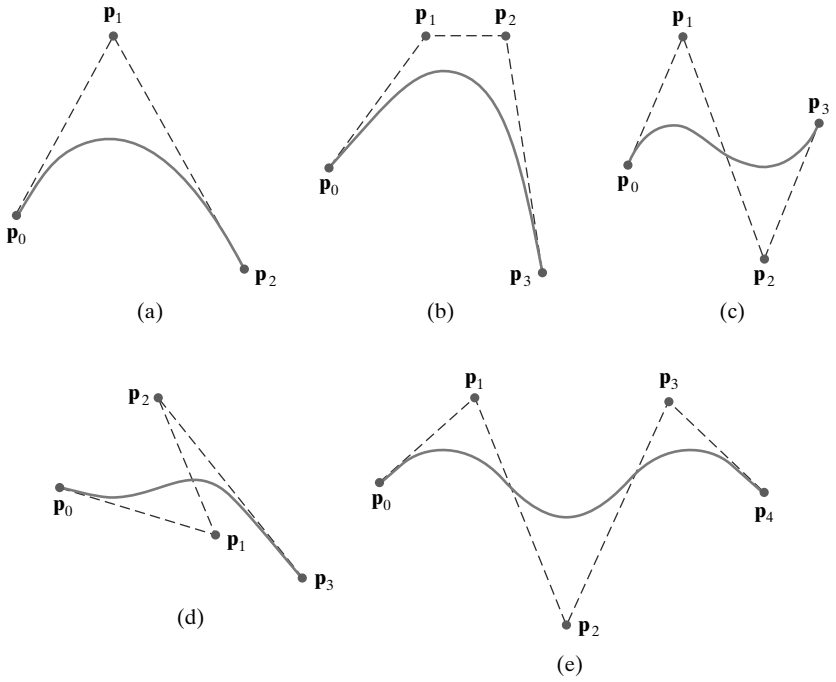


**FIGURE 20**
Examples of two-dimensional Bézier curves generated with three, four, and five control points. Dashed lines connect the control-point positions.

## Example Bézier Curve-Generating Program

An implementation for calculating the Bézier blending functions and generating a two-dimensional, cubic Bézier-spline curve is given in the following program. Four control points are defined in the *xy* plane, and 1000 pixel positions are plotted along the curve path using a pixel width of 4. Values for the binomial coefficients are calculated in procedure `binomialCoeffs`, and coordinate positions along the curve path are calculated in procedure `computeBezPt`. These values are passed to procedure `bezier`, and pixel positions are plotted using the OpenGL point-plotting routines. Alternatively, we could have approximated the curve path with straight-line sections, using fewer points. More efficient methods for generating coordinate positions along the path of a spline curve are explored in Section 15. For this example, the world-coordinate limits are set so that only the curve points are displayed within the viewport (Figure 21). If we also wanted to plot the control-point positions, the control graph, or the convex hull, we would need to extend the limits of the world-coordinate clipping window.

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

/*  Set initial size of the display window.  */
GLsizei winWidth = 600, winHeight = 600;

/*  Set size of world-coordinate clipping window.  */
GLfloat xwcMin = -50.0, xwcMax = 50.0;
GLfloat ywcMin = -50.0, ywcMax = 50.0;
```

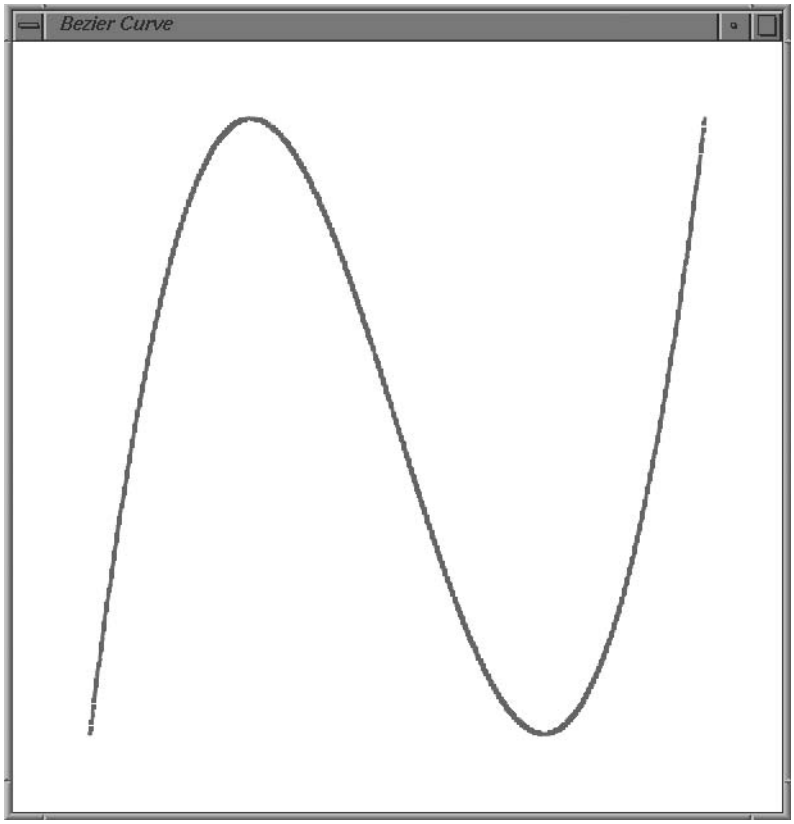


**FIGURE 21**
A Bézier curve displayed by the example program.

```
class wcPt3D {
   public:
      GLfloat x, y, z;
};

void init (void)
{
   /*  Set color of display window to white.  */
   glClearColor (1.0, 1.0, 1.0, 0.0);
}

void plotPoint (wcPt3D bezCurvePt)
{
    glBegin (GL_POINTS);
        glVertex2f (bezCurvePt.x, bezCurvePt.y);
    glEnd ( );
}

/*  Compute binomial coefficients C for given value of n.  */
void binomialCoeffs (GLint n, GLint * C)
{
   GLint k, j;

   for (k = 0;  k <= n;  k++) {
      /*  Compute n!/(k!(n - k)!).  */
```

```
      C [k] = 1;
      for (j = n;  j >= k + 1;  j--)
        C [k] *= j;
      for (j = n - k;  j >= 2;  j--)
        C [k] /= j;
   }
}

void computeBezPt (GLfloat u, wcPt3D * bezPt, GLint nCtrlPts,
                   wcPt3D * ctrlPts, GLint * C)
{
   GLint k, n = nCtrlPts - 1;
   GLfloat bezBlendFcn;

   bezPt->x = bezPt->y = bezPt->z = 0.0;

   /*  Compute blending functions and blend control points. */
   for (k = 0; k < nCtrlPts; k++) {
      bezBlendFcn = C [k] * pow (u, k) * pow (1 - u, n - k);
      bezPt->x += ctrlPts [k].x * bezBlendFcn;
      bezPt->y += ctrlPts [k].y * bezBlendFcn;
      bezPt->z += ctrlPts [k].z * bezBlendFcn;
   }
}

void bezier (wcPt3D * ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
   wcPt3D bezCurvePt;
   GLfloat u;
   GLint *C, k;

   /*  Allocate space for binomial coefficients  */
   C = new GLint [nCtrlPts];

   binomialCoeffs (nCtrlPts - 1, C);
   for (k = 0;  k <= nBezCurvePts;  k++) {
      u = GLfloat (k) / GLfloat (nBezCurvePts);
      computeBezPt (u, &bezCurvePt, nCtrlPts, ctrlPts, C);
      plotPoint (bezCurvePt);
   }
   delete [ ] C;
}

void displayFcn (void)
{
   /*  Set example number of control points and number of
    *  curve positions to be plotted along the Bezier curve.
    */
   GLint nCtrlPts = 4, nBezCurvePts = 1000;

   wcPt3D ctrlPts [4] = { {-40.0, -40.0, 0.0}, {-10.0, 200.0, 0.0},
                          {10.0, -200.0, 0.0}, {40.0, 40.0, 0.0} };

   glClear (GL_COLOR_BUFFER_BIT);   //  Clear display window.
```

```
   glPointSize (4);
   glColor3f (1.0, 0.0, 0.0);      //  Set point color to red.

   bezier (ctrlPts, nCtrlPts, nBezCurvePts);
   glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   /*  Maintain an aspect ratio of 1.0.  */
   glViewport (0, 0, newHeight, newHeight);

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );

   gluOrtho2D (xwcMin, xwcMax, ywcMin, ywcMax);

   glClear (GL_COLOR_BUFFER_BIT);
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (50, 50);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Bezier Curve");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);

   glutMainLoop ( );
}
```

## Properties of Bézier Curves

A very useful property of a Bézier curve is that the curve connects the first and last control points. Thus, a basic characteristic of any Bézier curve is that

$$\begin{aligned} \mathbf{P}(0) &= \mathbf{p}_0 \\ \mathbf{P}(1) &= \mathbf{p}_n \end{aligned} \tag{28}$$

Values for the parametric first derivatives of a Bézier curve at the endpoints can be calculated from control-point coordinates as

$$\begin{aligned} \mathbf{P}'(0) &= -n\mathbf{p}_0 + n\mathbf{p}_1 \\ \mathbf{P}'(1) &= -n\mathbf{p}_{n-1} + n\mathbf{p}_n \end{aligned} \tag{29}$$

From these expressions, we see that the slope at the beginning of the curve is along the line joining the first two control points, and the slope at the end of the curve is along the line joining the last two endpoints. Similarly, the parametric second derivatives of a Bézier curve at the endpoints are calculated as

$$\begin{aligned} \mathbf{P}''(0) &= n(n-1)[(\mathbf{p}_2 - \mathbf{p}_1) - (\mathbf{p}_1 - \mathbf{p}_0)] \\ \mathbf{P}''(1) &= n(n-1)[(\mathbf{p}_{n-2} - \mathbf{p}_{n-1}) - (\mathbf{p}_{n-1} - \mathbf{p}_n)] \end{aligned} \tag{30}$$