Set the ball rolling!

3 Transformations

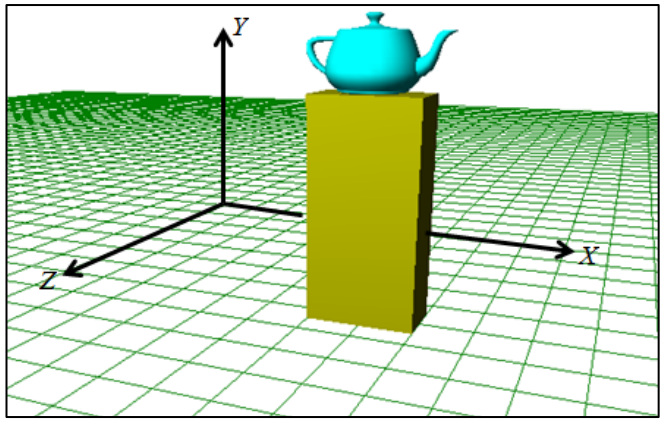
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

R. Mukundan (mukundan@canterbury.ac.nz)

Model-View Transformation

- Objects are created in their own local coordinate space and then transformed into the world coordinate space.
- They are transformed again into the coordinate space of the camera to generate the view as seen by the camera.

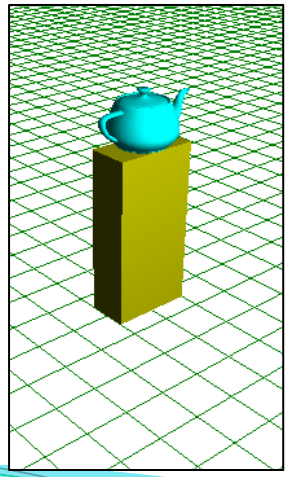
Model Transformation. Eg. Scale, Rotn, Translation



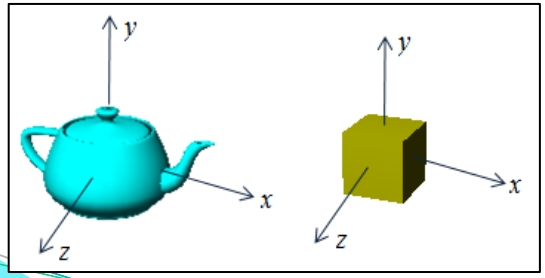
World Space

View Transformation

`gluLookAt(..)`



Camera Space



Modelling Space

`glutSolidCube(..)`
`glutSolidTeapot(..)`

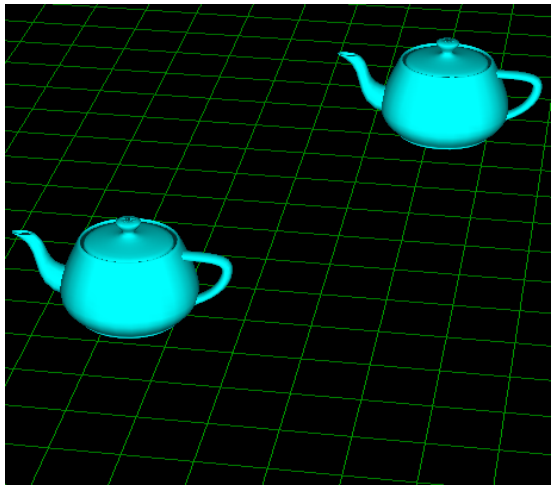
Transformations in OpenGL

- OpenGL supports the following fundamental three-dimensional transformations:
  - Translations: `glTranslatef(a, b, c);`
  - Rotations: `glRotatef(angle, l, m, n);`
  - Scale Transformations: `glScalef(sx, sy, sz);`
  - Generalized transformation: `glMultMatrixf(mat);`
- Transformations are stored as 4x4 matrices.
- Transformations form part of the **model-view matrix**.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

- OpenGL maintains a **transformation stack**. The top of the stack is the current transformation matrix.

Translation



- Equations:

  $x' = x + a$

  $y' = y + b$

  $z' = z + c$

- Matrix Form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Note: The above matrix representation is possible only if we use homogeneous coordinates.

- OpenGL function: `glTranslatef(a, b, c)`

Scaling

- Equations:

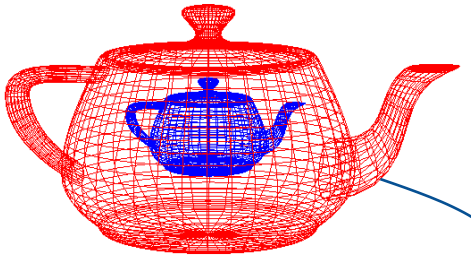
  $x' = x\,a$

  $y' = y\,b$

  $z' = z\,c$

- Matrix Form:

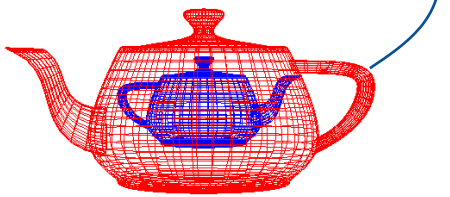
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



`glutScalef(2.5, 2.5, 2.5)`

Scaled teapot

- OpenGL function: `glScalef(a, b, c)`
- A negative scale factor corresponds to a reflection.
- A zero scale factor corresponds to a projection



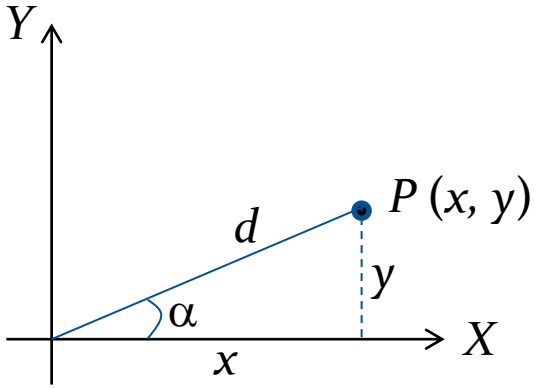
`glutScalef(-2, 2, 2)`

Rotational Transformation

Consider a point $P(x, y)$ at a distance d from the origin, and angle α from X-axis.

$$x = d\cos\alpha$$

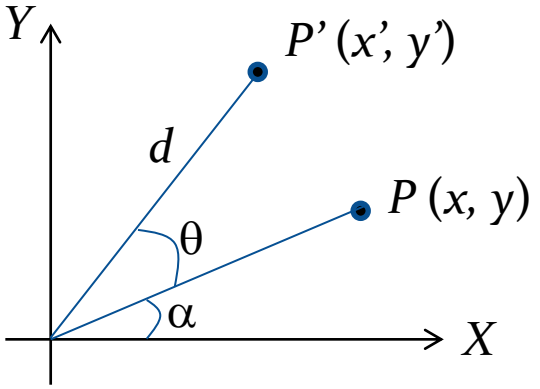
$$y = d\sin\alpha$$



If P undergoes a rotational transformation (about the z axis) by an angle θ, the new point P' has the same distance d, but angle $\alpha+\theta$:

$$x' = d\cos(\alpha+\theta) = d\cos\alpha\cos\theta - d\sin\alpha\sin\theta$$

$$y' = d\sin(\alpha+\theta) = d\cos\alpha\sin\theta + d\sin\alpha\cos\theta$$

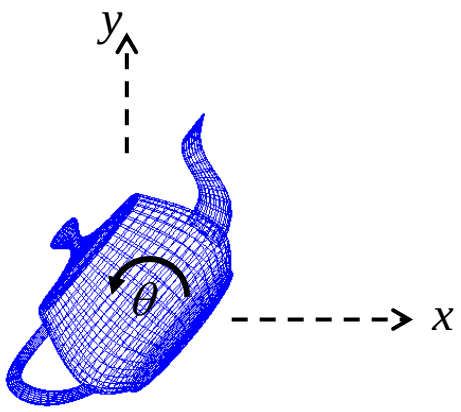


Therefore,

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

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

Rotation About the Z-axis



- Equations:

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

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

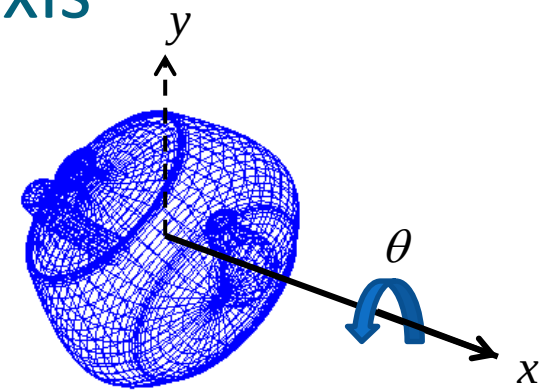
  $z' = z$

- Matrix Form:

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

- OpenGL function: `glRotatef(theta, 0, 0, 1)`

Rotation About the X-axis



- Equations:

  $x' = x$

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

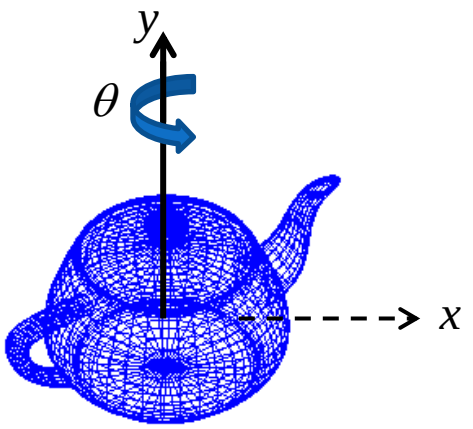
  $z' = y\sin\theta + z\cos\theta$

- Matrix Form:

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

- OpenGL function: `glRotatef(theta, 1, 0, 0)`

Rotation About the Y-axis



- Equations:

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

  $y' = y$

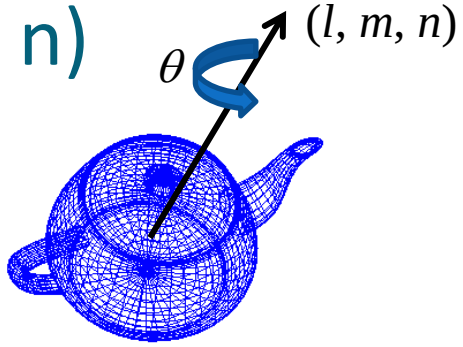
  $z' = -x\sin\theta + z\cos\theta$

- Matrix Form:

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

- OpenGL function: `glRotatef(theta, 0, 1, 0)`

Rotation About a Vector (l, m, n)



- The vector (*l*, *m*, *n*) need not be a unit vector – it will be converted to a unit vector before computing the transformation matrix.
- Matrix Form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} l^2(1-\cos\theta)+\cos\theta & lm(1-\cos\theta)-n\sin\theta & l\,n(1-\cos\theta)+m\sin\theta & 0 \\ lm(1-\cos\theta)+n\sin\theta & m^2(1-\cos\theta)+\cos\theta & mn(1-\cos\theta)-l\sin\theta & 0 \\ l\,n(1-\cos\theta)-m\sin\theta & mn(1-\cos\theta)+l\sin\theta & n^2(1-\cos\theta)+\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- All rotations are applied about vectors passing through the origin.
- OpenGL function:

```
glRotatef(theta, l, m, n)
```

Composite Transformations

- A transformation Q_1 followed by a transformation Q_2 followed by a transformation Q_3...
- Matrix Form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = [Q_3][Q_2][Q_1] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Order of Transformations

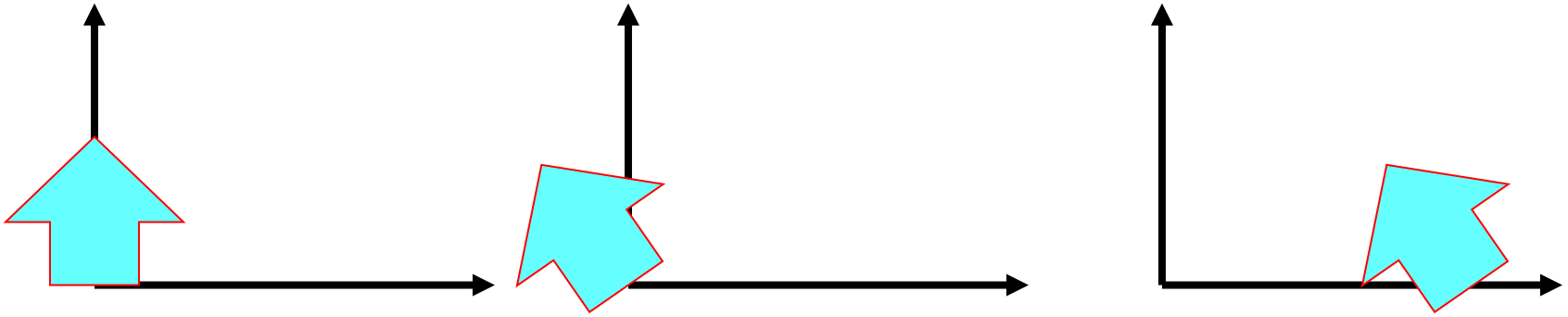
- The order in which a sequence of transformation is applied is important.
- OpenGL implementation:

```
glRotatef(10.0, 1.0, 0., 0.);
glTranslatef(0.0, 1.2, 0.5);
glRotatef(25.0, 0., 1.0, 0.0);
glScalef(2.5, 2.0, 1.0);
drawObject();
```

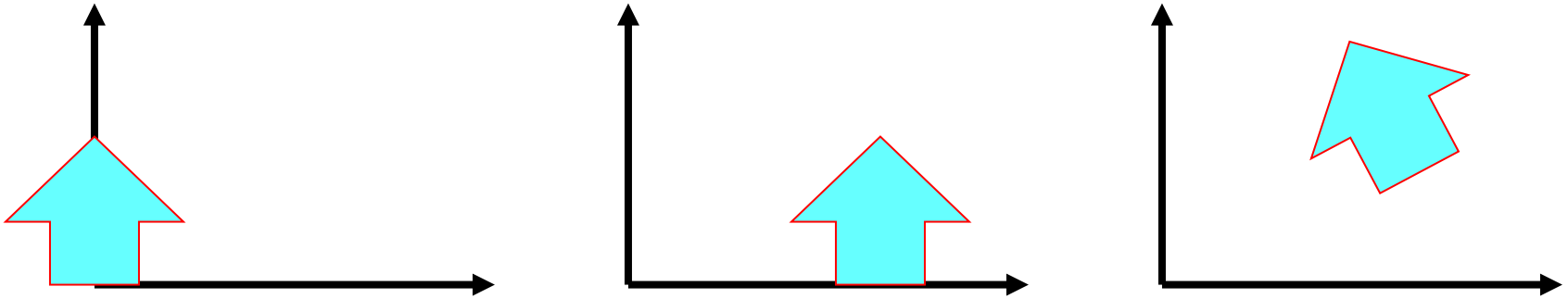
Order of Transformations

Order of Transformations

Rotation followed by Translation:



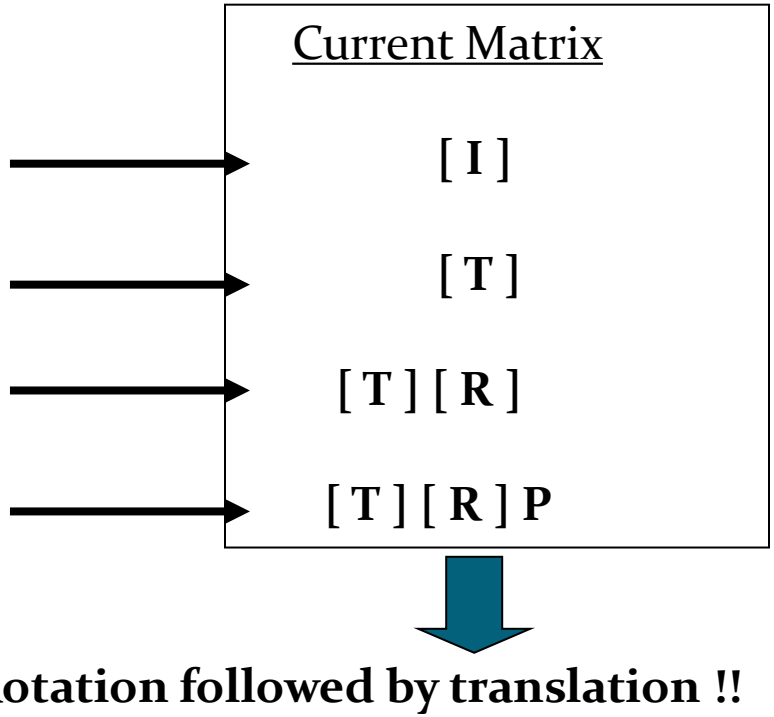
Translation followed by Rotation:



OpenGL Transformations

- OpenGL ***post***multiplies the current matrix with the new transformation matrix

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(tx, ty, 0);
glRotatef(theta, 0, 0, 1.0);
glVertex2f(x, y);
```



- We code the sequence of transformations in the opposite order from that required!

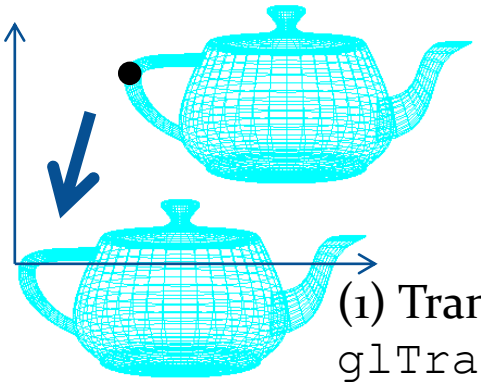
Euler Angles

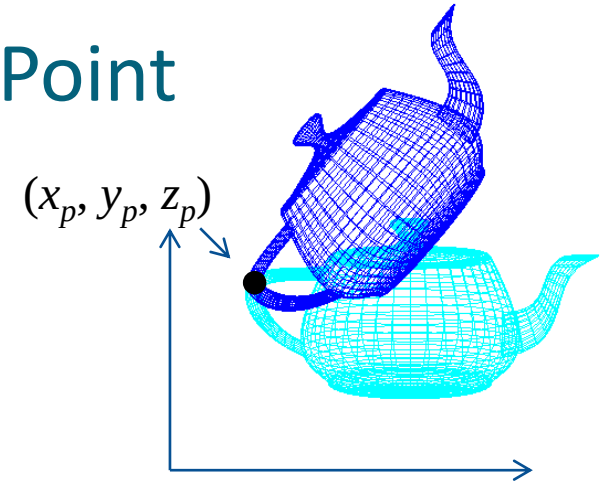
- A general rotation about the origin in 3D space can be decomposed into a sequence of three elementary rotations about the coordinate axes (typically in the following sequence):
  - A rotation about *x* by an angle ψ.
  - A rotation about *y* by an angle ϕ.
  - A rotation about *z* by an angle θ.
- The angles ψ, ϕ, θ are called Euler Angles.
- The composite rotation is given by

$$R_z(\theta)\, R_y(\phi)\, R_x(\psi)$$

```
glRotatef(theta, 0., 0., 1.);
glRotatef(phi, 0., 1., 0.);
glRotatef(psi, 1., 0., 0.);
```

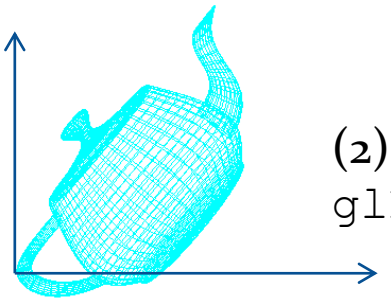
Rotation About a Pivot Point





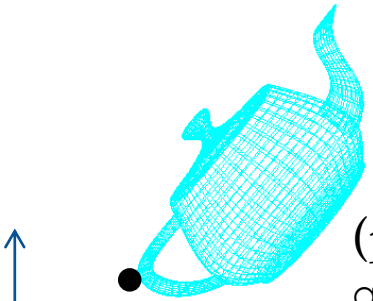
(1) Translate to origin:

```
glTranslatef(-xp, -yp, -zp);
```



(2) Rotate:

```
glRotatef(angle, l, m, n);
```



(3) Translate back to (xp, yp, zp):

```
glTranslatef(xp, yp, zp);
```

```
glTranslatef(xp, yp, zp);
glRotatef(angle, l, m, n);
glTranslatef(-xp, -yp, -zp);
glutWireTeapot(1.0);
```

Shear Transformation (2D)

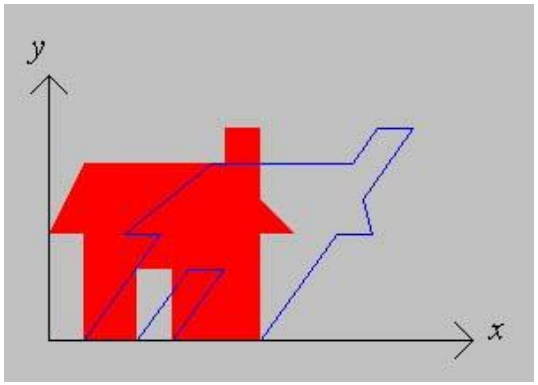
- A shear transformation along the *x*-axis shifts a point along the *x* direction by an amount proportional to the point's distance from the *x* axis.
- Equations:

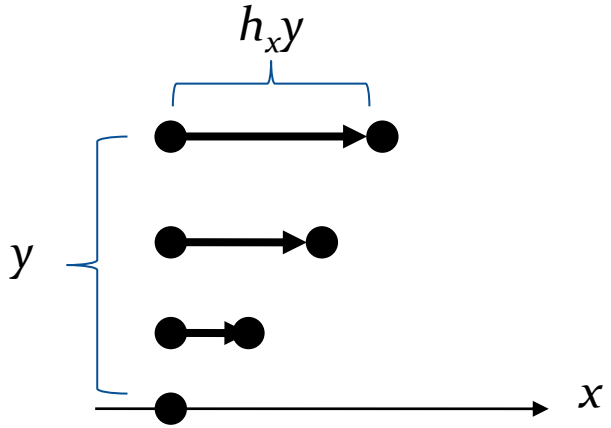
  $x' = x + h_x\ y$

  $y' = y$

  $z' = z$
- Matrix Form:

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





Shear Transformation (3D)

- If a three-dimensional point (x, y, z) is shifted along both x and z proportional to the point's distance from the xz-plane, then we have a shear transformation with equations

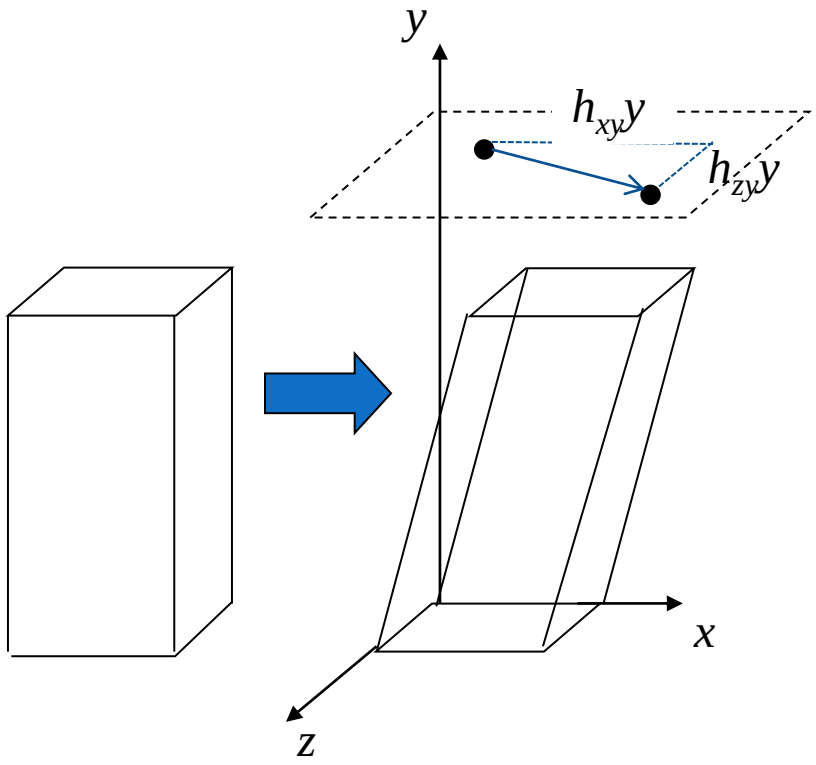
  $x' = x + h_{xy}\ y$

  $y' = y$

  $z' = z + h_{zy}\ y$

- Matrix form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & h_{xy} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Inverse Transformations

If $\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = [T] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ then, $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [T]^{-1} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$

Examples:

$$T^{-1}(t_x, t_y, t_z) = T(-t_x, -t_y, -t_z)$$

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

$$S^{-1}(s_x, s_y, s_z) = S\left(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z}\right)$$

$$H^{-1}(h_{xz}, h_{yz}) = H(-h_{xz}, -h_{yz})$$

General Properties

	Line	Angle	Distance	Volume
Translation	Yes	Yes	Yes	Yes
Rotation	Yes	Yes	Yes	Yes
Scaling	Yes	No	No	No
Reflection	Yes	Yes	Yes	Yes
Shear	Yes	No	No	Yes

Rigid-body Transformation

Affine Transformation

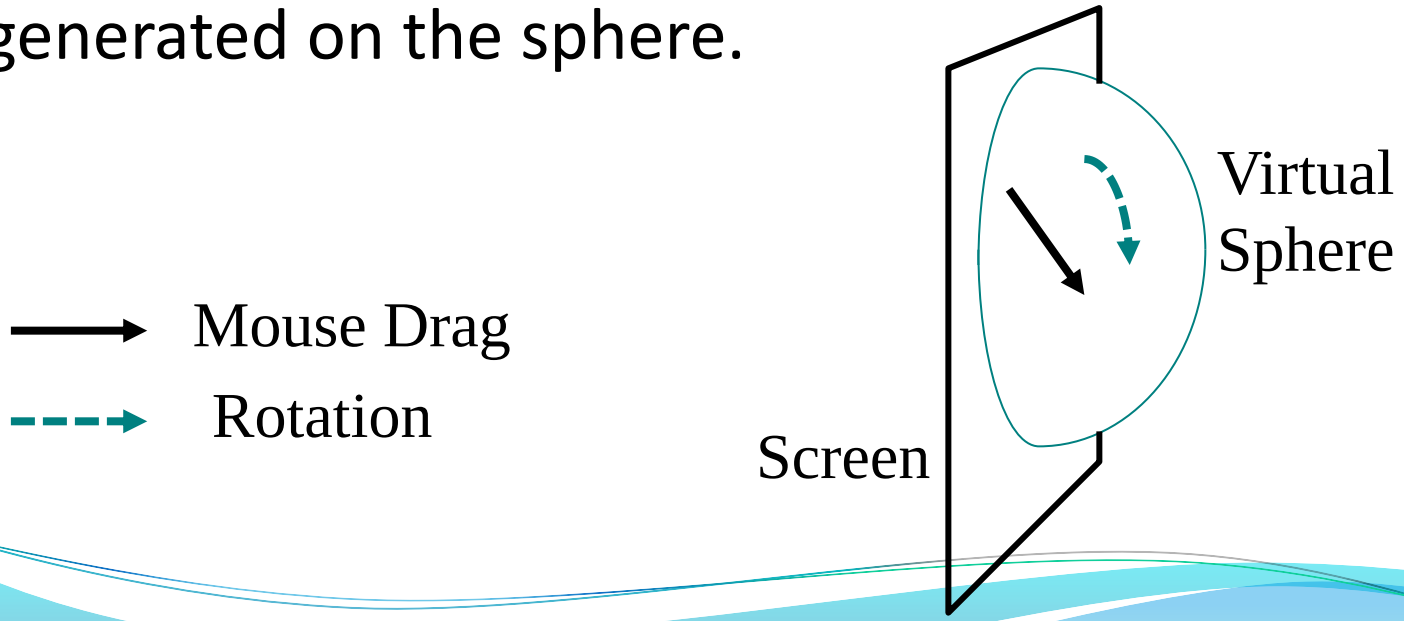
- A general linear transformation followed by a translation is called an affine transformation.
- Matrix form:

$$\begin{bmatrix} x'_p \\ y'_p \\ z'_p \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

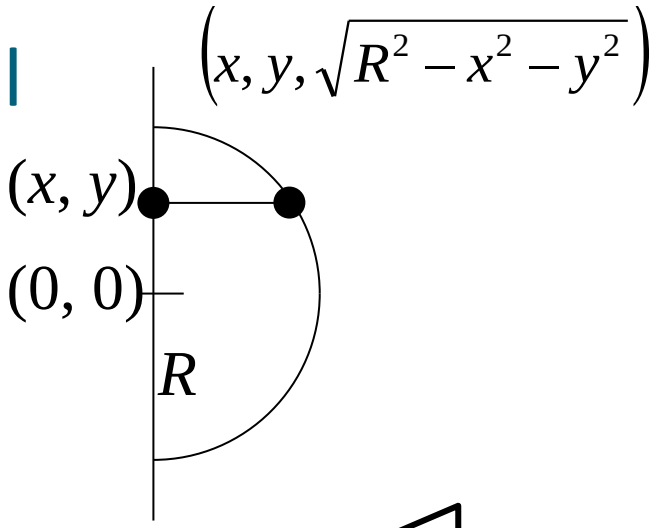
- Translation, rotation, scaling and shear transformations are all affine transformations.
- Under an affine transformation, line segments transform into line segments, and parallel lines transform into parallel lines.

Virtual Trackball

- A user interface for drag-rotating an object.
- Assume that the objects displayed on the screen are attached to a virtual sphere.
- When the mouse is dragged from one point to another on the screen, a corresponding path of rotation is generated on the sphere.



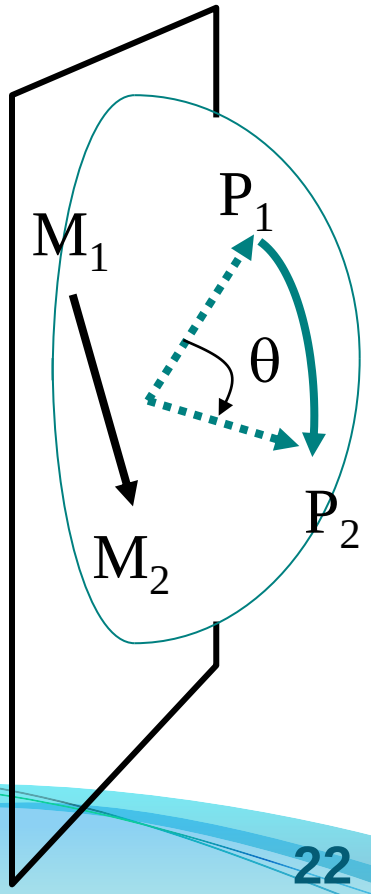
Virtual Trackball



- Let M_1M_2 be the path through which the mouse is dragged, and P_1 , P_2, the corresponding points on the virtual sphere.
- The angle of rotation is the angle between the vectors P_1 and P_2

$$\theta = \cos^{-1}(P_1 \cdot P_2)$$

- The axis of rotation is the axis perpendicular to both P_1 and P_2, given by $P_1 \times P_2$.



Matrix Stack

- A transformation matrix can be pushed into the matrix stack, saving it for later use.
- The top of stack represents the current transformation matrix
- The matrix stack is useful for applying different and independent sets of transformations to different objects.
- OpenGL:
  - `glPushMatrix()`: Create a copy of the current transformation matrix and push into the stack
  - `glPopMatrix()`: Remove the matrix at the top of stack

Matrix Stack: Example

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glPushMatrix();
   glColor3f(0., 1., 1.);
   glTranslatef(2.0, 1.5, 0.0);
   glutSolidCube(3.0);
   glPushMatrix();
         glColor3f(0., 0., 1.);
         glTranslatef(1.0, 1.8, 1.0);
         glRotatef(-45.0, 0., 1.0, 0.0);
         glutSolidTeapot(0.5);
   glPopMatrix();
   glPushMatrix();
         glColor3f(1., 0., 1.);
         glTranslatef(-1.0, 1.8, -1.0);
         glRotatef(30., 0., 0., 1.);
         glScalef(0.3, 0.3, 0.3);
         glutSolidDodecahedron();
   glPopMatrix();
glPopMatrix();
drawFloor();
```