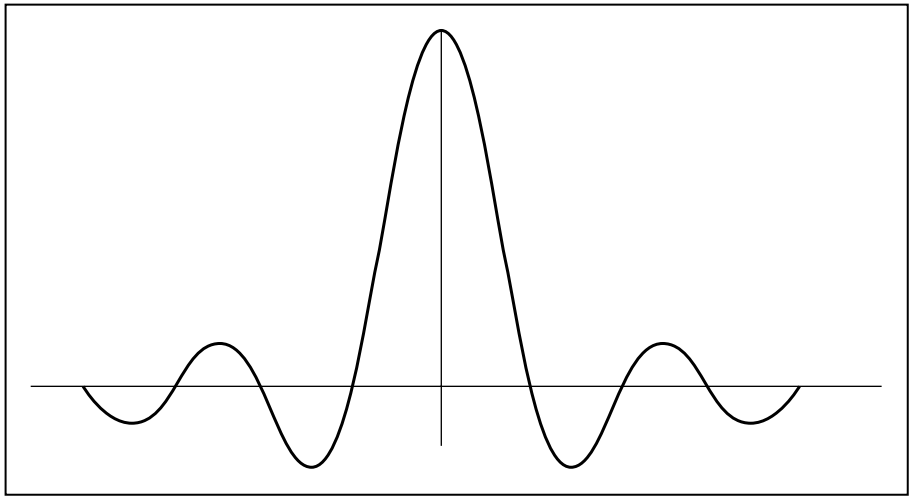
FIGURE 3.1 A plot of the "sinc" function.



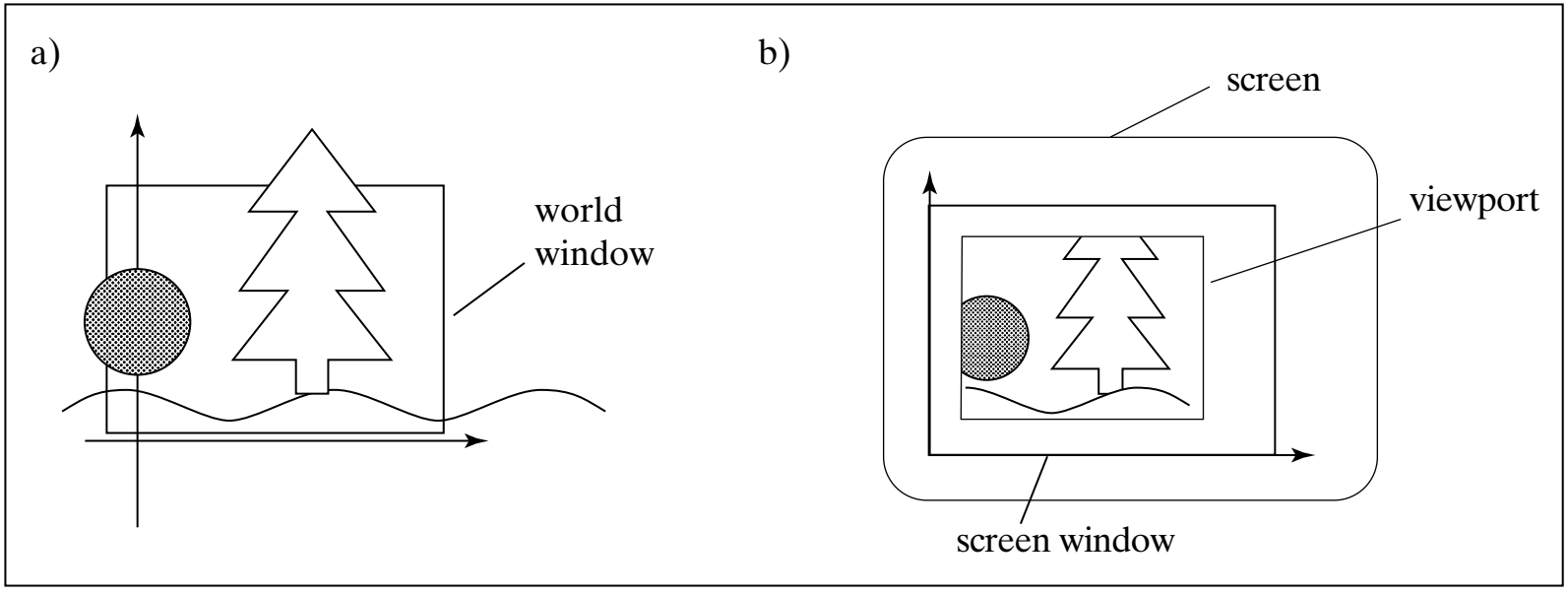


FIGURE 3.2 A world window and a viewport.

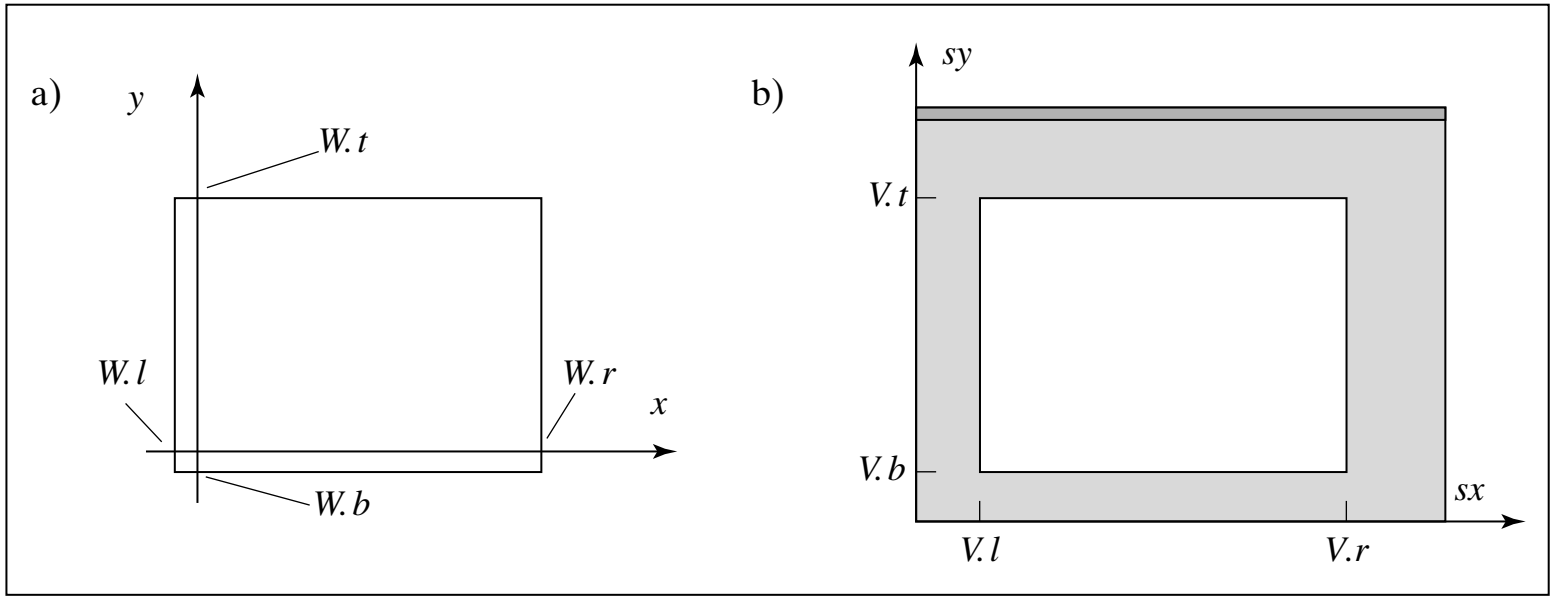


FIGURE 3.3 Specifying the window and viewport.

FIGURE 3.4 A picture mapped from a window to a viewport. Here some distortion is produced.

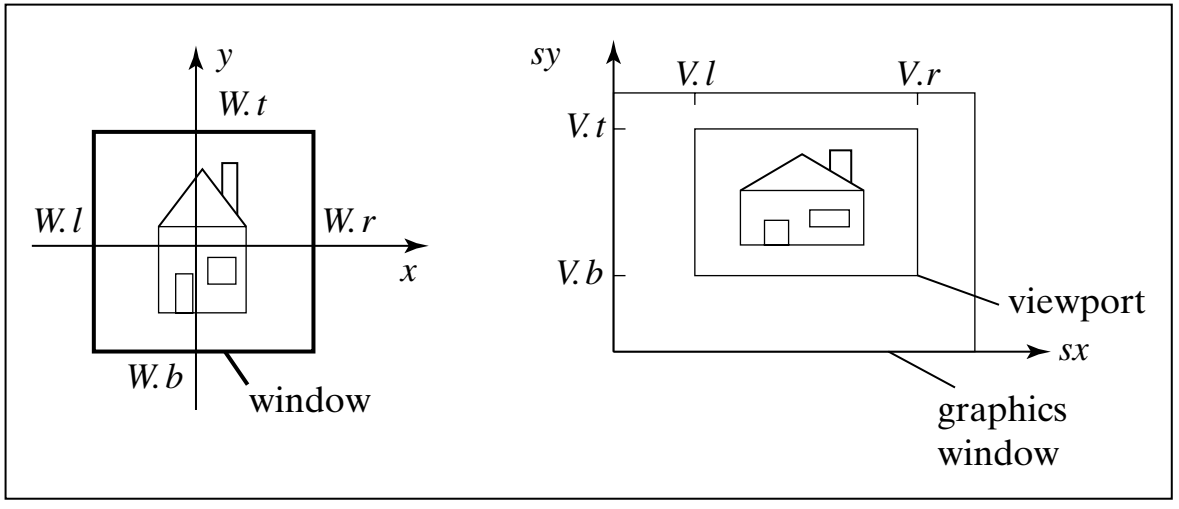
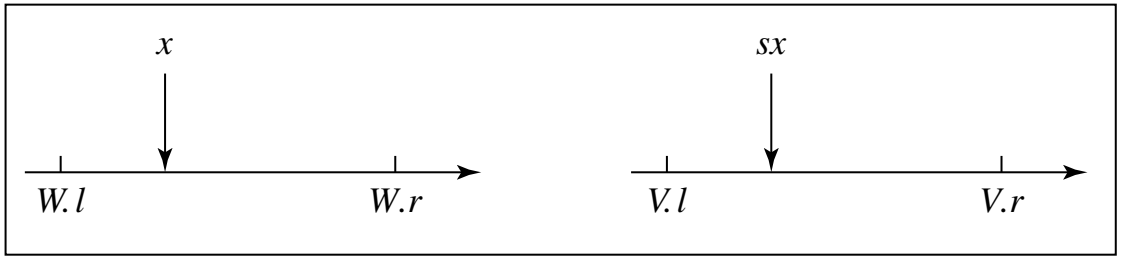


FIGURE 3.5 Proportionality in mapping x to sx.



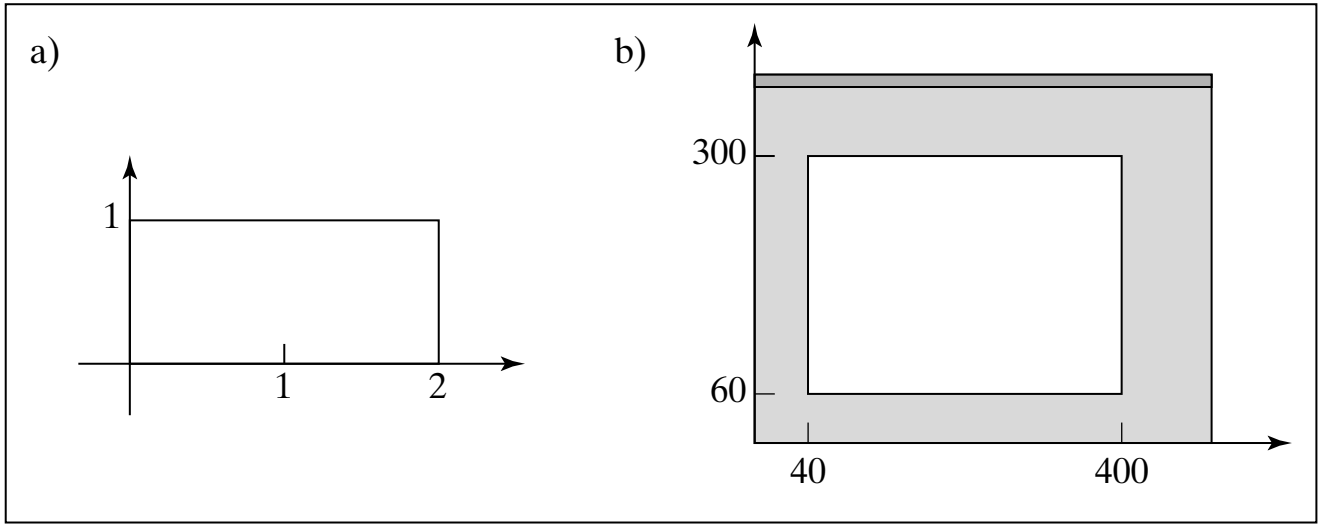


FIGURE 3.6 An example of a window and a viewport.

```
//---------------- setWindow ----------------------
void setWindow(float left, float right, float bottom, float top)
{
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(left, right, bottom, top);
}
//----------------- setViewport -------------------
void setViewport(int left, int right, int bottom, int top)
{
      glViewport(left, bottom, right - left, top - bottom);
}
```

FIGURE 3.7 Handy functions to set the window and viewport.

```
void myDisplay(void) // plot the sinc function, using world coordinates
{
   setWindow(-5.0, 5.0, -0.3, 1.0);   // set the window
   setViewport(0, 640, 0, 480);   // set the viewport
   glBegin(GL_LINE_STRIP);
   for(GLfloat x = -4.0; x < 4.0; x += 0.1)     // draw the plot
      glVertex2f(x, sin(3.14159 * x) / (3.14159 * x));
   glEnd();
   glFlush();
}
```

FIGURE 3.8 Plotting the sinc function.

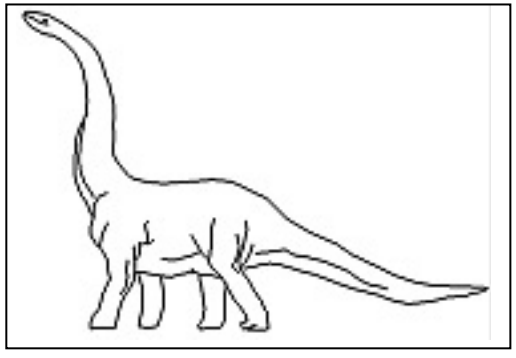


FIGURE 3.9 The dinosaur inside its world window.

FIGURE 3.10 Tiling the display with copies of the dinosaur.

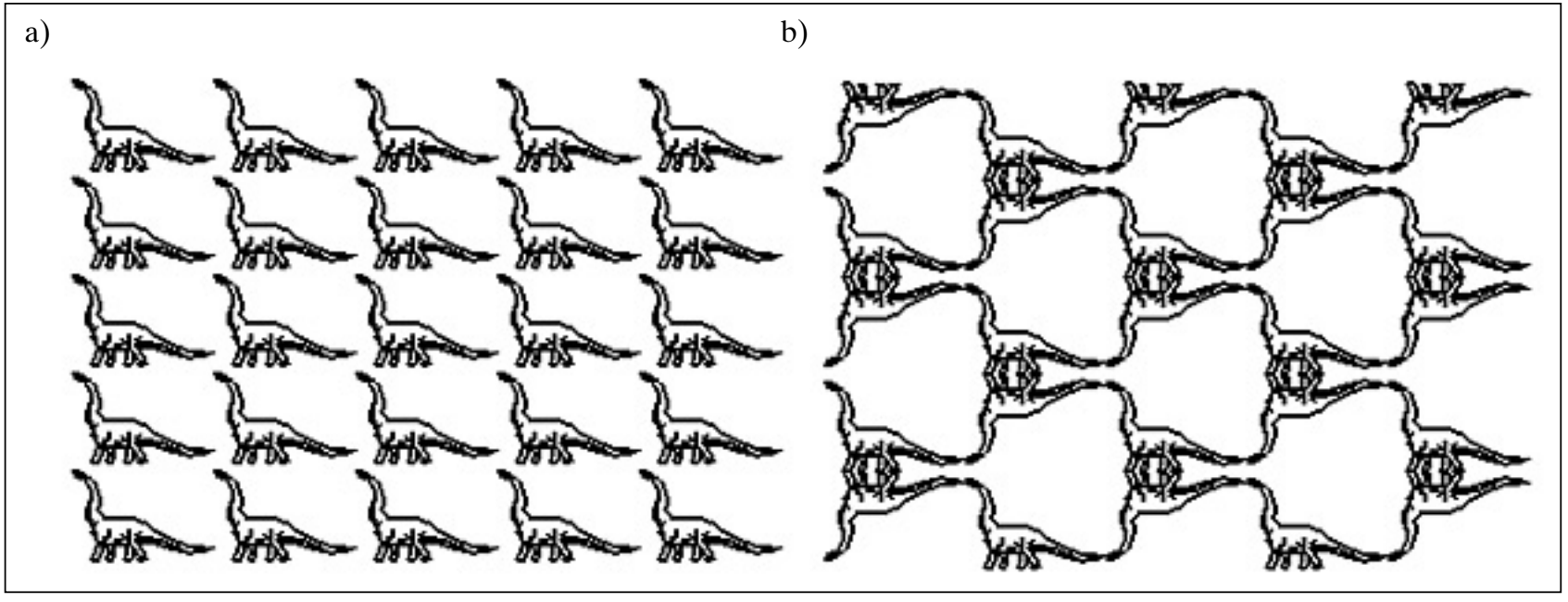


FIGURE 3.11 Using the window to clip parts of a figure.

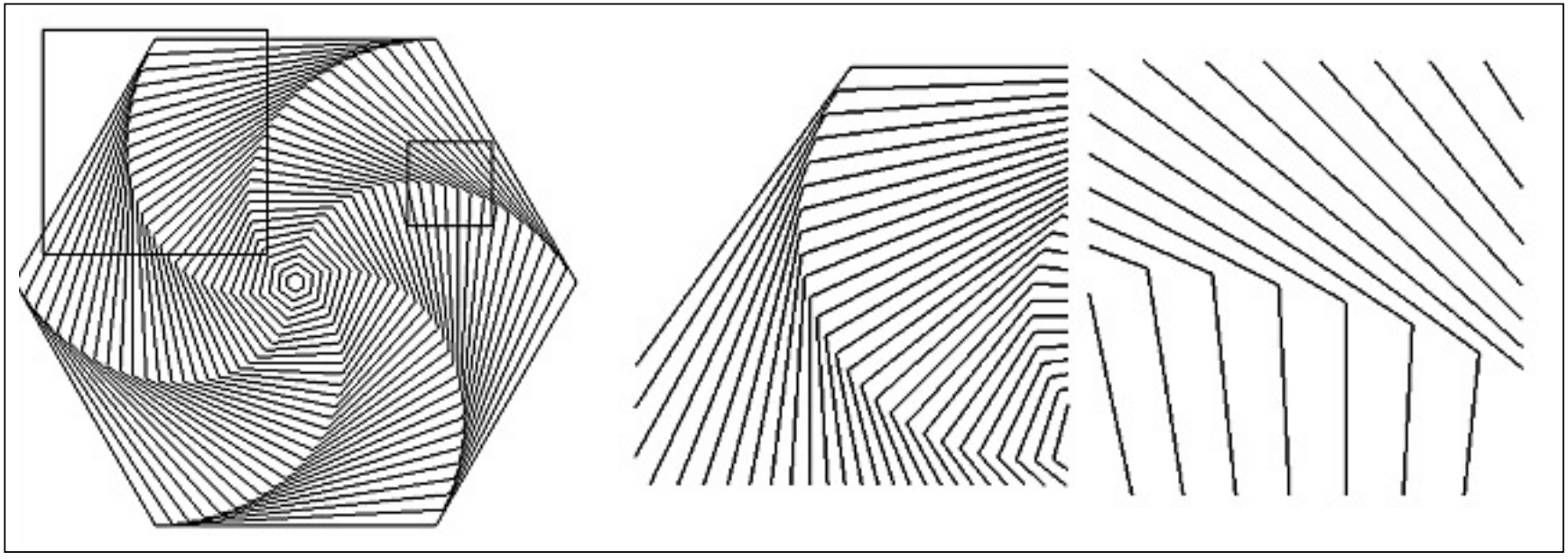
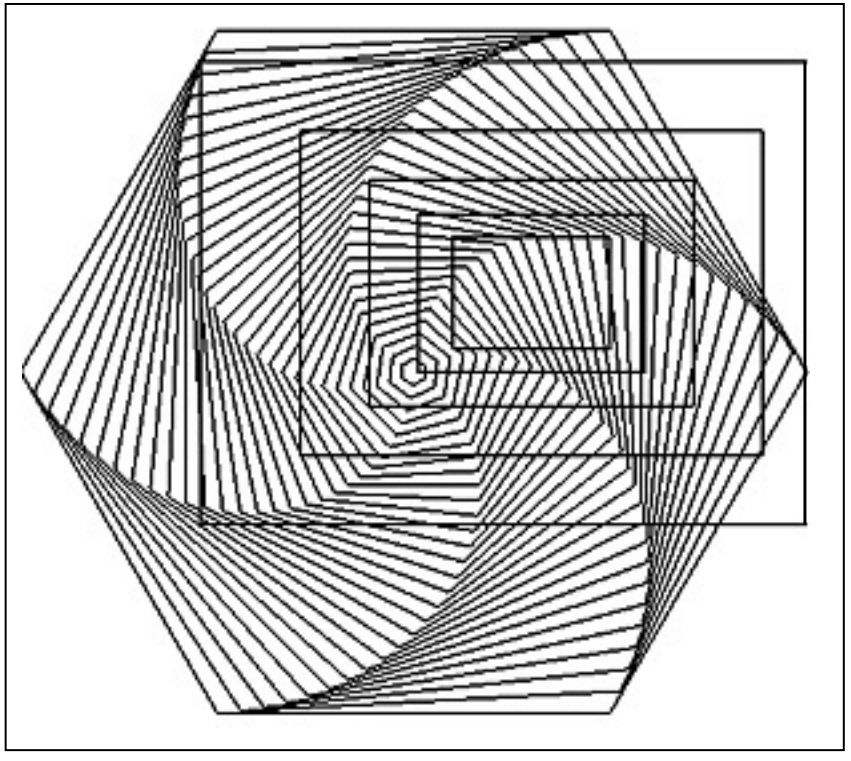


FIGURE 3.12 Zooming in on the swirl of hexagons.



```
float cx = 0.3, cy = 0.2; //center of the window
float H, W = 1.2, aspect = 0.7; // window properties
set the viewport
for(int frame = 0; frame < NumFrames; frame++) // for each frame
{
      clear the screen       // erase the previous figure
      W *= 0.7;              // reduce the window width
      H = W * aspect; // maintain the same aspect ratio
      setWindow(cx - W, cx + W, cy - H, cy + H); //set the next window
      hexSwirl();      // draw the object
}
```

FIGURE 3.13 Making an animation.

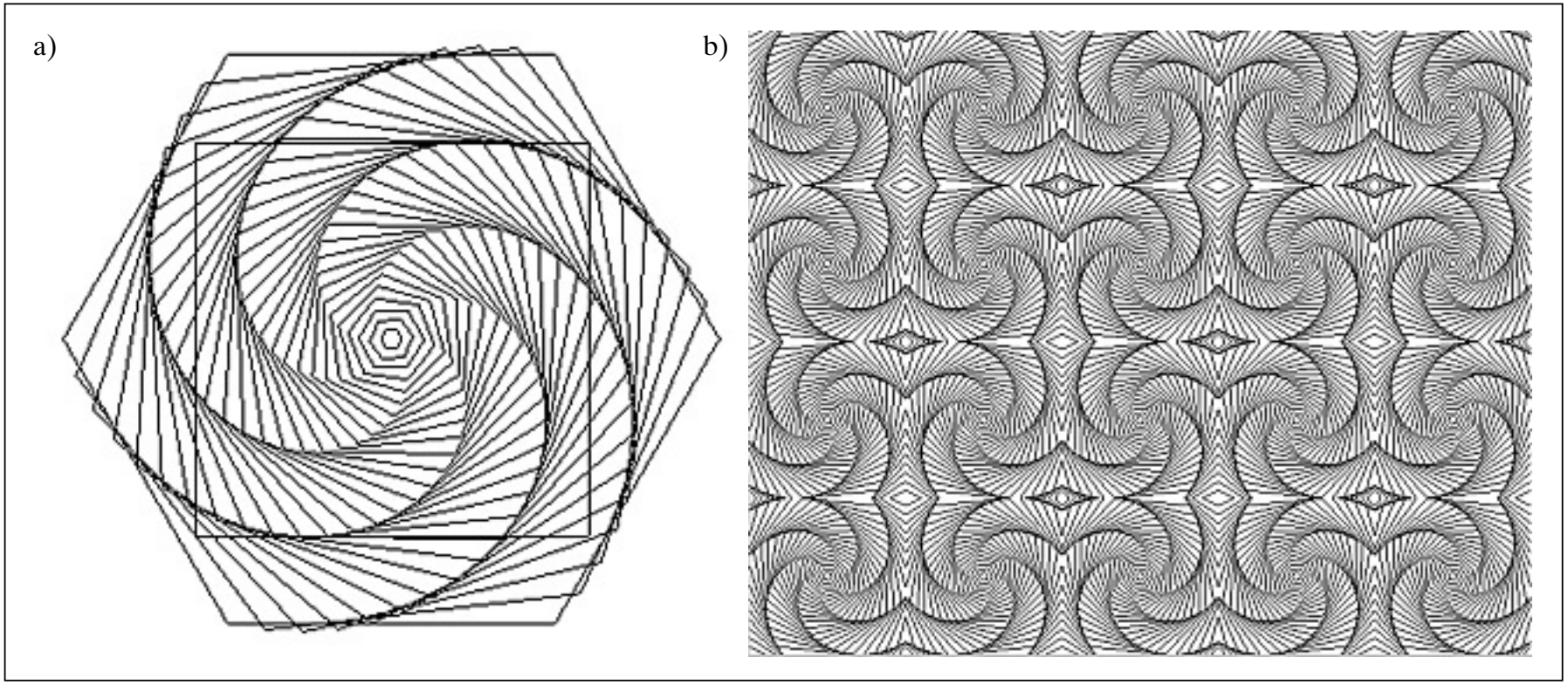


FIGURE 3.14 (a) Whirling hexagons in a fixed window. (b) A tiling formed using many viewports.

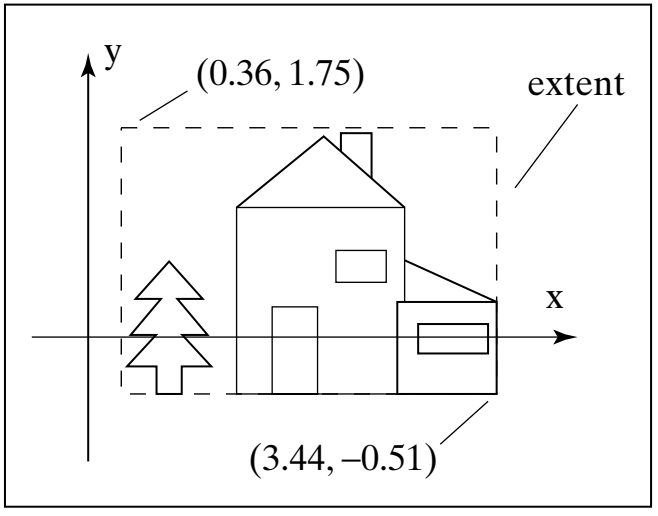
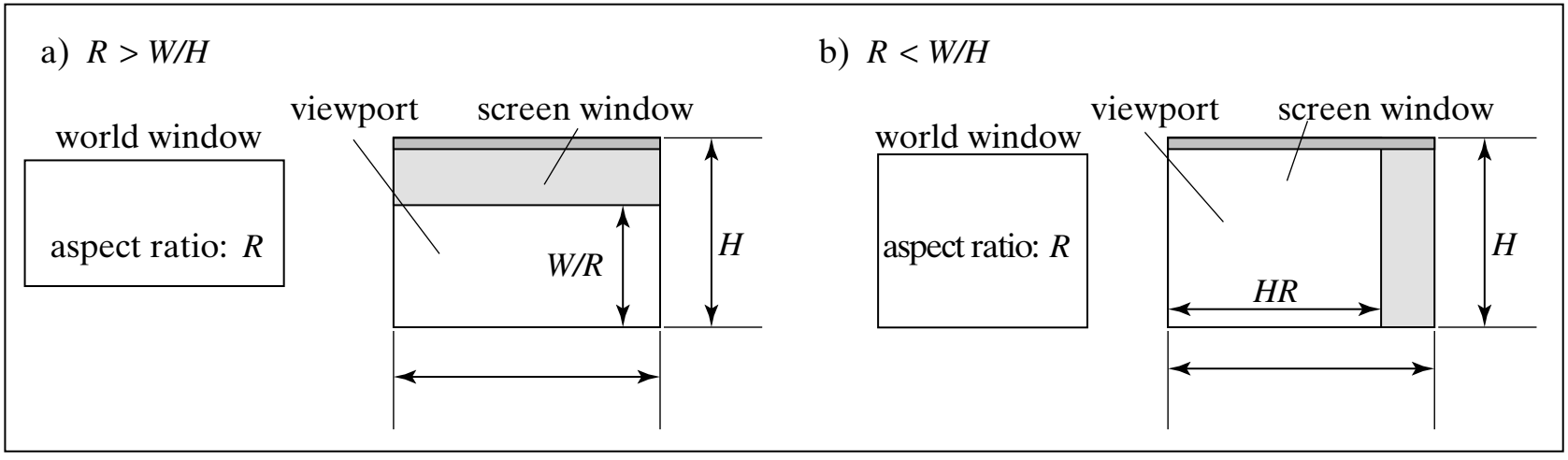


FIGURE 3.15 Using the extent as the window.

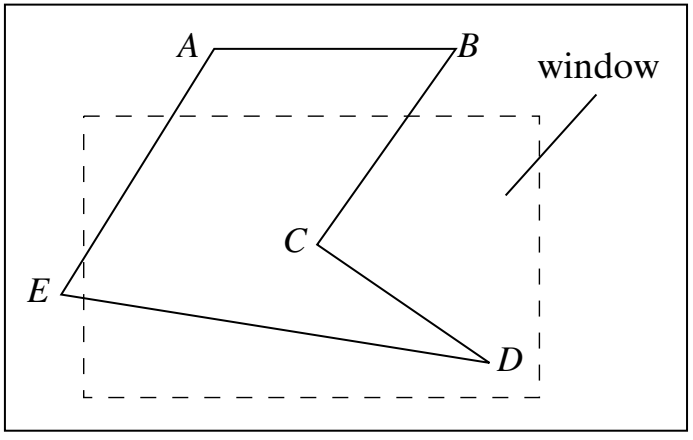
FIGURE 3.16 Possible aspect ratios for the world and screen windows.



```
void myReshape(GLsizei W, GLsizei H)
{
      if(R > W/H) // use (global) window aspect ratio
            setViewport(0, W, 0, W/R);
      else
            setViewport(0, H * R, 0, H);
}
```

FIGURE 3.17 Using a reshape function to set the largest matching viewport upon a resize event.

FIGURE 3.18 Clipping lines at window boundaries.



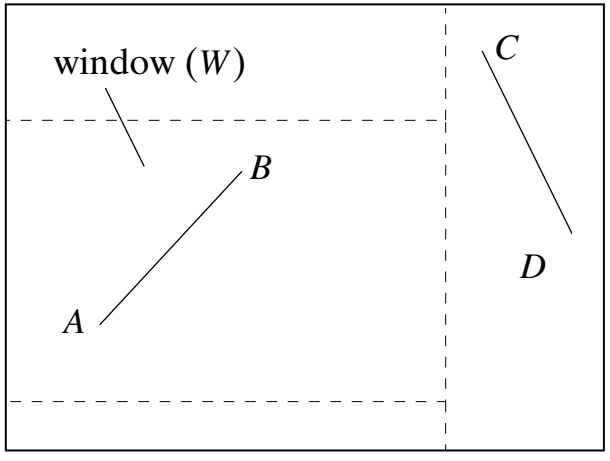


FIGURE 3.19 Trivial acceptance or rejection of a line segment.

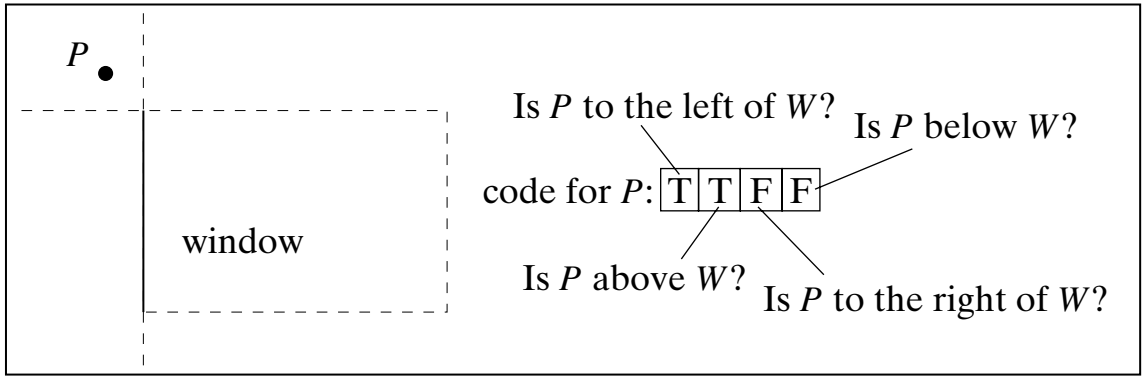


FIGURE 3.20 Encoding how point P is disposed with respect to the window.

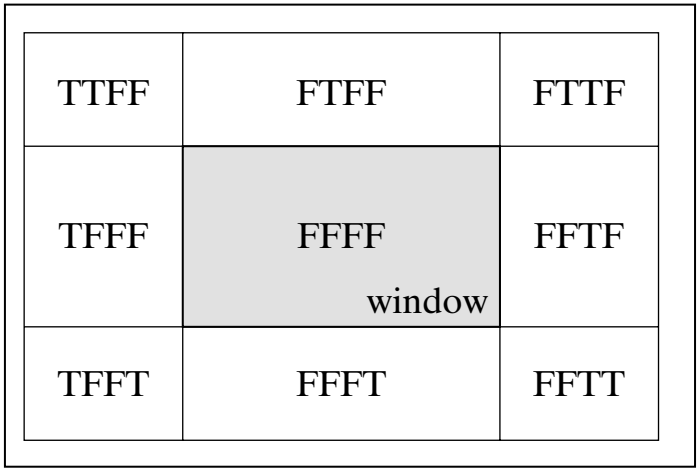
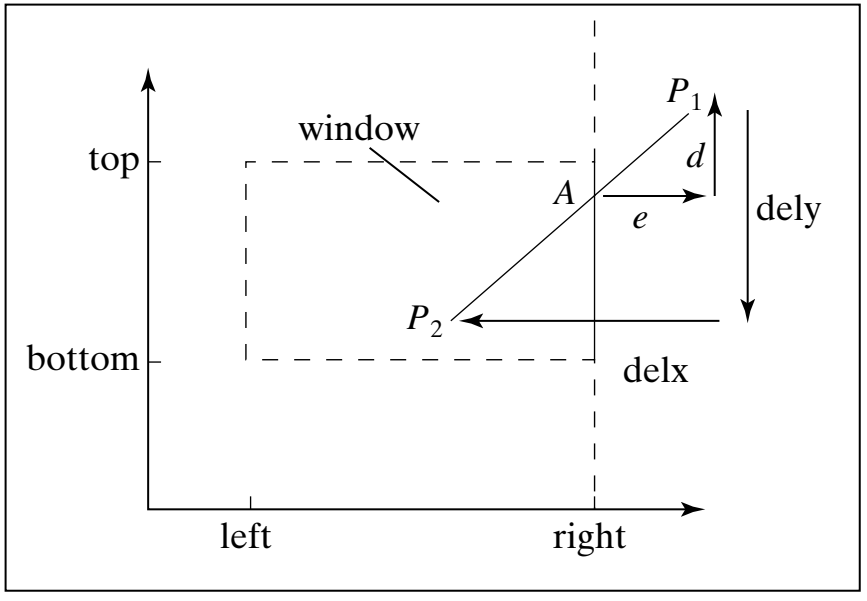


FIGURE 3.21 Inside–outside codes for a point.

FIGURE 3.22 Clipping a segment against an edge.



```
int clipSegment(Point2& p1, Point2& p2, RealRect W)
{
     do{
          if(trivial accept) return 1; // some portion survives
          if(trivial reject) return 0; // no portion survives

          if(p1 is outside )
          {
             if(p1 is to the left )  chop against the left edge
             else if(p1 is to the right ) chop against the right edge
             else if(p1 is below)  chop against the bottom edge
             else if(p1 is above)  chop against the top edge
          }
          else      // p2 is outside
          {
            if(p2 is to the left ) chop against the left edge
                 else if(p2 is to the right ) chop against the right edge
            else if(p2 is below)  chop against the bottom edge
            else if(p2 is above) chop against the top edge
          }
     }while(1);
}
```

FIGURE 3.23 The Cohen–Sutherland line clipper (pseudocode).

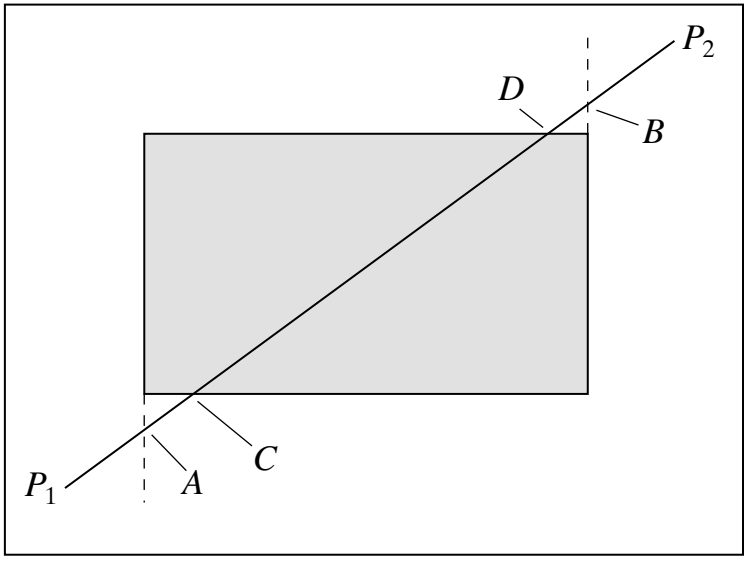


FIGURE 3.24 A segment that requires four clips.

```
class Canvas {
 public:
   Canvas(int width, int height, char* windowTitle); // constructor
   void setWindow(float l, float r, float b, float t);
   void setViewport(int l, int r, int b, int t);
   IntRect getViewport(void);  // divulge the viewport data
   RealRect getWindow(void); // divulge the window data
   float getWindowAspectRatio(void);
   void clearScreen();
   void setBackgroundColor(float r, float g, float b);
   void setColor(float r, float g, float b);
   void lineTo(float x, float y);
   void lineTo(Point2 p);
   void moveTo(float x, float y);
   void moveTo(Point2 p);
   others later
 private:
   Point2 CP;        // current position in the world
   IntRect viewport; // the current window
   RealRect window;  // the current viewport
   others later
};
```

FIGURE 3.25 The header file `Canvas.h`.

```
Canvas cvs(640, 480, "try out Canvas");   // global canvas object

//<<<<<<<<<<<<<<<<<<<<<<<<<<<< display >>>>>>>>>>>>>>>>>>>>>>>>
void display(void)
{
  cvs.clearScreen();  // clear screen
  cvs.setWindow(-10.0, 10.0, -10.0, 10.0);
  cvs.setViewport(10, 460, 10, 460);
  cvs.moveTo(0, -10.0);  // draw a line
  cvs.lineTo(0, 10.0);
  RealRect box( -2.0, 2.0, -1.0, 1.0); // construct a box
  box.draw(); // draw the box
  . . .
}
//<<<<<<<<<<<<<<<<<<<< main >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
void main(void)
{
  // the window is opened in the Canvas constructor
  cvs.setBackgroundColor(1.0, 1.0, 1.0); // background is white
  cvs.setColor(0.0, 0.0, 0.0);  // set drawing color
  glutDisplayFunc(display);
  glutMainLoop();
}
```

FIGURE 3.26 Typical usage of the *Canvas* class.

FIGURE 3.27 The constructor for the OpenGL version of *Canvas* .

```
//<<<<<<<<<<<<<<<<<<<< Canvas constructor >>>>>>>>>>>>>>>>>>>
Canvas:: Canvas(int width, int height, char* windowTitle)
{
  char* argv[1];    // dummy argument list for glutInit()
  char dummyString[8];
  argv[0] = dummyString; // hook up the pointer
  int argc = 1;     // to satisfy glutInit()

  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(width, height);
  glutInitWindowPosition(20, 20);
  glutCreateWindow(windowTitle); // open the screen window
  setWindow(0, (float)width, 0, (float)height); //default world window
  setViewport(0, width, 0, height); // default viewport
  CP.set(0.0f, 0.0f);   // initialize the CP to (0, 0)
}
```

```
//<<<<<<<<<<<<<<<<<<<<<<< moveTo >>>>>>>>>>>>>>>>>>>>>>>
void Canvas:: moveTo(float x, float y)
{
  CP.set(x, y);
}
//<<<<<<<<<<<<<<<<<<<<<<< lineTo >>>>>>>>>>>>>>>>>>>>>>>
void Canvas:: lineTo (float x, float y)
{
  glBegin (GL_LINES);
    glVertex2f ((GLfloat) CP.x, (GLfloat) CP.y);
    glVertex2f ((GLfloat) x, (GLfloat) y);     // draw the line
  glEnd();
  CP.set (x, y);     // update the CP
  glFlush();
}
//<<<<<<<<<<<<<<<<<<<<<<< set Window >>>>>>>>>>>>>>>>>>>>>>>
void Canvas:: setWindow (float l, float r, float b, float t)
{
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D ((GLdouble)l, (GLdouble)r, (GLdouble)b, (GLdouble)t);
  window.set (l, r, b, t);
}
```

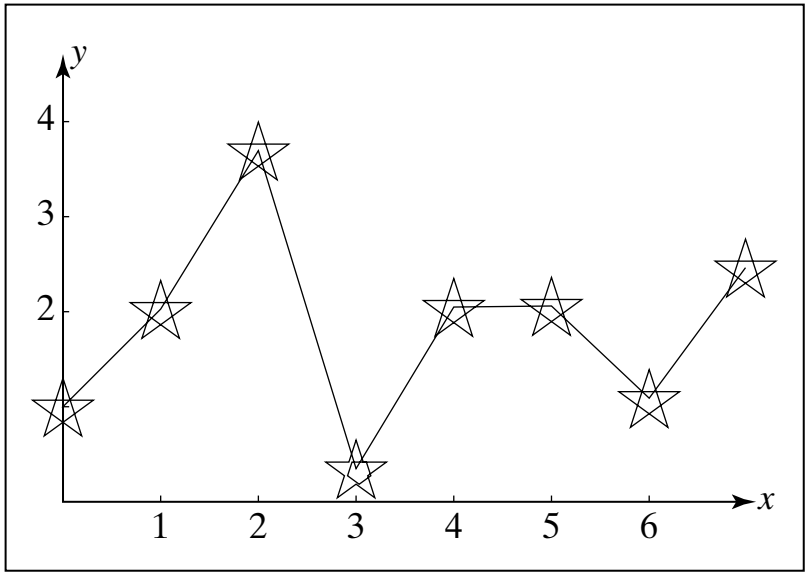
FIGURE 3.28 Implementation of some *Canvas* member functions.

```
void Canvas :: moveRel(float dx, float dy)
{
      CP.set(CP.x + dx, CP.y + dy);
}

void Canvas :: lineRel(float dx, float dy)
{
      float x = CP.x + dx, y = CP.y + dy;
      lineTo(x, y);
      CP.set(x, y);
}
```

FIGURE 3.29 The functions `moveRel()` and `lineRel()`.

FIGURE 3.30 Placing markers for emphasis.



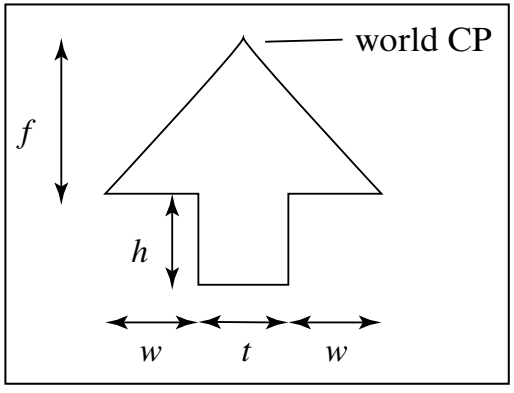


FIGURE 3.31 Model of an arrow.

```
void arrow(float f, float h, float t, float w)
{ // assumes global Canvas object: cvs
      cvs.lineRel(-w - t / 2, -f);   // down the left side
      cvs.lineRel(w, 0);
      cvs.lineRel(0, -h);
      cvs.lineRel(t, 0);     // across
      cvs.lineRel(0, h);     // back up
      cvs.lineRel(w, 0);
      cvs.lineRel(-w - t / 2, f);
}
```

FIGURE 3.32 Drawing an arrow using relative moves and draws.

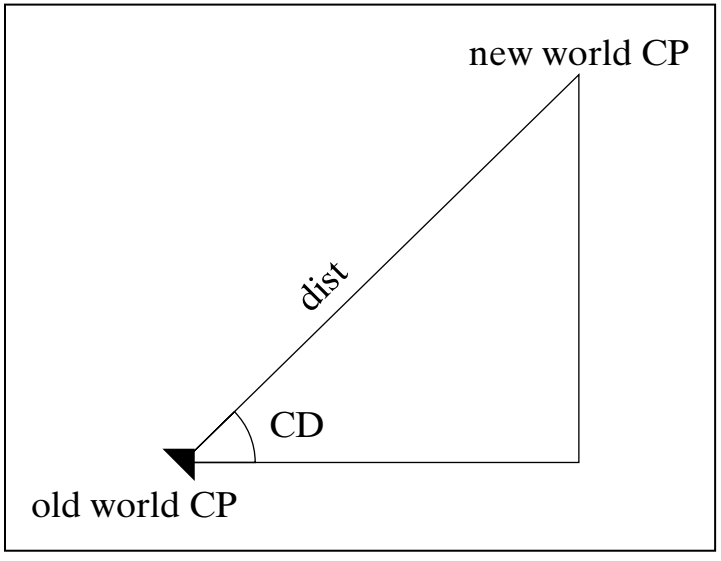


FIGURE 3.33 Effect of the `forward()` routine.

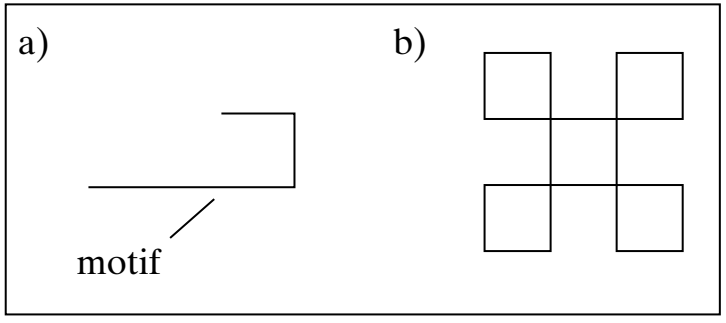


FIGURE 3.34 Building a figure out of several turtle motions.

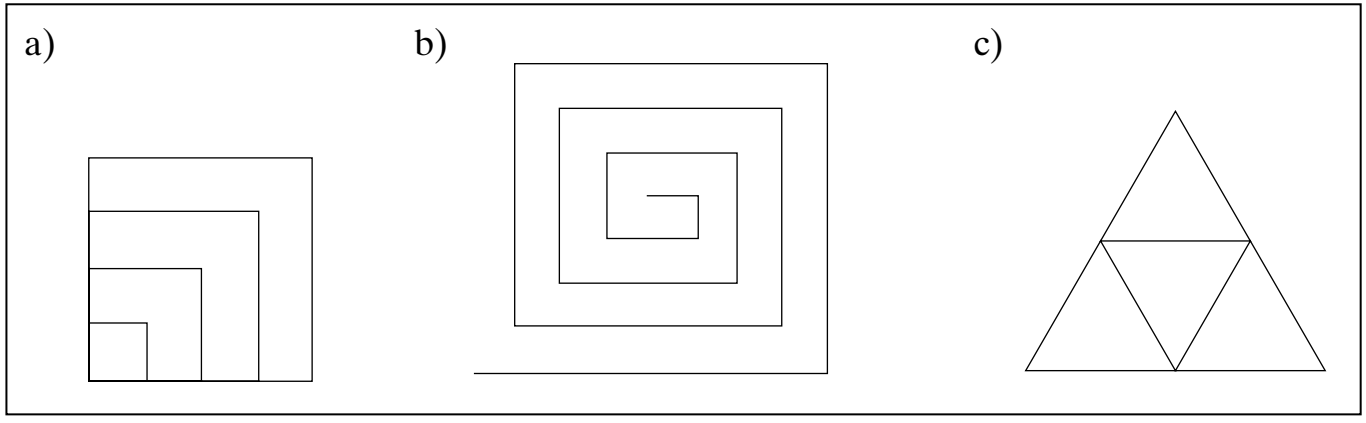


FIGURE 3.36 Other simple turtle figures.

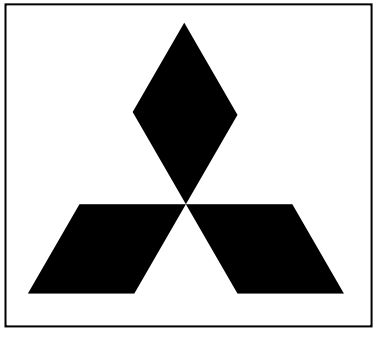


FIGURE 3.37 A famous logo.

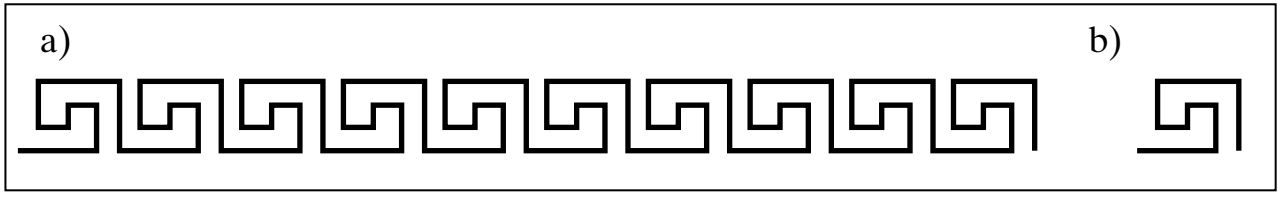


FIGURE 3.38 Example of a meander.

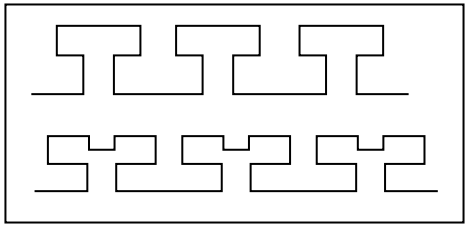
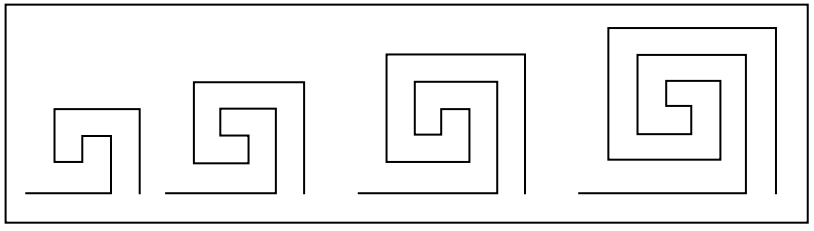


FIGURE 3.39 Additional figures for meanders.

FIGURE 3.40 Hierarchy of meander motifs.



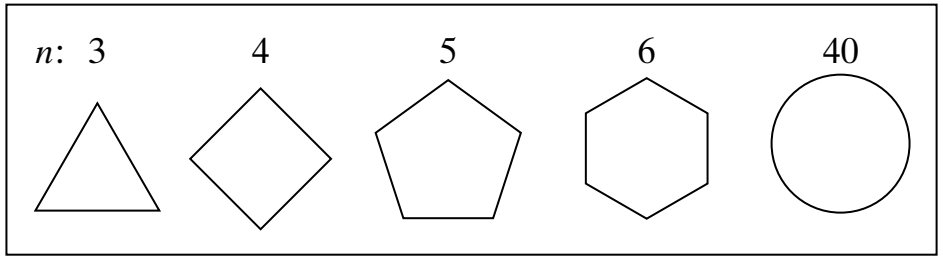


FIGURE 3.41 Examples of n-gons.

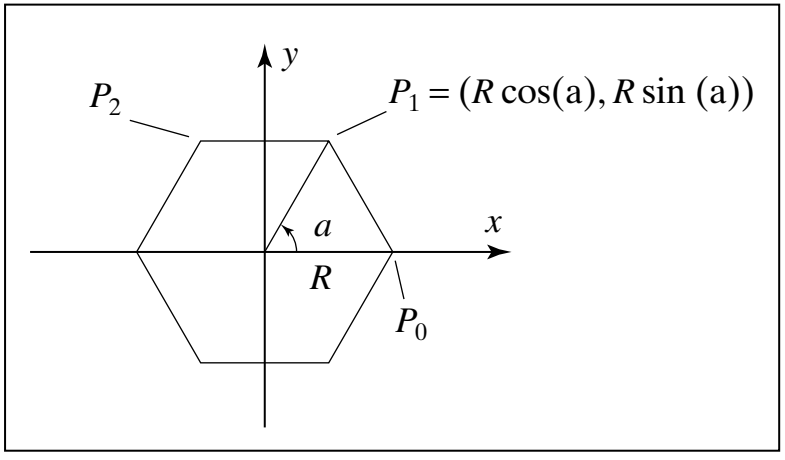


FIGURE 3.42 Finding the vertices of a 6-gon.

FIGURE 3.43 Building an *n*-gon in memory.

```
void ngon(int n, float cx, float cy, float radius, float rotAngle)
{       // assumes global Canvas object, cvs
   if(n < 3) return;       // bad number of sides
   double angle = rotAngle * 3.14159265 / 180;  // initial angle
   double angleInc = 2 * 3.14159265 /n;         //angle increment
   cvs. moveTo(radius * cos(angle) + cx, radius * sin(angle) + cy);
   for(int k = 0; k < n; k++)  // repeat n times
   {
       angle += angleInc;
       cvs.lineTo(radius * cos(angle) + cx, radius * sin(angle) + cy);
   }
}
```

FIGURE 3.44 Drawing a hexagon.

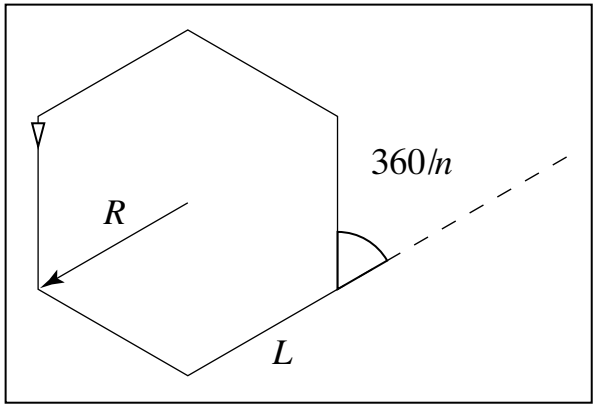
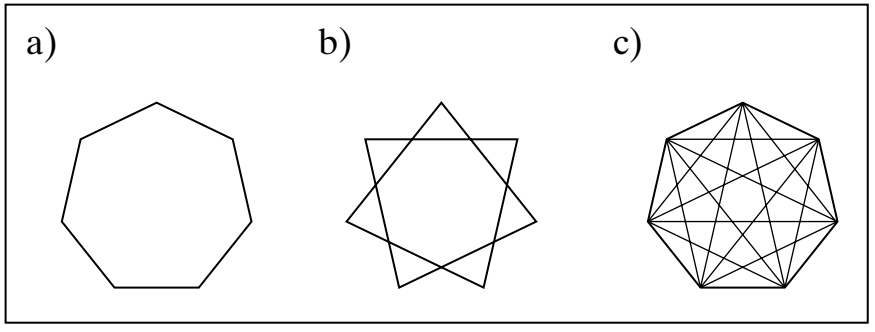


FIGURE 3.45 A 7-gon and its offspring. (a) The 7-gon. (b) A stellation. (c) A "7-rosette."



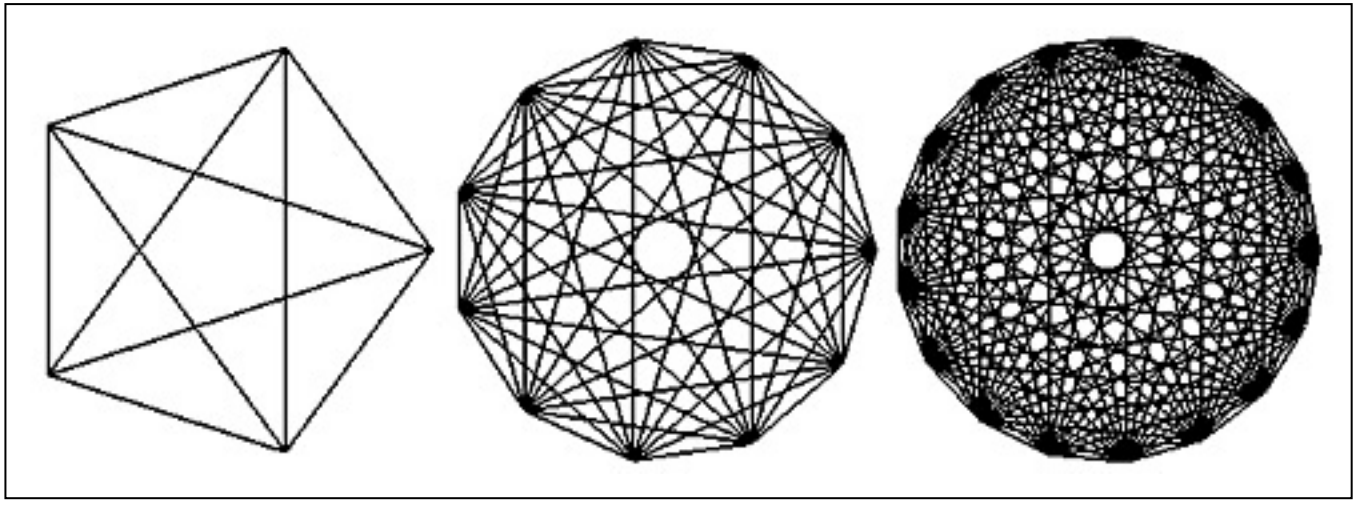


FIGURE 3.46 The 5-, 11-, and 17-rosettes.

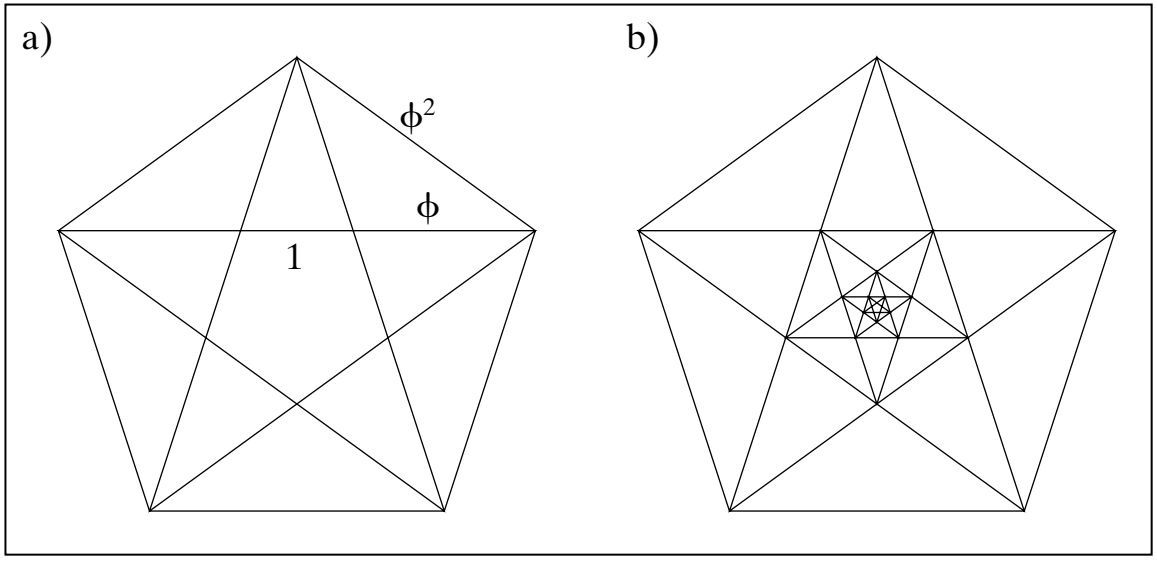


FIGURE 3.47 5-Rosette and infinite regressions of pentagons and pentagrams.

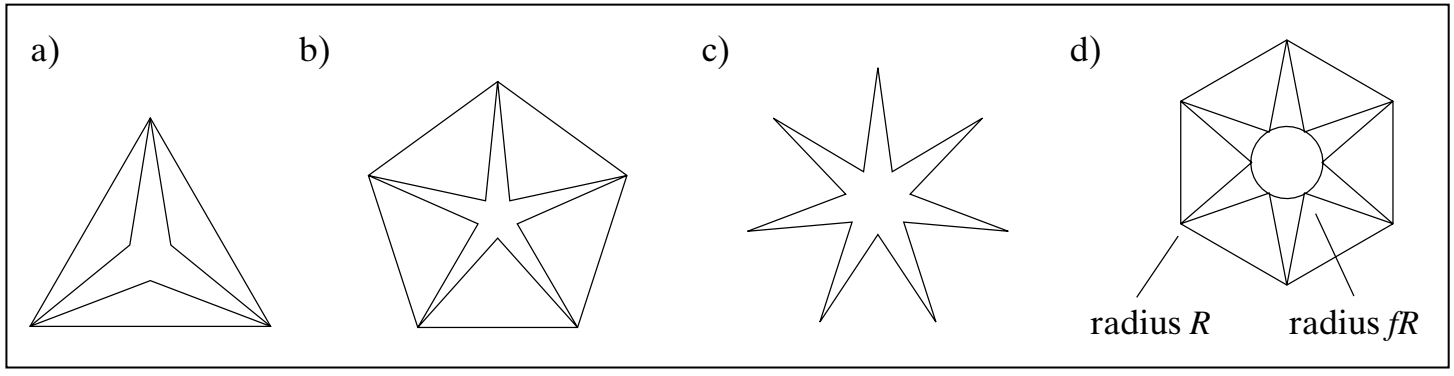


FIGURE 3.48 A family of famous logos.

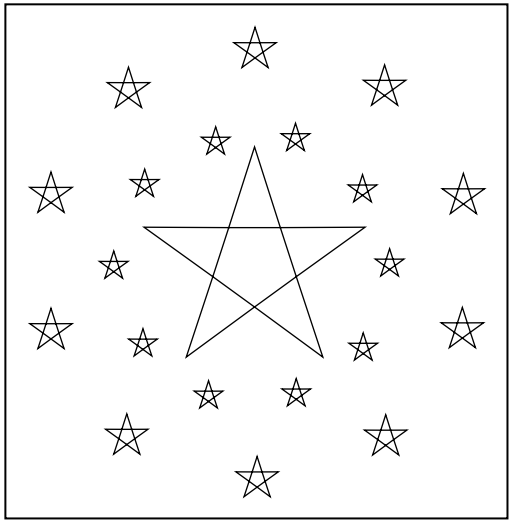


FIGURE 3.49 A star pattern.

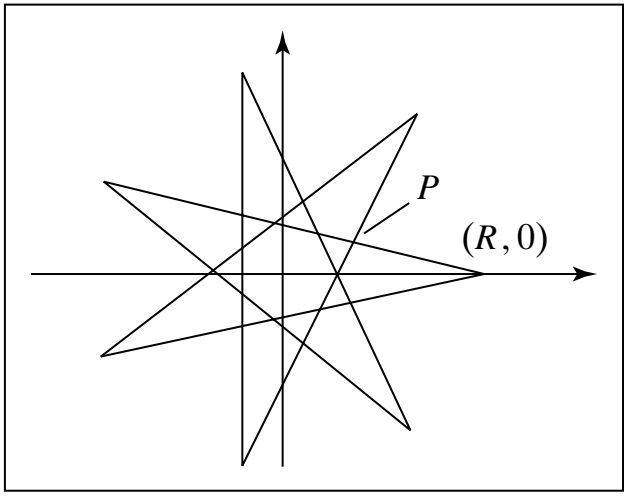


FIGURE 3.50 A "7-gram."

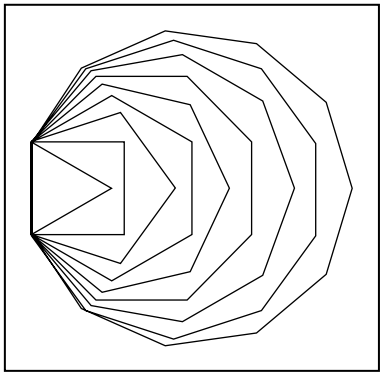


FIGURE 3.51 *n*-Gons sharing a common edge.

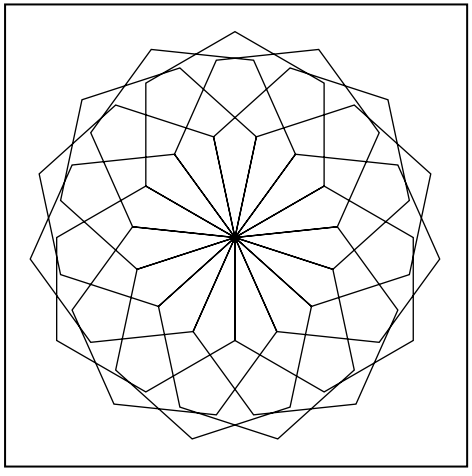


FIGURE 3.52 Repeated use of turtle commands.

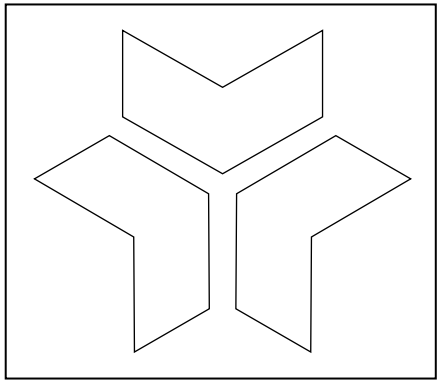


FIGURE 3.53 Logo of the University of Massachusetts.

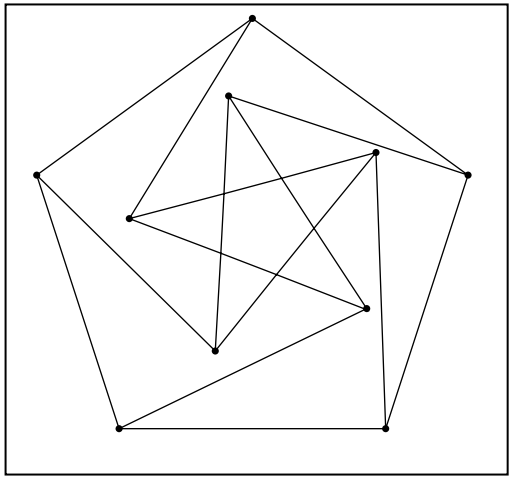
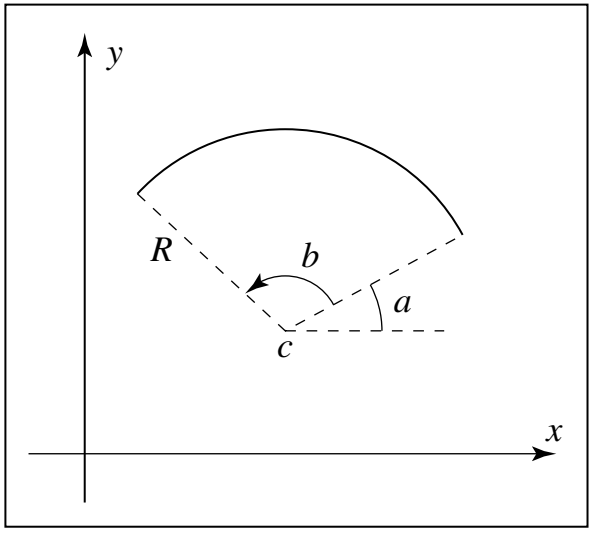


FIGURE 3.54 Rotating "penta-things."

```
void drawCircle(Point2 center, float radius)
{
      const int numVerts = 50;
       ngon(numVerts, center.getX(), center.getY(), radius, 0);
}
```

FIGURE 3.55 Drawing a circle based on a 50-gon.

FIGURE 3.56 Defining an arc.



```
void drawArc(Point2 center, float radius, float startAngle, float sweep)
{    // startAngle and sweep are in degrees
   const int n = 30; // number of intermediate segments in arc
   float angle = startAngle * 3.14159265 / 180; // initial angle in radians
   float angleInc = sweep * 3.14159265 /(180 * n); // angle increment
   float cx = center.getX(), cy = center.getY();
   cvs.moveTo(cx + radius * cos(angle), cy + radius * sin(angle));
   for(int k = 1; k < n; k++, angle += angleInc)
       cvs.lineTo(cx + radius * cos(angle), cy + radius * sin(angle));
}
```

FIGURE 3.57 Drawing an arc of a circle.

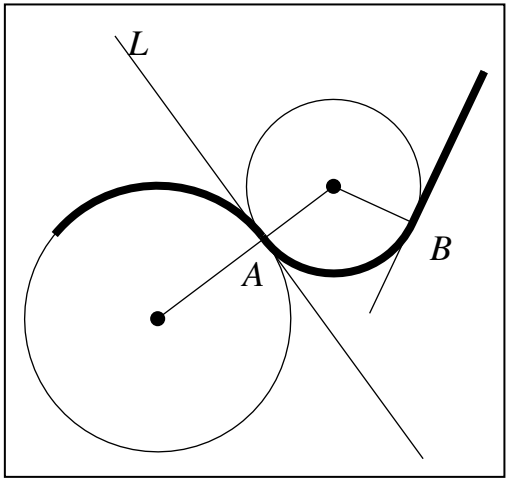


FIGURE 3.58 Blending arcs using tangent circles.

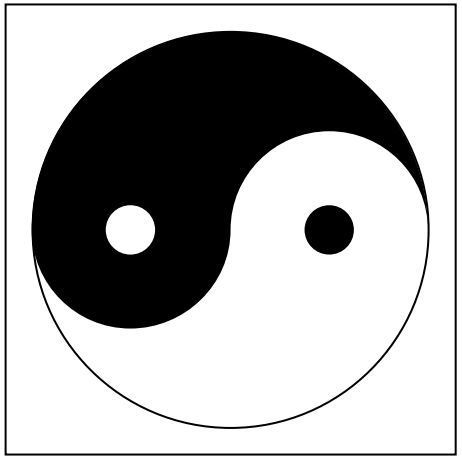


FIGURE 3.59 The yin–yang symbol.

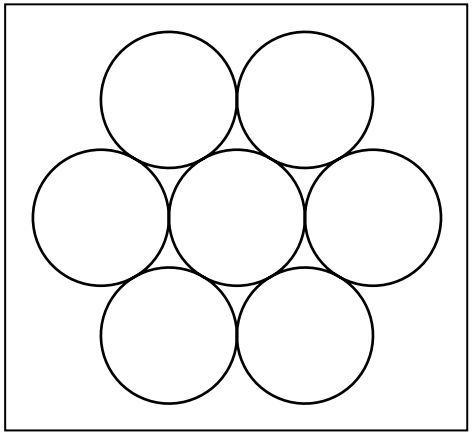


FIGURE 3.60 The seven circles.

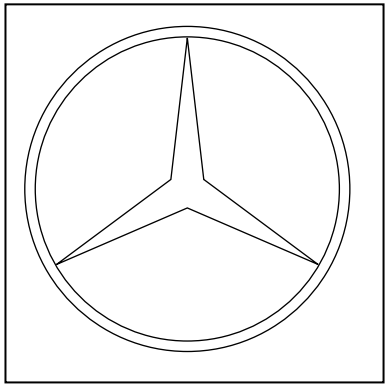


FIGURE 3.61 A famous logo.

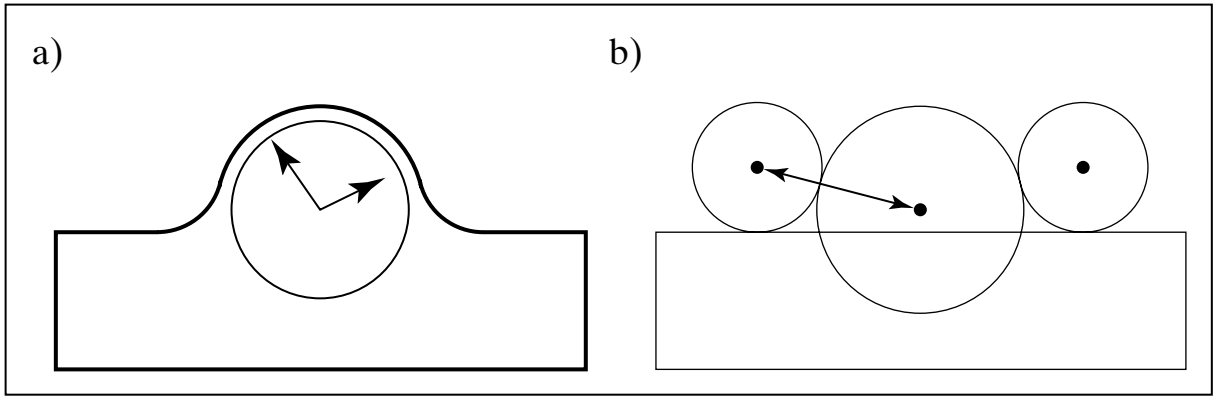


FIGURE 3.62 Blending arcs to form smooth curves.

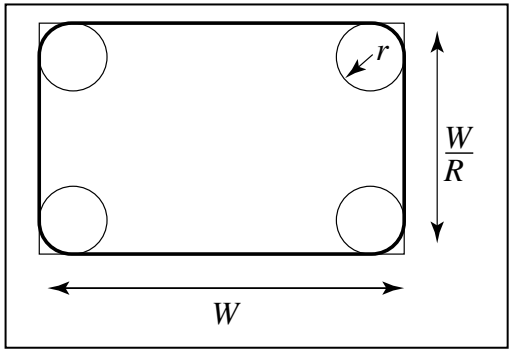
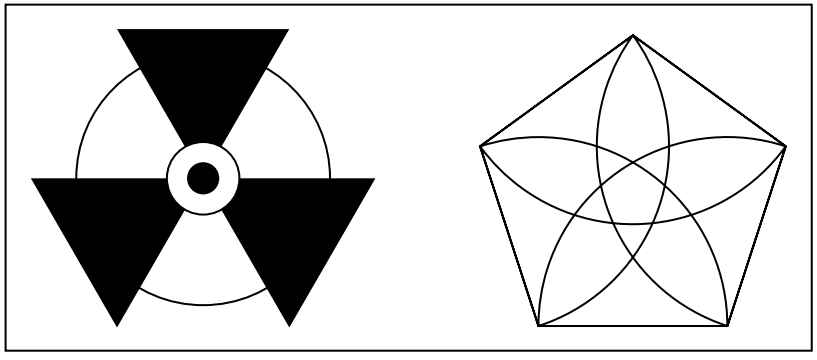


FIGURE 3.63 A rounded rectangle.

FIGURE 3.64 Shapes based on arcs.



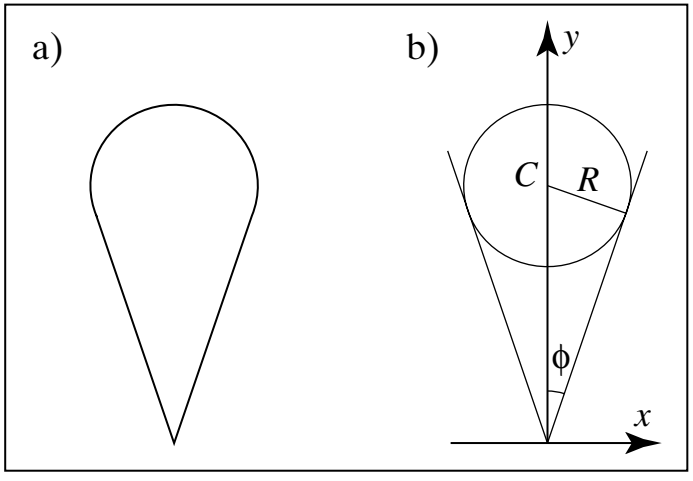


FIGURE 3.65 The teardrop and its construction.

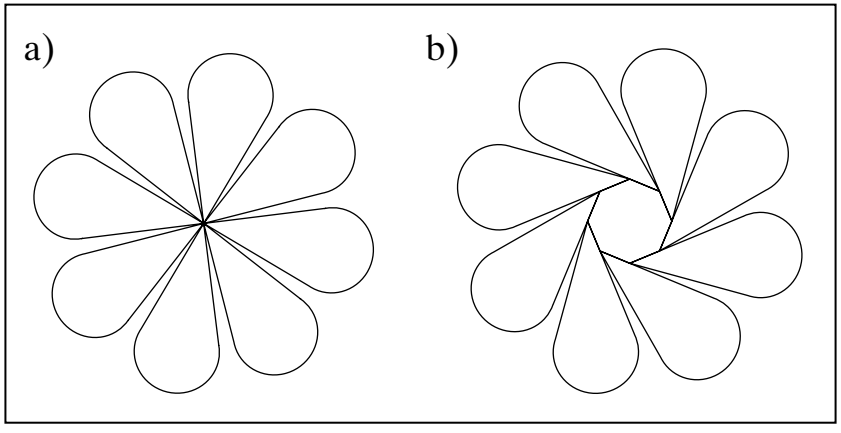


FIGURE 3.66 Some figures based on the teardrop.

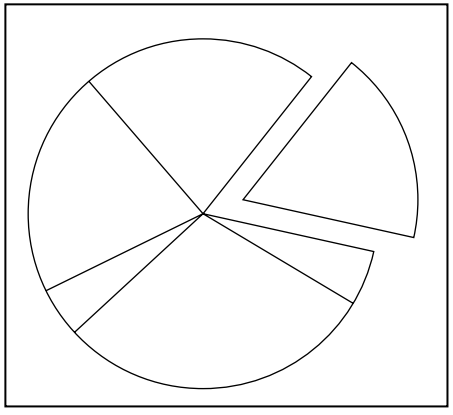


FIGURE 3.67 A pie chart.

FIGURE 3.68 Etch-a-Sketch drawings of parametric curves. (Drawing by Suzanne Casiello.)

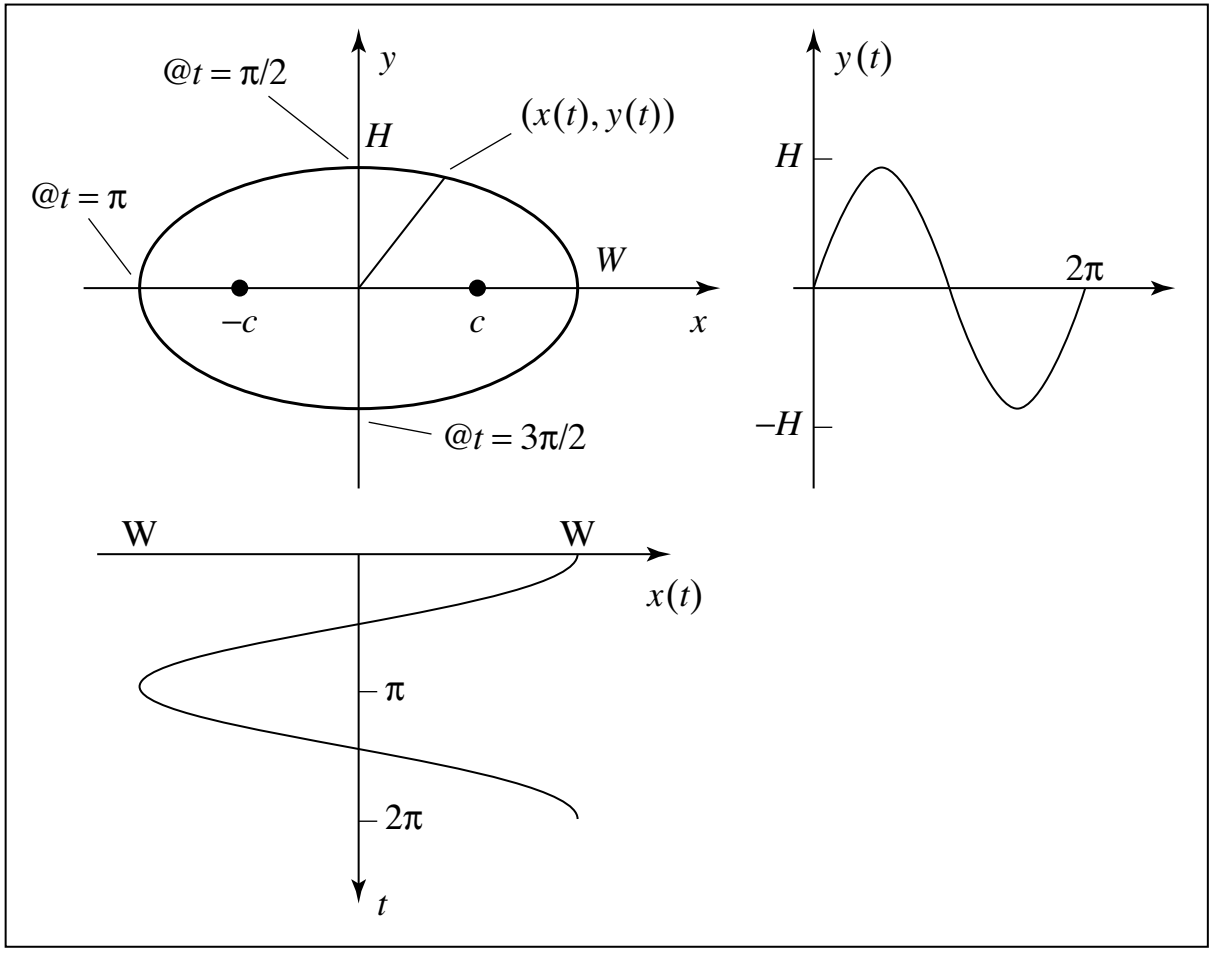
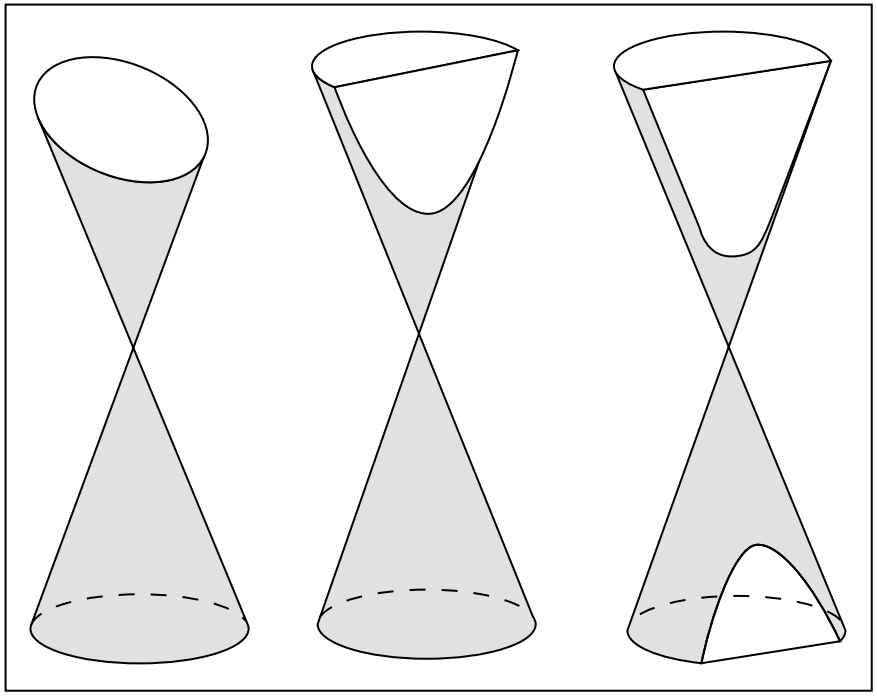


FIGURE 3.69 An ellipse described parametrically.

FIGURE 3.70 The classical conic sections.



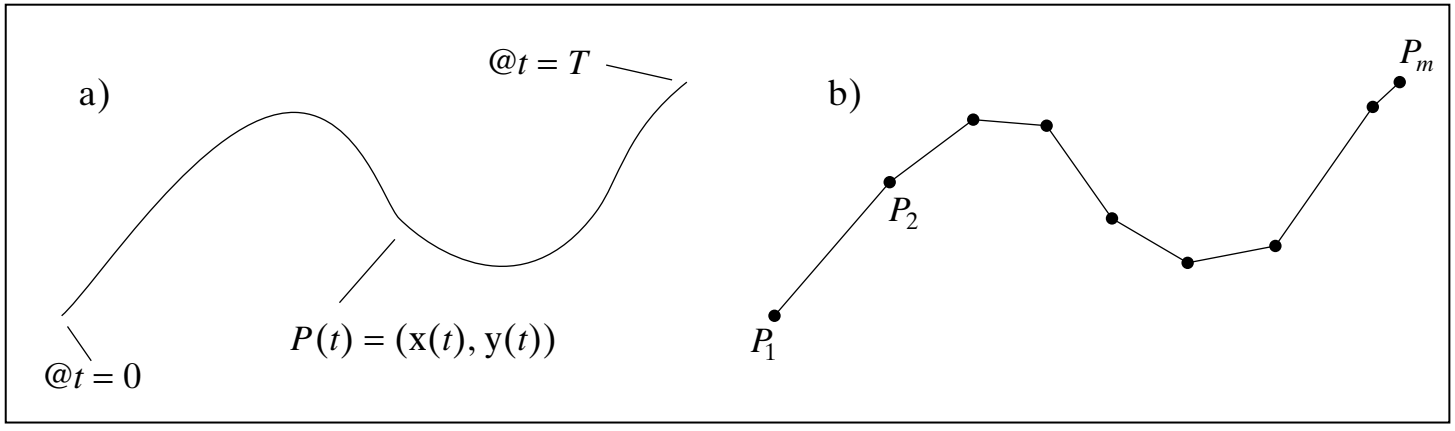


FIGURE 3.71 Approximating a curve by a polyline.

```
// draw the curve (x(t), y(t)) using
// the array t[0],..,t[n-1] of "sample-times"

glBegin(GL_LINES);
   for(int i = 0; i < n; i++)
      glVertex2f(x(t[i]), y(t[i]));
glEnd();
```

FIGURE 3.72 Drawing an ellipse using points equispaced in *t*.

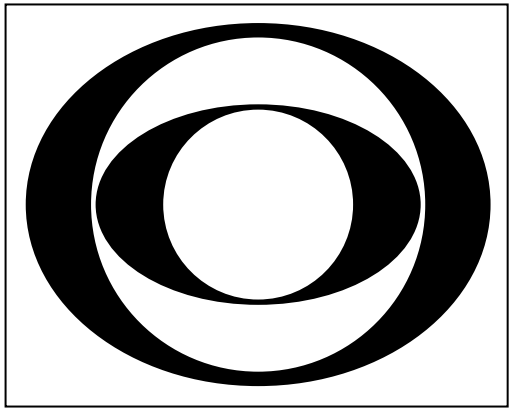


FIGURE 3.73 A familiar "eye" made of circles and ellipses.

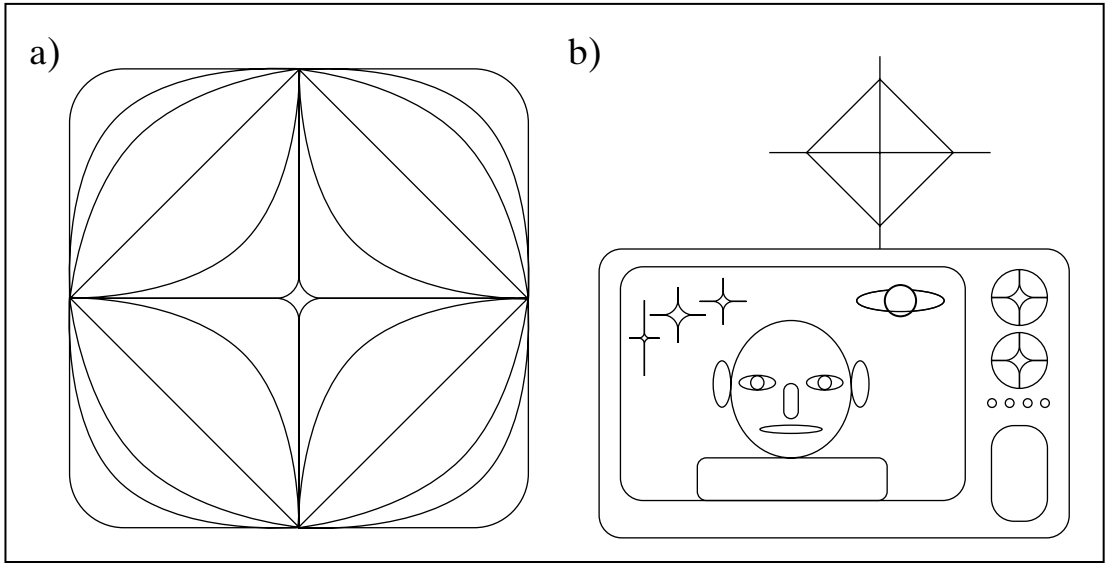


FIGURE 3.74 (a) Family of supercircles. (b) Scene composed of superellipses.

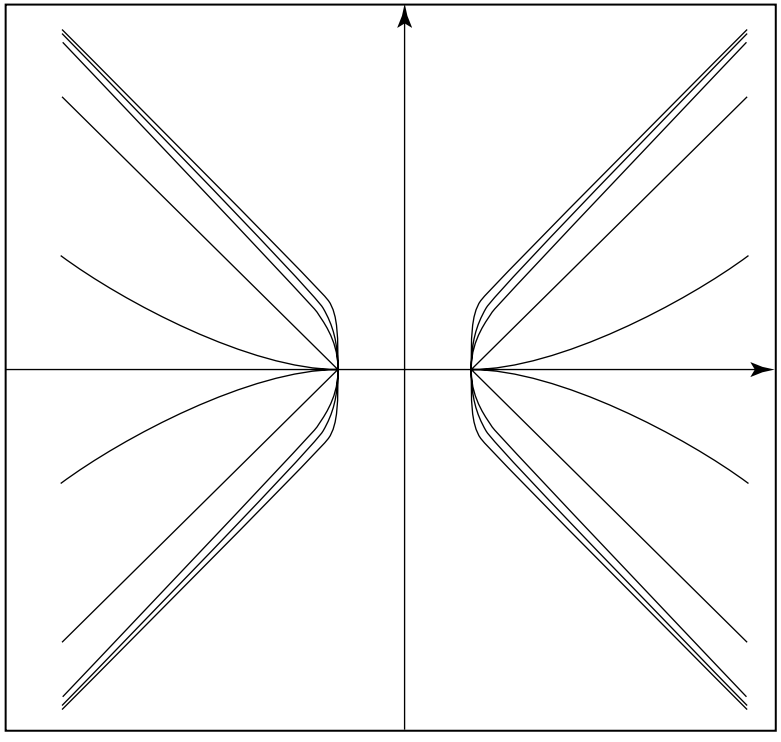


FIGURE 3.75 The superhyperbola family.

FIGURE 3.76 Polar coordinates.

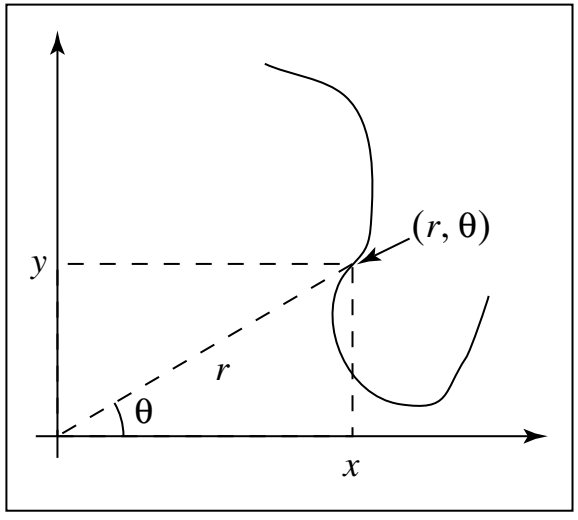
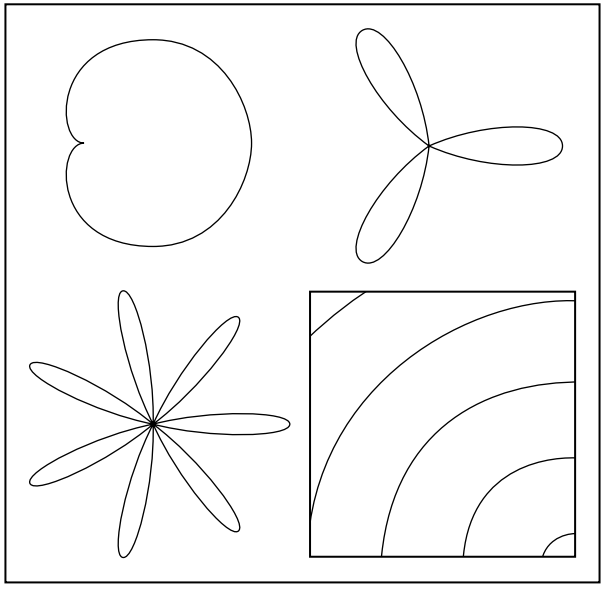


FIGURE 3.77 Examples of curves with simple polar forms.



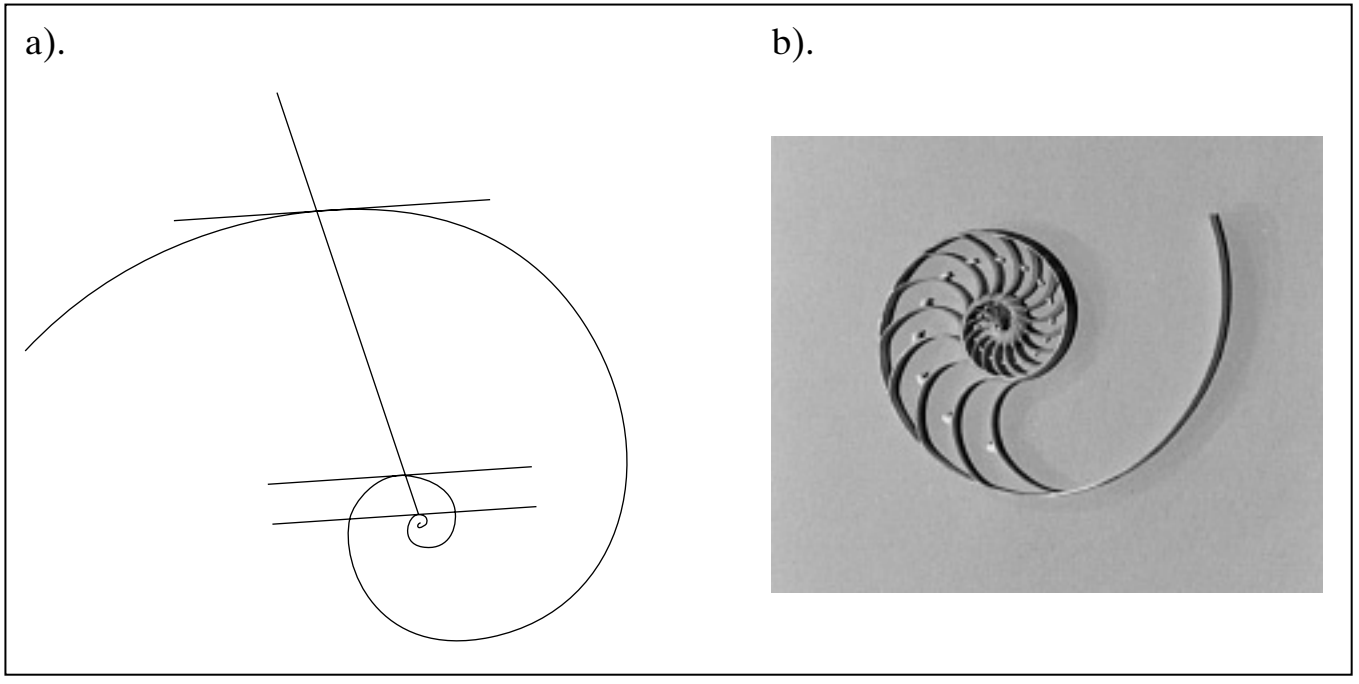


FIGURE 3.78 (a) The logarithmic spiral and (b) the chambered nautilus.

FIGURE 3.79 The helix, displayed as a stereo pair.

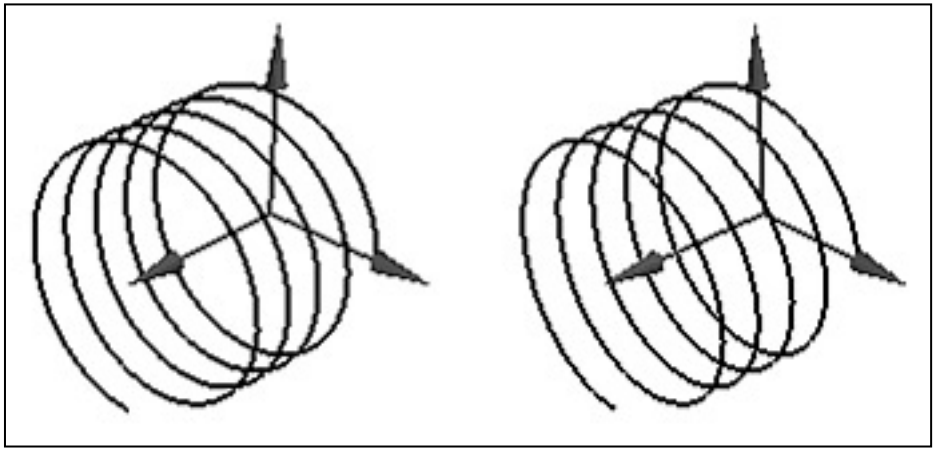


FIGURE 3.80 3.80. A toroidal spiral, displayed as a stereo pair.

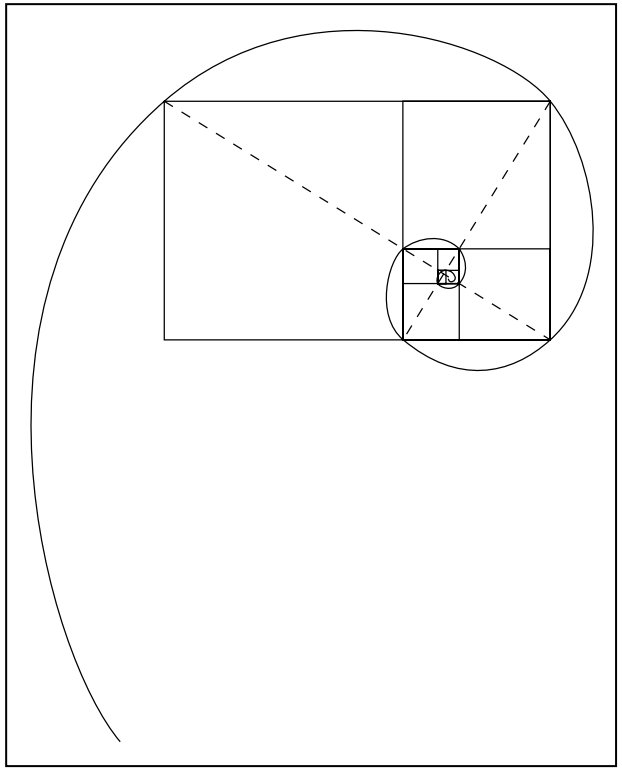
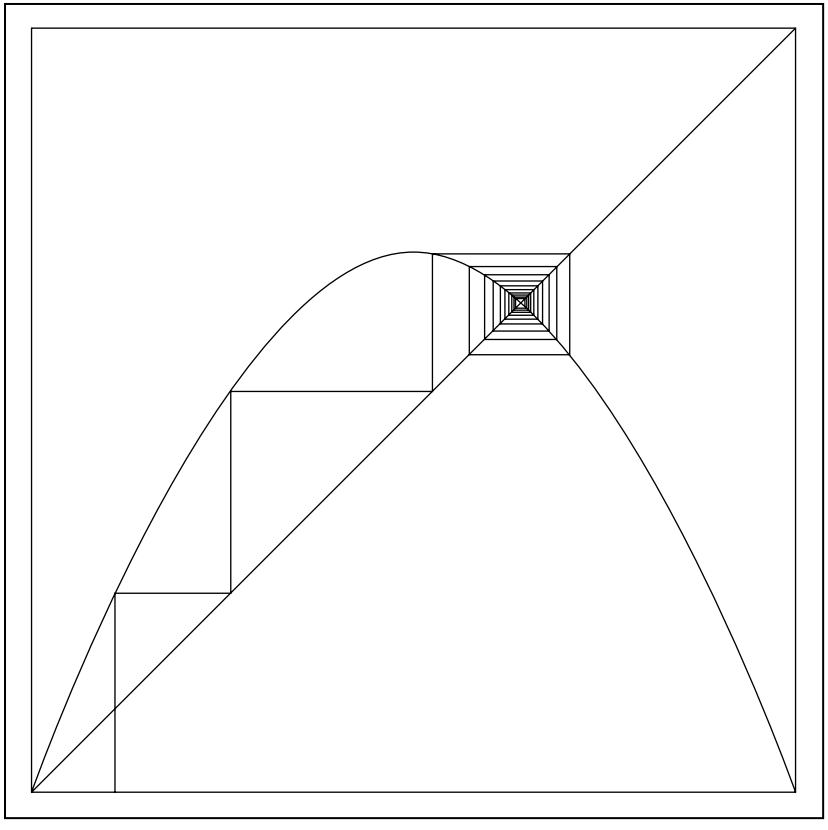


FIGURE 3.81 The spiral and the golden rectangle.

FIGURE 3.82 The logistic map for $\lambda = 0.7$.



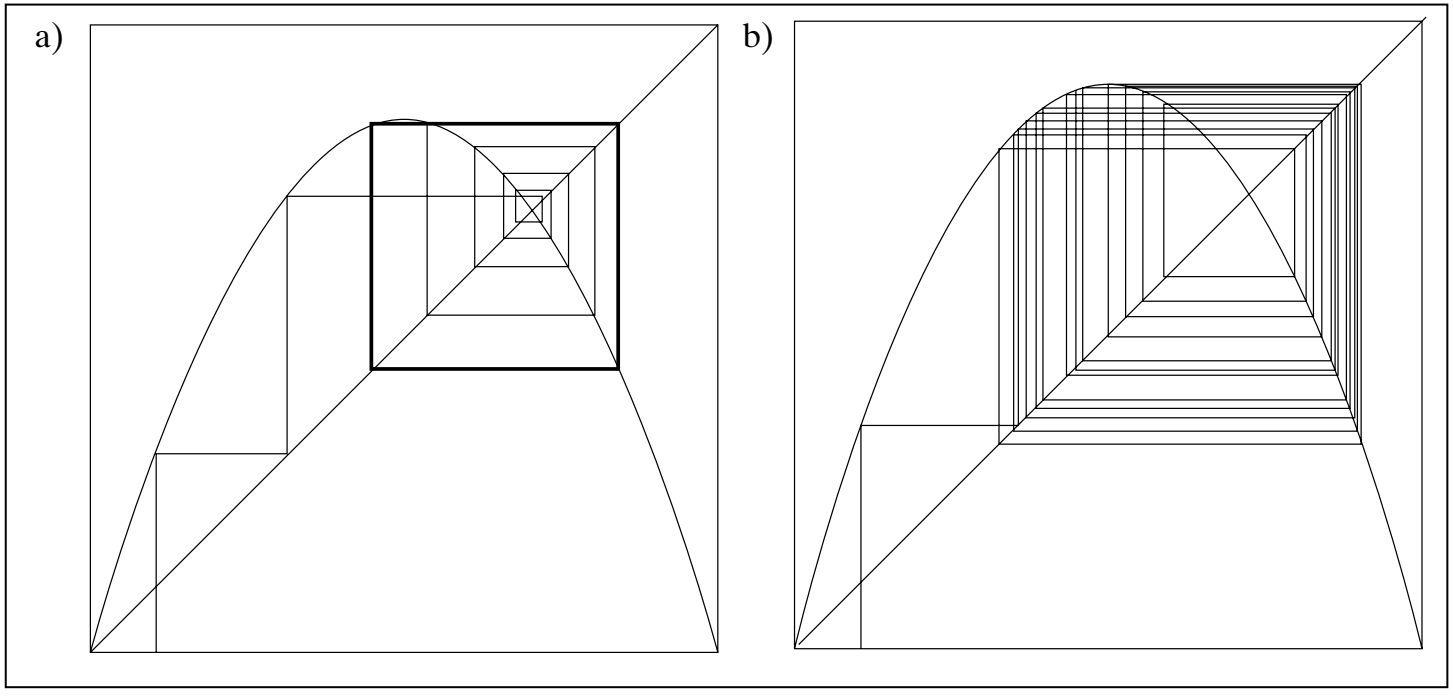


FIGURE 3.83 The logistic map for (a) $\lambda = 0.85$ and (b) $\lambda = 0.9$.

```
unsigned char code = 0;       // initially all bits are 0
...
if(P.x < window.l) code |= 8;    // set bit 3
if(P.y > window.t)  code |= 4;    // set bit 2
if(P.x > window.r) code |= 2;    // set bit 1
if(P.y < window.b) code |= 1;    // set bit 0
```

FIGURE 3.84 Setting bits in the "inside–outside" code word for a point *P*.

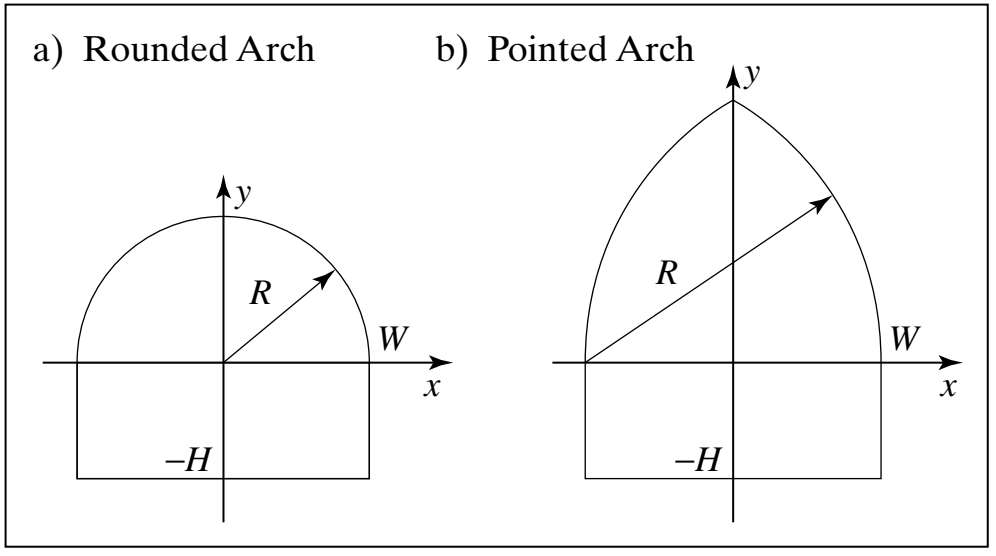
FIGURE 3.85 Chopping the segment that lies outside the window.

```
ChopLine(Point2 &P, unsigned char code)
{
      if(code & 8){      // to the Left
            P.y += (window.l - P.x) * dely / delx);
            P.x = window.l;
      }
      else if(code & 2){      // to the Right
            P.y += (window.r - P.x) * dely / delx;
            P.x = window.r;
      }
      else if(code & 1){        // below
            P.x += (window.b - P.y) * delx / dely;
            P.y = window.b;
      }
      else if(code & 4){      // above
            P.x += (window.t - P.y) * delx / dely;
            P.y = window.t;
      }
}
```

FIGURE 3.86 Interface for the Canvas class in Turbo C++.

```
class Canvas {
 public:
   Canvas(int width, int height);  // constructor
   setWindow(),setViewport(), lineTo(), etc .. as before
 private:
   Point2 CP;         // current position in the world
   IntRect viewport; // the current window
   RealRect window;  // the current viewport
   float mapA, mapB, mapC, mapD;  // data for the window-to-viewport mapping
   void makeMap(void); // builds the map
   int screenWidth, screenHeight;
   float delx,dely;       // increments for clipper
   char code1, code2;     // outside codes for clipper
   void ChopLine(tPoint2 &p, char c);
   int clipSegment(tPoint2 &p1, tPoint2 &p2);
};
```

FIGURE 3.87 Two basic arch forms.



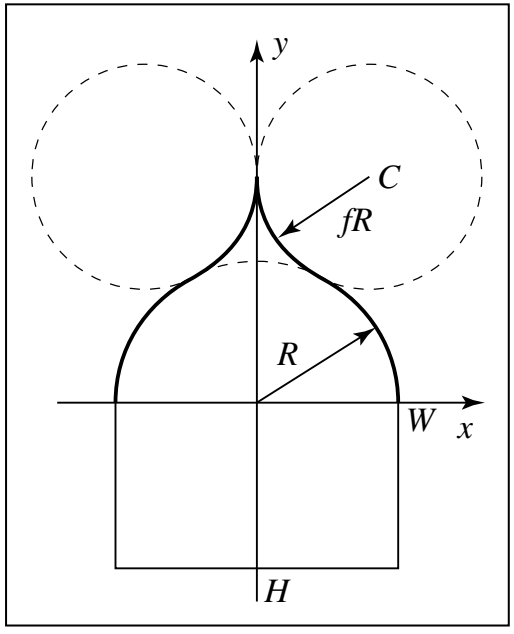
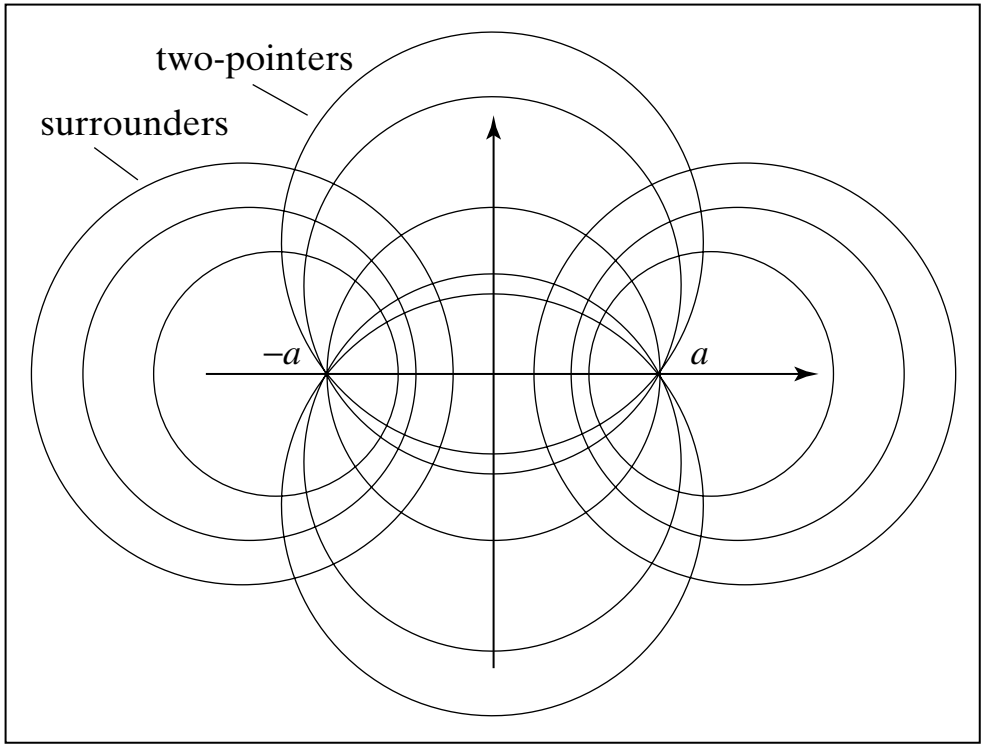


FIGURE 3.88 The ogee arch.

FIGURE 3.89 Families of orthogonal circles.



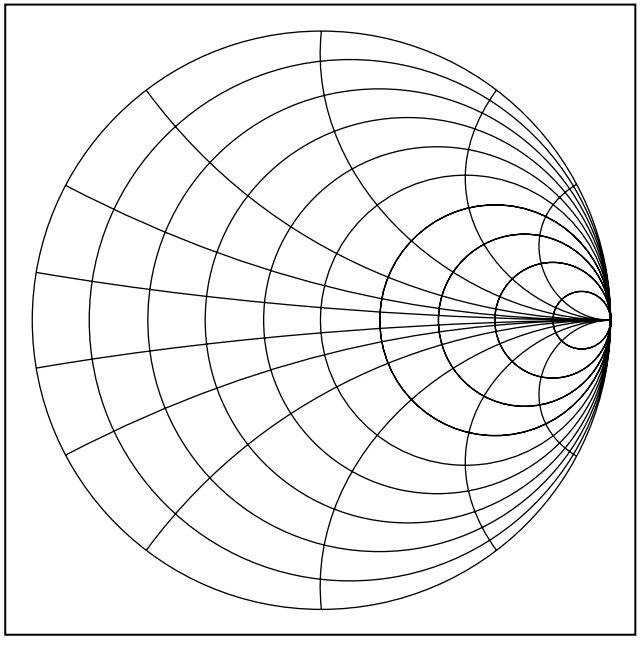


FIGURE 3.90 The Smith Chart.

FIGURE 3.91 Standard graphic symbols for the nand and nor gates.

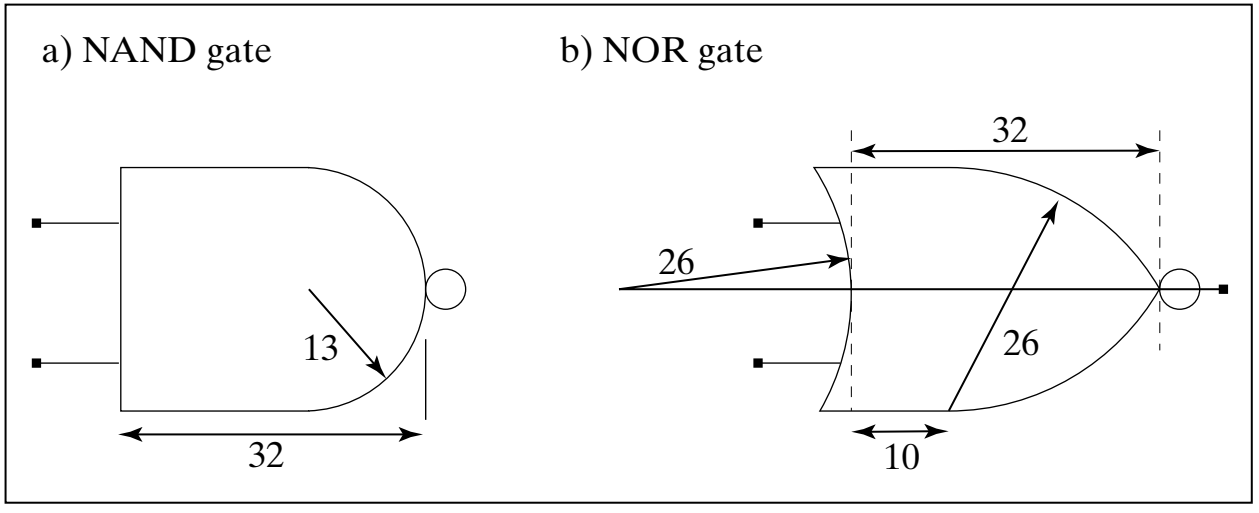
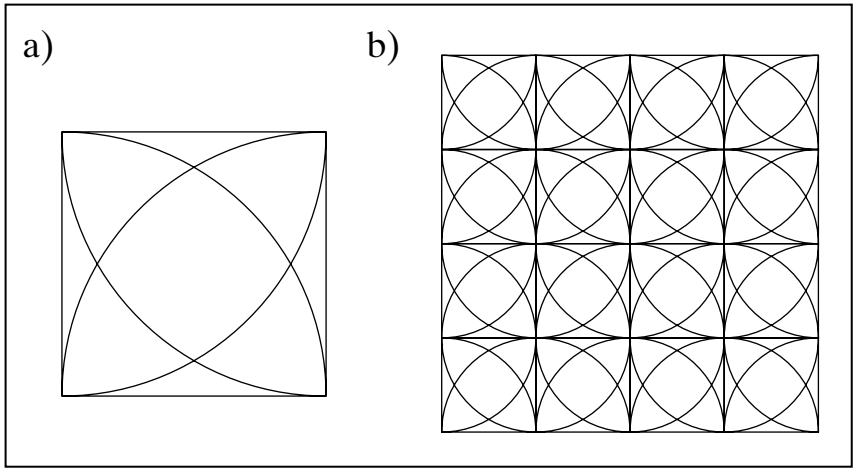


FIGURE 3.92 A motif and the resulting tiling.



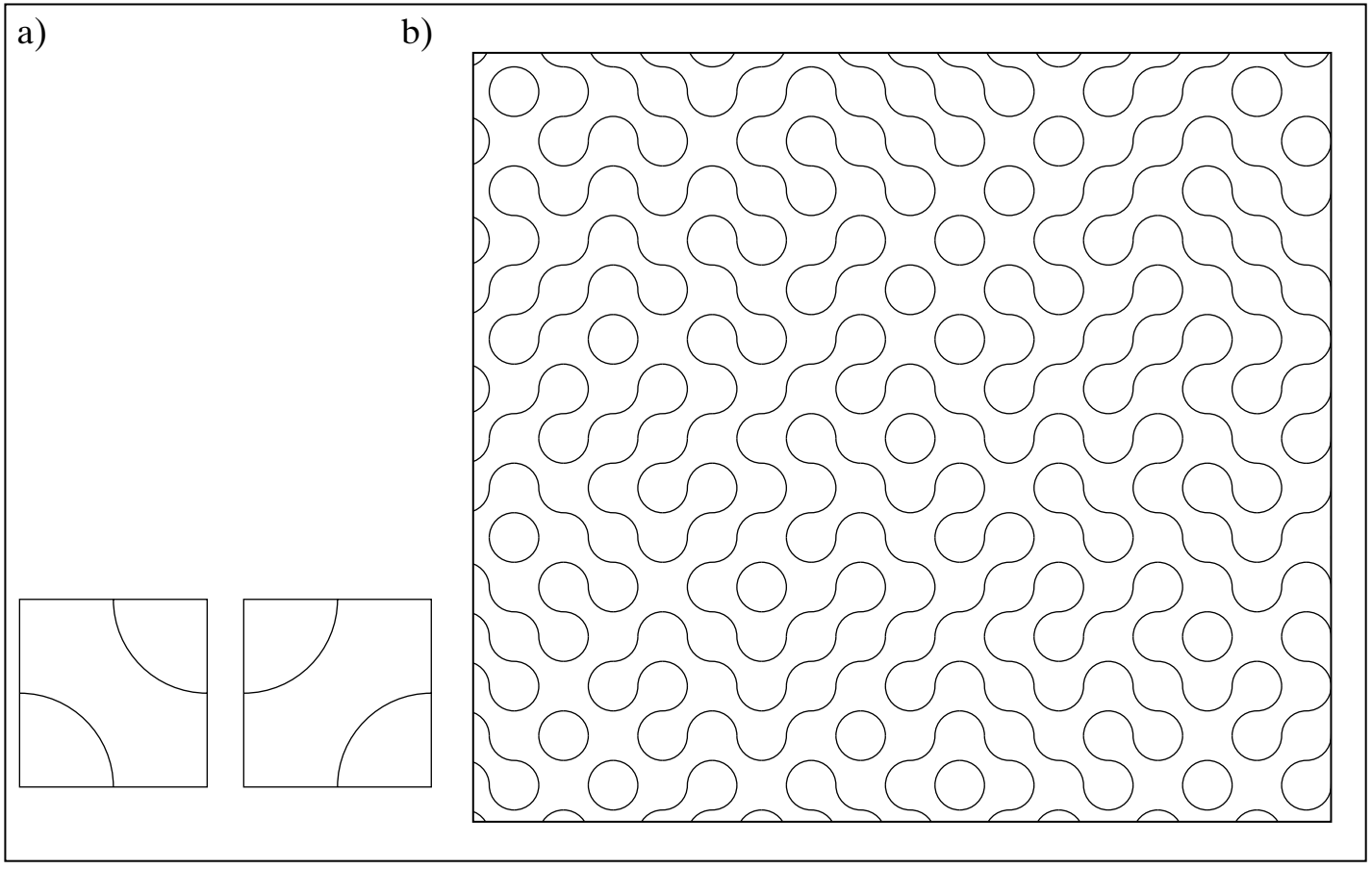


FIGURE 3.93 Truchet Tiles. (a) The two tiles. (b) A Truchet pattern.

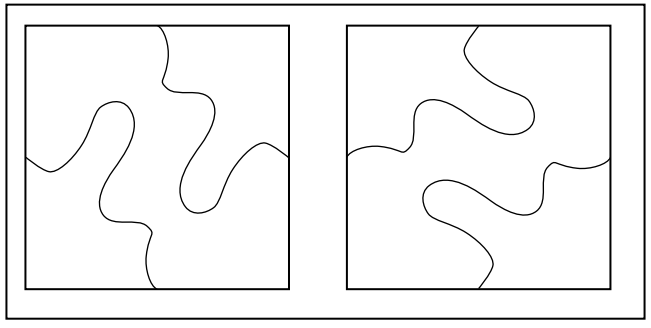
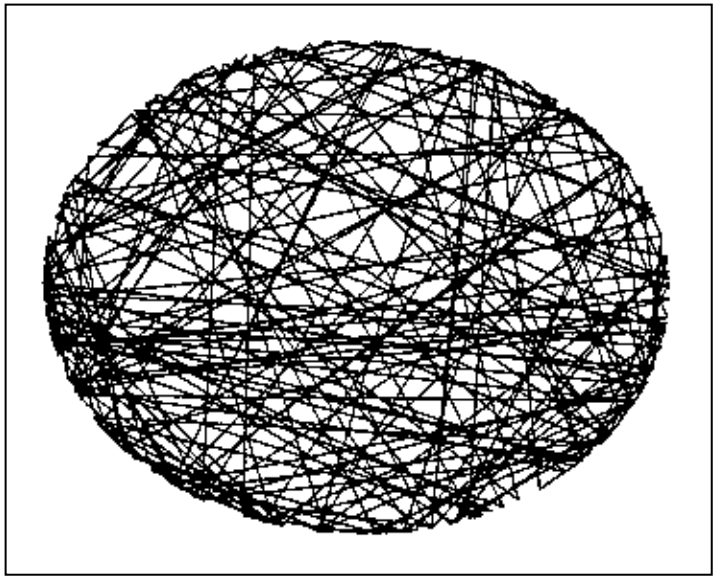


FIGURE 3.94 Extension of Truchet tiles.

FIGURE 3.95 A random ellipse polyline.



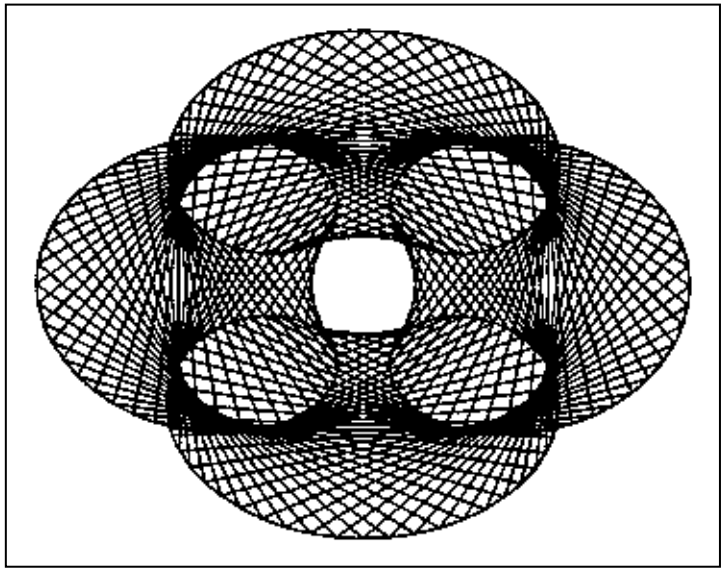


FIGURE 3.96 Adding "webs" to a curve.

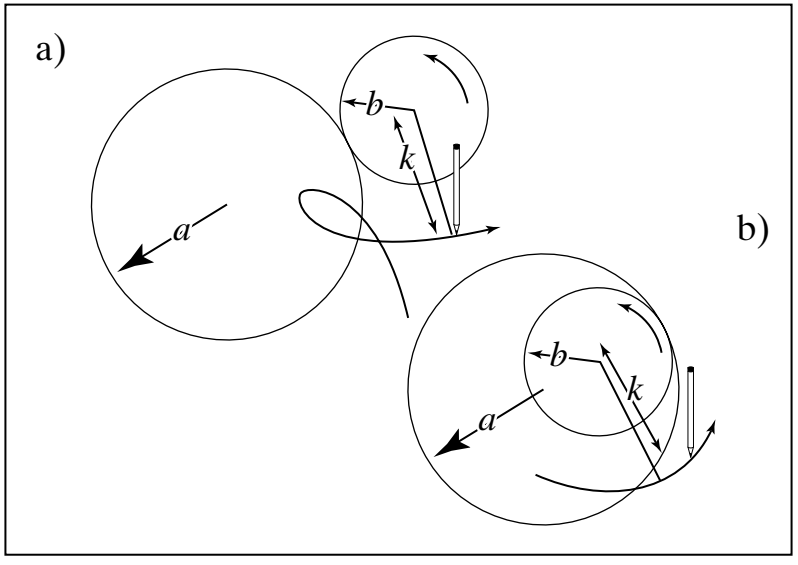


FIGURE 3.97 Circles rolling around circles.

FIGURE 3.98 Examples of cycloids. (a) Nephroid. (b) $a/b = 10$. (c) Deltoid. (d) Astroid.