Real Time Generation Of Smooth Curves Using Local Cubic Segments

Martin Roche and Wanxing Li

Robot Systems Divison
National Bureau Of Standards
Gaithersburg, MD 20899

1. *Introduction*

In applying parametric cubic curve interpolation procedures to engineering and scientific data we consider the condition that the only restriction is that the curve be smooth. By a smooth curve we mean at a given input defining point P_k, the tangent vector for the cubic segment defined between the points P_{k-1} and P_k has the same direction as the tangent vector for the cubic segment defined between the points P_k and P_{k+1} when each is evaluated at P_k. We present procedures that are suitable for the case:

(a) given that we have an interpolation curve for the string $P_1,...,P_{i-1}$, we wish to define a cubic segment between the points P_{i-1} and P_i which, when added to the previously defined interpolation curve will be a smooth curve that interpolates $P_1,...,P_{i-1}$, and P_i, while the point P_{i+1} is being computed.

The classical problem of interpolation is:

(b) the usual problem of interpolating a given string $P_1,...,P_n$ of points where all the points are known at the time the interpolation is being determined.

With this classical problem spline functions are quite often used for the interpolation. This method has been discounted for case (a) above because of the difficulty implementing a spline function in this case. Since all points are not available during real time processing, the use of the classical spline algorithms would be very time consuming for this application. That is the number of functional operations (multiplications and additions, etc.) rules out consideration of this technique. Because of this we have considered local construction procedures in place of the standard spline type of construction.

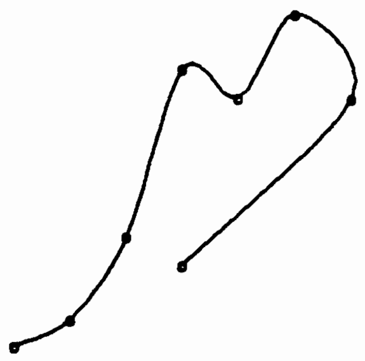

Figure 1 *A curve generated using the parabolic blending procedure.*

The parabolic blending (or Overhauser) procedure [1,2] which limits the required information needed to generate a cubic segment is attractive for both (a) and (b) listed above (Fig. 1). The procedure will be seen to limit the arithmetical operations and to minimize the computational speed. This procedure has, in the past, been used in generating curves for CAD applications. For these applications the defining points are known before one generates the curve which is case (b). But this curve type is such that it can lend itself to our application of generation of a curve as the point information is supplied which is case (a).

Another attractive procedure is the three points and a vector cubic segment definition method. For the three points the vector will be associated with the second point and a cubic segment would be defined between the second and third given points (Fig. 2). Coefficients for this case can be determined with a minimum number of operations. Other variations of this procedure could also be implemented easily.

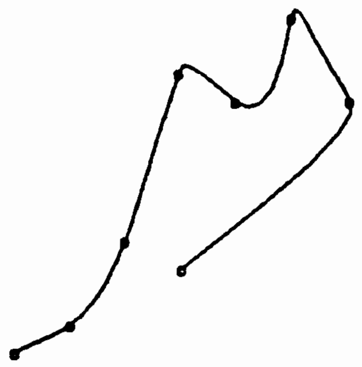

Figure 2 *Acurve generated using three points and a vector procedure. The vector defines the tangent value at the second point.*

It should be noted that in the construction of the curves of Figures 1, and 2, the same input coordinate values were used for the two figures.

2. Defining Cubic Curve Segments by Parabolic Blending

As the name suggests, parabolic blending involves the concept of using blending functions [3,4] to blend two parametrically defined second degree polynomials into a cubic polynomial over an interval. In particular, for four given points p_1 ,p_2, p_3 and p_4, a second degree polynomial, p , will be defined by the points p_1, p_2 and p_3 and another-second degree polynomial, q , will be defined by the points p_2, p_3, and p_4. We wish to select blending functions B1 and B2 such that: (i) B1 = B1(t) and B2 = B2(t) are functions of the same parameter; (ii) the resultant vector function C(t) = B1 p + B2 q has vector equal to p_2 for t= 0 and vector value equal to p_3 for t= 1 (Fig. 3).

Overhauser [1] selected B1= (1 - t) and B2 = t. Since p and q are second degree vector polynomials, we can write

$$p = p(r) = (r^2, r, 1)B \tag{1a}$$

$$q = q(s) = (s^2, s, 1)D \tag{1b}$$

where both B and D are 3x3 matrices. Writing the parameters r and s as linear functions of t (i.e. r = a t + b and s = c t + d), the resulting vector curve C is a parametric cubic:

$$C(t) = (t^3, t^2, t, 1)\ A ,$$

where A is a 4 x 4 matrix.

If we select the r values and the s values such that

$$p(0) = p_1 ,\ p(1/2) = p_2 \text{ and } p(1) = p_3$$
$$q(0) = p_2 ,\ q(1/2) = p_3 \text{ and } q(1) = p_4$$

we get from equations (1a) and (1b) that r and s relate to t by the equations

$$r = 0.5(t+1) \quad \text{and} \quad s = 0.5t \qquad (2).$$

Using equations (1) and (2), and some manipulations we arrive at

$$C(t) = [t^3\ t^2\ t\ 1]\ M \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \qquad (3a),$$

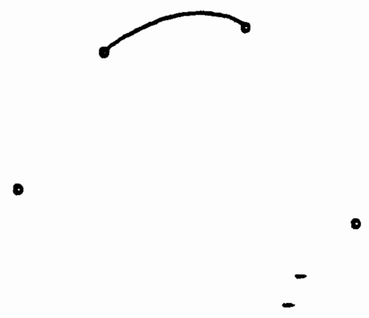

Figure 3 *Four points defining a parabolic blend between the second and third points are being displayed. Also displayed is the cubic segment defined by these points.*

where

$$M = \begin{bmatrix} -1/2 & 3/2 & -3/2 & 1/2 \\ 1 & -5/2 & 2 & -1/2 \\ -1/2 & 0 & 1/2 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

or

$$C(t) = a\, t^3 + b\, t^2 + c\, t + d \qquad (3b)$$

where

$$a = -1/2\, p_1 + 3/2\, p_2 - 3/2\, p_3 + 1/2\, p_4$$
$$b = p_1 + -5/2\, p_2 + 2\, p_3 + -1/2\, p_4$$
$$c = -1/2\, p_1 + 1/2\, p_3$$
$$d = p_2$$

(where details of the manipulations can be found in [1], or[2]). Note: One must keep in mind that each p_i is a vector, say $p_i = (x_i, y_i, z_i, w_i)$, so that the coefficients determined for the cubics are vectors (we are working with parametric equations).

In generating a complete curve, each curve segment is defined by the use of equation (3a). Special consideration is given to the first and the last segment. If p_1, p_2, and p_3 are the first three points generated, then to generate the cubic between p_1 and p_2, set $p_0 = p_1$ and use equation(3a).If p_{n-1} is the last point to be generated, then set $p_n = p_{n-1}$ and apply equation (3a) to the points p_{n-3}, p_{n-2}, p_{n-1} and p_n. It is worth noting that for a sequence of distinct points (p_i not equal to p_{i+1} for any i) that first derivative continuity at the boundary of two adjacent cubic curve segments is maintained, i.e, a smooth curve is generated (see [2] for details).

3. Three Points and A Vector Procedure

We are given given three points P_{-1}, P_0, and P_1, and a vector V and a cubic segment is to be defined between points P_0 and P_1 (Fig. 4)[5]. Let C(t) be the cubic polynomial such that

$$C'(0) = V$$

and assume a parameterization such that

$$C(-1) = P_{-1};$$
$$C(0) = P_0;$$
$$C(1) = P_1.$$

If

$$C(t) = at^3 + bt^2 + ct + d$$

is the cubic, then

$$\begin{cases} d = P_0; \\ c = V; \\ C(-1) = -a + b + -c + d = P_{-1}; \\ C(1) = a + b + c + d = P_1. \end{cases} \qquad (4)$$

Letting $q_1 = P_{-1} + (c - d)$ and $q_2 = P_1 - (c + d)$ we obtain

$$a = (-q_1 + q_2)/2 \qquad (5)$$
$$b = (q_1 + q_2)/2 \qquad (6).$$

The coefficients a, b, c, d are seen to be easily determined by employing system (4) and equations (5) and (6).

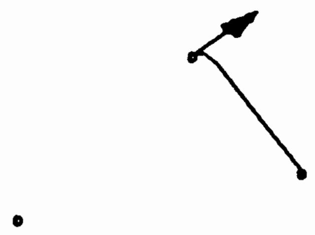

Figure 4 *The three points and a vector defining a cubic curve segment between the second and third points as outlined in Section 5 are being displayed. Also displayed is the cubic segment defined by the points and the vector.*

To define a complete curve using these cubic curve segments the following steps are required:

a. For the first cubic segment the following three steps are required:
(i) The first three points p_0, p_1 and p_2 of a sequence are given;
(ii) We assume at point p_0 the cubic vector segment $C_0(t)$ to be defined has all zero second derivative components. If

$$C_0(t) = at^3 + bt^2 + ct + d$$

is the cubic, then by our assumption b = 0 (the zero vector), and we also have $C_0(0) = p_1 = d$. Then by solving the system

$$C_0(1) = a + c + d = a + c + p_1 = p_1$$

$$C_0(2) = 8a+2c+d = 8a+2c+p_1 = p_2$$

we obtain the coefficient vectors a and c. Hence the cubic between the points p_0 and p_1 is determined.

(iii) Using the cubic coefficients of (ii), points between p_0 and p_1 are computed.

b. Given the point p_i and the fact that a cubic segment has been defined between the points p_{i-2} and p_{i-1}, the following steps are taken to determine the cubic for the segment between p_{i-1} and p_i:

(i) The vector V is determined by the cubic that was defined between p_{i-2} and p_{i-1}. In particular V is set equal to the tangent vector of that cubic evaluated at the point p_{i-1} ;

(iii) The three points p_{i-2},p_{i-1} and p_i and the vector V determine the cubic vector coefficiens as outlined above;

(iv) Points between p_{i-1} and p_i are computed.

A pocedure simular to the one just described could be implemented where C(t) is a cubic with $C'(-1) = V$ being given instead of $C'(0) = V$. Then equation (4) would become

$$\begin{cases} d = P_0 ; \\ V = 3a - 2b + c + ; \\ C(-1) = -a + b + -c + d = P_{-1} ; \\ C(1) = a + b + c + d = P_1 . \end{cases}$$

and the computation to determine the coefficients will be similar to the above [5]. We use the cubic segment in each case in the interval (-1, 0). In this case the construction will lag by one point.

4. Vector Considerations For The Three Points And Vector Construction

Because we are constructing parametric cubic segments using a tangent vector as part of its definition during the construction technique, caution must be used [4]. If the vector magnitude is greater than the distance between which the curve segment is being defined, the cubic could show a whip like effect[Fig. 5, 6, and 7].

In our implementation we resolved this dilemma by allowing for the scaling of each vector component by S / TV, where: S = a scale factor;
TV = the absolute value of the component of the vector that is largest in magnitude.
A more sophisticated procedure could be implemented by using the factor (S * DI)/TV, where S and TV are as above and
DI = the absolute value of the component that is the largest in magnitude of the vector which is the difference of the two points between which the cubic vector is to be defined.
By using this last procedure and restricting S to values between 0 and 1, the vector magnitude will always be less than the magnitude of the distance between which the curve segment is being defined. (DI could also have been selected as the distance between the two points.)

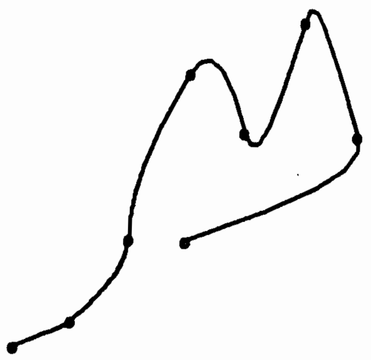

Figure 5 *An example using the procedure of section 3 and multiplying the components of the defining vector V by S/TV where S is equal to 15 is displayed.*

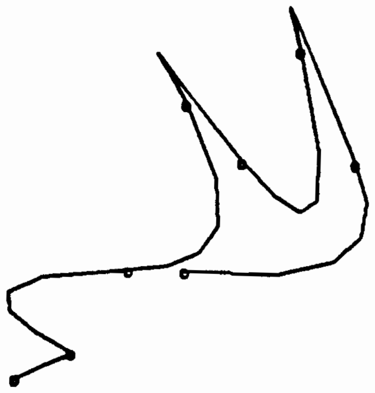

Figure 6 *An example using the procedure of section 3 and multiplying the components of the defining vector V by S/TV where S is equal to 35 is displayed. (The same input points as in figure 5 are used.)*

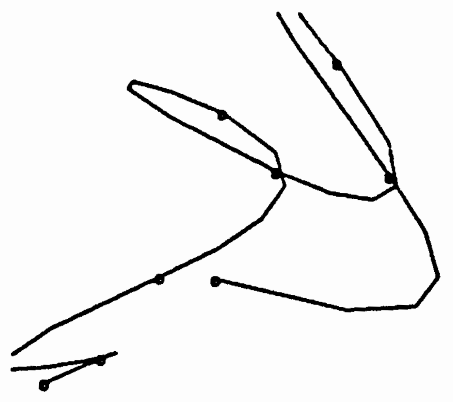

Figure 7 *An example using the procedure of Section 3 and multiplying the components of the defining vector V by S/TV where S is equal to 50 is displayed. (The same input points as in figure 5 are used.)*

5. Comparing The Required Number of Operations to Compute A Cubic Curve Segment

A criterion for suitability of a curve type is the number of computations required to compute points along a given curve segment. To determine the points, the cubic vector coefficients are computed to determine the expression

$$C(t) = a\,t^3 + b\,t^2 + c\,t + d \;;$$

then this expression is used to compute coordinate values along the curve. In each of the procedures discussed, the difference in computation time will be related to the determination of the coefficients. With this in mind one needs only to consider the operations involved in defining the vector coefficients to make a comparison of the different procedures. In comparing the number of operations performed to define vector coefficients for the above three cases, one needs only to consider one vector component. That is, one would solely need to determine the number of operations of the first vector component of each vector coefficient. With this in mind:

(i) One observes that for each cubic component for the Overhauser parabolic blending case there are nine multiplications and seven additions required to compute the coefficients.

(ii) For the three points and a vector case discussed in section 3, the coefficients for one component of a cubic segment are determined by four multiplications and divisions; the sum of the additions and subtractions will be eight (these numbers include the computations to compute the vector V). The multiplications of the scaling mentioned in section 4 could also be included bringing the number of multiplications to five or six depending upon how you wished to include the S term being divided by the TV term.

If speed is the sole criterion for selection, then one of the three points and a vector procedure or something similar to them would be selected. One will note that when using the three points and a vector procedure outlined above, at times a natural looking curve is not generated because of the lack of control of the slope vector at the point P_1 of section 3. With the other mentioned procedure (Parabolic Blending) the last point supplied is used as part of the procedure to define the slope vector value for the preceeding point. The curve generated by the Overhauser parabolic blending case has more aesthetic appeal, but has a slightly greater computational time. The proper choice of a curve type will depend on one's application.

References :

1. Overhauser, A., "Analytic Definition of Curves and Surfaces by Parabolic Blending," Technical Report No. SL68-40, Ford Motor Company Scientific Laboratry, May 8, 1968 .
2. Brewer, J.A.,"Three Dimensional Design by Graphical Man-Computer Communication," Purdue University, PhD Thesis, May, 1977.
3. Mortenson, M. E., *Geometric Modeling*, John Wiley and Sons, Inc., Somerset, New Jersey, 1985.
4. Forrest, A.R., "Curve and Surfaces for Computer Aided Design," University of Cambridge, PhD Thesis, July 1968.
5. Roche, M., and Li, W., "A Note on Real Time Parametric Cubic Segment Curve Generation," Intelligent Instruments & Computers, Vol. 5, No. 7, July 1987.

```
#define         VOID    int

/*   ********************************************   */
/*   ********************************************   */
/*           Parabolic Blending Procedure           */
/*   ********************************************   */
/*   Given a sequence of four points, this routine
     will determine the coefficients for a cubic
     curve segment by parabolic blending between
     the first and second points, the second and
     third points, or the third and fourth points
     [ See section 2].                               */

VOID Plot_Seg(q1, q2, q3, q4, sn, np, nd, nseg, q)
float   q1[], q2[], q3[], q4[] ;
                /* These are the defining
                 * points for the Overhauser curve segment.
                 */
int     sn ;    /* The segment to be plotted.
                 * The first , second , or last.
                 */
int     np ;    /* Number of segment points to be plotted.
                 */
int     nd ;    /* The dimension of the coordinate vectors.
                 */
int     nseg ;  /* The segment number for the segment to be generated.
                 */
float   q[10][3] ;
                /* the computed points along the
                 * cubic segment from points  p2 to p3.
                 */

{
 float  p1[4], p2[4], p3[4], p4[4] ;
 int    sp =2 ; /* Should the first point of the segment be
                 * plotted ? If yes , sp =1, if no , sp =2.
                 */
 int    i , PTT ;
 for(i=0 ; i < nd  ; i++)
   {
        p1[i] = q1[i] ;
        p2[i] = q2[i] ;
        p3[i] = q3[i] ;
        p4[i] = q4[i] ;
   }
 if(sn == 1)
   {
        for(i = 0 ; i < nd ; i++)
          {
                p2[i] = q1[i] ;
                p3[i] = q2[i] ;
                p4[i] = q3[i] ;
                sp =1 ;
          }
   }
 if(sn ==3)
   {
        for(i = 0 ; i < nd ; i++)
          {
                p1[i] = q2[i] ;
                p2[i] = q3[i] ;
                p3[i] = q4[i] ;
          }
   }
 PTT = O_seg(p1, p2, p3, p4, np, nd, q) ;
 Plot_Pac(q, np, nd, sp, nseg) ;
} /* END  OF  PLOT_SEG */


O_seg(p1, p2, p3, p4, NP, ND, q )
float   p1[], p2[], p3[], p4[] ;
                /* These are the defining points for
                 * Overhauser Curve segment defined
                 * between points  p2,  and  p3.
                 * For  i =0, 1, and 2,
                 * pi[0] - is the x-coordinate
                 * pi[1] - is the y-coordinate
                 * pi[2] - is the z-coordinate.
                 */
int     NP ;    /* This gives the number of points of the curve segment
                 * that will be computed . The computed points will be
                 * stored in the array q[][]. The first point will be
                 * (q[0][0], q[0][1], q[0][2] )=(p2[0], p2[1], p2[2] )
                 * and the last point will be (q[NP-1][0], q[NP-1][1],
                 * q[NP-1][2] ) =(p3[0], p3[1], p3[2] ).
                 */
int     ND ;    /* This gives the dimension of the system . ND can
                 * equal  1, 2, or 3.
                 */
float   q[10][3] ;
                /* The output coordinate array.
                 */
{
 int    i, j,  l ;
 float  t , qq  ;
                /* Parameter value for the curve segment at a given point.
                 */
 float  a[4][3] ;

/***********************************************************************
 *  Set
 *                 |-1/2  3/2  -3/2  1/2|
 *                 | 1   -5/2    2  -1/2|
 *          N  =   |-1/2   0     1/2   0|
 *                 | 0     1      0    0|  ,
 *                 |p1|
 *                 |p2|
 *          P  =   |p3|
 *                 |p4|                    ,
 *
 *  and            |a[0][]|
 *                 |a[1][]|
 *         A  =    |a[2][]|  =  N * P      ,
 *                 |a[3][]|
 *  then  the  n-th point is computed by the expression
 *
 *
 *  (q[n][0], q[n][1], q[n][2]) = (t**3, t**2, t, 1) *  A  .
 *
 *  Load the  A   matrix  .
 *
 *********************************************************************/
 for(i = 0 ; i < ND ; i++)
   {
        a[0][i] = - 0.5*p1[i]  + 1.5*p2[i] - 1.5*p3[i] + 0.5*p4[i] ;
        a[1][i] =       p1[i]  - 2.5*p2[i] + 2.0*p3[i] - 0.5*p4[i] ;
        a[2][i] = - 0.5*p1[i]              + 0.5*p3[i]             ;
        a[3][i] =                   p2[i]                          ;
   }
                /* Now compute the points along the cubic segment.
                 * The first and last are first loaded.
                 */
 for(j = 0  ;  j  <  ND  ;  j++)
   {
        q[0][j]  =  p2[j] ;
        q[NP-1][j] = p3[j] ;
   }
                /* The intermediate coordinates are loaded.
                 */
 for(i = 1 ;  i< (NP -1)  ; i++)
   {
        t = (float)i/ (NP -1) ;
        for(j = 0 ; j < ND ; j++)
          {
                qq = a[0][j] ;
                for(l = 0 ; l < 3 ; l++) qq = a[l+1][j]  +  t*qq ;
                q[i][j]  =  qq ;
          }
   }
 return(0) ;
} /* END  OF  O-seg */


/*                                                    */
/*    *********************************************   */
/*    *********************************************   */
/*    *********************************************   */
/*    *********************************************   */
#define VOID    int
#define abs(A)           ( (A) >= 0.0 ? (A) : (-A) )
#define max(A,B)         ( (A) >= (B) ? (A) : ( B) )
/*    *********************************************   */
/*    *********************************************   */
/*       Three Points And A Vector Procedure          */
/*    *********************************************   */
/*    Given a sequence of three points and a vector
      ( this vector will be computed internally )
      this routine will determine the coefficients
      for a cubic curve segment following the procedure
      of section 3.                                   */


VOID Plot_Seg(p0, p1, p2,vsf, np, nd, nseg, coef)
float   p0[], p1[], p2[] ;
                        /* These are the three points used to define
```

```
                             the cubic segment. The vector V is
                             internally computed.                          */
int      np  ;            /* Number of segment points to be plotted.  */
int      nd  ;            /* The dimension of the coordinate vectors. */
int      nseg  ;          /* The segment number for the segment
                             to be generated                          */
float    coef[4][4] ;     /* The coefficients for the cubic segment.  */
float    vsf ;            /* the vector scale factor                  */



{
 float   V[4] ;
 int     sp = 2 ;         /* Should the first point of the segment be
                             plotted ? If yes , sp =1, if no , sp =2. */
 int     i , iflag ;
 float   q[10][3]  ;      /* the computed points along the
                             cubic segment from points  p1 to p2
                             or from p2 to p3.                        */
 float   pm , ptt , TV ;
 float   tf = 0, tl  = 1 ;

/*
         The vector  V is internally computed.
                                                                      */
 if( nseg  ==1)
   {
/*       If nseg = 1 then this is the first segment of the
         curve to be generated. In this case the vector V
         is computed at t= 0.0 by assuming that the second
         derivative is zero at this point. Since the cubic
         passes through the points p0, p1 and p2
         V = 4/3*(p1 - p0) - 1/6*(p2 - p0)                            */

         TV = 0.0  ;
         i = 0  ;
         while(i < nd)
           {
                 V[i] = 1.33333*(p1[i] - p0[i])
                         + 0.1666666*(p1[i]  - p0[i]) ;
                 TV = max( abs( V[i] ), TV)  ;
                 i++ ;
           }
         TV = vsf/TV   ;
         i = 0 ;
         while(i < nd)
         while(i < nd)
           {
                 V[i] = V[i]*TV  ;
                 i++ ;
           }

/*       Call  C_coeff  to compute the coefficients.                  */
         C_coeff(p0, p1, p2, V, coef, nd) ;
   }
   else
/*       V is determined by the coefficients of the last
         cubic segment . That is if
         C(t) = coef[3][] +coef[2][]*t +coef[1][]*t**2
```

```
                + coef[0][]*t**3
        is the last cubic generated, then
        C'(t) = coef[2][] +2*coef[1][]*t +3*coef[0][]*t**2
        determines the vector V.                                          */
   {
        TV = 0.0  ;
        i = 0 ;
        while(i < nd)
          {
                V[i] = coef[2][i] +2*coef[1][i] +3*coef[0][i] ;
                TV = max( abs( V[i] ), TV)  ;
                i++ ;
          }
        TV = vsf/TV ;
        i = 0 ;
        while(i < nd)
          {
                V[i] = V[i]*TV  ;
                i++ ;
          }
/*      Call  CC_coeff  to compute the coefficients.                      */
        CC_coeff(p0, p1, p2, V, coef, nd)  ; }

/*      Call  Comp_pts to compute points along the cubic segment.         */
        Comp_pts(coef, np, nd , tf, tl ,q) ;

        Plot_Pac(q, np, nd, sp, nseg) ;
}       /* END  OF  PLOT_SEG */

VOID    C_coeff(p0, p1, p2, V, coef, nd)
float   p0[], p1[], p2[], V[], coef[4][4] ;
int     nd ;
{
 float  q1[4], q2[4] ;
 int    i ;
 float  sumx, sumy  ;

 i  = 0  ;
 while(i < nd)
   {
        coef[3][i] =  p0[i]  ;
        coef[2][i] =  V[i]   ;
        q1[i] = p1[i] - (coef[3][i] +coef[2][i] ) ;
        q2[i] = p2[i] - (coef[3][i] +2*coef[2][i] ) ;
        coef[1][i] = 2* q1[i]  - 0.25*q2[i]  ;
        coef[0][i] =  -q1[i] + 0.25*q2[i] ;
        i++ ;
   }
}       /* END  OF  THE  C_coeff  ROUTINE */

VOID    CC_coeff(p0, p1, p2, V, coef, nd)
float   p0[], p1[], p2[], V[], coef[4][4] ;
int     nd ;
{
 float  q1[4], q2[4]  ;
 int    i ;
 float  sumx, sumy  ;
 i  = 0  ;
 while(i < nd)
```

```
   {
        coef[3][i] =  p1[i]   ;
        coef[2][i] =  V[i]    ;
        q1[i] = p0[i] - (coef[3][i] -coef[2][i] ) ;
        q2[i] = p2[i] - (coef[3][i] +coef[2][i] ) ;
        coef[1][i] =  (q1[i]   + q2[i]   )/2.   ;
        coef[0][i] =  (-q1[i] + q2[i])/2. ;
        i++ ;
   }
}       /* END  OF  THE  CC_coeff  ROUTINE */

Comp_pts(coef, np, nd, tf, tl, q)
float   coef[4][4], tf, tl ;
        /* These are the coefficients for
           the cubic curve      segment
           points  that are to be computed between
           tf,  and  tl.                                                  */
int     np ;
        /* This gives the number of points of the curve segment
           that will be computed . The computed points will be
           stored in the array q[][]. The first point will be
           (q(0][0], q[0][1], q[0][2] )=(p2[0], p2[1], p2[2] )
           and the last point will be (q[NP-1][0], q[NP-1][1],
           q[NP-1][2] ) =(p3[0], p3[1], p3[2] ).                         */
int     nd ;
        /* This gives the dimension of the system . nd can
           equal  1, 2, or 3.                                             */
float   q[10][3] ;
        /* The output coordinate array .                                  */
{
 int    i, j,  l ;
 float  t  , qq  ;
        /* Parameter value for the curve segment at a given point.       */
 float  dt ;

 dt   =  (tl -  tf)/( np - 1 ) ;
/*      Now copute the points along the cubic segment   .                 */
 for(i = 0 ; i< np ; i++)
   {
        t =  tf + ((float)i)*dt ;
        for(j= 0 ; j < nd  ; j++)
          {
                qq = coef[0][j] ;
                for(l = 0 ; l < 3 ; l++) qq = coef[l+1][j]  +  t*qq ;
                q[i][j]  =  qq  ;
          }
   }
 return(0) ;
}       /* END  OF  O-seg */
```