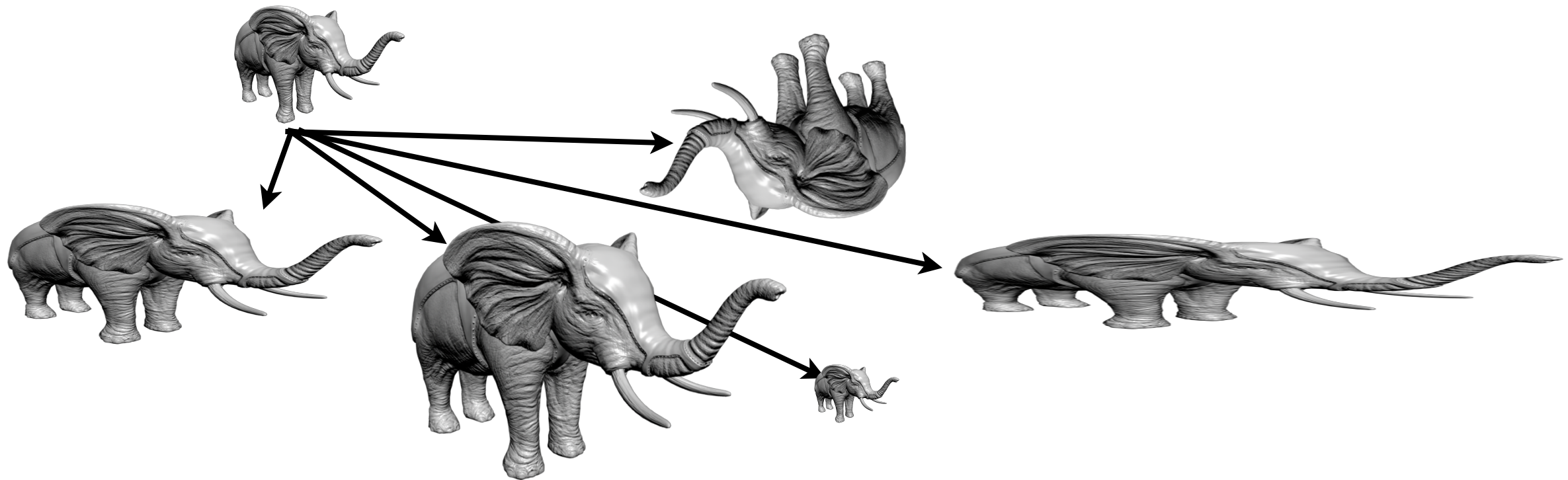
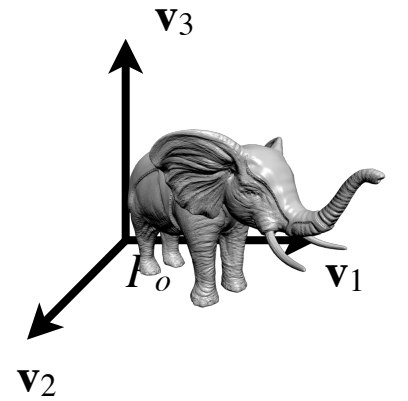
3D Primitives

- Curves - paths in space
- Triangle
 - three connected points in 3D
- Cube
 - 8 connected points in 3D
(or 12 triangles)
- Elephant
 - 22840 triangles



Transformations

- Define object **once** in convenient local coordinate frame
- Apply transformation **M** to position each instance of the object

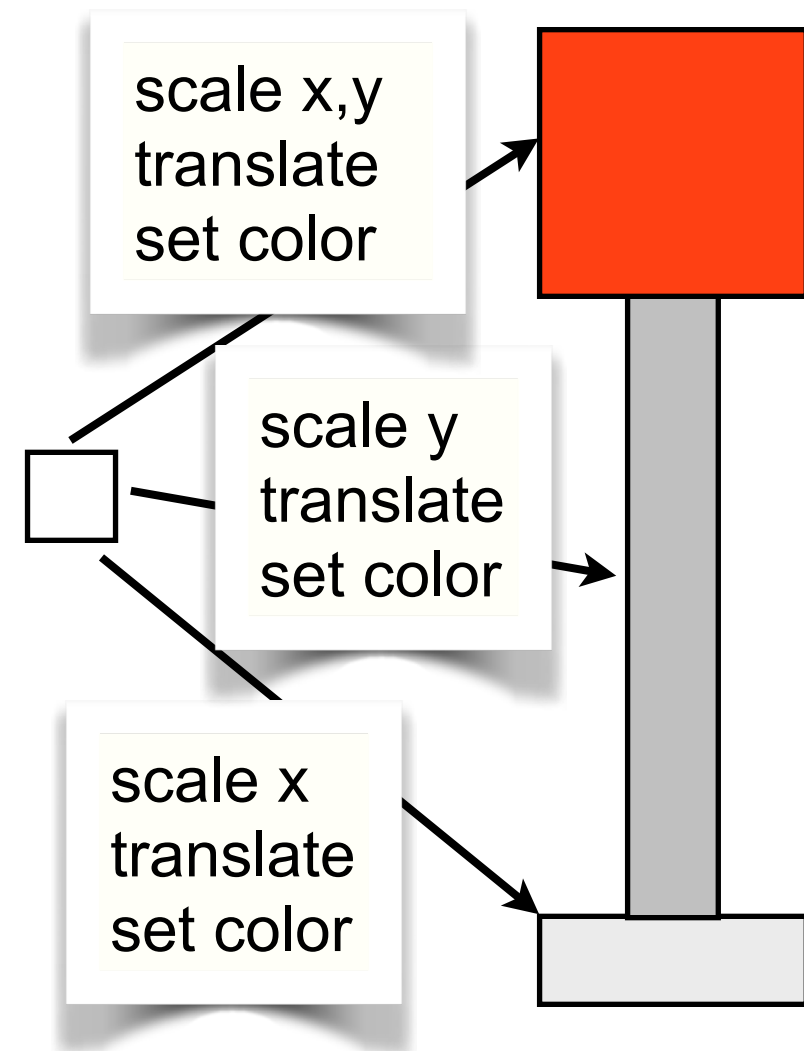


Transformations

- Maps points in one coordinate system to points in another coordinate system
- Useful for:
 - Position objects, lights and camera in a scene
 - Modeling
 - Change the shape of objects
 - Animations

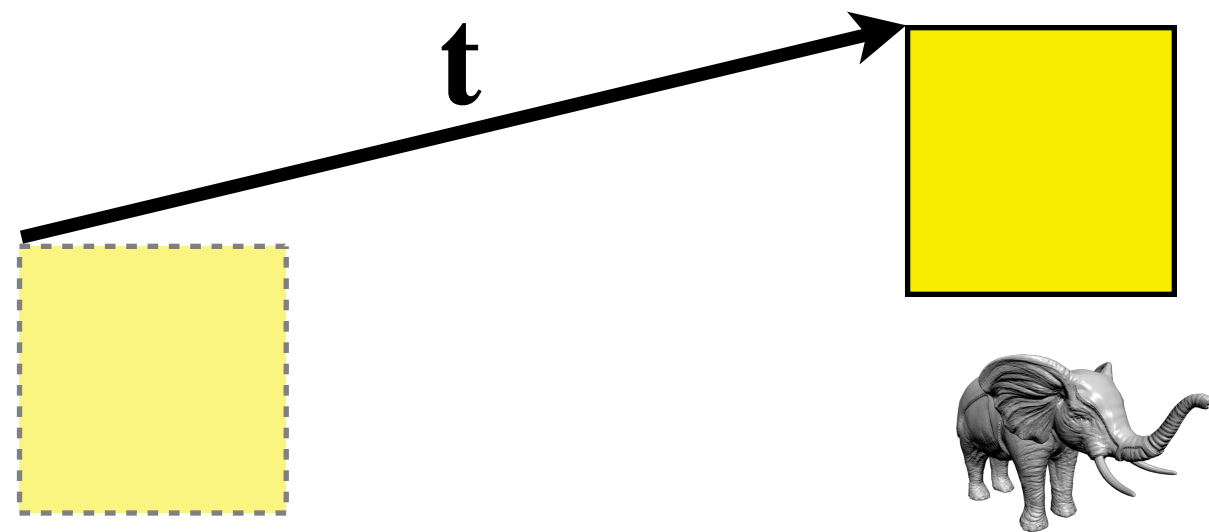
Example: Modeling

- Simple Modeling
 - Use basic primitives
 - Apply transforms to create more complex objects

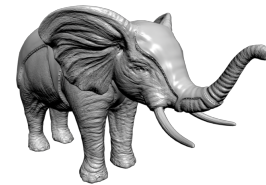


Translation

- Move along a vector
- Preserve distances
- Preserve angles



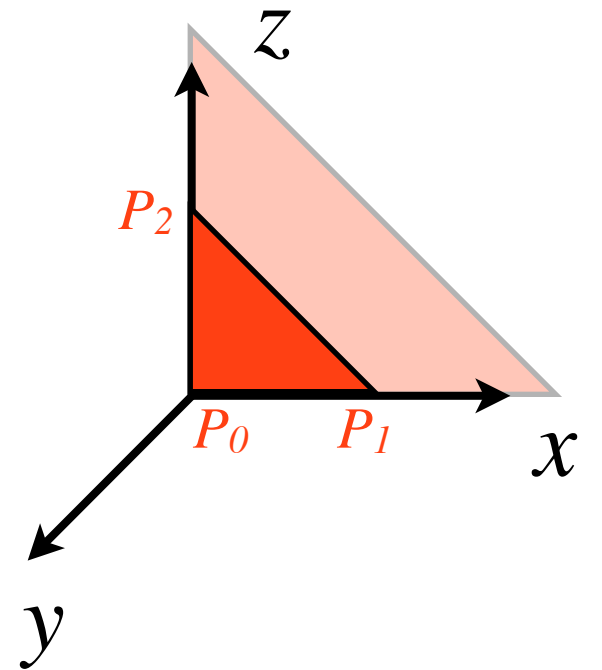
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \end{bmatrix}$$



$$P' = P + \mathbf{t}$$

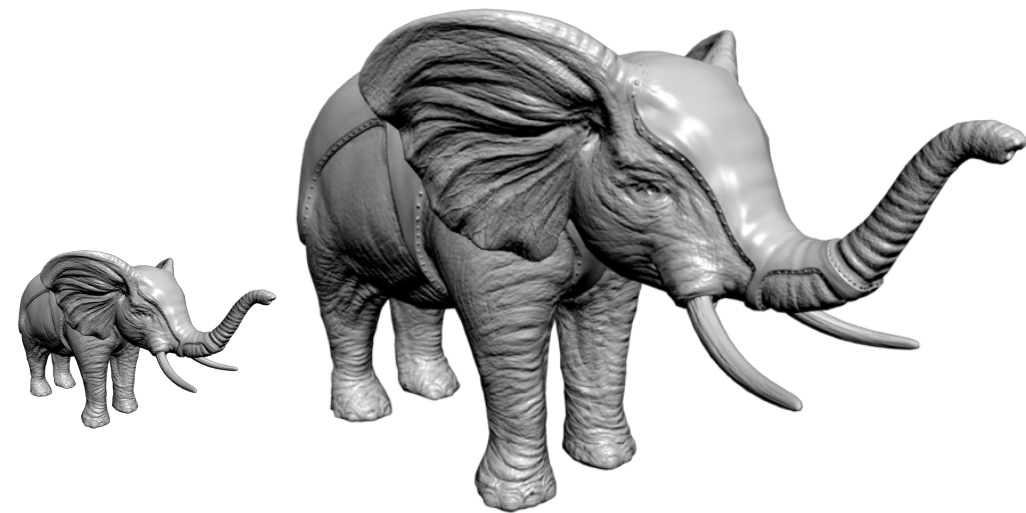
Uniform Scaling

- Scale each component with a scalar s
- Preserve angles
- **Don't** preserve distances



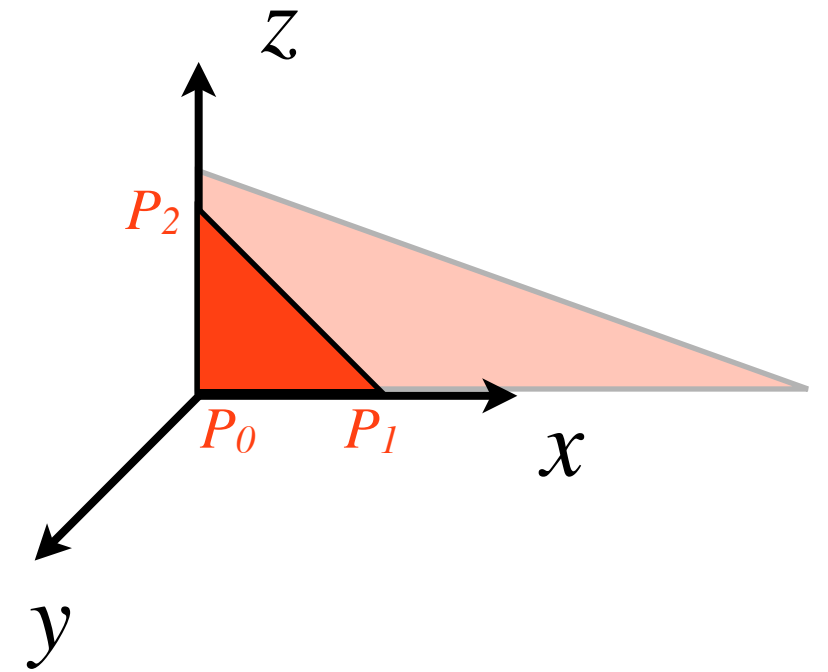
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} sx \\ sy \\ sz \end{bmatrix}$$

$$P_{\text{scaled}} = \mathbf{S}P$$



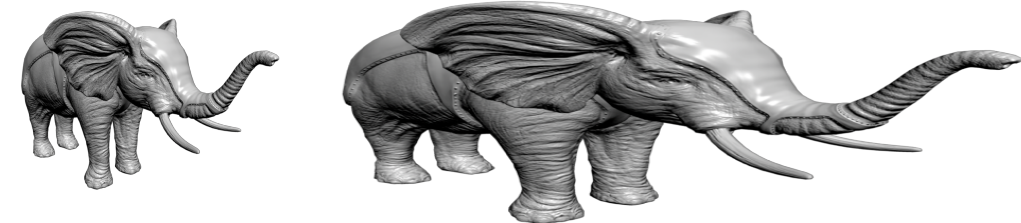
Non-uniform Scaling

- Scale each component with a different scalar s_i
- **Don't** preserve angles
- **Don't** preserve distances



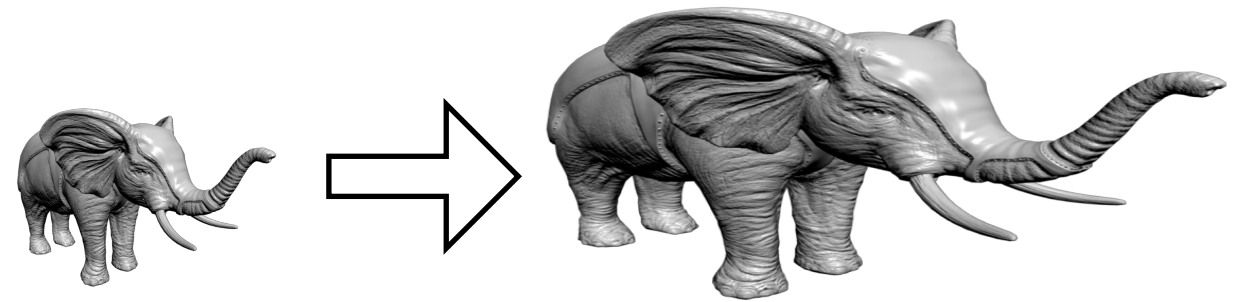
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix}$$

$$P_{\text{scaled}} = \mathbf{S}P$$

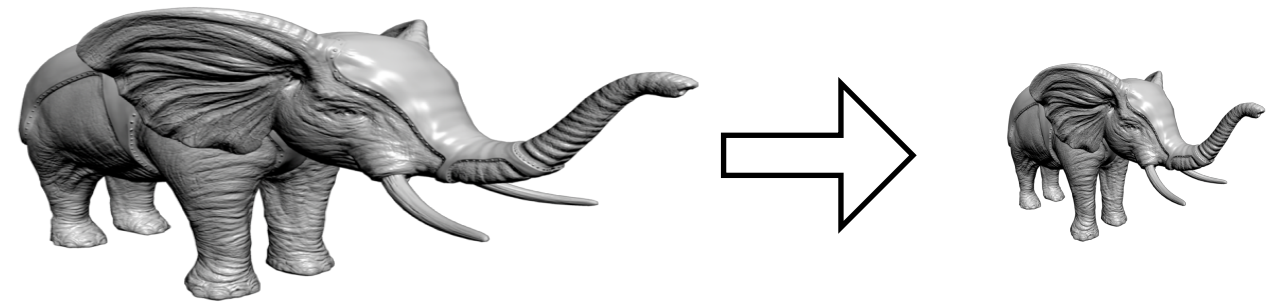


Inverse scaling transform

$$P_{\text{scaled}} = \mathbf{S}P$$



$$P = \mathbf{S}^{-1}P_{\text{scaled}}$$



$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

$$\mathbf{S}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & \frac{1}{s_z} \end{bmatrix}$$

Rotation in the plane

- Rotate vector \mathbf{u} to vector \mathbf{v}
- Express vectors in polar coordinates

$$\mathbf{u} = |\mathbf{u}|(\cos \alpha, \sin \alpha)$$

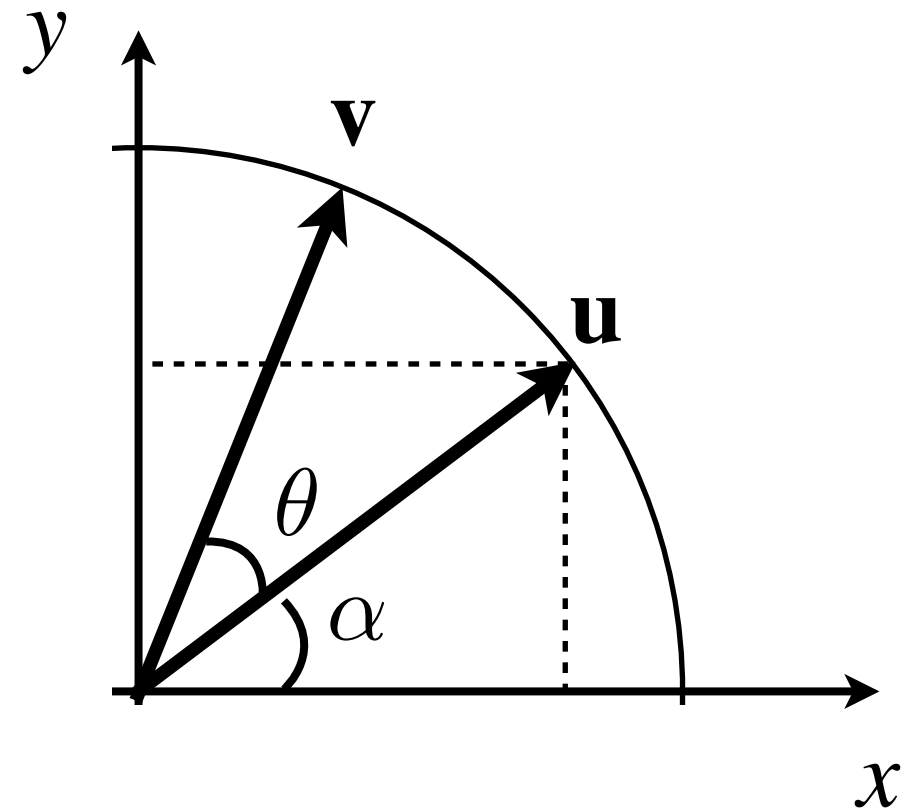
$$\mathbf{v} = |\mathbf{v}|(\cos(\alpha + \theta), \sin(\alpha + \theta))$$

- Use trigonometry

$$|\mathbf{u}| = |\mathbf{v}|$$

$$\cos(\alpha + \theta) = \cos \alpha \cos \theta - \sin \alpha \sin \theta$$

$$\sin(\alpha + \theta) = \cos \alpha \sin \theta + \sin \alpha \cos \theta$$



$$\mathbf{v} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \mathbf{u}$$

$$\mathbf{v} = \mathbf{R}(\theta) \mathbf{u}$$

Rotation in the plane

$$\mathbf{v} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \mathbf{u} \qquad \mathbf{v} = \mathbf{R}(\theta) \mathbf{u}$$

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

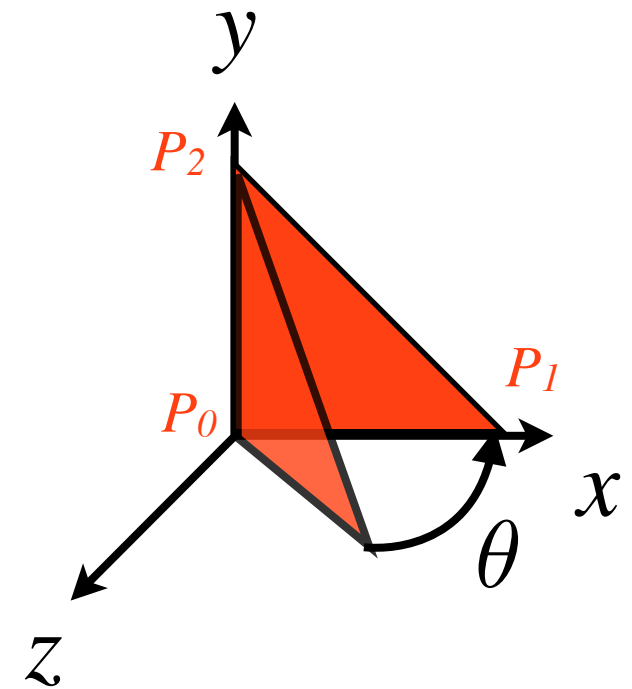
Note that: $\mathbf{R}^{-1}(\theta) = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} = \mathbf{R}^T(\theta) = \mathbf{R}(-\theta)$

Rotation matrices are **orthogonal matrices**

Rotation

- 3D Example: Rotation around y -axis

$$P_{r_y} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} P$$
$$\mathbf{R}_y(\theta)$$



- Rotations preserve angles and distances

Rotation Matrices

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

Rotation matrices
are orthonormal

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}^\top(\theta) = \mathbf{R}(-\theta)$$

$$\mathbf{R}\mathbf{R}^\top = \mathbf{1}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Classification of Transforms

Translation

Rotation

Rigid Body
preserves angles
and distances

Uniform Scaling

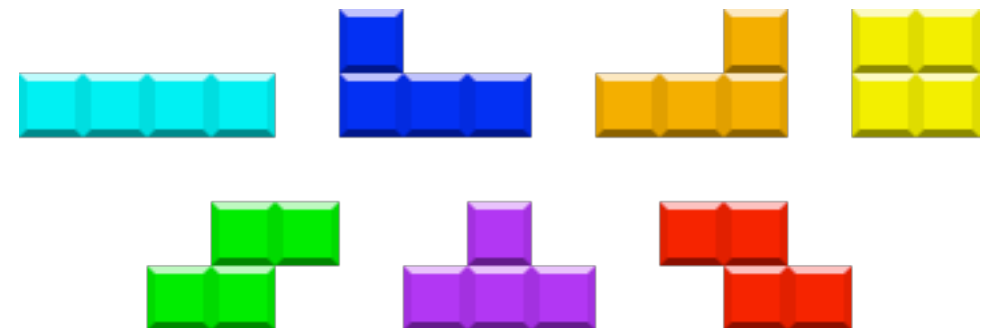
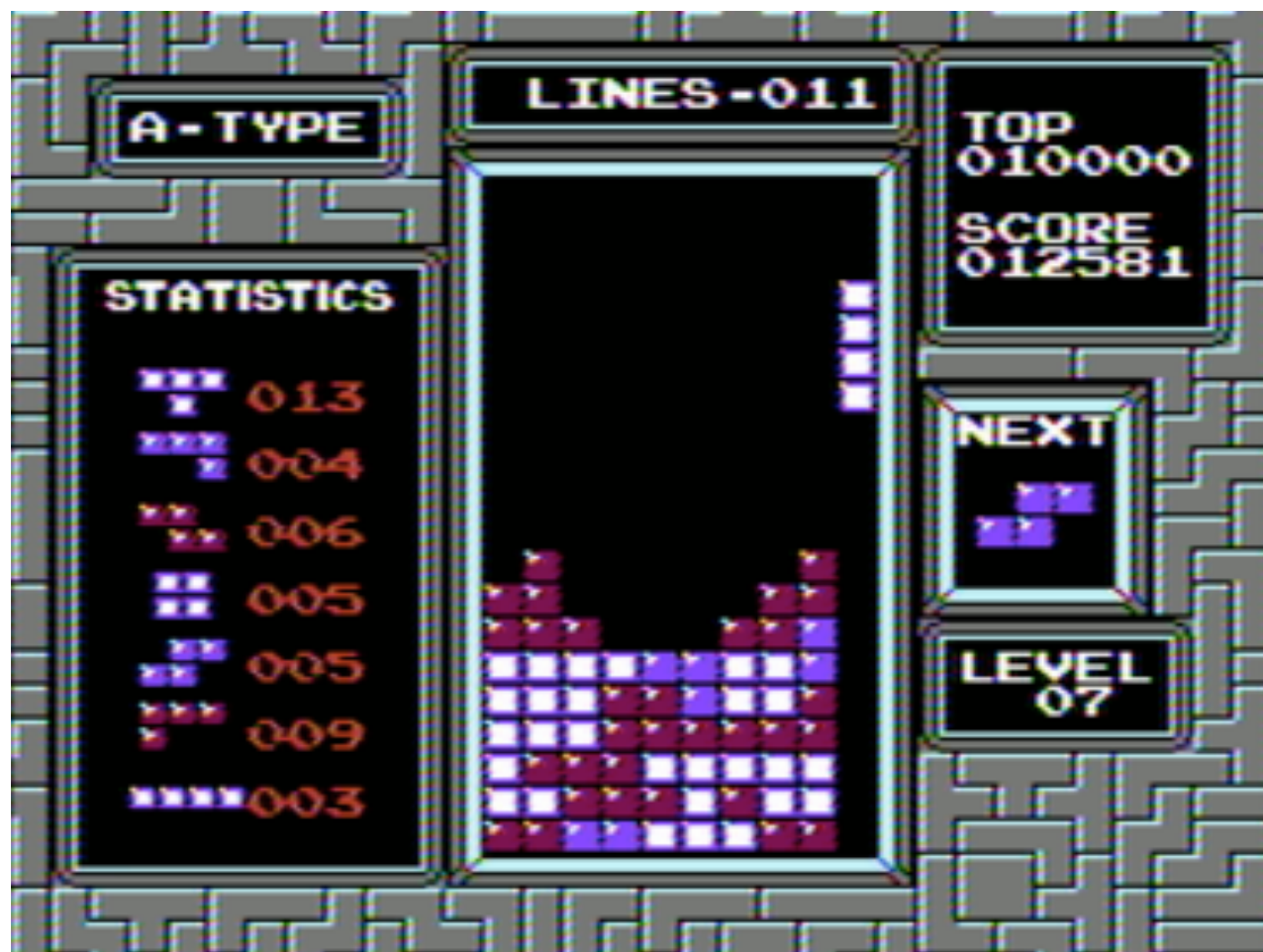
Similarity
preserves angles

Non-Uniform Scaling

Affine
preserves parallel
lines

Tetris

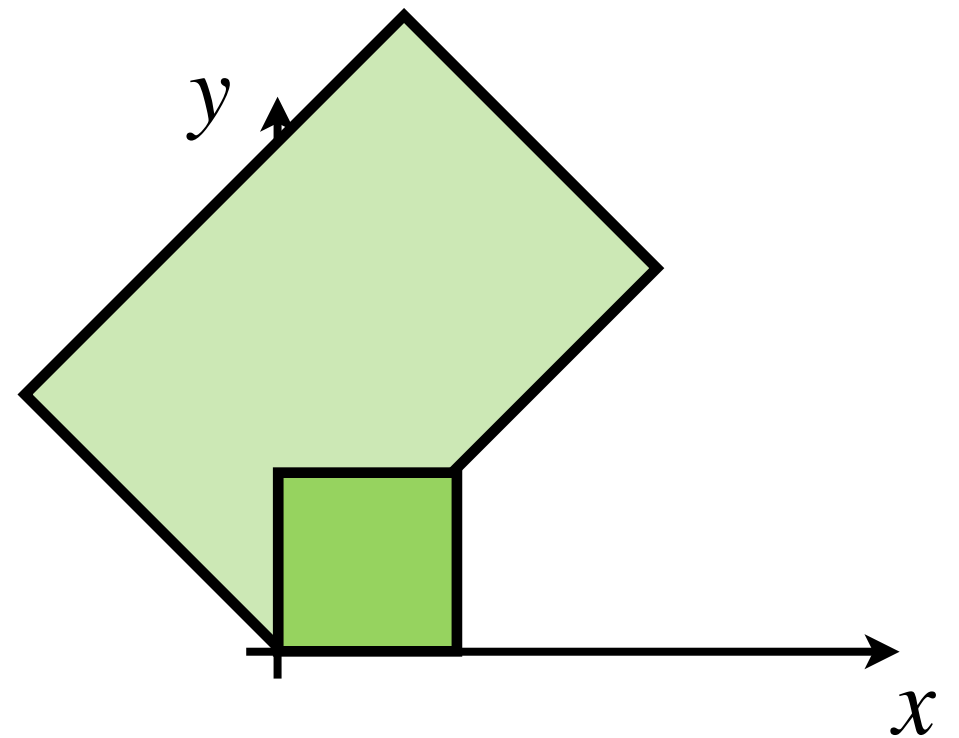
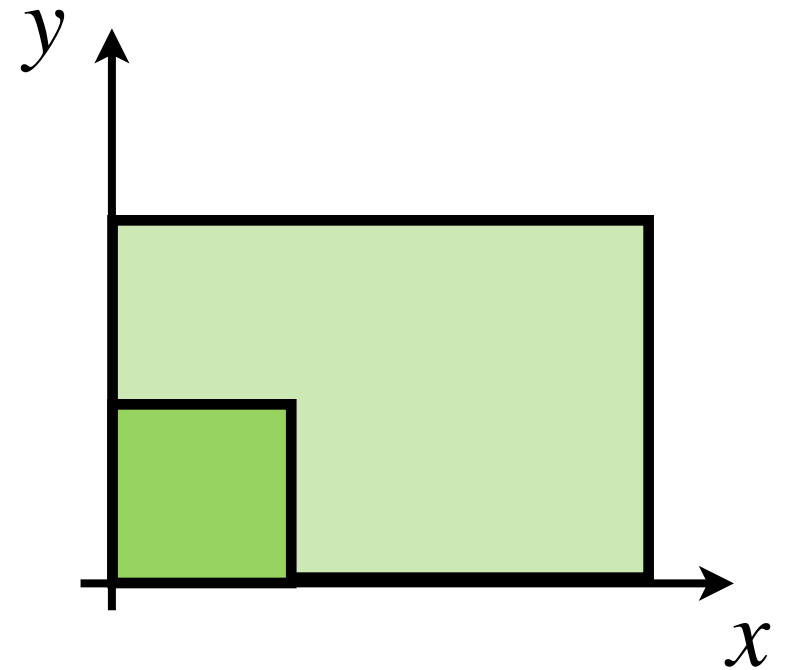
- Translation
- Rotation (increments of 90 degrees)



Composite Transform

- First: scale x by 3 and y by 2

... then rotate $\pi/4$ degrees around z -axis



Composite Transform

- First scale x by 3 and y by 2

$$P_s = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} P$$

... then rotate $\pi/4$ degrees around z -axis

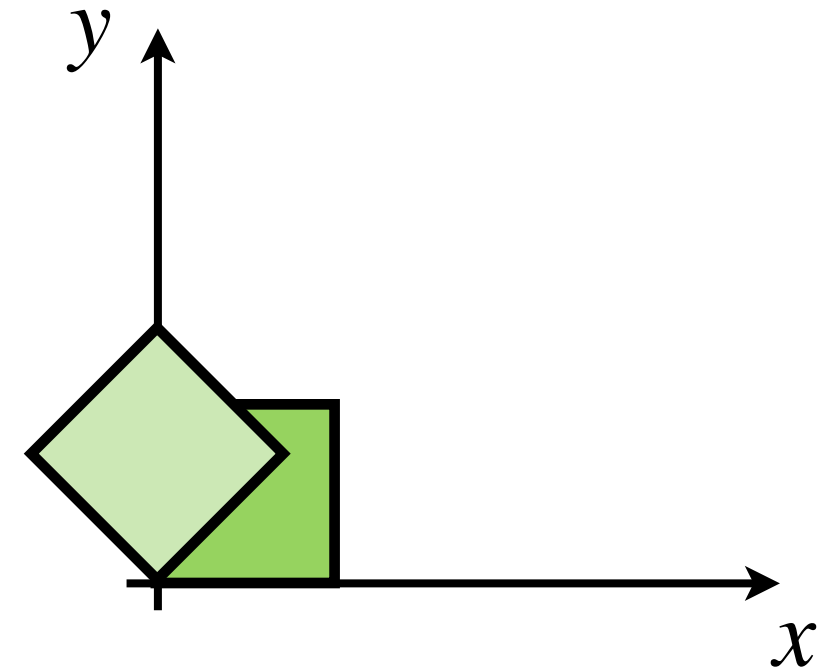
$$P_{s+r} = \begin{bmatrix} \cos \frac{\pi}{4} & -\sin \frac{\pi}{4} & 0 \\ \sin \frac{\pi}{4} & \cos \frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} P$$

$$P_{s+r} = \begin{bmatrix} 3 \cos \frac{\pi}{4} & -2 \sin \frac{\pi}{4} & 0 \\ 3 \sin \frac{\pi}{4} & 2 \cos \frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix} P$$

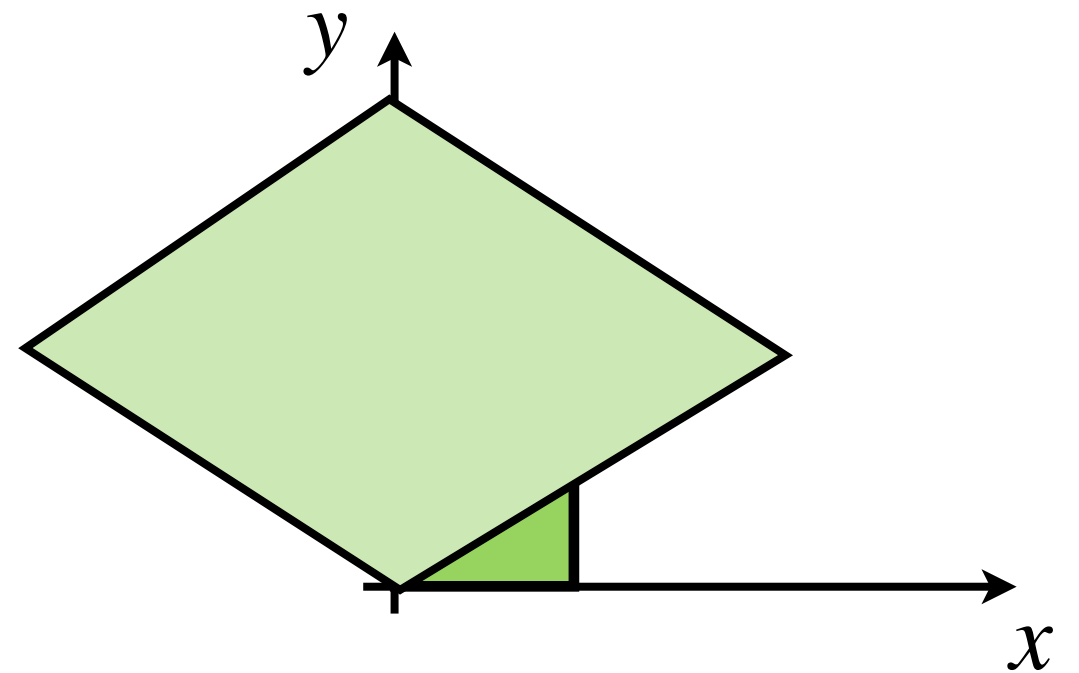
$$P_{s+r} = \mathbf{RS} \ P = \mathbf{MP}$$

Composite Transform II

- Now: **Swap order** of transforms
- First rotate $\pi/4$ degrees around z -axis



... then scale x by 3
and y by 2



Composite Transform II

- First rotate $\pi/4$ degrees around z -axis, then scale x by 3 and y by 2

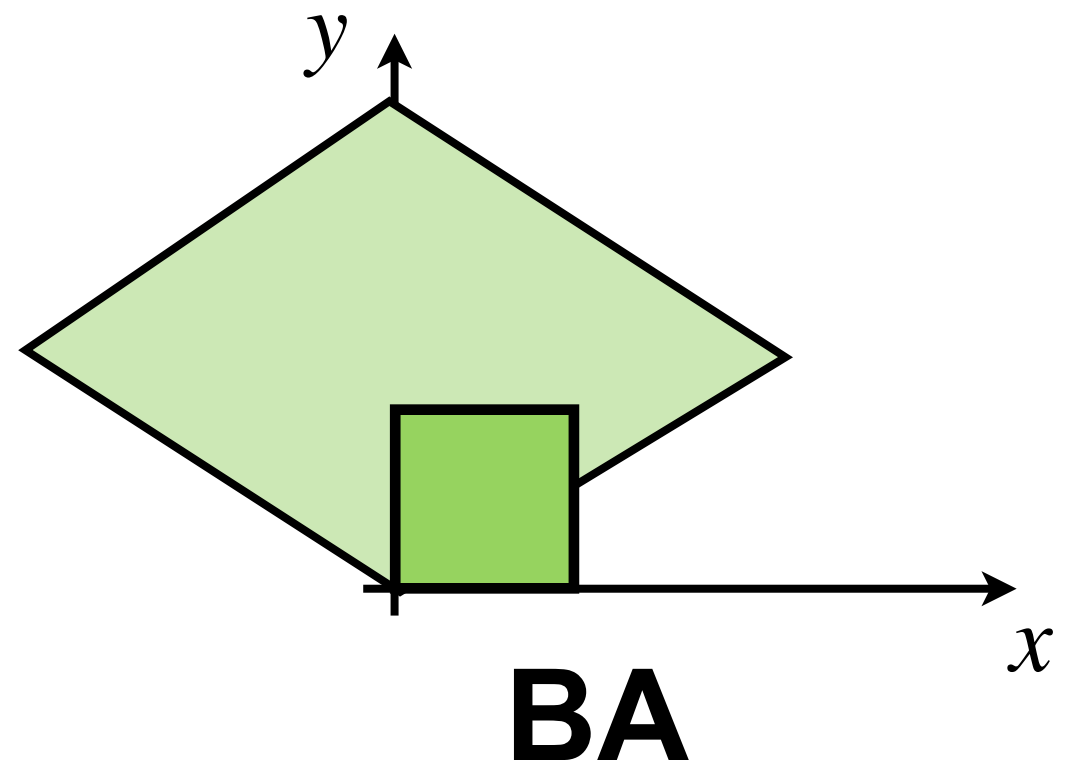
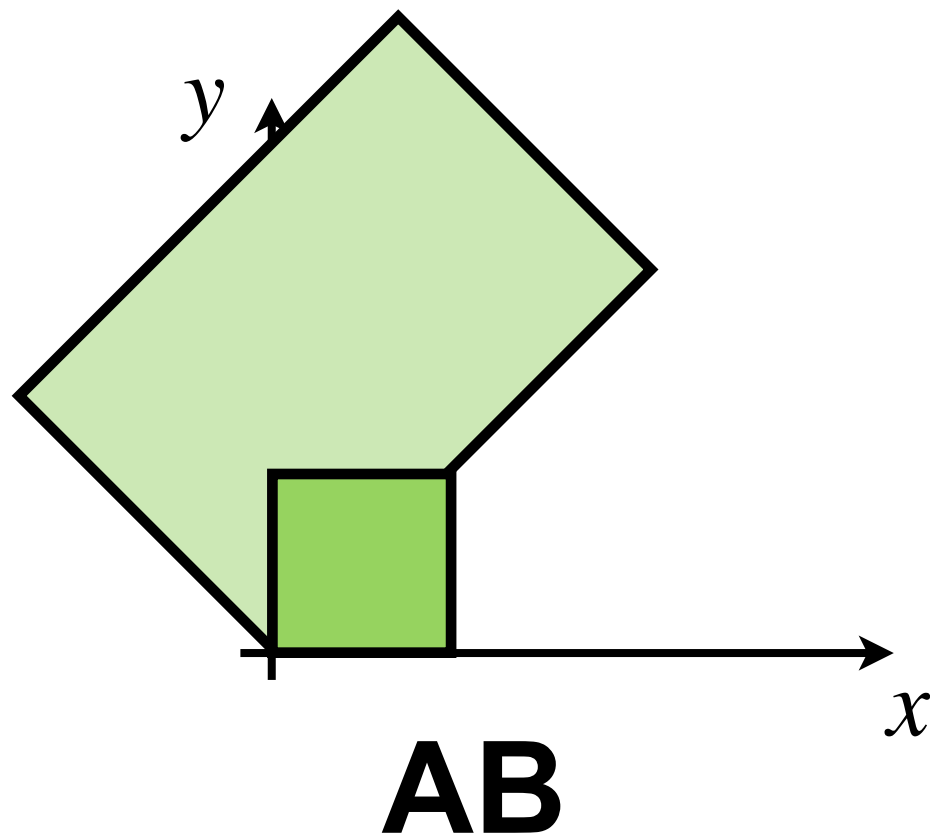
$$P_{r+s} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \frac{\pi}{4} & -\sin \frac{\pi}{4} & 0 \\ \sin \frac{\pi}{4} & \cos \frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix} P$$

$$P_{r+s} = \begin{bmatrix} 3 \cos \frac{\pi}{4} & -3 \sin \frac{\pi}{4} & 0 \\ 2 \sin \frac{\pi}{4} & 2 \cos \frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix} P$$

$$P_{r+s} = \mathbf{SR}P = \mathbf{N}P$$

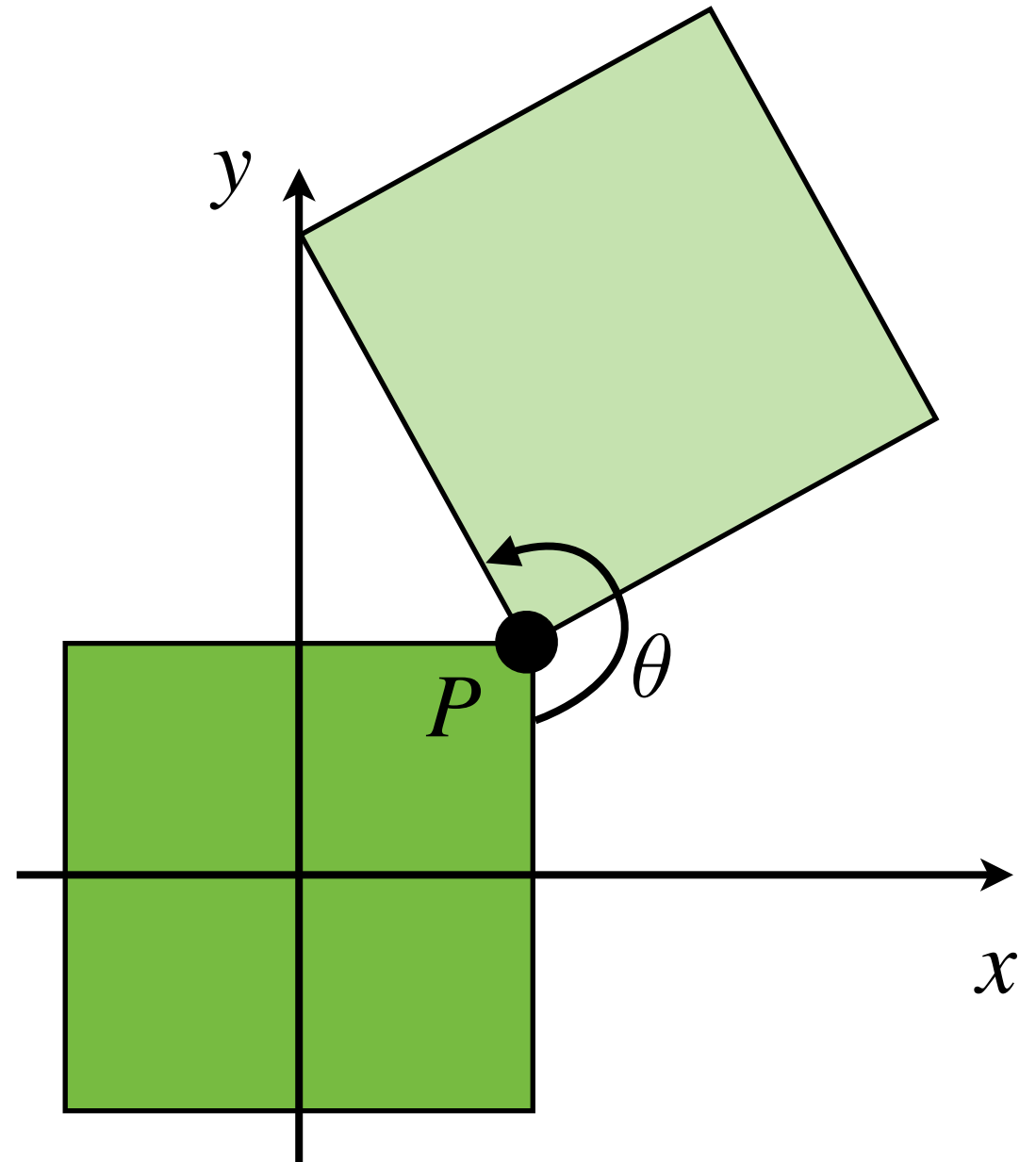
Composite Transforms

- The concatenation of transform matrices is a new matrix with the same dimensions
- Order matters. In general: **$AB \neq BA$**



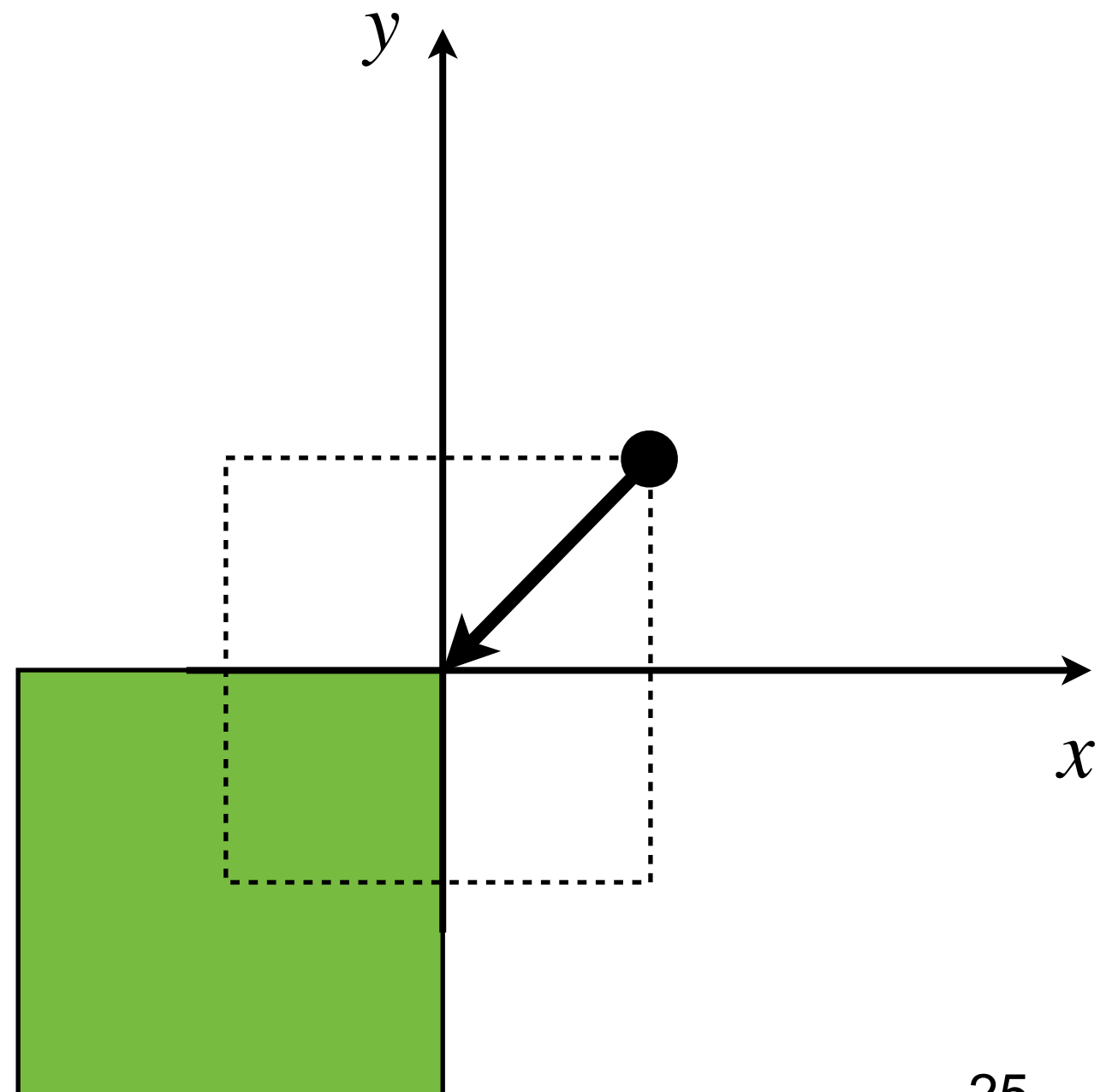
Composite Transforms III

- Example:
 - Z-rotation around point P



Combined Transforms III

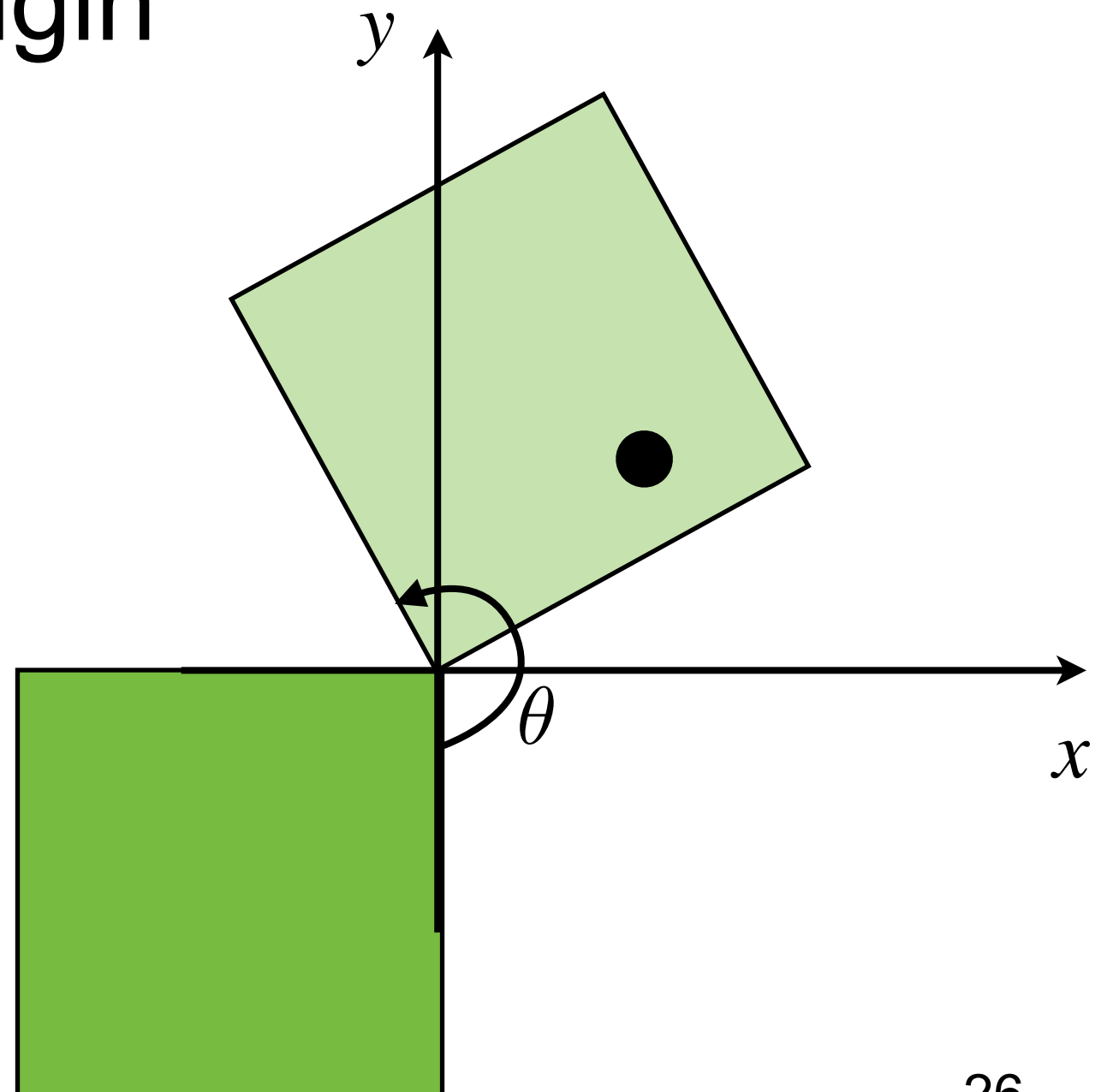
1. Translate to origin: $T(-P)$



Combined Transforms III

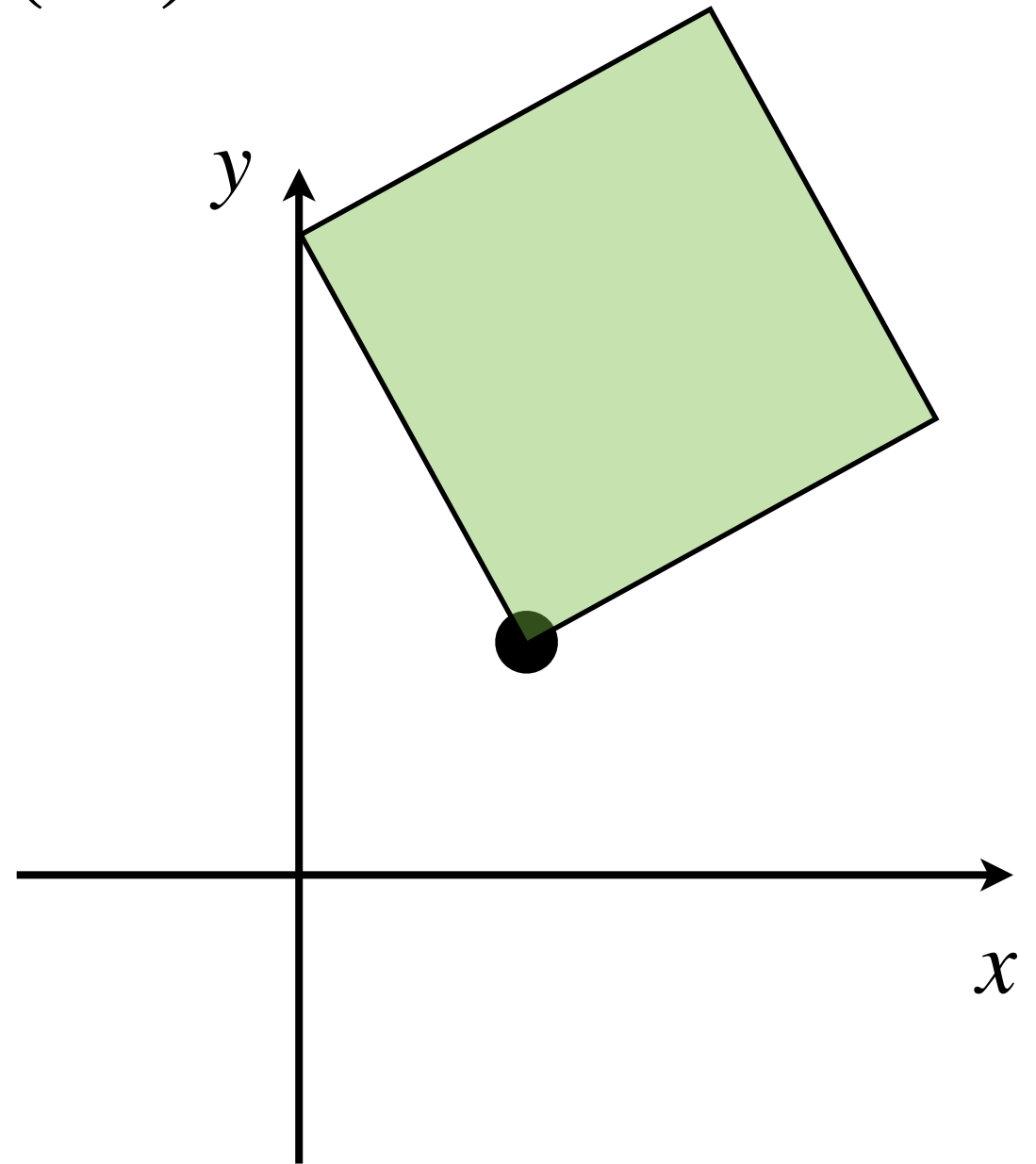
1. Translate to origin: $T(-P)$

2. Rotate around origin
 $R_z(\theta)$



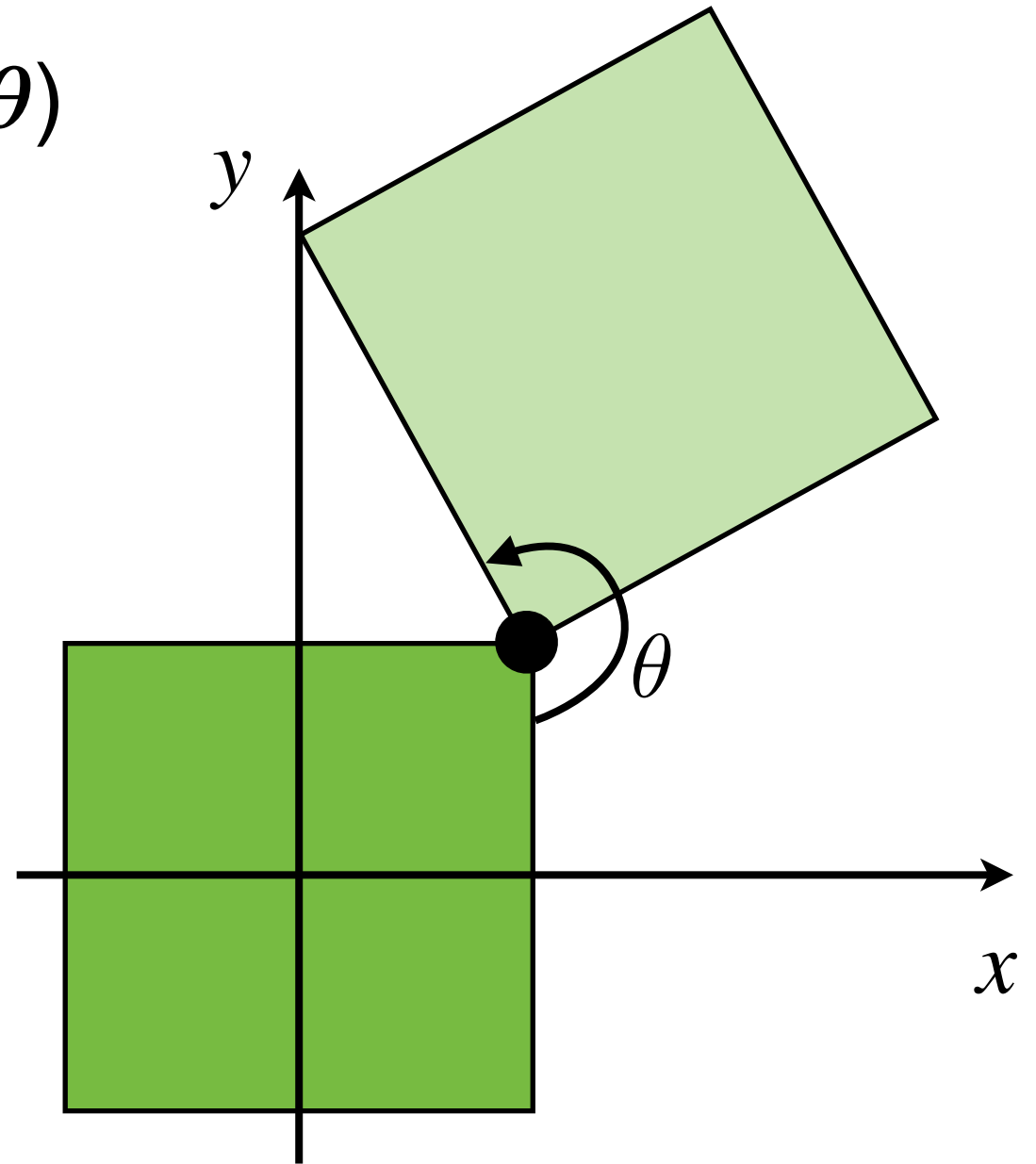
Combined Transforms III

1. Translate to origin: $T(-P)$
2. Rotate around origin
 $R_z(\theta)$
3. Translate back: $T(P)$



Combined Transforms III

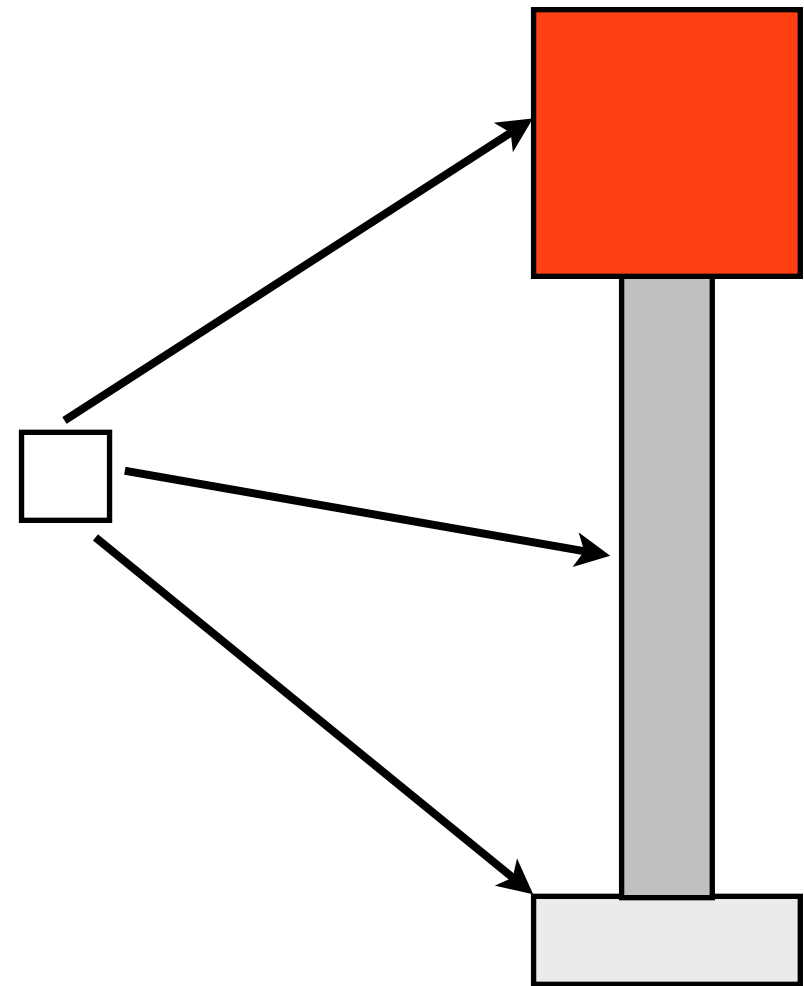
1. Translate to origin: $\mathbf{T}(-P)$
 2. Rotate around origin: $\mathbf{R}_z(\theta)$
 3. Translate back: $\mathbf{T}(P)$
- $\mathbf{M} = \mathbf{T}(P) \mathbf{R}_z(\theta) \mathbf{T}(-P)$



Modeling

- Simple Modeling
 - Use basic primitives
 - Apply transforms to create more complex objects

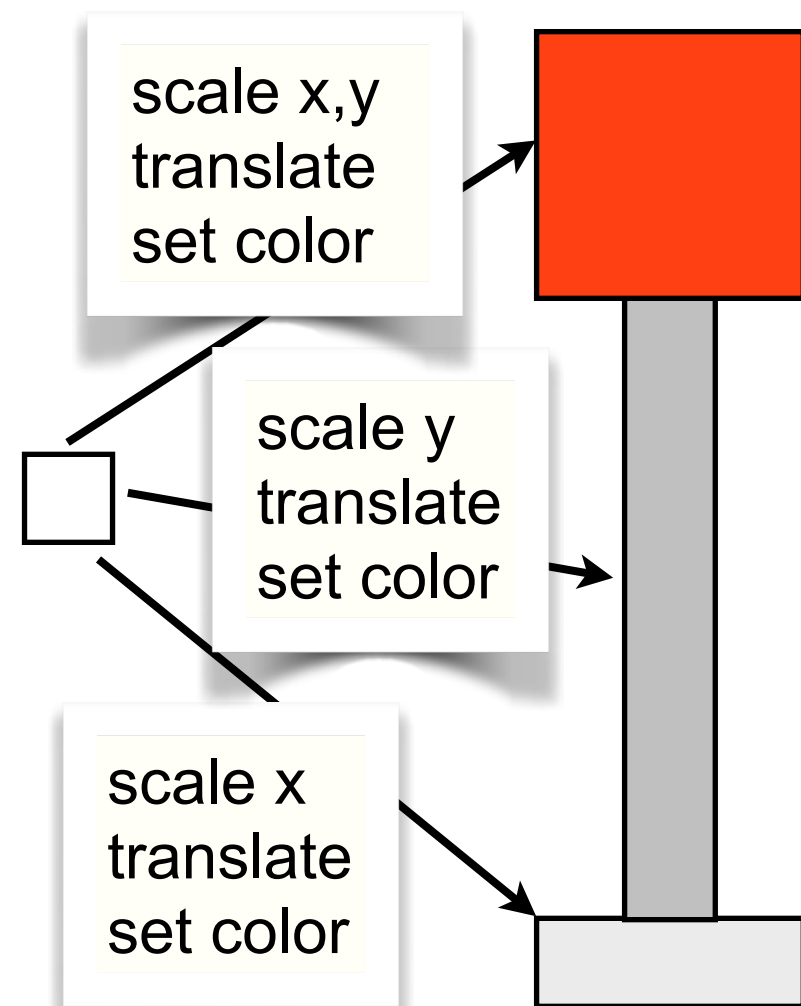
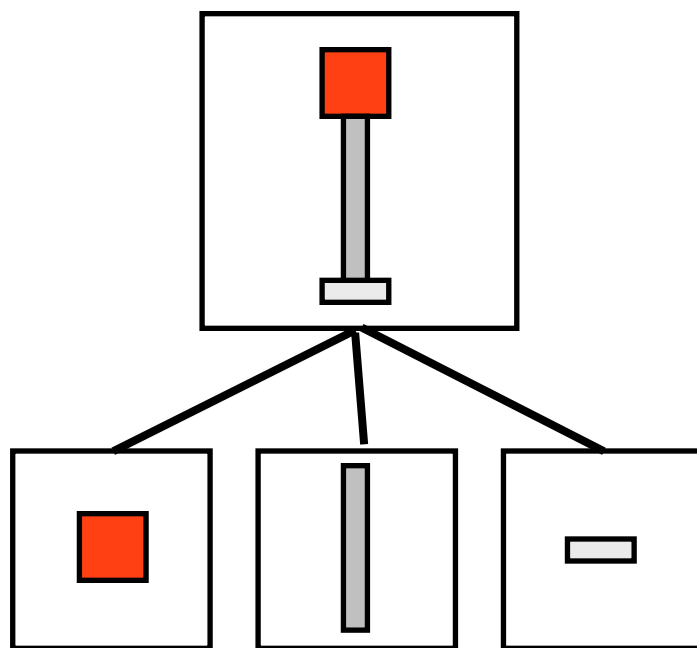
Use Blender or
Free student licenses of
Maya, 3D Studio Max, ...
<http://students.autodesk.com/>



Blender Modeling Introduction : <https://www.youtube.com/watch?v=7bHleRTEmZ0>

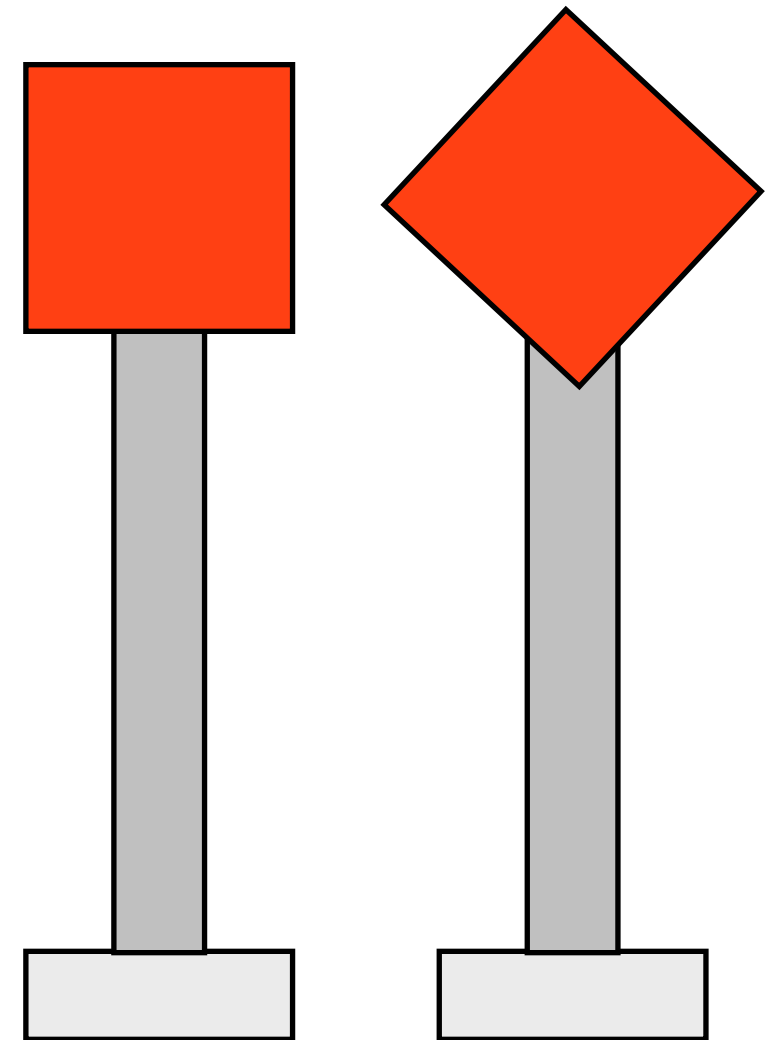
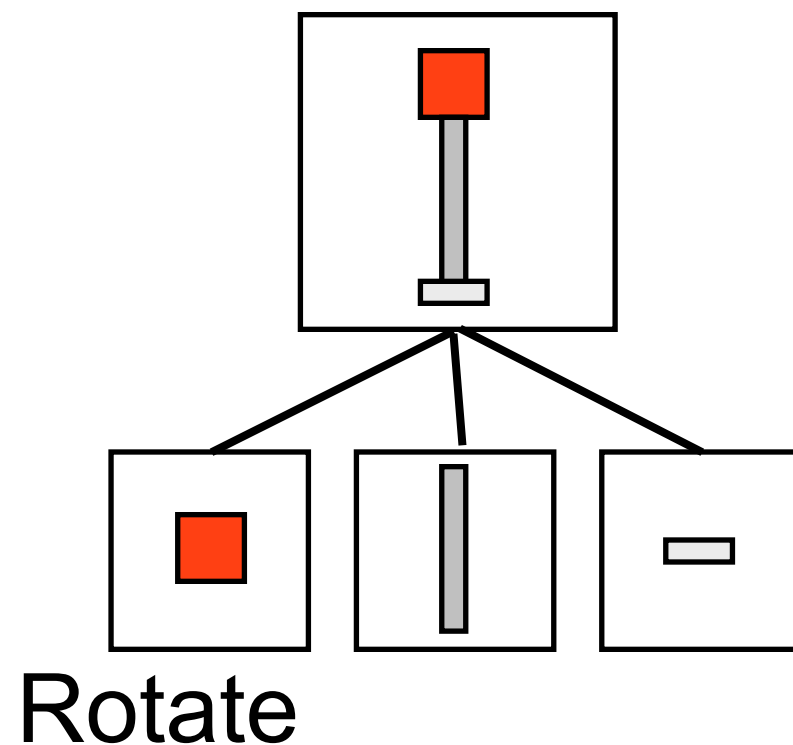
Modeling

- Simple Modeling
 - Use basic primitives
 - Apply transforms to create more complex objects
- Scene graph:



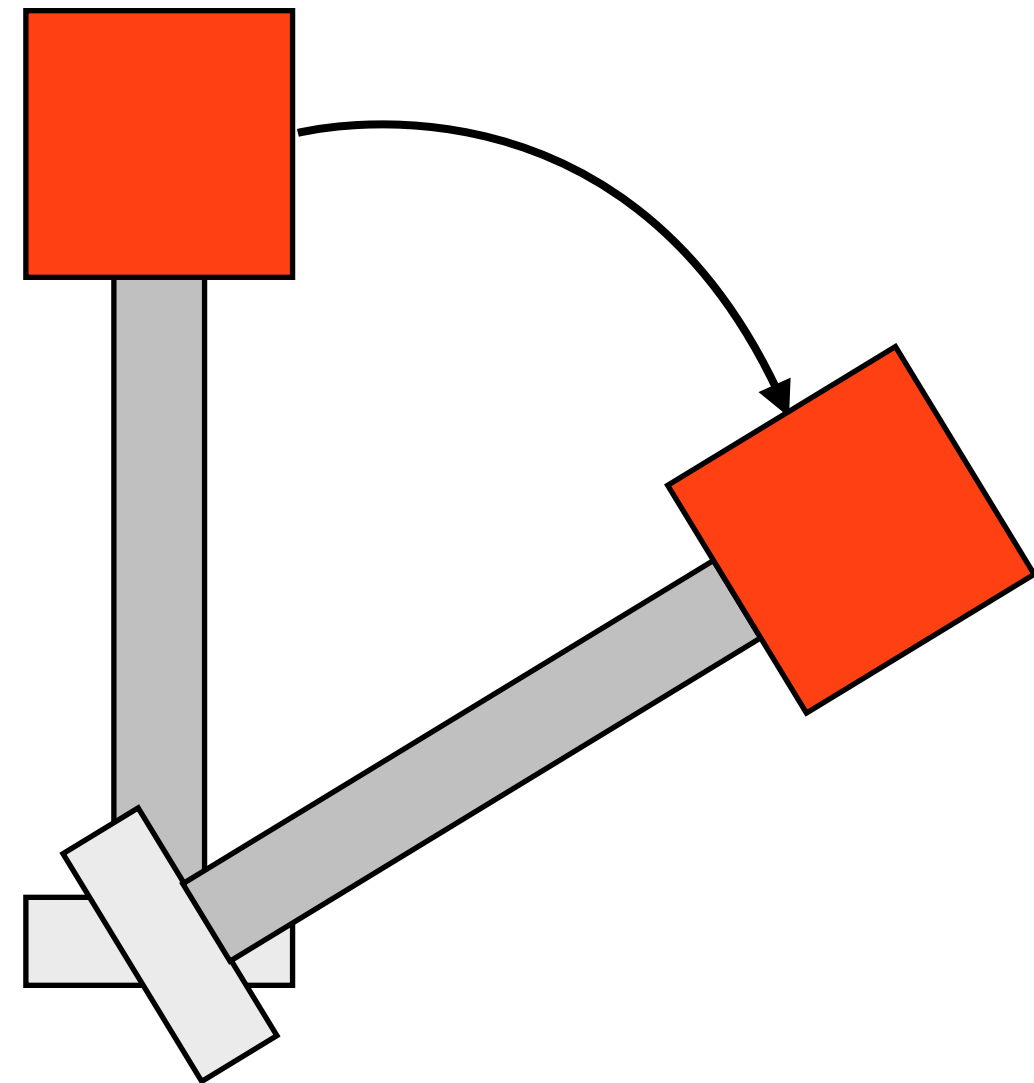
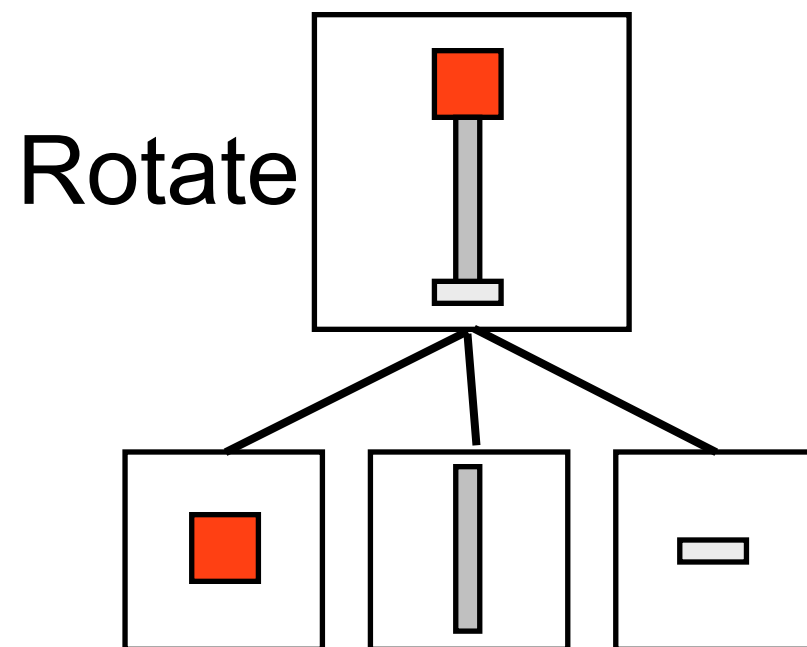
Hierarchical Transforms

- Create scene graph
 - Component of a scene
 - Each **node** can have a unique transform

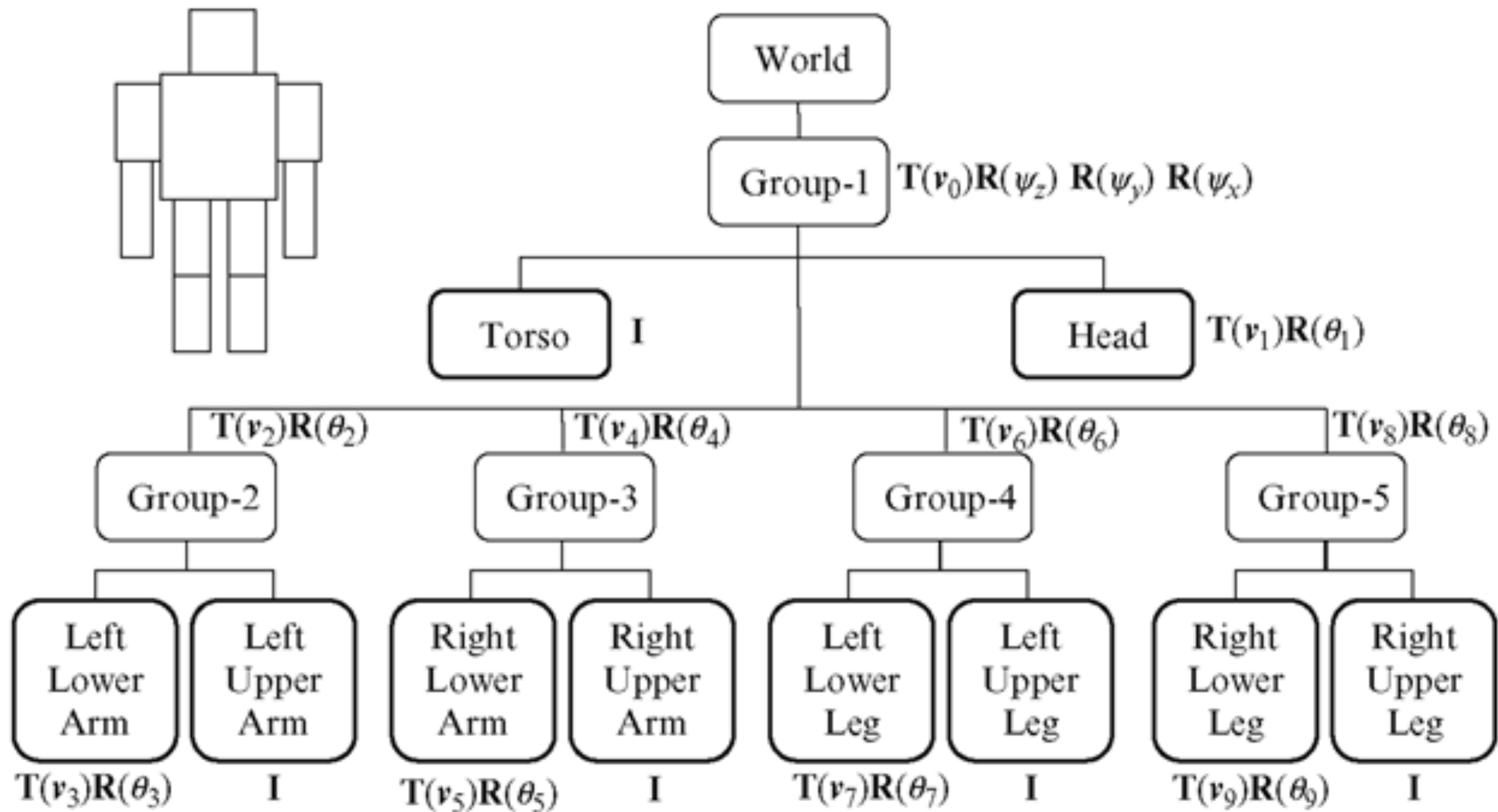


Hierarchical Transforms

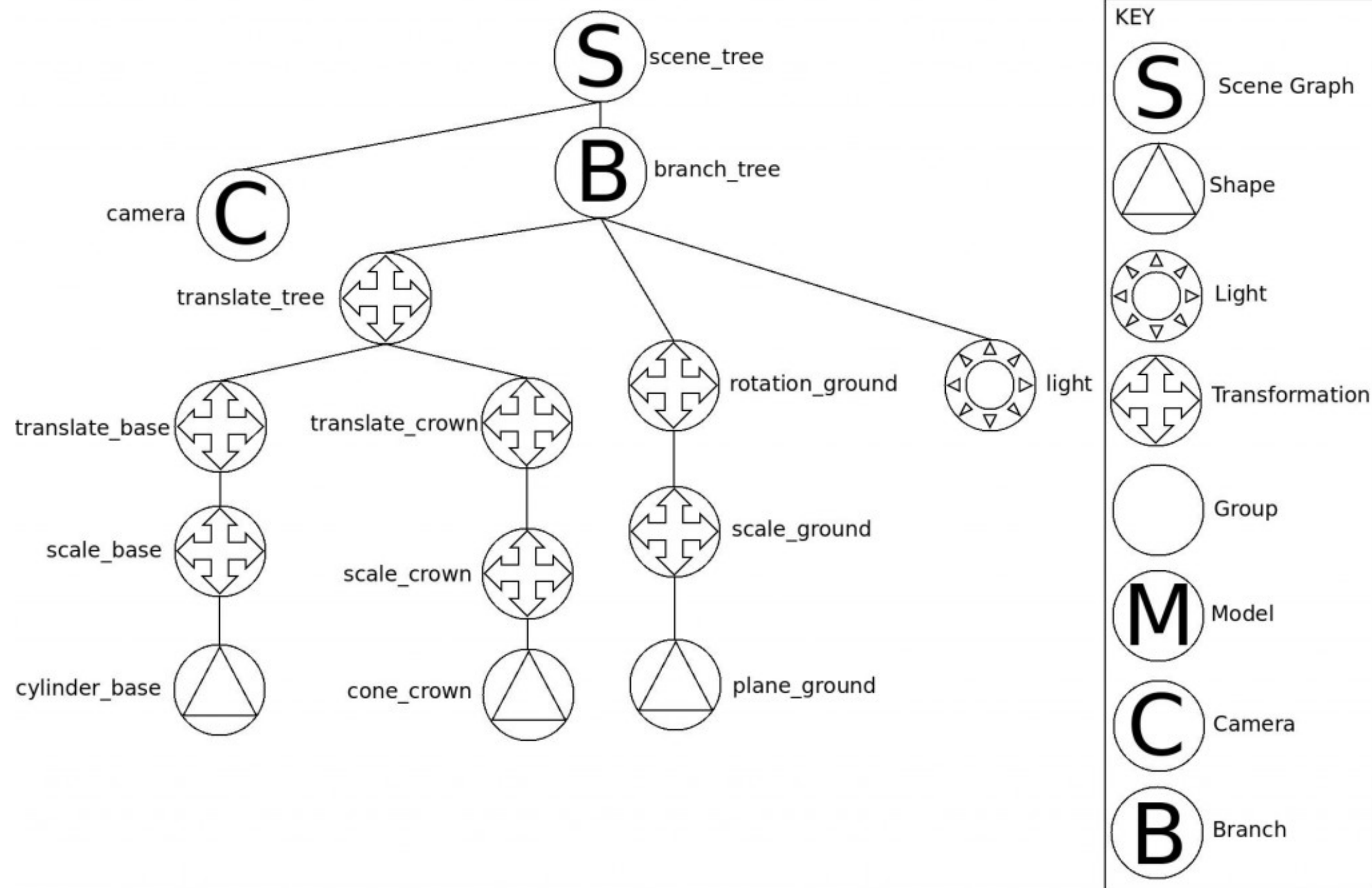
- Create new nodes
 - Apply transformation to nodes instead of individual parts



Scene Graph Example

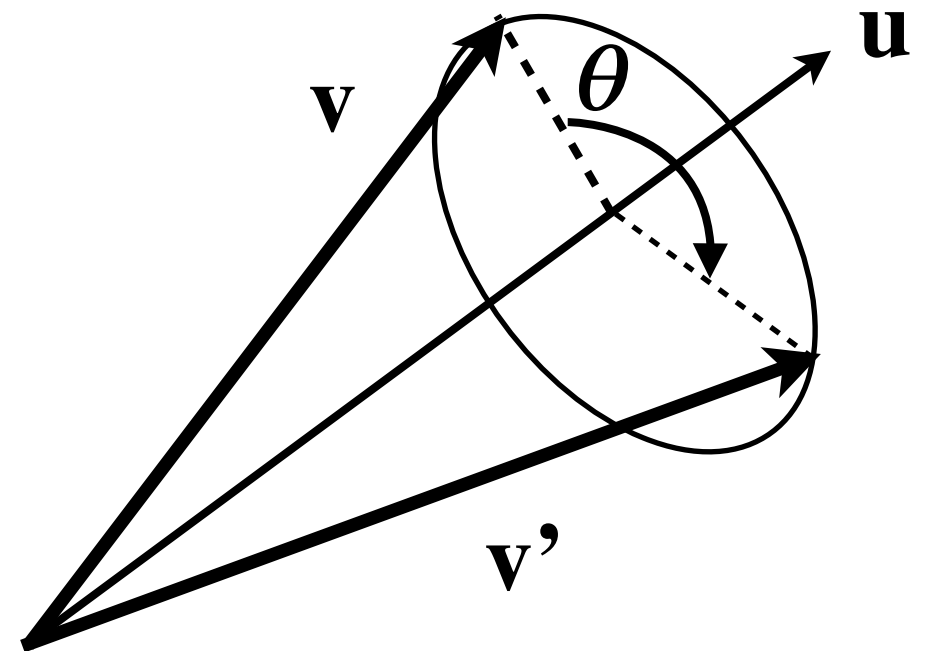


Scene Graph Example



Rotation around arbitrary vector

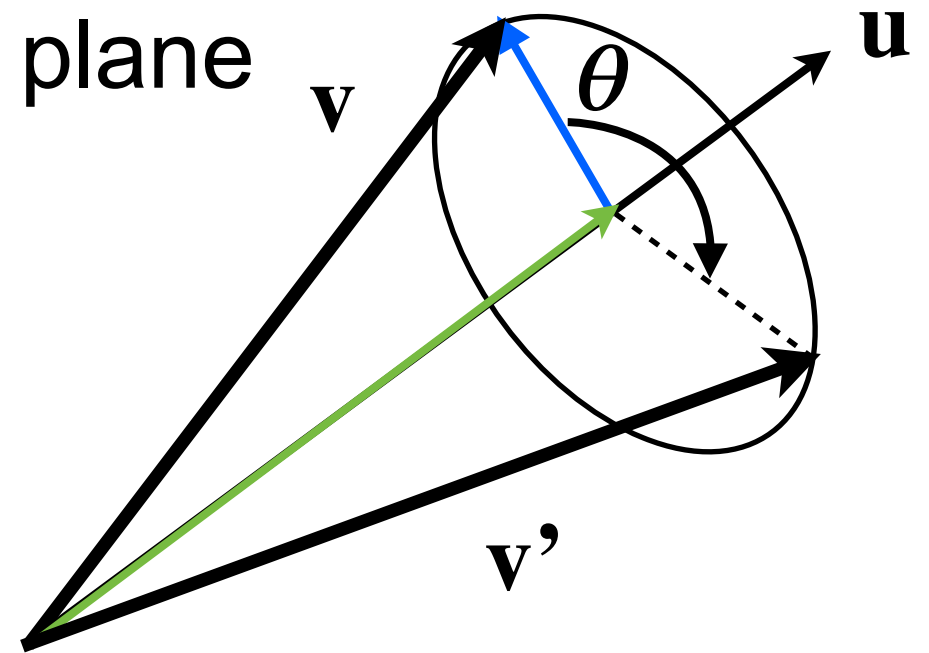
- Rotate vector \mathbf{v} θ degrees around the vector \mathbf{u}



Rotation around arbitrary vector

- Rotate vector \mathbf{v} θ degrees around the vector \mathbf{u}
 - Find plane orthogonal to \mathbf{u}
 - Find component of \mathbf{v} in this plane
 - Rotate the projection

Introduce unit vector notation: $\hat{\mathbf{u}} = \frac{\mathbf{u}}{|\mathbf{u}|}$



$$\mathbf{v}' = \cos(\theta) (\mathbf{v} - (\hat{\mathbf{u}} \cdot \mathbf{v}) \hat{\mathbf{u}}) + \sin(\theta) (\hat{\mathbf{u}} \times \mathbf{v}) + (\hat{\mathbf{u}} \cdot \mathbf{v}) \hat{\mathbf{u}}$$

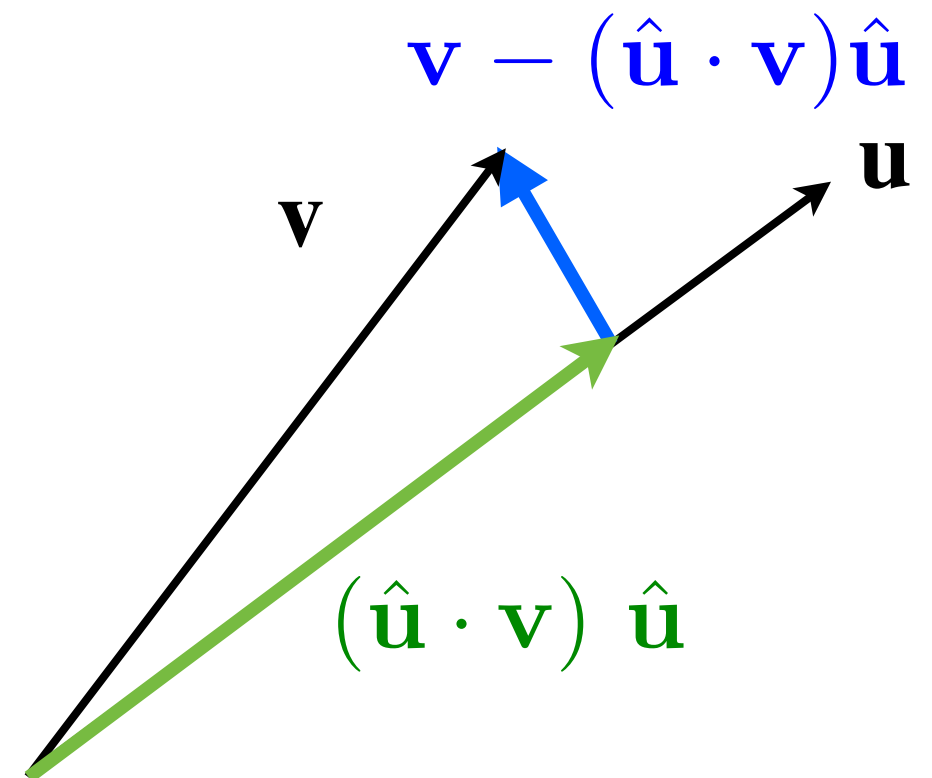
Rotation around arbitrary vector

- Find plane orthogonal to \mathbf{u}
 - Decompose \mathbf{v} into one component along \mathbf{u} (projection on \mathbf{u}):

$$(\hat{\mathbf{u}} \cdot \mathbf{v}) \hat{\mathbf{u}}$$

... and another component perpendicular to \mathbf{u}

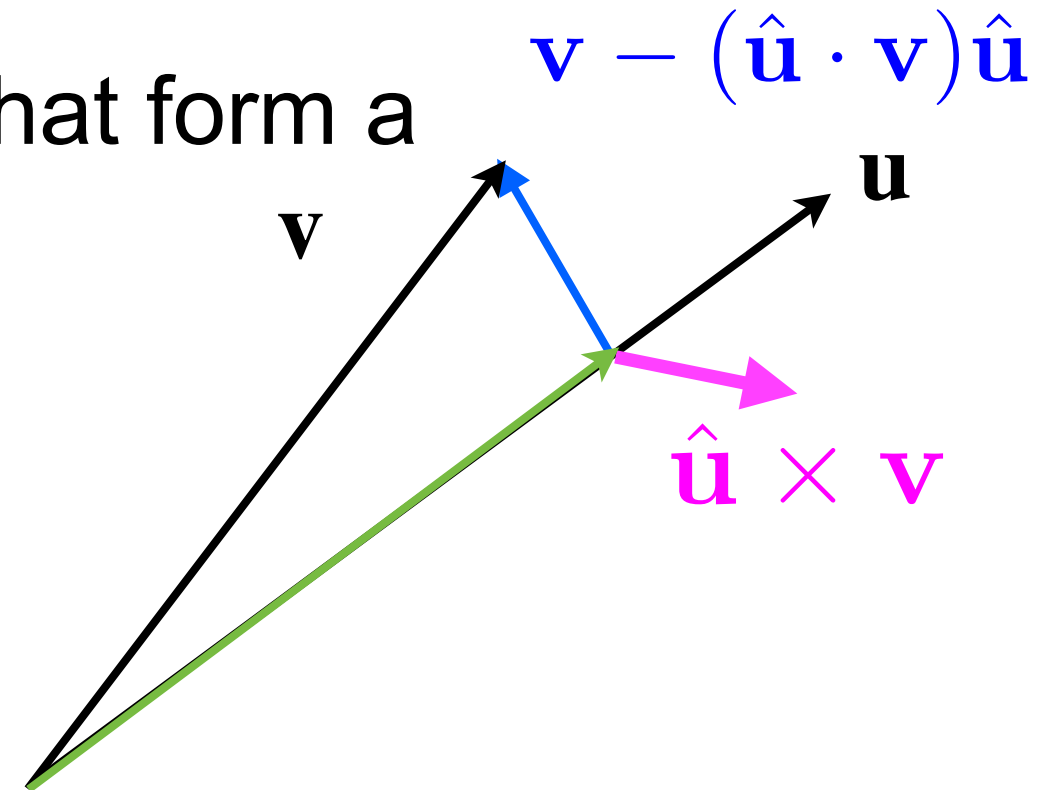
$$\mathbf{v} - (\hat{\mathbf{u}} \cdot \mathbf{v}) \hat{\mathbf{u}}$$



Rotation around arbitrary vector

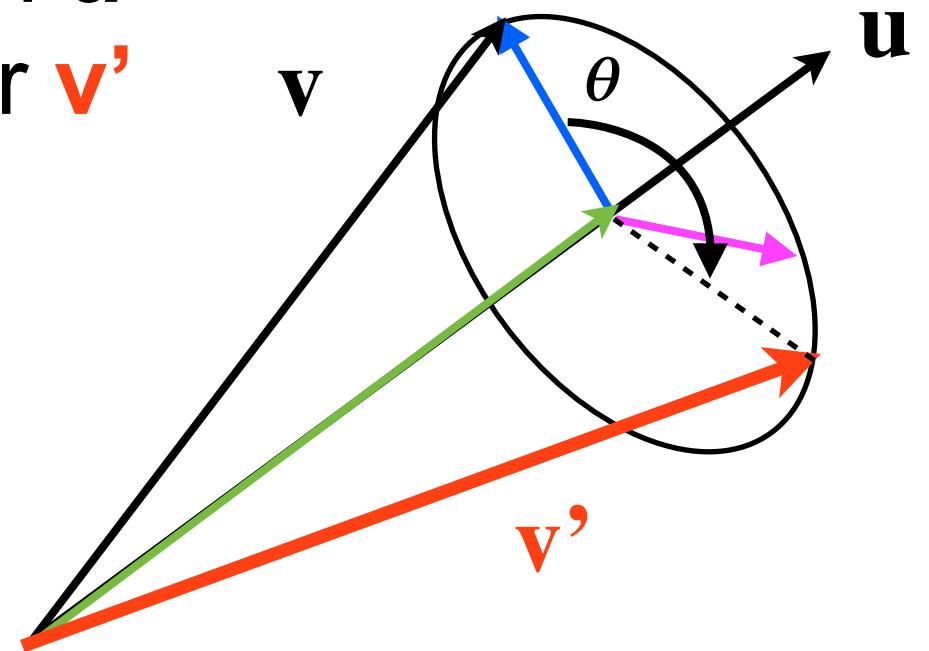
- Find plane orthogonal to \mathbf{u}
 - Find vector that is perpendicular to both \mathbf{u} and \mathbf{v} (use cross product)
 - We now have two vectors that form a plane orthogonal to \mathbf{u}

$$\mathbf{v} - (\hat{\mathbf{u}} \cdot \mathbf{v})\hat{\mathbf{u}} \quad \text{and} \quad \hat{\mathbf{u}} \times \mathbf{v}$$



Rotation around arbitrary vector

- Rotate within this plane
 - Rotate using the two basis vectors $\mathbf{v} - (\hat{\mathbf{u}} \cdot \mathbf{v})\hat{\mathbf{u}}$ and $\hat{\mathbf{u}} \times \mathbf{v}$ in the plane
 - Add in the projection of \mathbf{v} on \mathbf{u} $(\hat{\mathbf{u}} \cdot \mathbf{v})\hat{\mathbf{u}}$ to get the final vector \mathbf{v}'



$$\mathbf{v}' = \cos(\theta)(\mathbf{v} - (\hat{\mathbf{u}} \cdot \mathbf{v})\hat{\mathbf{u}}) + \sin(\theta) (\hat{\mathbf{u}} \times \mathbf{v}) + (\hat{\mathbf{u}} \cdot \mathbf{v})\hat{\mathbf{u}}$$

Homogeneous Coordinates

Homogeneous Coordinates

- Transformations of vectors and points

$$\mathbf{v}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

$$P' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

- Rotations and scalings can be multiplied together, but translations are a special case
- We want a unified representation

Vectors and points

- Remember: vectors are not points

- A vector: $\mathbf{v} = (v_x, v_y, v_z)$

- A point $P = P_o + (P_x, P_y, P_z)$
origin + vector

Homogeneous Coordinates

- Add a fourth dimension
 - The coordinate is 1 for points, 0 for vectors
 - Vector: $\mathbf{v} = (v_x, v_y, v_z, 0)$
 - Point: $P = (P_x, P_y, P_z, 1)$
- All affine transformations can be represented as matrix multiplications in homogeneous coordinates

Homogeneous Coordinates

- Transformation

$$\mathbf{v} = (v_x, v_y, v_z, 0)$$

$$P = (P_x, P_y, P_z, 1)$$

Point

$$P' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & t_x \\ a_{10} & a_{11} & a_{12} & t_y \\ a_{20} & a_{21} & a_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

Vector

$$\mathbf{v}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdot \\ a_{10} & a_{11} & a_{12} & \cdot \\ a_{20} & a_{21} & a_{22} & \cdot \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

Homogeneous Coordinates

- All transforms, including translations are expressed as 4x4 matrices
- This is the format graphics hardware APIs (OpenGL, Direct 3D) specify transformations
- In upcoming lecture, we will see that homogeneous coordinates are also useful for projections

Translation

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translate point:

$$\mathbf{T}P = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x + t_x \\ P_y + t_y \\ P_z + t_z \\ 1 \end{bmatrix}$$

Translate vector:

$$\mathbf{T}\mathbf{v} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

Vectors are
unmodified by
translations!

Rotation and Scaling

- Rotation

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Scaling

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

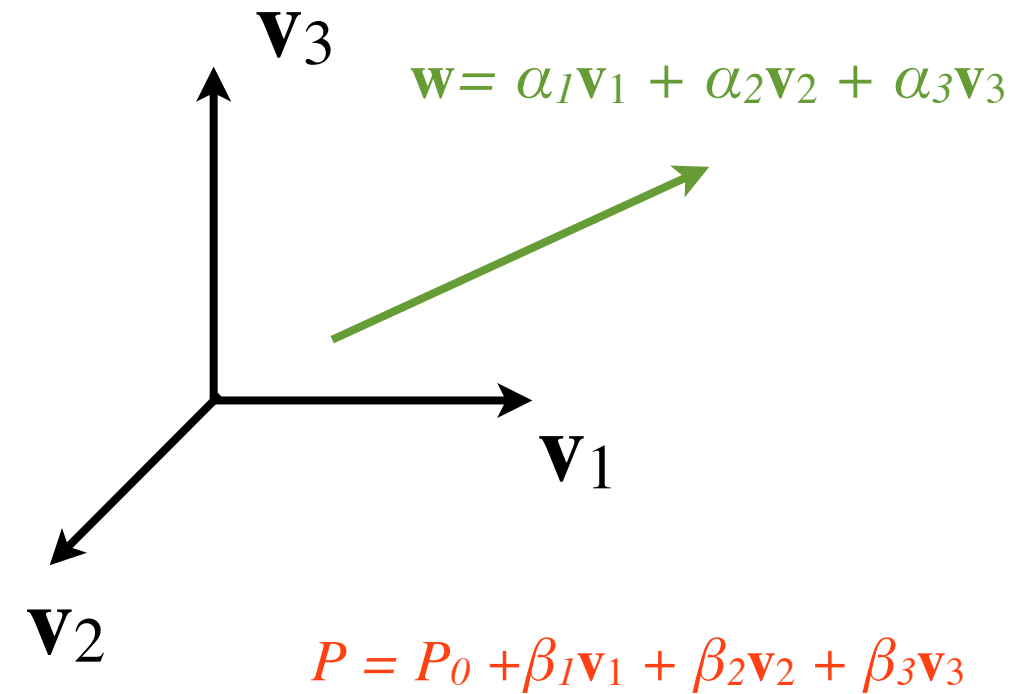
Coordinate Systems

Basis and Frame

- Basis

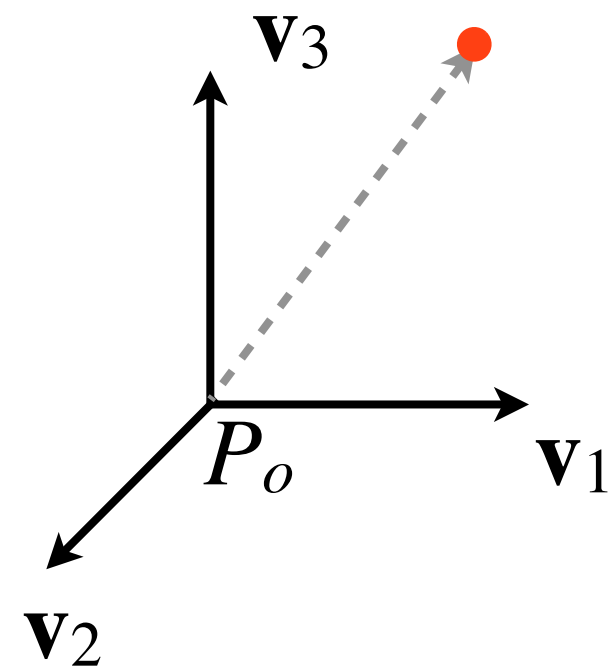
- Any 3D **vector** \mathbf{w} can be expressed in terms of three linearly independent vectors \mathbf{v}_1 , \mathbf{v}_2 and \mathbf{v}_3

$$\mathbf{w} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3$$



- Frame

- Basis vectors + origin(P_o)
 - A 3D **point** P expressed as $P = P_o + \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \beta_3 \mathbf{v}_3$



Point in Euclidean Frame

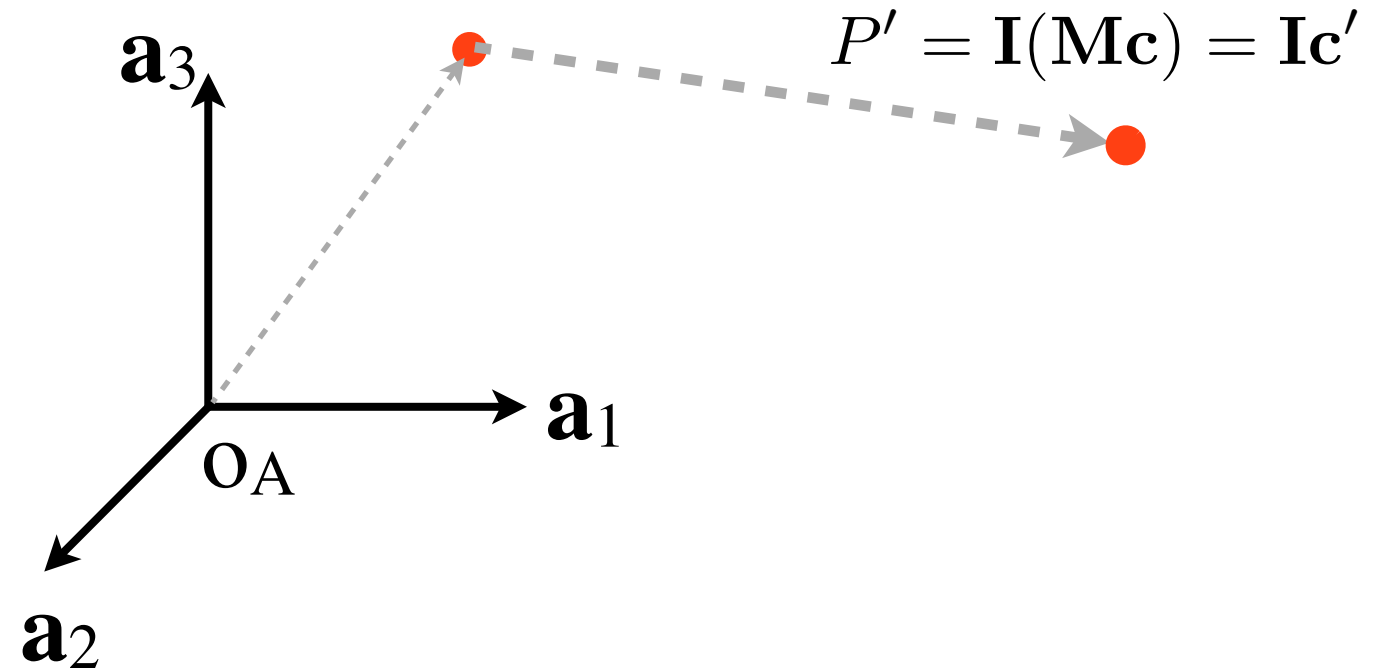
$$\begin{aligned}
 P &= \begin{matrix} \text{origin} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{matrix} + \alpha_1 \begin{matrix} \text{basis vectors} \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{matrix} + \alpha_2 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix} \text{ coordinate vector} \\
 &= \mathbf{Ic}
 \end{aligned}$$

Coordinate Frames

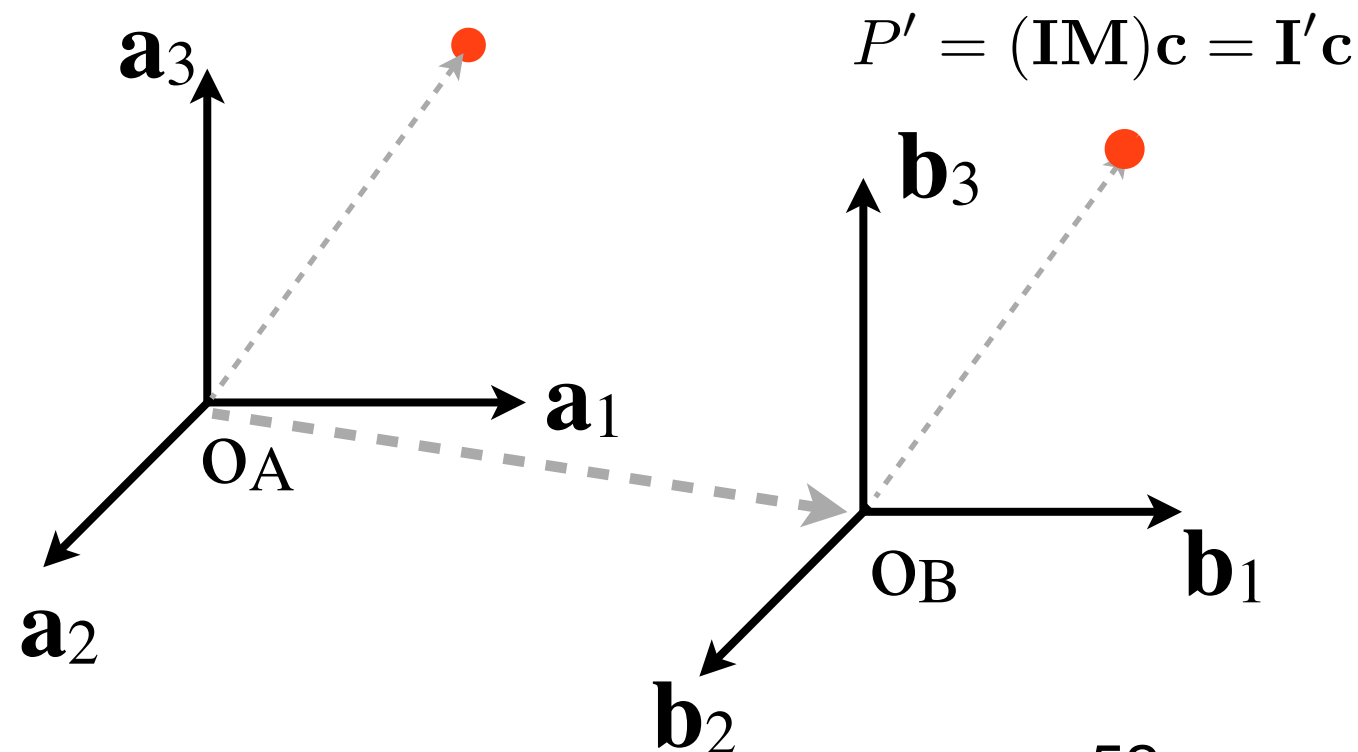
- Point P in frame \mathbf{I} (Euclidean frame) $P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix} = \mathbf{I}\mathbf{c}$
- Now, transform P with matrix \mathbf{M} $P' = \mathbf{I}\mathbf{M}\mathbf{c}$
- **Two interpretations:**
 1. Transform the point in the frame \mathbf{I} $P' = \mathbf{I}(\mathbf{M}\mathbf{c}) = \mathbf{I}\mathbf{c}'$
 2. Transform the frame itself $P' = (\mathbf{I}\mathbf{M})\mathbf{c} = \mathbf{I}'\mathbf{c}$

Coordinate Frames

1. Transform the point in the frame **I**

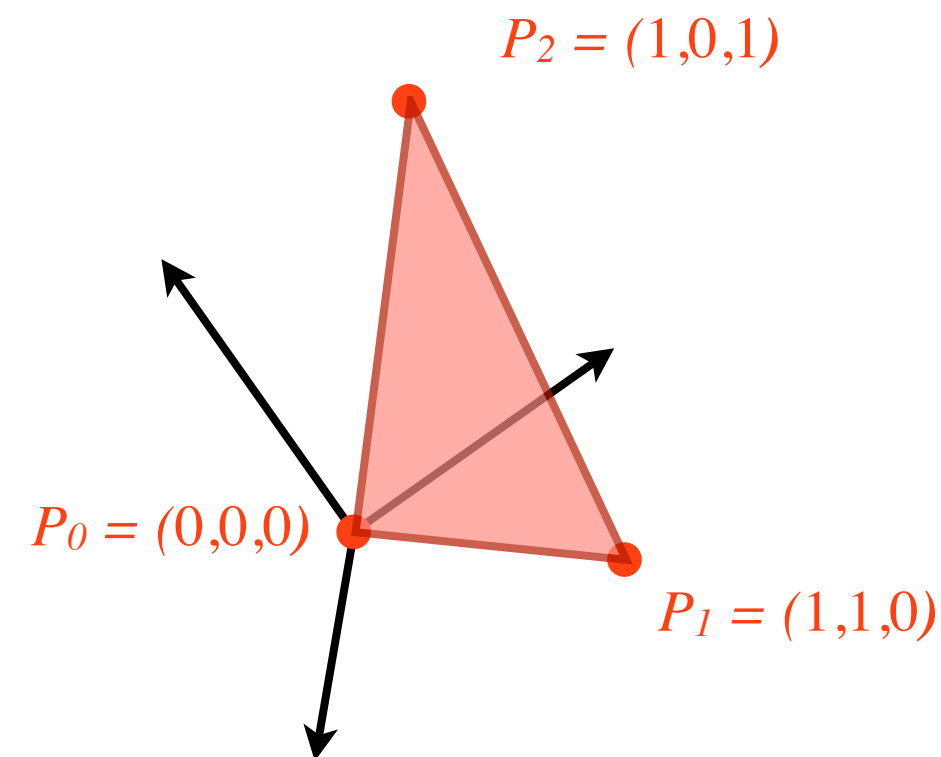
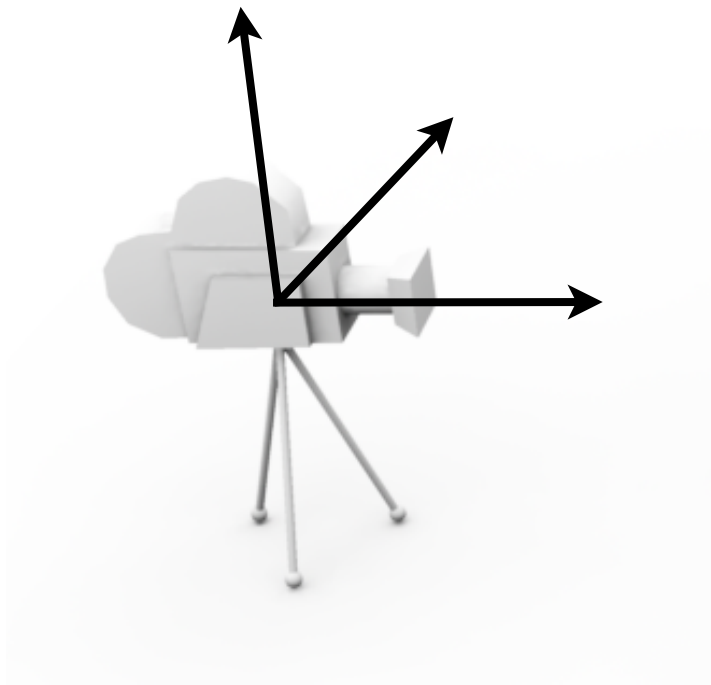


2. Transform the frame itself



Change of Coordinate System

- Problem at hand:
 - Given an object represented in a local coordinate system, how to express the object's coordinates from the camera's point of view?



Change of Coordinate System

- Point in frame **A**

$$P_A = \mathbf{A}\mathbf{c}_A$$

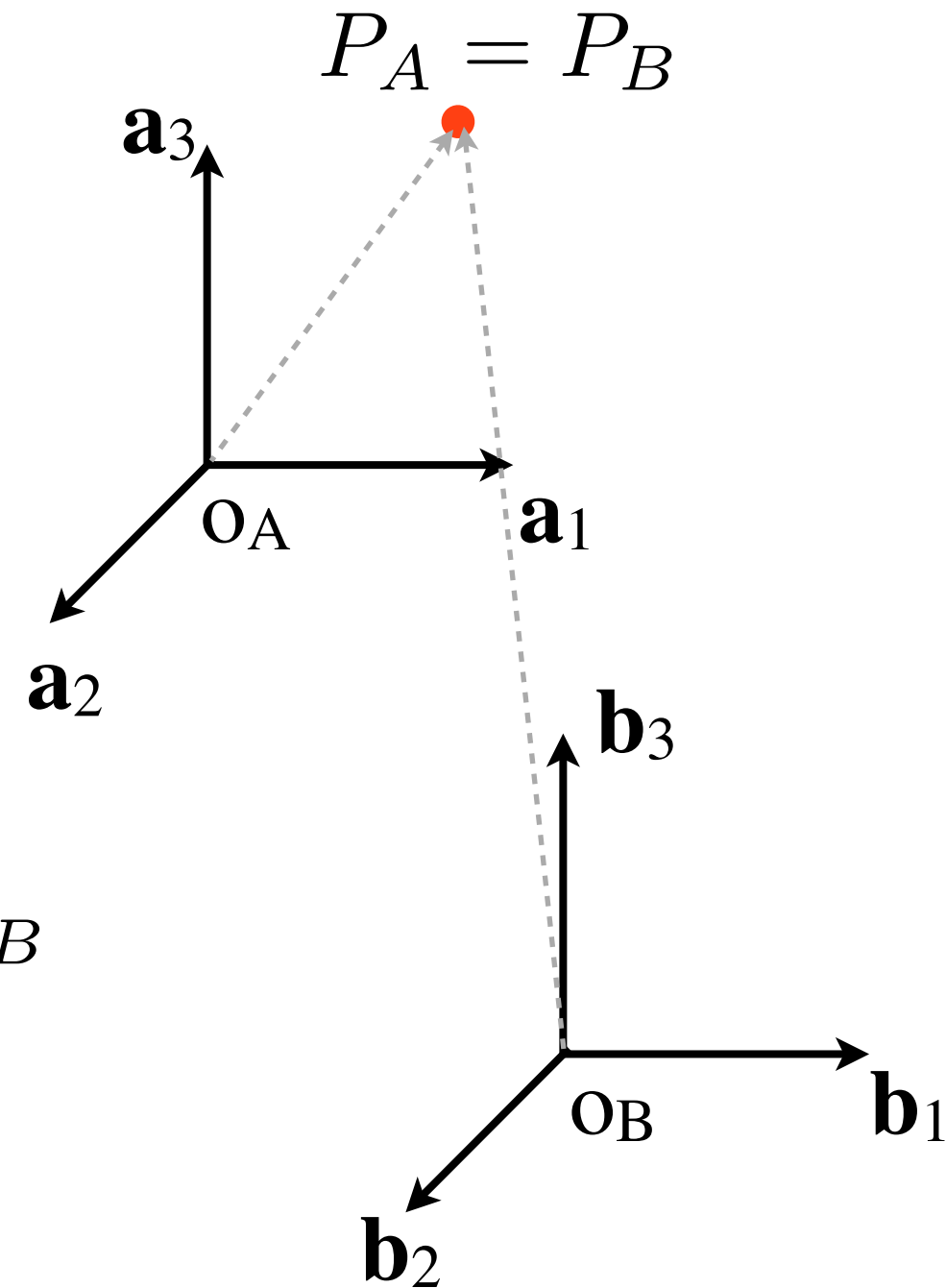
- Point in frame **B**

$$P_B = \mathbf{B}\mathbf{c}_B$$

- Same point:

$$P_A = P_B \implies \mathbf{c}_A = \mathbf{A}^{-1}\mathbf{B}\mathbf{c}_B$$

- If **A** is the Euclidean basis
basis $\mathbf{c}_A = \mathbf{B}\mathbf{c}_B$

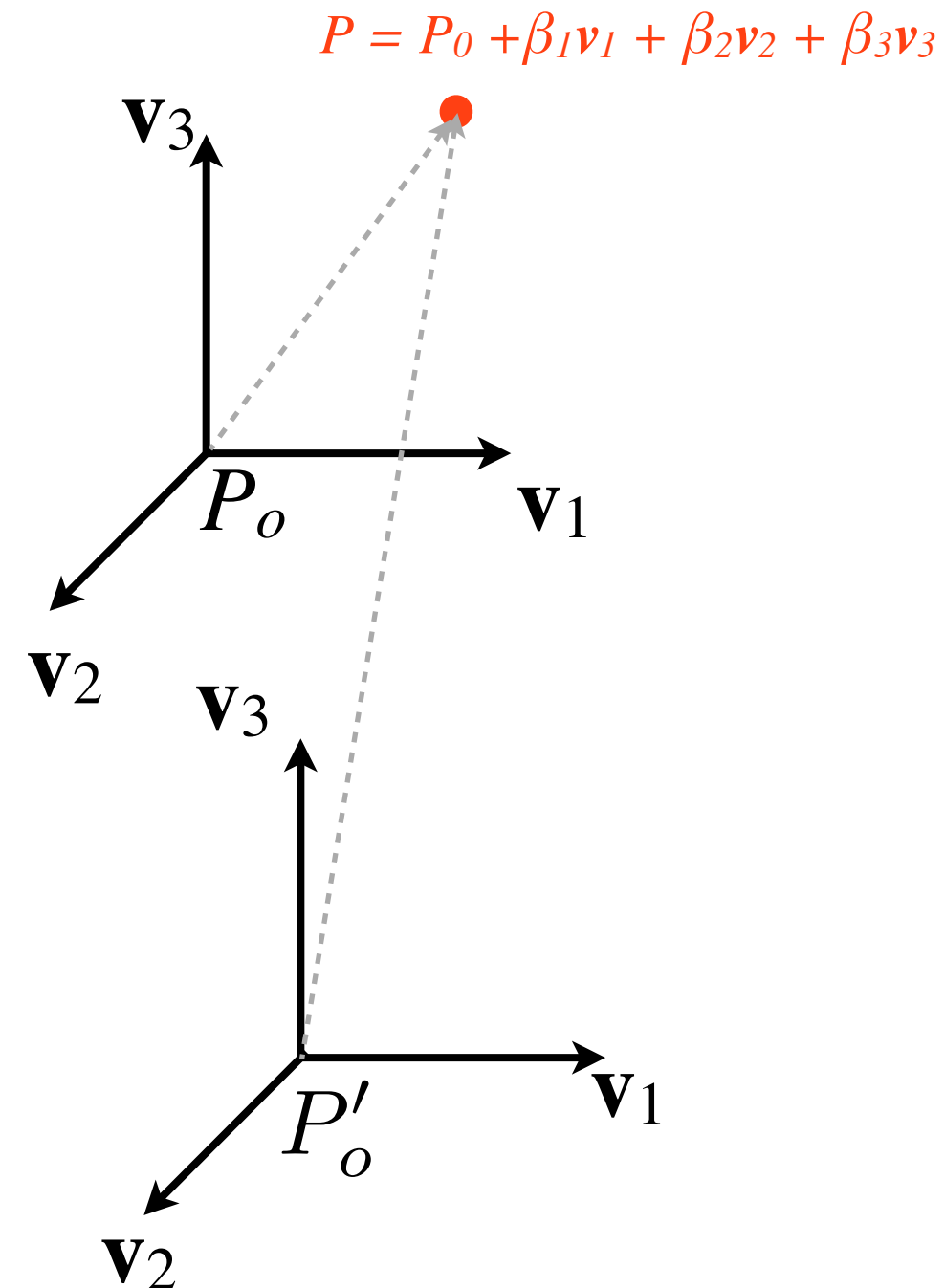


Change of Coordinate System

- Translation

- Simply change origin of coordinate frame
- Basis vectors are not affected by pure translation

$$\begin{aligned} P &= P_o + \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \beta_3 \mathbf{v}_3 \\ &= P'_o + \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \beta_3 \mathbf{v}_3 \end{aligned}$$



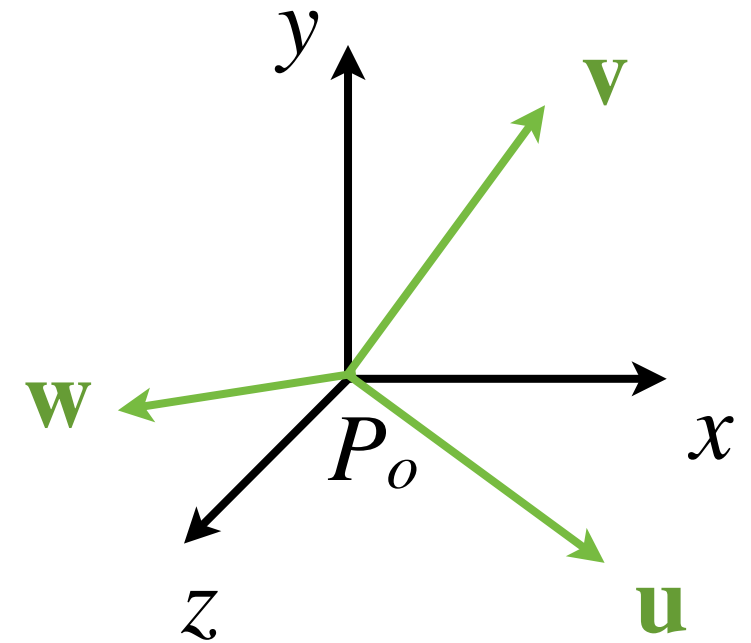
Change of Coordinate System

- Rotation and scaling
 - Express $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ in the basis $(1,0,0)^T, (0,1,0)^T$ and $(0,0,1)^T$

$$\mathbf{u} = (u_x, u_y, u_z)^T$$

$$\mathbf{v} = (v_x, v_y, v_z)^T$$

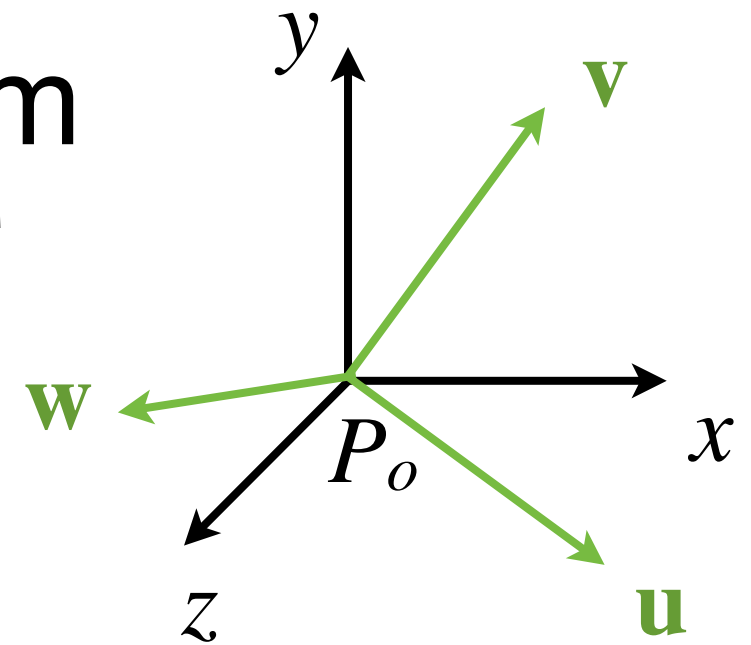
$$\mathbf{w} = (w_x, w_y, w_z)^T$$



$$\begin{bmatrix} | & | & | \\ \mathbf{u} & \mathbf{v} & \mathbf{w} \\ | & | & | \end{bmatrix}_{3 \times 3} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}_{3 \times 3}$$

Change of Coordinate System

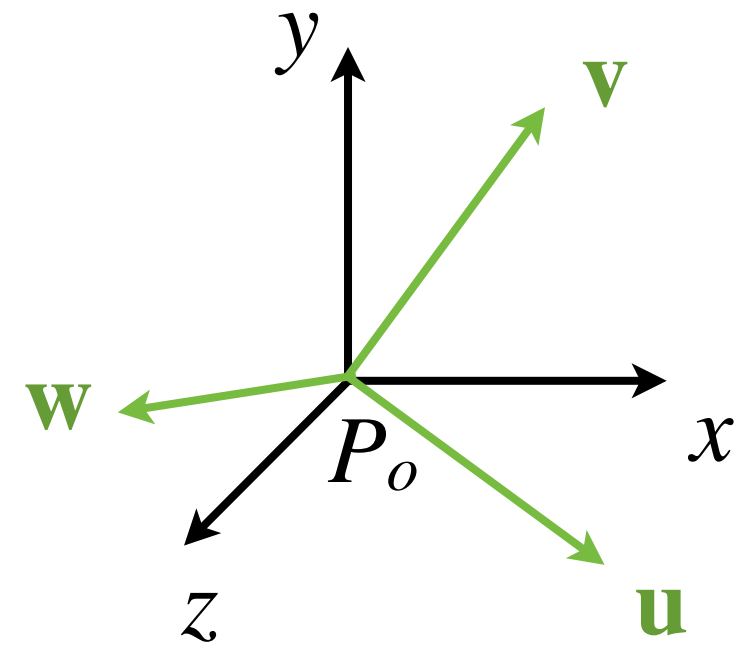
- The matrix \mathbf{M} transforms **from** the coordinate system $(1,0,0)^T$, $(0,1,0)^T$ and $(0,0,1)^T$ **to** the system $(\mathbf{u}, \mathbf{v}, \mathbf{w})$
- Example:



$$\mathbf{M} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \quad \mathbf{M} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{u}$$

Change of Coordinate System

- The matrix \mathbf{M}^{-1} transforms from $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ to the Euclidean system $(1,0,0)^T$, $(0,1,0)^T$ and $(0,0,1)^T$



- Example:

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

$$\mathbf{M}\mathbf{x} = \mathbf{u}$$

$$\mathbf{M}^{-1}\mathbf{M}\mathbf{x} = \mathbf{M}^{-1}\mathbf{u}$$

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{u}$$

$$\mathbf{M}^{-1} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}^{-1}$$

Change of Coordinate System

- Align origins
 - Translation
- Create rotation matrix from basis vectors
 - Express each basis vector of one coordinate system as a linear combination of the basis vectors of the other coordinate system

Orthonormal Basis

- Remember: If $\mathbf{u}\mathbf{v}\mathbf{w}$ is an orthonormal basis, we have: $\mathbf{M}^{-1} = \mathbf{M}^T$

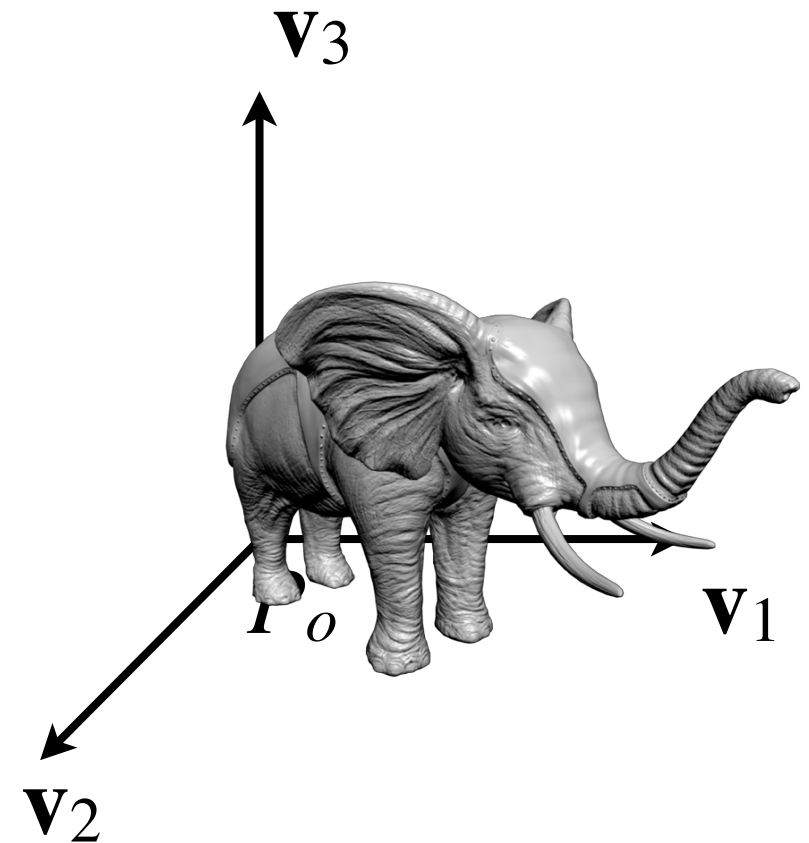
$$\mathbf{M} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}$$

$$\mathbf{M}^{-1} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}^{-1} = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix}$$

Common Coordinate Systems in Computer Graphics

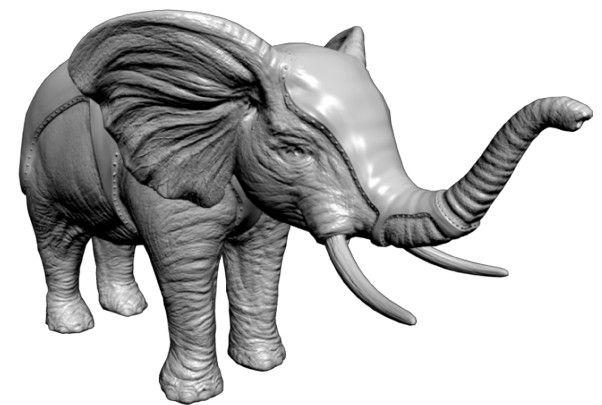
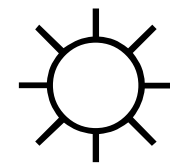
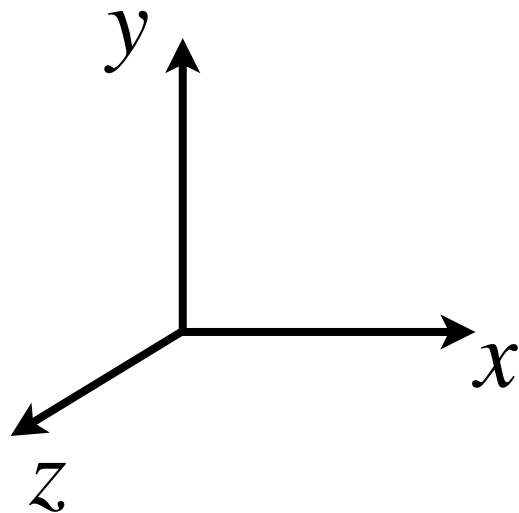
Object coordinates

- The local coordinate system the model is defined in
- Often called “Object space”, “Model space” or “Model coordinates”



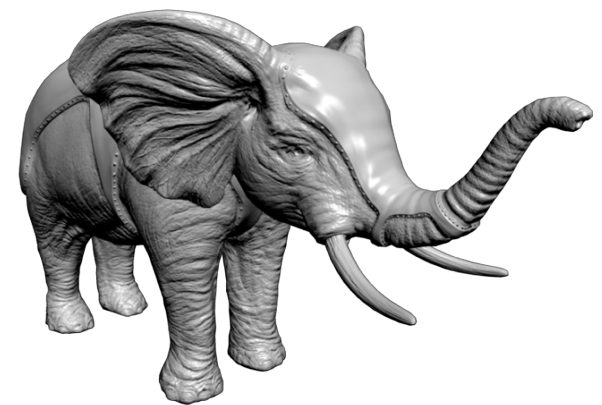
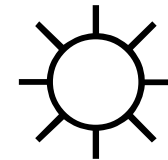
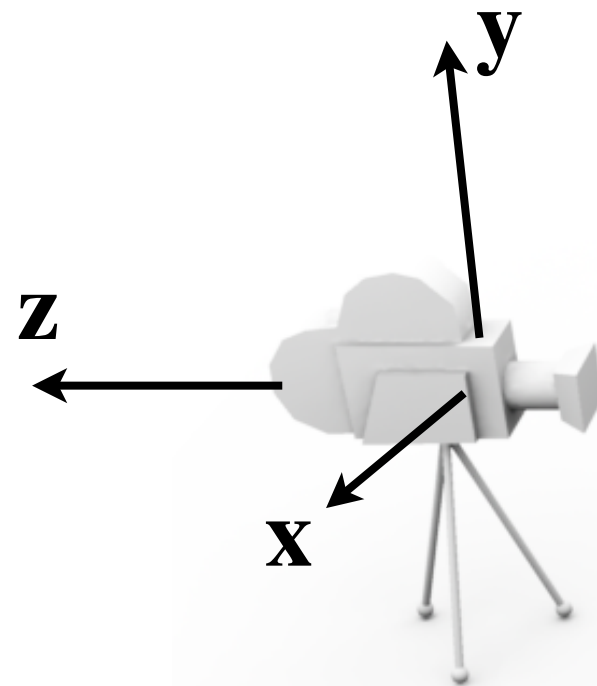
World coordinates

- Reference system in which all objects, lights and the camera are positioned
- Often referred to as “World space”



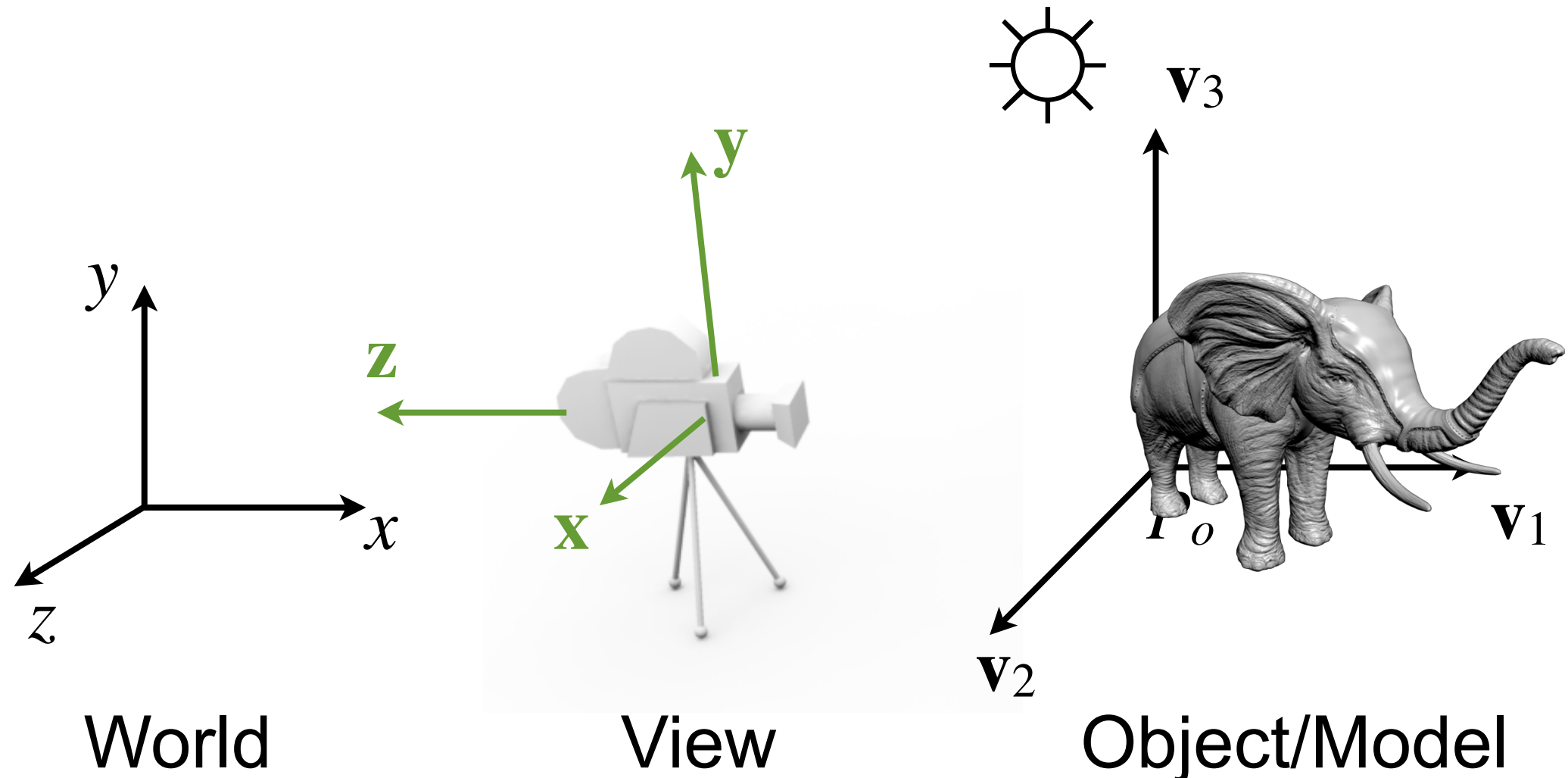
Eye (view) coordinates

- A coordinate system with the camera at center, looking along negative z
- “Camera space”



Change coordinate system

- Model or World matrix: From model to world
- View matrix: From world to view
- ModelView matrix: From model to view



Coordinate Systems

- Clip coordinates
 - after projection matrix has been applied
- Normalized Device Coordinates (NDC)
 - Projected position on screen $(x,y) : [-1,1] \times [-1,1]$
- Window (screen) coordinates
 - Pixel position

These coordinate systems will be explained in detail in a later lecture. For now: Note the names

VFX Breakdowns

- DNEG Avengers: Endgame
 - <https://youtu.be/pTffQIFFYR8?si=OVicNXn7F8RpTN4->
- Framestore Avengers: Infinity War
 - https://youtu.be/V5mS7BHmZJI?si=w9KiX6xuDu_ayTQ1

Next

- Wednesday Seminar
 - Assignment 1 - Solar System
- Lab 0 - set up - optional
- Next week
 - Monday lecture - Parametric Surfaces & Animation
 - Labs - Assignment approval sessions
 - Sign up now!

Extras

- Graphics pipeline matrices
- <http://www.realtimerendering.com/udacity/?load=demo/unit7-view-pipeline.js>