Modeling Objects by Polygonal Approximations

- Define volumetric objects in terms of surfaces patches that surround the volume
- Each surface patch is approximated by a set of polygons
- Each polygon is specified by a set of vertices
- To pass the object through the graphics pipeline, pass the vertices of all polygons through a number of transformations using homogeneous coordinates
- All transformation are linear in homogeneous coordinates, thus a implemented as matrix multiplications

Linear and Affine Transformations (Maps)

- A map $f()$ is linear if it preserves linear combinations, i.e., that is , for any scalars α and β, and any vectors p and q,

  $f(\alpha p+\beta q) = \alpha f(p)+ \beta f(q)$

- Affine map $f()$ preserve affine combinations of points, that is, for any scalars α and β, where $\alpha+\beta=1$, and any points P and Q,

  $f(\alpha P+\beta Q) = \alpha f(P)+ \beta f(Q)$.

Linear and Affine Maps

- Recall that a line is an affine combination of two pints (thus an *image of a line is a line* under affine map).
- A polygon is a convex combination of its vertices, thus under an affine map, the *image of the polygon is* a convex combination of the transformed vertices, a *polygon*
- The vertices (in homogeneous coordinates) go through the graphics pipeline
- At the rasterization stage, the interior points are generated when needed
- Affine transformations include rotation, translation & scaling

Bilinear Interpolation

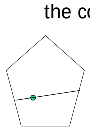
- Given the color at polygon vertices, assign color to the polygon points via bilinear interpolation:
  - An edge QR is convex combination of the two vertices Q and R, $0 \le \alpha \le 1$,

    $P(\alpha) = (1-\alpha)Q + \alpha R$
  - The color $C_{P(\alpha)}$ at an edge point $P(\alpha)$ is a linear interpolation of the color at the vertices C_Q, C_R

    $C_{P(\alpha)} = (1-\alpha)C_Q + \alpha C_R$

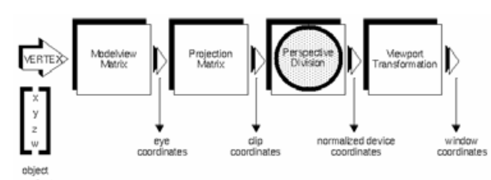
Bilinear Interpolation (cont)

- The color at an interior point is bilinear interpolation of the color at two edge points.



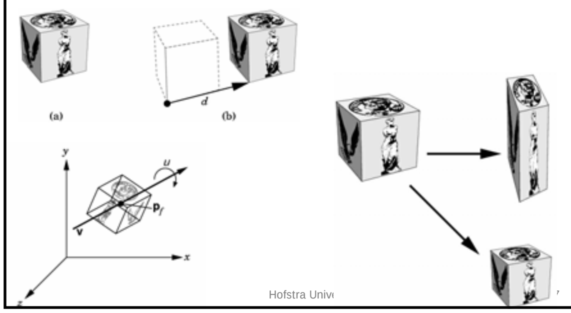
The polygon color is filled only when the polygon is displayed, during the the rasterization stage. The projection of the polygon is filled scan line by scan line. Each scan line intersects exactly 2 edges, thus color of an interior point is well-defined as bilinear interpolation of scan line intersections with the edges.

Modeling



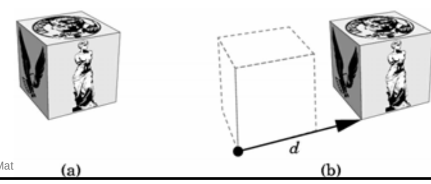
Affine Transformations

- Every affine transformation can be represented as a composition of translations, rotation, and scales (in some order)



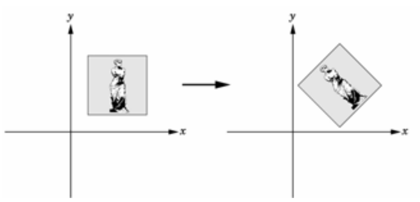
Translation

- Translation displaces points by a fixed distance in a given direction
- Only need to specify a displacement vector d
- Transformed points are given by
  $P' = P + d$



2D Rotations

Every 2D rotation has a fixed point



Rotations are represented by orthogonal matrices:
The rows (columns) are orthonormal.

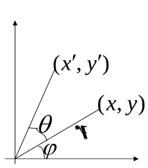
Matrix Representation of 2D Rotation around the origin

We want to find the representation of the transformation that rotates at angle θ about the origin.

Since we talk about origin, we have fixed a frame.

Given a point with coordinates (x,y), what are Coordinates (x',y') of the transformed point?

2D Rotation on θ with fixed point the origin



$$x = r\cos\varphi$$
$$y = r\sin\varphi$$
$$x' = r\cos(\theta+\varphi)$$
$$y' = r\sin(\theta+\varphi)$$

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

matrix representing the rotation

2D Rotation around the origin

- The origin is unchanged, called the fixed point of the transformation
- Extend 2D rotation to 3D. Use the right-handed system. Positive rotation is counter clockwise when looking down the axis of rotation toward the origin
- 2D rotation in the plane is equivalent to 3D rotation about the z axis: each point rotates in a plane perpendicular to z axis (i.e. z stays the same)

3D rotation on angle θ around the z axis

- The z axis is fixed by the rotation, the matrix representing the rotation is

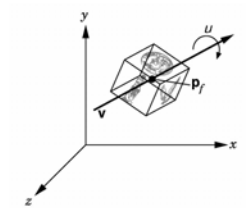
$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

Rotation in 3D around arbitrary axis

Must specify:
- rotation angle θ
- rotation axis, specified by a point $P_{f,}$ and a vector v

Note: openGL rotation is always around an axis through the origin

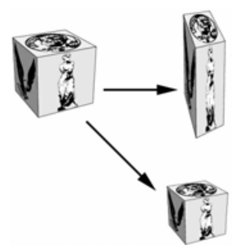


Rigid Body Transformation

•Rotation and translation are rigid-body transformations
•No combination of these transformations can alter the shape of an object

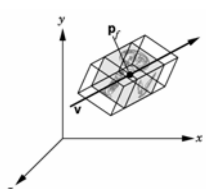
Non-rigid-body transformations

Scaling

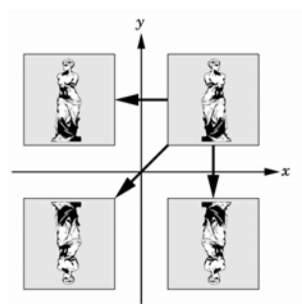


Scaling

- Must specify:
  - fixed point P_f
  - direction to scale
  - scale factor α
- $\alpha > 1$ larger
  $0 \le \alpha < 1$ smaller
  $-\alpha$ reflection
- Note: openGL scale more limited



Reflections



Transformations in Homogeneous Coordinates

- Graphics systems work with the homogeneous-coordinate representation of points and vectors
- This is what OpenGL does too
- In homogeneous coordinates, an affine transformation becomes a linear transformations and as such is represented by 4x4 matrix, ***M.***
- In homogeneous coordinates, the image of a point P, is the point ***M****P*, the image of a vector ***u*** , is the vector ***Mu.***

Transformations in Homogeneous Coordinates

- In homogeneous coordinates, each affine transformation is represented by a 4 x 4 matrix **M**
- To find the image **v** of a point/vector under the transformation, multiply **M** by the homogeneous coord. representation **u** of the point/vector

$$\mathbf{v} = \mathbf{M}\mathbf{u}, \quad \mathbf{M} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{d} \\ \mathbf{0} & 1 \end{bmatrix}, \ \mathbf{A}_{3\times3}, \mathbf{d}_{3\times1}, \mathbf{0}_{1\times3},$$

$\mathbf{A}$, rotations and scalings

$\mathbf{d}$, translations

- In affine coordinates, not every affine transformation can be represented by a matrix but it could be expressed in the form

$$\mathbf{v} = \mathbf{A}\mathbf{u} + \mathbf{d}$$

General affine in affine coord

$$p' = \mathbf{A}p + \mathbf{d}$$

Translation

- Translation is an operation that displaces points by a fixed distance and direction given by a vector *d*
- In affine coord. transformed points are given by

  $P' = P + d,$

  The affine coordinate equations are

$$x' = x + \alpha_x,$$

$$y' = y + \alpha_y,$$

$$z' = z + \alpha_z,$$

Translation

Matrix form in the homogeneous coordinates

$$\mathbf{p}' = \mathbf{T}\mathbf{p}, \quad \text{where}$$

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \\ 0 \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T is called the translation matrix .
The translation transformation is denoted by

$$T(\alpha_x, \alpha_y, \alpha_z) \quad \text{or} \quad T(d)$$

Translation: the inverse transformation

We can return to the original position by a displacement of –*d*, giving us the inverse:

$$\mathbf{T}^{-1}(\alpha_x, \alpha_y, \alpha_z) = \mathbf{T}(-\alpha_x, -\alpha_y, -\alpha_z) = \begin{bmatrix} 1 & 0 & 0 & -\alpha_x \\ 0 & 1 & 0 & -\alpha_y \\ 0 & 0 & 1 & -\alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translations commute, I.e. order does not matter

```
glTranslatef(dx,dy,dz);
If d1 and d2 are vectors, T(d1+d2)=T(d1)T(d2)
```


2D Rotation around a fixed point different than the origin

- 2D Rotation has a fixed point. We know the matrix representation for a rotation with fixed point the origin.
- Concatenate transformations to obtain the rotation with an arbitrary fixed point P
  - translate by d=O-P
  - rotate around the origin, O
  - translate back by -d

$$R_{P,\theta} = T(P-O) R_\theta T(O-P)$$

Scaling with fixed point the origin

- Scaling has a fixed point
- Let the fixed point be the origin
- Independent scaling along the coordinate axes

$x' = \beta_x x$
$y' = \beta_y y$
$z' = \beta_z z$

Scaling with fixed point the origin

The homogeneous-coordinate equations in matrix form

$$\mathbf{p}' = \mathbf{Sp},$$

where

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

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

Two scale transformations with the same fixed point commute.

3D Rotation around the x-axis

We derived the representation of the 3D rotation on angle Theta around the z axis, we use concatenation of transformations to derive the rotation around x-axis

$$R_x(\theta) = R_{z\to x} R_z(\theta) R_{x->z}$$

where $R_{x\to z}$ is a rotation aligning x-axis with the z-axis

3D Rotations around the x-axis (cont)

- First: find the rotation that aligns x-axis with the z-axis:
  - Rotations are represented by orthogonal matrices
  - For every orthogonal matrix, M:
    - M has orthonormal rows, i.e the dot product of a row with itself is 1, and the dot product of a row with a different row is 0
    - M sends its rows into the corresponding basis vectors

$$M\begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, M\begin{bmatrix} m_{21} \\ m_{22} \\ m_{23} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, M\begin{bmatrix} m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

    - Its transpose is its inverse

3D Rotations around the x-axis (cont)

- Find the rotation that aligns x-axis with the z-axis(cont):
  - This rotation is represented by an orthogonal matrix M
  - If we choose M in such a way that the third row is (1,0,0), it will send the x-axis into the z-axis
  - If we build an orthonormal basis with a third vector (1,0,0) and stack up the three vectors of the frame, we obtain that M
  - We choose that basis to contain the three coordinate vectors (0,1,0), (0,0,1), and (1,0,0), in this order. Then M sends (y,z,x) coordinate axis into (x,y,z)

$$M = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

And in homogeneous coordinates:

$$R_{x\to z} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

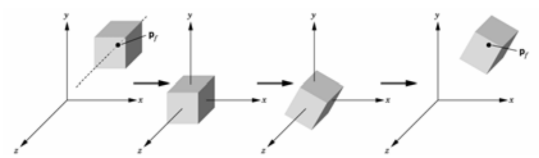
3D Rotations around the x-axis

- Since $R_{z\to x} = R_{x\to z}^{-1} = R'_{x\to z}$
- We obtain $\mathbf{R}_x(\theta) = R_{z\to x} R_z(\theta) R_{x\to z}$
- Thus

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

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

Rotation about axis not passing through origin, example: the axis is parallel to z-axis



Move the cube to the origin
Apply $R_z(\theta)$
Move back to original position

$$\mathbf{M} = \mathbf{T}(\mathbf{p}_f)\mathbf{R}_z(\theta)\mathbf{T}(-\mathbf{p}_f)$$

3D Rotations around and arbitrary axis through the origin, colinear with vector u

- Find the rotation that aligns u with the z axis
- Let u be unit vector (if not, normalize it).
- Next choose an orthonormal basis (u1,u2,u3), u3=u

$$R_{u\to z} = \begin{bmatrix} \mathbf{u_1} & 0 \\ \mathbf{u_2} & 0 \\ \mathbf{u_3} & 0 \\ \mathbf{0} & 1 \end{bmatrix}, \quad \mathbf{u_3} = \mathbf{u}, \quad R_{z\to u} = R'_{u\to z}$$

- Thus $\mathbf{R}_u(\theta) = R_{z\to u} R_z(\theta) R_{u\to z}$
- OpenGL, has a function for rotations around an axis through the origin

```
GlRotatef(theta,ux,uy,yz);
```


3D rotations around an arbitrary axis

- If the axis is in direction of a vector u, and is passing through an arbitrary point P

$$\mathbf{R}_{P;u}(\theta) = T(P-O)\, R_u(\theta)\, T(O-P)$$

- In OpenGL, if P(px,py,px), and u(ux,uy,uz), and we want to rotate on angle theta:

```
glTranslatef(px,py,pz);
glRotatef(theta, ux,uy,uz);
glTranslatef(-px,-py,-pz);
glBegin(GL_POINTS);
 …
glEnd();
```

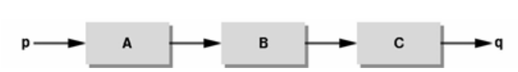

Scaling with an arbitrary fixed point; Composing Transformations

- We know how to scale with a fixed point origin. How do we scale fixing an arbitrary point P?
- Be careful when composing (concatenating) transformations: matrix multiplication is not commutative, and transformations composition is not commutative

Concatenation of Transformations

- We can multiply together sequences of transformations – concatenating
- Works well with pipeline architecture
- e.g., three successive transformations on a point $\mathbf{p}$ creates a new point $\mathbf{q}$

  $\mathbf{q} = \mathbf{CBAp}$



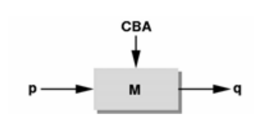
Concatenation of Transformations.

- If we have a lot of points to transform, then we can calculate

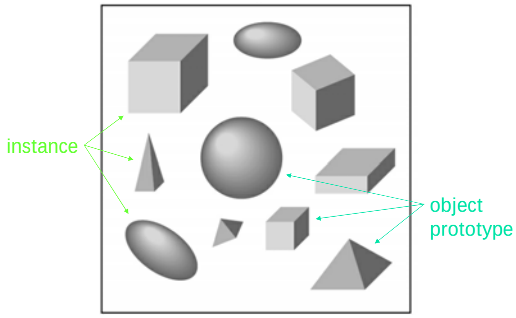
  $\mathbf{M} = \mathbf{CBA}$

  and then we use this matrix on each point

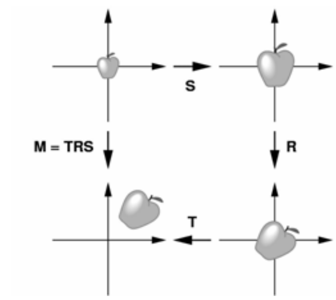
  $\mathbf{q} = \mathbf{Mp}$



Instance Transformation

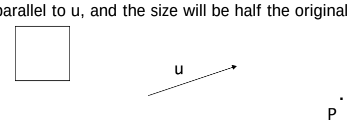


Instance Transformation



Instance Transformations

- Specify the affine transformation that will move the square so that its lower left corner will be at P, the vertical side will be parallel to u, and the size will be half the original size



Current Transformation Matrix

- Current Transformation Matrix (CTM) – defines the state of the graphics system. All drawings, (vertices) defined subsequently undergo that transformation.
- Changing the CTM, alters the state of the system.
- 4x4 matrix that can be altered by a set of functions provided by the graphics package
- Common to most systems. Part of the pipeline
- If **p** is a vertex, the pipeline produces **Cp**

Current Transformation Matrix

Let **C** denote the CTM.

CTM=I,	glLoadIdentity()
CTM=M (resets it),	glLoadMatrixf(pM)
CTM=CTM*M ,	glMultMatrixf(pM)

Application of the gl functions, post-multiplies CTM

```
glLoadIdentity();   // CTM=I
glMultMatrixf(pL);  // CTM=I*L
glMultMatrixf(pM);  // CTM=I*L*M
glBegin(GL_POINTS);
   glVertex3fv(v);
glEnd();
```

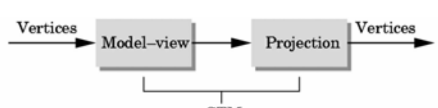
The point will be transformed according to CTM=I*L*M

Current Transformation Matrix

In OpenGL the CTM is the product of
model-view matrix (GL_MODELVIEW) and
projection matrix (GL_PROJECTION).

The model-view matrix is product of
viewing transformations and
modeling transformations

The projection matrix maps 3D to 2D.



Current Transformation Matrix

We select the matrix mode properly in order to set/change the model-view or the projection matrices.

glMatrixMode, set the desired matrix mode

```
 glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );
glRotatef(angle, vx, vy, vz);
glTranslatef(dx, dy, dz);
glScalef(sx, sy, sz);
 glMultMatrixf(pointer);
 glLoadMatrixf(pointer);
```


Order of Transformations

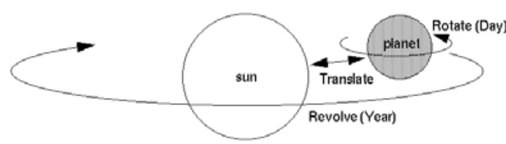
- We select the matrix mode properly in order to set/change the model-view or the projection matrices.
- Transformation specified most recently is the one applied first to the primitive

```
glMatrixModel(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);
 glBegin(GL_POLYGON);
  …
 glEnd();
```


World and Local Coordinate Systems

- An object moving relative to another moving object has a complicated motion:
  - A waving hand on a moving arm on a moving body
  - A rotating moon orbiting a planet orbiting a star
- Directly expressing such motions with transformations is difficult
- More indirect approach works better
- Notes: WUSTL

Example: planetary system




```
draw sun
rotate around Y by year
translate origin to orbit position
rotate around Y by day
draw moon at origin
```


Example: planetary system

```
// uses double buffering,
// glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);

void display() {
  glClearColor(GL_COLOR_BUFFER_BIT);
  glColor(1.0, 1.0, 1.0);

  glPushMatrix();
  glutWireSphere(1.0,20,16);   // draw sun
  glRotatef( year, 0.0, 1.0, 0.0);
  glTranslatef(2.0, 0.0,0.0);
  glRotatef( day, 0.0, 1.0, 0.0);
  glutWireSphere(0.2, 10, 8);   // draw moon
  glPopMatrix();

  glutSwapBuffers();
}
```