Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

CS 460/560

Computer Graphics

Professor Richard Eckert

Lecture # 12

February 28, 2001

Lecture 12

A. Homogeneous Geometric Transformations

B. A 2D Geometric Transformation Software Package

C. Shearing and Reflections

D. Coordinate System Transformations

Geometrical Transformations

- Alter or move objects on screen
- Affine Transformations:
  - Preserve straight lines
- Transform points in the object
  - Translation:
    - A Vector Sum
  - Rotation and Scaling:
    - Matrix Multiplies

Translation: Moving Objects

Translate point P by translation vector T

Components: tx, ty

Component equations:

x' = x + tx

y' = y + ty

This is a vector add:

P' = P + T

Scaling: Changing Size of Objects

Scaling Factors: sx, sy

P' = S*P

Component equations:

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

Scaling Matrix:

$$S = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix}$$

Rotation about Origin by q

Rotate point P by θ about origin

Component Equations:

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

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

P' = R*P

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

Transforming Objects

For example, lines

1. Transform every point & plot (too slow)
2. Transform endpoints, draw the line
   - Since these transformations are affine, result is the transformed line

Composite Transformations

- Successive transformations
- Want to come up with a composite transformation matrix, M_{comp}
  - Then P' = M_{comp}*P
  - Far fewer matrix multiplies
- Can't be done for our standard 2-D representation of points in space
- Need a new representation

Homogeneous Coordinates

- Redefine transformations so each is a matrix multiply
- Express each 2-D Cartesian point as a triple:
  - A 3-D vector in a "homogeneous" coordinate system

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} xh \\ yh \\ w \end{bmatrix} \quad \text{where } xh = w*x,\ yh = w*y$$

- Each (x,y) maps to an infinite number of homogeneous 3-D points, depending on w
- Take w=1
- Look at our affine geometric transformations

Homogeneous Translations

P' = P + T (Cartesian 2-D coordinates)

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} tx \\ ty \end{bmatrix}$$

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

(Homogeneous coords)
P' = T*P
What matrix is T?

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} t11 & t12 & t13 \\ t21 & t22 & t23 \\ t31 & t32 & t33 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

matrix multiplication	component eqns	Results
x'=t11*x+t12*y+t13	x'=x+tx;	t11=1, t12=0, t13=tx
y'=t21*x+t22*y+t23	y'=y+ty;	t21=0, t22=1, t23=ty
1 =t31*x+t32*y+t33		t31=0, t32=0, t33=1

So:

$$T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

Homogen. Scaling (wrt origin)

P' = S*P

Component Equations:
x' = sx*x, y' = sy*y

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s11 & s12 & s13 \\ s21 & s22 & s23 \\ s31 & s32 & s33 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Doing the matrix multiplication:

x' = s11*x + s12*y + s13
y' = s12*x + s22*y + s23
1 = s31*x + s32*y + s33

Comparing with component eqns:

s11=sx, s12=0, s13=0
s21=0, s22=sy, s23=0
s31=0, s32=0, s33=1

So:

$$S = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homogen. Rotation (about origin)

P' = R*P

Component Equations:
x' = x*cos(θ) - y*sin(θ), y' = x*sin(θ) + y*cos(θ)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Doing the matrix multiplication:

x' = r11*x + r12*y + r13
y' = r21*x + r22*y + r23
1 = r31*x + r32*y + r33

Comparing with component eqns:

r11= cos(θ) r12=-sin(θ) r13=0
r21= sin(θ) r22= cos(θ) r23=0
r31=0 r32=0 r33=1

So:

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

Composite Transformations with Homogeneous Coordinates

- All transformations implemented as homogeneous matrix multiplies
- Assume transformations T1, then T2, then T3:

  Homogeneous matrices are T1, T2, T3

  P' = T1*P

  P'' = T2*P' = T2*(T1*P) = (T2*T1)*P

  P'''=T3*P''=T3*((T2*T1)*P)=(T3*T2*T1)*P

  Composite transformation: T = T3*T2*T1

  P'''=T*P, Compute T just once!

Example

Rotate line from (5,5) to (10,5) by 90 about (5,5)

T1=T(-5,-5), T2=R(90), T3=T(5,5)

T=T3*T2*T1

```
    |1 0 5|   |cos90 -sin90 0|   |1 0 -5|
T = |0 1 5| * |sin90  cos90 0| * |0 1 -5|
    |0 0 1|   |  0     0    1|   |0 0  1|
   | 0  -1   10 |
T= | 1   0    0 |
   | 0   0    1 |
```

Example, continued

```
P1' = T*P1
      | 0 -1  10| |5|   |5|
P1' = | 1  0   0|*|5| = |5|
      | 0  0   1| |1|   |1|

      | 0 -1  10| |10|   | 5|
P2' = | 1  0   0|*| 5| = |10|
      | 0  0   1| | 1|   | 1|
i.e., P1' = (5,5),  P2' = (5,10)
```

Setting Up a General 2D Geometric Transformation Package

Multiplying a matrix & a vector: General 3D Formulation

```
|x'|    | a0 a1 a2 |   |x|
|y'|  = | a3 a4 a5 | * |y|
|z'|    | a6 a7 a8 |   |z|
So: x' = a0*x + a1*y + a2*z
    y' = a3*x + a4*y + a5*z
    z' = a6*x + a7*y + a8*z
(9 multiplies and 6 adds)
```

Multiplying a matrix & vector: Homogeneous Form

```
|x'|    | a0 a1 a2 |   |x|
|y'|  = | a3 a4 a5 | * |y|
| 1|    | 0  0  1  |   |1|
So: x' = a0*x + a1*y + a2
    y' = a3*x + a4*y + a5
(4 multiplies and 4 adds)
MUCH MORE EFFICIENT!
```

Multiplying 2 3D Matrices: General 3D Formulation

```
|c0 c1 c2|   |a0 a1 a2| |b0 b1 b2|
|c3 c4 c5| = |a3 a4 a5|*|b3 b4 b5|
|c6 c7 c8|   |a6 a7 a8| |b6 b7 b8|
So: c0 = a0*b0 + a1*b3 + a2*b6
    Eight more similar equations
(27 multiplies and 18 adds)
```

Multiplying 2 3D Matrices: Homogeneous Form

```
|c0 c1 c2|   |a0 a1 a2| |b0 b1 b2|
|c3 c4 c5| = |a3 a4 a5|*|b3 b4 b5|
|0  0  1 |   |0  0  1 | |0  0  1 |
So: c0 = a0*b0 + a1*b3 + 0
  (Similar equations for c1, c3, c4)
And: c2 = a0*b2 + a1*b5 +a2
  (Similar equation for c5)
(12 multiplies and 8 adds)
MUCH MORE EFFICIENT!
```

Much Better to Implement our Own Transformation Package

- In general, obtain transformed point P' from original point P:
- P' = M * P
- Set up a set of functions that will transform points
- Then devise other functions to do transformations on polygons
  - since a polygon is an array of points

- Store the 6 nontrivial homogeneous transformation elts. in a 1-D array A
  - The elements are a[i]
- Then represent any geometric transformation with the following matrix:

```
     | a[0] a[1] a[2] |
 M = | a[3] a[4] a[5] |
     |  0    0    1   |
```

- Define the following functions:
  - Enables us to set up and transform points and polygons:

```
settranslate(float a[6], float dx, float dy); // set xlate matrix
setscale(float a[6], float sx, float sy);     // set scaling matrix
setrotate(float a[6], float theta);           // set rotation matrix
combine(float c[6], float a[6], float b[6]);  // C = A * B
xformcoord(float c[6], POINT vi, POINT* vo);  // Vo=C*Vi
xformpoly(int n, POINT inpts[], POINT outpts[], float t[6]);
```

- The "Set" functions take parameters that define the translation, scaling, rotation and compute the transformation matrix elements a[i]
- The combine() function computes the composite transformation matrix elements of the matrix C which is equivalent to the concatenation of transformations A and B (C = A * B)

- The xformcoord(c[],Vi,Vo) function
  - Takes an input POINT (Vi, with x,y coordinates)
  - Generates an output POINT (Vo, with x',y' coordinates)
  - Result of the transformation represented by matrix C whose elements are c[i]

- The xformpoly(n,ipts[],opts[],t[]) function
  - takes an array of input POINTs (an input polygon)
  - and a transformation represented by matrix elements t[i]
  - generates an array of ouput POINTS (an output polygon)
    - result of the transformation represented by the t[i]
  - will make n calls to xformcoord()
    - n = number of points in input polygon

An Example--Rotating a Polygon about one of its Vertices by Angle q

- Rotation about (dx,dy) can be achieved by the composite transformation:
  1. Translate so vertex is at origin (-dx,-dy); Matrix T1
  2. Rotate about origin by θ; Matrix R
  3. Translate back (+dx,+dy); Matrix T2
- The composite transformation matrix would be: T = T2*R*T1

Some Sample Code: Rotating a Polygon about a Vertex

- See Online Notes
- Week 6: 2-D Geometric Transformation Software Package link
  - (toward the end of the document)

Example Code: rotating a polygon about a vertex

```
POINT p[4];                 // input polygon
POINT px[4];                // transformed polygon
int n=4;                    // number of vertices
int pts[]={0,0,50,0,50,70,0,70};  // poly vertex coordinates
float theta=30;             // the angle of rotation
float dx=50,dy=70;          // rotate about this vertex
float xlate[6];             // the transformation 'matrices'
float rotate[6];
float temp[6];
float final[6];
```

```
for (int i=0; i<n; i++)   // set up the input polygon
   {p[i].x=pts[2*i];
    p[i].y=pts[2*i+1];
DrawPolygon(p,n);      // draw original polygon
settranslate(xlate,-dx,-dy);   // set up T1 trans matrix
setrotate(rotate,theta);       // set up R rotaton matrix
combine (temp,rotate,xlate);  // compute R*T1  &...
                               // save in temp
settranslate(xlate,dx,dy);     // set up T2 trans matrix
combine(final,xlate,temp);   // compute T2*(R*T1) &...
                               // save in final
xformpoly(n,p,px,final);     // get transformed polygon px
DrawPolygon(px,n);           // draw transformed polygon
```

Setting Up More General Polygon Transformation Routines

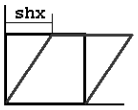
- trans_poly() could translate a polygon by tx,ty
- rotate_poly() could rotate a polygon by θ about point (tx,ty)
- scale_poly() could scale a polygon by sx, sy wrt (tx,ty)
- These would make calls to previously defined functions

General Polygon Transformation Function Prototypes

- void trans_poly(int n, POINT p[], POINT px[], int tx, int ty);
- void rotate_poly(int n, POINT p[], POINT px[], float theta, int tx, int ty);
- void scale_poly(int n, POINT p[], POINT px[], float sx, float sy, int tx, int ty);

Other 2D Affine Transformations

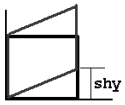
- Shearing (in x direction)
  - Move all points in object in x direction an amount proportional to y
  - Proportionality factor:
    - shx (x shearing factor)
  - Equations:




```
y′ = y
x′ = x + shx*y        |1  shx  0 |
P′ = SHX*P     SHX =  |0   1   0 |
                      |0   0   1 |
```

Shearing in y Direction

- Move all points in object in y direction an amount proportional to x
- Proportionality factor:
  - shy (y shearing factor)
- Equations:



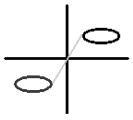

```
x′ = x
y′ = shy*x + y        |1   0   0 |
P′ = SHX*P     SHX =  |shy 1   0 |
                      |0   0   1 |
```

Reflections

- Reflect through origin

x --> -x

y--> -y



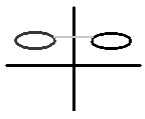
Equations:

```
x′ = -x
y′ = -y           |-1  0  0 |
P′ = Ro*P   Ro =  | 0 -1  0 |
                  | 0  0  1 |
```

Reflect Across y-axis

y --> y

x --> -x



Equations:

```
x′ = -x           |-1  0  0 |
y′ = y      Ry =  | 0  1  0 |
P′ = Ry*P         | 0  0  1 |
```

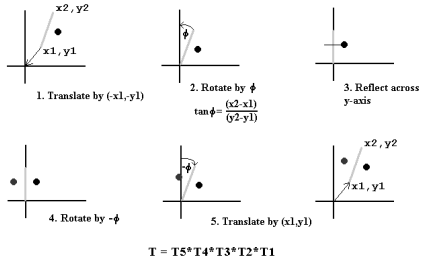
Reflect Across Arbitrary Line

Given line enpoints: (x1,y1), (x2,y2)

1. Tranlate by (-x1,-y1) [endpt at origin]
2. Rotate by ϕ [line coincides with y-axis]
3. Reflect across y-axis
4. Rotate by $-\phi$
5. Tranlate by (x1,y1)
6. Composite transformation:

   T = T(x1,y1)*R(-ϕ)*Ry*R(ϕ)*T(-x1,-y1)

Reflect Across a Line Endpoints (x1,y1), (x2,y2)

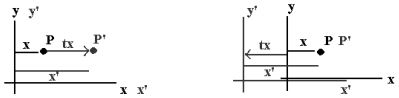


T = T5*T4*T3*T2*T1

Coordinate System Transformations

- Geometric Transformations:
  - Move object relative to stationary coordinate system (observer)
- Coordinate System Transformation:
  - Move coordinate system (observer) & hold objects stationary
  - Two common types
    - Coordinate System translation
    - Coordinate System rotation
  - Related to Geometric Transformations

Coordinate System Translation



Geometric Translation by + tx
x' = x + tx

Coordinate system Translation by -tx
x' = x + tx

So a Coordinate System Translation by vector -P is equivalent to a Geometric Translation by vector +P

$$T_G(P) <==> T_C(-P)$$

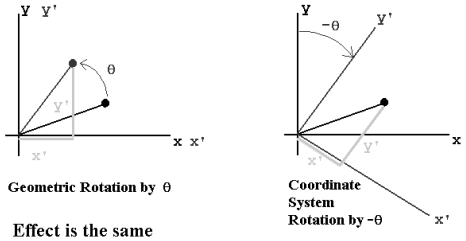
i.e., if P = (px,py):

$$T_G = \begin{vmatrix} 1 & 0 & px \\ 0 & 1 & py \\ 0 & 0 & 1 \end{vmatrix}$$

then:

$$T_C = \begin{vmatrix} 1 & 0 & -px \\ 0 & 1 & -py \\ 0 & 0 & 1 \end{vmatrix}$$

Coordinate System Rotation



Geometric Rotation by θ

Coordinate System Rotation by -θ

Effect is the same

$R_C(\theta) <==> R_G(-\theta)$