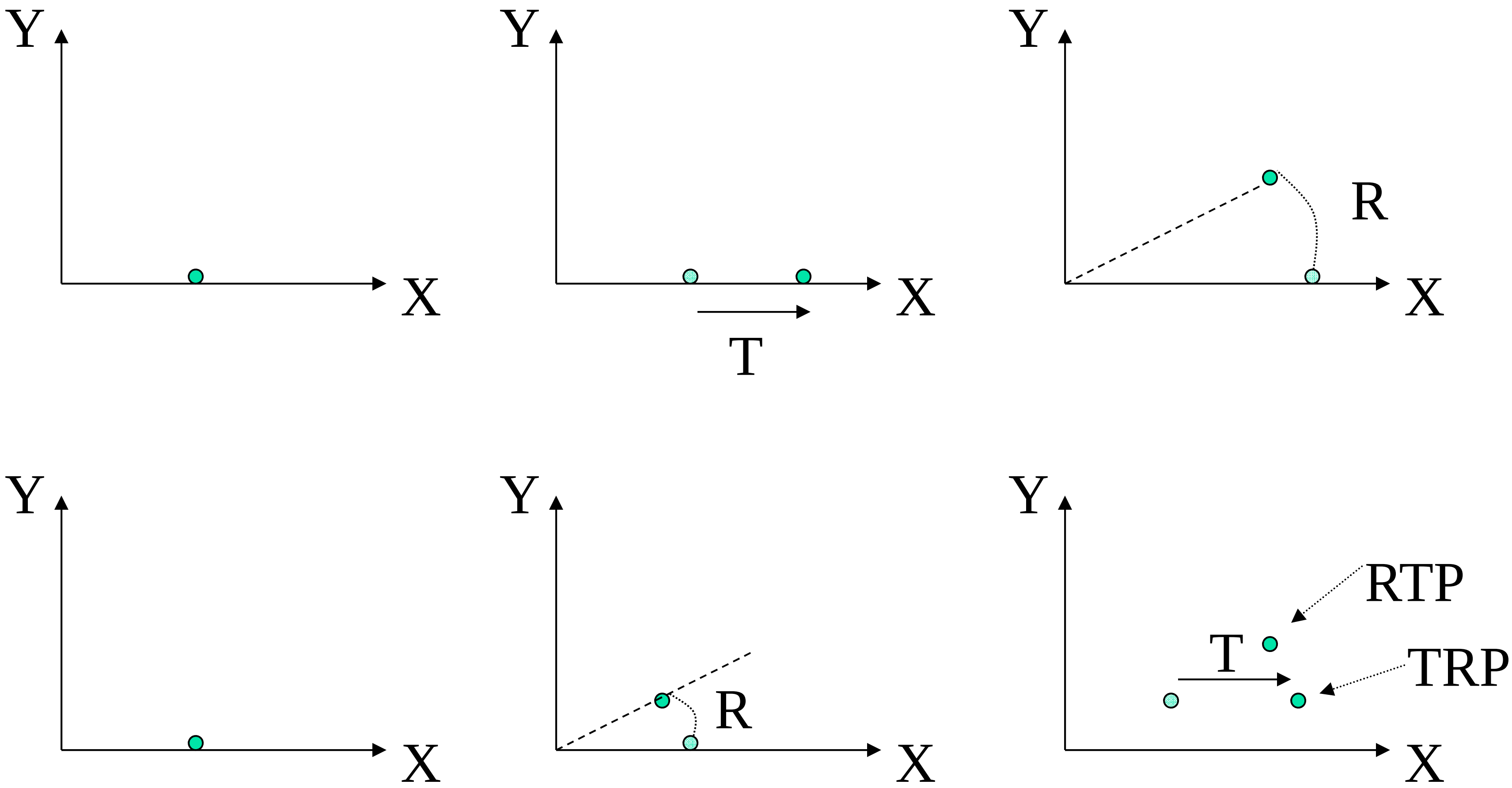
CS 112 – Transformations II

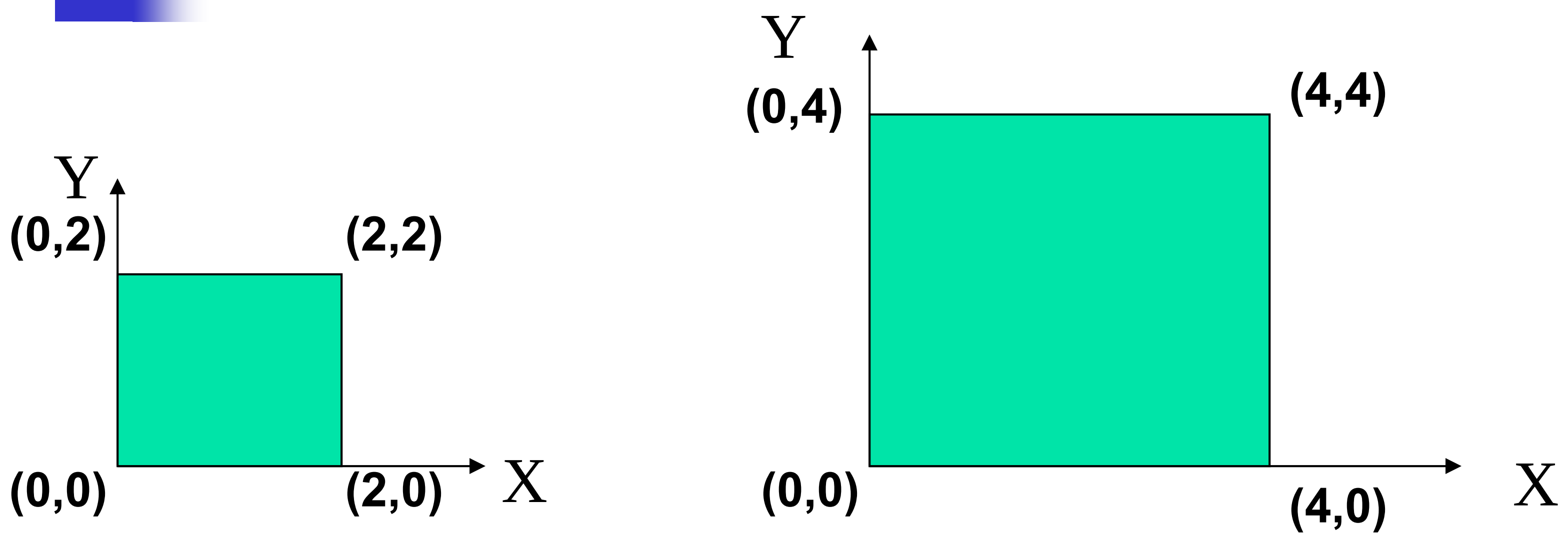
Composition of Transformations

- Example: A point P is first translated and then rotated. Translation matrix T, Rotation Matrix R.
  - After Translation: $P' = TP$
  - After Rotation: $P'' = RP' = RTP$
- Example: A point is first rotated and then translated.
  - After Rotation: $P' = RP$
  - After Translation: $P'' = TP' = TRP$
- Since matrix multiplication is not commutative,
  - $RTP \neq TRP$

Composition of Transformations

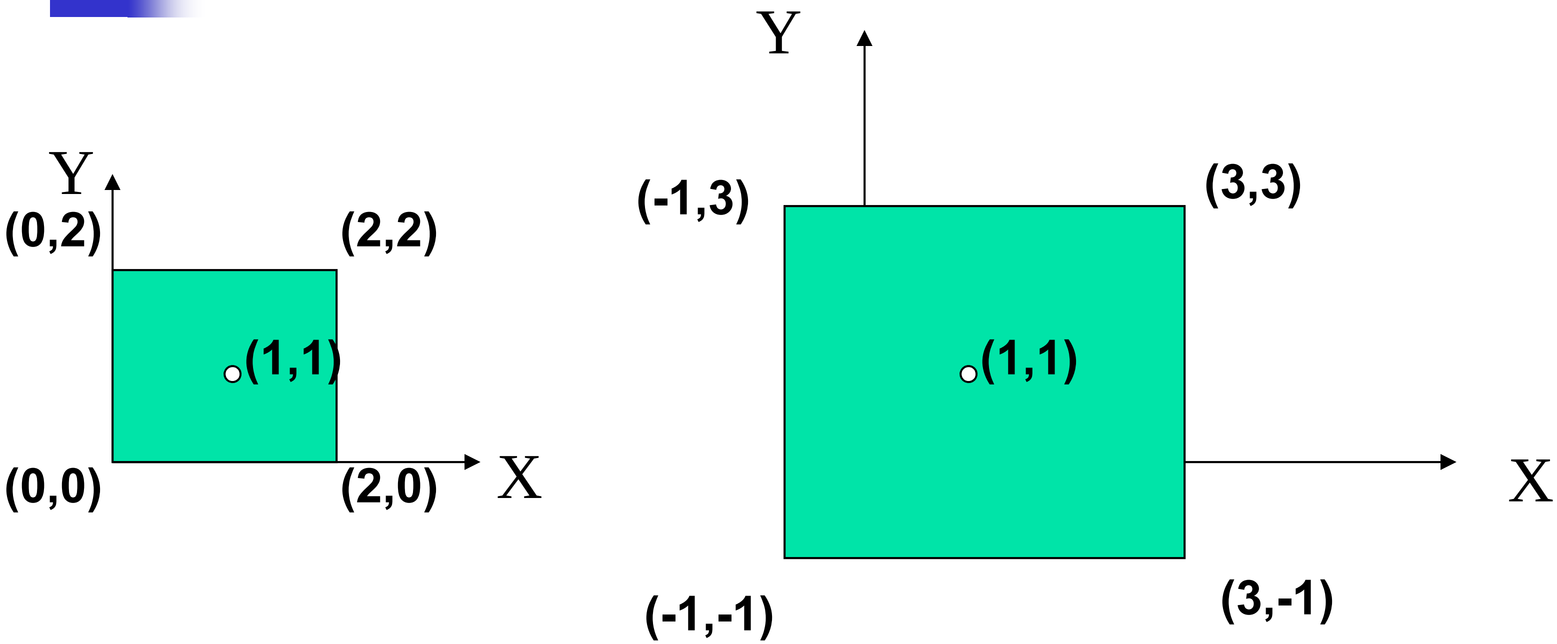


Scaling About a point



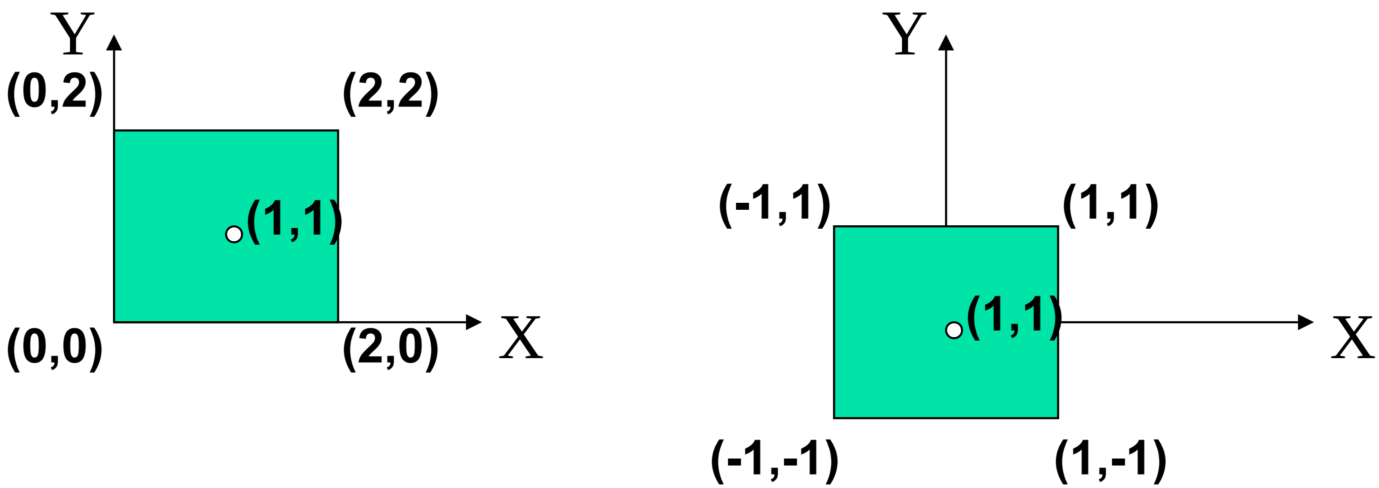
Scaling about origin -> Origin is fixed with transformation

Scaling About a point



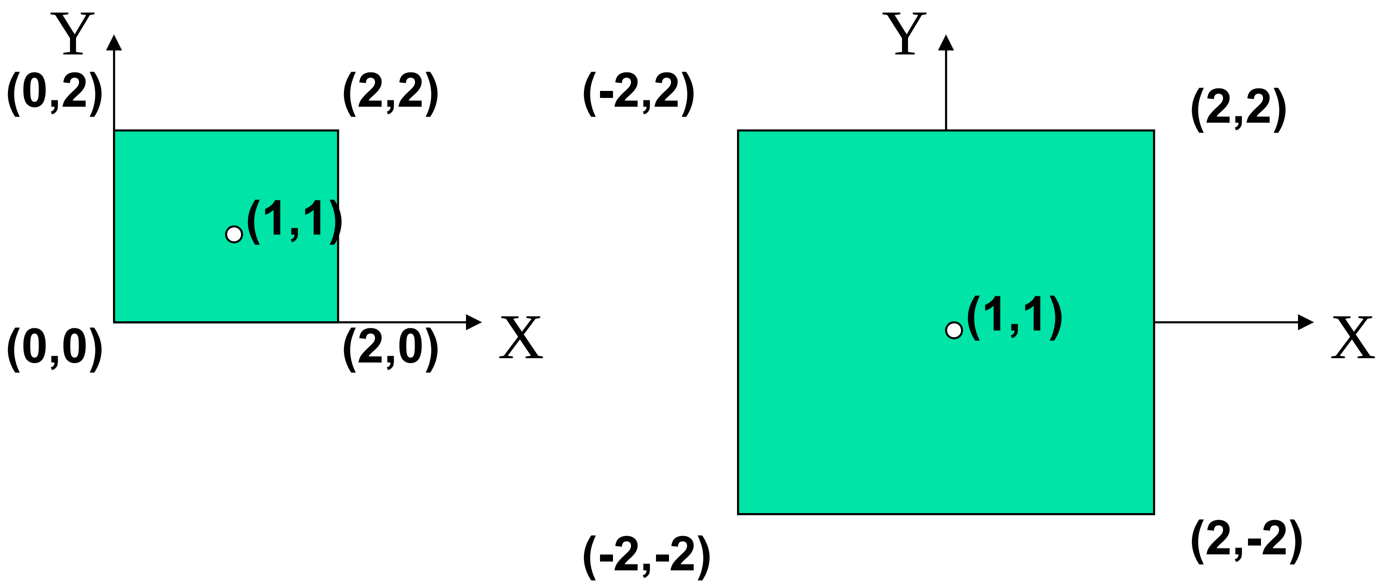
Scaling about center -> Center is fixed with transformation

Done by concatenation



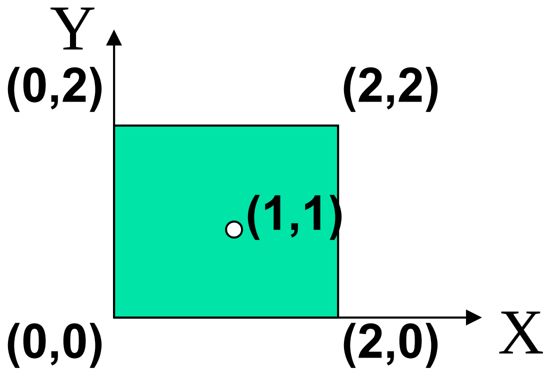
Translate so that center coincides with origin - T(-1,-1).

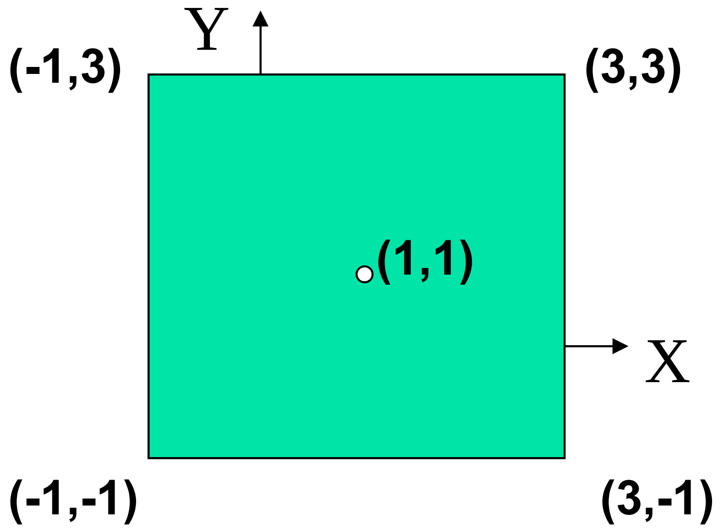
Done by concatenation



Scale the points about the center – S(2,2)

Done by concatenation





Translate it back by reverse parameters – T(1,1)

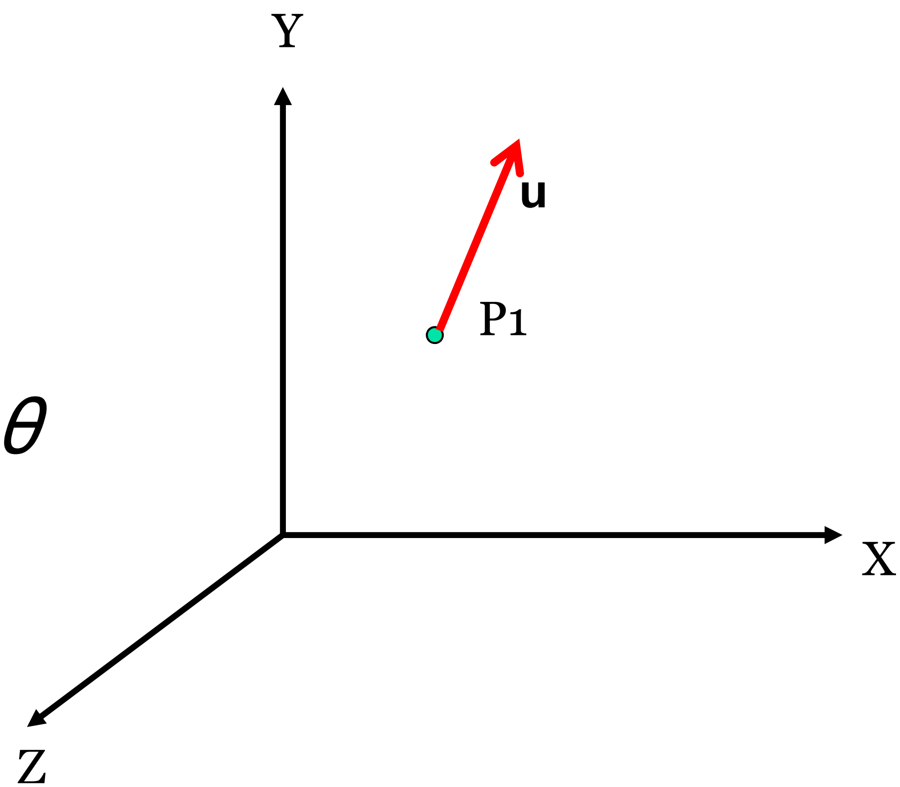
Total Transformation: T(1,1) S(2,2) T(-1,-1) P

Rotation about a fixed point

- z-axis rotation of θ about its center P_f
- Translate by $-P_f$: $T(-P_f)$
- Rotate about z-axis : $R_z(\theta)$
- Translate back by P_f : $T(P_f)$
- Total Transformation $M = T(P_f)R_z(\theta)T(-P_f)$

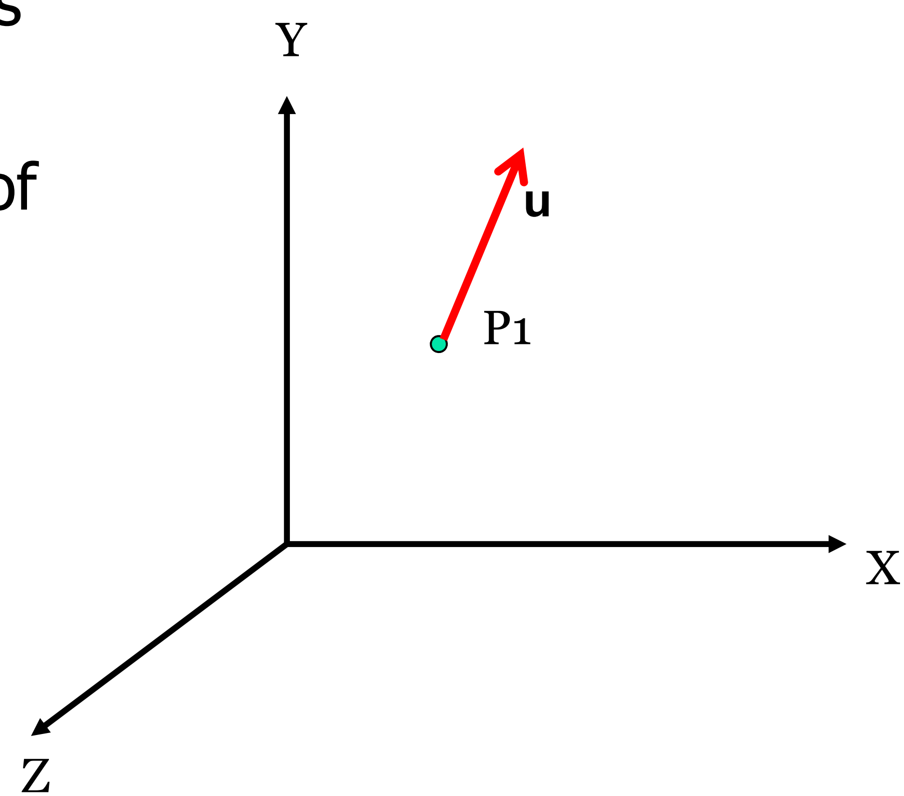
Rotation About an Arbitrary Axis

- Axis given by
  - Unit vector u
  - Rooted at point P_1
- Anticlockwise angle of rotation is θ
- Rotate all points u by θ



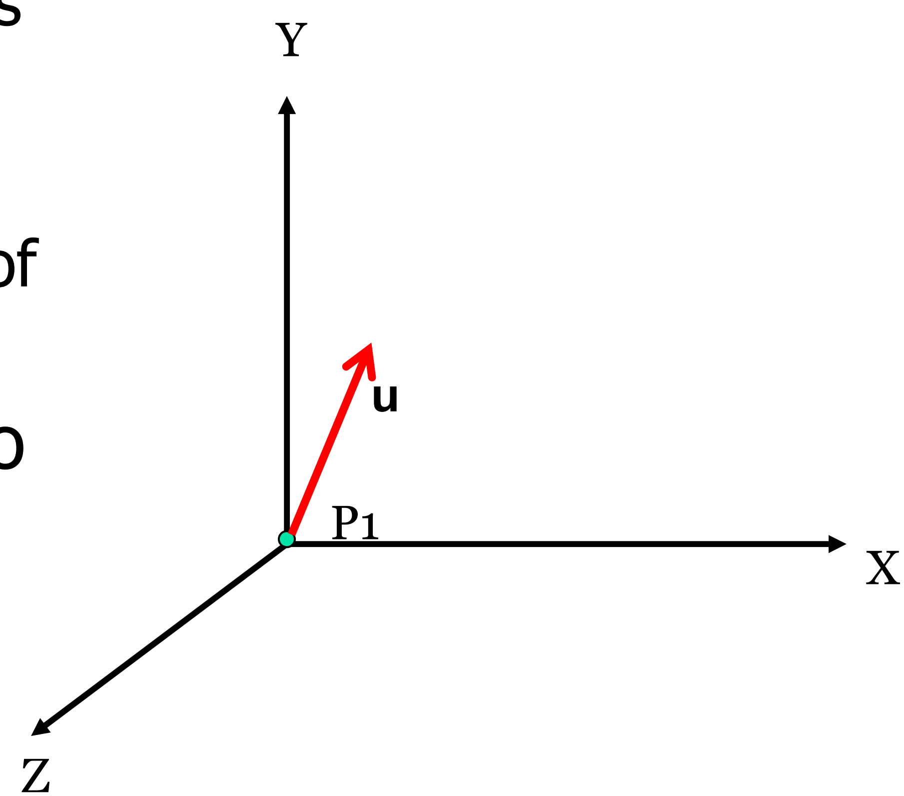
Rotation about an Arbitrary Axis

- Make u coincide with Z-axis
  - Translate P_1 to origin
    - Coincides one point of the axis with origin



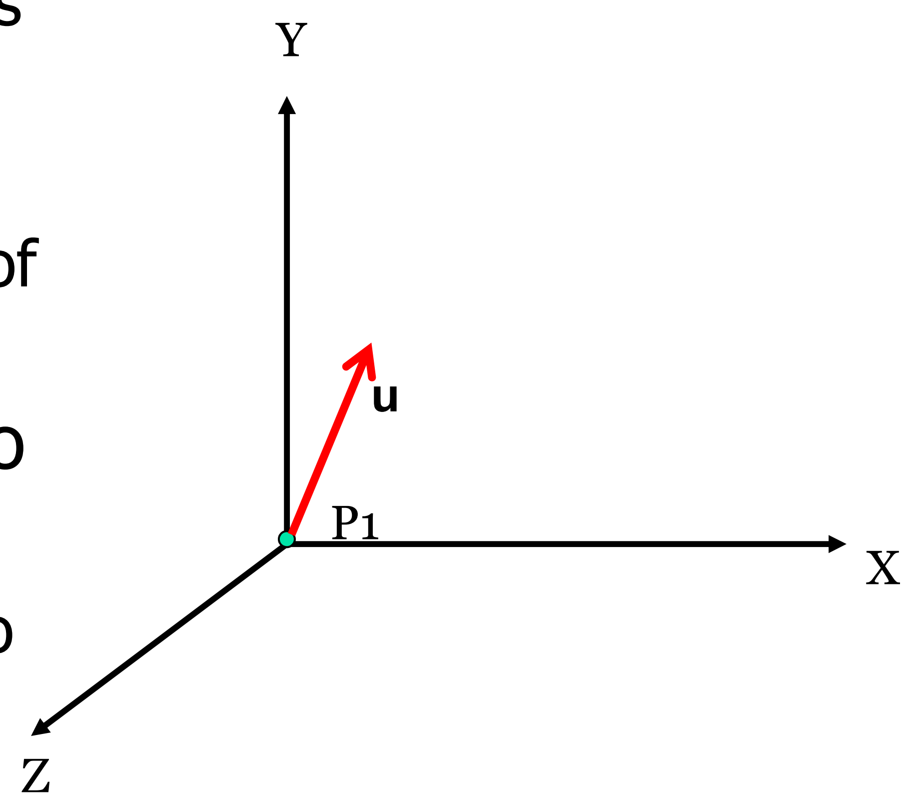
Rotation about an Arbitrary Axis

- Make u coincide with Z-axis
  - Translate P_1 to origin: $T(-P_1)$
    - Coincides one point of the axis with origin
  - Rotate shifted axis to coincide with Z axis



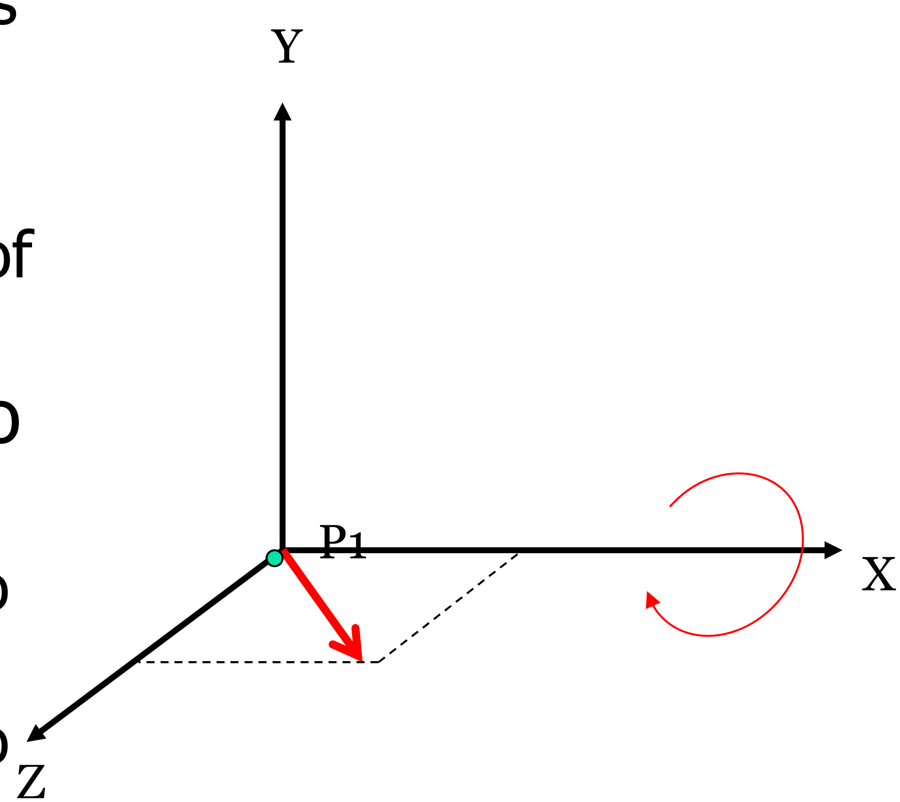
Rotation about an Arbitrary Axis

- Make u coincide with Z-axis
  - Translate P_1 to origin: $T(-P_1)$
    - Coincides one point of the axis with origin
  - Rotate shifted axis to coincide with Z axis
    - R_1: Rotate about X to lie on XZ plane



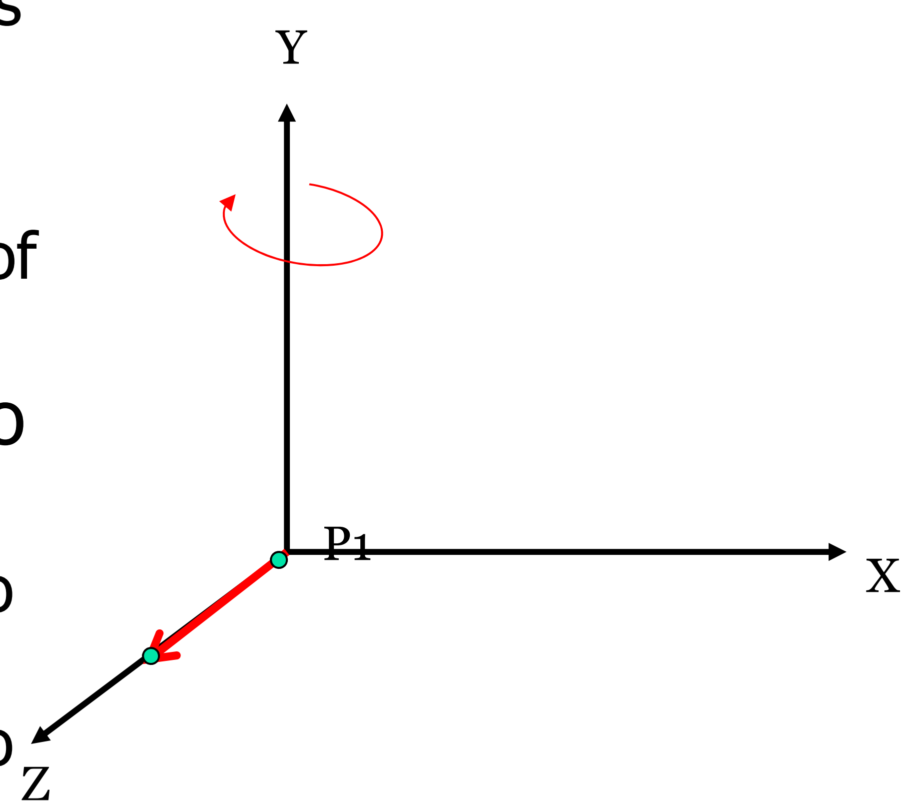
Rotation about an Arbitrary Axis

- Make u coincide with Z-axis
  - Translate P_1 to origin: T(-P1)
    - Coincides one point of the axis with origin
  - Rotate shifted axis to coincide with Z axis
    - R_1: Rotate about X to lie on XZ plane
    - R_2: Rotate about Y to lie on Z axis



Rotation about an Arbitrary Axis

- Make u coincide with Z-axis
  - Translate P_1 to origin: $T(-P_1)$
    - Coincides one point of the axis with origin
  - Rotate shifted axis to coincide with Z axis
    - R_1: Rotate about X to lie on XZ plane
    - R_2: Rotate about Y to lie on Z axis

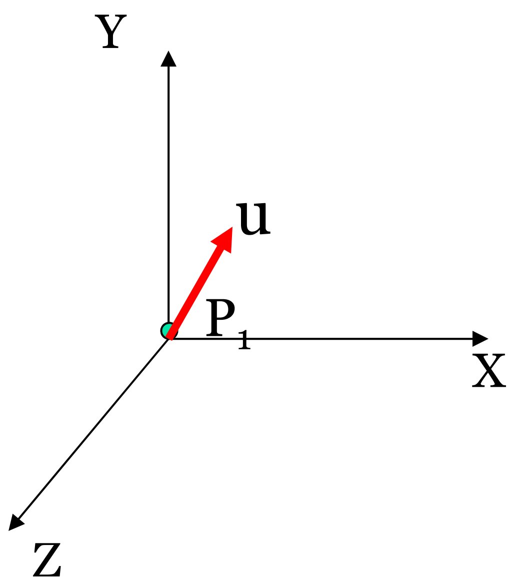


Rotation about an Arbitrary Axis

- Make the axis coincide with the Z-axis
  - Translation to move P_1 to the origin: $T(-P_1)$
    - Coincides one point of the axis with origin
  - Rotation to coincide the shifted axis with Z axis
    - R_1: Rotation around X such that the axis lies on the XZ plane.
    - R_2: Rotation around Y such that the axis coincides with the Z axis
- R_3: Rotate the scene around the Z axis by an angle θ
- Inverse transformations of R_2, R_1 and T to bring back the axis to the original position
- $M = T^{-1} R_1^{-1} R_2^{-1} R_3 R_2 R_1 T$

Translation

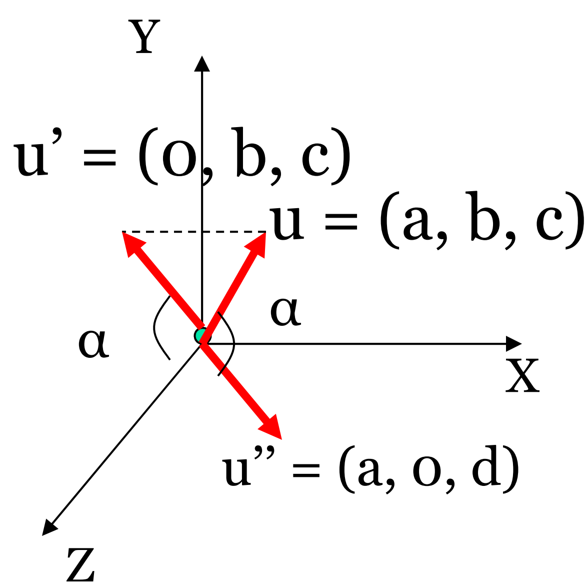
- After translation



$$u = \frac{V}{|V|} = (a, b, c)$$

Rotation about X axis

- Rotate u about X so that it coincides with XZ plane



Project u on YZ plane : u' (o, b, c)

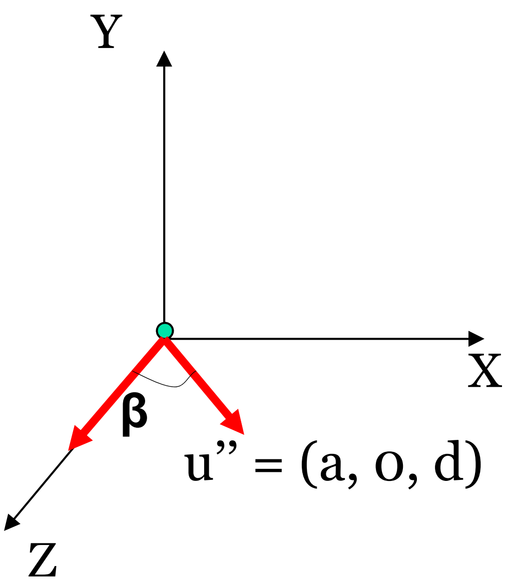
α is the angle made by u' with Z axis

$\cos\alpha = c/\sqrt{b^2+c^2} = c/d$
$\sin\alpha = b/d$

$$\mathbf{R}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Y axis

- Rotate u” about Y so that it coincides with Z axis



$\text{Cos}\ \beta = d/\sqrt{a^2+d^2} = d/\sqrt{a^2+b^2+c^2} = d$

$\text{Sin}\ \beta = a$

$$\mathbf{R_2} = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Z axis

- Rotate by θ about Z axis

$$\mathbf{R}_3 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $M = T^{-1} R_1^{-1} R_2^{-1} R_3(\theta) R_2(\beta) R_1(\alpha) T$

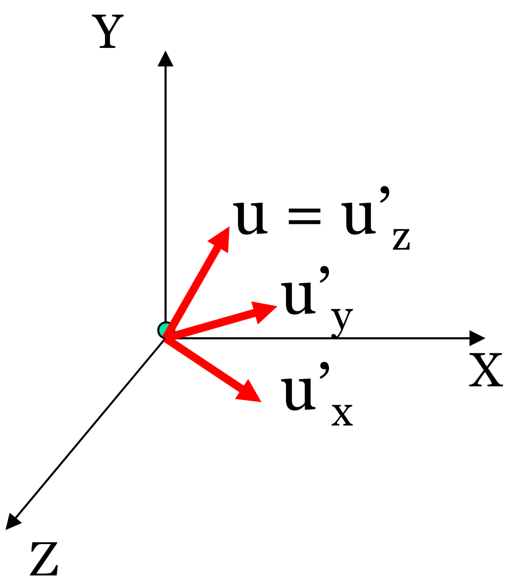
  $= T^{-1} R_x^{-1} R_y^{-1} R_z(\theta) R_y(\beta) R_x(\alpha) T$

  $= T^{-1} R_x(-\alpha) R_y(-\beta) R_z(\theta) R_y(\beta) R_x(\alpha) T$

Faster Way

- Faster way to find R_2R_1
  - u_x, u_y, u_z are unit vectors in the X, Y, Z direction

Set up a coordinate system where $u = u'_z$



$$u'_z = \frac{u}{|u|}$$

$$u'_y = \frac{u \times u_x}{|u \times u_x|}$$

$$u'_x = u'_y \times u'_z$$

$$R = R_2R_1 = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_1^{-1}R_2^{-1} = R^{-1}$$

Rigid and Affine Transformations

- Rigid (Does not deform the object)
  - Preserves angles and lengths
  - Rotation and translation
- Affine (Deforms in a restricted manner)
  - Preserves collinearity and ratio of lengths
  - Angles may not be preserved
  - Scaling and shear are affine but not rigid
  - Can be expressed as a combination of rotation, translation, scaling and shear

Transformations

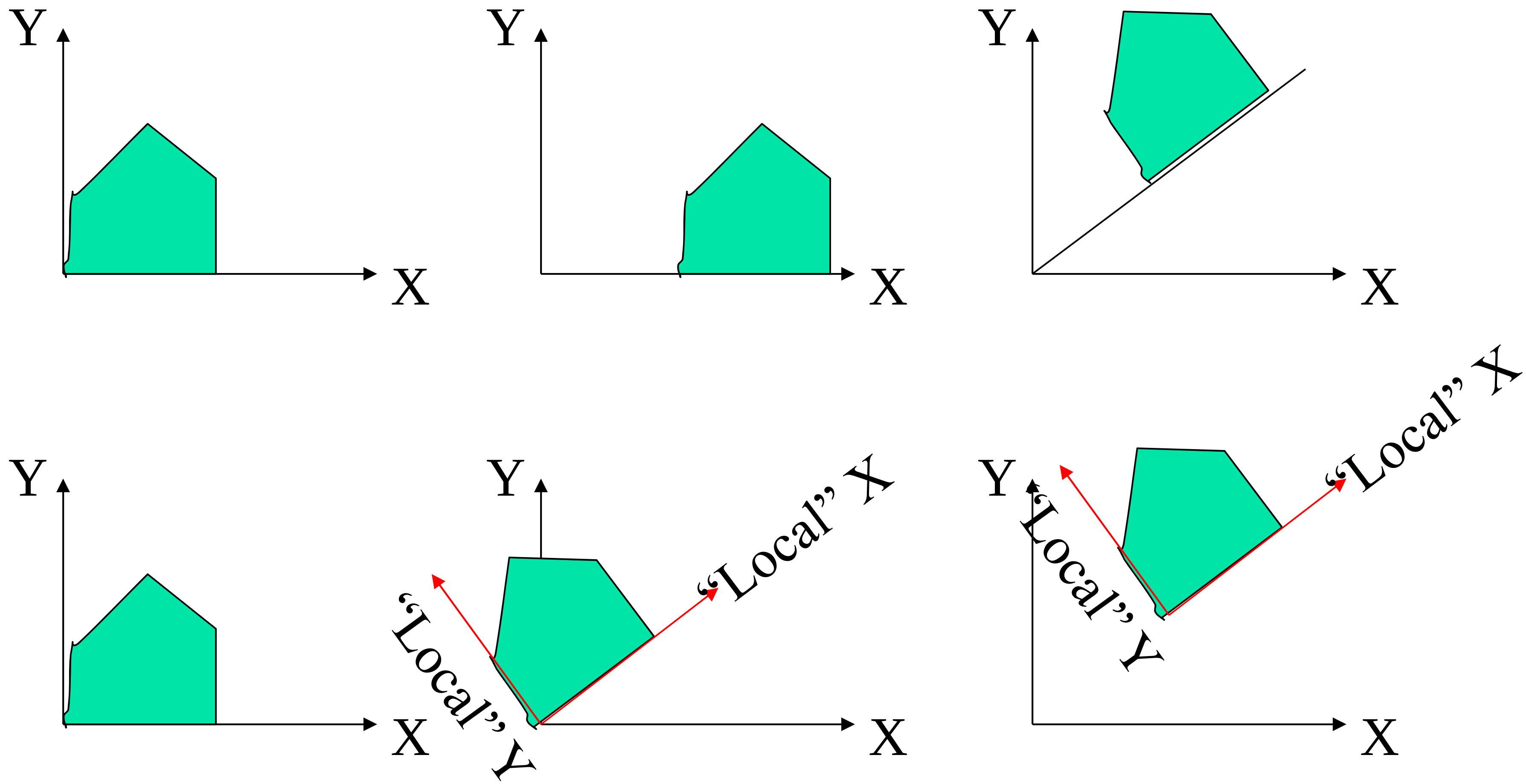
- Modelview Transformation generates modelview matrix (GL_MODELVIEW)
- Projection Transformation generates projection matrix (GL_PROJECTION)
- Premultiply modelview with projection and apply it to all the vertices of the model

Coordinate Systems

- You Say: A point P is “first translated” and “then rotated”.
- You Write: P’ = RTP (write Rotation first, then translation, then the point)
- Right to Left: “Global Coordinate System”
- Left to Right: “Local Coordinate System”
- Results of both are same
  - Since matrix multiplication is associative
  - Just the interpretation is different.

Local/Global Coordinate Systems

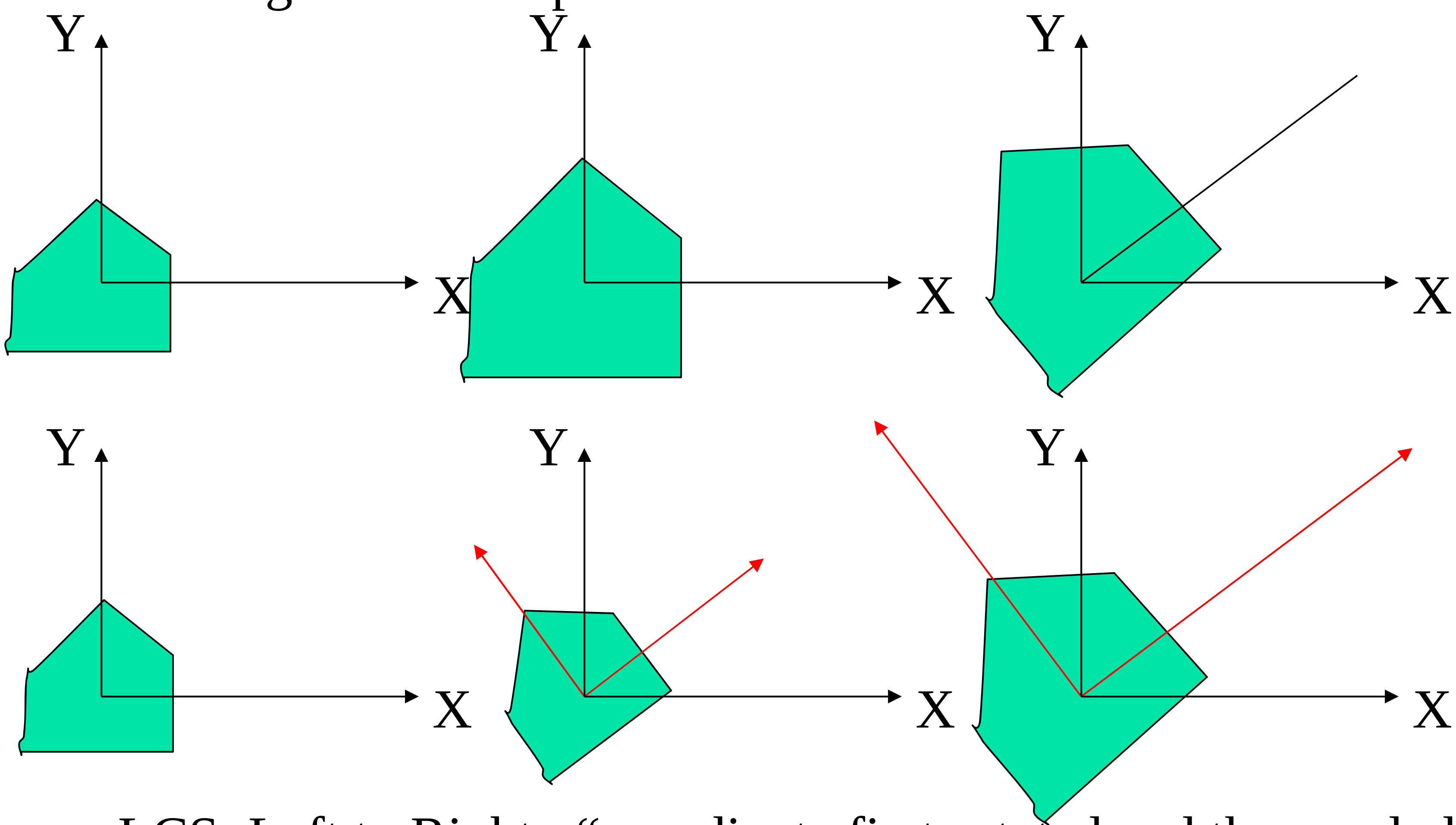
GCS: Right to Left: “point is first translated and then rotated”



LCS: Left to Right: “coordinate first rotated and then translated”

Local / Global Coordinate Systems

GCS: Right to Left: “point is first scaled and then rotated”



LCS: Left to Right: “coordinate first rotated and then scaled”

Coordinate Systems for Modelview

OpenGL follows LOCAL COORDINATE SYSTEM

```
glLoadIndentity()
glTranslate(…)
glRotate(…)
glScale(…)
DrawModel()
```

Means: TRS.P (You issue transformation commands in the order your write!!)

Loading, Pushing and Popping

- glLoadmatrix(myarray)
  - If it is easier to set up the matrix yourself, like shear
- glPushmatrix(), glPopMatrix()

```
glPushMatrix();
glTranslatef(…);
glScalef(…);
glPopMatrix();
```

OpenGL Stack

```
Function 1 (…)
glLoadIndentity()
glTranslate(…)
glRotate(…)
DrawModel(All objs)
Function 2(…)
    ⋮
```

```
Function 2 (…)
glPushmatrix(…)
glScale(…)
DrawModel(Obj A)
DrawModel(Obj B)
glPopMatrix(…)
```