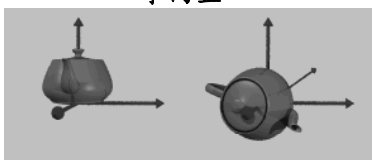
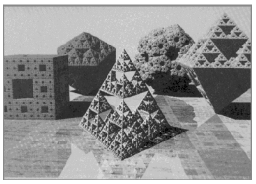
Chapter 4 Geometric Transformations

李同益



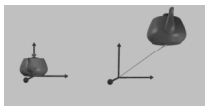
Modeling Transform

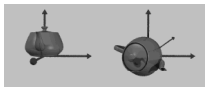
- **Specify transformation for objects**
  - Allow definitions of objects in own coordinate systems
  - Allow use of object definition multiple times in a scene

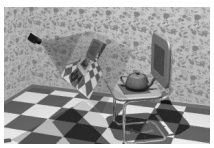


Overview

- 2D transformations
  - Basic 2-D transformations
  - Matrix representation
  - Matrix composition
- 3D transformations
  - Basic 3-D transformation
  - Same as 2-D
- Transformation Hierarchies
  - Scene graphs

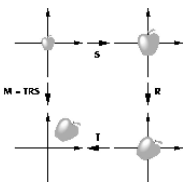




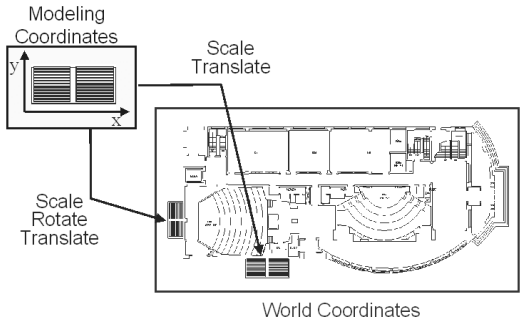


Instancing

- 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

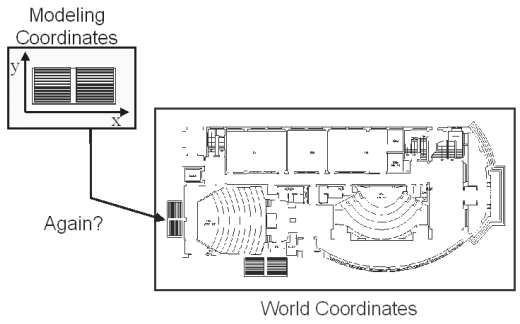


2-D Transformations



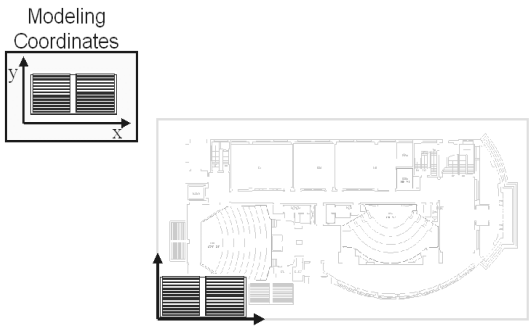
5

2-D Transformations



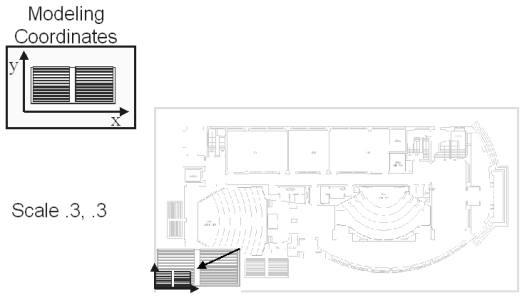
6

2-D Transformations



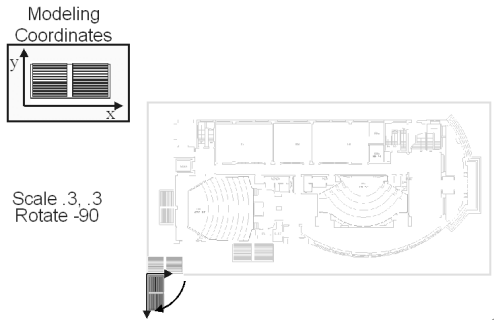
7

2-D Transformations



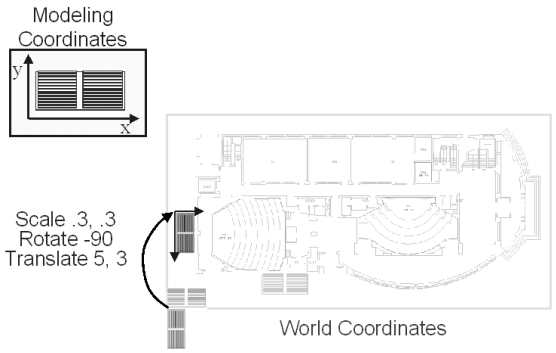
8

2-D Transformations



9

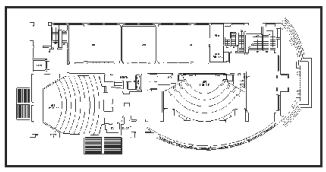
2-D Transformations



10

Basic 2D Transformations

- Translation:
  - x' = x + tx
  - y' = y + ty
- Scale:
  - x' = x * sx
  - y' = y * sy
- Shear:
  - x' = x + hx*y
  - y' = y + hy*x
- Rotation:
  - x' = x*cosΘ - y*sinΘ
  - y' = x*sinΘ + y*cosΘ

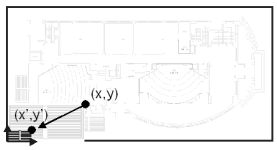


Transformations
can be combined
(with simple algebra)

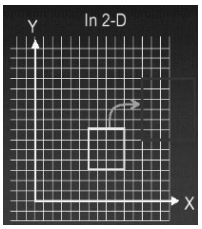
11

Basic 2D Transformations

- Translation:
  - x' = x + tx
  - y' = y + ty
- Scale:
  - x' = x * sx
  - y' = y * sy
- Shear:
  - x' = x + hx*y
  - y' = y + hy*x
- Rotation:
  - x' = x*cosΘ - y*sinΘ
  - y' = x*sinΘ + y*cosΘ

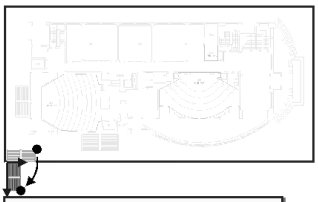


x' = x*sx
y' = y*sy



Basic 2D Transformations

- Translation:
  - x' = x + tx
  - y' = y + ty
- Scale:
  - x' = x * sx
  - y' = y * sy
- Shear:
  - x' = x + hx*y
  - y' = y + hy*x
- Rotation:
  - x' = x*cosΘ - y*sinΘ
  - y' = x*sinΘ + y*cosΘ

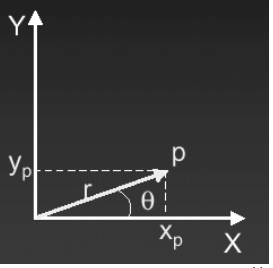


x' = (x*sx)*cosΘ – (y*sy)*sinΘ
y' = (x*sx)*sinΘ + (y*sy)*cosΘ

13

Rotation around the origin (2-D)

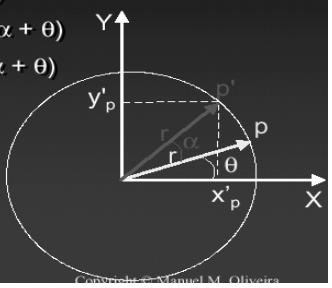
- $x_p = r \cos(\theta)$
- $y_p = r \sin(\theta)$



14

Rotation around the origin (2-D)

- $x_p = r \cos(\theta)$
- $y_p = r \sin(\theta)$
- $x'_p = r \cos(\alpha + \theta)$
- $y'_p = r \sin(\alpha + \theta)$



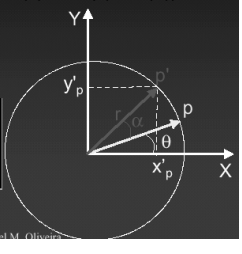
CSE528, Fall 2000 Copyright © Manuel M. Oliveira

15

Rotation around the origin (2-D)

- $x'_p = r \cos(\alpha + \theta) = r (\cos(\alpha) \cos(\theta) - \sin(\alpha) \sin(\theta))$
- $y'_p = r \sin(\alpha + \theta) = r(\sin(\alpha) \cos(\theta) + \sin(\theta) \cos(\alpha))$
- $x'_p = x_p \cos(\alpha) - y_p \sin(\alpha)$
- $y'_p = x_p \sin(\alpha) + y_p \cos(\alpha)$

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



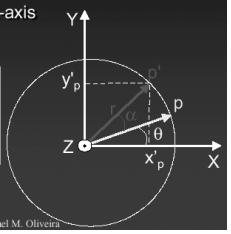
CSE528, Fall 2000 Copyright © Manuel M. Oliveira

16

Rotation (3-D)

- Rotations around the principal axis in 3-D can be derived from rotations in 2-D
- Example: rotation around Z-axis

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



Rotation (3-D)

- Rotation around X-axis

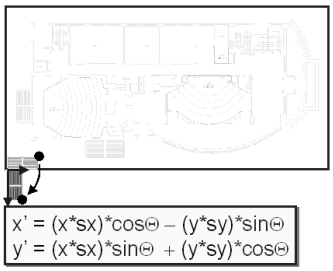
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Rotation around Y-axis

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

Basic 2D Transformations

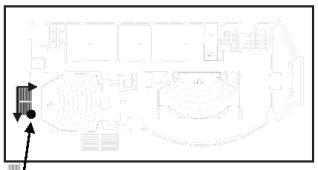
- Translation:
  - x' = x + tx
  - y' = y + ty
- Scale:
  - x' = x * sx
  - y' = y * sy
- Shear:
  - x' = x + hx*y
  - y' = y + hy*x
- Rotation:
  - x' = x*cosΘ - y*sinΘ
  - y' = x*sinΘ + y*cosΘ



x' = (x*sx)*cosΘ – (y*sy)*sinΘ
y' = (x*sx)*sinΘ + (y*sy)*cosΘ

Basic 2D Transformations

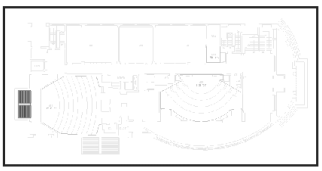
- Translation:
  - x' = x + tx
  - y' = y + ty
- Scale:
  - x' = x * sx
  - y' = y * sy
- Shear:
  - x' = x + hx*y
  - y' = y + hy*x
- Rotation:
  - x' = x*cosΘ - y*sinΘ
  - y' = x*sinΘ + y*cosΘ



x' = ((x*sx)*cosΘ – (y*sy)*sinΘ) + tx
y' = ((x*sx)*sinΘ + (y*sy)*cosΘ) + ty

Basic 2D Transformations

- Translation:
  - x' = x + tx
  - y' = y + ty
- Scale:
  - x' = x * sx
  - y' = y * sy
- Shear:
  - x' = x + hx*y
  - y' = y + hy*x
- Rotation:
  - x' = x*cosΘ - y*sinΘ
  - y' = x*sinΘ + y*cosΘ



x' = ((x*sx)*cosΘ – (y*sy)*sinΘ) + tx
y' = ((x*sx)*sinΘ + (y*sy)*cosΘ) + ty

21

Matrix Representation

- We can represent a 2D transformation by a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- Multiplying a matrix by a column vector corresponds to applying the transformation to a point

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned}$$

22

Matrix Representation

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Matrices are a convenient and efficient way to represent a sequence of transformations

23

2x2 Matrix

- What types of transformations can be represented with a 2x2 matrix?

2D Identity?

$$\begin{aligned} x' &= x \\ y' &= y \end{aligned} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Scale around (0,0)?

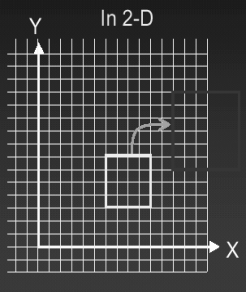
$$\begin{aligned} x' &= sx * x \\ y' &= sy * y \end{aligned} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

24

Scaling

- $x' = S_x x$
- $y' = S_y y$
- $z' = S_z z$

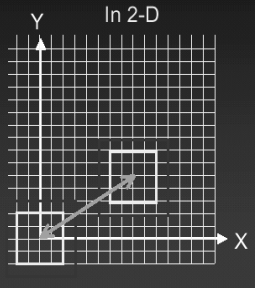
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



25

Scaling Around A Point

- Translate the point to the origin, then scale and translate the origin back to the point



26

2x2 Matrix

- What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

$$x' = \cos\Theta * x - \sin\Theta * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = \sin\Theta * x + \cos\Theta * y$$

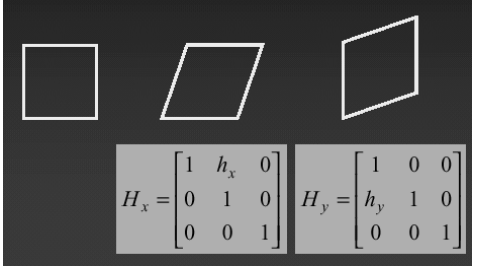
2D Shear?

$$x' = x + shx * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & shx \\ shy & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = shy * x + y$$

27

Shear (2-D)

- $x' = x + h_x y$
- $y' = y + h_y x$



$$H_x = \begin{bmatrix} 1 & h_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad H_y = \begin{bmatrix} 1 & 0 & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

28

Shear (3-D)

- The same idea, but with more degrees of freedom
- Most interesting case: shear of x and y along z
- $x' = x + h_x z$
- $y' = y + h_y z$

$$H_{xy} = \begin{bmatrix} 1 & 0 & h_x & 0 \\ 0 & 1 & h_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2x2 Matrix

- What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

$$x' = \cos\Theta * x - \sin\Theta * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = \sin\Theta * x + \cos\Theta * y$$

2D Shear?

$$x' = x + shx * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & shx \\ shy & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = shy * x + y$$

2x2 Matrix

- What types of transformations can be represented with a 2x2 matrix?

2D Mirror over Y axis?

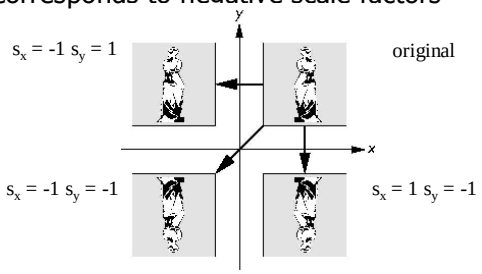
$$x' = -x \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = y$$

2D Mirror over (0,0)?

$$x' = -x \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = -y$$

Reflection (mirror)

corresponds to negative scale factors



2x2 Matrix

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

$$x' = x + tx$$
$$y' = y + ty$$

NO!

Only linear 2D transformations can be represented with a 2x2 matrix

33

2D Translation

- 2D translation can be represented by a 3x3 matrix
  - Point represented with homogeneous coordinates

$$x' = x + tx$$
$$y' = y + ty$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

34

Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

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

Rotate

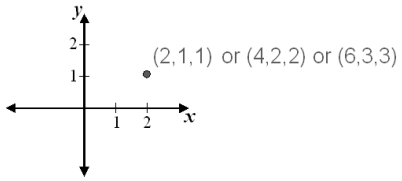
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

35

Homogeneous Coordinates

- Add a 3rd coordinate to every 2D point
  - (x, y, w) represents a point at location (x/w, y/w)
  - (x, y, 0) represents a point at infinity
  - (0, 0, 0) is not allowed



Convenient coordinate system to represent many useful transformations

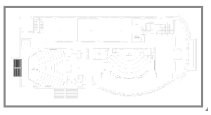
36

Matrix Composition

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

p' = T(tx,ty) R(Θ) S(sx,sy) **p**

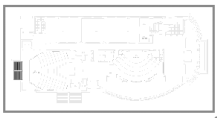


37

Matrix Composition

- Matrices are a convenient and efficient way to represent a sequence of transformations
  - General purpose representation
  - Hardware matrix multiply
  - Efficiency with premultiplication
    - » Matrix multiplication is associative

p' = (T * (R * (S***p**)))
p' = (T*R*S) * **p**



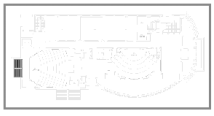
38

Matrix Composition

- Be aware: order of transformations matters
  - » Matrix multiplication is not commutative

p' = T * R * S * **p**

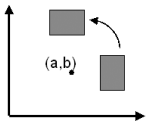
"Global" "Local"



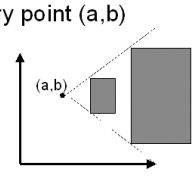
39

Matrix Composition

- Rotate by Θ around arbitrary point (a,b)
  - M=T(-a,-b) * R(Θ) * T(a,b)



- Scale by sx,sy around arbitrary point (a,b)
  - M=T(-a,-b) * S(sx,sy) * T(a,b)



40

3D Transformations

- Same idea as 2D transformations
  - Homogeneous coordinates: (x,y,z,w)
  - 4x4 transformation matrices

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

41

Basic 3D Transformations

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

Identity

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Translation

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

Mirror over X axis

42

Basic 3D Transformations

Rotate around Z axis:

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

Rotate around Y axis:

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

Rotate around X axis:

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

43

Rotation About a Fixed Point other than the Origin

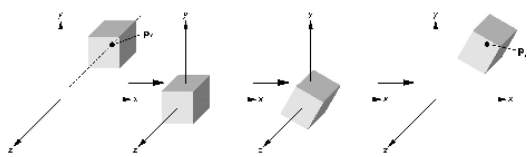
Move fixed point to origin

Rotate

Move fixed point back

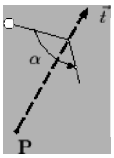
$\mathbf{M} = \mathbf{PT}(-p_f)\,\mathbf{R}(\theta)\,\mathbf{T}(p_f)$ (i.e.,P is a row major)

$\mathbf{M} = \mathbf{T}(p_f)\,\mathbf{R}(\theta)\,\mathbf{T}(-p_f)\mathbf{P}$ (i.e, P is a column major)



GENERAL ROTATION ABOUT ANAXIS

An axis in space is specified by a point **P** and a vector direction t
Suppose that we wish to rotate an object about this arbitrary axis.



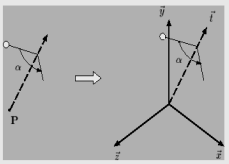
45

Developing the General Rotation Matrix

First assume that the axis of rotation can be specified in terms of Cartesian coordinates, i.e. can be represented by the point $\mathbf{P} = (x_p, y_p, z_p)$ and the vector $\vec{t} = < x_t, y_t, z_t >$. Then a rotation of α degrees about this axis can be defined by concatenating the following transformations

- Translate so that the point $\mathbf{P}$ moves to the origin

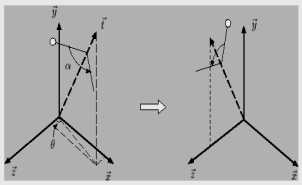
$$T_{(-x_p,-y_p,-z_p)}$$,



46

Developing the General Rotation Matrix

- Use the elementary rotation transformations to rotate the vector until it coincides with one of the coordinate axes. To do this, first rotate the vector until it is in the yz plane by using a rotation $R_{y,-\theta}$ about the y axis.

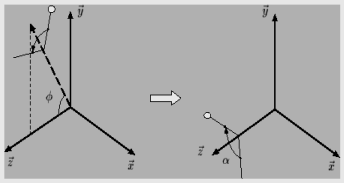


where $\theta = \arctan\left(\frac{x_t}{z_t}\right)$, and then use an x-axis rotation, of $R_{x,\phi}$, to rotate the vector until it coincides with the z axis.

47

Developing the General Rotation Matrix

where $\theta = \arctan\left(\frac{x_t}{z_t}\right)$, and then use an x-axis rotation, of $R_{x,\phi}$, to rotate the vector until it coincides with the z axis

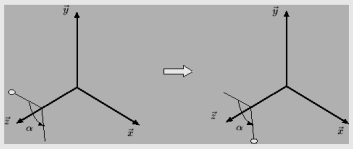


where $\phi = \arctan\left(\frac{y_v}{\sqrt{x_v^2+z_v^2}}\right)$

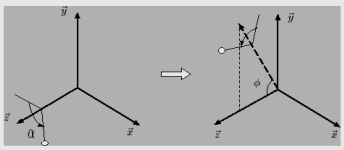
48

Developing the General Rotation Matrix

- Then use an α rotation about the z axis, $R_{z;\alpha}$.

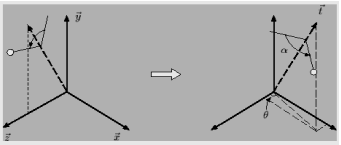


- Use rotations and translations to reverse the first two processes: First by a rotation $R_{x;-\phi}$, about the x axis

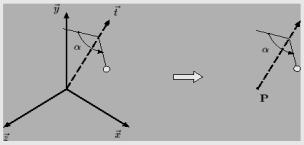


then by a rotation $R_{y;\theta}$ about the y axis

Developing the General Rotation Matrix



and finally using the translation, $T_{(x_p,y_p,z_p)}$ to translate back to the original axis.

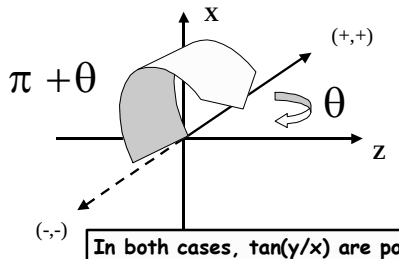


The matrix representation of the general rotation is given by the product of the above transformations.

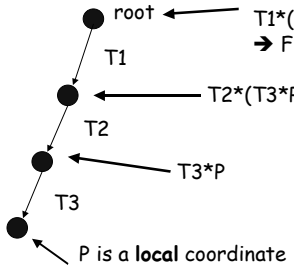
$$T_{(-x_p,-y_p,-z_p)}R_{y;-\theta}R_{x;\phi}R_{z;\alpha}R_{x;-\phi}R_{y;\theta}T_{(x_p,y_p,z_p)}$$

Developing the General Rotation Matrix

- Be careful .....



In both cases, tan(y/x) are positive.
So, we need to carefully choose
it by checking the signs of x and y



T1*(T2(T3*P))
➔ Final **global** coord. of P

T2*(T3*P)

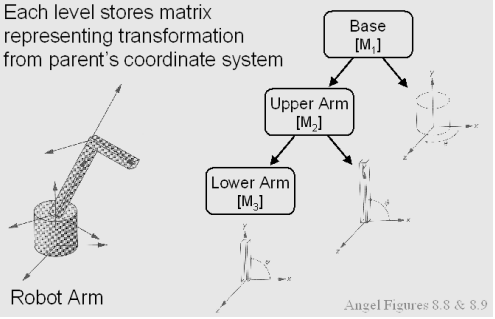
T3*P

```
Transform
# (T1)
{
 Transform {
  # (T2)

    Transform {
     # (T3)
                }
   }
}
```

Transformation Hierarchies

- Build scene with hierarchy of coordinate systems
  - Each level stores matrix representing transformation from parent's coordinate system



53

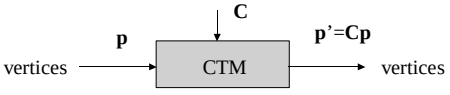
OpenGL Matrices

- In OpenGL matrices are part of the state
- Three types
  - Model-View (**GL_MODEL_VIEW**)
  - Projection (**GL_PROJECTION**)
  - Texture (**GL_TEXTURE**) (ignore for now)
- Single set of functions for manipulation
- Select which to manipulated by
  - `glMatrixMode(GL_MODEL_VIEW);`
  - `glMatrixMode(GL_PROJECTION);`

54

Current Transformation Matrix (CTM)

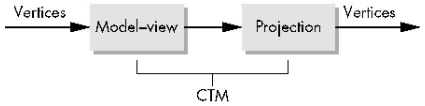
- Conceptually there is a 4 x 4 homogeneous coordinate matrix, the current transformation matrix (CTM) that is part of the state and is applied to all vertices that pass down the pipeline
- The CTM is defined in the user program and loaded into a transformation unit



55

CTM in OpenGL

- OpenGL has a model-view and a projection matrix in the pipeline which are concatenated together to form the CTM
- Can manipulate each by first setting the matrix mode



56

Rotation, Translation, Scaling

Load an identity matrix:

```
glLoadIdentity()
```

Multiply on right:

```
glRotatef(theta, vx, vy, vz)
```

theta in degrees, (**vx, vy, vz**) define axis of rotation

```
glTranslatef(dx, dy, dz)
glScalef( sx, sy, sz)
```

Each has a float (f) and double (d) format (**glScaled**)

57

Example

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

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0, 2.0, 3.0);
glRotatef(30.0, 0.0, 0.0, .10);
glTranslatef(-1.0, -2.0, -3.0);
```

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

58

Arbitrary Matrices

- Can load and multiply by matrices defined in the application program

```
glLoadMatrixf(m)
glMultMatrixf(m)
```

- The matrix **m** is a one dimension array of 16 elements which are the components of the desired 4 x 4 matrix stored by columns
- In **glMultMatrixf, m** multiplies the existing matrix on the right

59

Matrix Stacks

- In many situations we want to save transformation matrices for use later
  - Traversing hierarchical data structures (Chapter 9)
  - Avoiding state changes when executing display lists
- OpenGL maintains stacks for each type of matrix
  - Access present type (as set by **glMatrixMode)** by

```
glPushMatrix()
glPopMatrix()
```

60

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 (**`colorcube.c`**) in a standard way
  - Centered at origin
  - Sides aligned with axes
  - Will discuss modeling in next lecture

61

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();
}
```

62

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;
 }
```

63

Display callback

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   glRotatef(theta[0], 1.0, 0.0, 0.0);
   glRotatef(theta[1], 0.0, 1.0, 0.0);
   glRotatef(theta[2], 0.0, 0.0, 1.0);
   colorcube();
   glutSwapBuffers();
}
```

Note that because of fixed from of callbacks, variables such as **`theta`** and **`axis`** must be defined as globals

Camera information is in standard reshape callback

64

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 **`gluLookAt`** (Chapter 5)
  - Build models of obejcts
- The projection matrix is used to define the view volume and to select a camera lens
- Although both are manipulated by the same functions, we have to be careful because incremental changes are always made by postmultiplication
  - For example, rotating model-view and projection matrices by the same matrix are not equivalent operations. Postmultiplication of the model-view matrix is equivalent to premultiplication of the projection matrix

65

OpenGL transformation Matrices

- Matrix that is applied to all primitives is the product of
  - the model-view matrix (GL_MODELVIEW), and
  - the projection matrix (GL_PROJECTION)
- We can
  - load a matrix, or
    glLoadMatrix(pointer_to_matrix)
  - set a matrix to the identity matrix
    glLoadIdentity()

66

OpenGL transformation Matrices

- We alter the selected matrix with

```
glMultMatrix(pointer_to_matrix), or
glRotatef(angle, vx,vy, vz)
glTranslatef(dx, dy, dz)
glScalef(sx, sy, sz)
```

- Example: rotation about a fixed point

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity();
glTranslatef(4.0, 5.0, 6.0);
glRotatef(45.0, 1.0, 2.0, 3.0);
glTranslatef(-4.0, -5.0, -6.0);
```

67

OpenGL transformation Matrices

Pushing and popping matrices

- Sometimes we need to do a transformation and then return to the same state as before its execution. This can be done by pushing the matrix on a stack before we do the transformation.

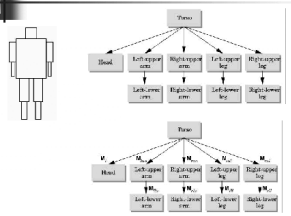
```
glPushMatrix();
glTranslatef(…);
glRotatef(….);
glScalef(…);

glPopMatrix();
```

68

OpenGL transformation Matrices

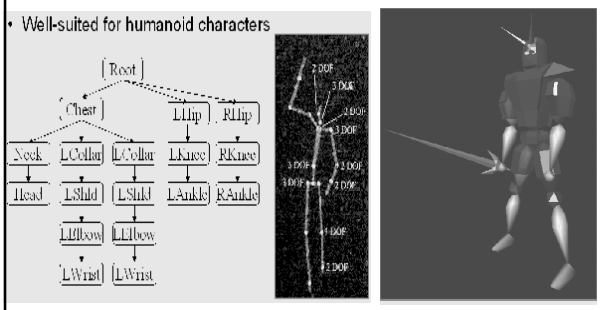
- Object hierarchical models



```
figure ()
{
glPushMatrix();   glPopMatrix();
torso();          glPushMatrix();
glTranslate       glTranslate
glRotate 3        glRoate3
head();           right_upper_leg();
glPopMatrix();    glPopMarix();
glPushMatrix();   glPushMatrix();
glTranslate       ……
glRoate3
left_upper_leg();
glTranslate
glRotate3
left_lower_leg();
```

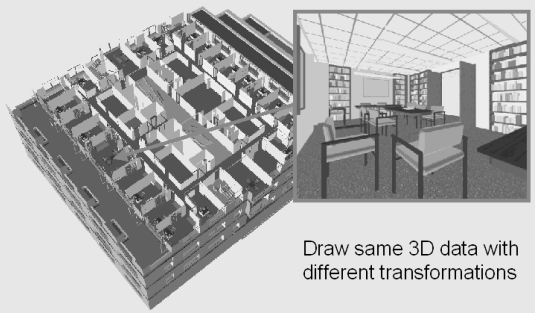

Transformation Example 1

- Well-suited for humanoid characters



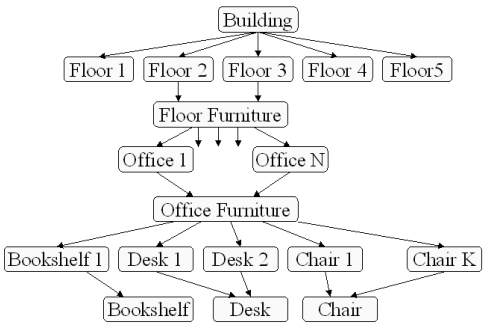
Transformation Example 2

- An object may appear in a scene multiple times



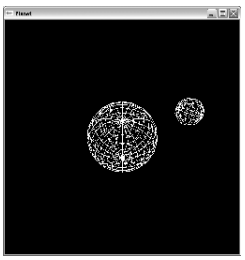
Draw same 3D data with different transformations

Transformation Example 2



Example -1

- planet.c
  - Control:
    - ‘d’
    - ‘y’
    - ‘a’
    - ‘A’
    - ESC



Example -2

```
#include <GL/glut.h>
static GLfloat year=0.0f, day=0.0f;

void init()
{    glClearColor(0.0, 0.0, 0.0, 0.0);                    }

void GL_reshape(GLsizei w, GLsizei h)          // GLUT reshape function
{
    glViewport(0, 0, w, h);                    // viewport transformation
    glMatrixMode(GL_PROJECTION);               // projection transformation
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 20.0);
    glMatrixMode(GL_MODELVIEW);                // viewing and modeling transformation
    glLoadIdentity();
    gluLookAt(0.0, 3.0, 5.0,       // eye
              0.0, 0.0, 0.0,       // center
              0.0, 1.0, 0.0);      // up
}
```


Example -3

```
void GL_display()       // GLUT display function
{
    // clear the buffer
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0, 1.0, 1.0);
    glPushMatrix();
        glutWireSphere(1.0, 20, 16);       // the Sun
        glRotatef(year, 0.0, 1.0, 0.0);
        glTranslatef(3.0, 0.0, 0.0);
        glRotatef(day, 0.0, 1.0, 0.0);
        glutWireSphere(0.5, 10, 8);        // the Planet
    glPopMatrix();
    // swap the front and back buffers
    glutSwapBuffers();
}
```


Example -4

```
void GL_idle()          // GLUT idle function
{
        day += 10.0;
        if(day > 360.0)  day -= 360.0;

        year += 1.0;
        if(year > 360.0) year -= 360.0;

        // recall GL_display() function
        glutPostRedisplay();
}
```

Example -5

```
void GL_keyboard(unsigned char key, int x, int y)          // GLUT keyboard function
{
    switch(key)
    {
        case 'd':   day += 10.0;
                    if(day > 360.0) day -= 360.0;
                    glutPostRedisplay();
                    break;
        case 'y':   year += 1.0;
                    if(year > 360.0) year -= 360.0;
                    glutPostRedisplay();
                    break;
        case 'a':   glutIdleFunc(GL_idle); // assign idle function
                    break;
        case 'A':   glutIdleFunc(0);
                    break;
        case 27:    exit(0);
    }
}
```

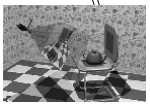

Example -6

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("Planet");
    init();
    glutDisplayFunc(GL_display);
    glutReshapeFunc(GL_reshape);
    glutKeyboardFunc(GL_keyboard);
    glutMainLoop();
    return 0;
}
```

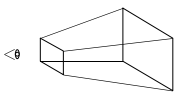

Viewing Transformations

- Position the camera/eye in the scene
  - place the tripod down; aim camera
- To "fly through" a scene
  - change viewing transformation and redraw scene
- **gluLookAt(eye_x, eye_y, eye_z, aim_x, aim_y, aim_z, up_x, up_y, up_z)**
  - up vector determines unique orientation
  - careful of degenerate positions

tripod



Projection Transformation

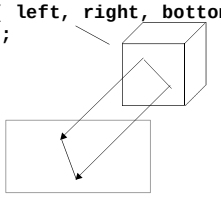


- Shape of viewing frustum
- Perspective projection
  - **gluPerspective(*fovy, aspect, zNear, zFar*)**
  - **glFrustum(*left, right, bottom, top, zNear, zFar*)**
- Orthographic parallel projection
  - **glOrtho(*left, right, bottom, top, zNear, zFar*)**
  - **gluOrtho2D(*left, right, bottom, top*)**
    - calls **glOrtho** with z values near zero

Applying Projection Transformations

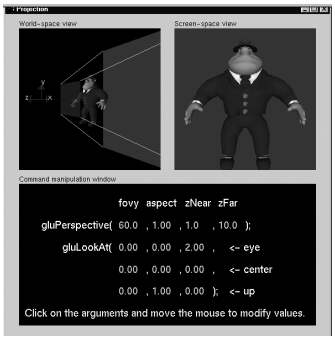
- Typical use (orthographic projection)

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glOrtho( left, right, bottom, top, zNear,
  zFar );
```



81

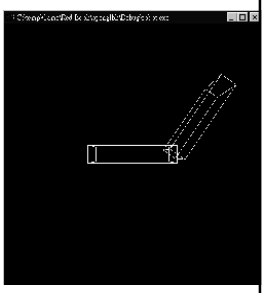
Projection Tutorial



82

Another Example: Robot

```
void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
    case 's':
      shoulder = (shoulder + 5) % 360;
      glutPostRedisplay();
      break;
    case 'S':
      shoulder = (shoulder - 5) % 360;
      glutPostRedisplay();
      break;
    case 'e':
      elbow = (elbow + 5) % 360;
      glutPostRedisplay();
      break;
    case 'E':
      elbow = (elbow - 5) % 360;
      glutPostRedisplay();
      break;
    case 27:
      exit(0);
      break;
    default:
      break;
  }
}
```



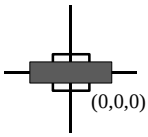
83

```
void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glRotatef ((GLfloat) shoulder, 0.0, 0.0, 1.0);
  glTranslatef (1.0, 0.0, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glutWireCube (1.0);
  glPopMatrix();

  glTranslatef (1.0, 0.0, 0.0);
  glRotatef ((GLfloat) elbow, 0.0, 0.0, 1.0);
  glTranslatef (1.0, 0.0, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glutWireCube (1.0);
  glPopMatrix();

  glPopMatrix();
  glutSwapBuffers();
}
```

Rotation center is at (0,0,-1) instead of (0,0,0)



glTranslate(2.0,0.0,0.0);
glTranslate(-1,0,0.0,0.0);

shoulder

elbow