# M-R Paradigm, Implementation of Affine Transformations & Quaternions

**CSE606: Computer Graphics**

**Jaya Sreevalsan Nair, IIIT Bangalore**

**February 03, 2025**

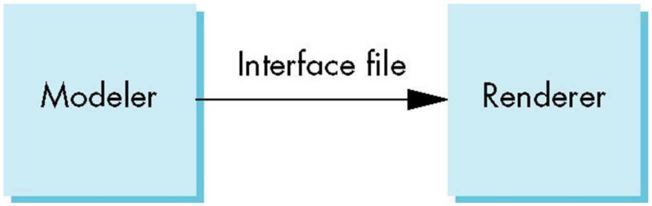
# Modeling-Rendering Paradigm

# Modularity in Graphics Programming

Two-step process: (a) Modeling objects, and (b) Rendering scene.

- Different software/hardware for the processes.
- Considered as modules.
- Separability of modules.
- Design of a scene using interactive graphics; rendering with light sources using high-performance cluster (intensive computation).
- e.g., Pixar’s Renderman.

Modeling-rendering Paradigm




**Example: Scene graph**

Image courtesy: Edward Angel

# Creating a Model

Various methods for model creation:

- Measurements/design plans
- Scanning
  - Scanning rooms: https://youtu.be/XA7FMoNAK9M and https://youtu.be/I_kG_kSYFUU
  - https://youtu.be/4GiLAOtjHNo Presidential Portrait
- Using modeling tools, e.g. Blender, CAD tools, etc.
- Reconstruction from images
- ...

# Recap - Ingredients in Graphics Programming

Graphics programming is the main part of the rendering module.

Ingredients:

1. Objects -- output of the modeling module, geometry (3D, complex, etc.)
2. A viewer (or camera)
3. Light sources (important for 3D realism)
4. Material properties (needed for rendering)

# Object Transformation

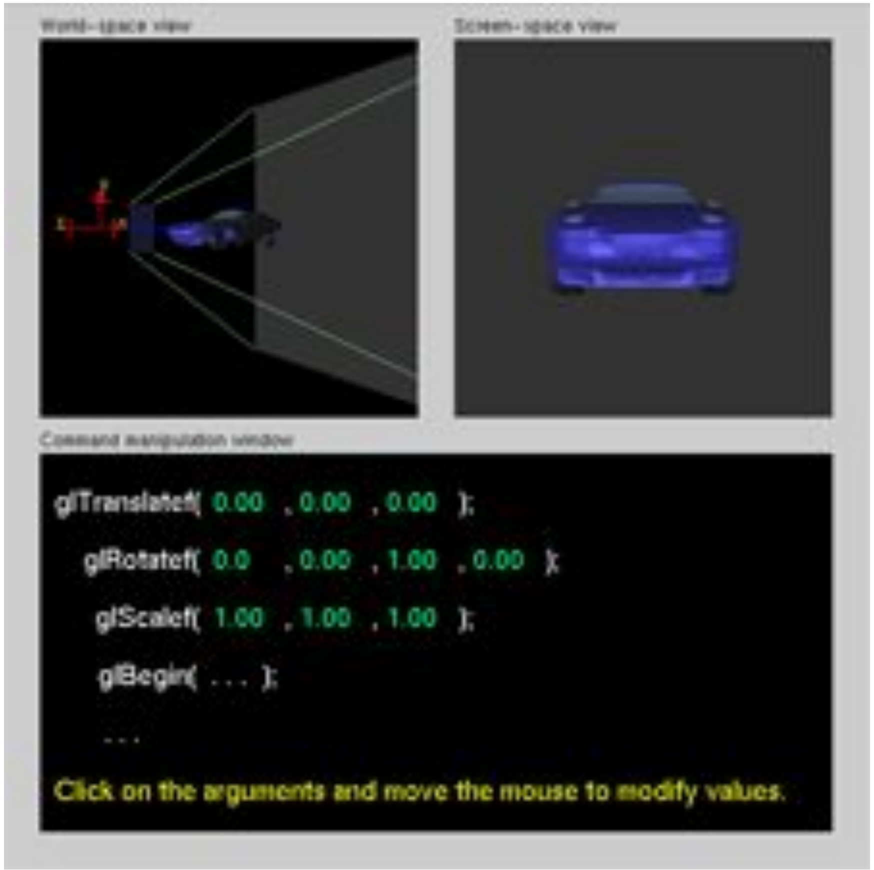





Image courtesy: Nate Robins

# Object Projection

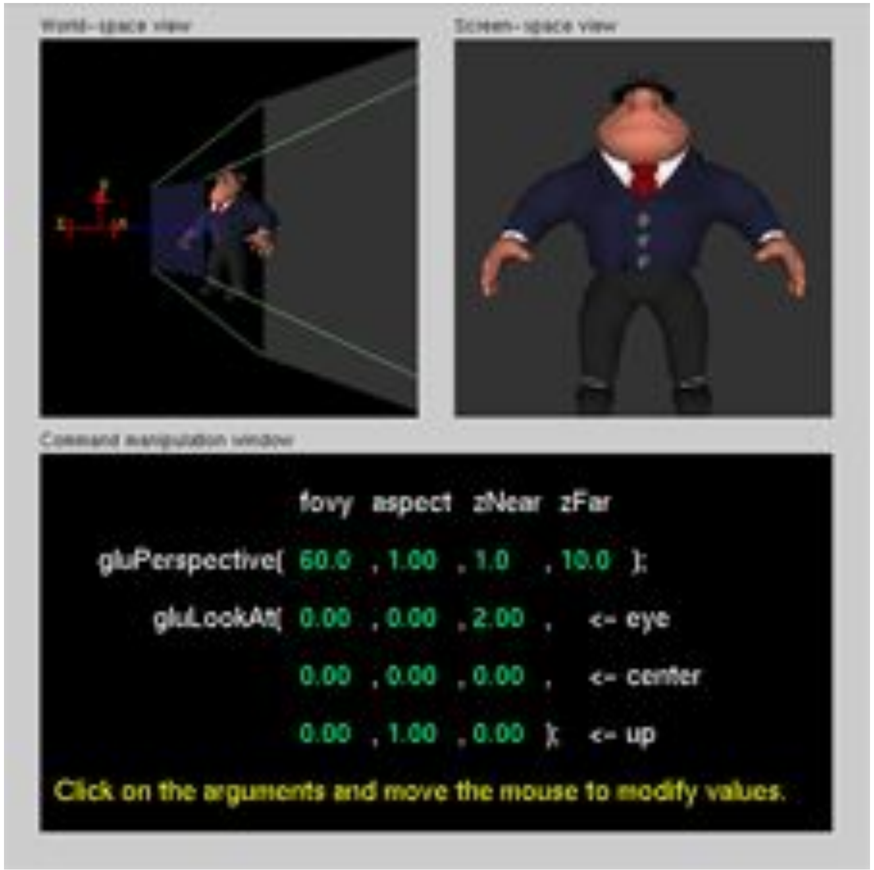



Image courtesy: Nate Robins

# Concatenation & Implementation of Affine Transformations

# Concatenation

Construction of a variety of affine transformations is done by multiplying sequences of basic transformations.

- **q**=CBA.**p**=C(B(A.**p**))
- To achieve pipeline transformation, use the associativity property of matrix multiplication, to get: **q** = (CBA).**p** = M.**p**
    - Optimizing computation, as M can be applied uniformly to multiple primitives.
    - The **order** of the transformations is preserved owing to the non-commutative property of matrix multiplication.

# Concatenation

Construction of a variety of affine transformations is done by multiplying sequences of basic transformations.

- **q**=CBA.**p**=C(B(A.**p**))
- To achieve pipeline transformation, use the associativity property of matrix multiplication, to get: **q** = (CBA).**p** = M.**p**
  - Optimizing computation, as M can be applied uniformly to multiple primitives.
  - The **order** of the transformations is preserved owing to the non-commutative property of matrix multiplication.

*Finding a concatenated matrix reduces to a single set of matrix-matrix multiplications followed by matrix-vector multiplications; unlike multiple matrix-vector multiplications at each vertex.*

- *This drastically reduces the "repeated" associative multiplications, thus reducing the overall number of computations.*

# Examples of Transformation Concatenation

1. Rotation about a fixed point: Move reference origin to the fixed point, rotate, move back to the original position of origin.
2. General rotation (of object centered at origin): Multiply rotation about x-, y-, and z-axes in the order it occurs.
3. Instance transformation: Translate origin to center of mass of object ($c_m$), scale, rotate, translate back.
4. Rotation about an arbitrary axis: Rotation about a direction vector **u** by an angle $\theta$.

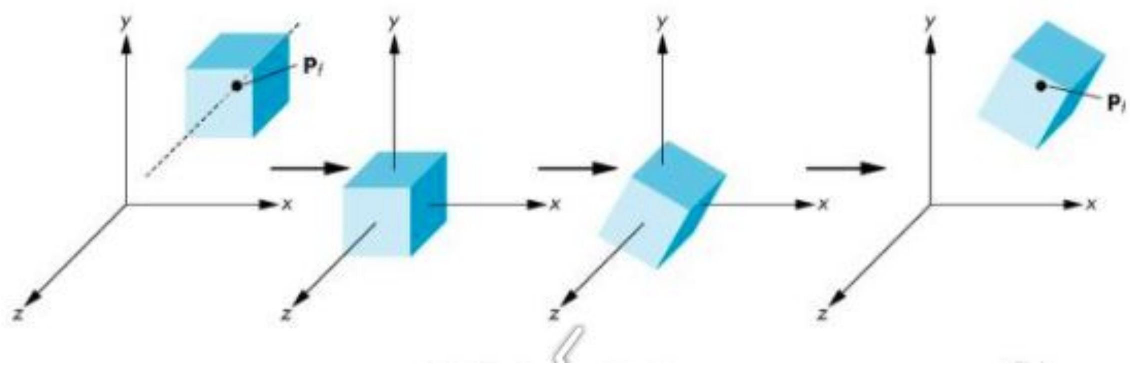
# Example1. Rotation about a Fixed Point

Move reference origin to fixed point, rotate, move back to the original position of origin.

$$M = T(p_f).\, R_z(\theta).\, T(-p_f)$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & x_f - x_f\cos\theta + y_f\sin\theta \\ \sin\theta & \cos\theta & 0 & y_f - x_f\sin\theta - y_f\cos\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Image Courtesy: Edward Angel



# Example2: General Rotation

For an object centered at origin, Multiply rotation about x-, y-, and z-axes in the order it occurs. The angles are called **Euler angles**.

$$R = R_x(\theta_x).\, R_y(\theta_y).\, R_z(\theta_z)$$

# Example3: Instance Transformation

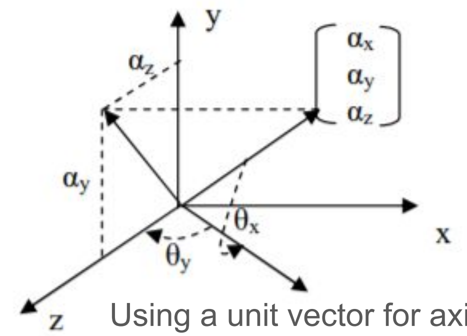
Translate origin to center of mass of object ($c_m$), scale, rotate, translate back.

$$M = T(c_m)RST(-c_m)$$

# Example4: Rotation about an Arbitrary Axis

Rotation about a direction vector **u** by an angle $\theta$.

1. Move the fixed point of (object) frame to origin of reference frame.
2. Use unit vector û from origin, and use rotations about x-axis and y-axis to align the vector along z-axis.
3. Rotate by $\theta$ about z-axis.
4. Rotate back about y- and x-axes.
5. Move back the fixed point.




Using a unit vector for axis

https://www.chegg.com/homework-help/questions-and-answers/computer-graphics-rotation-around-arbitrary-axis-decomposed-series-transformations-look-fo-q44460310

# Example4: Rotation about an Arbitrary Axis

Rotation about a direction vector **u** by an angle $\theta$.

1. Move the fixed point of (object) frame to origin of reference frame.
2. Use unit vector û from origin, and use rotations about x-axis and y-axis to align the vector along z-axis.
3. Rotate by $\theta$ about z-axis.
4. Rotate back about y- and x-axes.
5. Move back the fixed point.

$$\mathbf{M} = \mathbf{T}(p_0)\mathbf{R}_x(-\theta_x)\mathbf{R}_y(-\theta_y)\mathbf{R}_z(\theta)\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)\mathbf{T}(-p_0)$$

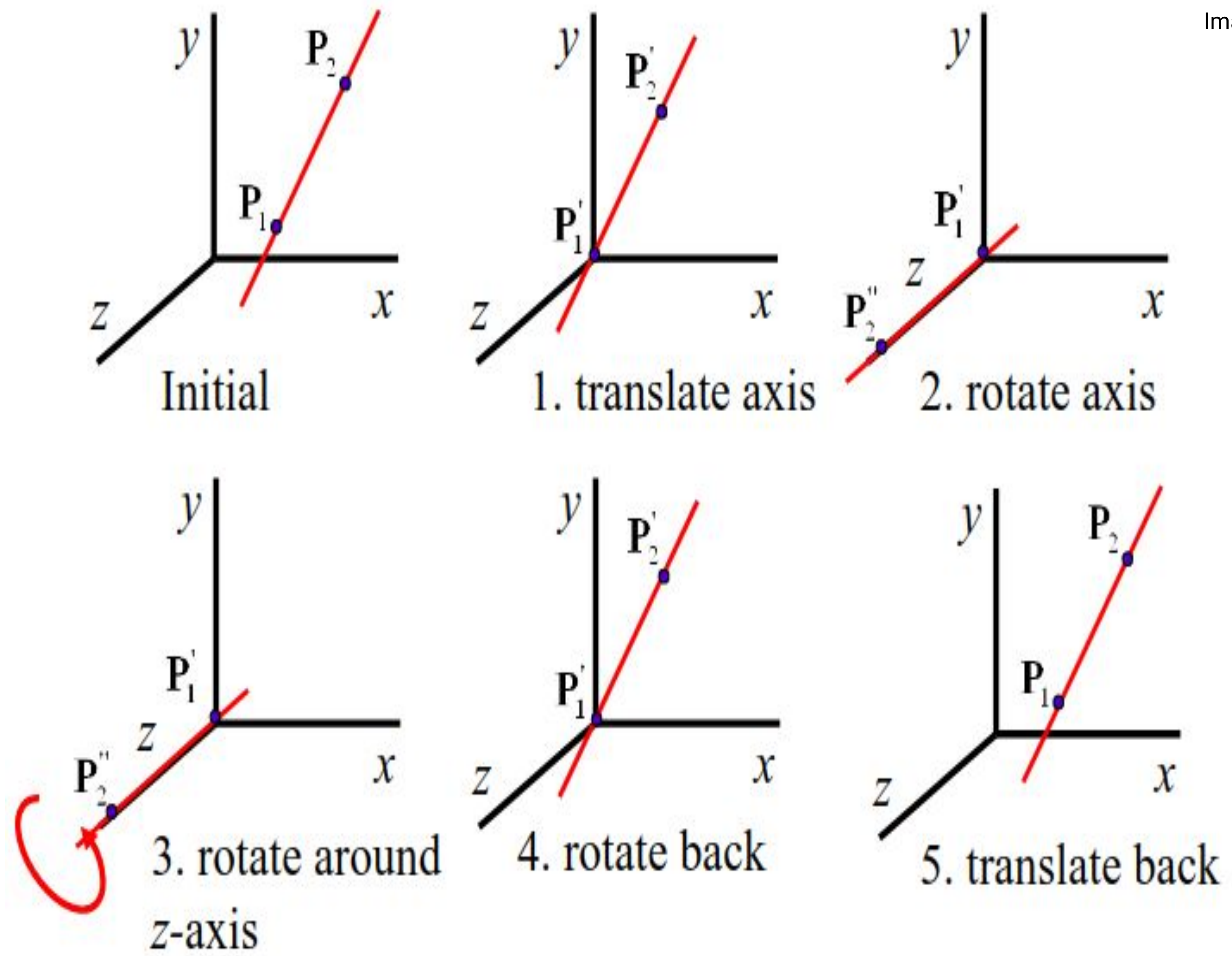
$$\hat{\mathbf{u}} = [\alpha_x \quad \alpha_y \quad \alpha_z]^T,$$

$$d = \sqrt{\alpha_y^2 + \alpha_z^2} \text{ and } \sqrt{d^2 + \alpha_x^2} = 1,$$

$$\theta_x = \cos^{-1}\left(\frac{\alpha_z}{d}\right) \text{ and } \theta_y = \cos^{-1}(d).$$

# Example4: Rotation about an Arbitrary Axis

Image courtesy: Jack van Wijk



# Explanation

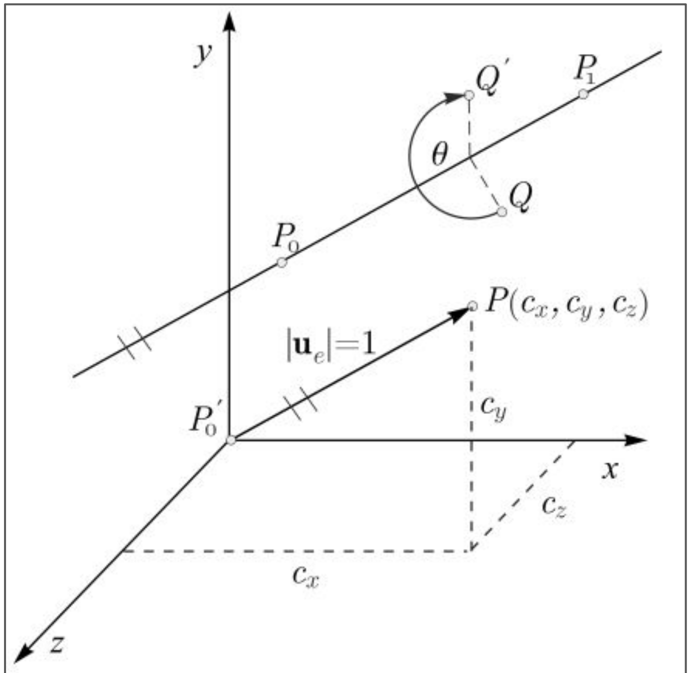



Figure 1: Rotation about an arbitrary axis

In Fig. 2 the direction cosines are satisfied the following equation:

$$c_x^2 + c_y^2 + c_z^2 = 1,$$
$$\cos\phi_x = c_x, \quad \cos\phi_y = c_y, \quad \cos\phi_z = c_z.$$

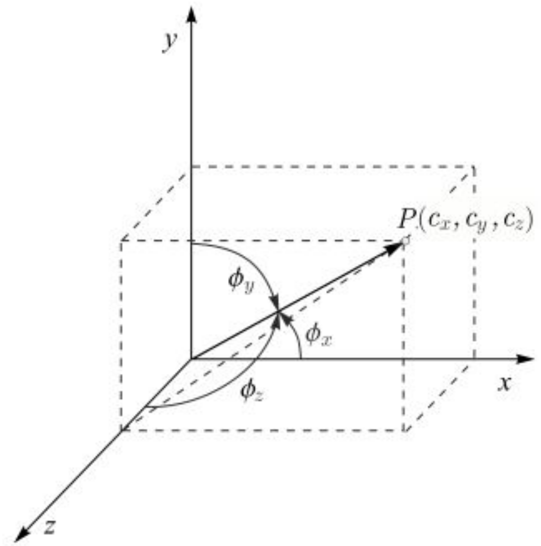



Figure 2: Direction cosines

Kovács, E., 2012, January. Rotation about an arbitrary axis and reflection through an arbitrary plane. In *Annales Mathematicae et Informaticae* (Vol. 40, pp. 175-186).

# Explanation

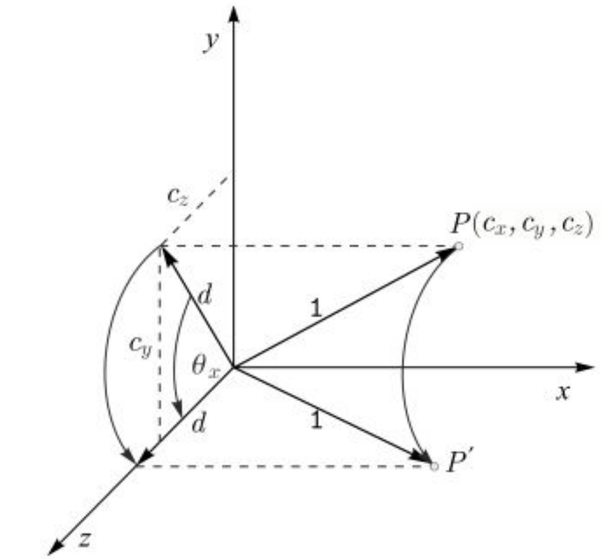



Figure 3: Rotation around $x$-axis

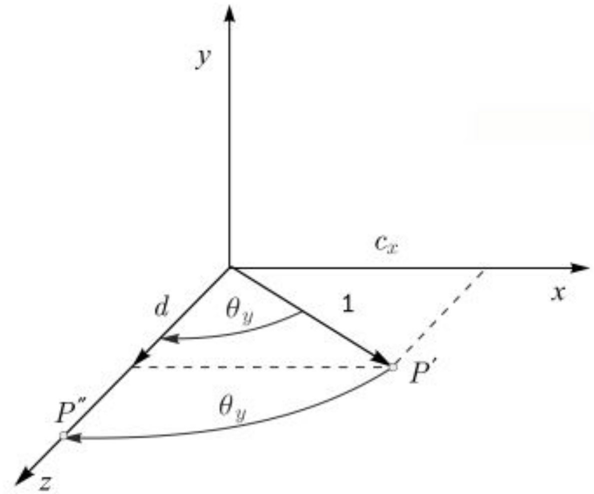
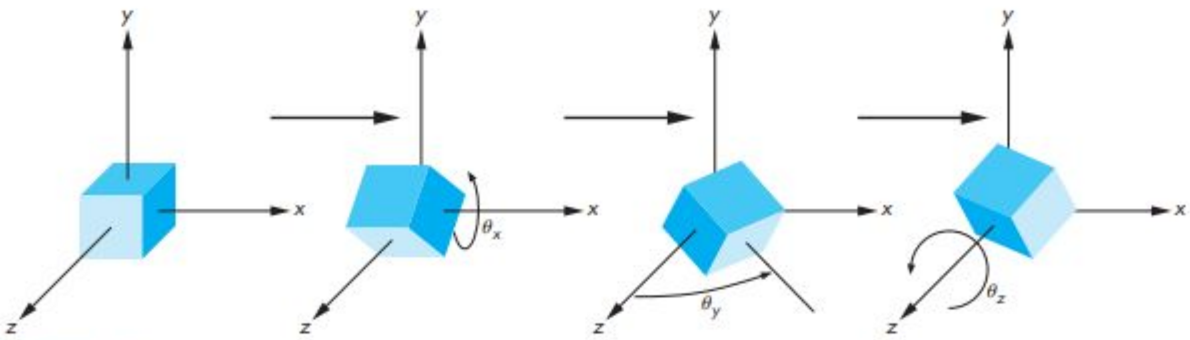



Figure 4: Rotation around $y$-axis





Kovács, E., 2012, January. Rotation about an arbitrary axis and reflection through an arbitrary plane. In *Annales Mathematicae et Informaticae* (Vol. 40, pp. 175-186).
Angel, Edward. Interactive computer graphics : a top-down approach with shader-based OpenGL / Edward Angel, David Shreiner. — 6th ed.

# Implementation of Affine Transformations

Method-1: Using mouse buttons for overall control of an object using widget (GLUT) API:

- To control x- and y- rotations – using left-right and up-down movements of the mouse with left button down.
- To control speed of rotation: use speed of motion of mouse, or distance from center of screen.
- To translate in x- and y- axes – using left-right and up-down movements of the mouse with right button down.
- To zoom in and out – using up-down movements of the mouse with middle button down – either as translation in z-axis or scaling up and down.

# Implementation of Affine Transformations

Method-2: Using virtual trackball -- using a 2D mouse for a 3D rotation:

- For a rotation from point $p_1$ to $p_2$ on the trackball ⇒ rotation about **n**, for **n** $= p_1 \times p_2$
- Angle of rotation $\theta$, such that $|\sin \theta| = |n|$ ⇒ $\sin \theta \cong \theta$.

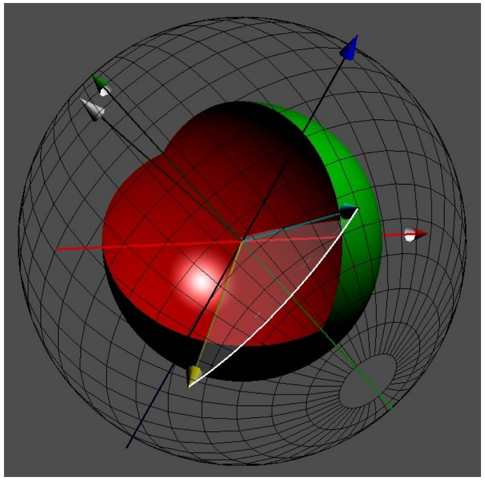


Image Courtesy: University of Copenhagen.

# 3D Transformations

# 3D Transformation Matrices: Scaling

Origin is the scaling-invariant (fixed) point.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} \beta_x x \\ \beta_y y \\ \beta_z z \\ 1 \end{bmatrix}$$

$$\Rightarrow S(\beta_x, \beta_y, \beta_z) = \begin{bmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow S^{-1}(\beta_x, \beta_y, \beta_z) = S(\frac{1}{\beta_x}, \frac{1}{\beta_y}, \frac{1}{\beta_z})$$

# 3D Transformation Matrices: Rotation

Transformation matrices for rotation about x-, y-, z-axes by an angle of $\theta_x$, $\theta_y$, $\theta_z$, respectively: $R_x(\theta_x)$, $R_y(\theta_y)$, $R_z(\theta_z)$.

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# 3D Transformation Matrices: Rotation

Inverse of any rotation matrix: $$R^{-1}(\theta) = R(-\theta)$$

Since $\cos(-\theta) = \cos\theta$ and $\sin(-\theta) = -\sin\theta$, we get: Inverse = Transpose.

Thus, a rotation transformation matrix is an **orthogonal** matrix.

- Computation of inverse of rotation matrices thus becomes easier.

# 3D Transformation Matrices: Concatenation of Rotations

To construct desired rotation (about any arbitrary axis): Define origin as the fixed point, and implement sequence of rotations, such that:

$$R = R_i \cdot R_j \cdot R_k \ldots$$

Even in the concatenation of rotations, the concatenated matrix is orthogonal.

$$R^{-1} = R^T$$

# 3D Transformation Matrices: Shear

Shearing in x-axis by angle $\theta$:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x + y\cot\theta \\ y \\ z \\ 1 \end{bmatrix}$$

$$\Rightarrow H_x(\theta) = \begin{bmatrix} 1 & \cot\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow H_x^{-1}(\theta) = H_x(-\theta)$$

# Video of the Day

Evolution of GPUs (2000-20, Nvidia Geforce)

https://youtu.be/8_-1FxpAiMQ

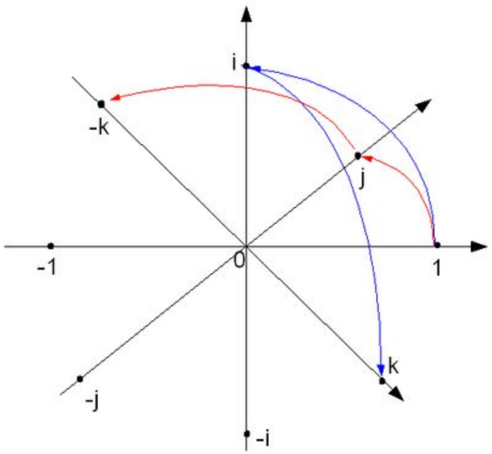
# Introduction to Quaternions

# Introduction to Quaternions

Quaternions: Extensions to complex numbers; lesser intuitive method for describing & manipulating rotations.

Highly recommended videos:

- https://www.youtube.com/watch?v=d4EgbgTm0Bg&t=280s
    - [What are quaternions, and how do you visualize them? A story of four dimensions. (2018) by 3Blue1Brown]
- https://www.youtube.com/watch?v=zjMuIxRvygQ
    - [Quaternions and 3d rotation, explained interactively. (2018) by 3Blue1Brown]




Graphical representation of quaternion units product as 90°-rotation in 4D-space

$ij = k$
$ji = -k$
$ij = -ji$

# Quaternions

Using complex numbers to describe rotations, using polar representations:

$$e^{i\theta} = \cos\theta + i\sin\theta$$

$$c = a + ib = \sqrt{a^2 + b^2}\, e^{i\theta}$$

To describe, we need axis of rotation **q** and angle of rotation ($q_0$); where quaternion ($q_0$,**q**) has:

$$\mathbf{q} = \begin{pmatrix} q_1 & q_2 & q_3 \end{pmatrix}$$

$$\mathbf{q} = q_1 \,.\, \mathbf{i} + q_2 \,.\, \mathbf{j} + q_3 \,.\, \mathbf{k}$$

Base rules:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k}$$

$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i}$$

$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j}$$

# Properties

Quaternion algebra is denoted as $\mathbb{H}$ in the honor of Sir William Rowan Hamilton, who invented it.

For $a = (q_0, \mathbf{q})$ and $b = (p_0, \mathbf{p})$,

$$a + b = (q_0 + p_0, \mathbf{q} + \mathbf{p})$$

$$ab = (q_0 p_0 - \mathbf{q} . \mathbf{p}, q_0 \mathbf{p} + p_0 \mathbf{q} + \mathbf{q} \times \mathbf{p})$$

$$|a| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = \sqrt{q_0^2 + \mathbf{q} . \mathbf{q}}$$

Multiplicative identity of quaternion $= (1, \mathbf{0})$

Multiplicative inverse,

$$a^{-1} = \frac{1}{|a|^2} . (q_0, -\mathbf{q}) = \frac{1}{|a|^2} . \bar{a}.$$

where $\bar{a}$ is the conjugate of $a$.

Nice property of conjugation:

$$\bar{pq} = \bar{q} . \bar{p}$$

# Summary

- Modeling-Rendering Paradigm
- Concatenation of transformations with examples
- Matrices for 3D transformations
- Introduction to quaternions