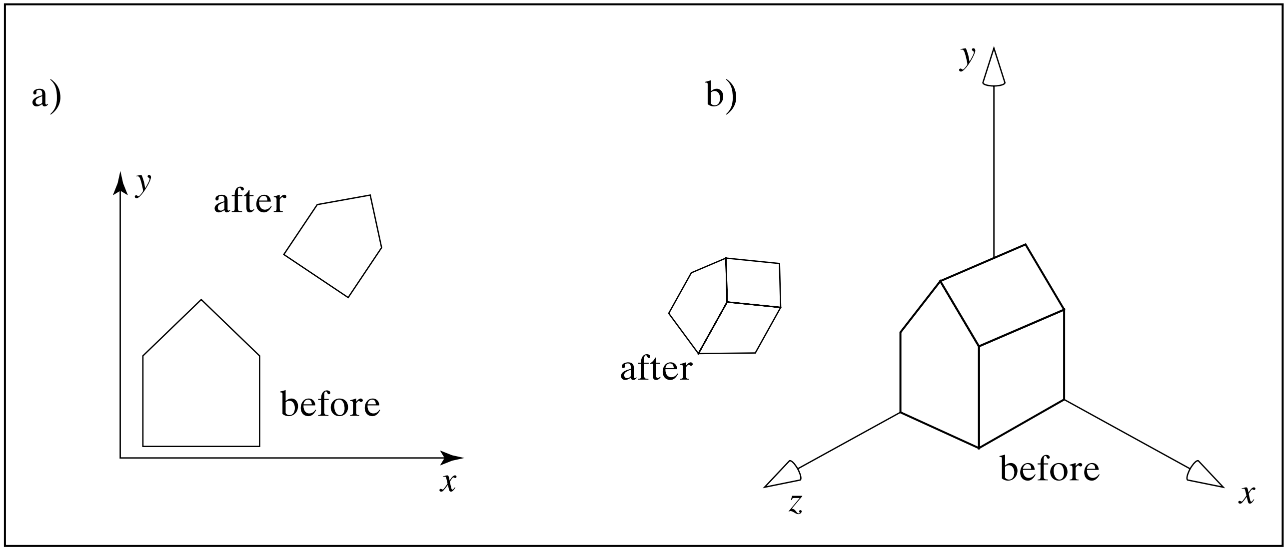


FIGURE 5.1 Drawings of objects before and after they are transformed.

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

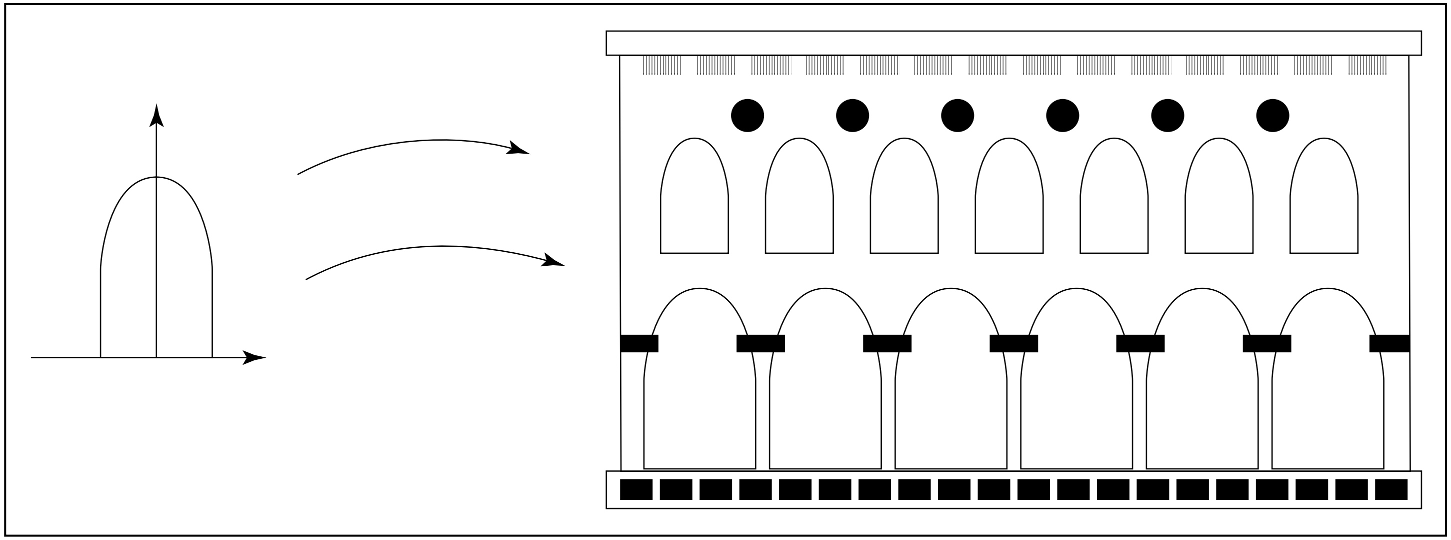
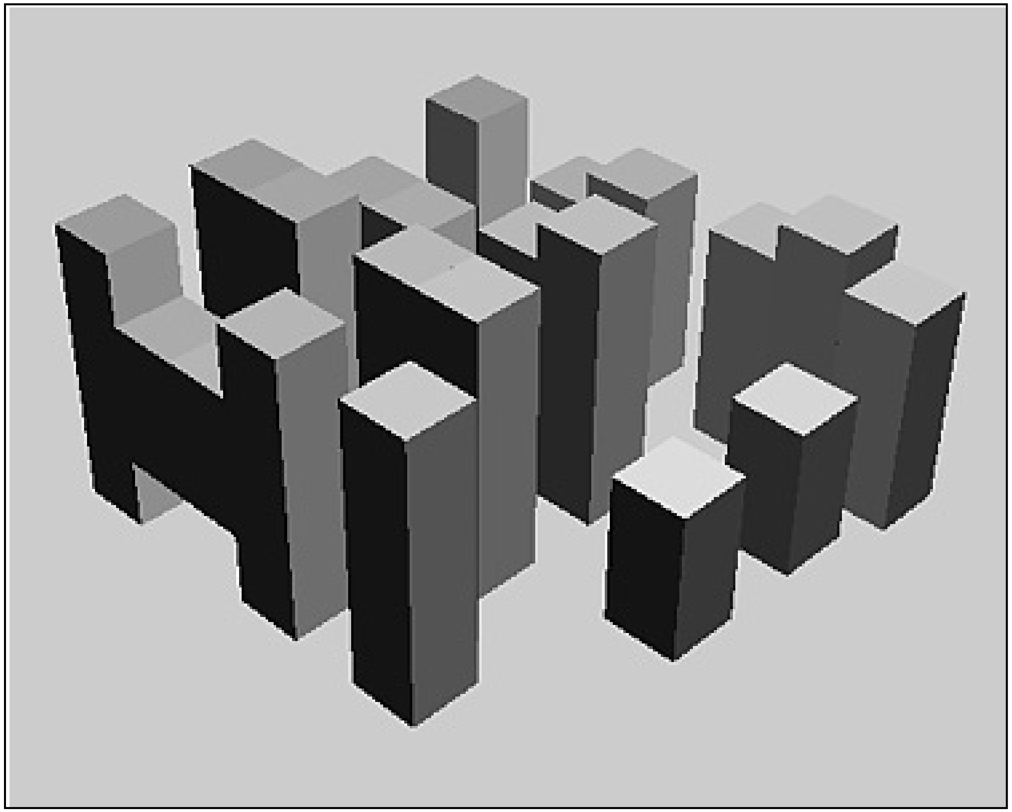


FIGURE 5.2 Composing a picture from many instances of a simple form.

FIGURE 5.3 Composing a 3D scene from primitives.



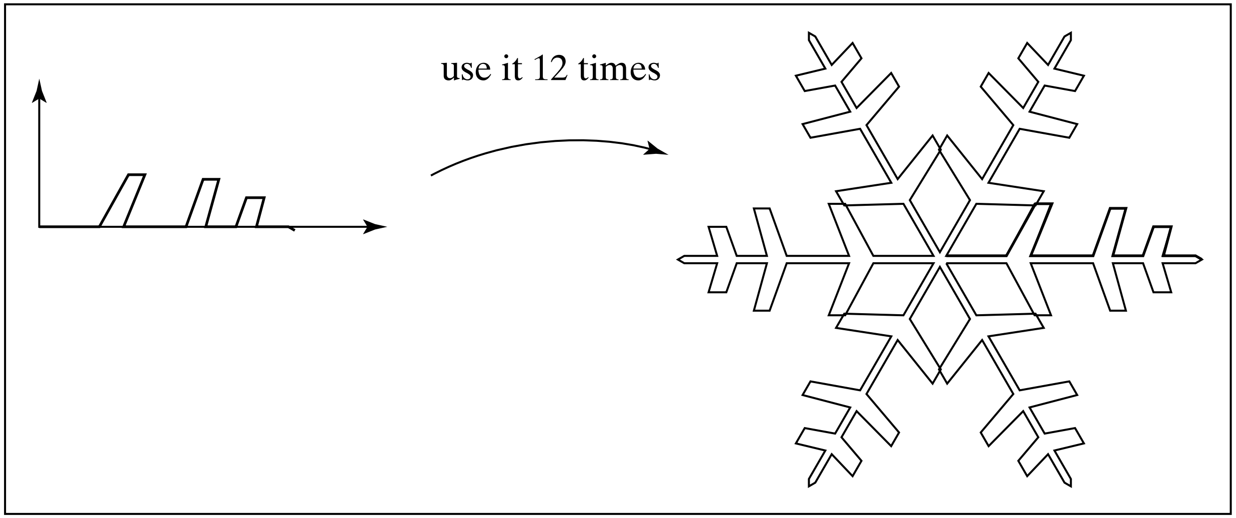
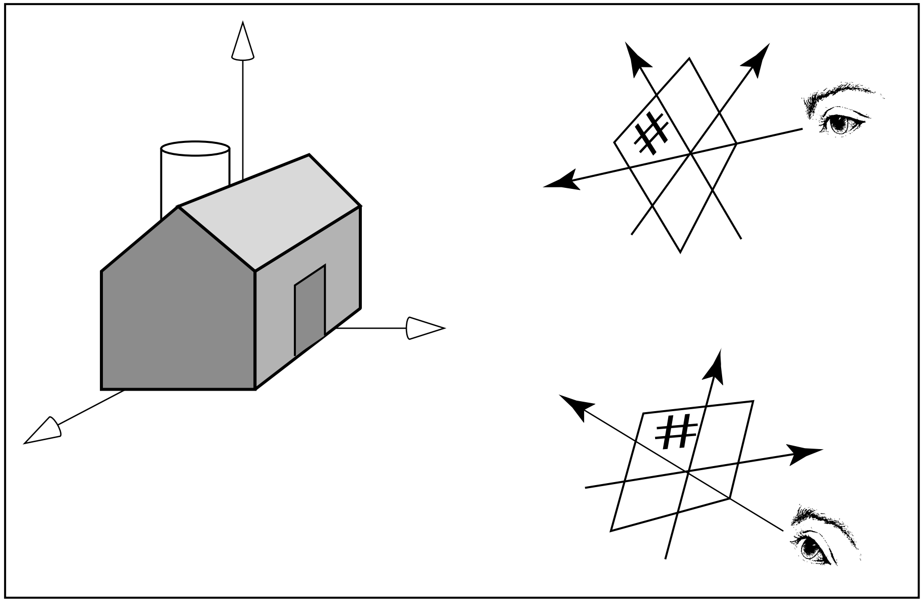


FIGURE 5.4 Using a "motif" to build up a figure.

FIGURE 5.5 Viewing a scene from different vantage points.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

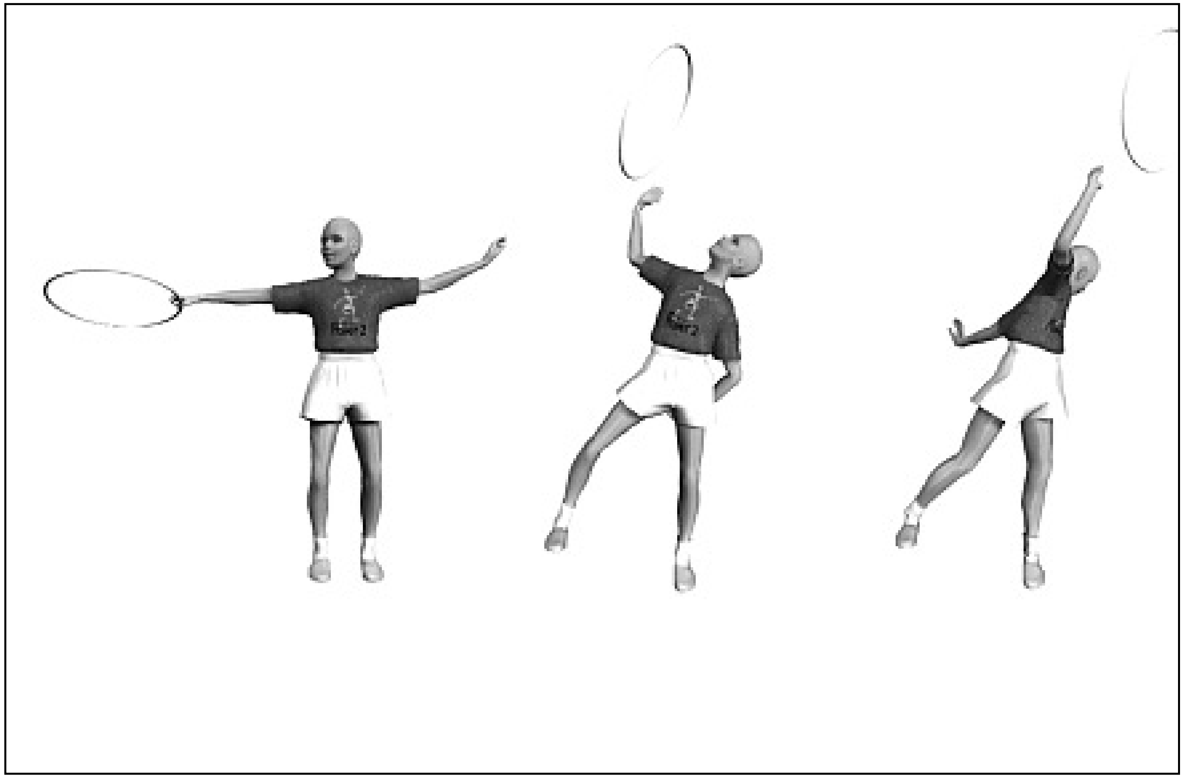


FIGURE 5.6 Animating by transforming shapes.

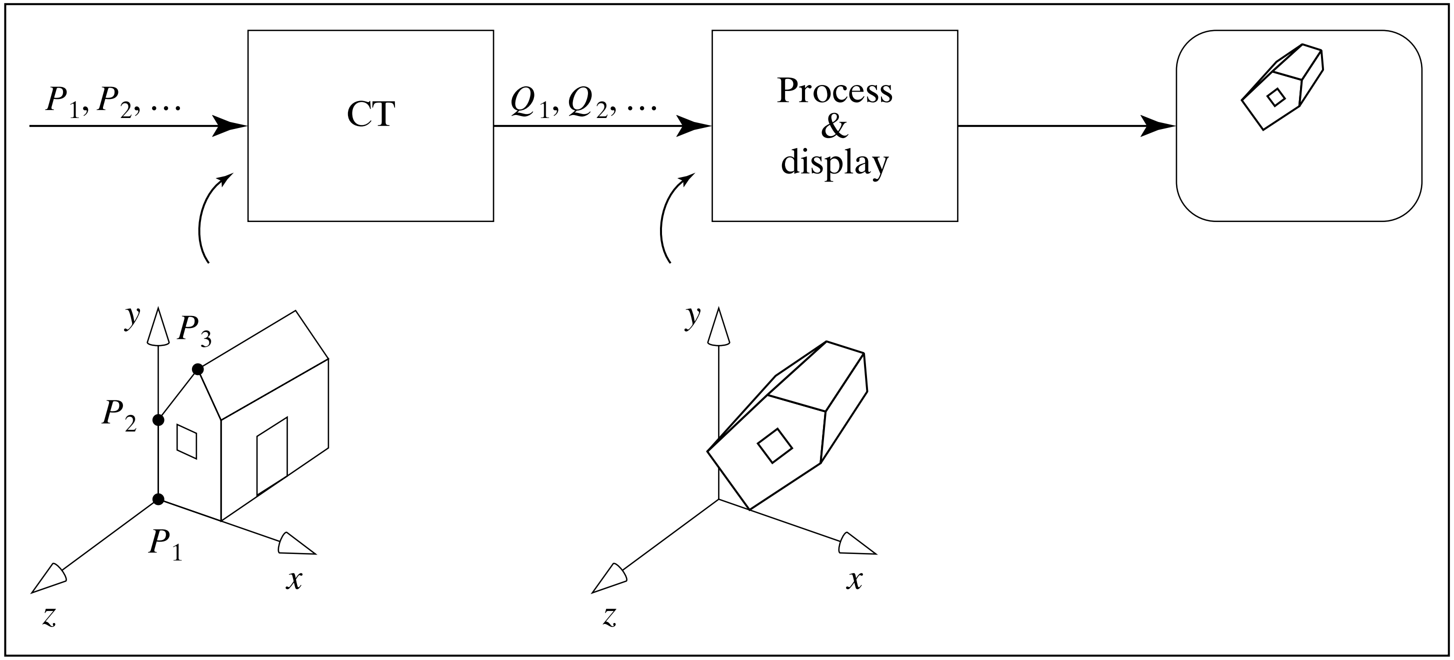
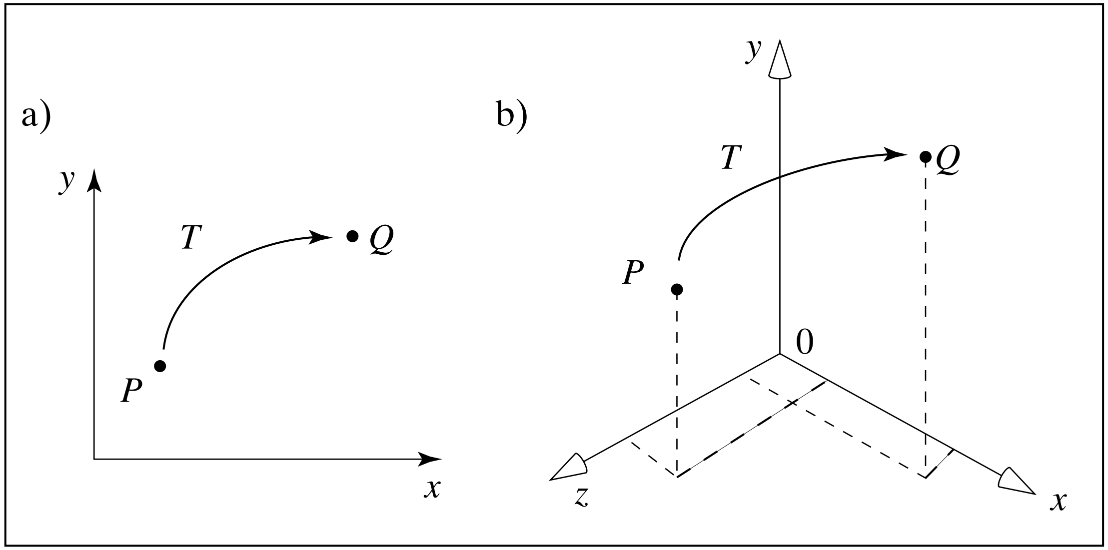


FIGURE 5.7 The OpenGL pipeline.

FIGURE 5.8 Mapping points into new points.



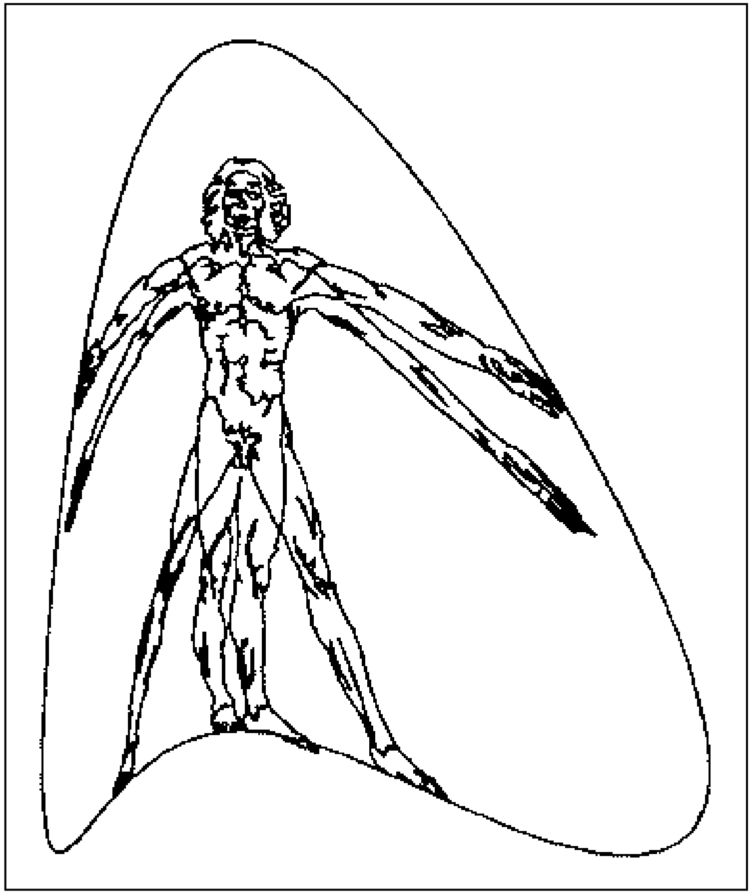


FIGURE 5.9 A complex warping of a figure.

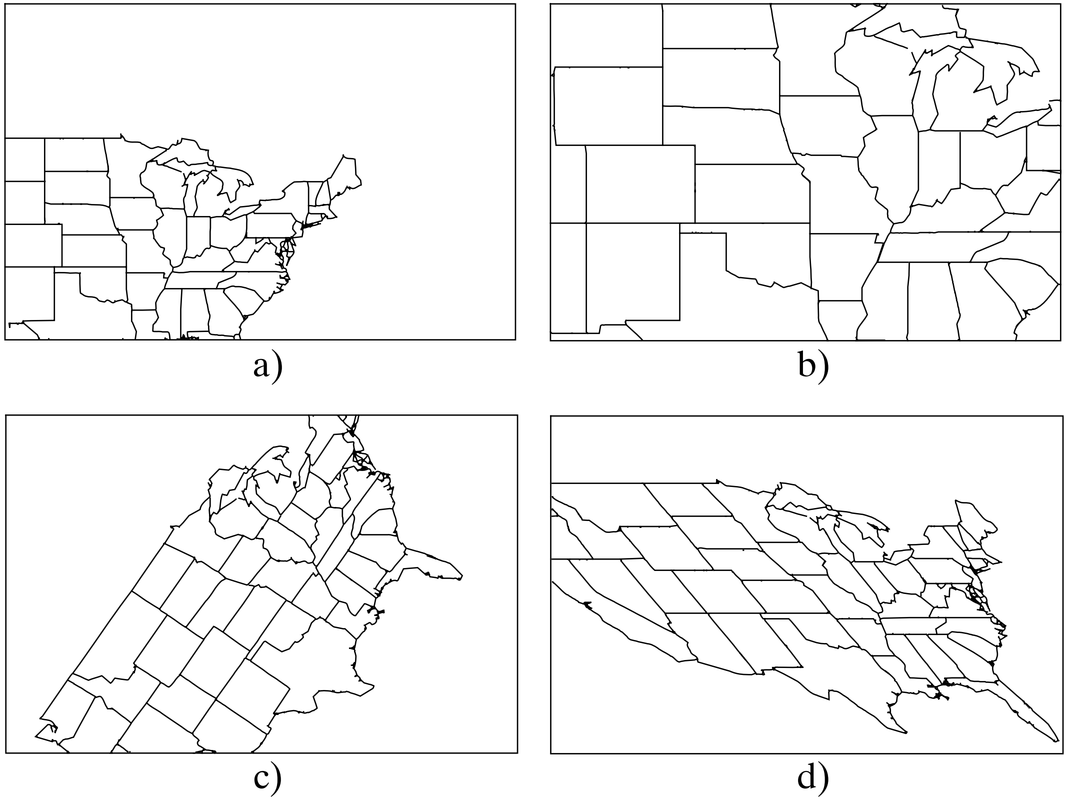
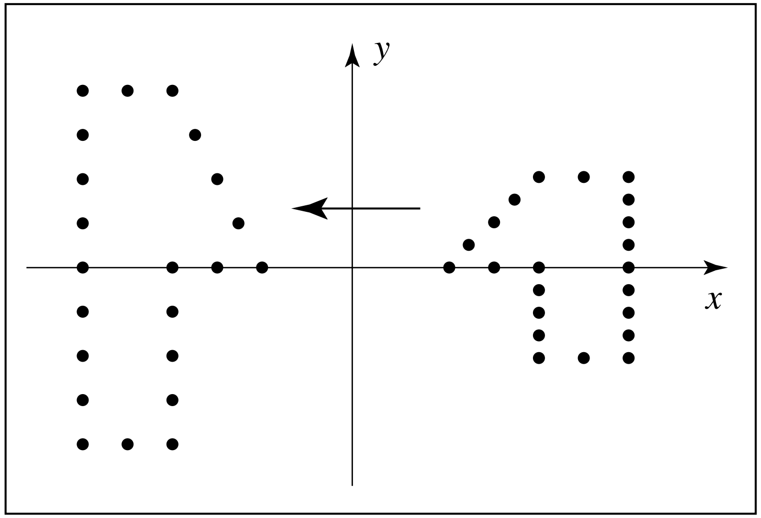


FIGURE 5.10 Transformations of a map: (a) translation; (b) scaling; (c) rotation; (d) shear.

FIGURE 5.11 A scaling and a reflection.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

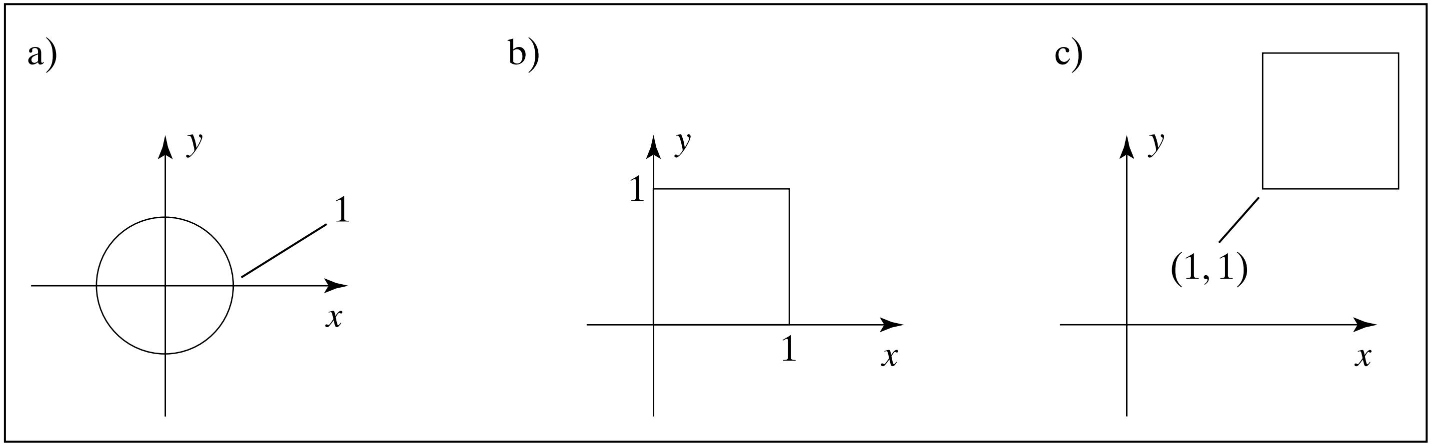


FIGURE 5.12 Objects to be scaled.

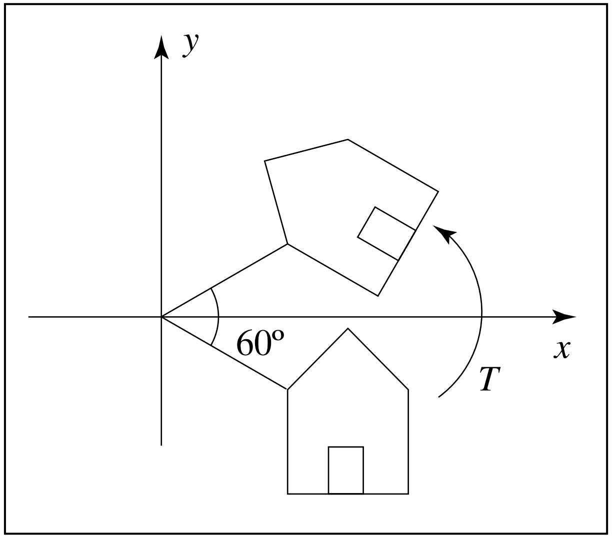


FIGURE 5.13 Rotation of points through an angle of 60°.

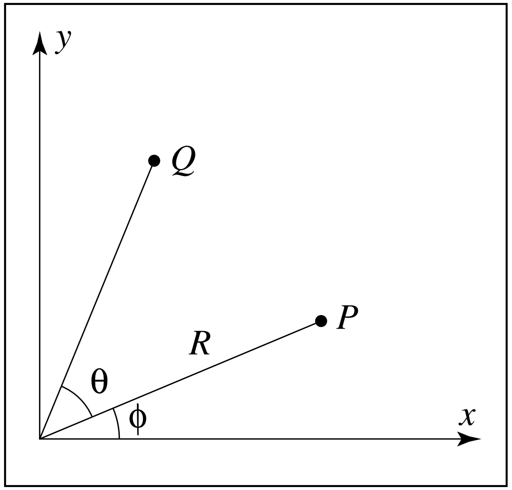


FIGURE 5.14 Derivation of the rotation mapping.

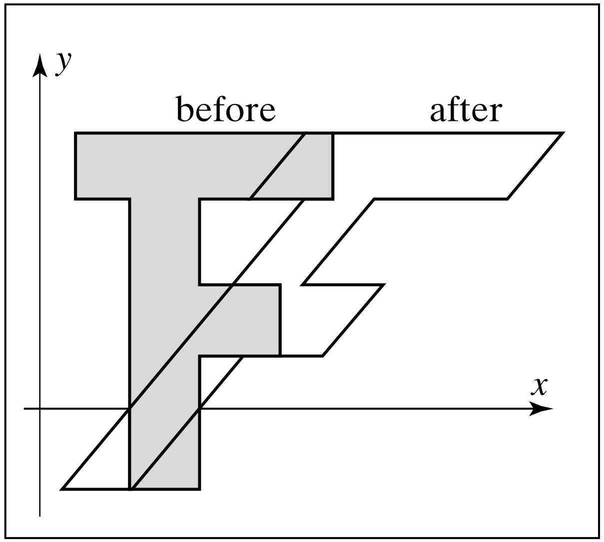


FIGURE 5.15 An example of shearing.

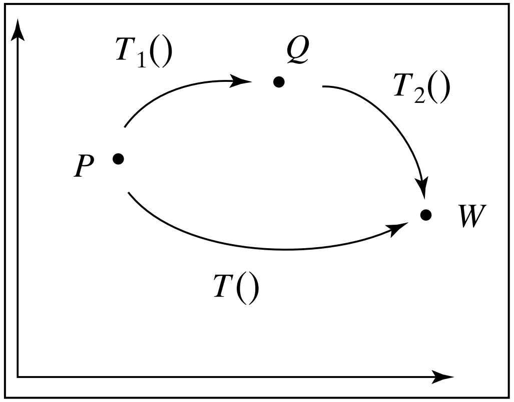


FIGURE 5.16 The composition of two transformations.

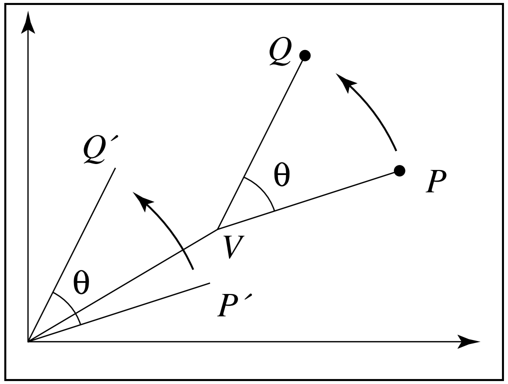


FIGURE 5.17 Rotation about a point.

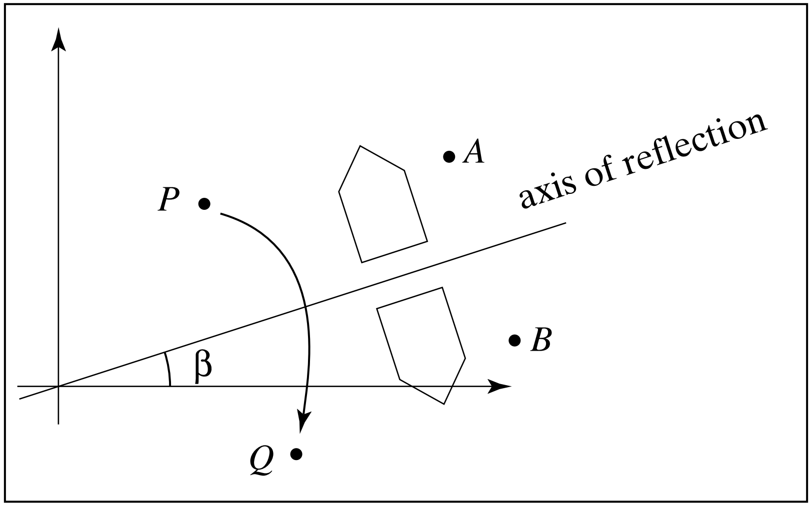


FIGURE 5.18 Reflecting a point about a tilted axis.

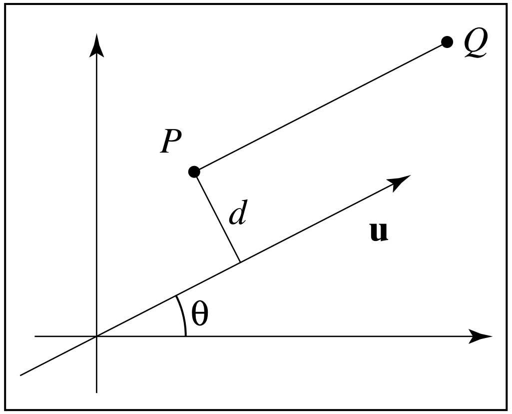


FIGURE 5.19 Shearing along a tilted axis.

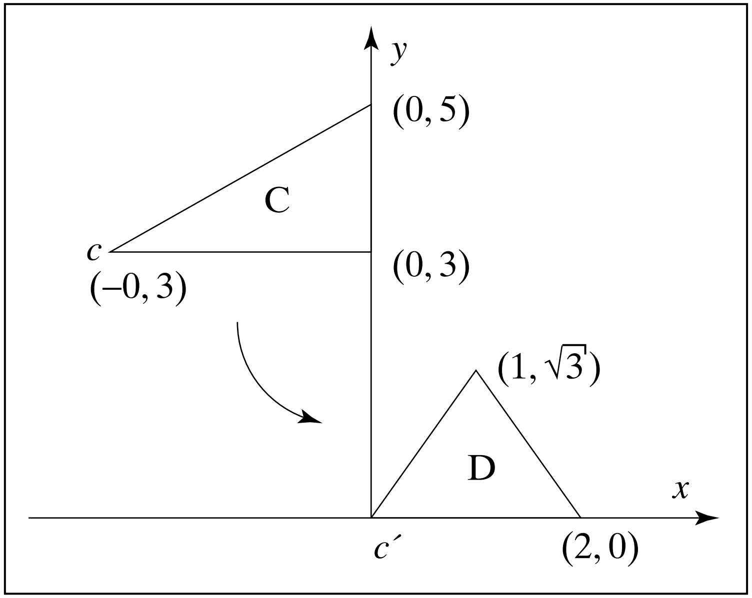
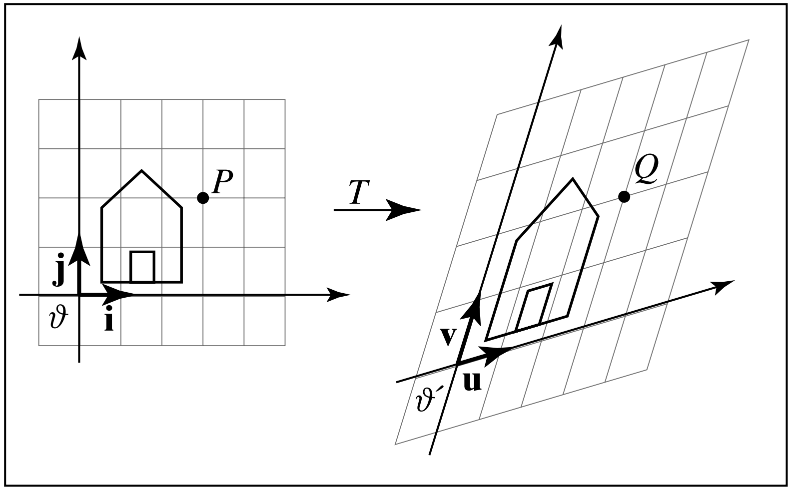


FIGURE 5.20 Converting one triangle into another.

FIGURE 5.21 A transformed grid.



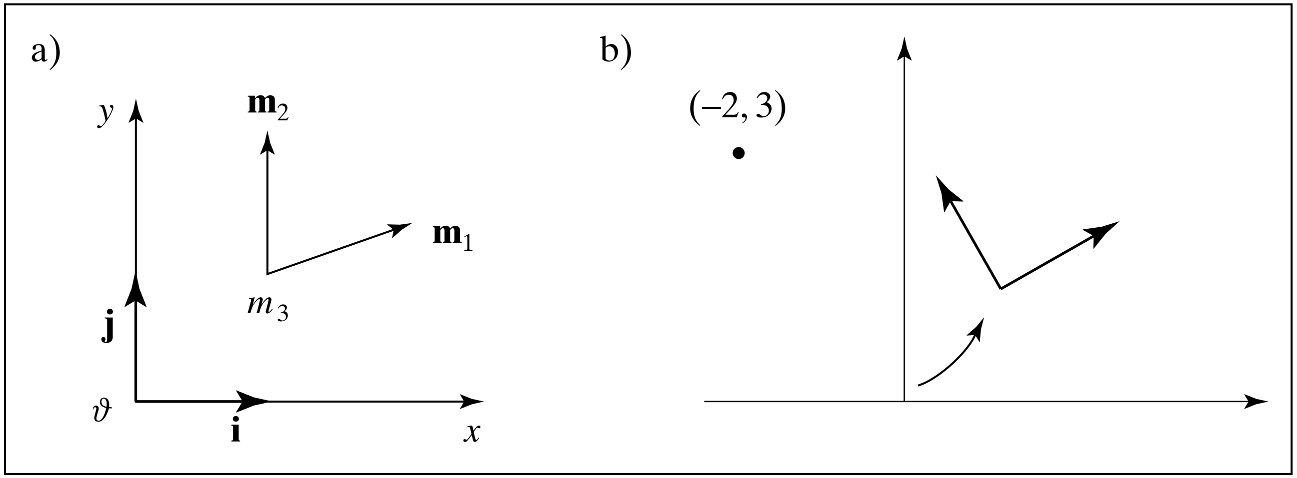
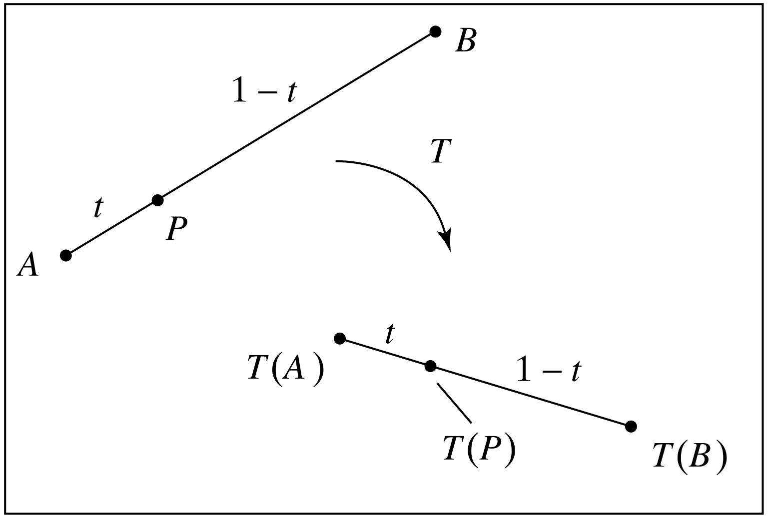
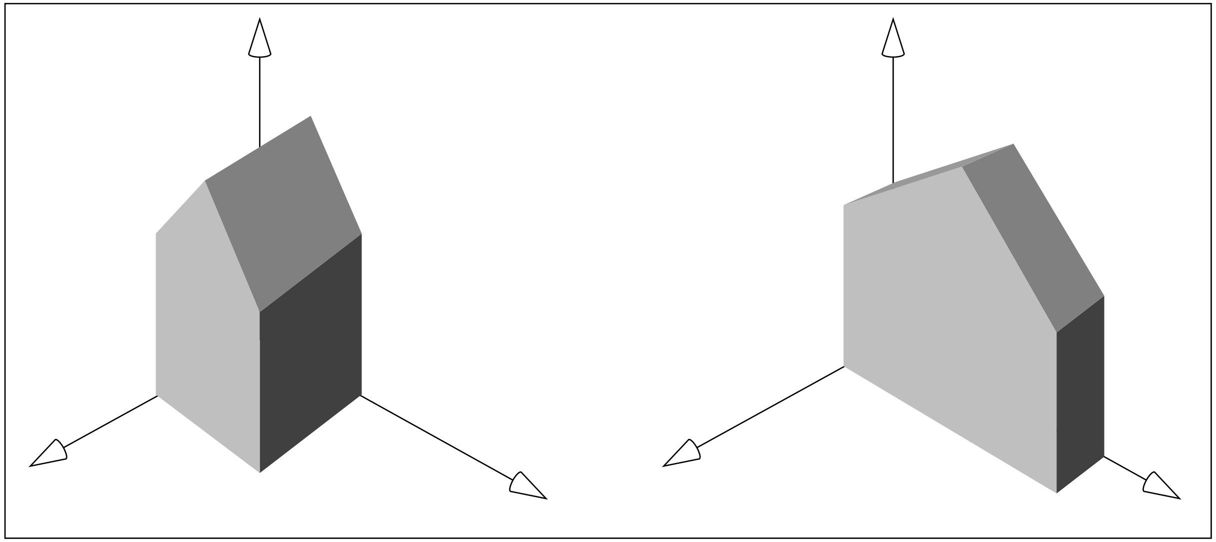


FIGURE 5.22 The transformation forms a new coordinate frame.

FIGURE 5.23 Relative ratios are preserved.



GURE 5.24 Scaling the basic
arn.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

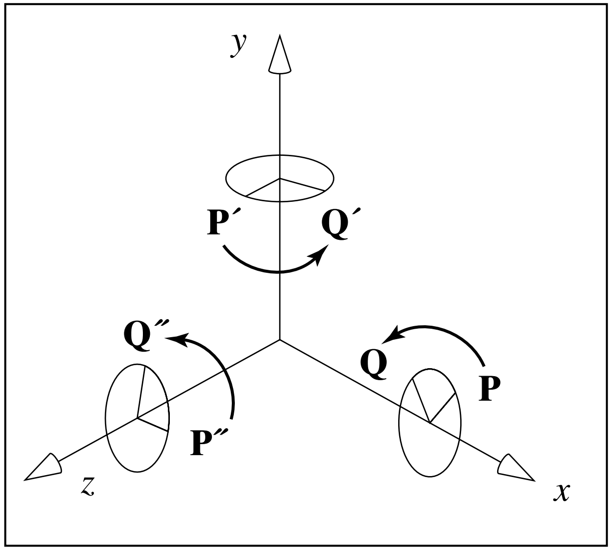


FIGURE 5.25 Positive rotations about the three axes.

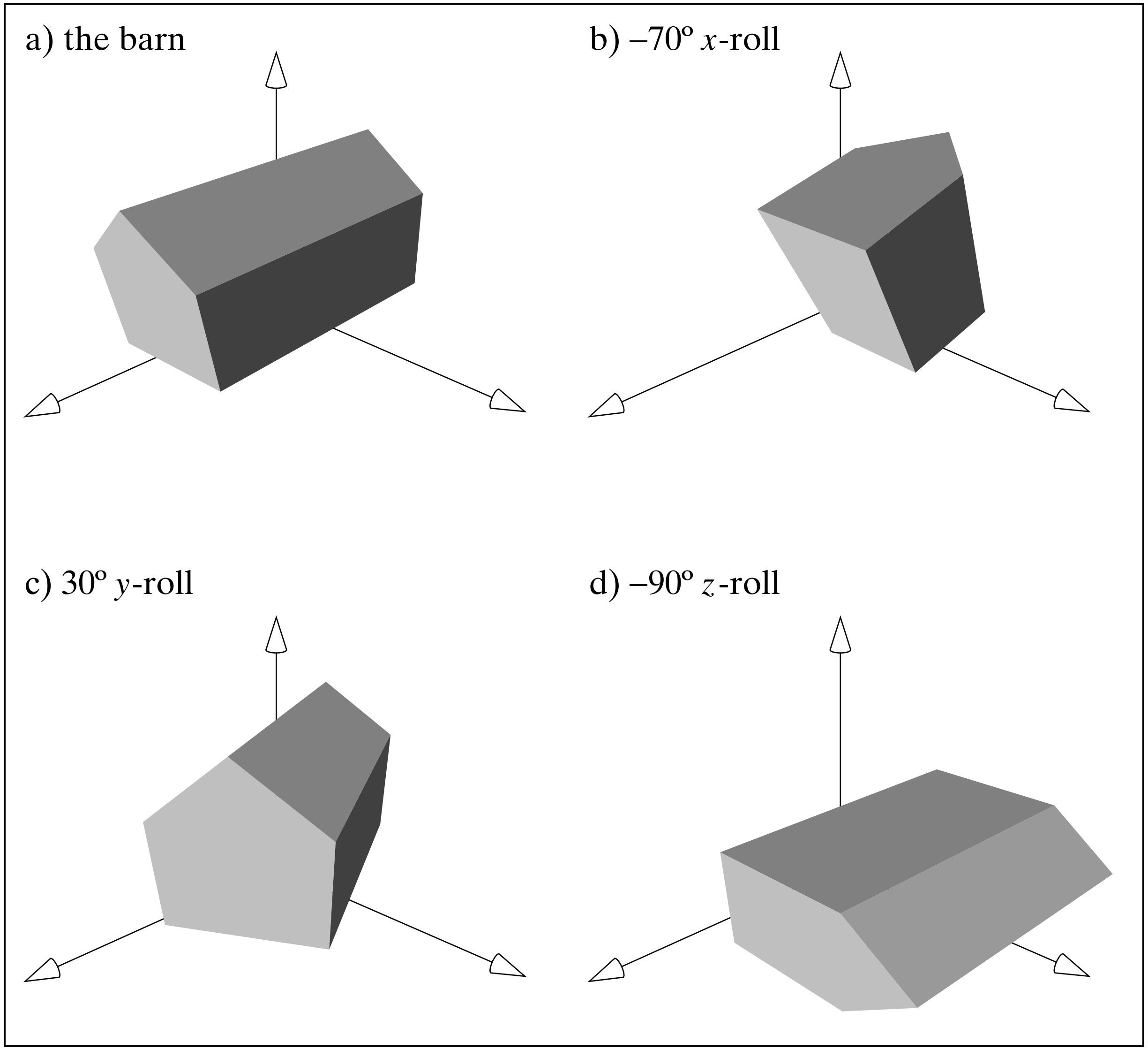
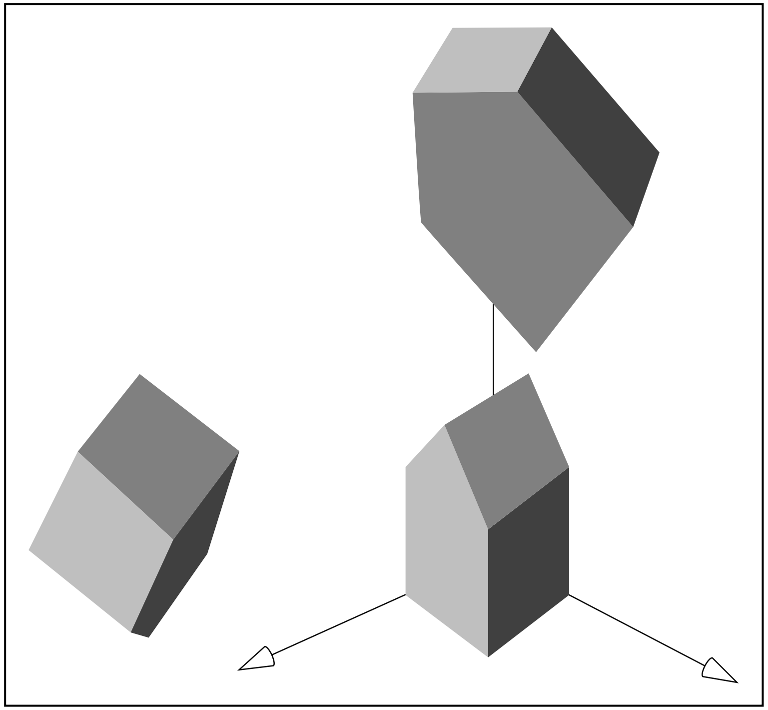


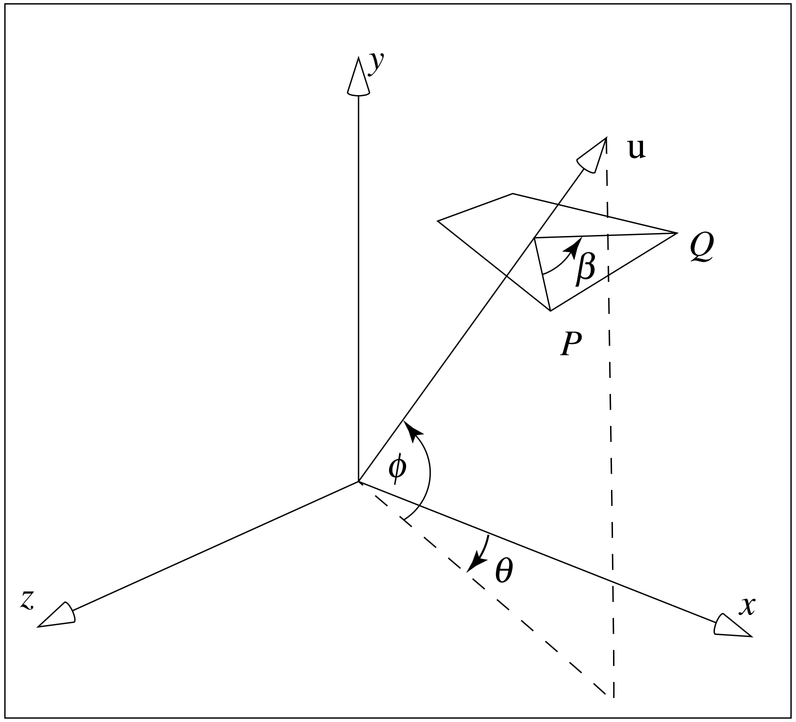
FIGURE 5.26 Rotating the basic barn.

FIGURE 5.27 Composing 3D affine transformations.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

FIGURE 5.28 Rotation about an axis through the origin.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

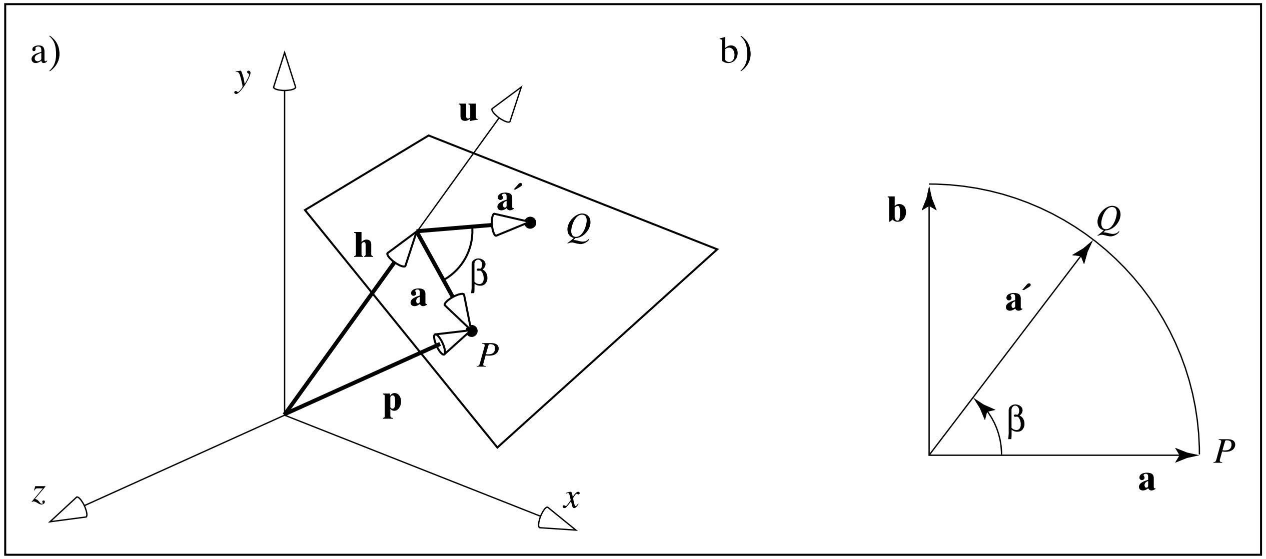


FIGURE 5.29 *P* rotates to *Q* in the plane of rotation.

FIGURE 5.30 The basic barn rotated about axis **u**.

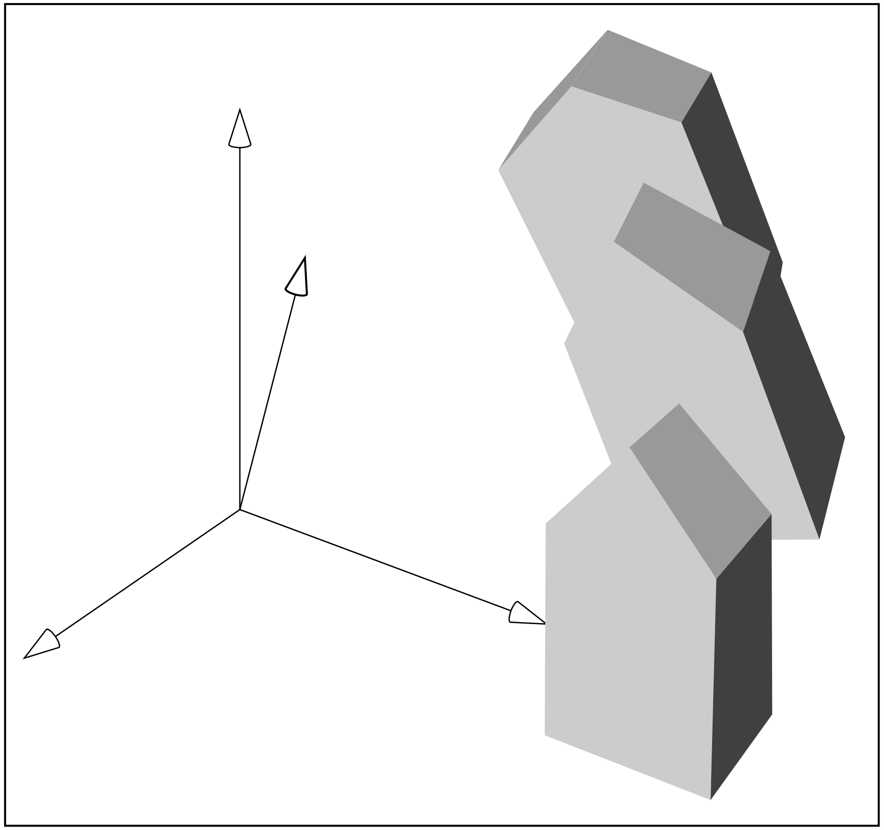
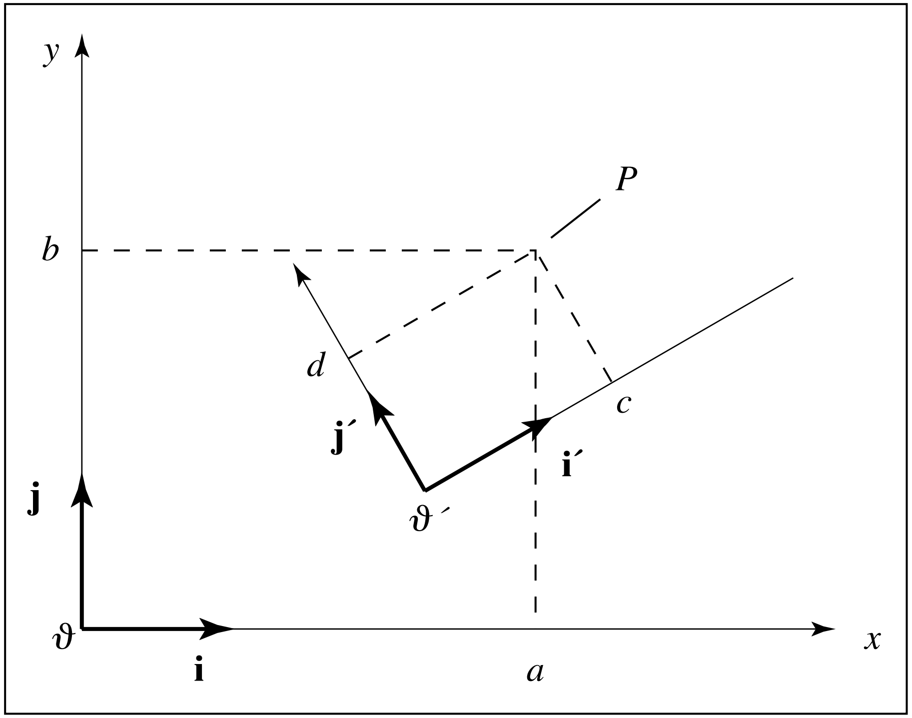


FIGURE 5.31 Transforming a coordinate frame.



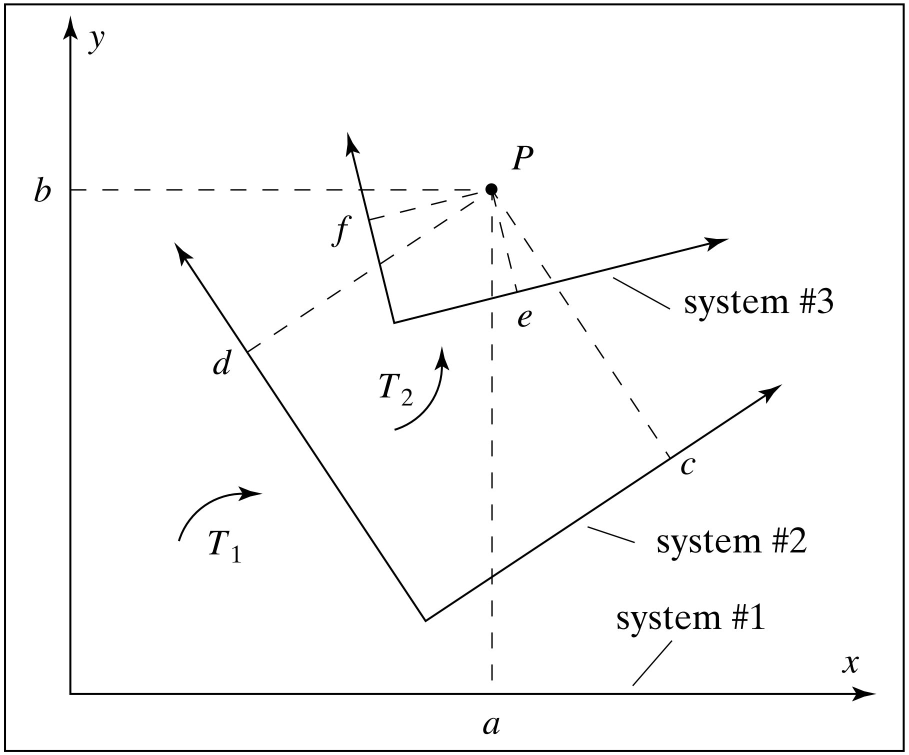


FIGURE 5.32 Transforming a coordinate system twice.

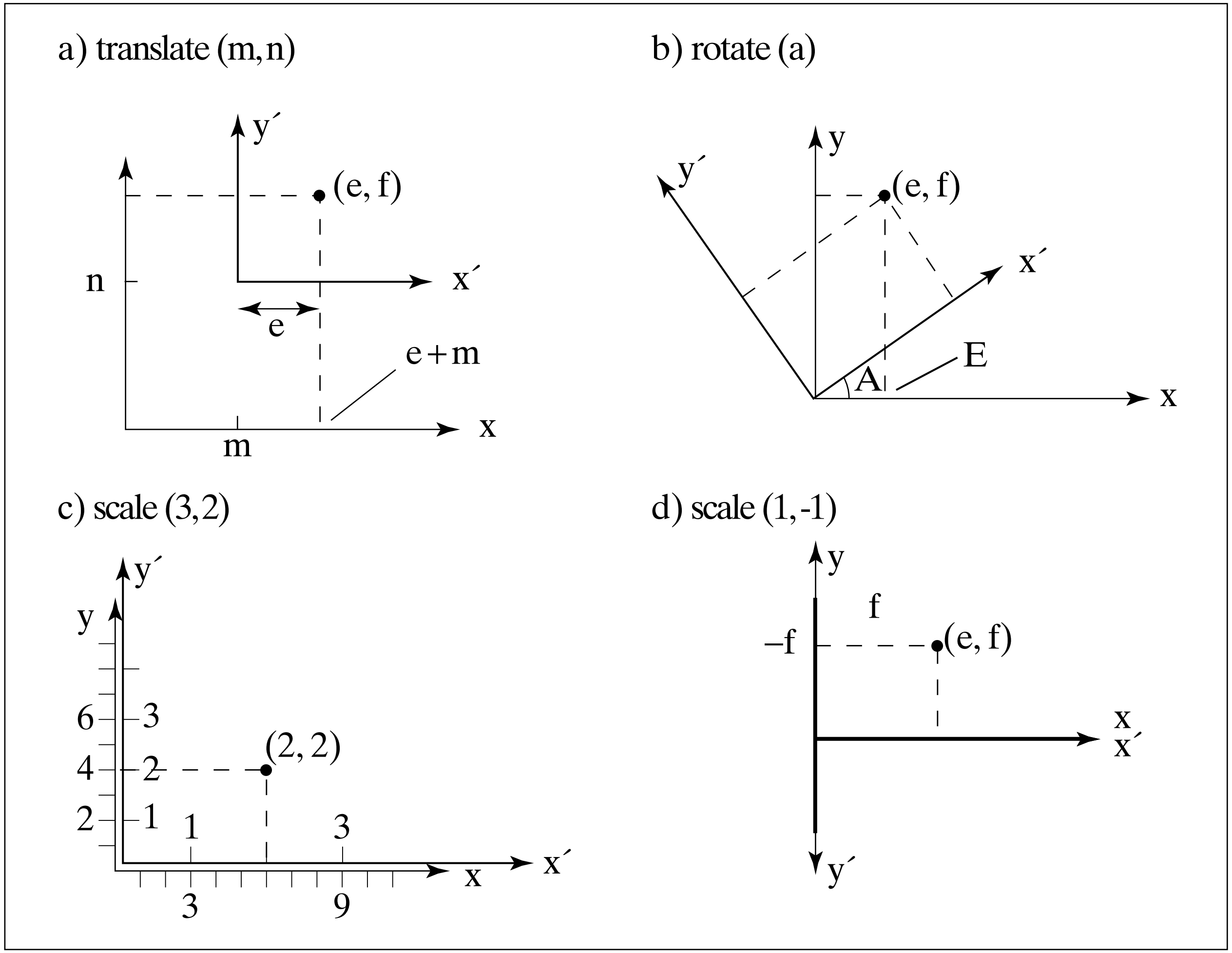
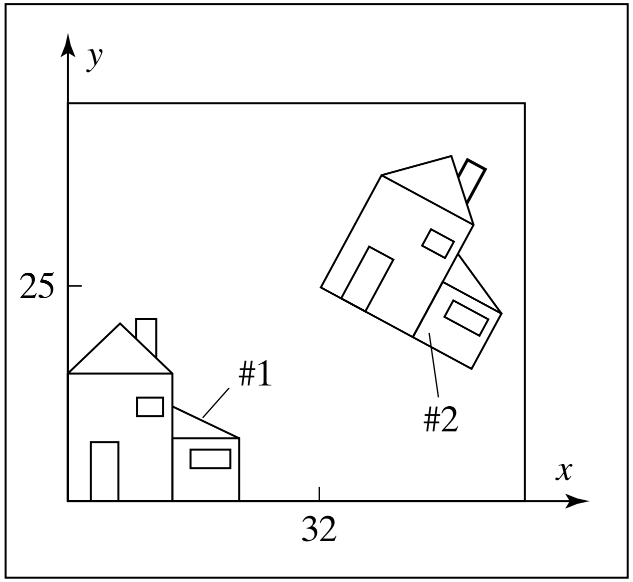


FIGURE 5.33 Elementary changes between coordinate systems.

FIGURE 5.34 Drawing a rotated and translated house.



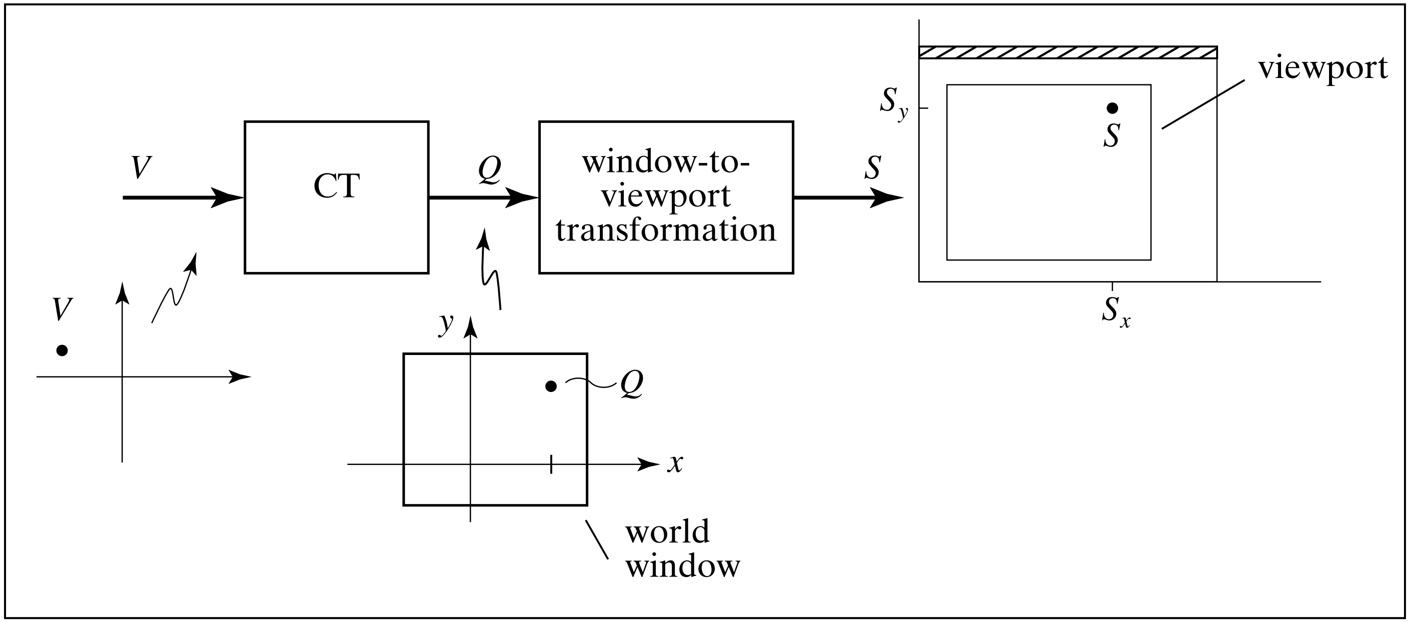


FIGURE 5.35 The current transformation is applied to vertices.

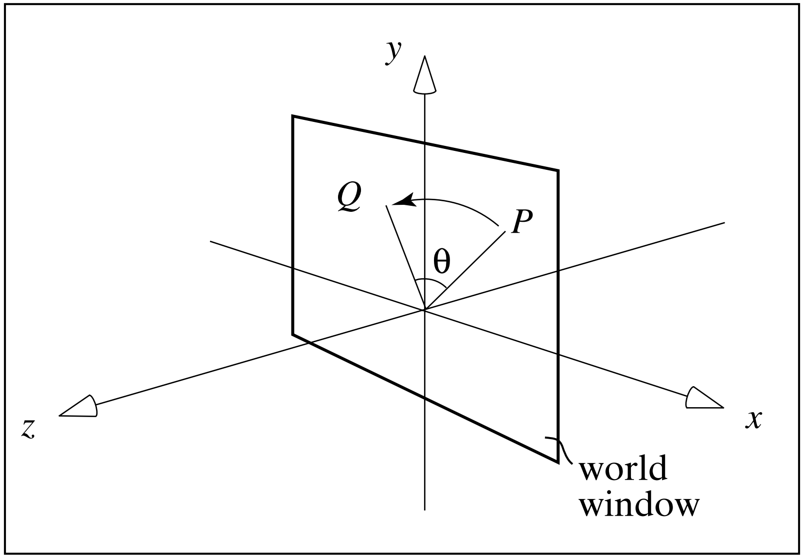


FIGURE 5.36 2D drawing takes place in the *xy*-plane.

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

```
//<<<<<<<<<<<<<<< initCT >>>>>>>>>>>>>>>>>
void Canvas:: initCT(void)
{
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();        // set CT to the identity matrix
}
//<<<<<<<<<<<<<< scale2D >>>>>>>>>>>>>>>>>>>>
void Canvas:: scale2D(double sx, double sy)
{
      glMatrixMode(GL_MODELVIEW);
      glScaled(sx, sy, 1.0); // set CT to CT * (2D scaling)
}
//<<<<<<<<<<<<<<< translate2D >>>>>>>>>>>>>>>>>
void Canvas:: translate2D(double dx, double dy)
{
      glMatrixMode(GL_MODELVIEW);
      glTranslated(dx, dy, 0); // set CT to CT * (2D translation)
}
//<<<<<<<<<<<<<<<< rotate2D >>>>>>>>>>>>>>>>>>>>
void Canvas:: rotate2D(double angle)
{
      glMatrixMode(GL_MODELVIEW);
      glRotated(angle, 0.0, 0.0, 1.0); // set CT to CT * (2D rotation)
}
```

FIGURE 5.37 Routines to manage the CT for 2D transformations.

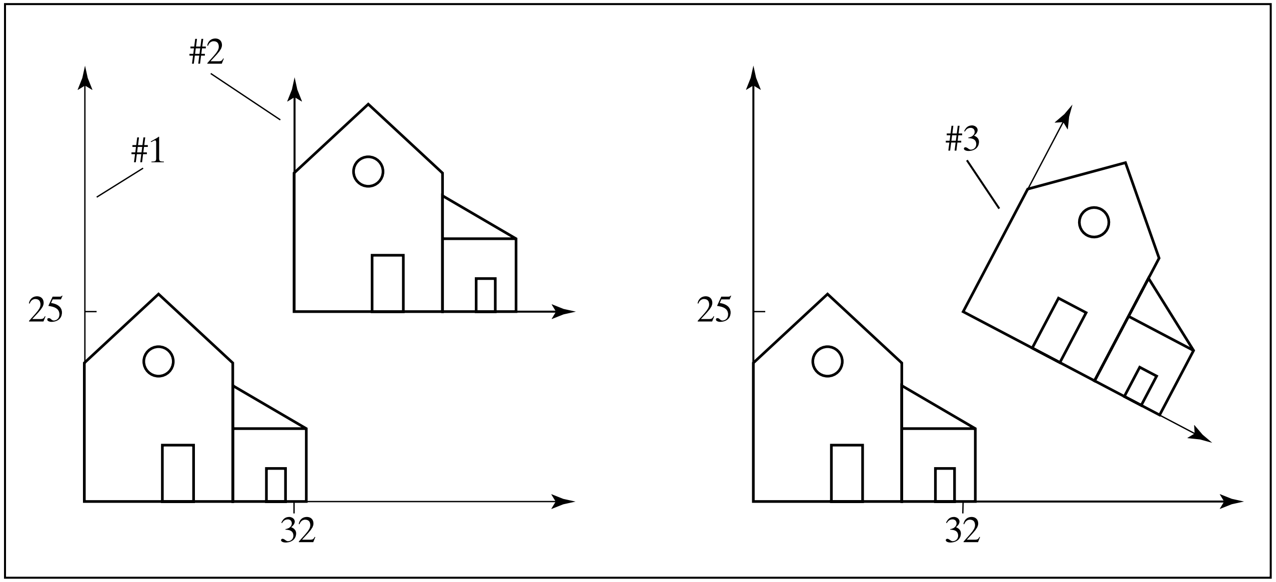


FIGURE 5.38 The same transformation viewed as a sequence of changes of coordinate systems.

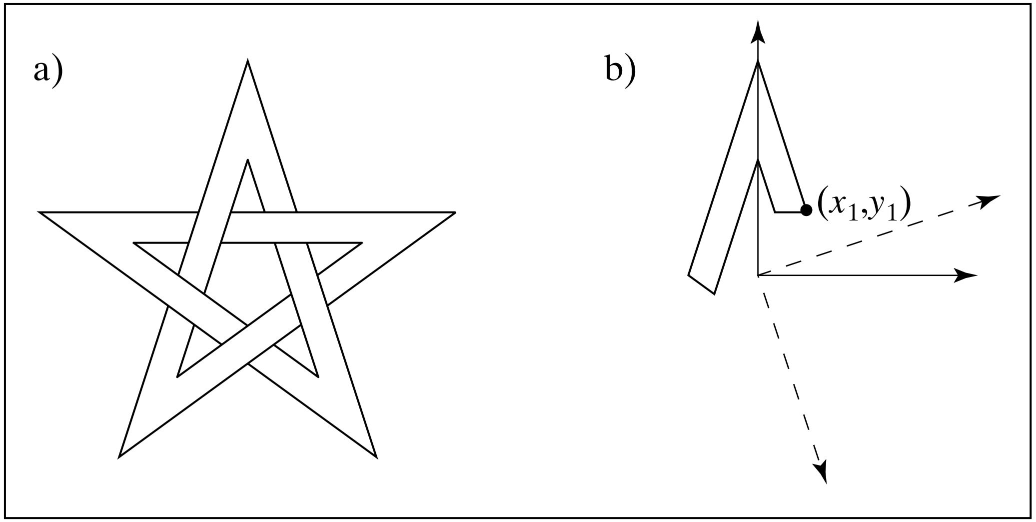
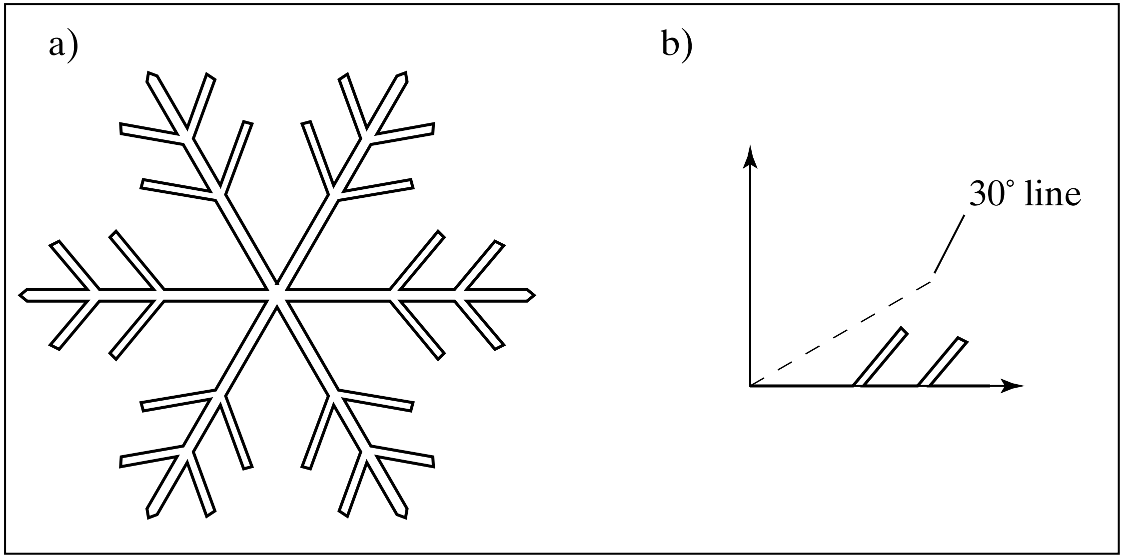


FIGURE 5.39 Using successive rotations of the coordinate system.

FIGURE 5.40 Designing a snowflake.



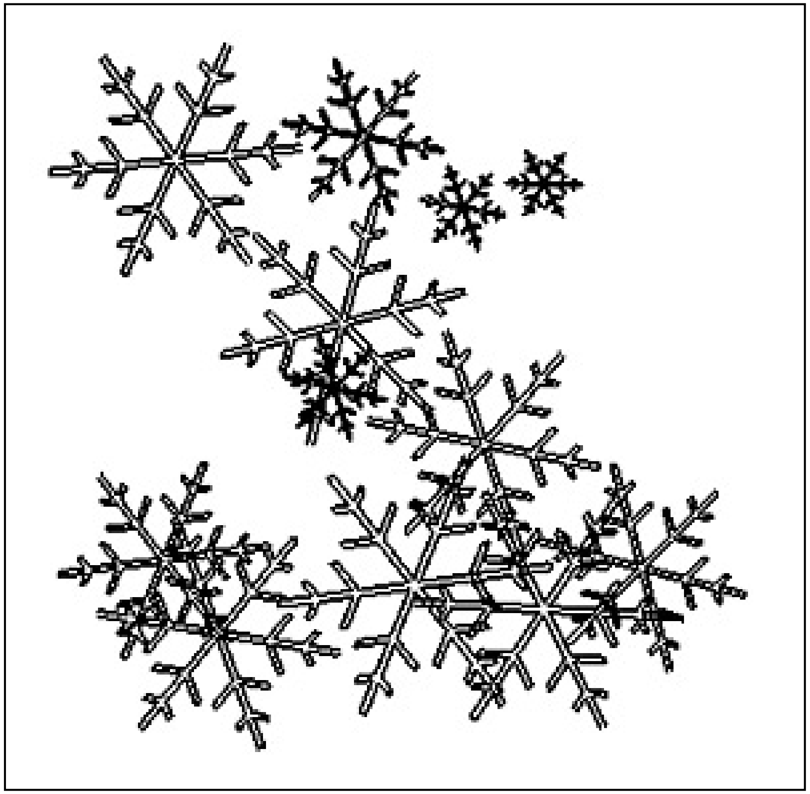
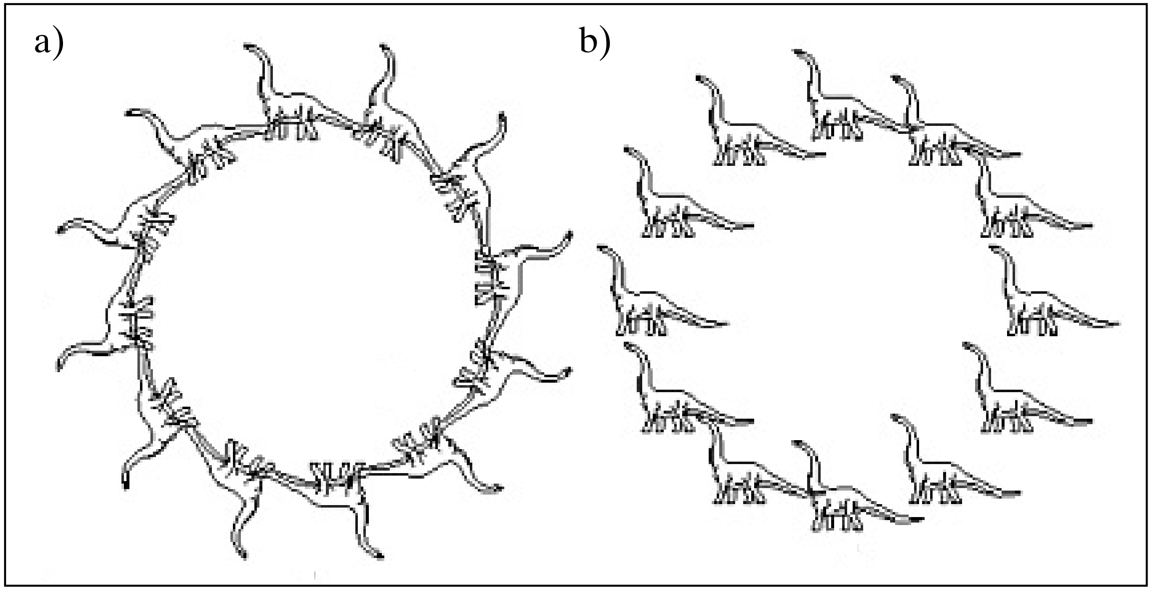


FIGURE 5.41 A flurry of snowflakes.

FIGURE 5.42 Two patterns based on a motif. (a) Each motif is rotated separately. (b) All motifs are upright.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

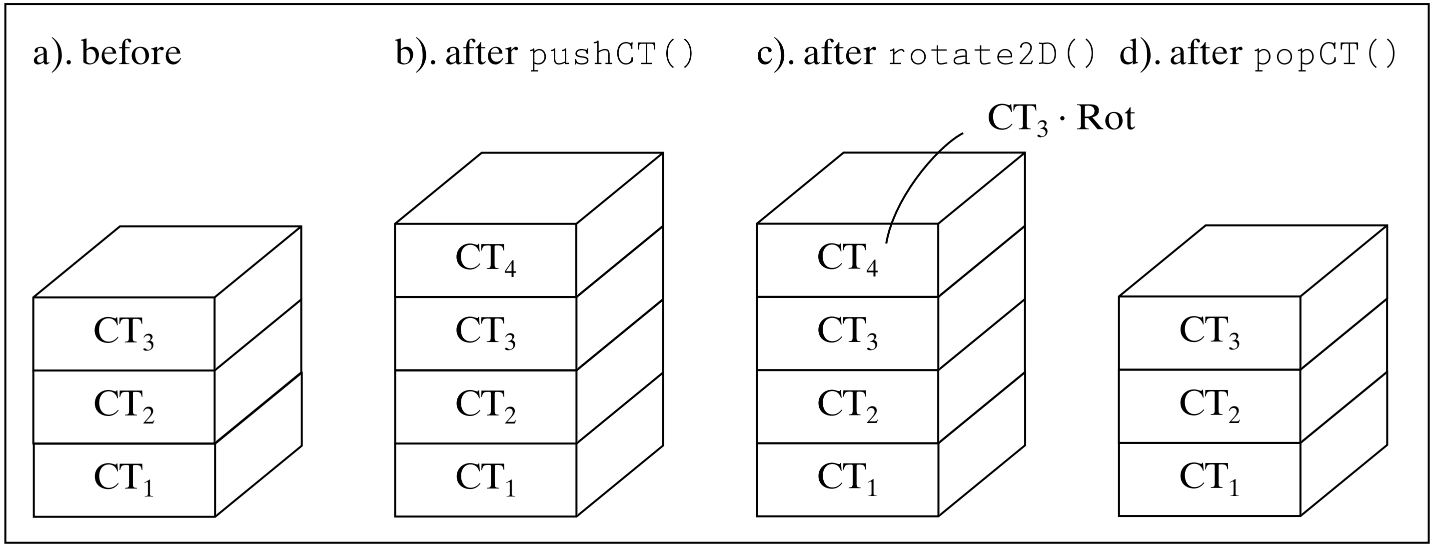


FIGURE 5.43 Manipulating a stack of CT's.

FIGURE 5.44 Routines to save and restore CT's.

```
void Canvas:: pushCT(void)
{
      glMatrixMode(GL_MODELVIEW);
      glPushMatrix();           // push a copy of the top matrix
}
void Canvas:: popCT(void)
{
      glMatrixMode(GL_MODELVIEW);
      glPopMatrix();            // pop the top matrix from the stack
}
```

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

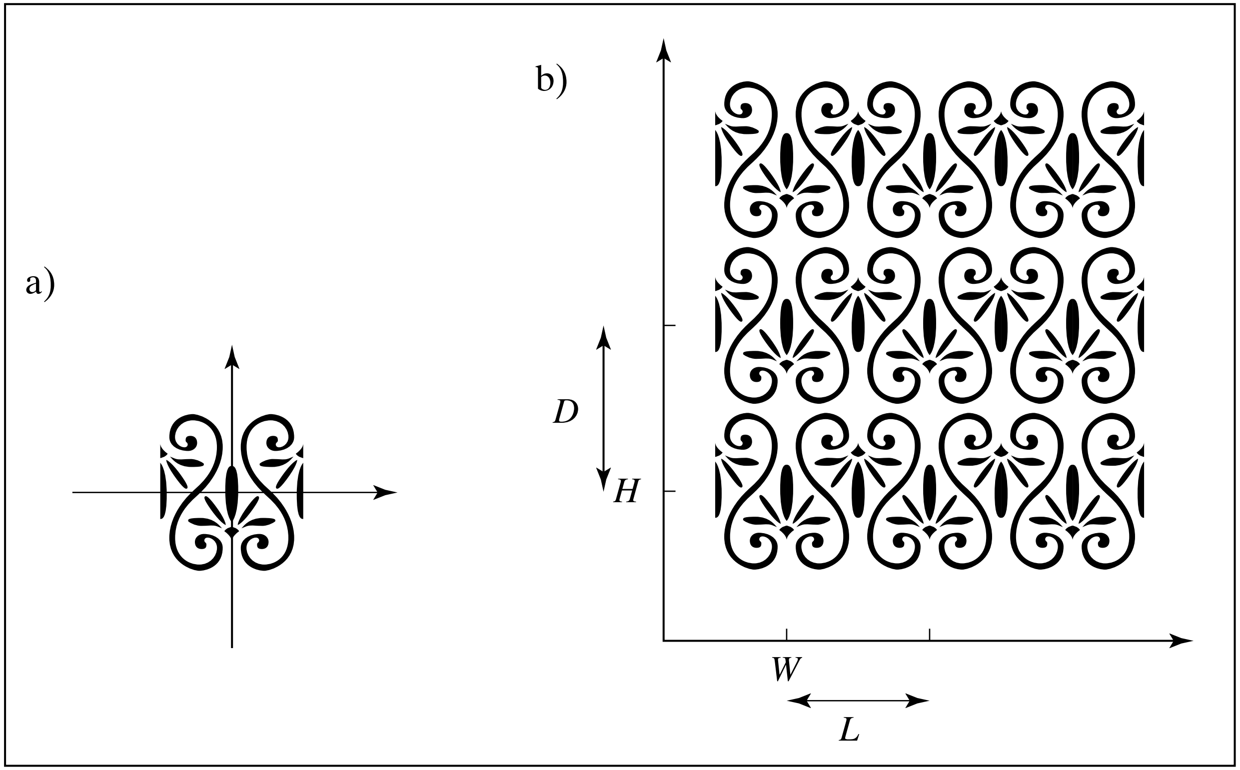


FIGURE 5.45 A tiling based on a motif. (a) The motif. (b) The tiling.

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

```
cvs.pushCT();    // so we can return here
cvs.translate2D(W, H);        // position for the first motif
for(row = 0; row < 3; row++) // draw each row
{
  cvs.pushCT();
  for(col = 0; col < 4; col++)// draw the next row of motifs
  {
      motif();
      cvs.translate2D(L, 0);  // move to the right
  }
  cvs.popCT();    // back to the start of this row
  cvs.translate2D(0, D); // move up to the next row
}
cvs.popCT();   // back to where we started
```

FIGURE 5.46 Drawing a hexagonal tiling.

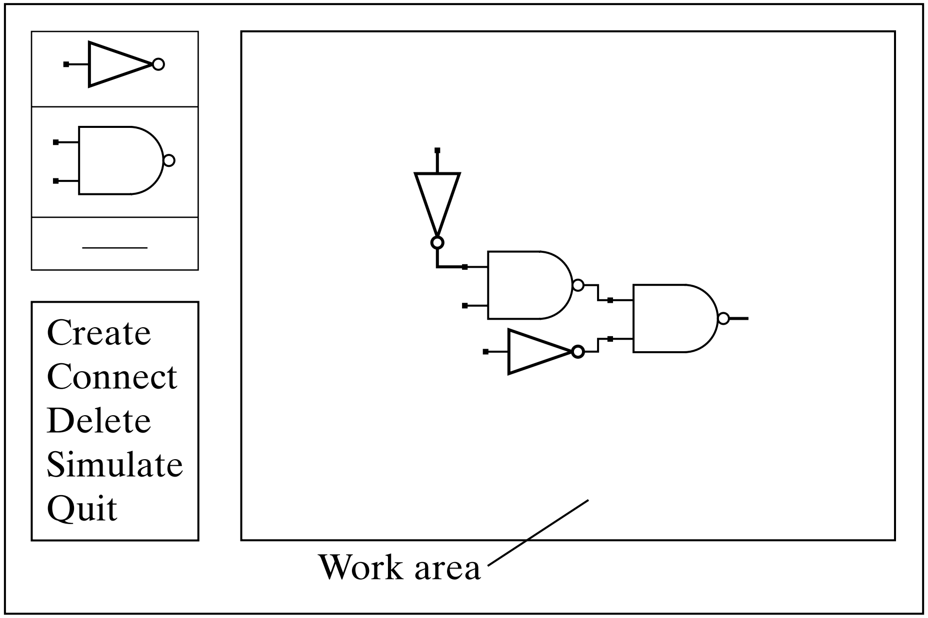


FIGURE 5.47 Creating instances in a pick-and-place application.

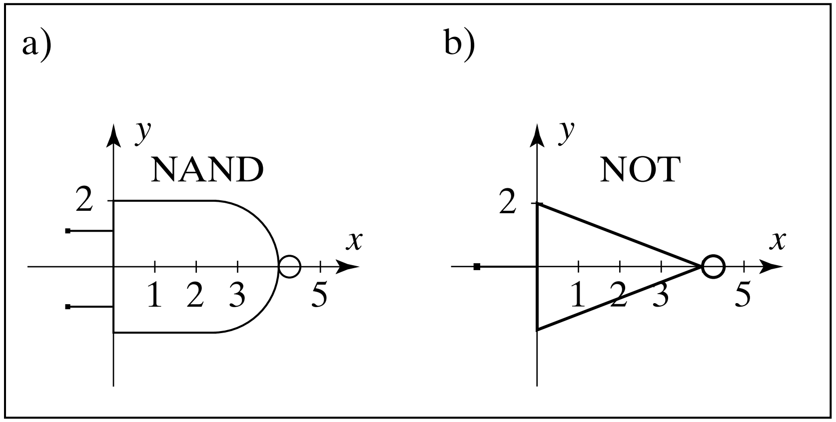
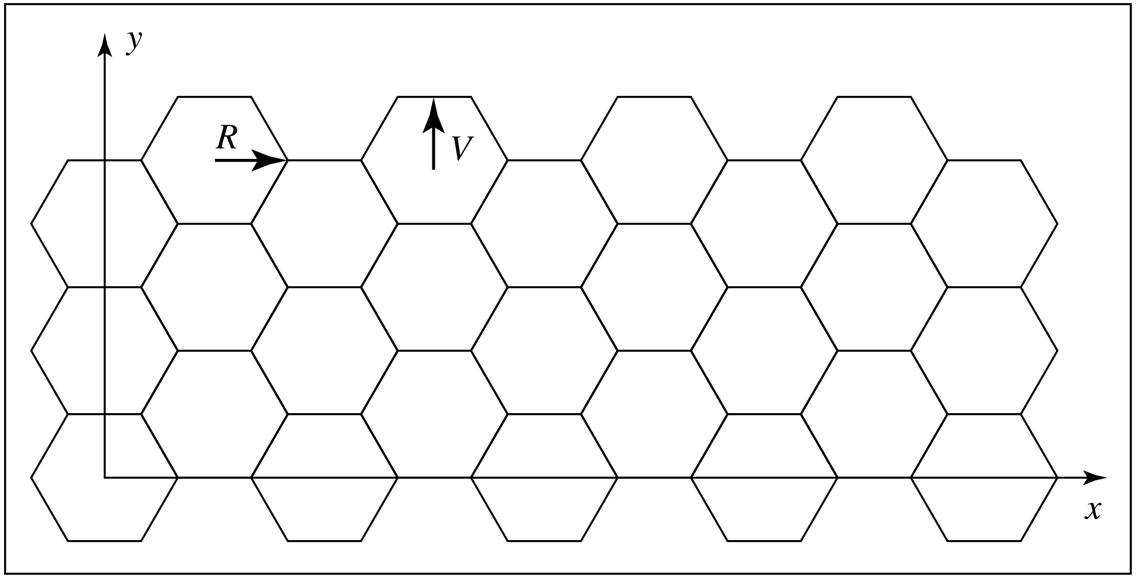


FIGURE 5.48 Each type of gate is defined in its own coordinate system.

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

FIGURE 5.49 A simple hexagonal tiling.



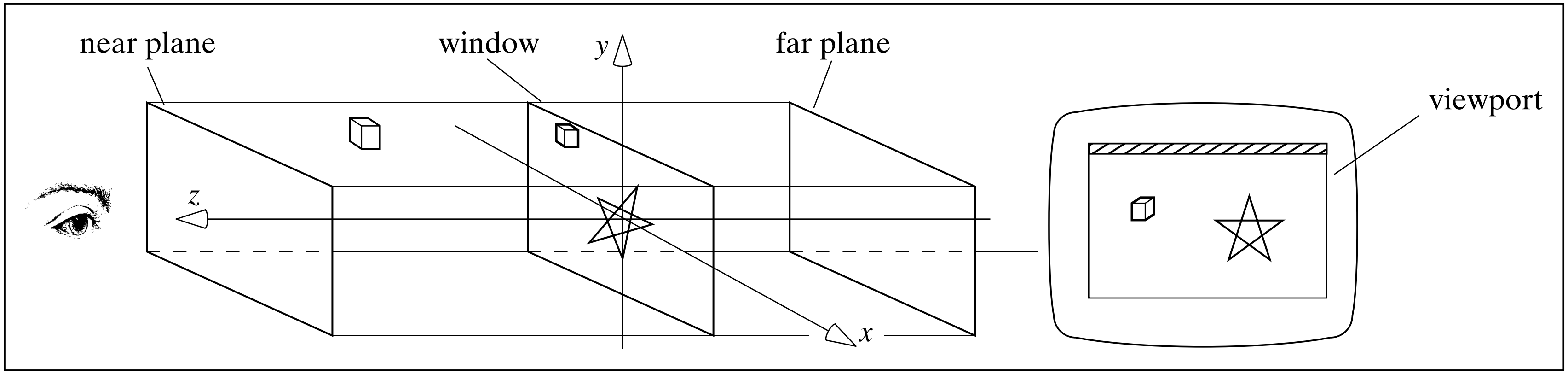


FIGURE 5.50 Simple viewing used in OpenGL for 2D drawing.

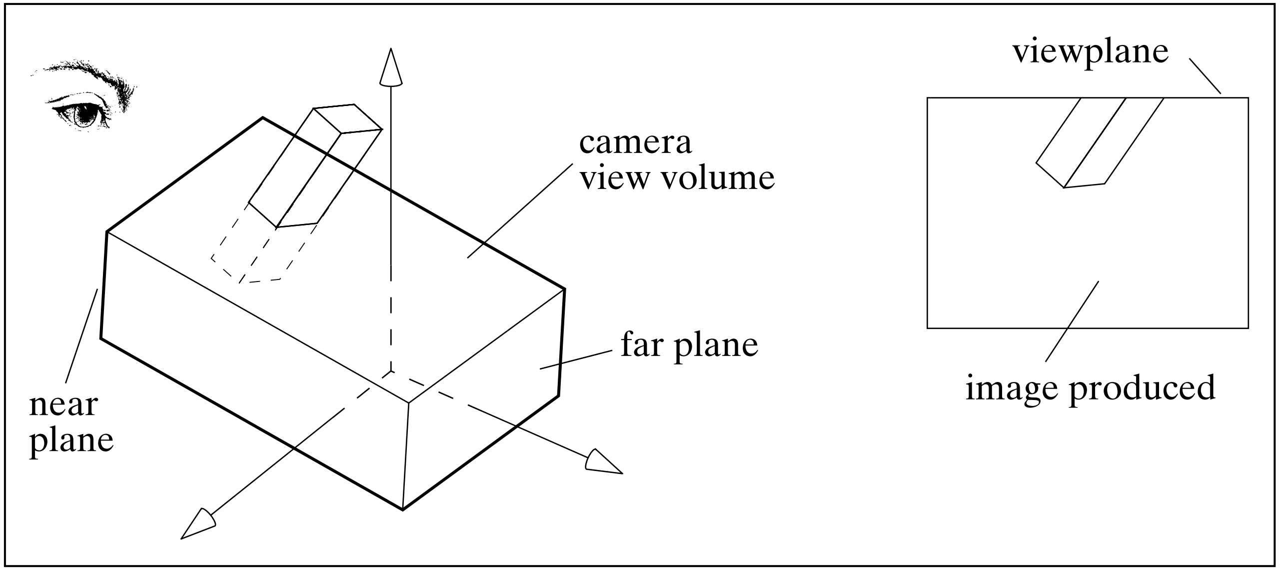
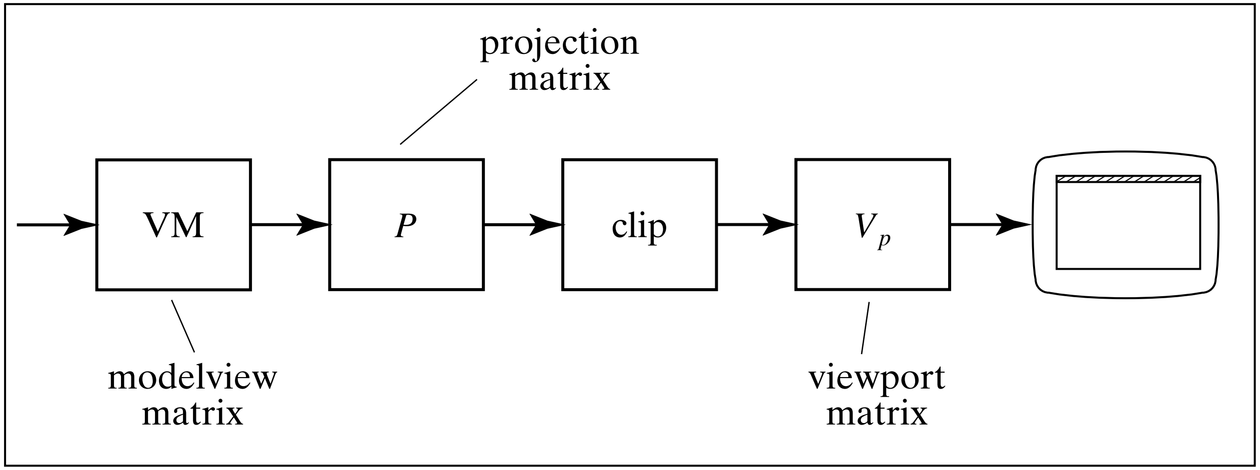


FIGURE 5.51 A camera to produce parallel views of a scene.

FIGURE 5.52 The OpenGL pipeline (slightly simplified).



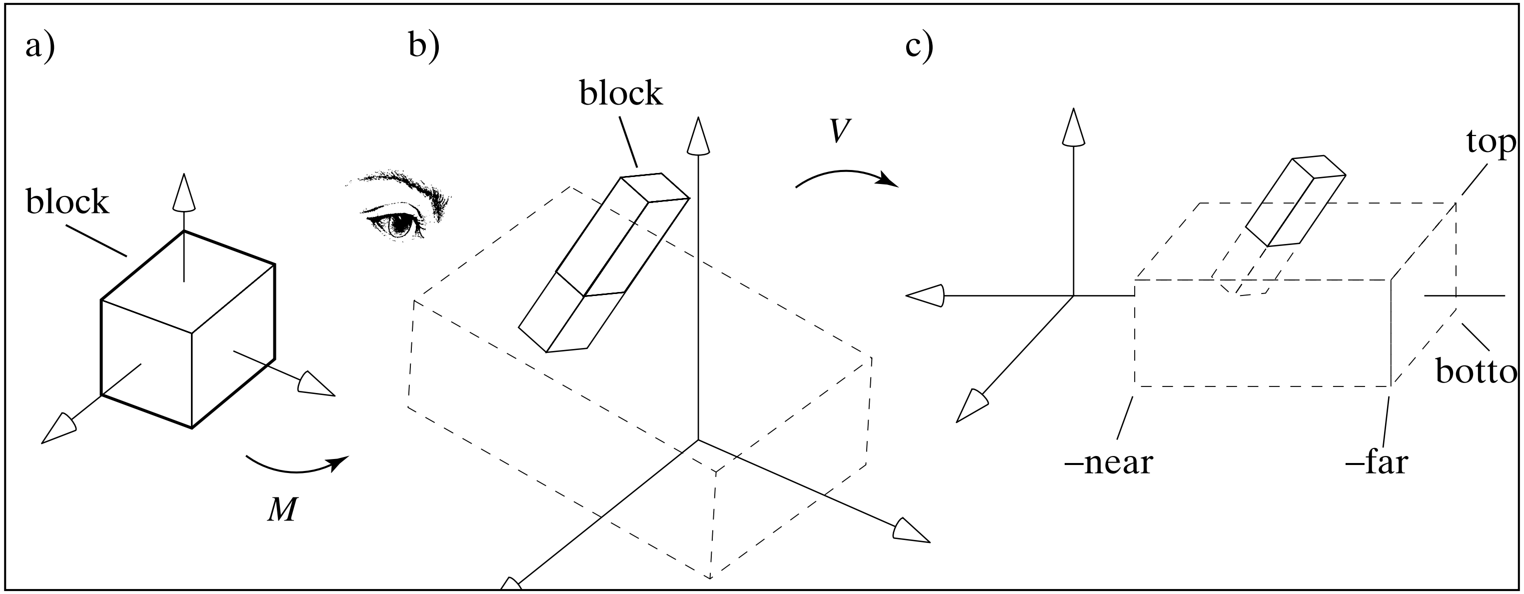


FIGURE 5.53 Effect of the modelview matrix in the graphics pipeline. (a) Before the transformations. (b) After the modeling transformation. (c) After the modelview transformation.

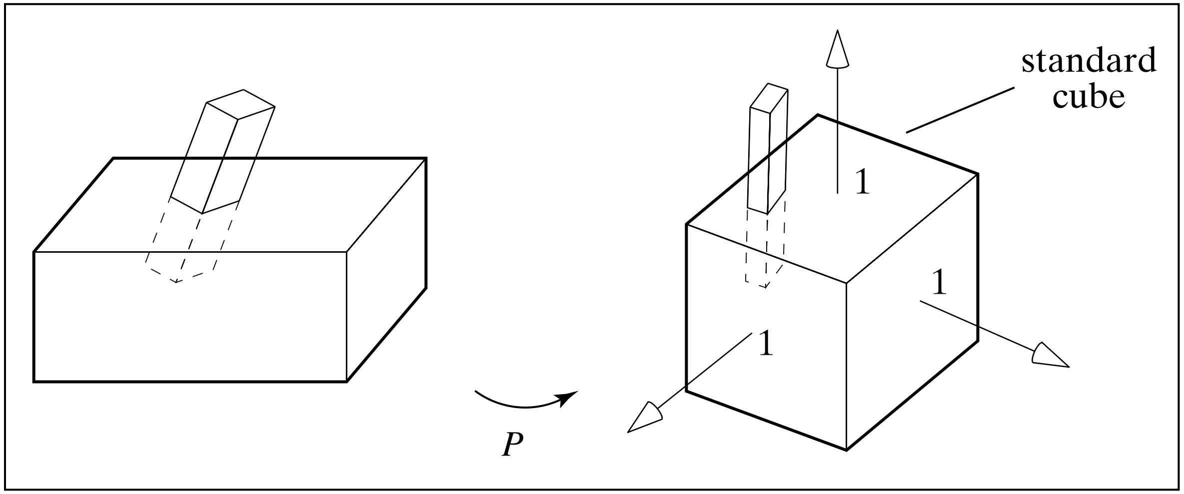
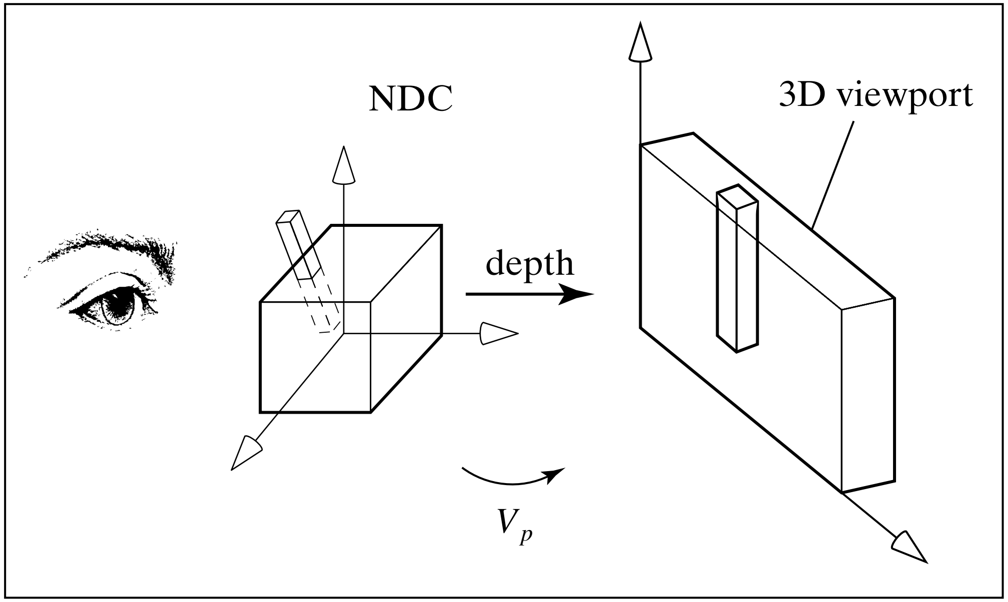


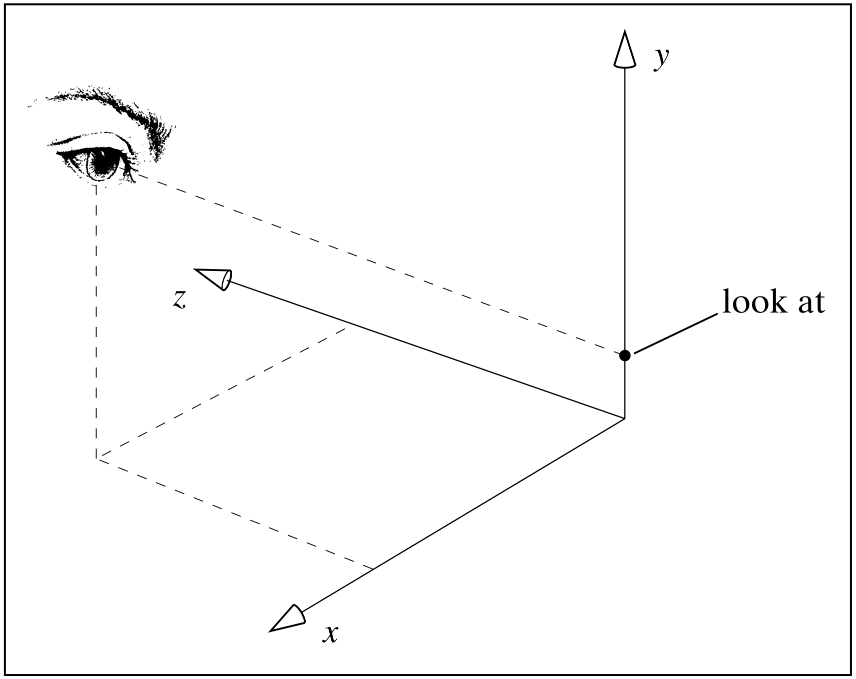
FIGURE 5.54 Effect of the projection matrix (for parallel projections).

FIGURE 5.55 Effect of the viewport transformation.



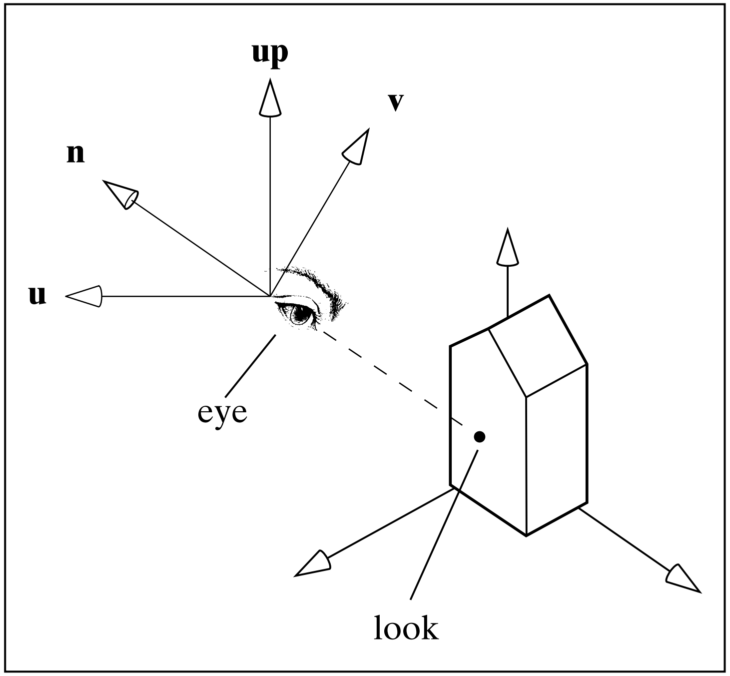
from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

FIGURE 5.56 Setting a camera with `gluLookAt()`.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

FIGURE 5.57 Converting from world to camera coordinates.



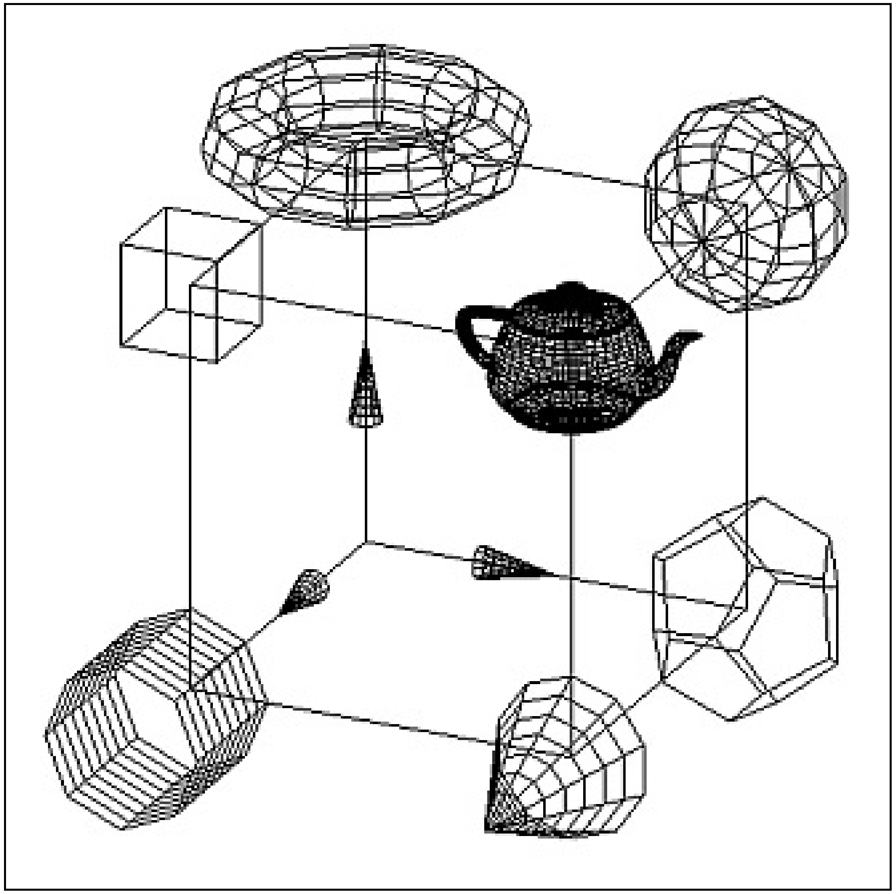


FIGURE 5.59 Wire-frame drawing of various primitive shapes.

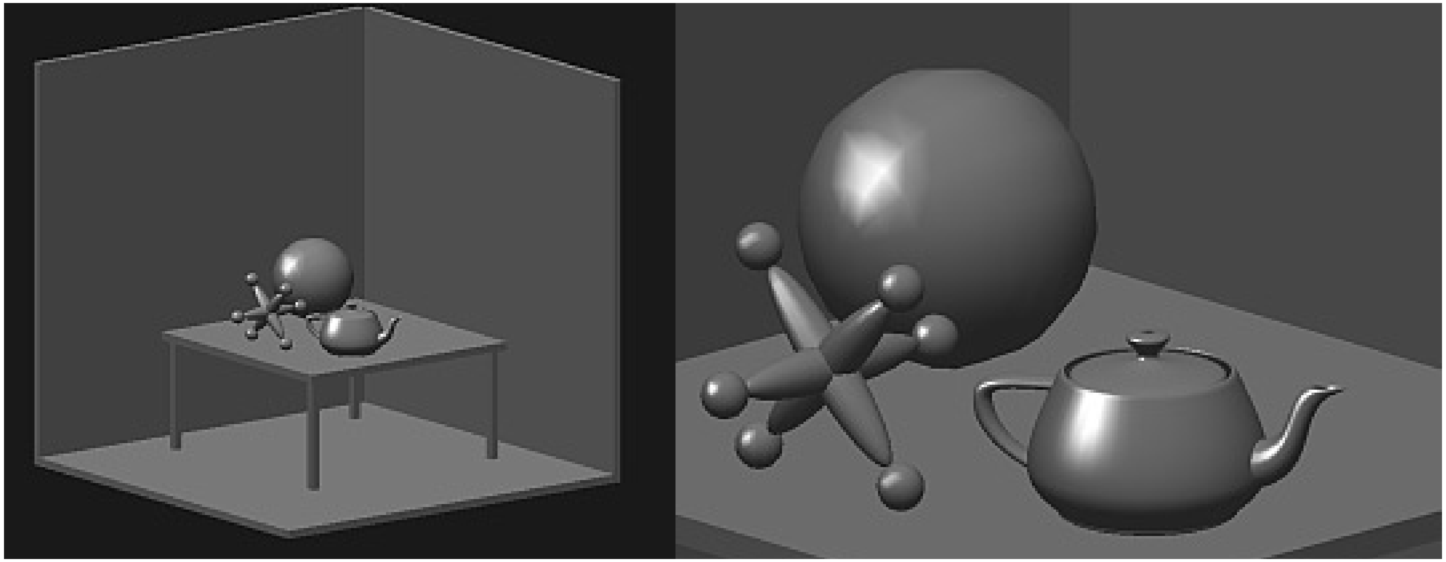
FIGURE 5.60
Complete program to draw Figure 5.59 using OpenGL.

```
#include <windows.h>  //suitable when using Windows 95/98/NT
#include <gl/Gl.h>
#include <gl/Glu.h>
#include <gl/glut.h>
//<<<<<<<<<<<<<<<<<<< axis >>>>>>>>>>>>>>
void axis(double length)
{ // draw a z-axis, with cone at end
      glPushMatrix();
      glBegin(GL_LINES);
         glVertex3d(0, 0, 0); glVertex3d(0,0,length); // along the z-axis
      glEnd();
      glTranslated(0, 0,length -0.2);
      glutWireCone(0.04, 0.2, 12, 9);
      glPopMatrix();
}
//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< displayWire >>>>>>>>>>>>>>>>>>>>>>
void displayWire(void)
{
      glMatrixMode(GL_PROJECTION); // set the view volume shape
      glLoadIdentity();
      glOrtho(-2.0*64/48.0, 2.0*64/48.0, -2.0, 2.0, 0.1, 100);
      glMatrixMode(GL_MODELVIEW); // position and aim the camera
      glLoadIdentity();
      gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

      glClear(GL_COLOR_BUFFER_BIT); // clear the screen
      glColor3d(0,0,0: // draw black lines
      axis(0.5);      // z-axis
      glPushMatrix();
      glRotated(90, 0,1.0, 0);
      axis(0.5);     // y-axis
      glRotated(-90.0, 1, 0, 0);
      axis(0.5);     // z-axis
      glPopMatrix();

      glPushMatrix();
      glTranslated(0.5, 0.5, 0.5); // big cube at (0.5, 0.5, 0.5)
      glutWireCube(1.0);
      glPopMatrix();

      glPushMatrix();
      glTranslated(1.0,1.0,0); // sphere at (1,1,0)
      glutWireSphere(0.25, 10, 8);
      glPopMatrix();

      glPushMatrix();
      glTranslated(1.0,0,1.0); // cone at (1,0,1)
      glutWireCone(0.2, 0.5, 10, 8);
      glPopMatrix();

      glPushMatrix();
      glTranslated(1,1,1);
      glutWireTeapot(0.2); // teapot at (1,1,1)
      glPopMatrix();
```

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

FIGURE 5.60 (*Continued*)

```
      glPushMatrix();
      glTranslated(0, 1.0 ,0); // torus at (0,1,0)
      glRotated(90.0, 1,0,0);
      glutWireTorus(0.1, 0.3, 10,10);
      glPopMatrix();

      glPushMatrix();
      glTranslated(1.0, 0 ,0); // dodecahedron at (1,0,0)
      glScaled(0.15, 0.15, 0.15);
      glutWireDodecahedron();
      glPopMatrix();

      glPushMatrix();
      glTranslated(0, 1.0 ,1.0); // small cube at (0,1,1)
      glutWireCube(0.25);
      glPopMatrix();

      glPushMatrix();
      glTranslated(0, 0 ,1.0); // cylinder at (0,0,1)
      GLUquadricObj * qobj;
      qobj = gluNewQuadric();
      gluQuadricDrawStyle(qobj,GLU_LINE);
      gluCylinder(qobj, 0.2, 0.2, 0.4, 8,8);
      glPopMatrix();
      glFlush();
}
//<<<<<<<<<<<<<<<<<<<<<<< main >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
void main(int argc, char **argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
      glutInitWindowSize(640,480);
      glutInitWindowPosition(100, 100);
      glutCreateWindow("Transformation testbed - wire frames");
      glutDisplayFunc(displayWire);
      glClearColor(1.0f, 1.0f, 1.0f,0.0f);  // background is white
      glViewport(0, 0, 640, 480);
      glutMainLoop();
}
```



a) b)

FIGURE 5.61 A simple 3D scene (a) using a large view volume and (b) using a small view volume.

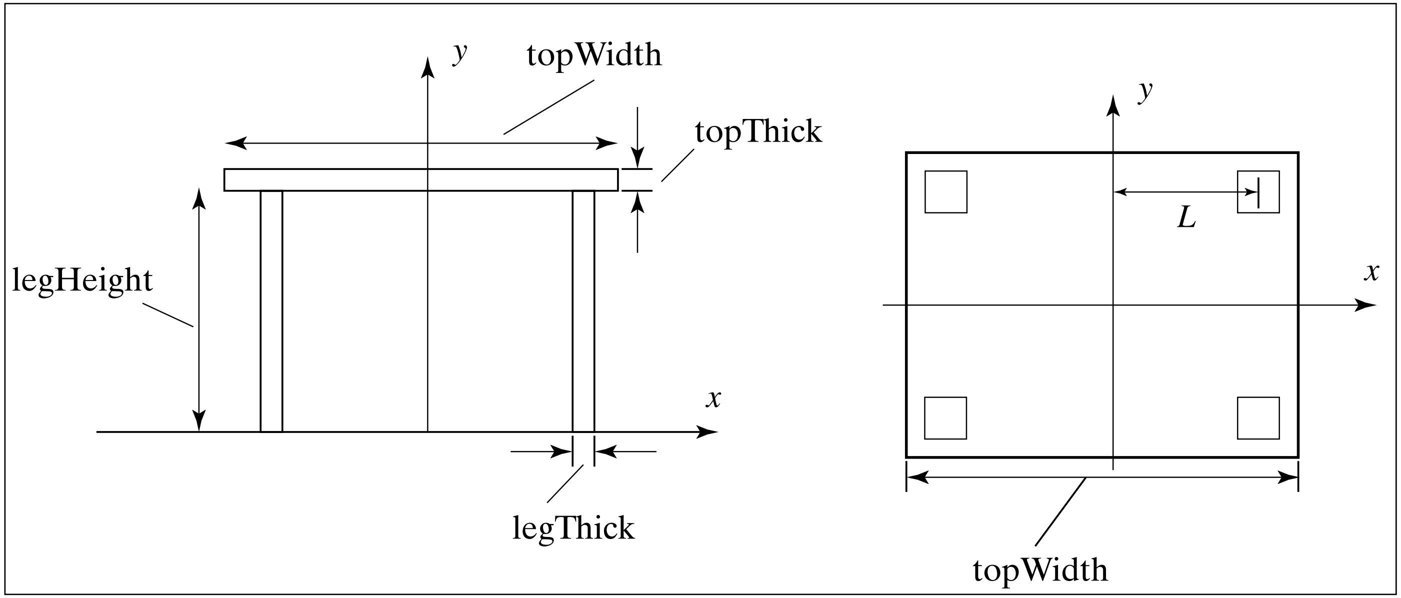


FIGURE 5.62 Designing the table.

FIGURE 5.63 Complete program to draw the shaded scene.

```
#include <windows.h>
#include <iostream.h>
#include <gl/Gl.h>
#include <gl/Glu.h>
#include <gl/glut.h>
//<<<<<<<<<<<<<<< wall >>>>>>>>>>>>>>>>
void wall(double thickness)
{ // draw thin wall with top = xz-plane, corner at origin
      glPushMatrix();
      glTranslated(0.5, 0.5 * thickness,  0.5);
      glScaled(1.0, thickness, 1.0);
      glutSolidCube(1.0);
      glPopMatrix();
}
//<<<<<<<<<<<<<<<<<< tableLeg >>>>>>>>>>>>>>>>>>>
void tableLeg(double thick, double len)
{
      glPushMatrix();
      glTranslated(0, len/2, 0);
      glScaled(thick, len, thick);
      glutSolidCube(1.0);
      glPopMatrix();
}
//<<<<<<<<<<<<<<<<<<<<< jack part >>>>>>>>>>>>>
void jackPart()
{ // draw one axis of the unit jack - a stretched sphere
      glPushMatrix();
      glScaled(0.2,0.2,1.0);
      glutSolidSphere(1,15,15);
      glPopMatrix();
      glPushMatrix();
      glTranslated(0,0,1.2); // ball on one end
      glutSolidSphere(0.2,15,15);
      glTranslated(0,0, -2.4);
      glutSolidSphere(0.2,15,15); // ball on the other end
      glPopMatrix();
}
//<<<<<<<<<<<<<<<<<<< jack >>>>>>>>>>>>>>>>>>>>
void jack()
{ // draw a unit jack out of spheroids
      glPushMatrix();
      jackPart();
      glRotated(90.0, 0, 1, 0);
      jackPart();
      glRotated(90.0, 1,0,0);
      jackPart();
      glPopMatrix();
}
```

FIGURE 5.63
(*Continued*)

```
//<<<<<<<<<<<<<<<<<<<<<<<< table >>>>>>>>>>>>>>>>>>>>
void table(double topWid, double topThick, double legThick, double legLen)
{ // draw the table - a top and four legs
      glPushMatrix(); // draw the table top
      glTranslated(0, legLen, 0);
      glScaled(topWid, topThick, topWid);
      glutSolidCube(1.0);
      glPopMatrix();
      double dist = 0.95 * topWid/2.0 - legThick / 2.0;
      glPushMatrix();
      glTranslated(dist, 0, dist);
      tableLeg(legThick, legLen);
      glTranslated(0, 0, -2 * dist);
      tableLeg(legThick, legLen);
      glTranslated(-2 * dist, 0, 2*dist);
      tableLeg(legThick, legLen);
      glTranslated(0, 0, -2*dist);
      tableLeg(legThick, legLen);
      glPopMatrix();
}
//<<<<<<<<<<<<<<<<<<<<< displaySolid >>>>>>>>>>>>>>>>>>>>>>
void displaySolid(void)
{
 //set properties of the surface material
      GLfloat mat_ambient[] = { 0.7f, 0.7f, 0.7f, 1.0f}; // gray
      GLfloat mat_diffuse[] = {0.6f, 0.6f, 0.6f, 1.0f};
      GLfloat mat_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
      GLfloat mat_shininess[] = {50.0f};
      glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient);
      glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
      glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
      glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);
      // set the light source properties
      GLfloat lightIntensity[] = {0.7f, 0.7f, 0.7f, 1.0f};
      GLfloat light_position[] = {2.0f, 6.0f, 3.0f, 0.0f};
      glLightfv(GL_LIGHT0, GL_POSITION, light_position);
      glLightfv(GL_LIGHT0, GL_DIFFUSE, lightIntensity);
      // set the camera
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      double winHt = 1.0; // half-height of the window
      glOrtho(-winHt*64/48.0, winHt*64/48.0, -winHt, winHt, 0.1, 100.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      gluLookAt(2.3, 1.3, 2, 0, 0.25, 0, 0.0,1.0,0.0);
// start drawing
      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT); // clear the screen
      glPushMatrix();
      glTranslated(0.4, 0.4, 0.6);
      glRotated(45,0,0,1);
      glScaled(0.08, 0.08, 0.08);
```

FIGURE 5.63 (*Continued*)

```
jack();    // draw the jack
      glPopMatrix();
      glPushMatrix();
      glTranslated(0.6, 0.38, 0.5);
      glRotated(30,0,1,0);
      glutSolidTeapot(0.08);        // draw the teapot
      glPopMatrix();
      glPushMatrix();
      glTranslated(0.25, 0.42, 0.35);// draw the sphere
      glutSolidSphere(0.1, 15, 15);
      glPopMatrix();
      glPushMatrix();
      glTranslated(0.4, 0, 0.4);
      table(0.6, 0.02, 0.02, 0.3); // draw the table
      glPopMatrix();
      wall(0.02);                  // wall #1: in xz-plane
      glPushMatrix();
      glRotated(90.0, 0.0, 0.0, 1.0);
      wall(0.02);                  // wall #2: in yz-plane
      glPopMatrix();
      glPushMatrix();
      glRotated(-90.0,1.0, 0.0, 0.0);
      wall(0.02);                  // wall #3: in xy-plane
      glPopMatrix();
      glFlush();
}
//<<<<<<<<<<<<<<<<<<<<<< main >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
void main(int argc, char **argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB| GLUT_DEPTH);
      glutInitWindowSize(640,480);
      glutInitWindowPosition(100, 100);
      glutCreateWindow("shaded example - 3D scene");
      glutDisplayFunc(displaySolid);
      glEnable(GL_LIGHTING); // enable the light source
      glEnable(GL_LIGHT0);
      glShadeModel(GL_SMOOTH);
      glEnable(GL_DEPTH_TEST); // for removal of hidden surfaces
      glEnable(GL_NORMALIZE); // normalize vectors for proper shading
      glClearColor(0.1f,0.1f,0.1f,0.0f);  // background is light gray
      glViewport(0, 0, 640, 480);
      glutMainLoop();
}
```

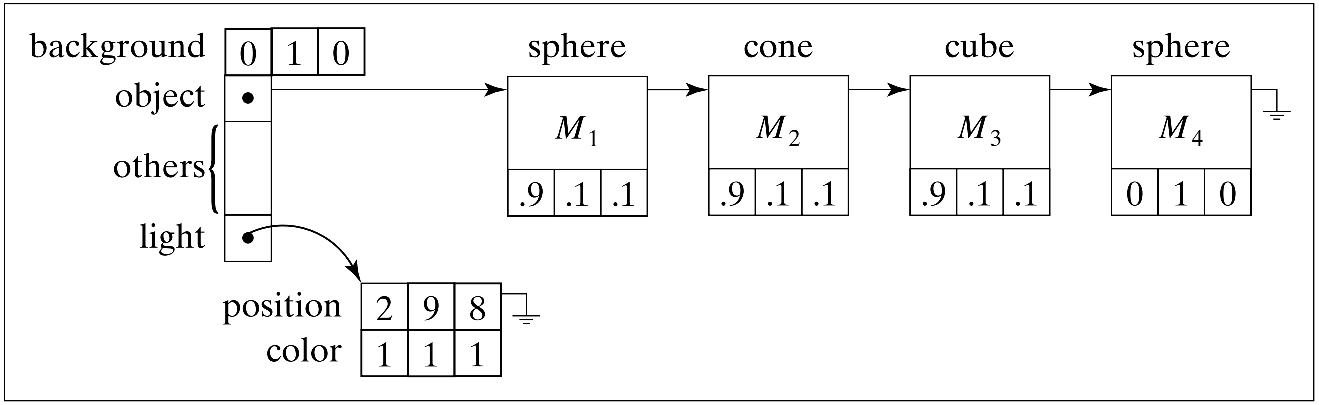


FIGURE 5.64 An object of the `Scene` class.

```
void Sphere :: drawOpenGL()
{
  tellMaterialsGL();  //pass material data to OpenGL
  glPushMatrix();
  glMultMatrixf(transf.m); // load this object's matrix
  glutSolidSphere(1.0,10,12); // draw a sphere
  glPopMatrix();
}
void Cone :: drawOpenGL()
{
  tellMaterialsGL();//pass material data to OpenGL
  glPushMatrix();
  glMultMatrixf(transf.m); // load this object's matrix
  glutSolidCone(1.0,1.0, 10,12); // draw a cone
  glPopMatrix();
}
```

FIGURE 5.65 The `drawOpenGL()` methods for two shapes.

FIGURE 5.66 Drawing a scene read in from an SDL file.

```
#include "SDL.h"
//################################ GLOBALS ###########################
Scene scn;  // construct the scene object
//<<<<<<<<<<<<<<<<<<<<<<<< displaySDL >>>>>>>>>>>>>>>>>>>>>>>>>
void displaySDL(void)
{
      glMatrixMode(GL_PROJECTION); //set the camera
      glLoadIdentity();
      double winHt = 1.0; // half-height of the window
      glOrtho(-winHt*64/48.0, winHt*64/48.0, -winHt, winHt, 0.1, 100.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      gluLookAt(2.3, 1.3, 2, 0, 0.25, 0, 0.0,1.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT); // clear screen
      scn.drawSceneOpenGL();
} // end of display
//<<<<<<<<<<<<<<<<<<<<<<< main >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
void main(int argc, char **argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_RGB |GLUT_DEPTH);
      glutInitWindowSize(640, 480);
      glutInitWindowPosition(100, 100);
      glutCreateWindow("read and draw an SDL scene");
      glutDisplayFunc(displaySDL);
      glShadeModel(GL_SMOOTH);
      glEnable(GL_DEPTH_TEST);
      glEnable(GL_NORMALIZE);
      glViewport(0, 0, 640, 480);
      scn.read("myScene1.dat"); //read the SDL file and build the objects
      glEnable(GL_LIGHTING);
      scn.makeLightsOpenGL(); // scan the light list and make OpenGL lights
      glutMainLoop();
}
```

FIGURE 5.67 The SDL file to create the scene of Figure 5.61.

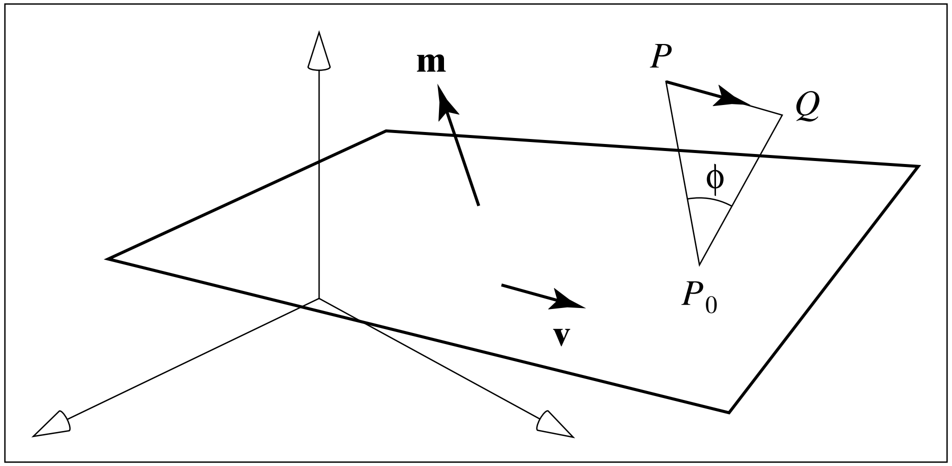
```
! - myScene1.dat
light 20 60 30 .7 .7 .7 !put a light at (20,60,30),color:(.7, .7, .7)
ambient .7 .7 .7 ! set material properties for all of the objects
diffuse .6 .6 .6
specular 1 1 1
specularExponent 50

def jackPart{ push scale .2 .2 1 sphere pop
push translate  0 0 1.2 scale .2 .2 .2 sphere pop
push translate 0 0 -1.2 scale .2 .2 .2 sphere pop
}

def jack{ push use jackPart
rotate 90 0 1 0 use jackPart
rotate 90 1 0 0 use jackPart pop
}

def wall{push translate 1 .01 1 scale 1 .02 1 cube pop}
def leg {push translate 0 .15 0 scale .01 .15 .01 cube pop}

def table{
push translate  0  .3  0 scale .3 .01 .3 cube pop !table top
push
translate .275  0 .275 use leg
translate  0 0 -.55 use leg
translate -.55 0 .55 use leg
translate 0  0 -.55 use leg pop
}
!now add the objects themselves
push translate .4 .4 .6 rotate 45 0 0 1 scale .08 .08 .08 use jack pop
push translate .25 .42 .35 scale .1 .1 .1 sphere pop
push translate .6 .38 .5 rotate 30 0 1 0 scale .08 .08 .08 teapot pop
push translate 0.4 0 0.4 use table pop

use wall
push rotate 90 0 0 1 use wall pop
push rotate -90 1 0 0 use wall pop
```

```
void drawArc2(RealPoint c, double R,
       double startangle, double sweep) // in degrees
{
    #define n 30
    #define RadPerDeg .01745329
    double delang =  RadPerDeg * sweep / n;
    double T =  tan(delang/2);         // tan. of half angle
    double S =  2 * T/(1 + T * T);      // sine of half angle
    double snR = R * sin(RadPerDeg * startangle);
    double csR = R * cos(RadPerDeg * startangle);
    moveTo(c.x + csR, c.y + snR);
    for(int i = 1; i < n; i++)
  {
       snR += T * csR;      // build next snR, csR pair
       csR -= S * snR;
       snR += T * csR;
       lineTo(c.x + csR, c.y + snR);
  }
}
```

FIGURE 5.68 A fast arc drawer.

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

FIGURE 5.69 Defining a shear in 3D.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill