Geometric Transformations

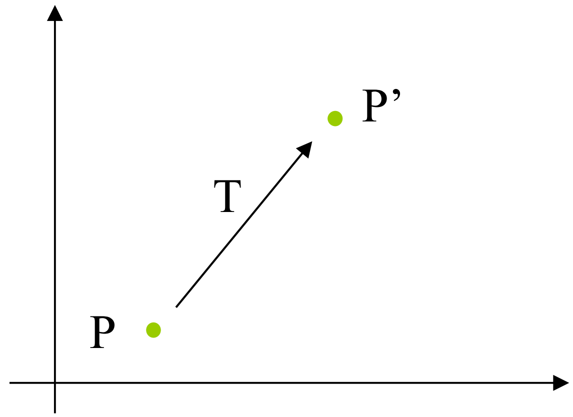
- Translation
- Rotation
- Scaling
- Reflection
- Shear

Geometric Transformations

- Changing an object’s position (translation), orientation (rotation) or size (scaling)
- Others transformations: reflection and shearing operations

Basic 2D Geometric Transformations

- 2D Translation



  - $x' = x + t_x$, $y' = y + t_y$

$$P = \begin{bmatrix} x \\ \hline y \end{bmatrix}, P' = \begin{bmatrix} x' \\ \hline y' \end{bmatrix}, T = \begin{bmatrix} t_x \\ \hline t_y \end{bmatrix}$$

  - P'=P+T
  - Translation moves the object without deformation

Basic 2D Geometric Transformations

- 2D Translation
  - To move a line segment, apply the transformation equation to each of the two line endpoints and redraw the line between new endpoints
  - To move a polygon, apply the transformation equation to coordinates of each vertex and regenerate the polygon using the new set of vertex coordinates

2D Translation Routine

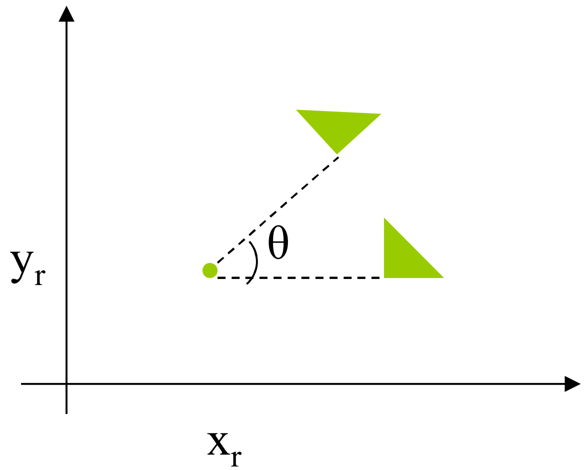
```
class wcPt2D {
    public:
       GLfloat x, y;
  };

  void translatePolygon (wcPt2D * verts, GLint nVerts, GLfloat tx, GLfloat ty)
  {
     GLint k;

     for (k = 0; k < nVerts; k++) {
        verts [k].x = verts [k].x + tx;
        verts [k].y = verts [k].y + ty;
     }
     glBegin (GL_POLYGON);
        for (k = 0; k < nVerts; k++)
           glVertex2f (verts [k].x, verts [k].y);
     glEnd ( );
  }
```

Basic 2D Geometric Transformations

- 2D Rotation
  - Rotation axis
  - Rotation angle
  - rotation point or pivot point (x_r, y_r)

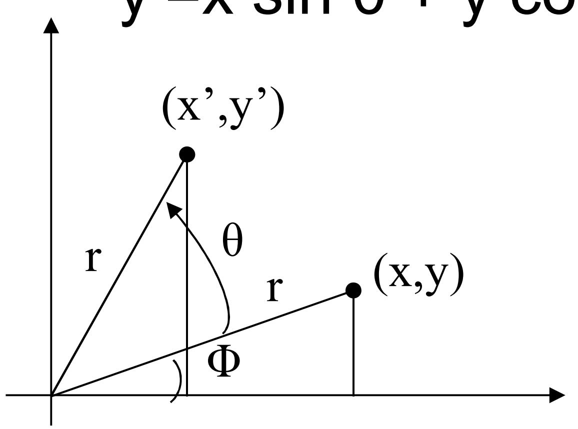


Basic 2D Geometric Transformations

- 2D Rotation
  - At first, suppose the pivot point is at the origin
  - $x' = r \cos(\theta + \Phi) = r \cos\theta \cos\Phi - r \sin\theta \sin\Phi$

    $y' = r \sin(\theta + \Phi) = r \cos\theta \sin\Phi + r \sin\theta \cos\Phi$
  - $x = r \cos\Phi$, $y = r \sin\Phi$
  - $x' = x \cos\theta - y \sin\theta$

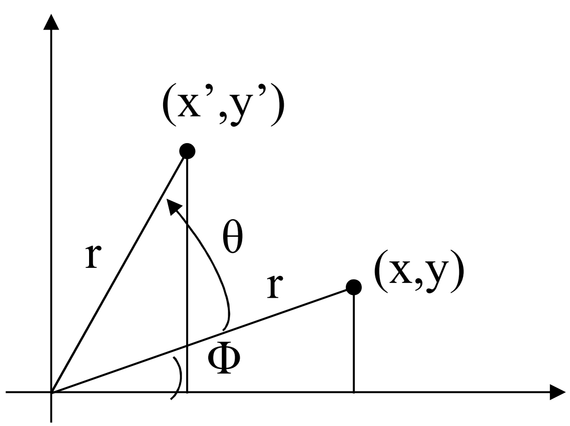
    $y' = x \sin\theta + y \cos\theta$



Basic 2D Geometric Transformations

- 2D Rotation
  - P'=R·P

$$R = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix}$$



Basic 2D Geometric Transformations

- 2D Rotation
  - Rotation for a point about any specified position (x_r,y_r)

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

    $y'=y_r+(x - x_r) \sin\theta + (y - y_r) \cos\theta$
  - Rotations also move objects without deformation
  - A line is rotated by applying the rotation formula to each of the endpoints and redrawing the line between the new end points
  - A polygon is rotated by applying the rotation formula to each of the vertices and redrawing the polygon using new vertex coordinates

2D Rotation Routine

```
class wcPt2D {
   public:
      GLfloat x, y;
};

void rotatePolygon (wcPt2D * verts, GLint nVerts, wcPt2D pivPt, GLdouble theta)
{
   wcPt2D * vertsRot;
   GLint k;

   for (k = 0; k < nVerts; k++) {
      vertsRot [k].x = pivPt.x + (verts [k].x - pivPt.x) * cos (theta) - (verts [k].y - pivPt.y) * sin (theta);
      vertsRot [k].y = pivPt.y + (verts [k].x - pivPt.x) * sin (theta) + (verts [k].y - pivPt.y) * cos (theta);
   }

   glBegin (GL_POLYGON);
      for (k = 0; k < nVerts; k++)
         glVertex2f (vertsRot [k].x, vertsRot [k].y);
   glEnd ( );
}
```

Basic 2D Geometric Transformations

- 2D Scaling
  - Scaling is used to alter the size of an object
  - Simple 2D scaling is performed by multiplying object positions (x, y) by scaling factors s_x and s_y

    $x' = x \cdot s_x$

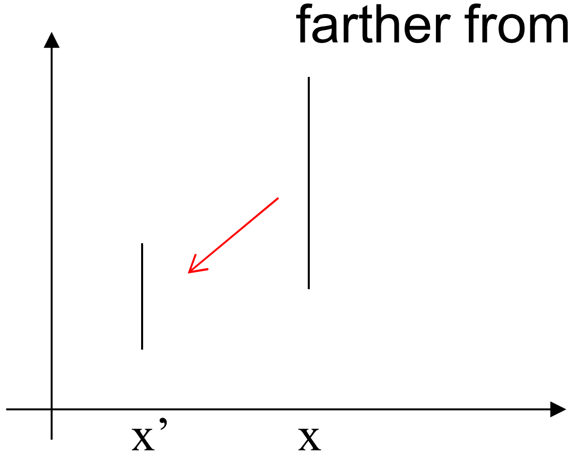
    $y' = y \cdot s_x$

    $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

    or $P' = S \cdot P$

Basic 2D Geometric Transformations

- 2D Scaling
  - Any positive value can be used as scaling factor
    - Values less than 1 reduce the size of the object
    - Values greater than 1 enlarge the object
    - If scaling factor is 1 then the object stays unchanged
    - If $s_x = s_y$, we call it <u>uniform scaling</u>
    - If scaling factor <1, then the object moves closer to the origin and If scaling factor >1, then the object moves farther from the origin



Basic 2D Geometric Transformations

- 2D Scaling
  - We can control the location of the scaled object by choosing a position called the **fixed point (x_f,y_f)**

    $x' - x_f = (x - x_f)\, s_x$ $\quad$ $y' - y_f = (y - y_f)\, s_y$

    $x'=x \cdot s_x + x_f\,(1 - s_x)$

    $y'=y \cdot s_y + y_f\,(1 - s_y)$
  - Polygons are scaled by applying the above formula to each vertex, then regenerating the polygon using the transformed vertices

2D Scaling Routine

```
class wcPt2D {
   public:
     GLfloat x, y;
 };
 void scalePolygon (wcPt2D * verts, GLint nVerts, wcPt2D fixedPt, GLfloat sx,
GLfloat sy)
 {
   wcPt2D vertsNew;
   GLint k;

   for (k = 0; k < n; k++) {
     vertsNew [k].x = verts [k].x * sx + fixedPt.x * (1 - sx);
     vertsNew [k].y = verts [k].y * sy + fixedPt.y * (1 - sy);
   }
   glBegin (GL_POLYGON);
     for (k = 0; k < n; k++)
       glVertex2v (vertsNew [k].x, vertsNew [k].y);
   glEnd ( );
 }
```

Matrix Representations and Homogeneous Coordinates

- Many graphics applications involve sequences of geometric transformations
  - Animations
  - Design and picture construction applications
- We will now consider matrix representations of these operations
  - Sequences of transformations can be efficiently processed using matrices

Matrix Representations and Homogeneous Coordinates

- $P' = M_1 \cdot P + M_2$
  - P and P' are column vectors
  - M_1 is a 2 by 2 array containing multiplicative factors
  - M_2 is a 2 element column matrix containing translational terms
  - For translation M_1 is the identity matrix
  - For rotation or scaling, M_2 contains the translational terms associated with the pivot point or scaling fixed point

Matrix Representations and Homogeneous Coordinates

- To produce a sequence of operations, such as scaling followed by rotation then translation, we could calculate the transformed coordinates one step at a time
- A more efficient approach is to combine transformations, without calculating intermediate coordinate values

Matrix Representations and Homogeneous Coordinates

- Multiplicative and translational terms for a 2D geometric transformation can be combined into a single matrix if we expand the representations to 3 by 3 matrices
  – We can use the third column for translation terms, and all transformation equations can be expressed as matrix multiplications

Matrix Representations and Homogeneous Coordinates

- Expand each 2D coordinate (x,y) to three element representation (x_h,y_h,h) called **homogenous coordinates**
- h is the homogenous parameter such that

$$x = x_h/h, \quad y = y_h/h,$$

- A convenient choice is to choose $h = 1$

Matrix Representations and Homogeneous Coordinates

- 2D Translation Matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or, $\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$

Matrix Representations and Homogeneous Coordinates

- 2D Rotation Matrix

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

or, **P'** = **R**(θ)·**P**

Matrix Representations and Homogeneous Coordinates

- 2D Scaling Matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or, $\mathbf{P'} = \mathbf{S}(s_x, s_y) \cdot \mathbf{P}$

Inverse Transformations

- 2D Inverse Translation Matrix

$$T^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Transformations

- 2D Inverse Rotation Matrix

$$R^{-1} = \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Transformations

- 2D Inverse Scaling Matrix

$$S^{-1} = \begin{bmatrix} \dfrac{1}{s_x} & 0 & 0 \\ 0 & \dfrac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Composite Transformations

- We can setup a sequence of transformations as a **composite transformation matrix** by calculating the product of the individual transformations
- $P' = M_2 \cdot M_1 \cdot P$

  $P' = M \cdot P$

2D Composite Transformations

- Composite 2D Translations

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

2D Composite Transformations

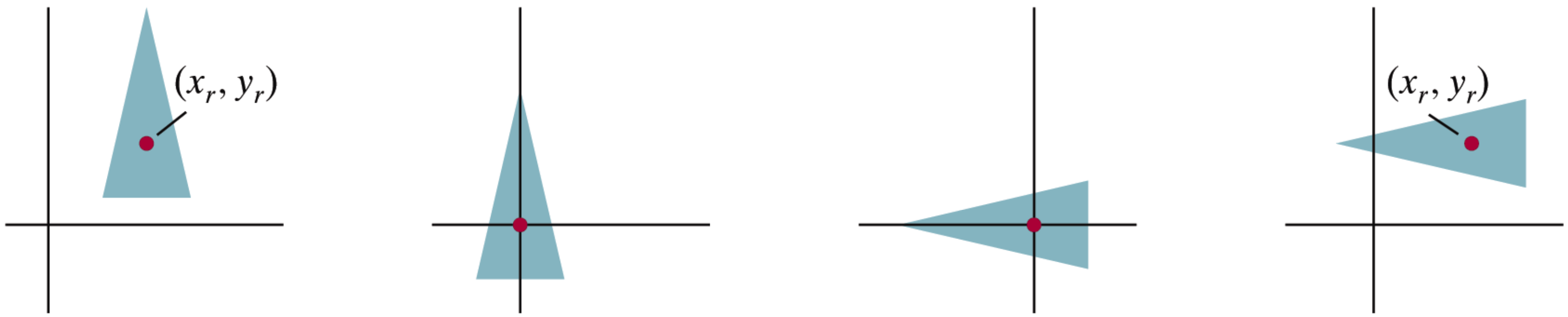
- Composite 2D Rotations

$$\begin{bmatrix} \cos\Theta_2 & -\sin\Theta_2 & 0 \\ \sin\Theta_2 & \cos\Theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 & 0 \\ \sin\Theta_1 & \cos\Theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\Theta_1 + \Theta_2) & -\sin(\Theta_1 + \Theta_2) & 0 \\ \sin(\Theta_1 + \Theta_2) & \cos(\Theta_1 + \Theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Composite Transformations

- Composite 2D Scaling

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



(a)
Original Position of Object and Pivot Point

(b)
Translation of Object so that Pivot Point (x_r, y_r) is at Origin

(c)
Rotation about Origin

(d)
Translation of Object so that the Pivot Point is Returned to Position (x_r, y_r)

Figure 5-9

A trans-formation sequence for rotating an object about a specified pivot point using the rotation matrix $\mathbf{R}(\theta)$ of transformation 5-19.

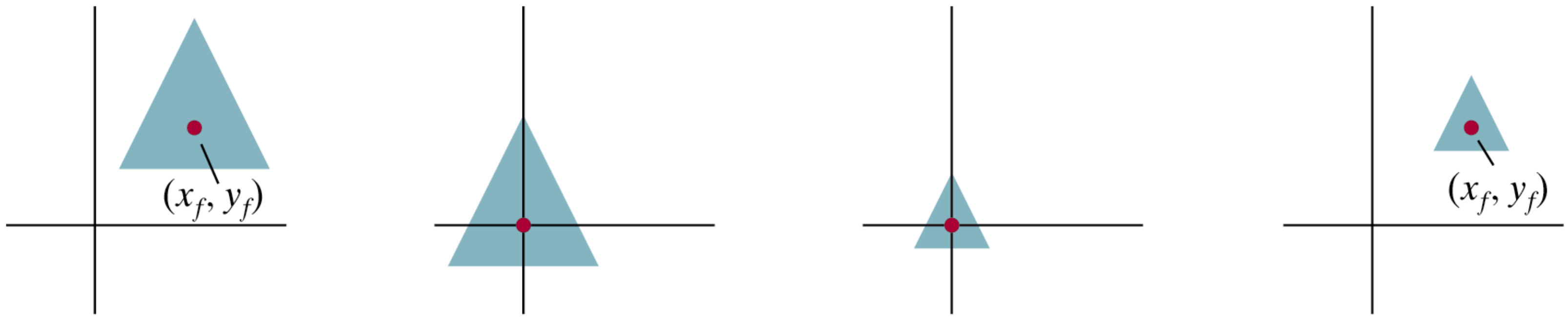
Computer Graphics with Open GL, Third Edition, by Donald Hearn and M.Pauline Baker.
ISBN 0-13-0-15390-7

2D Composite Transformations

- General 2D Pivot-Point Rotation

$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\Theta & -\sin\Theta & x_r(1-\cos\Theta) + y_r \sin\Theta \\ \sin\Theta & \cos\Theta & y_r(1-\cos\Theta) - x_r \sin\Theta \\ 0 & 0 & 1 \end{bmatrix}$$



(a) Original Position of Object and Fixed Point

(b) Translate Object so that Fixed Point (x_f, y_f) is at Origin

(c) Scale Object with Respect to Origin

(d) Translate Object so that the Fixed Point is Returned to Position (x_f, y_f)

Figure 5-10

A trans-formation sequence for scaling an object with respect to a specified fixed position using the scaling matrix $\mathbf{S}(s_x, s_y)$ of transformation 5-21.

Computer Graphics with Open GL, Third Edition, by Donald Hearn and M.Pauline Baker.

2D Composite Transformations

- General 2D Pivot-Point Rotation

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

2D Composite Transformations

- Matrix Concatenation Properties
  - $M_3 \cdot M_2 \cdot M_1 = (M_3 \cdot M_2) \cdot M_1 = M_3 \cdot (M_2 \cdot M_1)$
  - $M_2 \cdot M_1 \neq M_1 \cdot M_2$

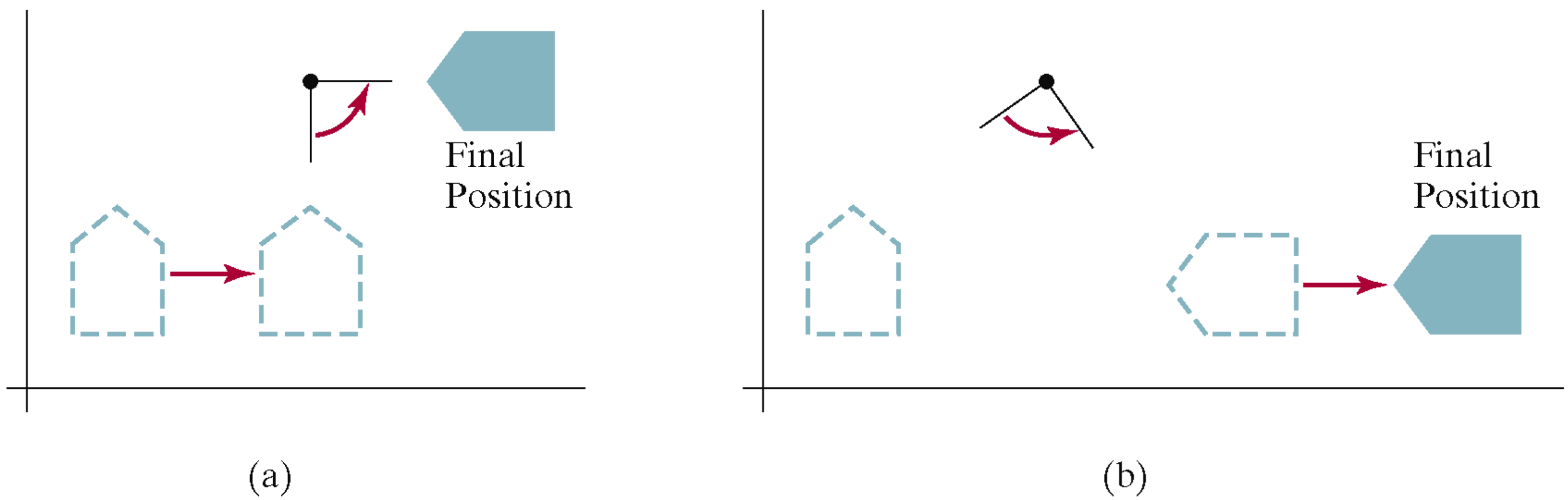


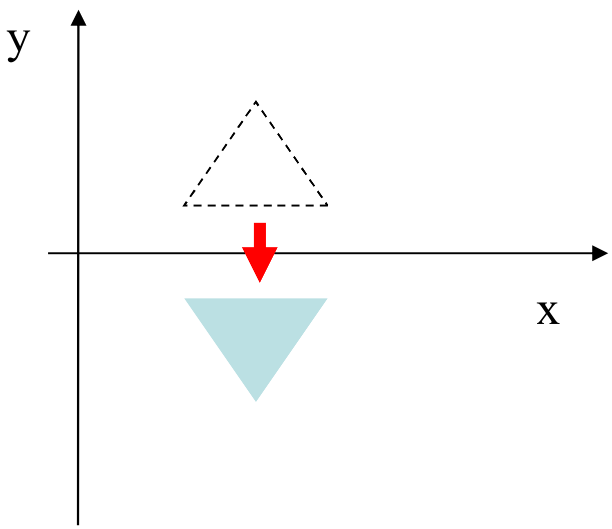
Figure 5-13

Reversing the order in which a sequence of transformations is performed may affect the transformed position of an object. In (a), an object is first translated in the x direction, then rotated counterclockwise through an angle of 45°. In (b), the object is first rotated 45° counterclockwise, then translated in the x direction.

Computer Graphics with Open GL, Third Edition, by Donald Hearn and M.Pauline Baker.
ISBN 0-13-0-15390-7

Other Two Dimensional Transformations

- Reflection
  - Transformation that produces a mirror image of an object



- Reflection
  - Image is generated relative to an axis of reflection by rotating the object 180° about the reflection axis
  - Reflection about the line y=0 (the x axis)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Other Two Dimensional Transformations

- Reflection
  - Reflection about the line x=0 (the y axis)

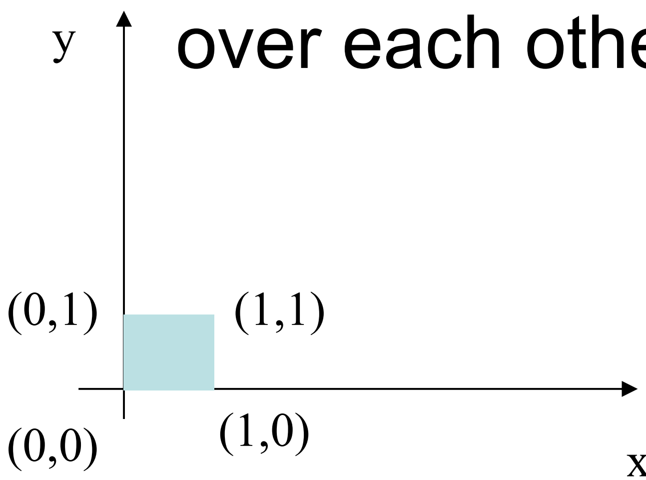
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

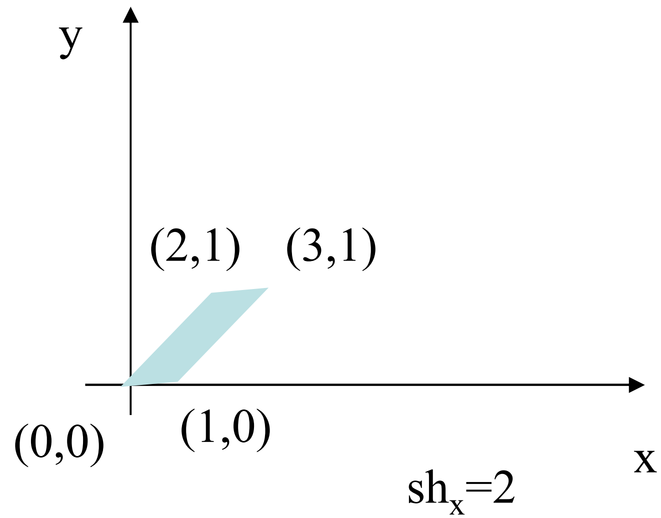
  - Reflection about the origin

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Other Two Dimensional Transformations

- Shear
  - Transformation that distorts the shape of an object such that the transformed shape appears as the object was composed of internal layers that had been caused to slide over each other





Other Two Dimensional Transformations

- Shear
  - An x-direction shear relative to the x axis

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{aligned} x' &= x + sh_x \cdot y \\ y' &= y \end{aligned}$$

  - An y-direction shear relative to the y axis

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Geometric Transformations in Three-Dimensional Space

Geometric Transformations in Three-Dimensional Space

- 3D transformation methods are extended from 2D methods by including considerations for the z coordinate
- A 3D homogenous coordinate is represented as a four-element column vector
  - Each geometric transformation operator is a 4 by 4 matrix

3D Translation

- $x' = x + t_x$, $y' = y + t_y$, $z' = z + t_z$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- $P' = P \cdot T$

3D Rotation

- Positive rotation angles produce counterclockwise rotations about a coordinate axis, assuming that we are looking in the negative direction along that coordinate axis

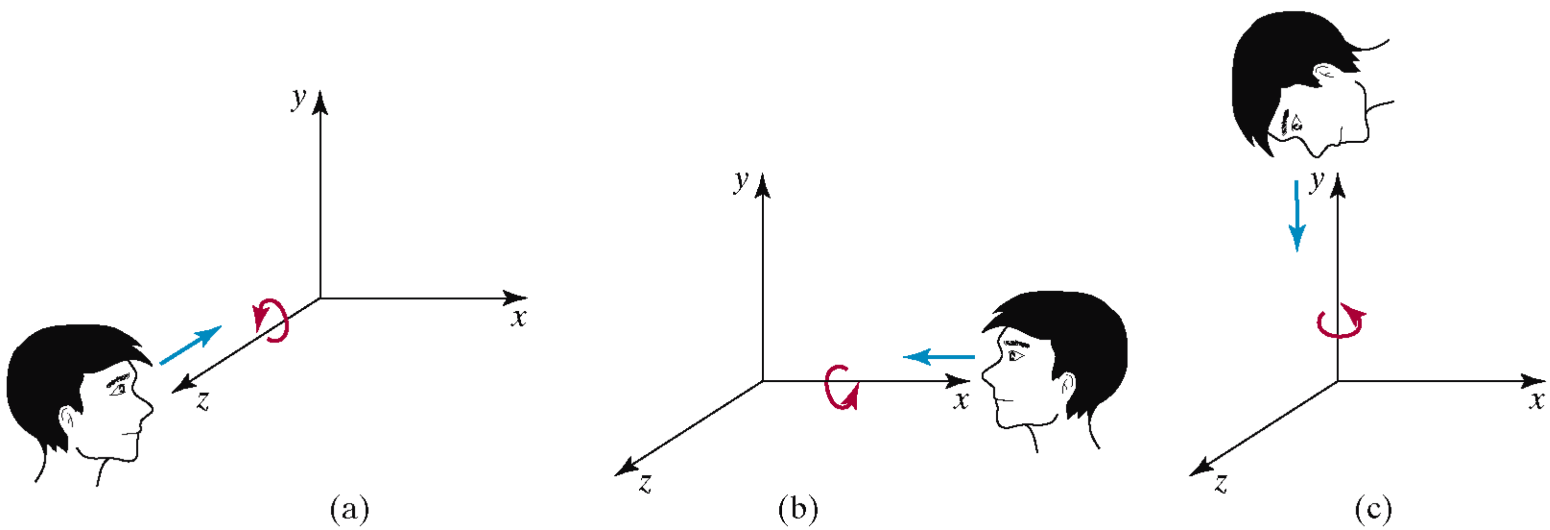


Figure 5-36

Positive rotations about a coordinate axis are counterclockwise, when looking along the positive half of the axis toward the origin.

Computer Graphics with Open GL, Third Edition, by Donald Hearn and M.Pauline Baker.
ISBN 0-13-0-15390-7

3D Rotation

- z-axis rotation
  - $x' = x \cos\theta - y \sin\theta$
  - $y' = x \sin\theta + y \cos\theta$
  - $z' = z$

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

or, $\mathbf{P'} = \mathbf{R_z}(\theta) \cdot \mathbf{P}$

3D Scaling

- Scaling relative to the coordinate origin

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

or, **P’ = S·P**

3D Scaling

- Scaling with respect to a fixed point (x_f, y_f, z_f)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

OpenGL Geometric Transformation Functions

- A separate function is available for each of the basic geometric transformations AND
- All transformations are specified in **three** dimensions

Basic OpenGL Geometric Transformations

- Translation
  - glTranslate* (tx, ty, tz);
    - * is either f or d
    - tx, ty and tz are any real number
    - For 2D, set tz=0.0
- Rotation
  - glRotate* (theta, vx, vy, vz);
    - * is either f or d
    - theta is rotation angle in degrees
    - Vector v=(vx, vy, vz) defines the orientation for a rotation axis that passes through the coordinate origin
    - For 2D, set tz=0.0

Basic OpenGL Geometric Transformations

- Scaling
  - glScale* (sx, sy, sz);
    - * is either f or d
    - sx, sy and sz are any real number
    - Negative values generate **reflection**

OpenGL Matrix Operations

- Modelview matrix, used to store and combine geometric transformations
  - glMatrixMode(GL_MODELVIEW);
- A call to a transformation routine generates a matrix that is multiplied by the current matrix
- To assign the identity matrix to the current matrix
  - glLoadIdentity();

OpenGL Matrix Stacks

- OpenGL maintains a matrix stack for transformations
- Initially the modelview stack contains only the identity matrix
- To save the current matrix on the stack
  - glPushMatrix( );
- To restore the matrix from the stack
  - glPopMatrix( );

OpenGL Transformation Examples

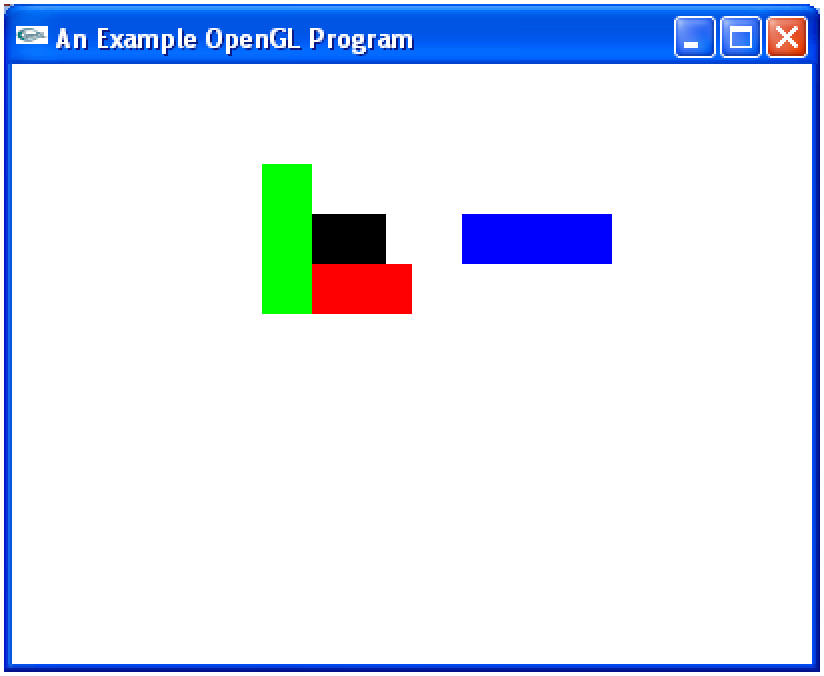
```
glMatrixMode (GL_MODELVIEW);

glColor3f (0.0, 0.0, 1.0);
glRecti(50,100,200,150);

glColor3f (1.0, 0.0, 0.0);
glTranslatef(-200.0,-50.0, 0.0);
glRecti(50,100,200,150);

glLoadIdentity();
glColor3f (0.0, 1.0, 0.0);
glRotatef(90.0,0.0, 0.0,1.0);
glRecti(50,100,200,150);

glLoadIdentity();
glColor3f (0.0, 0.0, 0.0);
glScalef(-0.5,1.0, 1.0);
glRecti(50,100,200,150);
```



OpenGL Transformation Examples

- This is more efficient

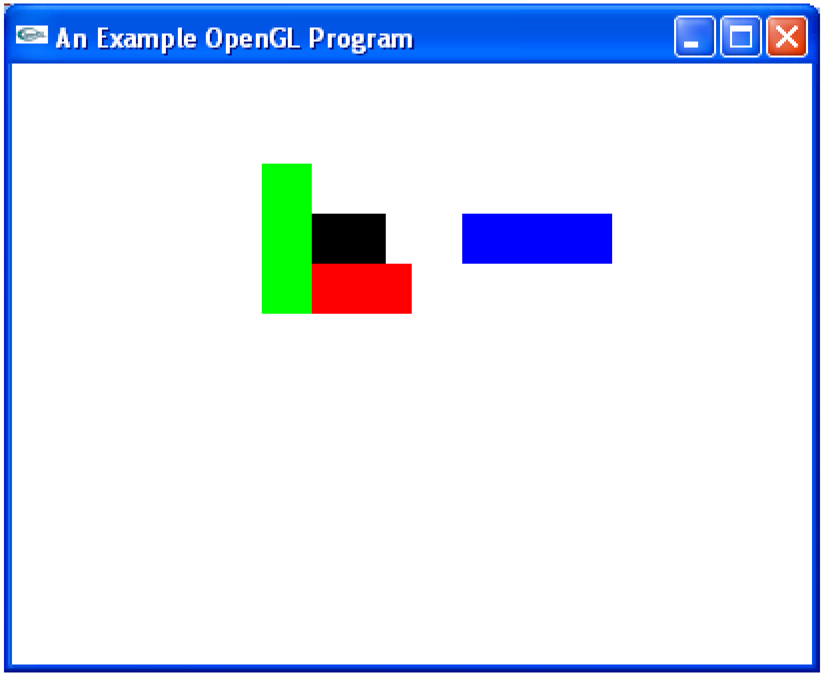
```
glMatrixMode (GL_MODELVIEW);

glColor3f (0.0, 0.0, 1.0);
glRecti(50,100,200,150);

glPushMatrix();
glColor3f (1.0, 0.0, 0.0);
glTranslatef(-200.0,-50.0, 0.0);
glRecti(50,100,200,150);

glPopMatrix();
glPushMatrix();
glColor3f (0.0, 1.0, 0.0);
glRotatef(90.0,0.0, 0.0,1.0);
glRecti(50,100,200,150);

glPopMatrix();
glColor3f (0.0, 0.0, 0.0);
glScalef(-0.5,1.0, 1.0);
glRecti(50,100,200,150);
```



OpenGL Transformation Routines

- For example, assume we want to do in the following order:
  - translate by +2, -3, +4,
  - rotate by 45^0 around axis formed between origin and 1, 1, 1
  - scale with respect to the origin by 2 in each direction.
- Our code would be

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();                    //start with identity
glScalef(2.0,2.0,2.0);   //Note: Start with the LAST operation
glRotatef(45.0,1.0,1.0,1.0);
glTranslatef(2.0,-3.0, 4.0); //End with the FIRST operation
```