



## Transformations

CS 432 Interactive Computer Graphics  
Prof. David E. Breen  
Department of Computer Science

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

1



## Objectives

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

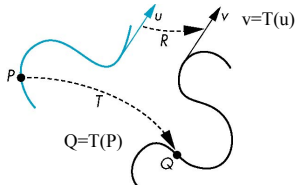
E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2



## General Transformations

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



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3



## Affine Transformations

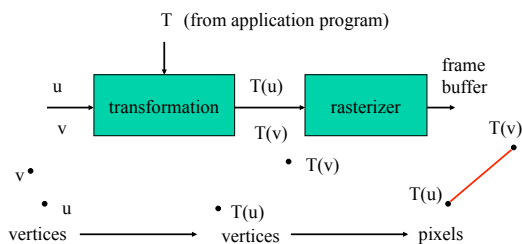
- Line preserving
- Characteristic of many physically important transformations
  - Rigid body transformations: rotation, translation
  - Scaling, shear
- Importance in graphics is that we need only transform endpoints of line segments and let implementation draw line segment between the transformed endpoints

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4



## Pipeline Implementation



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

5



## Notation

We will be working with both coordinate-free representations of transformations and representations within a particular frame

$P, Q, R$ : points in an affine space

$u, v, w$ : vectors in an affine space

$\alpha, \beta, \gamma$ : scalars

$p, q, r$ : representations of points


-array of 4 scalars in homogeneous coordinates

$u, v, w$ : representations of points

-array of 4 scalars in homogeneous coordinates

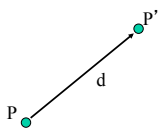
E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

6



## Translation


- Move (translate, displace) a point to a new location



- Displacement determined by a vector  $d$ 
  - Three degrees of freedom
  - $P' = P + d$

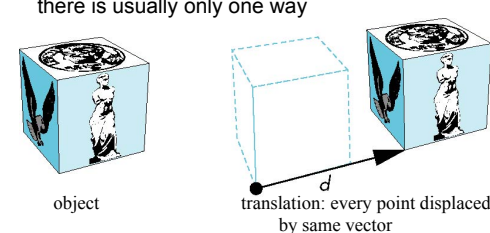
E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

7




## How many ways?

Although we can move a point to a new location in infinite ways, when we move many points there is usually only one way



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

8



## Translation Using Representations

Using the homogeneous coordinate representation in some frame

$$p = [x \ y \ z \ 1]^T$$

$$p' = [x' \ y' \ z' \ 1]^T$$

$$d = [dx \ dy \ dz \ 0]^T$$


Hence  $p' = p + d$  or

$$\begin{aligned} x' &= x + d_x \\ y' &= y + d_y \\ z' &= z + d_z \end{aligned}$$

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

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

9



## Translation Matrix

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


$p' = T p$  where

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

This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

10




## Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{aligned} x' &= x + d_x \\ y' &= y + d_y \\ z' &= z + d_z \end{aligned}$$

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

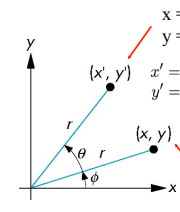
11



## Rotation (2D)

Consider rotation about the origin by  $\theta$  degrees

- radius stays the same, angle increases by  $\theta$



$$\begin{aligned} x &= r \cos(\phi) \\ y &= r \sin(\phi) \end{aligned}$$

$$\begin{aligned} x' &= r \cos(\theta + \phi) = r \cos \phi \cos \theta - r \sin \phi \sin \theta \\ y' &= r \sin(\theta + \phi) = r \cos \phi \sin \theta + r \sin \phi \cos \theta \end{aligned}$$

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

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

12



## 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

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

- or in homogeneous coordinates

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

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

1  
3



## Rotation Matrix

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

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

1  
4



## Rotation Matrix

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

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

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

1  
5



## 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

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

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

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

1  
6



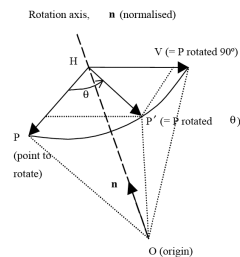
## Rotation Around an Arbitrary Axis

- Rotate a point P around axis  $\mathbf{n}(x,y,z)$  by angle  $\theta$

$$\mathbf{R} = \begin{bmatrix} tx^2 + c & txy + sz & txz - sy & 0 \\ txy - sz & ty^2 + c & tyz + sx & 0 \\ txz + sy & tyz - sx & tz^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $c = \cos(\theta)$
- $s = \sin(\theta)$
- $t = (1 - c)$

Graphics Gems I, p. 466 & 498



1  
7




## Rotation Around an Arbitrary Axis

- Also can be expressed as the Rodrigues Formula

$$\mathbf{P}_{rot} = \mathbf{P} \cos(\theta) + (\mathbf{n} \times \mathbf{P}) \sin(\theta) + \mathbf{n}(\mathbf{n} \cdot \mathbf{P})(1 - \cos(\theta))$$

1  
8

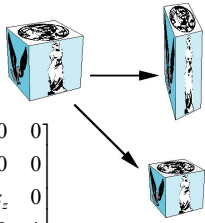


## Scaling

---

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


$$\begin{aligned} x' &= s_x x \\ y' &= s_y y \\ z' &= s_z z \\ \mathbf{p}' &= \mathbf{S} \mathbf{p} \end{aligned}$$



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

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

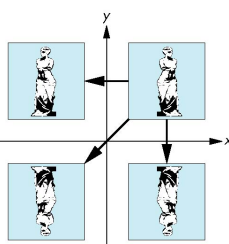
1  
9



## Reflection

---

corresponds to negative scale factors



original


$s_x = -1 \ s_y = 1$

$s_x = -1 \ s_y = -1$

$s_x = 1 \ s_y = -1$

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2  
0




## Inverses

---

- Although we could compute inverse matrices by general formulas, we can use simple geometric observations
  - Translation:  $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
  - Rotation:  $\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$ 
    - Holds for any rotation matrix
    - Note that since  $\cos(-\theta) = \cos(\theta)$  and  $\sin(-\theta) = -\sin(\theta)$
  - Scaling:  $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2  
1




## Concatenation

---

- We can form arbitrary affine transformation matrices by multiplying together rotation, translation, and scaling matrices
- Because the same transformation is applied to many vertices, the cost of forming a matrix  $\mathbf{M} = \mathbf{ABCD}$  is not significant compared to the cost of computing  $\mathbf{M}\mathbf{p}$  for many vertices  $\mathbf{p}$
- The difficult part is how to form a desired transformation from the specifications in the application

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2  
2




## Order of Transformations

---

- Note that matrix on the right is the first applied
- Mathematically, the following are equivalent
 
$$\mathbf{p}' = \mathbf{ABCp} = \mathbf{A}(\mathbf{B}(\mathbf{Cp})) \quad // \text{pre-multiply}$$
- Note many references use column matrices to represent points. In terms of column matrices
 
$$\mathbf{p}'^T = \mathbf{p}^T \mathbf{C}^T \mathbf{B}^T \mathbf{A}^T \quad // \text{post-multiply}$$

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

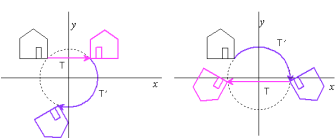
2  
3



## Properties of Transformation Matrices

---

- Note that matrix multiplication is not commutative
- i.e. in general  $\mathbf{M}_1 \mathbf{M}_2 \neq \mathbf{M}_2 \mathbf{M}_1$



- $\mathbf{T}$  – reflection around y axis
- $\mathbf{T}'$  – rotation in the plane

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2  
4



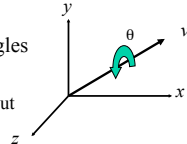
## General Rotation About the Origin

A rotation by  $\theta$  about an arbitrary axis can be decomposed into the concatenation of rotations about the  $x$ ,  $y$ , and  $z$  axes

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

$\theta_x, \theta_y, \theta_z$  are called the Euler angles

Note that rotations do not commute. We can use rotations in another order but with different angles.



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2  
5



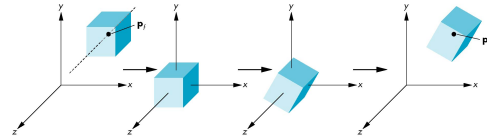
## Rotation About a Fixed Point other than the Origin

Move fixed point to origin

Rotate

Move fixed point back

$$\mathbf{M} = \mathbf{T}(p_f) \mathbf{R}(\theta) \mathbf{T}(-p_f)$$



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

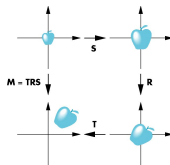
2  
6



## 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



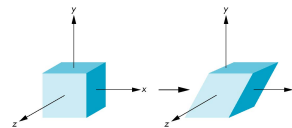
E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2  
7



## Shear

- Helpful to add one more basic transformation
- Equivalent to pulling faces in opposite directions



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2  
8

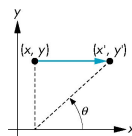


## Shear Matrix

Consider simple shear along  $x$  axis

$$\begin{aligned} x' &= x + y \cot \theta \\ y' &= y \\ z' &= z \end{aligned}$$

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



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

2  
9



## OpenGL Transformations

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3  
0



## Objectives

- Learn how to carry out transformations in OpenGL
  - Rotation
  - Translation
  - Scaling
- Introduce mat.h and vec.h transformations
  - Model-view
  - Projection

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3  
1



## Pre 3.1 OpenGL Matrices

- In OpenGL matrices were part of the state
- Multiple types
  - Model-View (`GL_MODELVIEW`)
  - Projection (`GL_PROJECTION`)
  - Texture (`GL_TEXTURE`)
  - Color (`GL_COLOR`)
- Single set of functions for manipulation
- Select which to manipulated by
  - `glMatrixMode (GL_MODELVIEW) ;`
  - `glMatrixMode (GL_PROJECTION) ;`

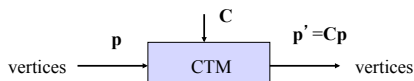
E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3  
2



## Current Transformation Matrix (CTM)

- Conceptually there was a 4 x 4 homogeneous coordinate matrix, the *current transformation matrix* (CTM) that is part of the state and is applied to all vertices that pass down the pipeline
- The CTM was defined in the user program and loaded into a transformation unit



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3  
3



## Rotation about a Fixed Point

Start with identity matrix:  $C \leftarrow I$   
 Move fixed point to origin:  $C \leftarrow CT$   
 Rotate:  $C \leftarrow CR$   
 Move fixed point back:  $C \leftarrow CT^{-1}$

Result:  $C = TRT^{-1}$  which is **backwards**.

This result is a consequence of doing postmultiplications. Let's try again.

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3  
5



## Reversing the Order

We want  $C = T^{-1} R T$  so we must do the operations in the following order

$C \leftarrow I$   
 $C \leftarrow CT^{-1}$   
 $C \leftarrow CR$   
 $C \leftarrow CT$

Note that the last operation specified is the first executed in the program

Recall  $p' = ABCp = A(B(Cp))$  // pre-multiply

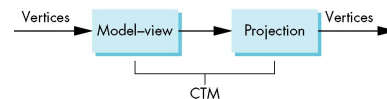
E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3  
6



## CTM in OpenGL

- OpenGL had a model-view and a projection matrix in the pipeline which were concatenated together to form the CTM
- We will emulate this process



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3  
7



## vec.h and mat.h

Define basic types

- `vec2`, `vec3`, `vec4`
  - 2, 3 & 4 element arrays
- `mat2`, `mat3`, `mat4`
  - 2x2, 3x3 & 4x4 matrices
- Overloaded operators
  - `+`, `-`, `*`, `/`, `+=`, `-=`, `*=`, `/=`, `[]`
- Generators
  - `RotateX`, `RotateY`, `RotateZ`, `Translate`, `Scale`
  - `Ortho`, `Ortho2D`, `Frustum`, `Perspective`, `LookAt`

3



## Rotation, Translation, Scaling

Create an identity matrix:

```
mat4 m = identity();
```

Multiply on right by rotation matrix of `theta` in degrees where (`vx`, `vy`, `vz`) define axis of rotation

```
mat4 r = Rotate(theta, vx, vy, vz);
m = m*r;
```

Do same with translation and scaling:

```
mat4 s = Scale(sx, sy, sz);
mat4 t = Translate(dx, dy, dz);
m = m*s*t;
```

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

3



## Example

- Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

```
mat4 m = identity();
m = Translate(1.0, 2.0, 3.0) *
    Rotate(30.0, 0.0, 0.0, 1.0) *
    Translate(-1.0, -2.0, -3.0);
```

- Remember that last matrix specified in the program is the first applied

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4



## Arbitrary Matrices

- Can load and multiply by matrices defined in the application program
- Matrices are stored as one dimensional array of 16 elements which are the components of the desired 4 x 4 matrix stored by columns
- OpenGL functions that have matrices as parameters allow the application to send the matrix or its transpose

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4



## Matrix Stacks

- In many situations we want to save transformation matrices for use later
  - Traversing hierarchical data structures (Chapter 8)
  - Avoiding state changes when executing display lists
- Pre 3.1 OpenGL maintained stacks for each type of matrix
- Easy to create the same functionality with a simple stack class

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4



## Using Transformations

- Example: use `idle` function to rotate a cube and `mouse` function to change direction of rotation
- Start with a program that draws a cube in a standard way
  - Centered at origin
  - Sides aligned with axes
  - Will discuss modeling in next lecture

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4



## main.c

```

void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
        GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("colorcube");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutIdleFunc(spinCube);
    glutMouseFunc(mouse);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}

```

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4  
5



## Idle and Mouse callbacks

```

void spinCube()
{
    theta[axis] += 2.0;
    if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
    glutPostRedisplay();
}

void mouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        axis = 0;
    if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
        axis = 1;
    if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        axis = 2;
}

```

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4  
6



## Display callback

We can form matrix in application and send to shader and let shader do the rotation or we can send the angle and axis to the shader and let the shader form the transformation matrix and then do the rotation

More efficient than transforming data in application and resending the data

```

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glUniform(...); //or glUniformMatrix
    glDrawArrays(...);
    glutSwapBuffers();
}

```

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4  
7



## Using the Model-view Matrix

- In OpenGL the model-view matrix is used to
  - Position the camera
    - Can be done by rotations and translations but is often easier to use a LookAt function
  - Build models of objects
- The projection matrix is used to define the view volume and to select a camera lens
- Although these matrices are no longer part of the OpenGL state, it is usually a good strategy to create them in our own applications

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4  
8



## Smooth Rotation

- From a practical standpoint, we often want to use transformations to move and reorient an object smoothly
  - Problem: find a sequence of model-view matrices  $M_0, M_1, \dots, M_n$  so that when they are applied successively to one or more objects we see a smooth transition
- For orientating an object, we can use the fact that every rotation corresponds to part of a great circle on a sphere
  - Find the axis of rotation and angle
  - Virtual trackball (see text)

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

4  
9



## Incremental Rotation

- Consider the two approaches
  - For a sequence of rotation matrices  $R_0, R_1, \dots, R_n$ , find the Euler angles for each and use  $R_i = R_{iz} R_{iy} R_{ix}$ 
    - Not very efficient
  - Use the final positions to determine the axis and angle of rotation, then increment only the angle
- Quaternions can be more efficient than either

E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

5  
0





## Quaternions

- Extension of imaginary numbers from two to three dimensions
- Requires one real and three imaginary components **i, j, k**

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

- Quaternions can express rotations on sphere smoothly and efficiently. Process:
  - Model-view matrix  $\rightarrow$  quaternion
  - Carry out operations with quaternions
  - Quaternion  $\rightarrow$  Model-view matrix