Introduction to Computer Graphics

2. Transformations

I-Chen Lin

National Yang Ming Chiao Tung University

Textbook: E.Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson

Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson

Intended Learning Outcomes

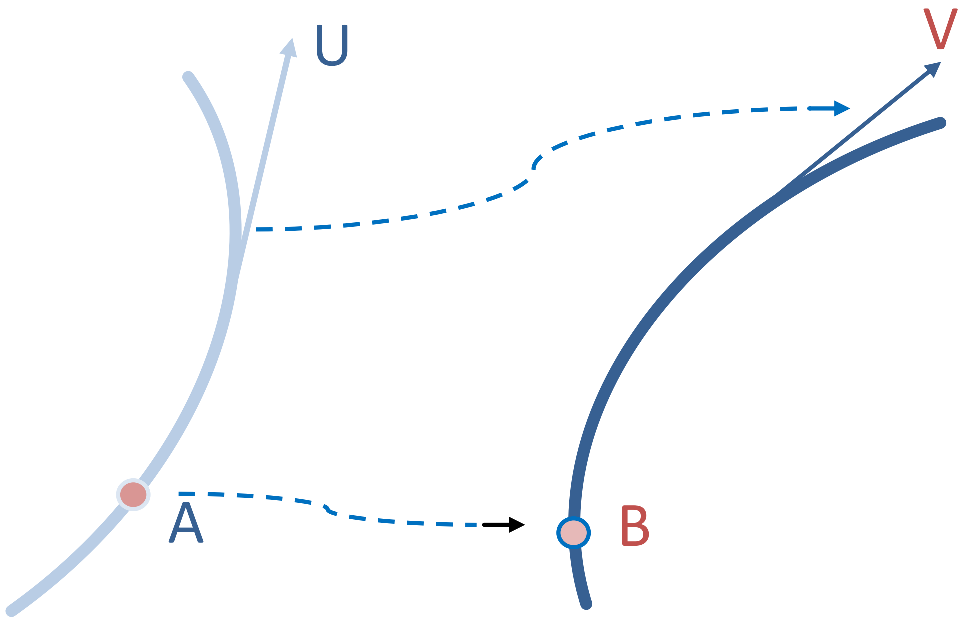
- On completion of this chapter, a student will be able to:
  - Identify the basic transformations for 3D objects.
  - Apply the basic transformations for object movement in a 3D scene.
  - Explain and write the pseudo codes in OpenGL style with a sequence of transformations.

Outline

- Introduce standard transformations
  - Rotation
  - Translation
  - Scaling
  - Shear
- Derive homogeneous coordinate transformation matrices
- Learn to build arbitrary transformation matrices from simple transformations

General Transformations

- A transformation maps points to other points and/or vectors to other vectors



Affine Transformations

- A transformation that **preserves** lines and parallelism
  - maps parallel lines to parallel lines
- Characteristic of many physically important transformations
  - Rigid body transformations: **rotation**, **translation**
  - **Scaling**, **shear**

Translation

- Using the homogeneous coordinate representation in some frame

  $\boldsymbol{p} = [\, x\ y\ z\ 1]^T$

  $\boldsymbol{p}' = [\, x'\ y'\ z'\ 1]^T$

  $\boldsymbol{d} = [\, d_x\ d_y\ d_z\ 0]^T$

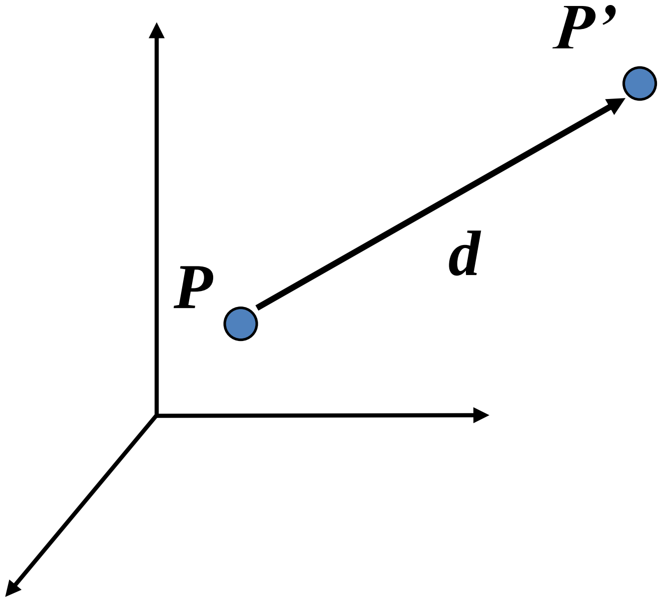
note that this expression is in four dimensions and expresses point = vector + point

Hence $\boldsymbol{p}' = \boldsymbol{p} + \boldsymbol{d}$ or

$x' = x + d_x$

$y' = y + d_y$

$z' = z + d_z$



Translation Matrix

- We can also express translation using a 4 x 4 matrix **T** in homogeneous coordinates

p′=Tp

where

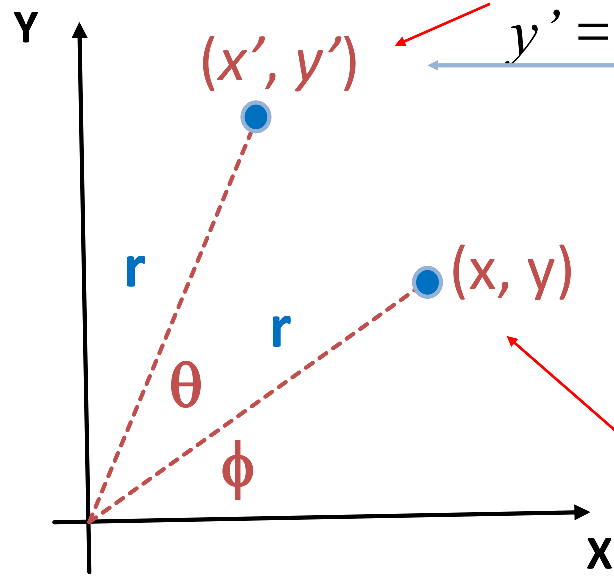
$$\boldsymbol{T} = \boldsymbol{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Why do we use a matrix form instead of vector addition?

Rotation (2D)

- Consider rotation about the origin by *q* degrees
  - radius stays the same, angle increases by *q*

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



$$x = r\cos\phi$$
$$y = r\sin\phi$$

trigonometric identities

$$\sin(\theta+\varphi) = \sin\theta\cos\phi + \cos\theta\sin\phi$$
$$\cos(\theta+\varphi) = \cos\theta\cos\phi - \sin\theta\sin\phi$$

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

Rotation about the z axis

- Rotation about *z* axis in three dimensions
  - leaves all points with the same *z*
  - Equivalent to rotation in two dimensions in planes of constant *z*

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

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

$$z' = z$$

  - or in homogeneous coordinates

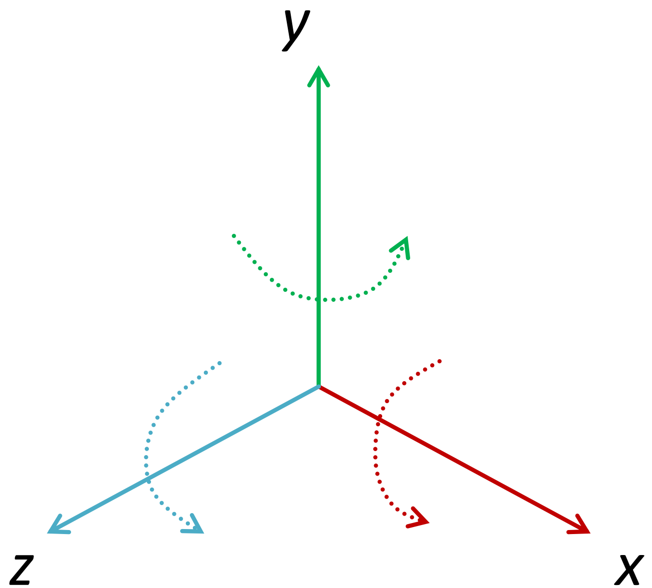
$$\boldsymbol{p}' = \boldsymbol{R}_z(\theta)\boldsymbol{p}$$

Rotation Matrix

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

Rotation about x and y axes

- Same argument as for rotation about *z* axis
  - For rotation about *x* axis, *x* is unchanged
  - For rotation about *y* axis, *y* is unchanged



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

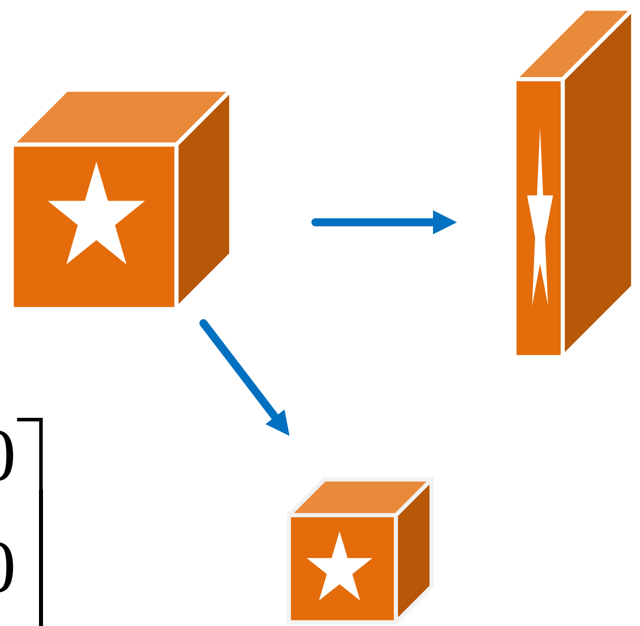
$$\boldsymbol{R} = \boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

- Expand or contract along each axis (fixed point of origin)

$$x' = s_x x$$
$$y' = s_y x$$
$$z' = s_z x$$

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



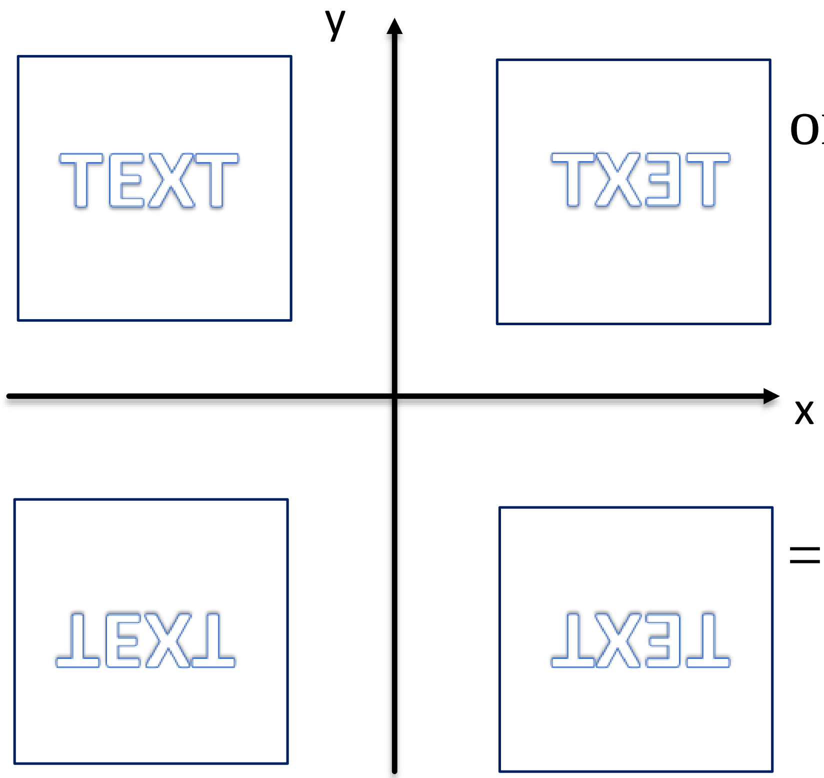
$$\boldsymbol{S} = \boldsymbol{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection

- corresponds to negative scale factors

$s_x = -1 \; s_y = 1$

original



$s_x = -1 \; s_y = -1$

$= 1 \; s_y = -1$

Inverses

- Compute inverse matrices by general formulas, or use simple geometric observations

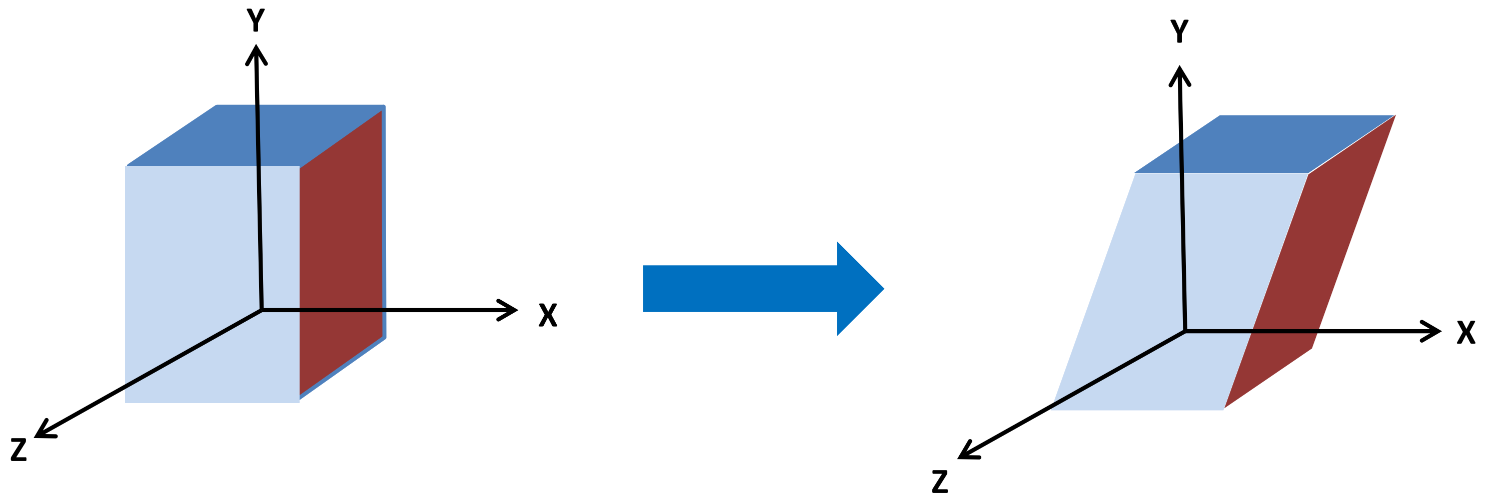
  - Translation: $\boldsymbol{T}^{-1}(d_x, d_y, d_z) = \boldsymbol{T}(-d_x, -d_y, -d_z)$
  - Rotation: $\boldsymbol{R}^{-1}(q) = \boldsymbol{R}(-q)$
    - Holds for any rotation matrix
    - Since cos(-θ) = cos(θ) ; sin(-θ)=-sin(θ)

    $\boldsymbol{R}^{-1}(q) = \boldsymbol{R}^{T}(q)$

  - Scaling: $\boldsymbol{S}^{-1}(s_x, s_y, s_z) = \boldsymbol{S}(1/s_x, 1/s_y, 1/s_z)$

Shear

- Equivalent to pulling faces in opposite directions



Shear Matrix

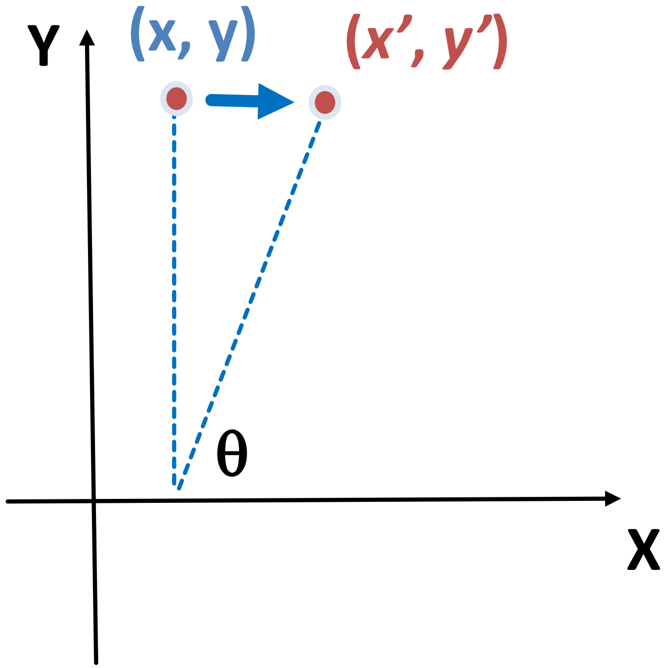
- Consider simple shear along *x* axis

  $x' = x + y \cot \theta$

  $y' = y$

  $z' = z$

$$\mathbf{H}(\theta) = \begin{bmatrix} 1 & \cot\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Concatenation

- Form arbitrary affine transformation matrices by multiplying together rotation, translation, and scaling matrices.

$$\boxed{\begin{array}{l} \textit{for each } i \\ \quad ABCDp_i\,, \end{array}}$$

or

$$\boxed{\begin{array}{l} M = ABCD, \\ \textit{for each } i \\ \quad Mp_i \end{array}}$$

Order of Transformations

- Note that matrix on the right is the first applied
- Mathematically, the following are equivalent

$$\boldsymbol{p}' = \boldsymbol{ABCp} = \boldsymbol{A}(\boldsymbol{B}(\boldsymbol{Cp}))$$

- Note many references use column matrices to represent points. In terms of column matrices

$$\boldsymbol{p}'^{T} = \boldsymbol{p}^{T}\boldsymbol{C}^{T}\boldsymbol{B}^{T}\boldsymbol{A}^{T}$$

Matrix multiplication is associative !

General Rotation about the Origin

- Decompose into the concatenation of rotations about the *x*, *y*, and *z* axes

$$\boldsymbol{R}(\theta) = \boldsymbol{R}_z(\theta_z)\, \boldsymbol{R}_y(\theta_y)\, \boldsymbol{R}_x(\theta_x)$$

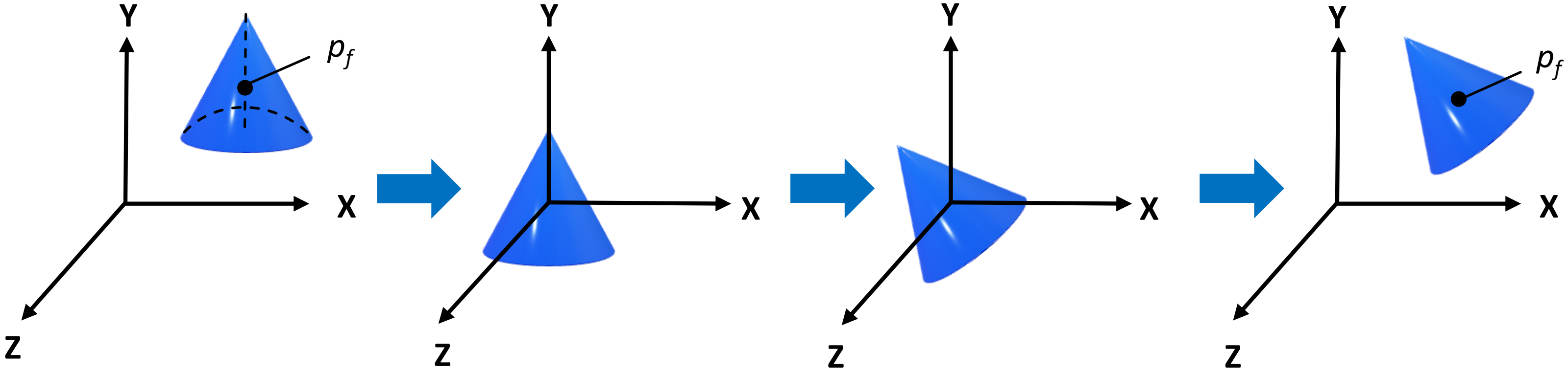
$\theta_x, \theta_y, \theta_z$ are rotation angles with respect to the corresponding axes.

- Commutative?

Rotation about a Fixed Point other than the Origin

1. Move fixed point to origin
2. Rotate
3. Move fixed point back

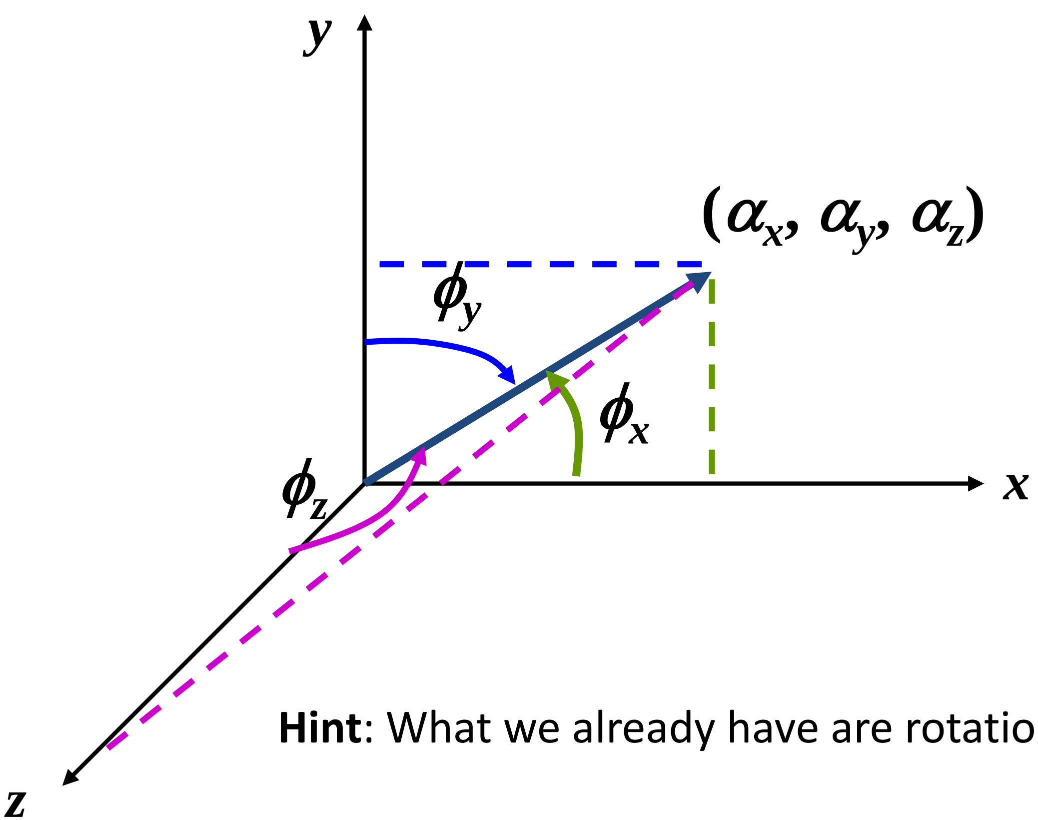
$$\boldsymbol{M} = \boldsymbol{T}(p_f)\, \boldsymbol{R}(q)\, \boldsymbol{T}(-p_f)$$



Rotation about an Arbitrary Axis

- Rotate around an axis vector *u.*

$v = u/|u| = [\alpha_x, \alpha_y, \alpha_z]^T$



$$\cos\phi_x = \alpha_x$$

$$\cos\phi_y = \alpha_y$$

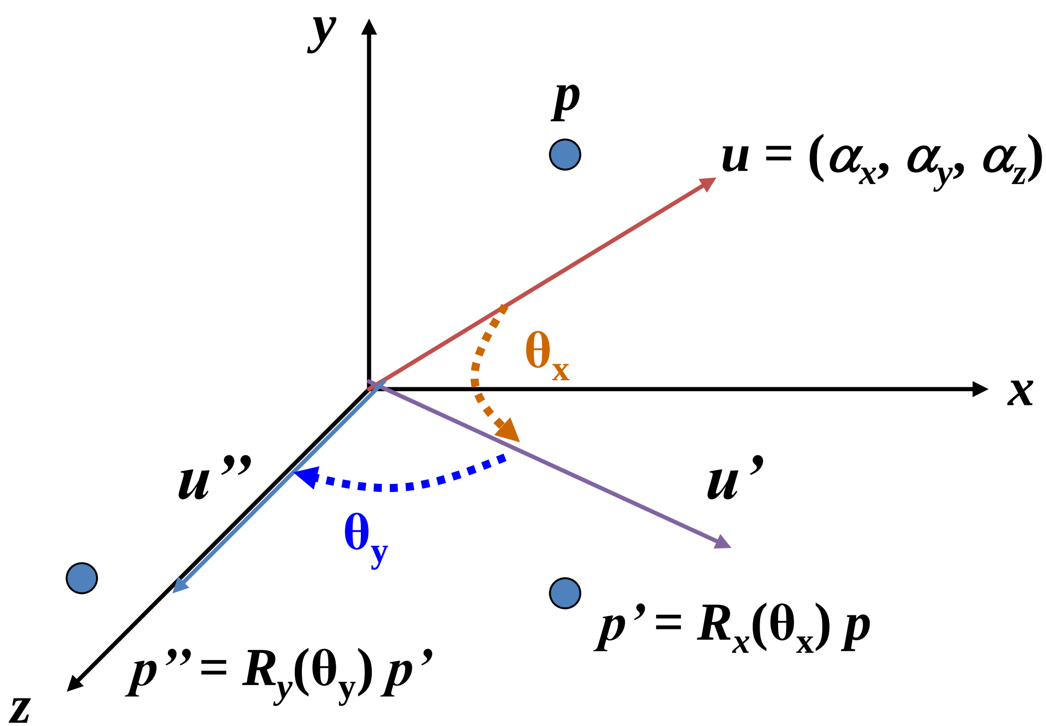
$$\cos\phi_z = \alpha_z$$

$$\cos\phi_x + \cos\phi_y + \cos\phi_z = 1$$

Hint: What we already have are rotations around x, or y, or z axes.

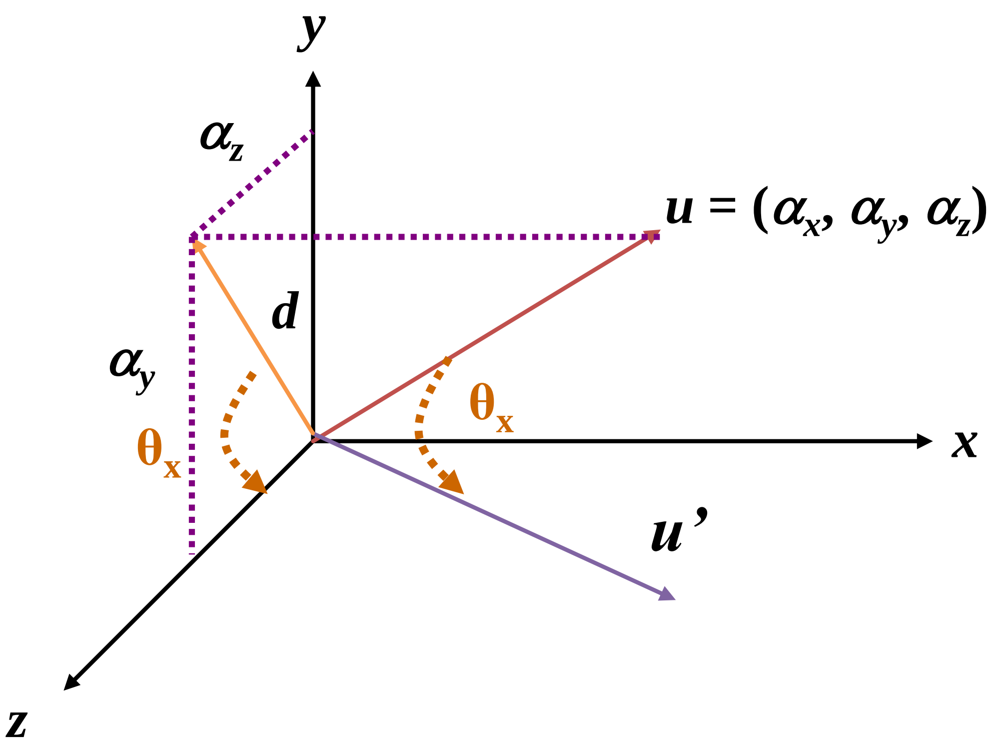
Rotation about an Arbitrary Axis

1. Rotate the axis vector to match *z* (*x* or *y*) axis. [R_{axis}]
2. Rotate around *z* axis. [$R_z(\theta)$]
3. Rotate the axis vector back. [R_{axis}^{-1}]



$$R_{axis} = R_y(\theta_y)R_x(\theta_x)$$

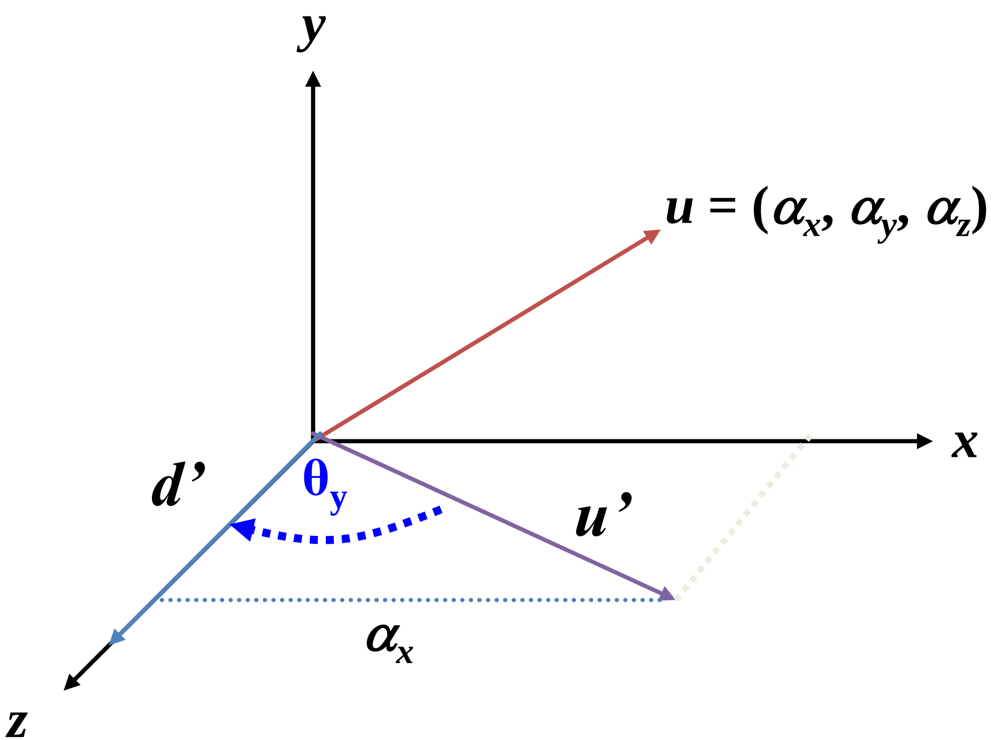
$R_x(\theta_x)$



$$a_y^{\ 2} + a_z^{\ 2} = d^2$$

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \alpha_z/d & -\alpha_y/d & 0 \\ 0 & \alpha_y/d & \alpha_z/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$R_y(\theta_y)$



$$\|u'\| = \|u\| = 1$$

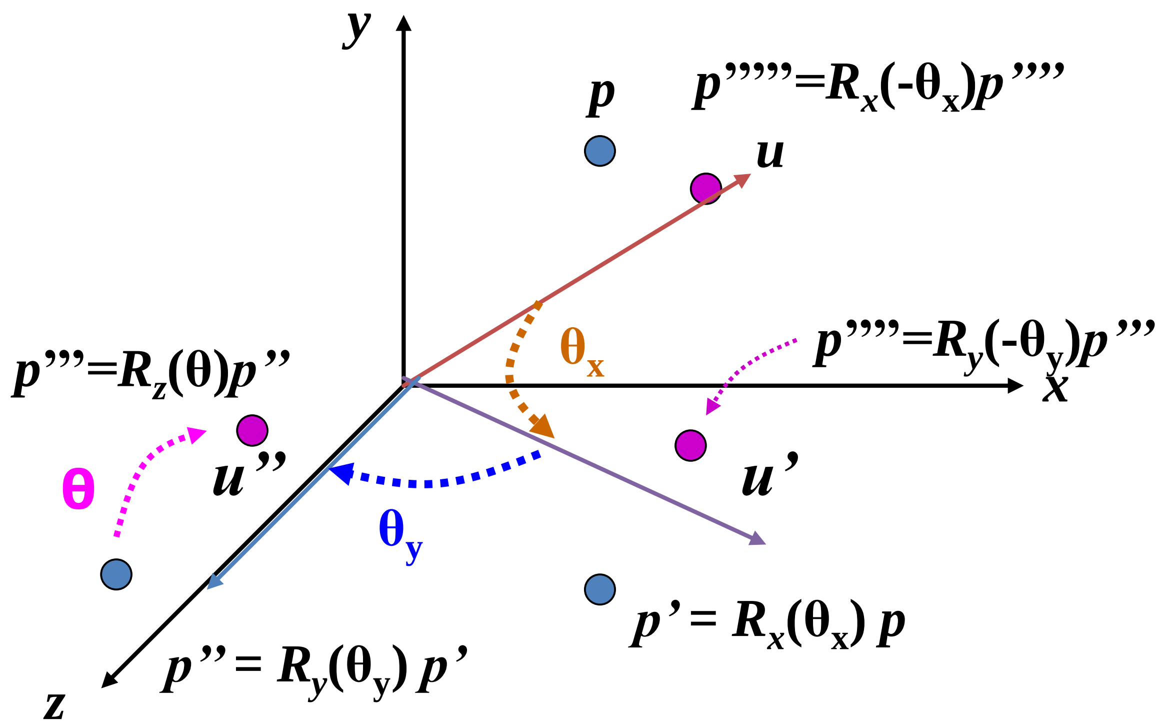
$$d'^2 + a_x{}^2 = u'^2$$

$$R_y(\theta_y) = \begin{bmatrix} d' & 0 & -\alpha_x & 0 \\ 0 & 1 & 0 & 0 \\ \alpha_x & 0 & d' & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

$$M = R_{axis}^{-1}\, R_z(\theta)\, R_{axis}$$

$$= R_x(-\theta_x)\, R_y(-\theta_y)\, R_z(\theta)\, R_y(\theta_y)\, R_x(\theta_x)$$



Instancing

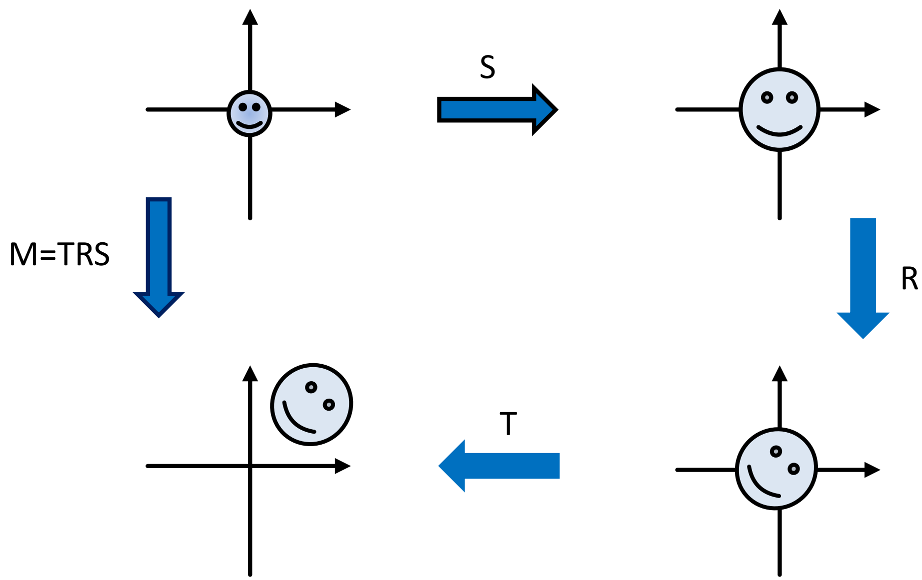
- In modeling, we often start with a simple object centered at the origin, oriented with the axis, and at a standard size

- We apply an *instance transformation* to its vertices to

  Scale

  Orient

  Locate

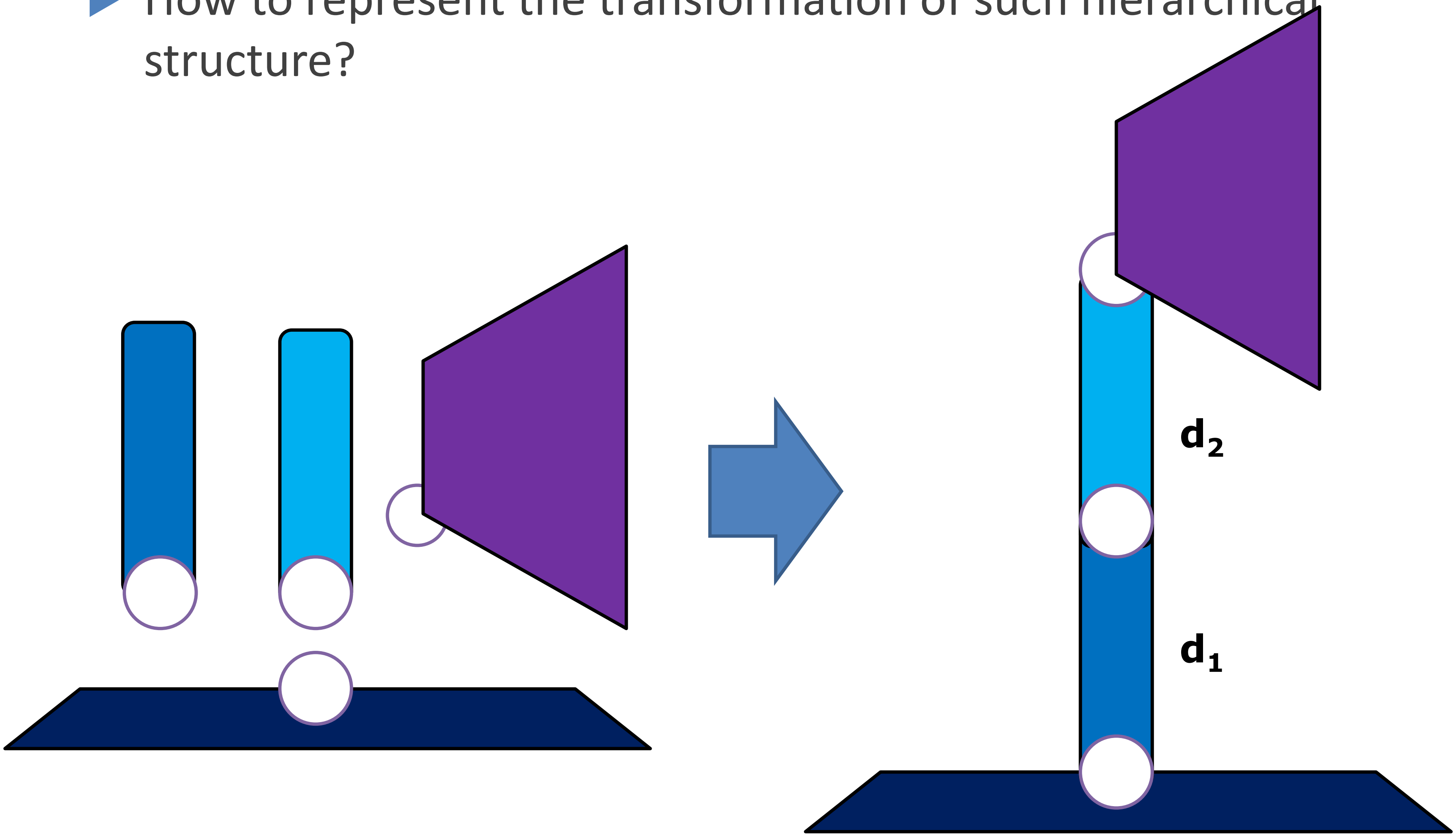


Hierarchical structure

- In addition to separate instances, plenty of objects consist of hierarchical sub-components , e.g. skeletons, desk lamps, excavators, etc.

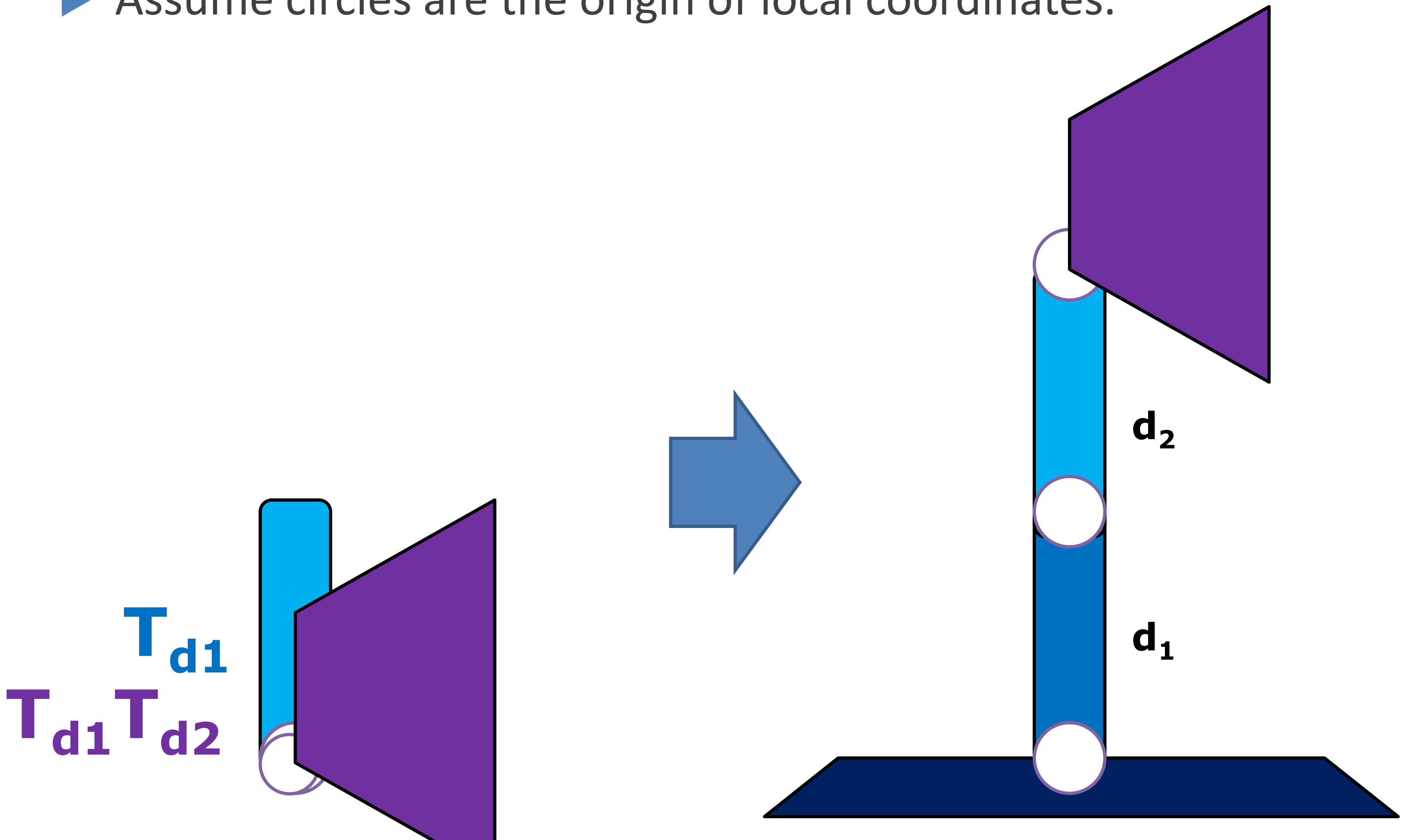
Hierarchical structure (cont.)

- How to represent the transformation of such hierarchical structure?



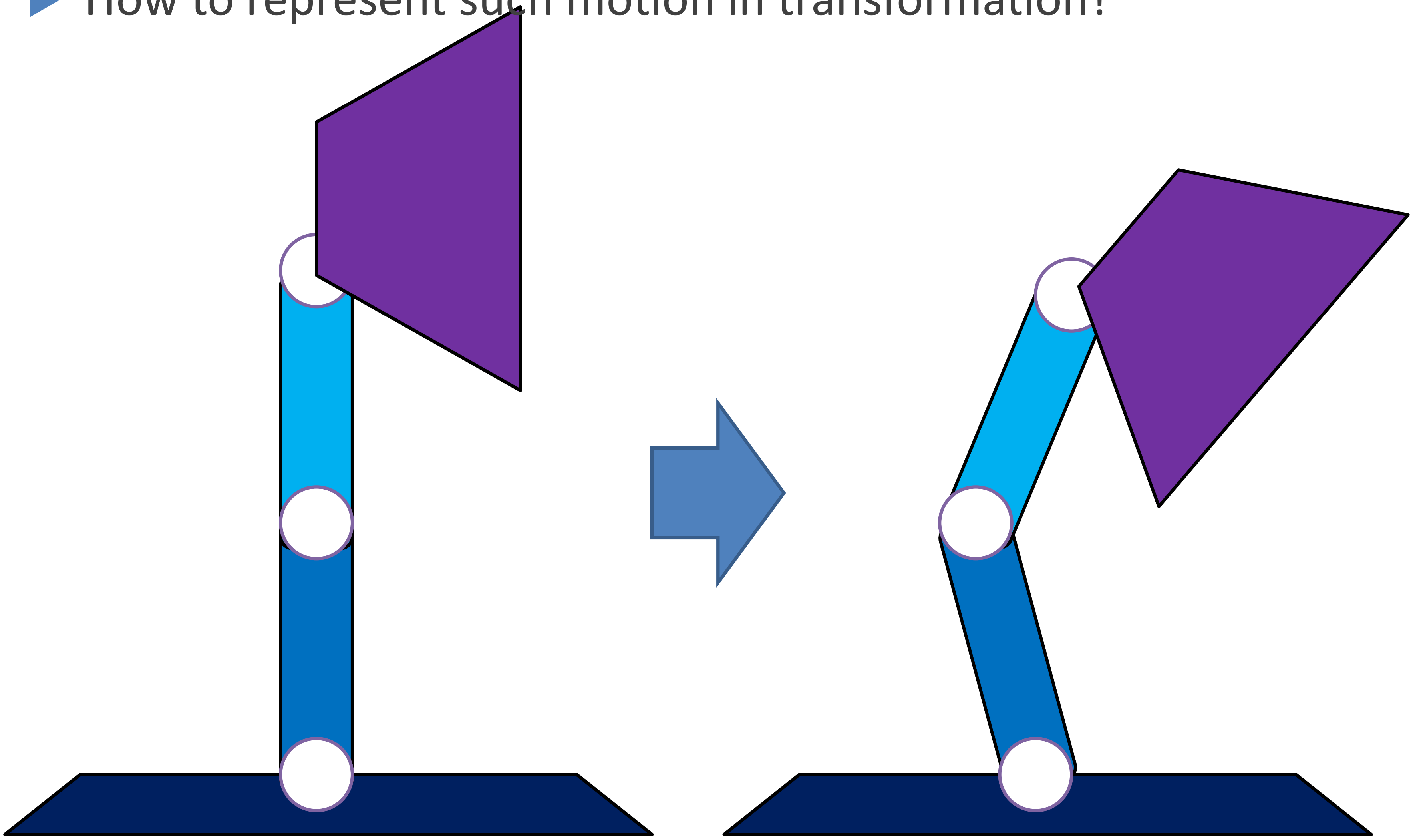
Hierarchical structure (cont.)

- Assume circles are the origin of local coordinates.



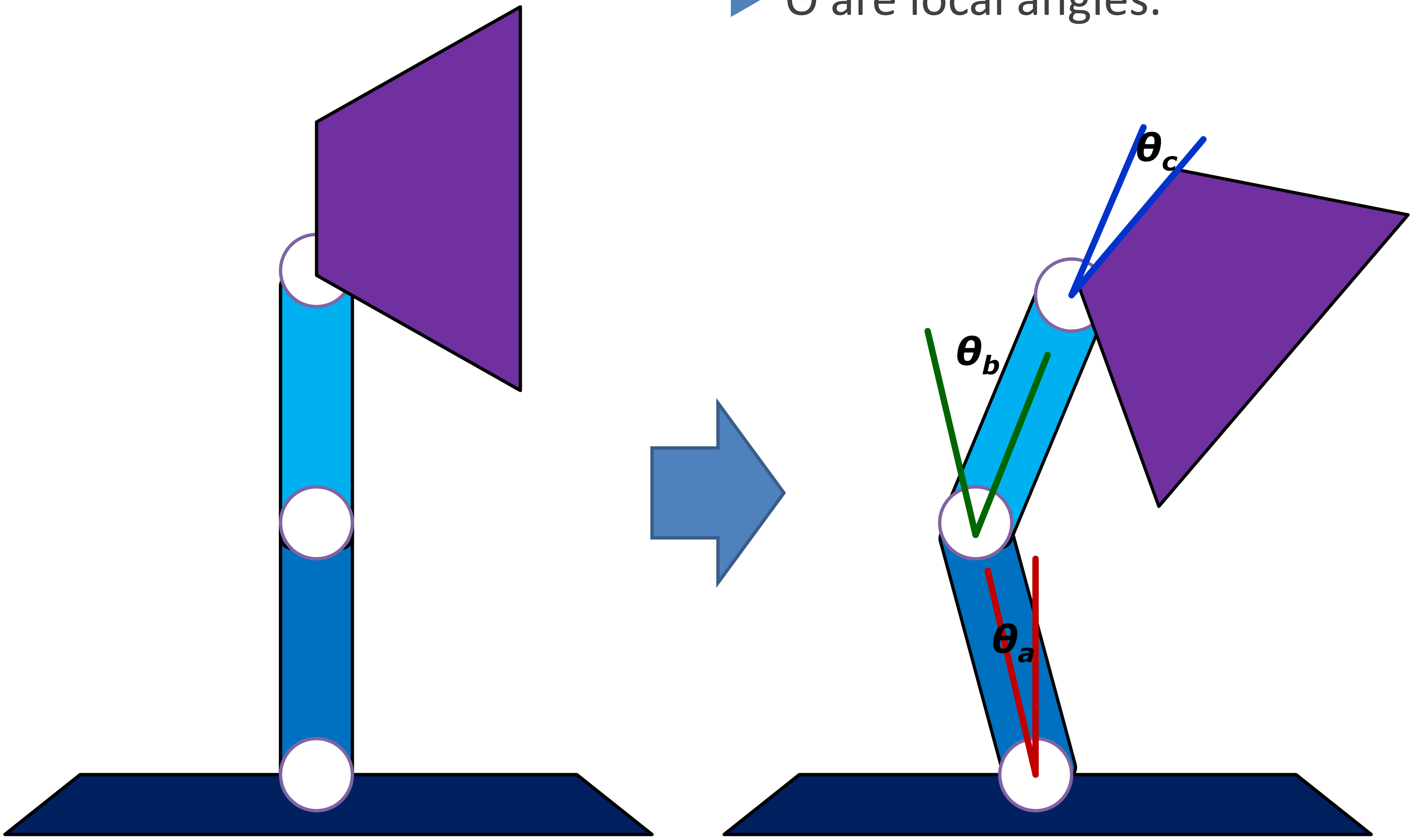
Hierarchical structure (cont.)

- How to represent such motion in transformation?



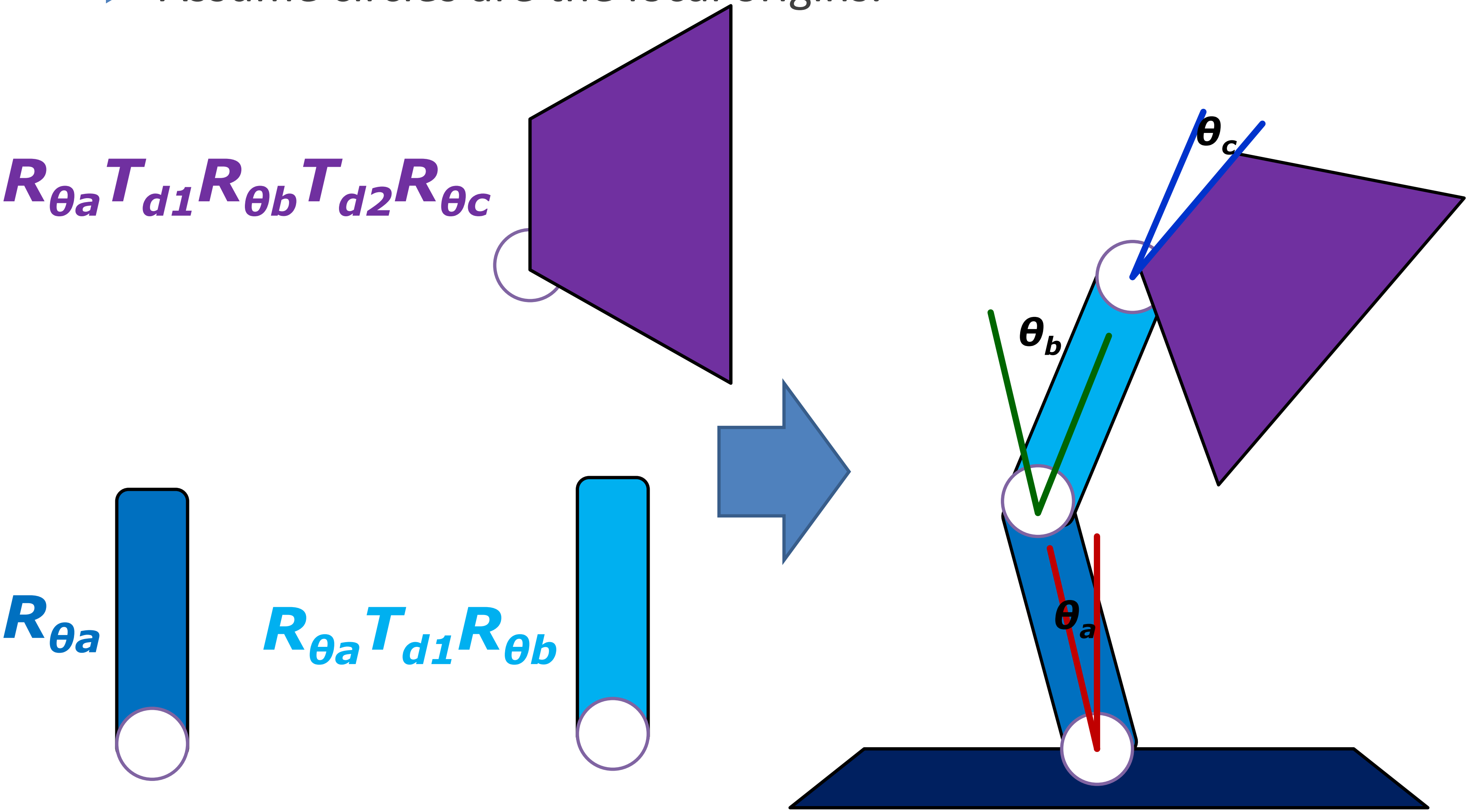
Hierarchical transformation

- Θ are local angles.



Hierarchical transformation (cont.)

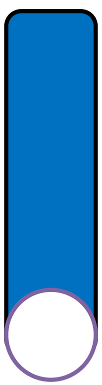
- Assume circles are the local origins.



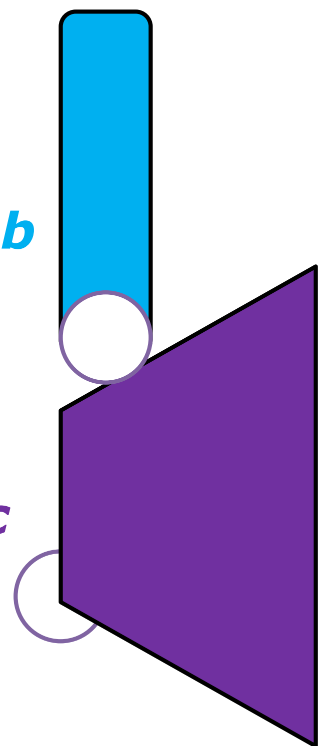
Hierarchical transformation (cont.)

- There are common sub-transformation.
- We can avoid redundant matrix multiplication by stack mechanism.
  - Hierarchical coordinates.

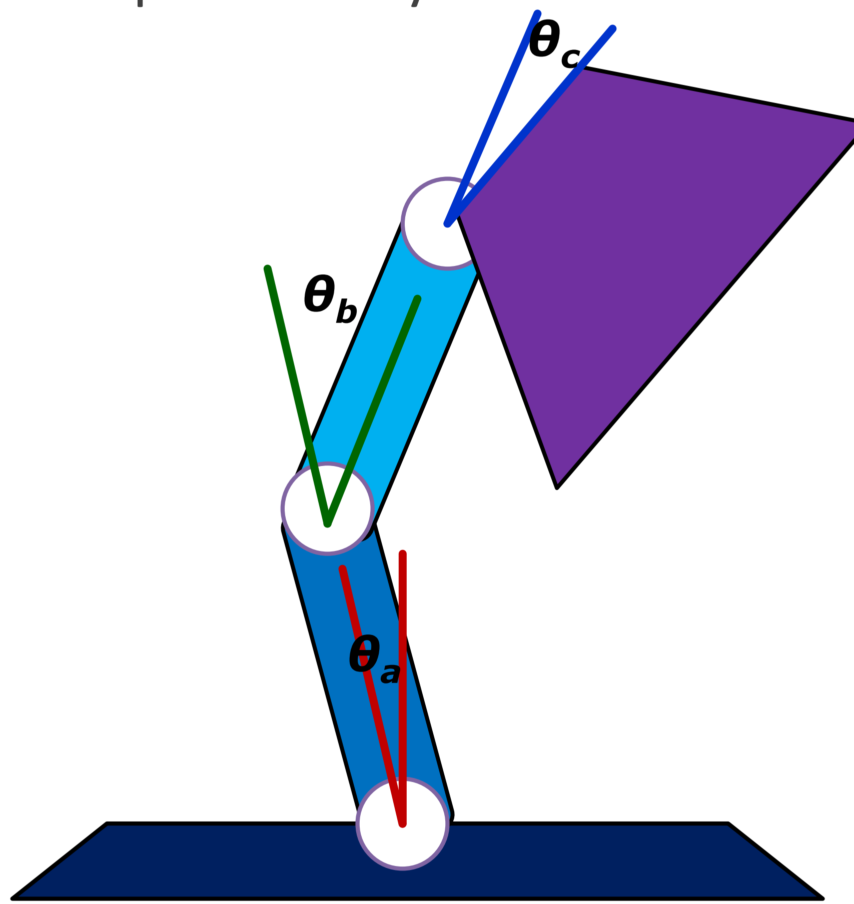
$R_{\theta a}$



$\underline{R_{\theta a}}T_{d1}R_{\theta b}$



$\underline{\underline{R_{\theta a}}T_{d1}R_{\theta b}}T_{d2}R_{\theta c}$



Matrix in OpenGL style (Legacy)

………….

“**Draw the base**”

glRotate($\boldsymbol{\theta}_a$);

glPushMatrix();

“**Draw the dark blue arm**”

glPopMatrix();

glTranslate($\mathbf{d}_1$);

glRotate($\boldsymbol{\theta}_b$);

glPushMatrix();

“**Draw the light blue arm**”

glPopMatrix();

glTranslate($\mathbf{d}_2$);

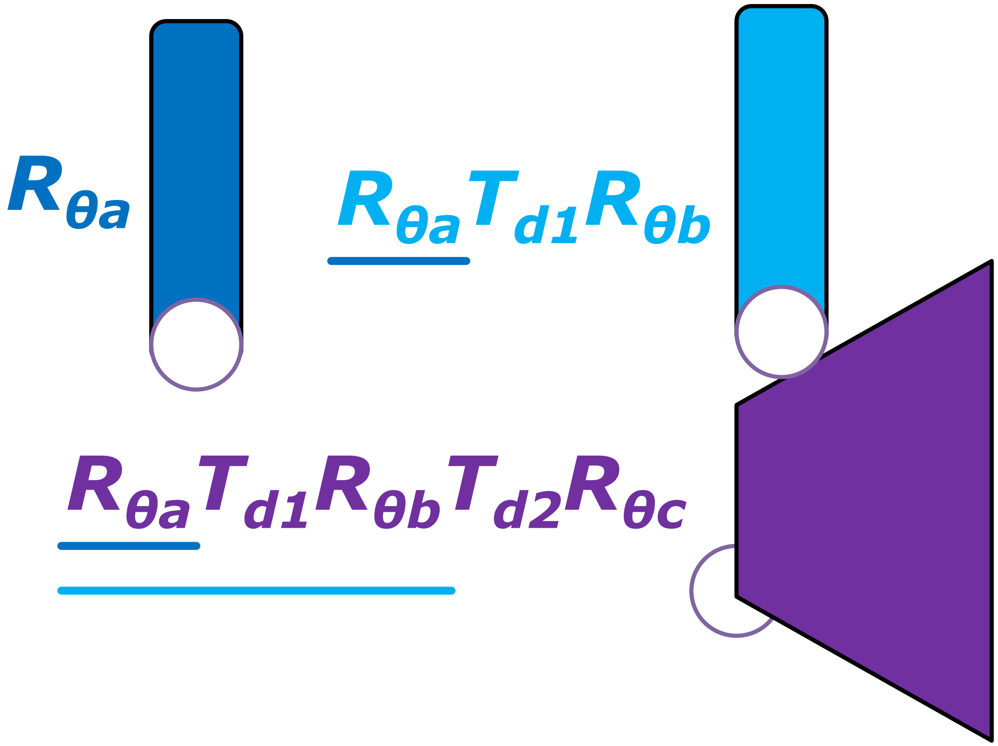
glRotate($\boldsymbol{\theta}_c$);

glPushMatrix();

“**Draw the lampshade**”

glPopMatrix();

………….



How to deal with branches?

*With **glm** lib.*   glm::translate(X, vec3) -> X * glm::translate(Identity, vec3)

Matrix in OpenGL style (Modern)

………….

"**Draw the base**"

MatA = glm::rotate(Mat, $\boldsymbol{\theta}_a$);

"Pass the MatA"

"**Draw the dark blue arm**"

Mat1 = glm::translate(MatA, $\mathbf{d}_1$);

MatB = glm::rotate(Mat1, $\boldsymbol{\theta}_b$);

"Pass the MatB"

"**Draw the light blue arm**"

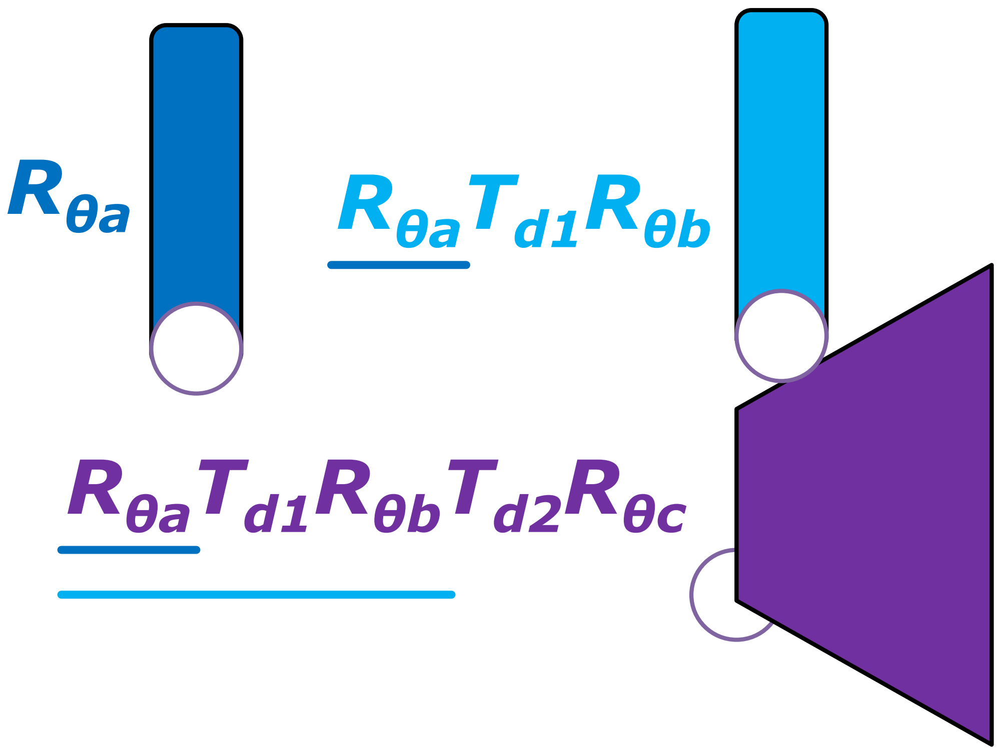
Mat2 = glm::translate(MatB, $\mathbf{d}_2$);

MatC = glm::rotate(Mat2, $\boldsymbol{\theta}_c$);

"Pass the MatC"

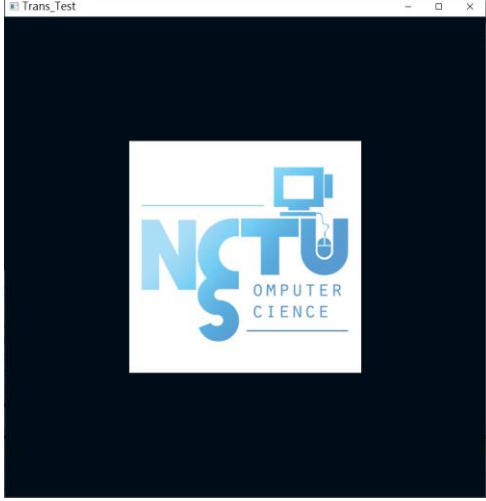
"**Draw the lampshade**"

………….



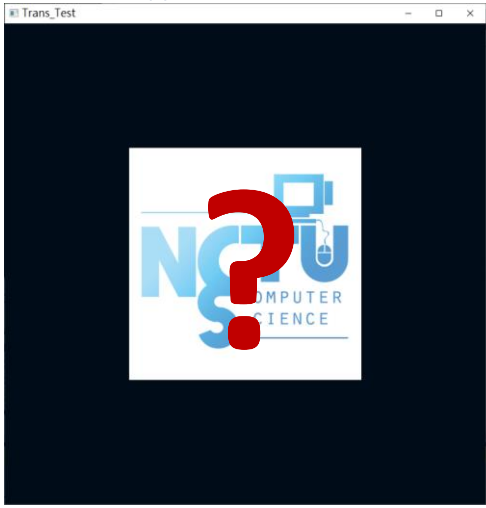
How to deal with branches?

A Modern-OpenGL Example



- Given a box located at the origin,
  What's the result with the following trans?

```
glm::mat4 model = glm::mat4(1.0f);

model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));

model = glm::translate(model, glm::vec3(0.0f, 0.8f, 0.0f));

model = glm::rotate(model, glm::radians(30.0f), glm::vec3(0.0f, 1.0f, 0.0f));

model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));

//For all i, BoxPts(i) = model * BoxPts(i)
```



*With **glm** lib.* glm::translate(X, vec3) -> X * glm::translate(Identity, vec3)

A Modern-OpenGL Example (cont.)

glm::mat4 **model** = glm::mat4(1.0f);

I

model = glm::**rotate**(**model**, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));

$\boldsymbol{m*R_z(45^o)}$

model = glm::**translate**(**model**, glm::vec3(0.0f, 0.8f, 0.0f));

$\boldsymbol{m*T(0, 0.8, 0)}$

model = glm::**rotate**(**model**, glm::radians(30.0f), glm::vec3(0.0f, 1.0f, 0.0f));

$\boldsymbol{m*R_y(30^o)}$

model = glm::**scale**(**model**, glm::vec3(0.5f, 0.5f, 0.5f));

$\boldsymbol{m*S(0.5, 0.5, 0.5)}$

//For all i, BoxPts(i) = model * BoxPts(i)

$\boldsymbol{m*BoxPt(i)}$

$R_z(45^o)*T(0, 0.8, 0)*R_y(30^o)*S(0.5, 0.5, 0.5)*BoxPts(i)$

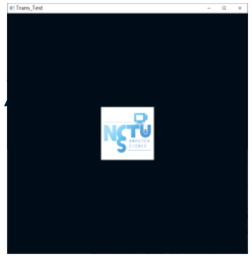
Direction of instruction execution

*With **glm** lib.* glm::translate(X, vec3) -> X * glm::translate(Identity, vec3)

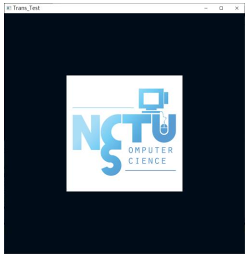
A Modern-OpenGL Example (cont.)

$R_z(45^o)*T(0, 0.8, 0)*R_y(30^o)*S(0.5, 0.5, 0.5)*BoxPts(i)$

```
glm::mat4 model = glm::mat4(1.0f);

model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));

model = glm::translate(model, glm::vec3(0.0f, 0.8f, 0.0f));

model = glm::rotate(model, glm::radians(30.0f), glm::vec3(0.0f, 1.0f, 0.0f));

model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));

//For all i, BoxPts(i) = model * BoxPts(i)
```



S(0.5, 0.5, 0.5) *BoxPt(i)



You can infer the final pose of the box in this way

*With **glm** lib.* glm::translate(X, vec3) -> X * glm::translate(Identity, vec3)

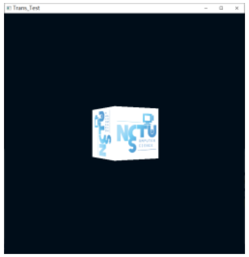
A Modern-OpenGL Example (cont.)

$R_z(45^o)*T(0, 0.8, 0)*R_y(30^o)*S(0.5, 0.5, 0.5)*BoxPts(i)$

glm::mat4 **model** = glm::mat4(1.0f);

model = glm::**rotate**(**model**, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));

model = glm::**translate**(**model**, glm::vec3(0.0f, 0.8f, 0.0f));



model = glm::**rotate**(**model**, glm::radians(30.0f), glm::vec3(0.0f, 1.0f, 0.0f));

$R_y(30^o)*Pt'(i)$

model = glm::**scale**(**model**, glm::vec3(0.5f, 0.5f, 0.5f));

//For all i, BoxPts(i) = model * BoxPts(i)

You can infer the final pose of the box in this way

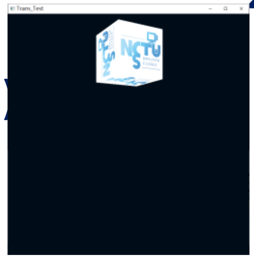
*With **glm** lib.* glm::translate(X, vec3) -> X * glm::translate(Identity, vec3)

A Modern-OpenGL Example (cont.)

$R_z(45^o)*T(0, 0.8, 0)*R_y(30^o)*S(0.5, 0.5, 0.5)*BoxPts(i)$

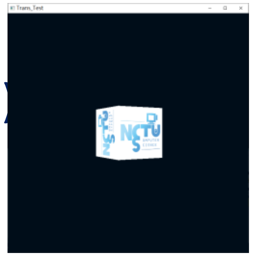
glm::mat4 **model** = glm::mat4(1.0f);

model = glm::**rotate**(**model**, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f))



model = glm::**translate**(**model**, glm::vec3(0.0f, 0.8f, 0.0f));

T(0, 0.8, 0) *Pt''(i)



model = glm::**rotate**(**model**, glm::radians(30.0f), glm::vec3(0.0f, 1.0f, 0.0f))

model = glm::**scale**(**model**, glm::vec3(0.5f, 0.5f, 0.5f));

//For all i, BoxPts(i) = model * BoxPts(i)

You can infer the final pose of the box in this way

*With **glm** lib.* glm::translate(X, vec3) -> X * glm::translate(Identity, vec3)

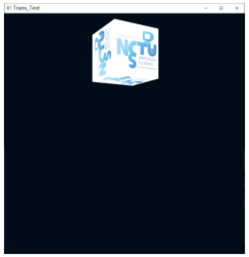
A Modern-OpenGL Example (cont.)

$R_z(45^o)*T(0,\ 0.8,\ 0)*R_y(30^o)*S(0.5,\ 0.5,\ 0.5)*$

glm::mat4 **model** = glm::mat4(1.0f);

model = glm::**rotate**(**model**, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));

$R_z(45^o)$ *Pt'''(i)



model = glm::**translate**(**model**, glm::vec3(0.0f, 0.8f, 0.0f));

model = glm::**rotate**(**model**, glm::radians(30.0f), glm::vec3(0.0f, 1.0f, 0.0f)

model = glm::**scale**(**model**, glm::vec3(0.5f, 0.5f, 0.5f));

//For all i, BoxPts(i) = model * BoxPts(i)

You can infer the final pose of the box in this way

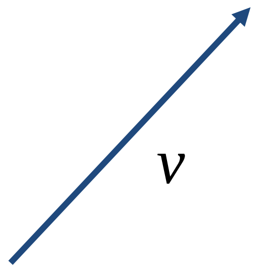
Appendix

Basic Elements

- Geometry:
  - the relationships among objects in an *n-dimensional space*
  - Computer graphics mainly focuses on *three dimensions*.
- Want a minimum set of primitives from which we can build more sophisticated objects
- We will need three basic elements
  - Scalars
  - Vectors
  - Points

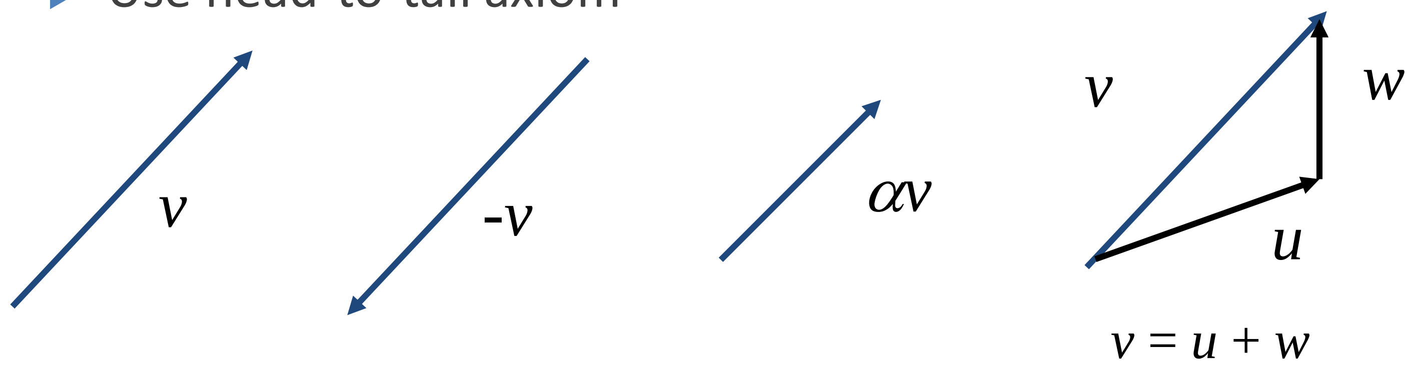
Vectors

- Physical definition: a vector is a quantity with two attributes
  - Direction
  - Magnitude
- Examples include
  - Force
  - Velocity
  - Directed line segments
    - Most important example for graphics
    - Can map to other types



Vector Operations

- Every vector has an inverse
  - Same magnitude but points in opposite direction
- Every vector can be multiplied by a scalar
- There is a zero vector
  - Zero magnitude, undefined orientation
- The sum of any two vectors is a vector
  - Use head-to-tail axiom



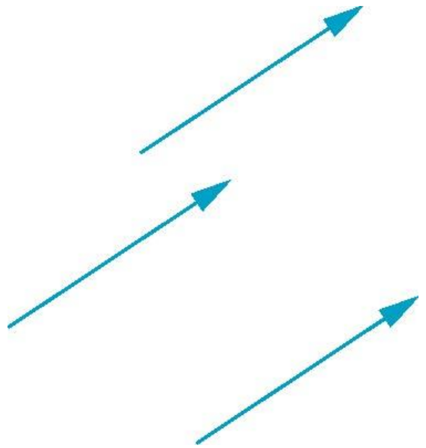
Linear Vector Spaces

- Mathematical system for manipulating vectors
- Operations
  - Scalar-vector multiplication $u=\alpha v$
  - Vector-vector addition: $w=u+v$
- Expressions such as

  $v=u+2w-3r$
- Make sense in a vector space

Vectors Lack Position

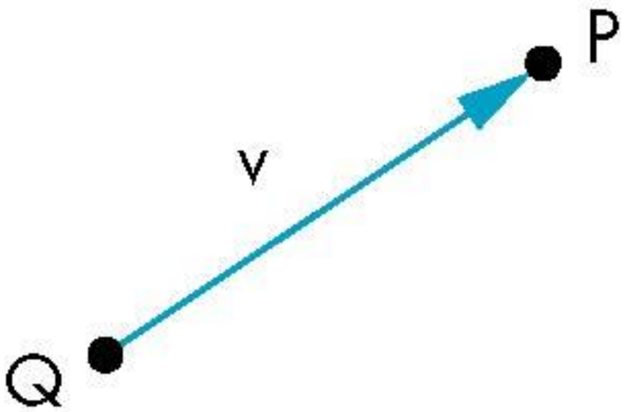
- These vectors are identical
  - Same length and magnitude



- Vectors spaces insufficient for geometry
  - Need points

Points

- Location in space
- Operations allowed between points and vectors
  - Point-point subtraction yields a vector
  - Equivalent to point-vector addition



$$v = P - Q$$

$$P = v + Q$$

Affine Spaces

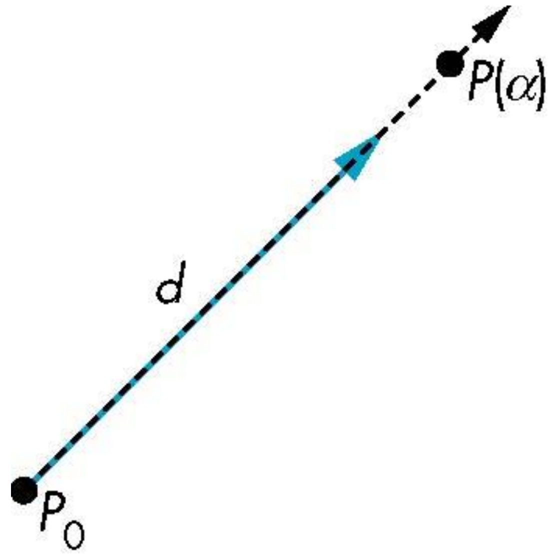
- Point + a vector space

- Operations
  - Vector-vector addition
  - Scalar-vector multiplication
  - Point-vector addition
  - Scalar-scalar operations

- For any point define
  - $1 \bullet P = P$
  - $0 \bullet P = \mathbf{0}$ (zero vector)

Lines

- Consider all points of the form
  - $P(\alpha)=P_0 + \alpha\ \mathbf{d}$
  - Set of all points that pass through P_0 in the direction of the vector $\mathbf{d}$



Parametric Form

- This form is known as the parametric form of the line
  - More robust and general than other forms
  - Extends to curves and surfaces

- Two-dimensional forms
  - **Explicit**: $y = mx + h$
  - **Implicit**: $ax + by + c = 0$
  - **Parametric**:

$$x(\alpha) = \alpha x_0 + (1-\alpha)x_1$$

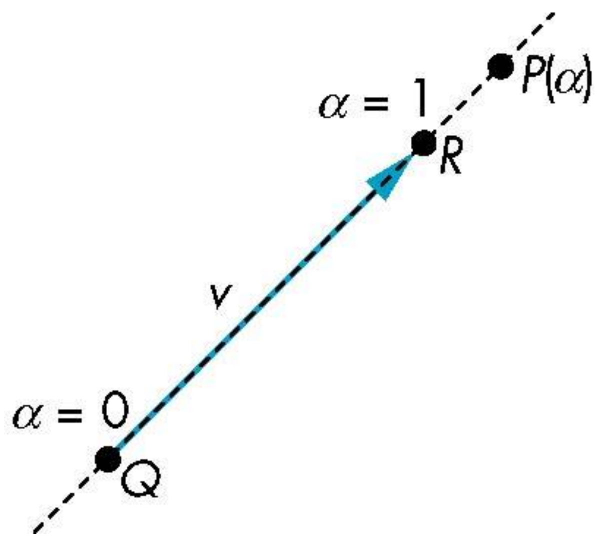
$$y(\alpha) = \alpha y_0 + (1-\alpha)y_1$$

Rays and Line Segments

- α >= 0, *ray* leaving P_0 in the direction $\mathbf{d}$
- If we use two points to define v, then

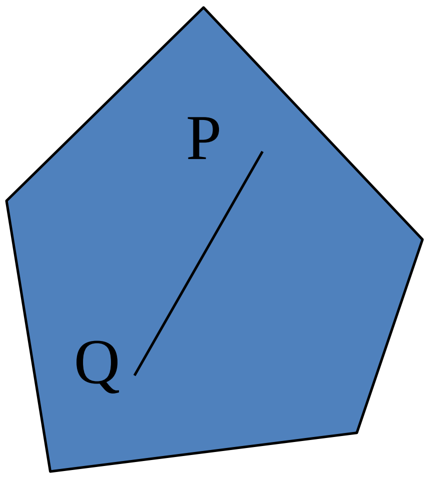
  $$P(\alpha) = Q + \alpha (R-Q) = Q + \alpha v = \alpha R + (1-\alpha)Q$$

- $0 <= \alpha <= 1$, *line segment* joining R and Q

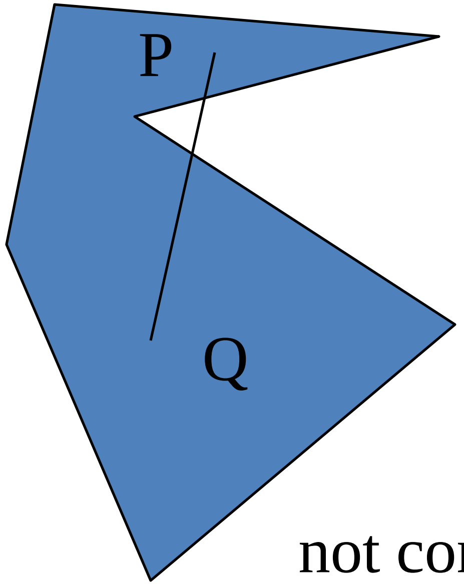


Convexity

- *Convex* iff:
  - for any two points in the object all points on the line segment between these points are also in the object



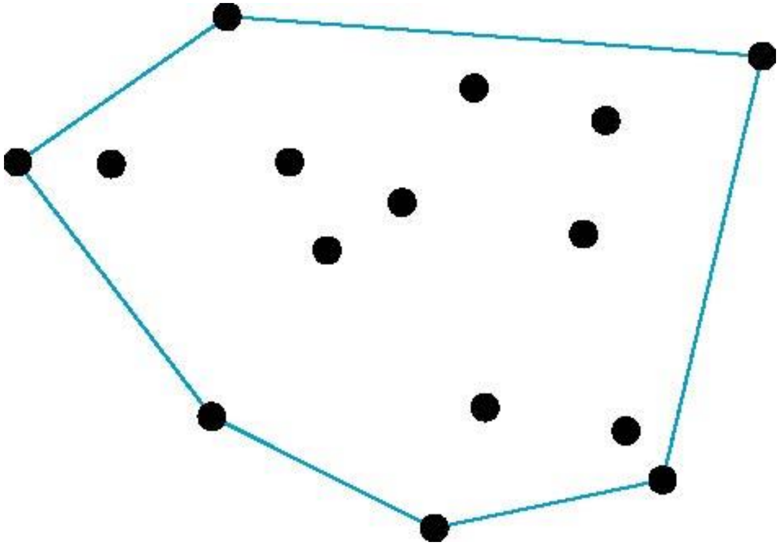
convex



not convex

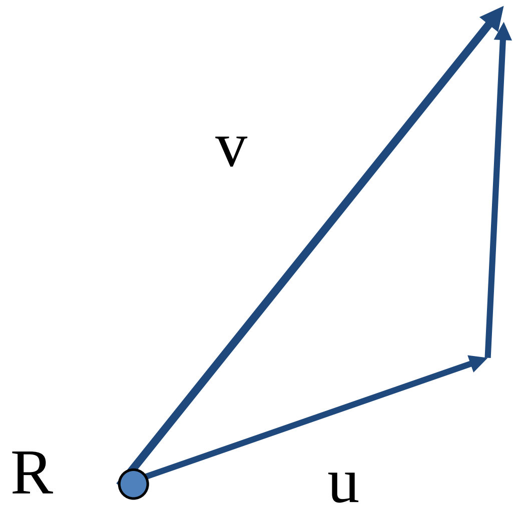
Convex Hull

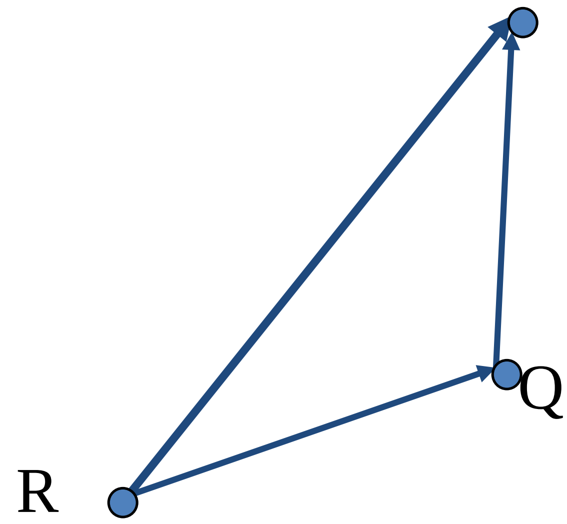
- Smallest convex object containing $P_1, P_2, \ldots\ldots P_n$
- Formed by "shrink wrapping" points



Planes

- A plane can be defined by a point and two vectors or by three points





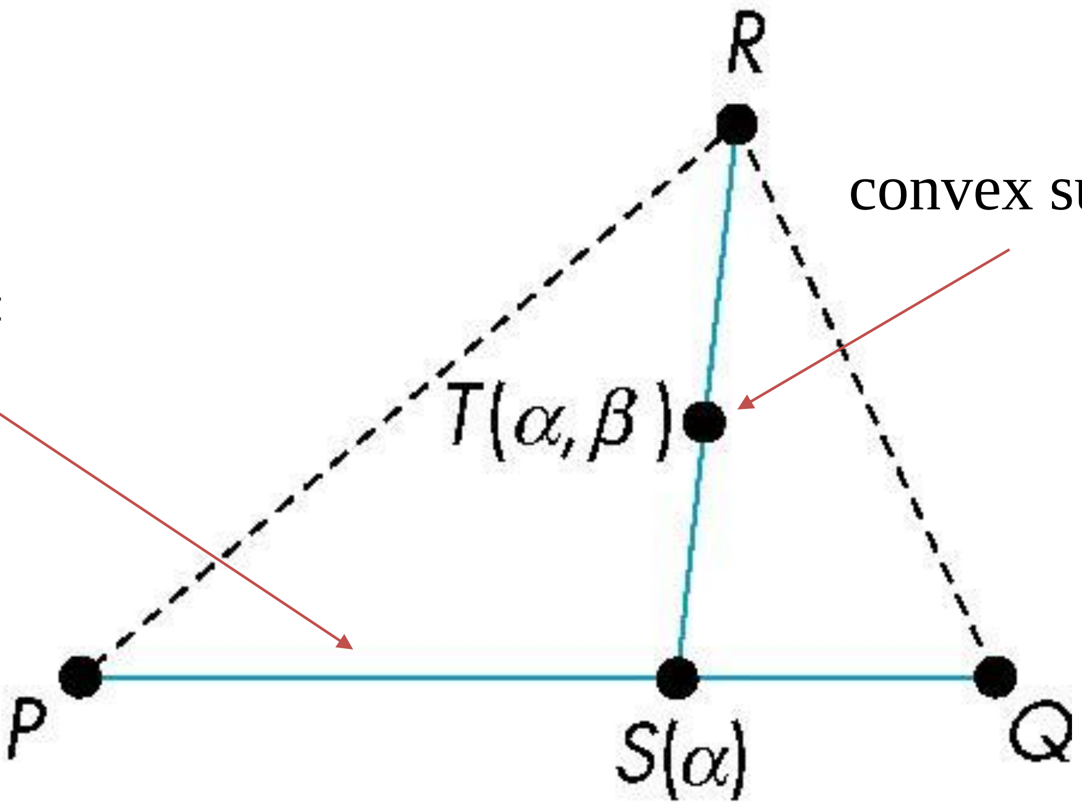
$P(\alpha,\beta)=R+\alpha u+\beta v$

$P(\alpha,\beta)=R+\alpha(Q-R)+\beta(P-Q)$

Triangles

convex sum of S(α) and R

convex sum c



for 0<=α,β<=1, we get all points in triangle

Barycentric Coordinates

- Triangle is convex so any point inside can be represented as an affine sum

$P(a_1, a_2, a_3) = a_1 P + a_2 Q + a_3 R$

where $a_1 + a_2 + a_3 = 1$, and $a_i >= 0$

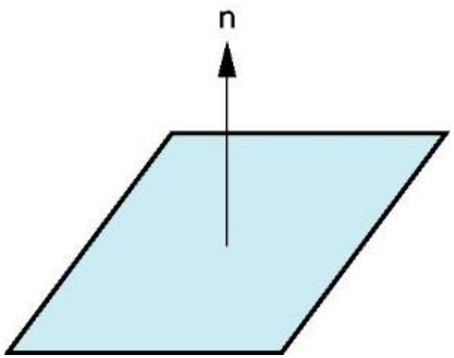
- The representation is called the barycentric coordinate representation of P

Normals

- Every plane has a vector n normal (perpendicular, orthogonal) to it

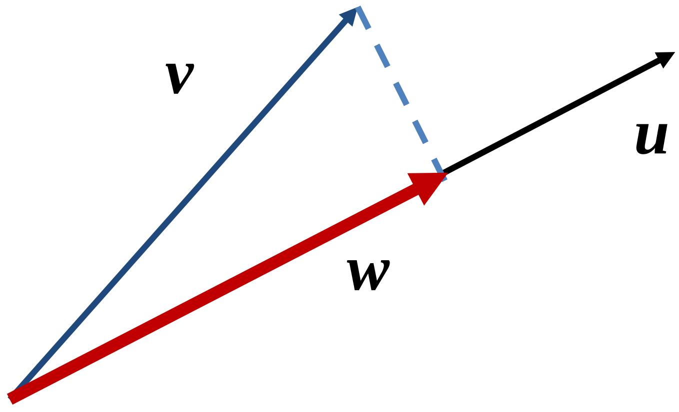
- From point-two vector form $P(\alpha,\beta)=R+\alpha u+\beta v$, we know we can use the cross product to find $n = u \times v$ and the equivalent form

$$(P(\alpha)\text{-}P) \cdot n=0$$



Dot product

- $u = [x_1, x_2, x_3]^T$
- $v = [y_1, y_2, y_3]^T$
- $u \cdot v = x_1y_1 + x_2y_2 + x_3y_3 = |u||v|\cos\theta$
- Projection

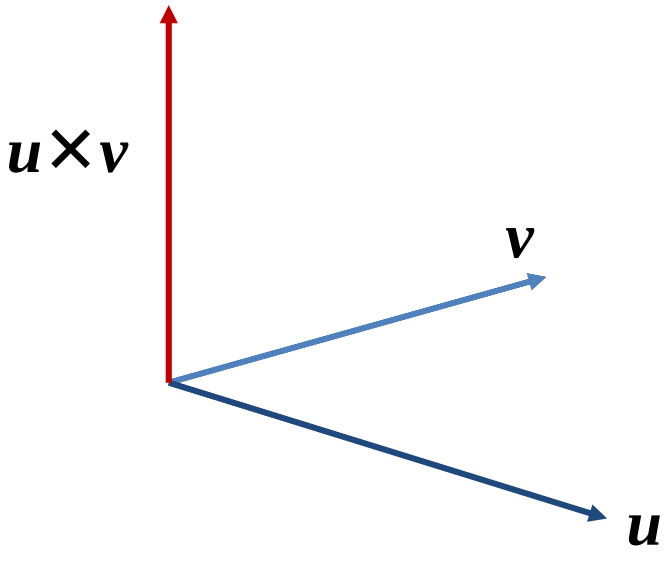


$$w = (|v|\cos\theta)\, unit(u)$$
$$= \left(|v|\frac{u \bullet v}{|u||v|}\right)\frac{u}{|u|}$$
$$= \left(\frac{u \bullet v}{|u|^2}\right)u$$

Cross Product

- $u = [x_1, x_2, x_3]^T$
- $v = [y_1, y_2, y_3]^T$
- $| u \times v | = |u||v||\sin\theta|$

$$w = u \times v = \begin{bmatrix} x_2 y_3 - x_3 y_2 \\ x_3 y_1 - x_1 y_3 \\ x_1 y_2 - x_2 y_1 \end{bmatrix}$$



Linear Independence

- A set of vectors $v_1, v_2, \ldots, v_n$ is *linearly independent* if
  $\alpha_1 v_1 + \alpha_2 v_2 + .. \alpha_n v_n = 0$ iff $\alpha_1 = \alpha_2 = \ldots = 0$

- If a set of vectors is linearly independent, we cannot represent one in terms of the others

- If a set of vectors is linearly dependent, as least one can be written in terms of the others

Dimension

- Dimension of a space
  - In a vector space, the maximum number of linearly independent vectors is fixed
- Basis
  - In an *n*-dimensional space, any set of n linearly independent vectors form a *basis* for the space
- Given a basis $v_1, v_2, \ldots, v_n$, any vector *v* can be written as

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$$

where the $\{\alpha_i\}$ are unique

Representation

- Need a frame of reference to relate points and objects to our physical world.
  - For example, where is a point? Can't answer without a reference system

  - World coordinates
  - Camera coordinates

Coordinate Systems

- Consider a basis $v_1, v_2, \ldots, v_n$
- A vector is written $v = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$
- The list of scalars $\{\alpha_1, \alpha_2, \ldots \alpha_n\}$ is the *representation* of v with respect to the given basis
- We can write the representation as a row or column array of scalars

$$\mathbf{a} = [\alpha_1 \ \ \alpha_2 \ \ \ldots \ \ \alpha_n]^T = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ . \\ \alpha_n \end{bmatrix}$$

Example

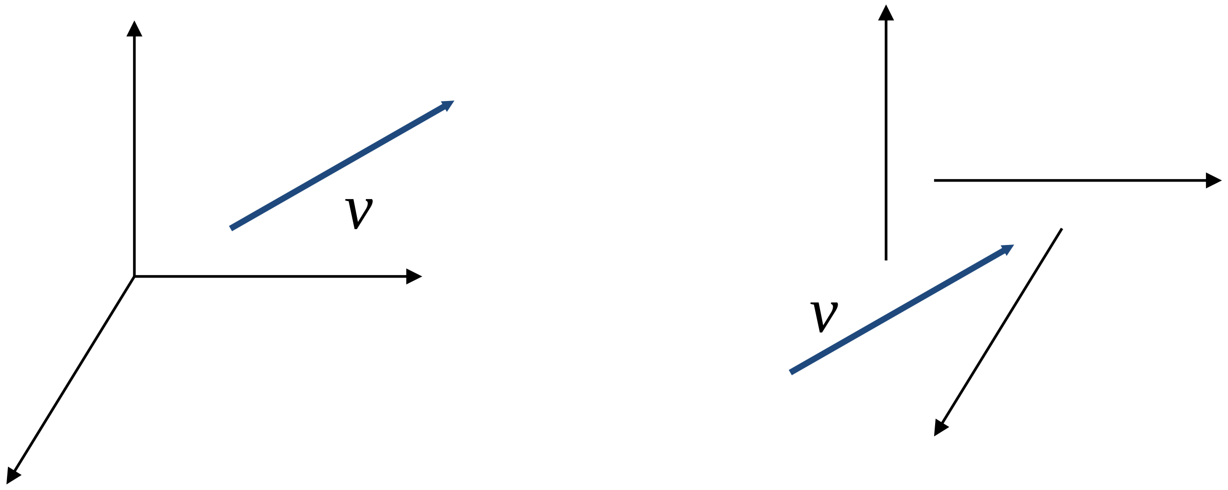
- $v=2v_1+3v_2-4v_3$

- $\mathbf{a}=[2\ 3\ -4]^T$

- Note that this representation is with respect to a particular basis

Coordinate Systems

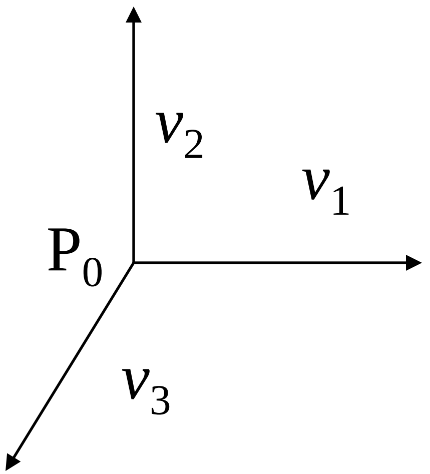
- Which is correct?



- Both are because vectors have no fixed location

Frames

- A coordinate system is insufficient to represent points
- If we work in an affine space we can add a single point, the *origin*, to the basis vectors to form a *frame*



Representation in a Frame

- Frame determined by (P_0, v_1, v_2, v_3)

- Within this frame, every vector can be written as

  $v=\alpha_1 v_1+ \alpha_2 v_2 +\ldots.+\alpha_n v_n$

- Every point can be written as

  $P = P_0 + \beta_1 v_1+ \beta_2 v_2 +\ldots.+\beta_n v_n$

Confusing Points and Vectors

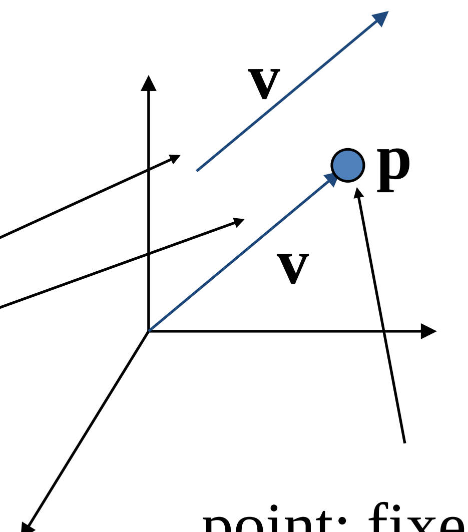
Consider the point and the vector

$$P = P_0 + \beta_1 v_1 + \beta_2 v_2 + \ldots + \beta_n v_n$$

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$$

They appear to have the similar representations

$\mathbf{p} = [\beta_1\ \beta_2\ \beta_3]$ $\mathbf{v} = [\alpha_1\ \alpha_2\ \alpha_3]$



Vector can be placed anywhere

point: fixed

A Single Representation

- If we define $0 \cdot P = \mathbf{0}$ and $1 \cdot P = P$ then we can write

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = [\alpha_1\ \alpha_2\ \alpha_3\ 0\ ]\,[v_1\ v_2\ v_3\ P_0]^{\mathrm{T}}$$

$$P = P_0 + \beta_1 \mathrm{v}_1 + \beta_2 \mathrm{v}_2 + \beta_3 \mathrm{v}_3 = [\beta_1\ \beta_2\ \beta_3\ 1\ ]\,[v_1\ v_2\ v_3\ P_0]^{\mathrm{T}}$$

- Thus we obtain the four-dimensional <u>*homogeneous coordinate*</u> representation

$$\mathbf{v} = [\alpha_1\ \alpha_2\ \alpha_3\ 0\ ]^{\mathrm{T}}$$

$$\mathbf{p} = [\beta_1\ \beta_2\ \beta_3\ 1\ ]^{\mathrm{T}}$$

Homogeneous Coordinates

- A three dimensional point $[x\ y\ z]$ is given as

  $\boldsymbol{p} = [x'\ y'\ z'\ w]^{\mathrm{T}} = [wx\ wy\ wz\ w]^{\mathrm{T}}$

- We return to a three dimensional point (for $w \neq 0$) by

  $x = x'/w$; $y = y'/w$; $z = z'/w$

- If $w=0$, a vector.

- Homogeneous coordinates replaces points in three dimensions by lines through the origin in four dimensions.

Homogeneous Coordinates and Computer Graphics

- Homogeneous coordinates are key to all computer graphics systems
  - All standard transformations (rotation, translation, scaling) can be implemented with matrix multiplications using 4 x 4 matrices
  - Hardware pipeline works with 4 dimensional representations
  - For orthographic viewing, we can maintain $w=0$ for vectors and $w=1$ for points
  - For perspective we need a *perspective division*

Change of Coordinate Systems

- Consider two representations of a the same vector with respect to two different bases. The representations are

$$\mathbf{a}=[\alpha_1\ \alpha_2\ \ \alpha_3\ ]$$
$$\mathbf{b}=[\beta_1\ \ \beta_2\ \ \beta_3]$$

where

$$\mathrm{v}=\alpha_1 v_1+\ \alpha_2 v_2\ +\alpha_3 v_3=[\alpha_1\ \alpha_2\ \alpha_3]\ [v_1\ v_2\ v_3]^{\mathrm{T}}$$
$$=\beta_1 u_1+\ \beta_2 u_2\ +\beta_3 u_3=[\beta_1\ \beta_2\ \beta_3]\ [u_1\ u_2\ u_3]^{\mathrm{T}}$$

Representing second basis in terms of first

- Each of the basis vectors, u1,u2, u3, are vectors that can be represented in terms of the first basis

$$u_1 = \gamma_{11}v_1+\gamma_{12}v_2+\gamma_{13}v_3$$
$$u_2 = \gamma_{21}v_1+\gamma_{22}v_2+\gamma_{23}v_3$$
$$u_3 = \gamma_{31}v_1+\gamma_{32}v_2+\gamma_{33}v_3$$