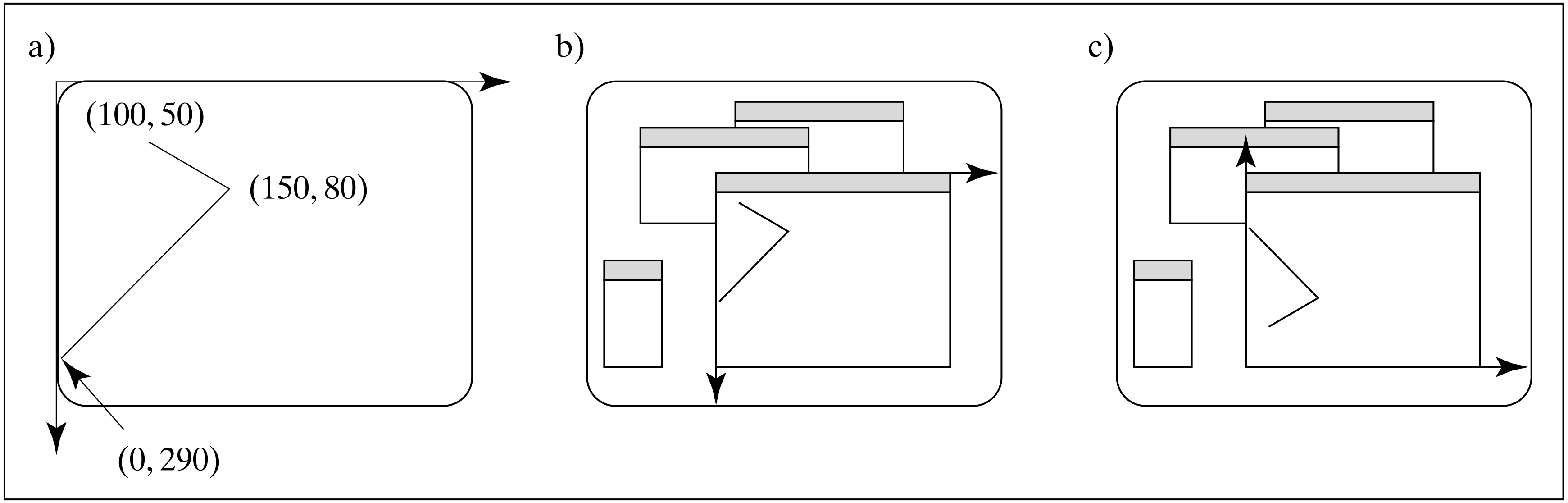


FIGURE 2.1 Some common varieties of display layouts.

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

FIGURE 2.2 A skeleton of an event-driven program using OpenGL.

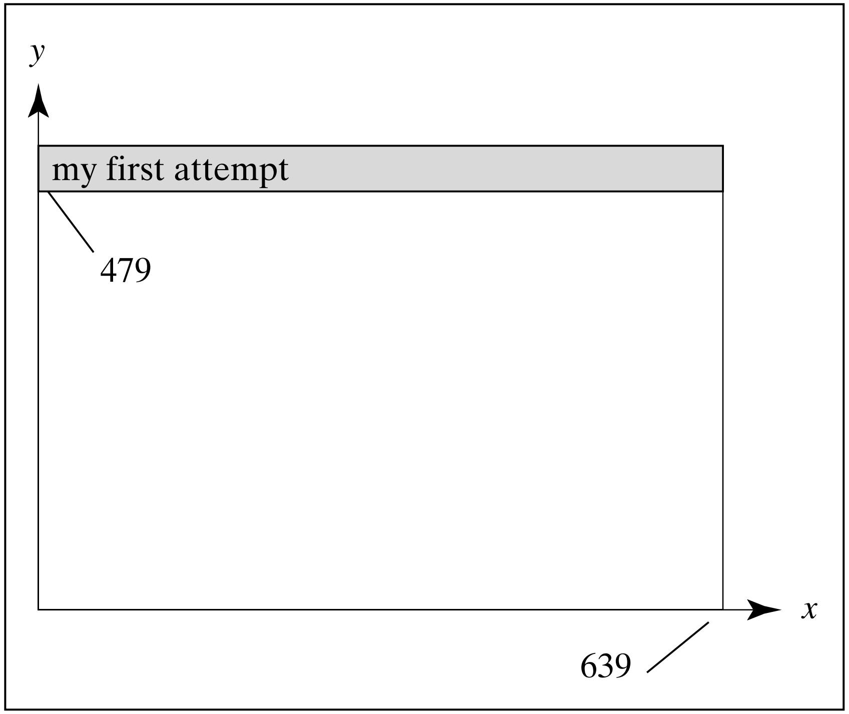
```
void main()
{
        initializ e things[5]
        create a screen windo w
        glutDisplayFunc(myDisplay);   // register the redraw function
        glutReshapeFunc(myReshape);   // register the reshape function
        glutMouseFunc(myMouse);       // register the mouse action function
        glutKeyboardFunc(myKeyboard); // register the keyboard action function
        perhaps initializ e other things
        glutMainLoop();               // enter the unending main loop
}
        all of the callbac k functions are defined here
```

FIGURE 2.3 Code using the OpenGL Utility Toolkit to open the initial window for drawing.

```
// appropriate #includes go here - see Appendix 1

void main(int argc, char** argv)
{
      glutInit(&argc, argv); // initialize the toolkit
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // set the display mode
      glutInitWindowSize(640,480); // set window size
      glutInitWindowPosition(100, 150); // set the window position on screen
      glutCreateWindow("my first attempt"); // open the screen window

      // register the callback functions
      glutDisplayFunc(myDisplay);
      glutReshapeFunc(myReshape);
      glutMouseFunc(myMouse);
      glutKeyboardFunc(myKeyboard);

      myInit();               // additional initializations as necessary
      glutMainLoop();         // go into a perpetual loop

}
```

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

FIGURE 2.4 The initial coordinate system for drawing.



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

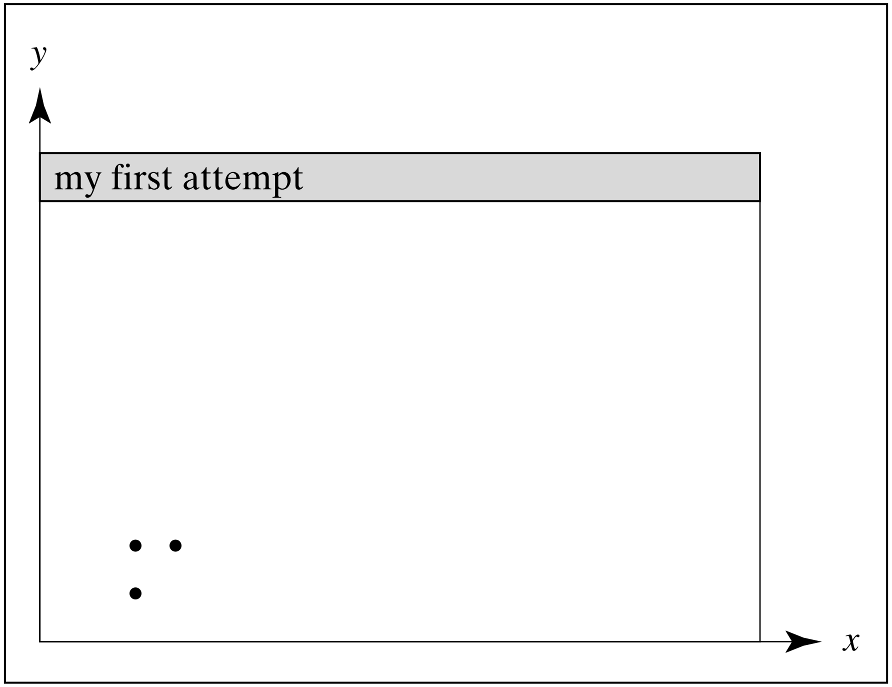


FIGURE 2.5 Drawing three dots.

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

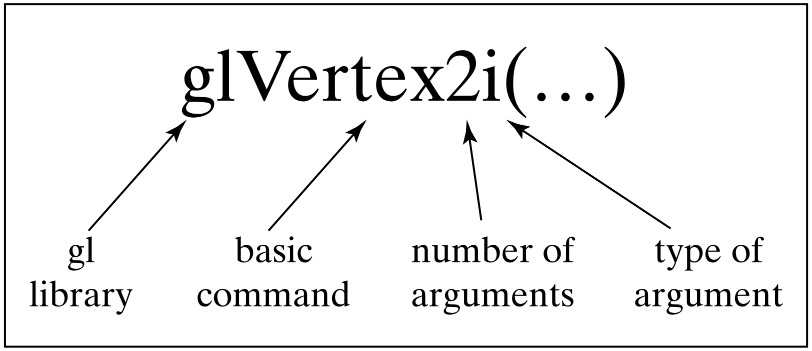


FIGURE 2.6 Format of OpenGL commands.

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

FIGURE 2.7 Command suffixes and argument data types.

Suffix	Data type	Typical C or C++ type	OpenGL type name
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating point	float	GLfloat, GLclampf
d	64-bit floating point	double	GLdouble, GLclampd
ub	8-bit unsigned number	unsigned char	GLubyte, GLboolean
us	16-bit unsigned number	unsigned short	GLushort
ui	32-bit unsigned number	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

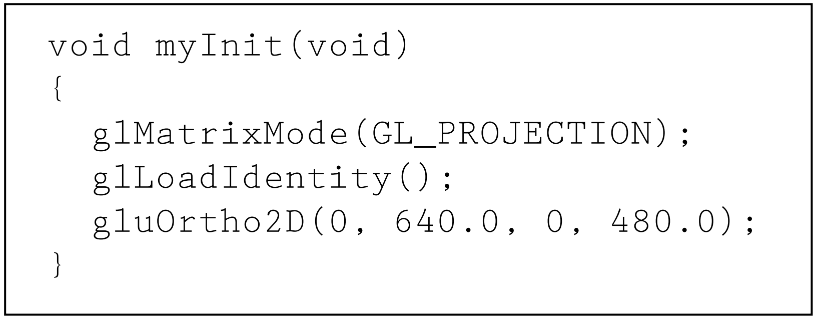
from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

```
void drawDot(GLint x, GLint y)
{     // draw dot at integer point (x, y)
  glBegin(GL_POINTS);
    glVertex2i(x, y);
  glEnd();
}
```

FIGURE 2.8 Encapsulating OpenGL details in the generic function `drawDot()`.[6]

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

FIGURE 2.9 Establishing a simple coordinate system.



```
void myInit(void)
{
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0, 640.0, 0, 480.0);
}
```

FIGURE 2.10 A complete OpenGL program to draw three dots.

```
#include<windows.h>   // use as needed for your system
#include<gl/Gl.h>
#include<gl/glut.h>
//<<<<<<<<<<<<<<<<<<<<<<< myInit >>>>>>>>>>>>>>>>>>>>
 void myInit(void)
 {
    glClearColor(1.0,1.0,1.0,0.0);       // set white background color
    glColor3f(0.0f, 0.0f, 0.0f);          // set the drawing color
    glPointSize(4.0);                 // a 'dot' is 4 by 4 pixels
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
//<<<<<<<<<<<<<<<<<<<<<<<< myDisplay >>>>>>>>>>>>>>>>>
void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
    glBegin(GL_POINTS);
        glVertex2i(100, 50);         // draw three points
        glVertex2i(100, 130);
        glVertex2i(150, 130);
    glEnd();
    glFlush();                   // send all output to display
}
//<<<<<<<<<<<<<<<<<<<<<<<< main >>>>>>>>>>>>>>>>>>>>>>
void main(int argc, char** argv)
{
    glutInit(&argc, argv);          // initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // set display mode
    glutInitWindowSize(640,480);     // set window size
    glutInitWindowPosition(100, 150); // set window position on screen
    glutCreateWindow("my first attempt"); // open the screen window
    glutDisplayFunc(myDisplay);     // register redraw function
    myInit();
    glutMainLoop();                 // go into a perpetual loop
}
```

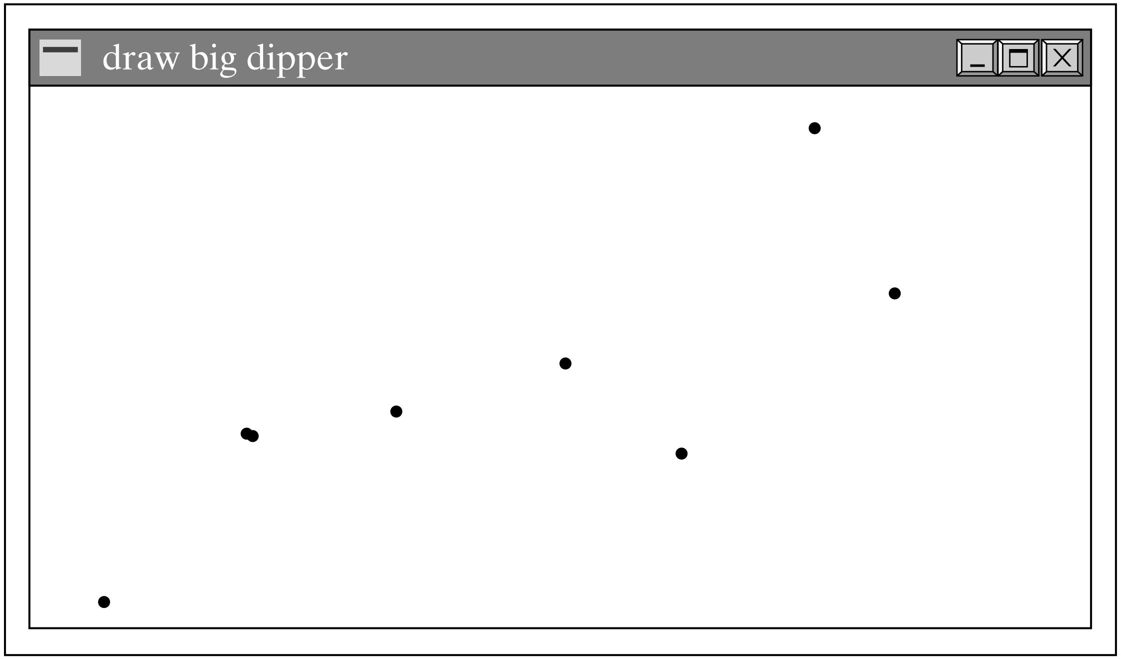
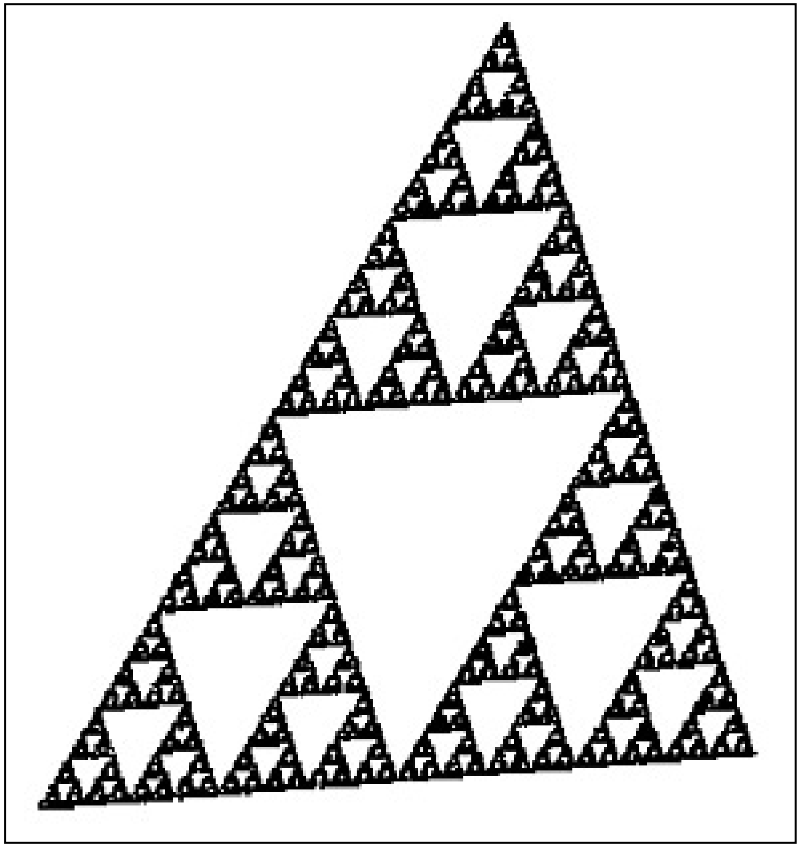


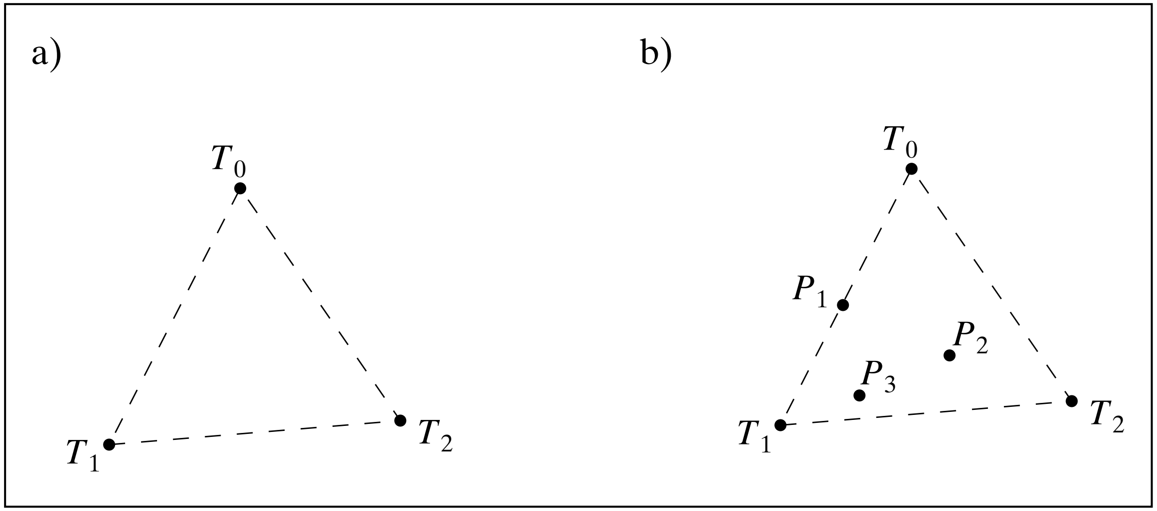
FIGURE 2.11 Two simple dot constellations.

FIGURE 2.12 The Sierpinski Gasket.



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

FIGURE 2.13 Building the Sierpinski gasket.

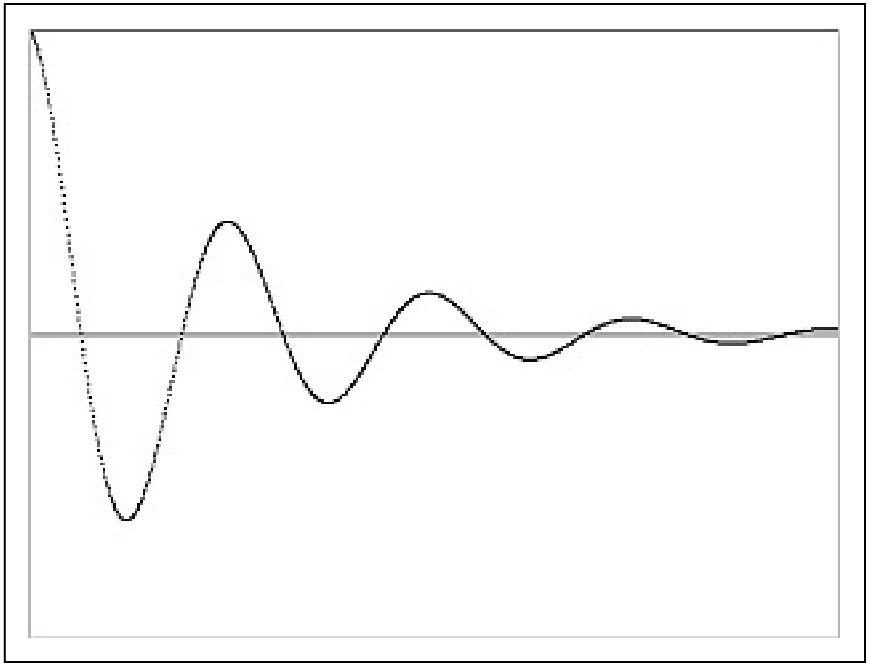


from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

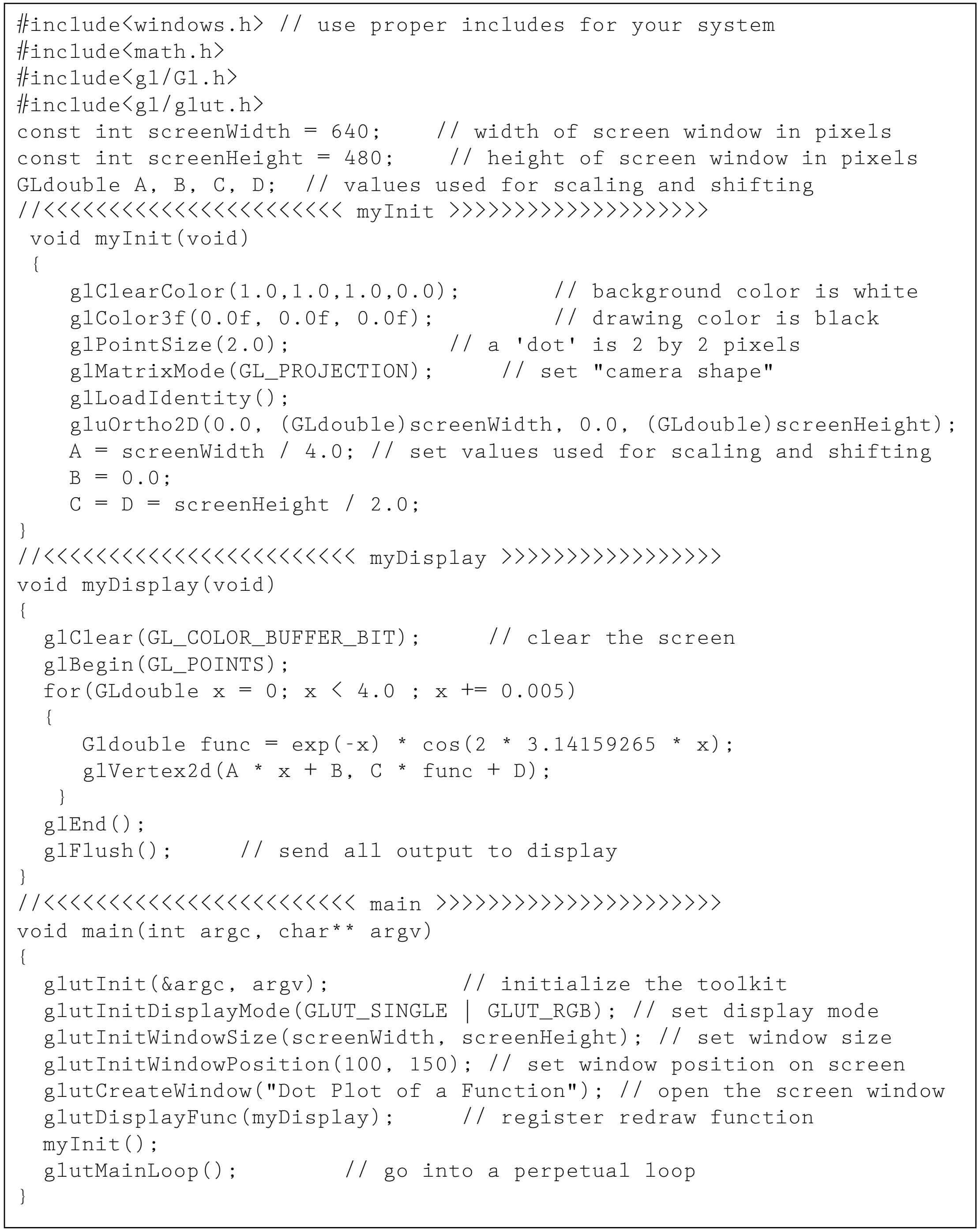
```
void Sierpinski(void)
{
  GLintPoint T[3]= {{10,10},{300,30},{200, 300}};

  int index = random(3);         // 0, 1, or 2 equally likely
  GLintPoint point = T[index];   // initial point
  drawDot(point.x, point.y);     // draw initial point
  for(int i = 0; i < 1000; i++)  // draw 1000 dots
  {
    index = random(3);
    point.x = (point.x + T[index].x) / 2;
    point.y = (point.y + T[index].y) / 2;
    drawDot(point.x,point.y);
  }
  glFlush();
}
```

FIGURE 2.14 Generating the Sierpinski gasket.

FIGURE 2.15 A "dot plot" of $e^{-x}\cos(2\pi x)$ versus x.



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



```
#include<windows.h> // use proper includes for your system
#include<math.h>
#include<gl/Gl.h>
#include<gl/glut.h>
const int screenWidth = 640;    // width of screen window in pixels
const int screenHeight = 480;    // height of screen window in pixels
GLdouble A, B, C, D;  // values used for scaling and shifting
//<<<<<<<<<<<<<<<<<<<<<<< myInit >>>>>>>>>>>>>>>>>>>>
 void myInit(void)
 {
    glClearColor(1.0,1.0,1.0,0.0);       // background color is white
    glColor3f(0.0f, 0.0f, 0.0f);         // drawing color is black
    glPointSize(2.0);            // a 'dot' is 2 by 2 pixels
    glMatrixMode(GL_PROJECTION);     // set "camera shape"
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)screenWidth, 0.0, (GLdouble)screenHeight);
    A = screenWidth / 4.0; // set values used for scaling and shifting
    B = 0.0;
    C = D = screenHeight / 2.0;
}
//<<<<<<<<<<<<<<<<<<<<<<<< myDisplay >>>>>>>>>>>>>>>>>
void myDisplay(void)
{
  glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
  glBegin(GL_POINTS);
  for(GLdouble x = 0; x < 4.0 ; x += 0.005)
  {
     Gldouble func = exp(-x) * cos(2 * 3.14159265 * x);
     glVertex2d(A * x + B, C * func + D);
   }
  glEnd();
  glFlush();     // send all output to display
}
//<<<<<<<<<<<<<<<<<<<<<<<< main >>>>>>>>>>>>>>>>>>>>>>
void main(int argc, char** argv)
{
  glutInit(&argc, argv);          // initialize the toolkit
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // set display mode
  glutInitWindowSize(screenWidth, screenHeight); // set window size
  glutInitWindowPosition(100, 150); // set window position on screen
  glutCreateWindow("Dot Plot of a Function"); // open the screen window
  glutDisplayFunc(myDisplay);     // register redraw function
  myInit();
  glutMainLoop();        // go into a perpetual loop
}
```

FIGURE 2.16 A complete program for drawing the "dot plot" of a function.

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

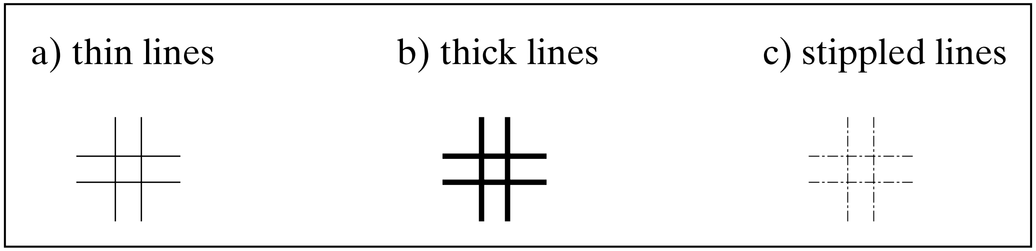


FIGURE 2.17 Simple picture built from four lines.

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

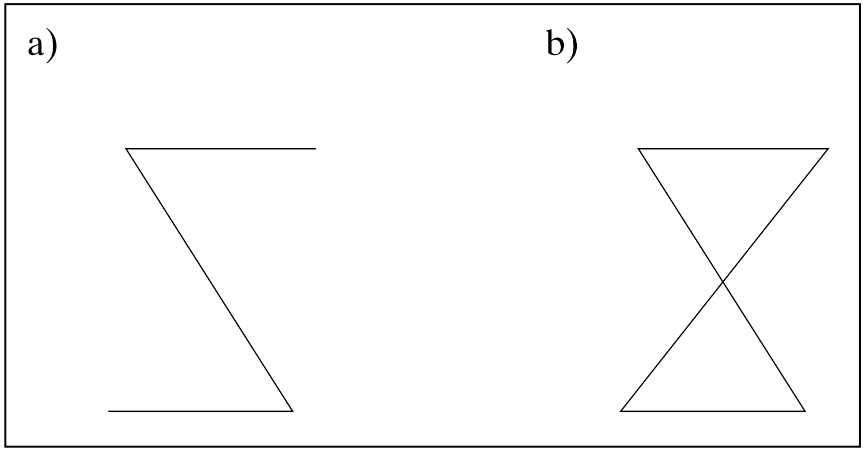
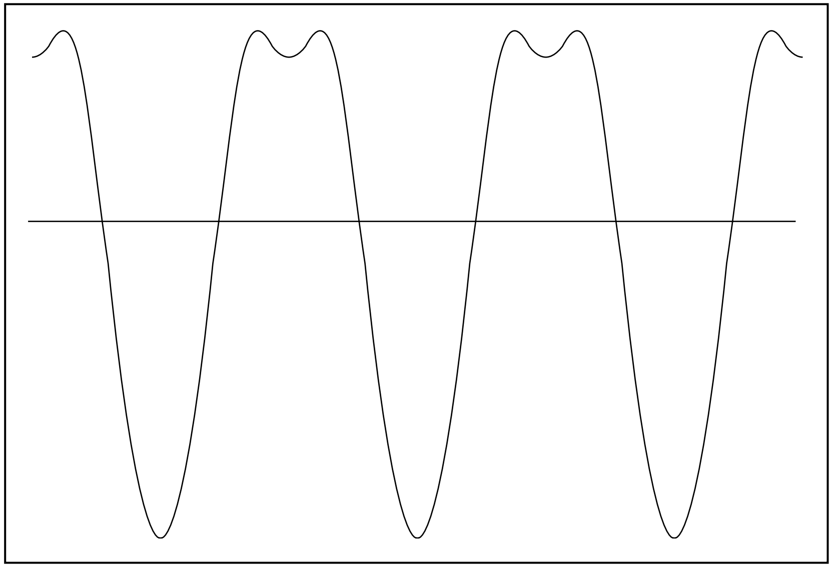


FIGURE 2.18 A polyline and a polygon.

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

FIGURE 2.19 A plot of a mathematical formula.



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

FIGURE 2.20 Plotting a function using a line graph.

```
glBegin(GL_LINE_STRIP);
for(Gldouble x = 0; x < 4.0; x += 0.005)
{
        define func
        glVertex2d(A * x + B, C * func + D);
}
glEnd();
glFlush;
```

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

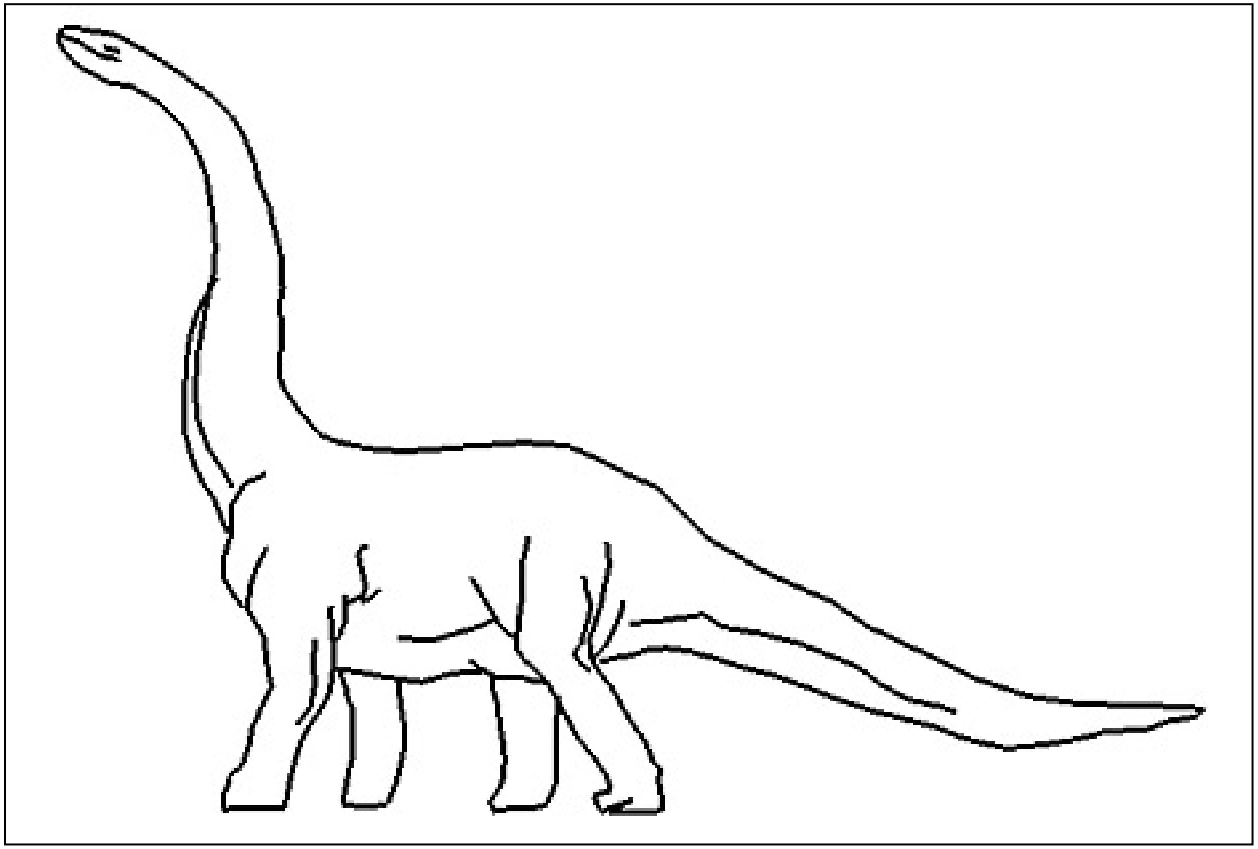


FIGURE 2.21 Drawing polylines stored in a file.

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

```
#include <fstream.h>
void drawPolyLineFile(char * fileName)
{
     fstream inStream;
     inStream.open(fileName, ios ::in); // open the file
     if(inStream.fail())
        return;
     glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
     GLint numpolys, numLines, x ,y;
     inStream >> numpolys;             // read the number of polylines
     for(int j = 0; j < numpolys; j++) // read each polyline
     {
        inStream >> numLines;
        glBegin(GL_LINE_STRIP);        // draw the next polyline
        for (int i = 0; i < numLines; i++)
        {
            inStream >> x >> y;        // read the next x, y pair
            glVertex2i(x, y);
        }
        glEnd();
     }
     glFlush();
     inStream.close();
}
```

FIGURE 2.22 Drawing polylines stored in a file.

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill
© 2001 by Prentice Hall / Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458

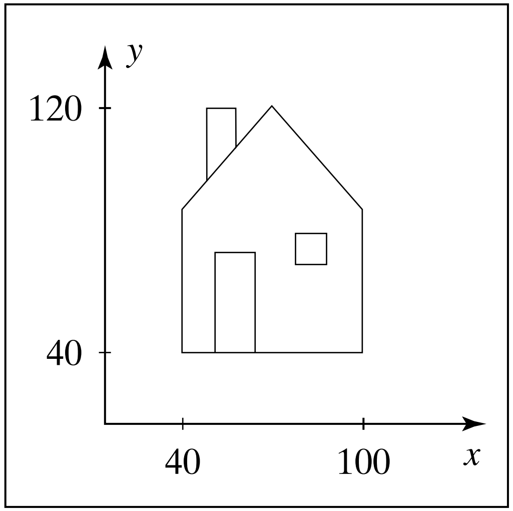


FIGURE 2.23 A House.

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

```
void hardwiredHouse(void)
{
   glBegin(GL_LINE_LOOP);
   glVertex2i(40, 40); // draw the shell of house
   glVertex2i(40, 90);
   glVertex2i(70, 120);
   glVertex2i(100, 90);
   glVertex2i(100, 40);
 glEnd();
 glBegin(GL_LINE_STRIP);
   glVertex2i(50, 100); // draw the chimney
   glVertex2i(50, 120);
   glVertex2i(60, 120);
   glVertex2i(60, 110);
 glEnd();
    . . . // draw the door
    . . . // draw the window
}
```

FIGURE 2.24 Drawing a house with “hardwired” dimensions.

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

```
void parameterizedHouse(GLintPoint peak, GLint width, GLint height)
 // the top of house is at the peak; the size of house is given
 //  by the height and width
{
 glBegin(GL_LINE_LOOP);
   glVertex2i(peak.x,             peak.y);  // draw shell of house
   glVertex2i(peak.x + width / 2, peak.y - 3 * height /8);
   glVertex2i(peak.x + width / 2  peak.y -     height);
   glVertex2i(peak.x - width / 2, peak.y -     height);
   glVertex2i(peak.x - width / 2, peak.y - 3 * height /8);
 glEnd();
 draw the c himney in the same fashion
 draw the door
 draw the windo w
}
```

FIGURE 2.25 Drawing a parameterized house.

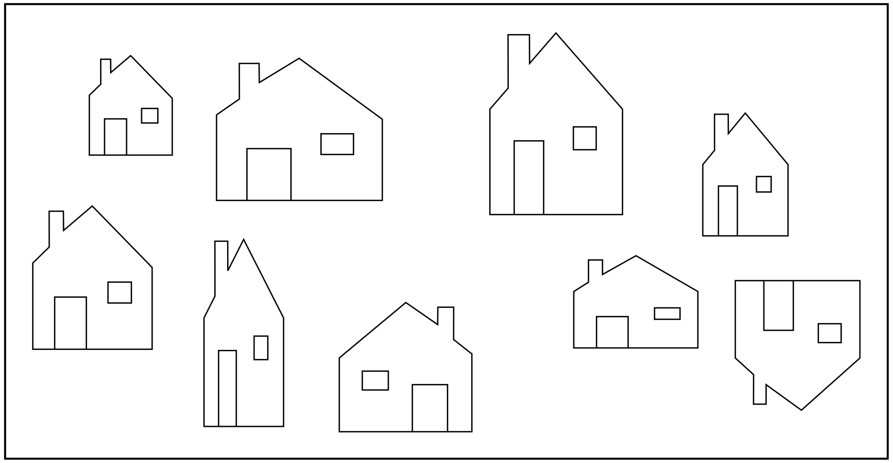


FIGURE 2.26 A “village” of houses drawn using `parameterizedHouse()`.

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

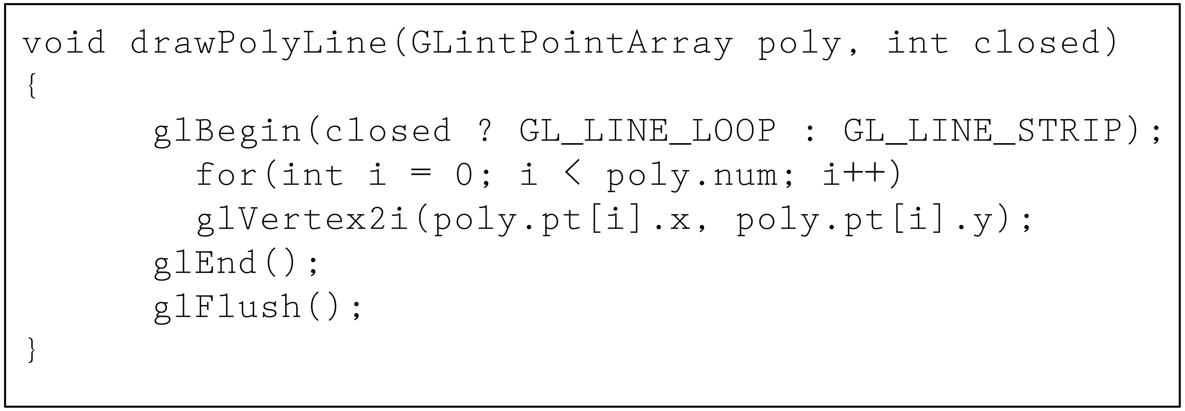
```
class GLintPointArray{
   const int MAX_NUM = 100;
   public:
      int num;
      GLintPoint pt[MAX_NUM];
};
```

FIGURE 2.27 Data type for a linked list of vertices.

Prentice Hall

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

FIGURE 2.28 A linked list data type, and drawing a polyline or polygon.



```
void drawPolyLine(GLintPointArray poly, int closed)
{
      glBegin(closed ? GL_LINE_LOOP : GL_LINE_STRIP);
        for(int i = 0; i < poly.num; i++)
        glVertex2i(poly.pt[i].x, poly.pt[i].y);
      glEnd();
      glFlush();
}
```

FIGURE 2.29 Defining `moveto()` and `lineto()` in OpenGL.

```
GLintPoint CP;         // global current position

//<<<<<<<<<<<<< moveto >>>>>>>>>>>>>>
void moveto(GLint x, GLint y)
{
   CP.x = x; CP.y = y; // update the CP
}
//<<<<<<<<<<<< lineTo >>>>>>>>>>>>>>>>>
void lineto(GLint x, GLint y)
{
   glBegin(GL_LINES);  // draw the line
      glVertex2i(CP.x, CP.y);
      glVertex2i(x, y);
   glEnd();
   glFlush();
   CP.x = x; CP.y = y; // update the CP
}
```

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

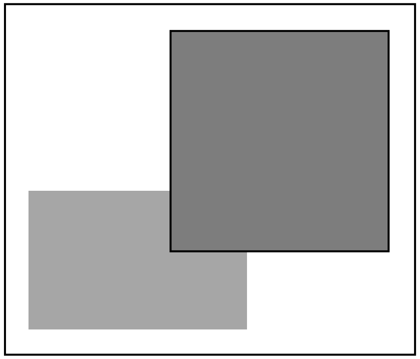


FIGURE 2.30 Two aligned rectangles filled with colors.

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

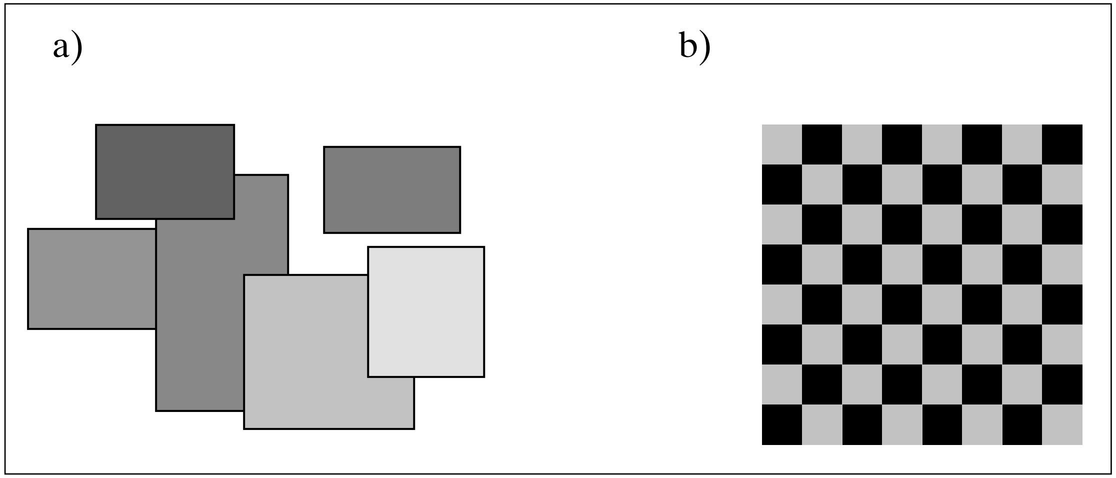
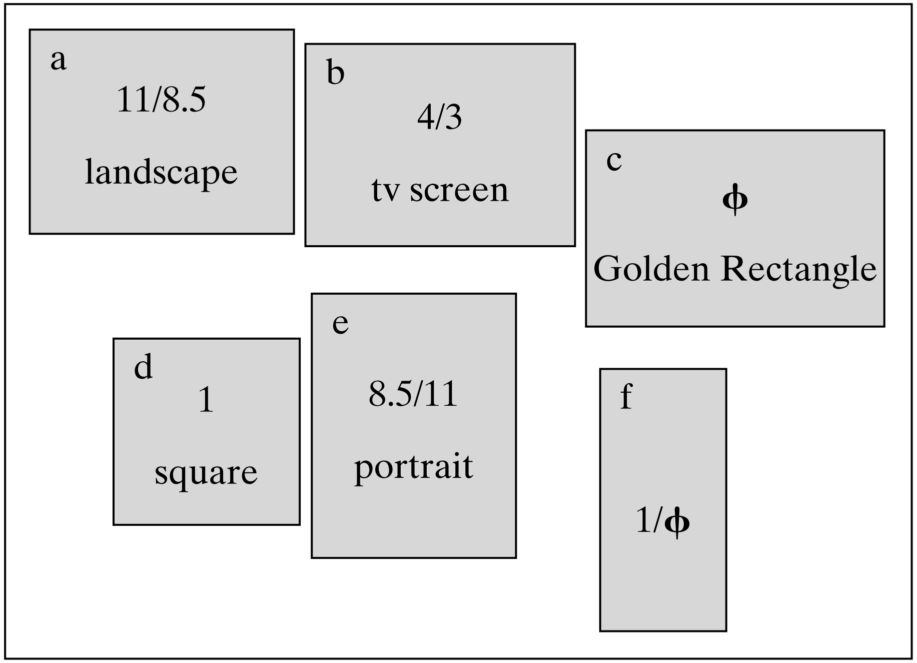


FIGURE 2.31 (a) Random flurry of rectangles. (b) A checkerboard.

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

FIGURE 2.32 Examples of aspect ratios of aligned rectangles.



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

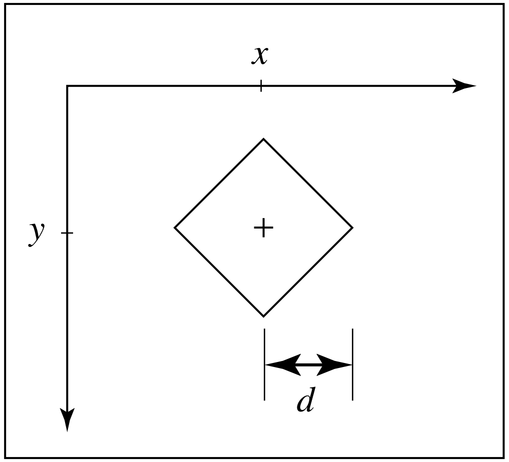


FIGURE 2.33 A simple diamond.

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

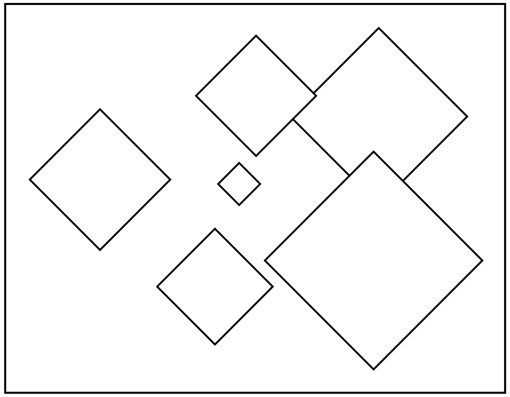


FIGURE 2.34 A “flurry” of diamonds.

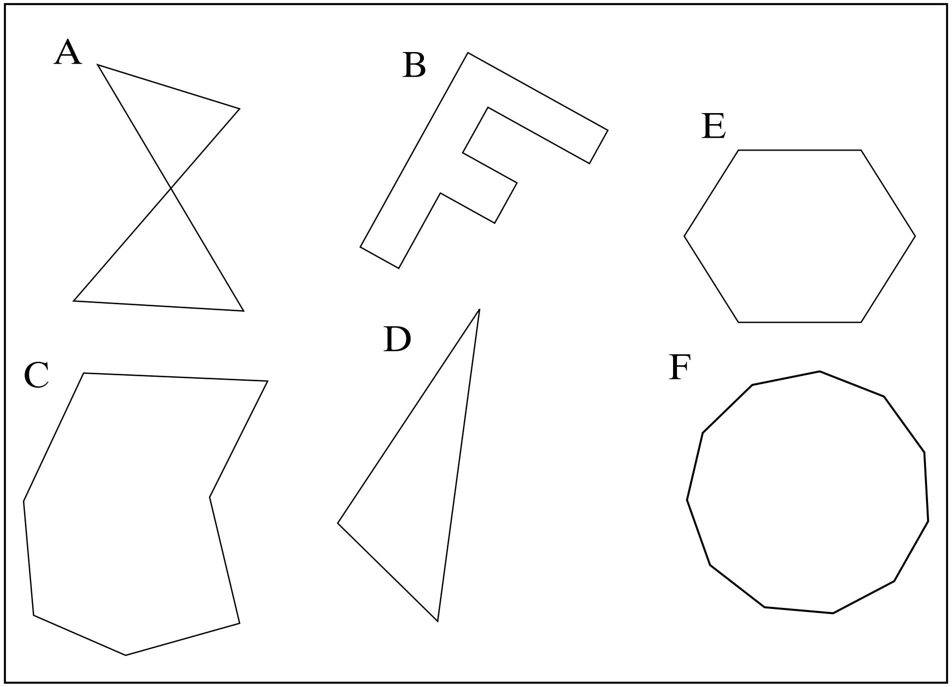
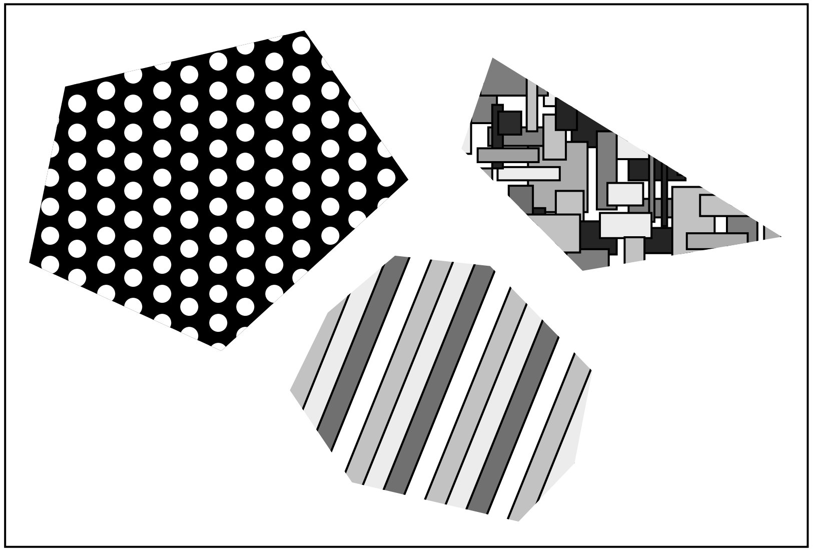


FIGURE 2.35 Convex and nonconvex polygons.

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

FIGURE 2.36 Several filled convex polygons.



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

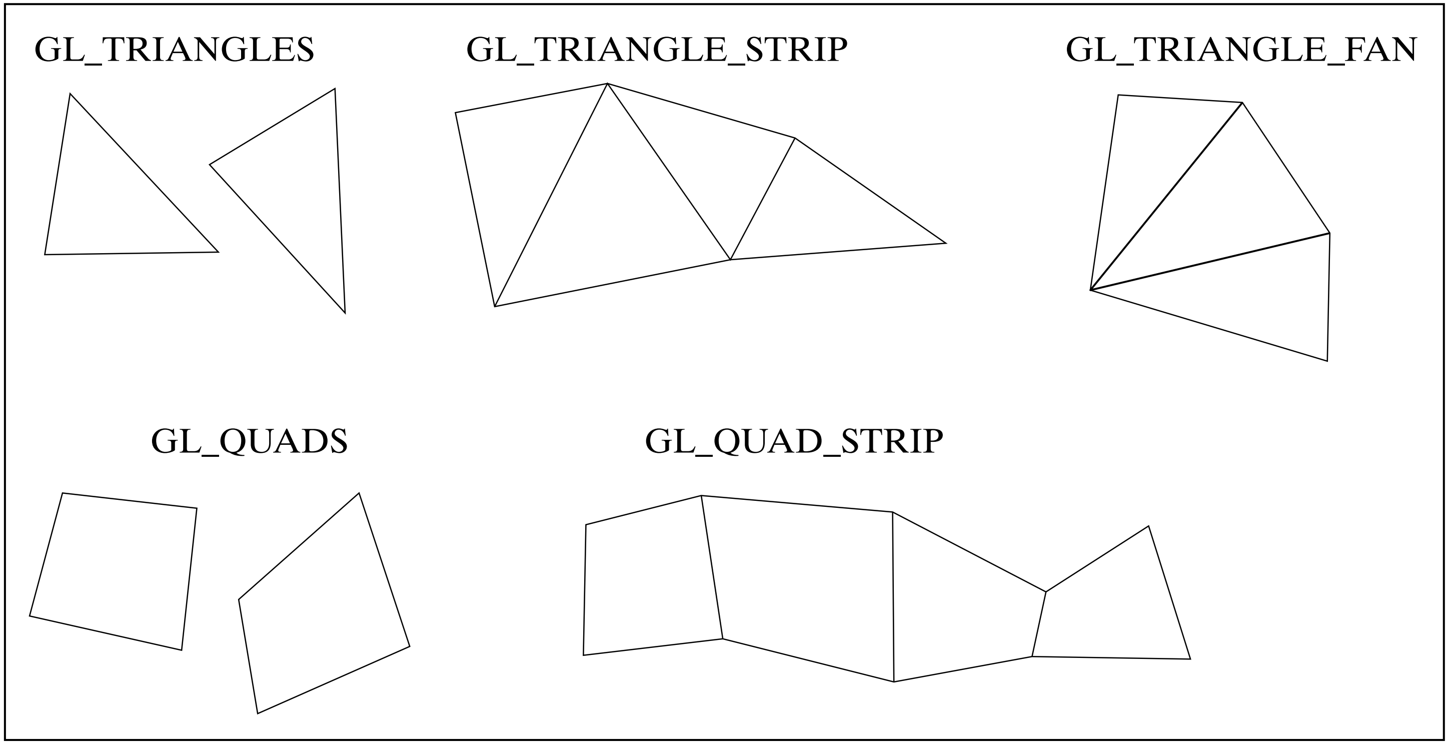


FIGURE 2.37 Other geometric primitive types.

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

FIGURE 2.38 A callback routine to draw rectangles entered with the mouse.

```
void myMouse(int button, int state, int x, int y)
{
    static GLintPoint corner[2];
    static int numCorners = 0;           // initial value is 0
    if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
       corner[numCorners].x = x;
       corner[numCorners].y = screenHeight - y;    // flip y coordinate
       numCorners++;                       // have another point
       if(numCorners == 2)
       {
          glRecti(corner[0].x, corner[0].y, corner[1].x, corner[1].y);
          numCorners = 0;       // back to 0 corners
       }
    }
    else if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
             glClear(GL_COLOR_BUFFER_BIT);          // clear the window
    glFlush();
}
```

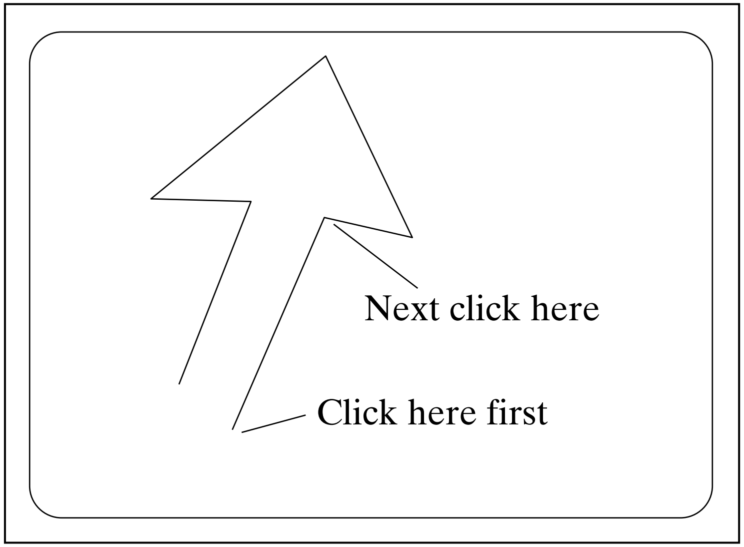


FIGURE 2.39 Interactive creation of a polyline.

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

```
void myMouse(int button, int state, int x, int y)
{
    #define NUM 20
    static GLintPoint List[NUM];
    static int last = -1;             // last index used so far

   // test for mouse button as well as for a full array
     if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && last < (NUM -1))
     {
        List[++last].x = x;       // add new point to list
        List[  last].y = screenHeight - y;
        glClear(GL_COLOR_BUFFER_BIT);     // clear the screen
        glBegin(GL_LINE_STRIP);       // redraw the polyline
          for(int i = 0; i <= last; i++)
            glVertex2i(List[i].x, List[i].y);
        glEnd();
        glFlush();
   }
   else if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
     last = -1;          // reset the list to empty
}
```

FIGURE 2.40 A polyline drawer based on mouse clicks.

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill
© 2001 by Prentice Hall / Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458

```
void myKeyboard(unsigned char theKey, int mouseX, int mouseY)
{
  GLint x = mouseX;
  GLint y = screenHeight - mouseY; // flip the y value as always
  switch(theKey)
  {
    case 'p':
       drawDot(x, y);   // draw a dot at the mouse position
       break;
    case GLUT_KEY_LEFT: List[++last].x = x; // add a point
                        List[  last].y = y;
       break;
    case 'E':
       exit(-1);        //terminate the program
    default:
     break;             // do nothing
 }
}
```

FIGURE 2.41 An example of the keyboard callback function.

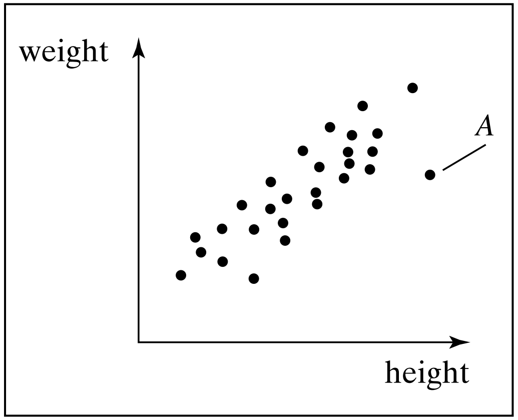


FIGURE 2.42 A scatter plot of people's height versus weight.

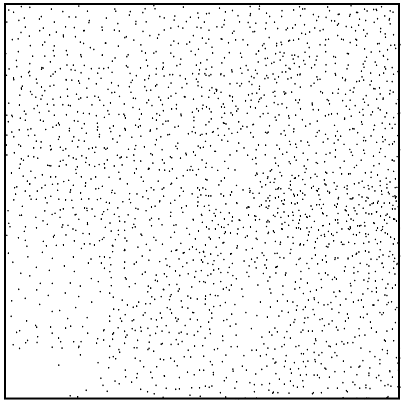


FIGURE 2.43 A constellation of 500 random dots.

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

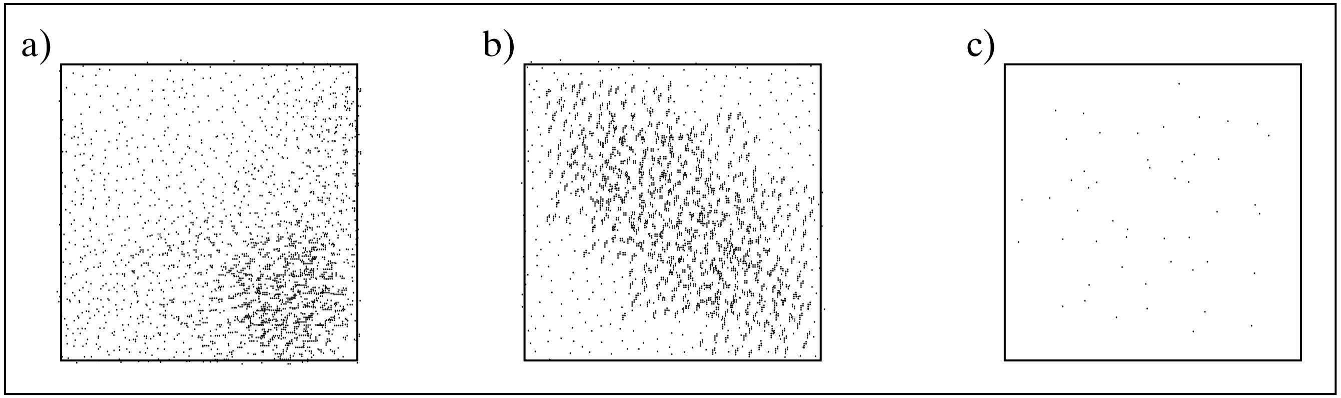
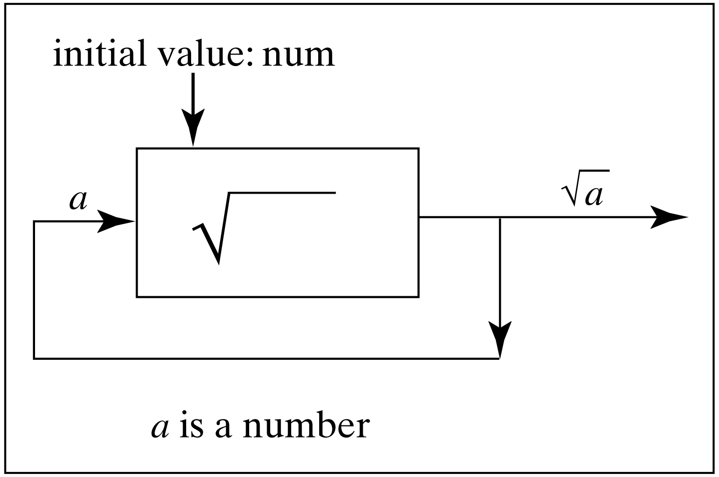


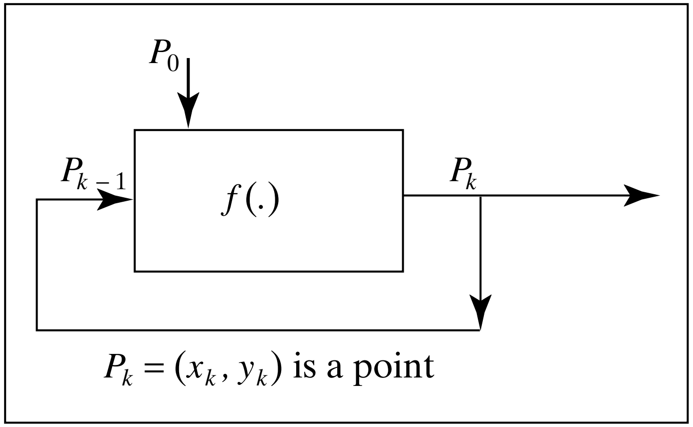
FIGURE 2.44 Scatter plots produced by inferior random-number generators.

FIGURE 2.45 Taking the square root repetitively.



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

FIGURE 2.46 Iterated function sequence generator for points.



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

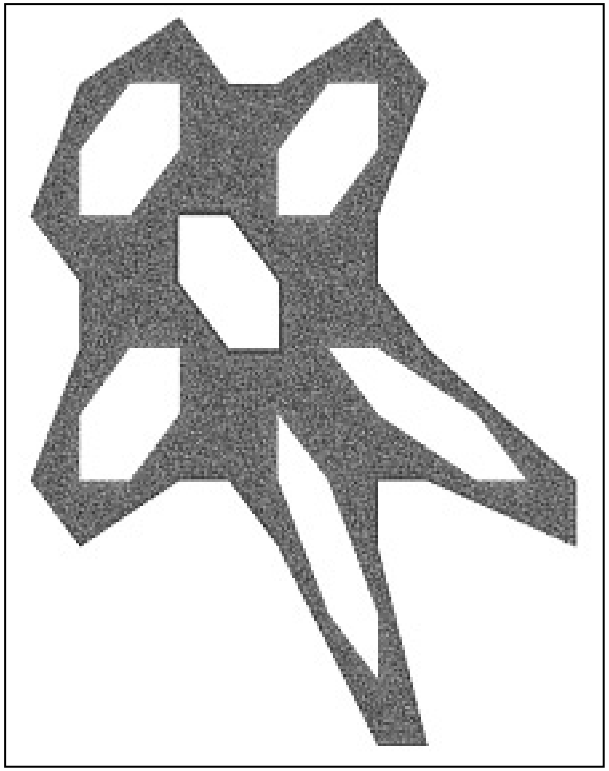


FIGURE 2.47 A typical gingerbread man.

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

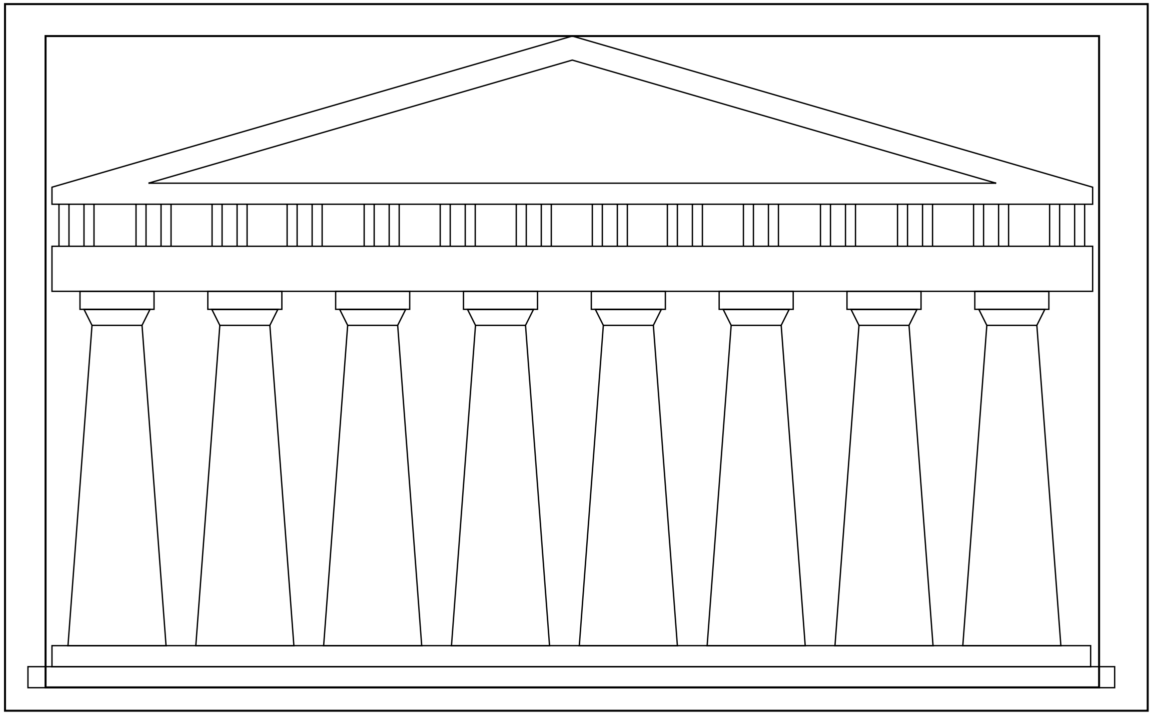
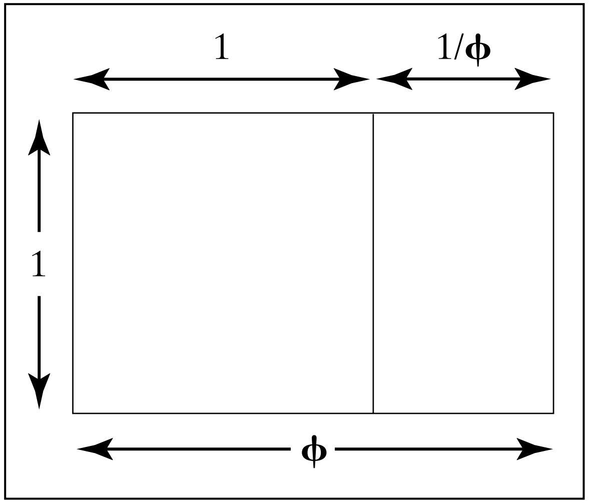


FIGURE 2.48 The Greek Parthenon fitting within a golden rectangle.

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

FIGURE 2.49 The golden rectangle.



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

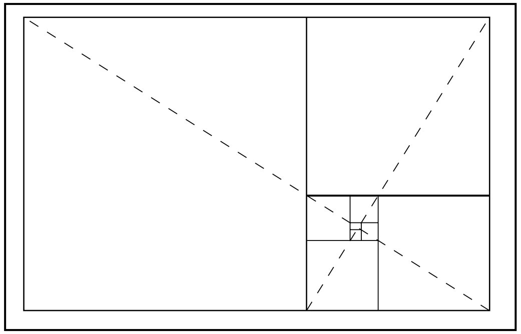


FIGURE 2.50 Infinite regressions of the golden rectangle.

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

pattern	factor	resulting stipple
0xFF00	1	....
0xFF00	2	....
0x5555	1	. . . .
0x3333	2	.. ..
0x7733	1	... ...

FIGURE 2.51 Sample stipple patterns.

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

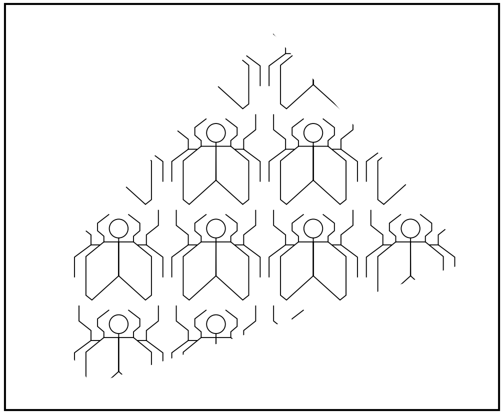


FIGURE 2.52 A sample stippled polygon.

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

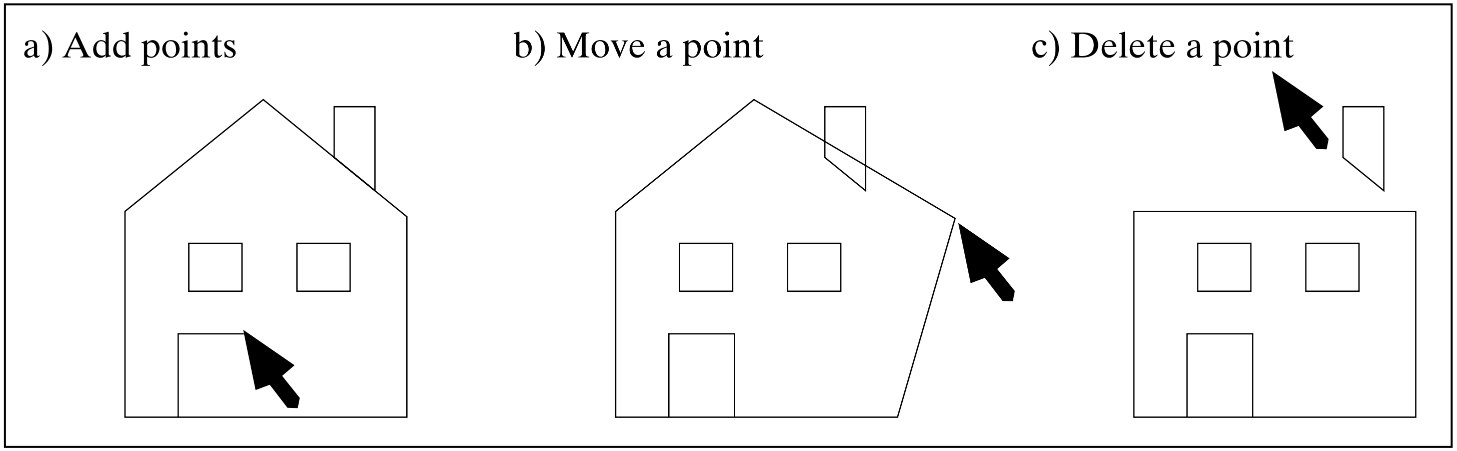


FIGURE 2.53 Creating and editing polylines.

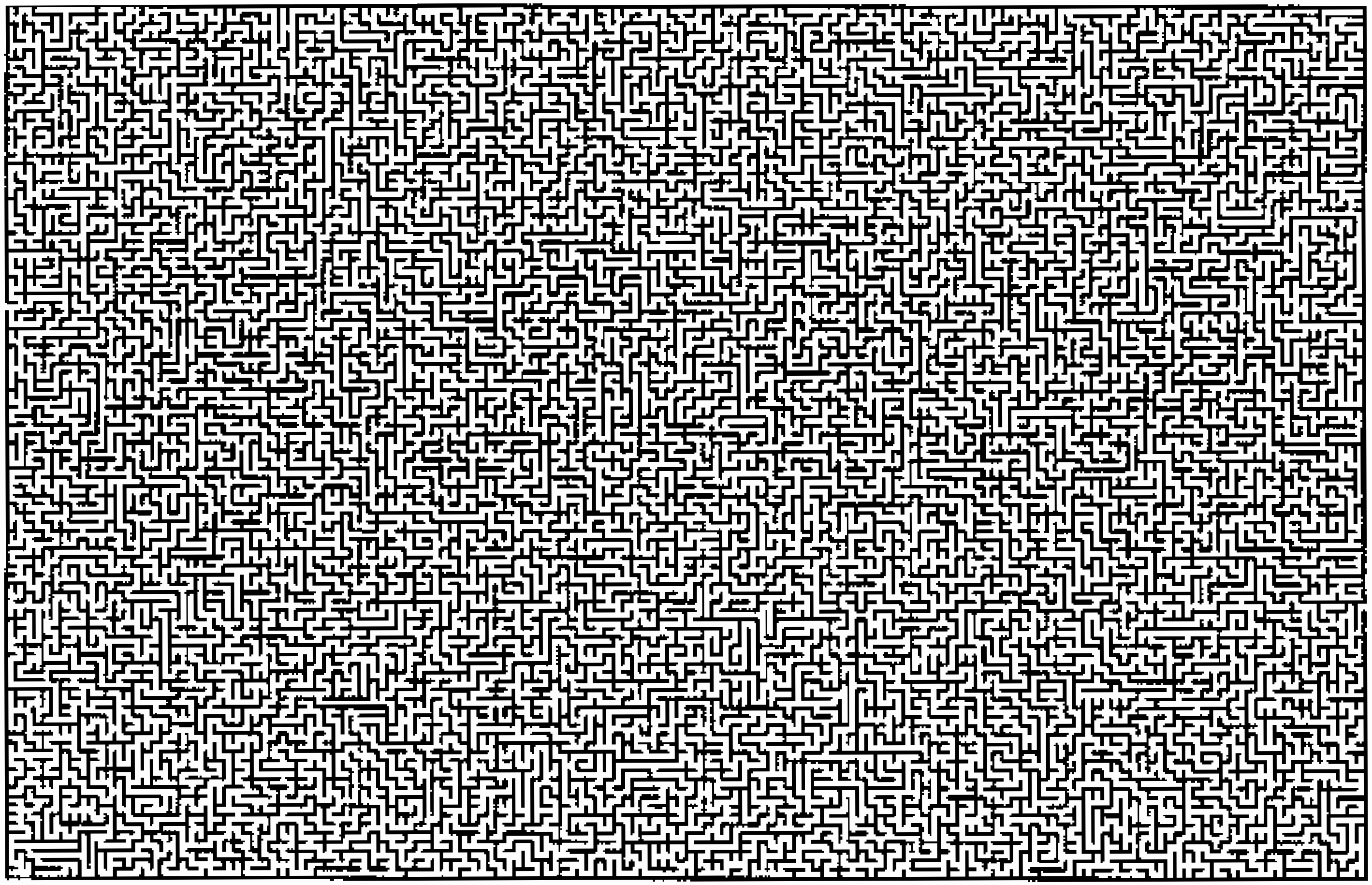


FIGURE 2.54 A maze.

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

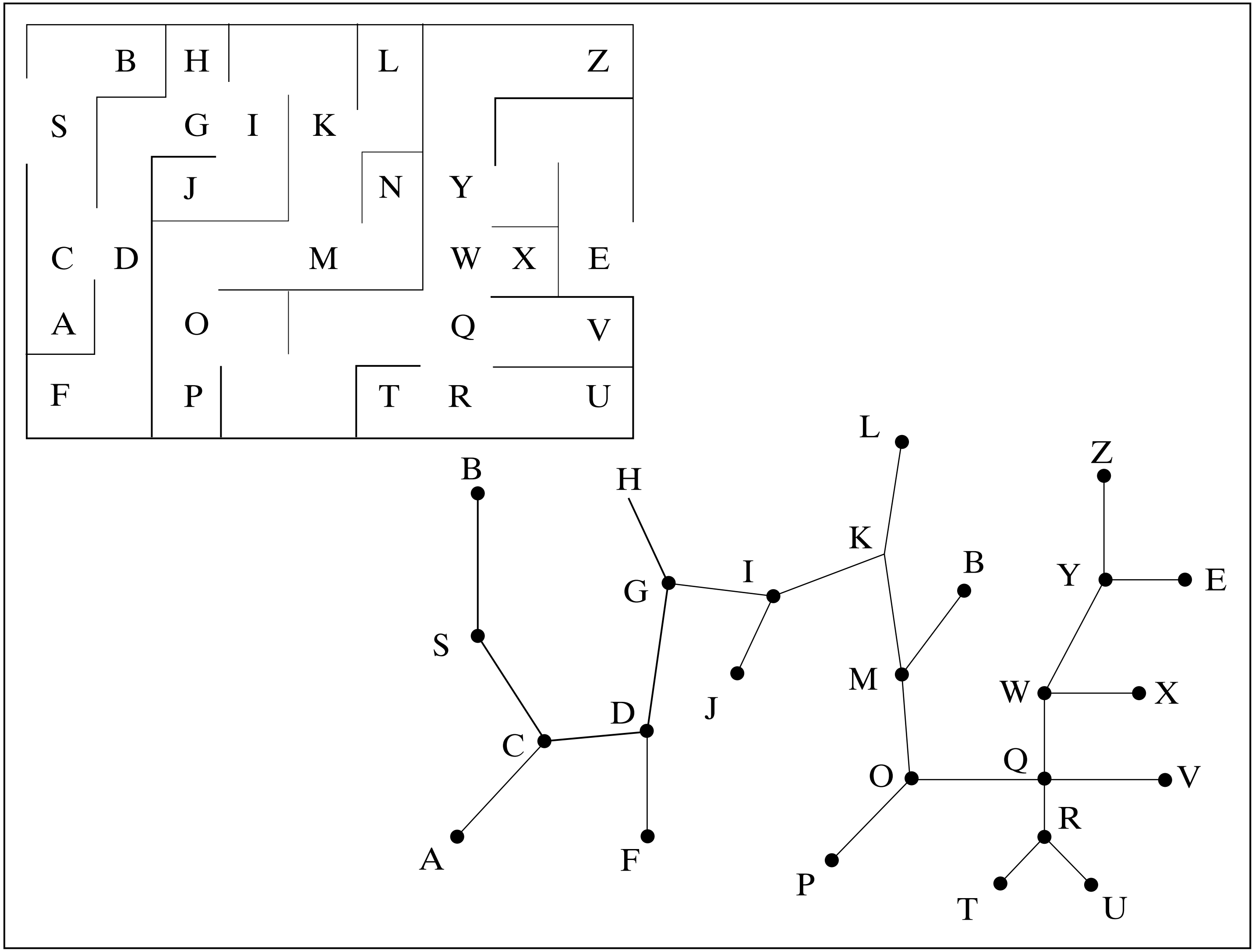


FIGURE 2.55 A simple maze and its graph.

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