NPS-53-88-001

NAVAL POSTGRADUATE SCHOOL

Monterey, California

CALCULATING THE SELF-INTERSECTIONS OF BEZIER CURVES

Dieter Lasser

March 1988

Prepared for:

National Research Council
2101 Constitution Ave. N.W.
Washington, DC 20418

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

Rear Admiral R. C. Austin
Superintendent

K. T. Marshall
Acting Provost

The work reported herein was supported with funds from the National Research Council, 2101 Consitution Avenue, N.W., Washington, DC 20418.

This report was prepared by:

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS-53-88-001	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Calculating the Self-Intersections of Bezier Curves	5. TYPE OF REPORT & PERIOD COVERED: Technical Report 8 87 - 10 87
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Dieter Lasser	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School - Code 53, Monterey, CA 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: March 1988
	13. NUMBER OF PAGES: 36
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CAD, CAGD, Bezier Curves, Spline Curves, Intersection, Self-Intersection

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Abstract: A user-friendly *divide-and-conquer* algorithm is presented for finding all the self-intersection points of a parametric curve in the Bernstein-Bezier representation. The underlying idea of the algorithm is to deal with the Bezier polygon instead of the curve description itself. By alternately subdividing the Bezier polygon and estimating the self-intersection regions the self-intersection points are finally approximated by straight line intersections of the refined Bezier polygons. The algorithm also calculates the parameter values of the self-intersection points. In addition to the *convex hull* and the *approximation property* of the Bezier polygon the working of the algorithm is based on a very intuitive *angle criterion*.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S N 0102-LF-014-6601

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Calculating the Self-Intersections of Bezier Curves

Dieter Lasser
Fachbereich Mathematik, AG3
Technische Hochschule Darmstadt
6100 Darmstadt, West Germany

Present address: Naval Postgraduate School
Department of Mathematics
Monterey, CA 93943
USA

Abstract: A user-friendly *'divide-and-conquer'* algorithm is presented for finding all the self-intersection points of a parametric curve in the Bernstein-Bezier representation. The underlying idea of the algorithm is to deal with the Bezier polygon instead of the curve description itself. By alternately subdividing the Bezier polygon and estimating the self-intersection regions the self-intersection points are finally approximated by straight line intersections of the refined Bezier polygons. The algorithm also calculates the parameter values of the self-intersection points. In addition to the *convex hull* and the *approximation property* of the Bezier polygon the working of the algorithm is based on a very intuitive *angle criterion*.

0. Introduction

For two explicit given curves $f_1(x)$ and $f_2(x)$ intersection points of $f_1(x)$ and $f_2(x)$ can be calculated using numerical methods like Newton's method by rewritting the problem as that of finding the roots (zeros) of the function $F(x) = f_1(x) - f_2(x)$. If the equation of one curve is given in implicit resp. explicit form and the other in parametric form, we can substitute the parametric form into the implicit resp. explicit equation. The (usually non-linear) equation we obtain can be solved by Newton's method again. If both curves are given implicitly as (non-linear) functions $f_1(x, y)$ and $f_2(x, y)$ of x and y or as parameterized curves $x_1 = x_1(t)$, $y_1 = y_1(t)$ and $x_2 = x_2(\tau)$, $y_2 = y_2(\tau)$ we have to solve the two equations $f_1(x, y) = 0$ and $f_2(x, y) = 0$ resp. $x_1(t) - x_2(\tau) = 0$ and $y_1(t) - y_2(\tau) = 0$ simultaneously, what can also be done by Newton's method [*Faux,Pratt '83*]. A geometricaly based modification of the methods described by Faux and Pratt to calculate the intersection points of two parameterized curves was given by Hoschek in [*Hoschek '85*]. Hoschek's method works also for the problem of calculating the self-intersections of a curve. Self-intersections of a curve can appear for example as boundaries of loops of parallel curves, often called offset curves. [*Arnold '86*], [*Farouki '85*], [*Hoschek '85, '87*], [*Klass '83*], [*Lyche,Mφrken '87*], [*Tiller,Hansen '84*]. For the loop removal the self-intersection points have to be detected. For rational curves this can also be done by algebraic methods which have been introduced in the area of CAGD by Sederberg, Goldmann and Anderson. They described in [*Sederberg '84*], [*Sederberg et al. '84, '85*] and [*Goldmann '85*] a method of classical algebraic geometry for solving the curve-curve intersection problem for rational planar and non-planar curves and in [*Sederberg et al. '85*] a method for finding the double points and by this the self-intersection points of planar rational cubics (see also [*Salmon 1879*], [*Hilton '32*], [*Walker '50*]).

In CAGD the B-spline-Bezier representation of curves is very popular and therefore it is of importance to have (self-)intersection algorithms for this type of curve representation too, so that no conversion of the curve description [*Dannenberg,Nowacki '85*], [*Hoschek '87*] is necessary. Curve-curve intersection algorithms for B-spline-Bezier representations have been described by [*Lane et al. '80*], [*Cohen et al. '80*] and for quadratics by [*Yang et al. '86*]. Yang calculates the intersection points using an algebraic method while the algorithms of Lane and Cohen are subdivision algorithms taking into account the geometric relationship between the curve and its defining control polygon. Pure subdivision algorithms are very time-consuming and need a lot of storage space [*Griffiths '75*] but they can accelerated by using in addition an estimation of the intersection region which yields to the so called *'divide-and-conquer'* algorithms. For B-spline-Bezier representations the estimation of those parts that do not participate in the intersection can be done by using the convex hull property [*Lane et al. '80*], [*Peng '84*] (see also part I of this paper) or, rougher but more easily and quickly handled by min-max boxes (see part II of this paper). An estimation using min-max boxes can also be done for non-B-spline-Bezier representations [*Koparkar,Mudur '83*].
A disadvantage of the subdivision and even of the more advanced divide-and-conquer algorithms against the algebraic based intersection algorithms might be that they are more time-consuming because of the subdivision process [*Sederberg,Parry '86*]. But the great advantages of the divide-and-conquer algorithms are that

- they are very user-friendly - no worry about *'suitable'* starting points,
- they find independently - that means without any interactive disruption to the user - all intersection points within the specified tolerance,
- they can be formulated easily for arbitrary polynomial degree and for non-rational and rational representations, and
- they are numerically very stable because of the extraordinary numerical properties of the Bernstein polynomials [*Farouki,Rajan '87*], [*Sederberg,Parry '86*].

Because of these favorable properties of the intersection algorithms based on the B-spline-Bezier representation using divide-and-conquer methods and because of the reason named above we would like to have also a self-intersection algorithm of this kind. The existing curve-curve algorithms can not be used directly by doing the curve input twice, because in this case the divide-and-conquer method will fail in the sense that no elimination of curve parts that do not participate in the self-intersection will be possible. Furthermore the final calculation of the self-intersection points, done by intersecting straight line segments defined by the control polygon will also fail by doing the same polygon input twice.

The only self-intersection algorithm for B-spline-Bezier representations I know was given in [*Tiller,Hansen '84*]. They calculate self-intersections of (rational) B-spline curves in a two step procedure. First they find the intersections of the control polygon with itself and then they use an iterative method (e.g. Newton) to improve the approximate solution found in step one. They know that this method can fail, because a curve can have a loop even though its control polygon has no self-intersection, but by using control polygons which approximate their curves very closely, i.e. building up the curve by a *'large'* number of segments, they try to make sure to be *'on the safe side'*. Although Tiller and Hansen are working with B-spline techniques, their algorithm dosen't belong to the powerful class of the divide-and-conquer algorithms because their algorithm dosen't use the typical kind of strategy of the divide-and-conquer algorithms for the evaluation of the self-intersections.

The algorithm presented here is a user-friendly divide-and-conquer algorithm for finding all the self-intersection points, including their parameter values, of a parameterized non-rational or rational curve of arbitrary degree in Bezier representation. For the creating of the algorithm the geometric relationship between the curve and its defining control polygon was fully taken into account. By alternately subdividing the Bezier polygon and estimating the self-intersection regions the self-intersection points are finally approximated by straight line intersections of the refined Bezier polygon. In addition to the convex hull property and the approximation property of the Bezier polygon the algorithm is based on a very intuitive angle criterion which is together with the convex hull property used for estimating the self-intersection region of the curve.
Because a curve-curve intersection algorithm is an important part of the self-intersection algorithm of part III of the paper, and because the final calculation of the self-intersection points and its parameter values is done in the same way as in the curve-curve algorithm, a short explanation of a

divide-and-conquer algorithm for calculating the intersection points of two parameterized non-rational or rational curves of arbitrary degree in Bezier representation is given in part II. The curve-curve algorithm described there differs from the *'classical'* one introduced by Lane [*Lane et al. '80*] in some *'details'*, mainly in the concept of the *'control unit'* and in the final calculation of the intersection points and its parameter values.
Part IV finally includes a short description of how to calculate the self-intersections of a Bezier spline curve.
All algorithms are written for planar curves, but for the extension to spatial curves only *'a third equation for the z-coordinate'* has to be added everywhere where coordinates have to be evaluated. The paper starts with some introductory words on the Bezier representation of (planar) curves.

I. Bezier Curves

A (planar) **Bezier curve** is defined by

$$\mathbf{B}(u) = \sum_{k=0}^{m} \mathbf{b}_k B_k^m(u)$$

where $\mathbf{b}_k = (x_k, y_k) \in \mathbb{R}^2$, $u \in [0,1]$ and

$$B_k^m(u) = \binom{m}{k} u^k (1-u)^{m-k}$$

are the (ordinary) **Bernstein polynomials** of degree m in u. The coefficients $\mathbf{b}_k \in \mathbb{R}^2$ are called **Bezier points.** They form in their natural ordering given by their subscripts the vertices of the so called **Bezier polygon** (see Figure 1).

It is possible to build up complex Bezier spline curves from a number of Bezier curve segments. The conditions for C^r continuity of adjacent curve segments can be found in [*Boehm et al '84*].

The Bezier description of a curve is a very powerful tool because the expansion in terms of Bernstein polynomials yield to a geometric relationship between the curve and its defining Bezier points. For example:

- the Bezier polygon gives a rough impression of the Bezier curve (see Figure 1),
- the curve has its endpoints at $\mathbf{b}_0$ and $\mathbf{b}_m$ with tangent vectors defined by $\mathbf{b}_0$, $\mathbf{b}_1$ and by $\mathbf{b}_m$, $\mathbf{b}_{m-1}$ (see Figure 1),
- *convex hull property:* the Bezier curve lies completely within the convex hull of its Bezier polygon (see Figure 2),
- the curve point $\mathbf{B}(u_0)$, for any $u_0 \in [0,1]$ can be computed by repeated de Casteljau steps by the recursion formula

$$\mathbf{b}_\alpha^\beta(u_0) = (1-u_0)\,\mathbf{b}_\alpha^{\beta-1}(u_0) + u_0\,\mathbf{b}_{\alpha+1}^{\beta}(u_0)$$

  where $\mathbf{b}_\alpha^0 \equiv \mathbf{b}_\alpha$ and $\mathbf{B}(u_0) = \mathbf{b}_0^m$ (see Figure 3).

The point $u = u_0$ subdivides a Bezier curve into two C^m continuous segments. Each segment is again a Bezier curve of the same degree as the original one. The Bezier points of these two segments are *'byproducts'* of the de Casteljau construction for the evaluation of the point $\mathbf{B}(u_0)$. They are given by $\mathbf{b}_0^k$ and $\mathbf{b}_k^m$ $(k = 0, \ldots, m)$. The subdivision process may be repeated yielding a sequence of polygons. For this sequence of polygons we have the important

- *approximation property:* if the u_0 are dense in $[0,1]$ the sequence of polygons converges to the curve.

Figure 4 illustrates how the curve can be fixed using the approximation and the convex hull property.

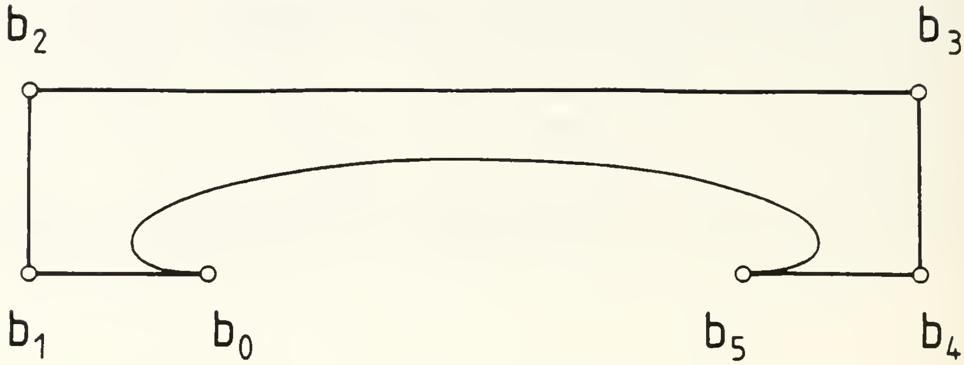


Figure 1. planar Bezier curve of degree five

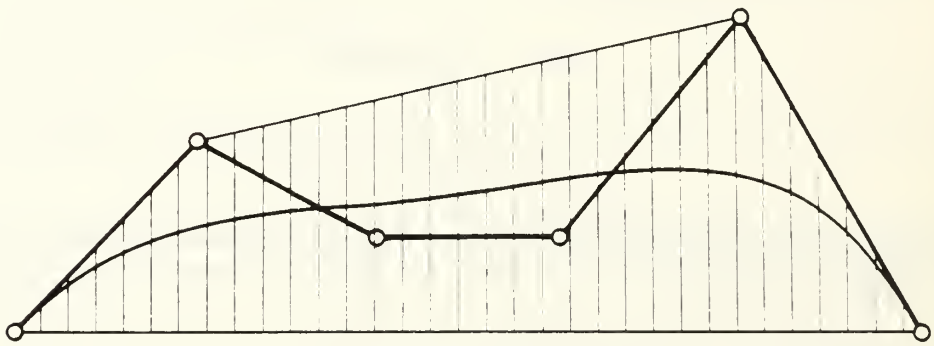


Figure 2. convex hull property

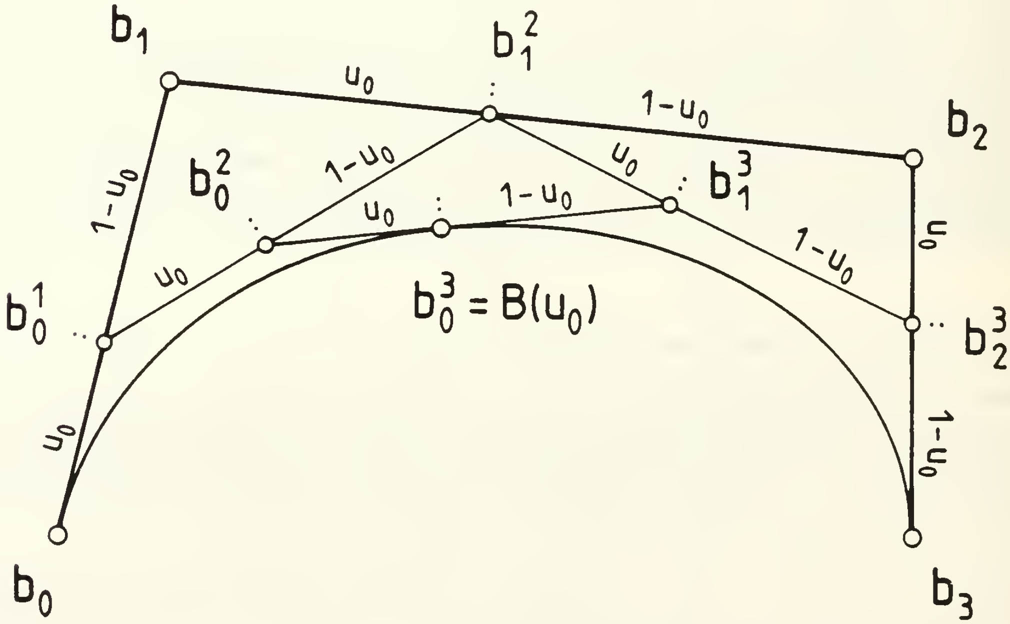


Figure 3. de Casteljau construction

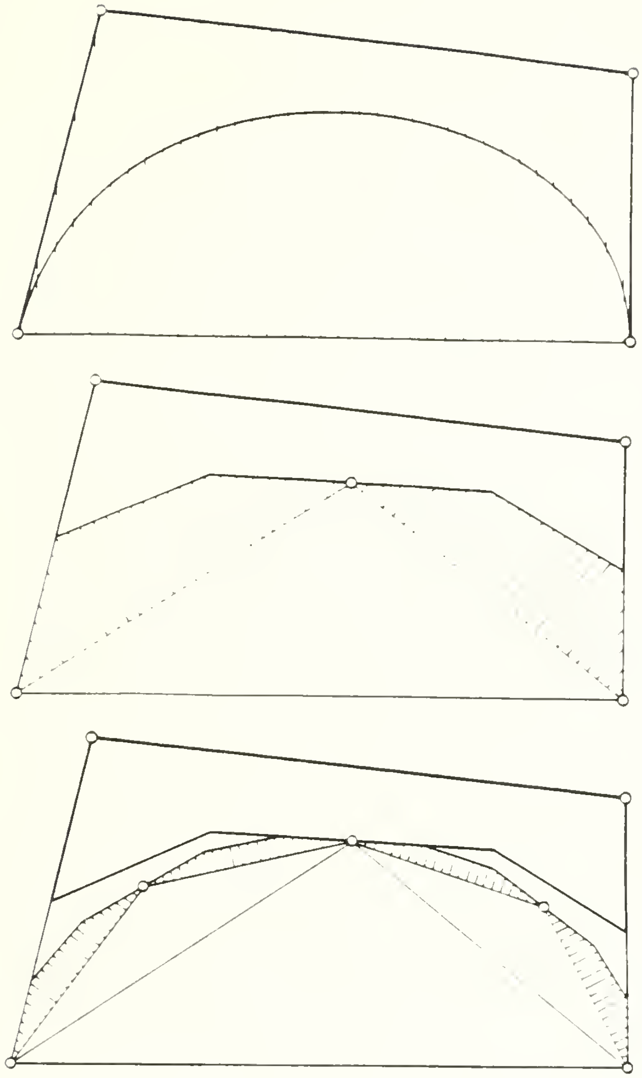


Figure 4. fixing the curve by the approximation and the convex hull property

A **rational** (planar) **Bezier curve** can be defined by

$$\mathbf{R}(u) = \sum_{k=0}^{m} \mathbf{b}_k R_k^m(u)$$

where $\mathbf{b}_k = (x_k, y_k) \in \mathbb{R}^2$, $u \in [0,1]$ and

$$R_k^m(u) = \frac{\beta_k B_k^m(u)}{\sum_{j=0}^{m} \beta_j B_j^m(u)}$$

are the **rational Bernstein polynomial** of degree m in u with weights $\beta_k \in \mathbb{R}$ [*Piegl '86*].
Figure 5 compares the (ordinary) Bernstein polynomials $B_k^m(u)$ and the rational Bernstein polynomials $R_k^m(u)$ with $\beta_k > 0$ for all k.
If we demand $\beta_k > 0$ for all k we have all the properties and algorithms for rational Bezier curves which we have for ordinary i.e. non-rational curves [*Farin '83*], [*Tiller '83*], therefore there is in this case no principle difference between a curve-curve resp. a curve self-intersection algorithm for non-rational and for rational Bezier curves.

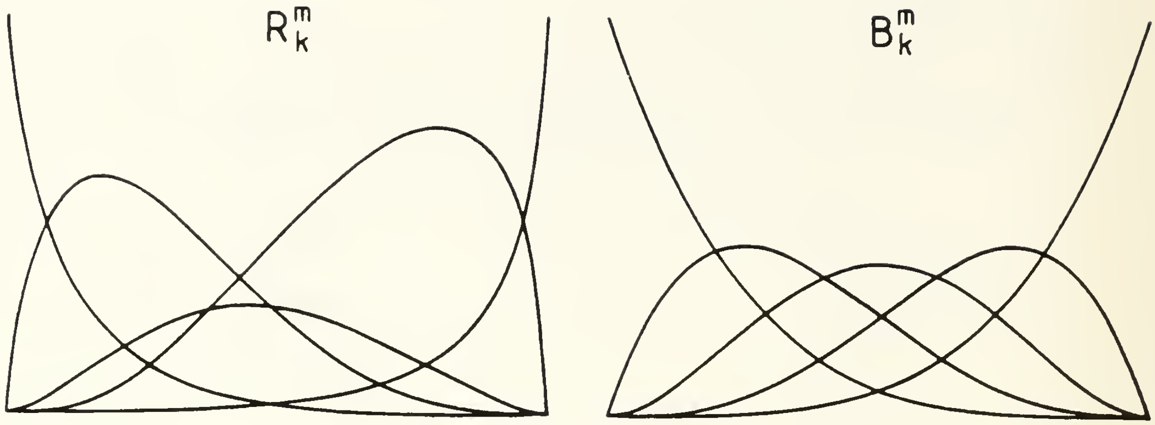


Figure 5. ordinary and rational Bernstein polynomial of degree four, $(\beta_0, \ldots, \beta_4) = (1, 3, 2, 5, 1)$

II. Curve-Curve Algorithm

The underlying idea of the curve-curve algorithm is to deal with the Bezier polygon instead of the curve description itself, using the relations between polygon and curve mentioned above.
The program of the algorithm is to subdivide both curves repeatedly which yields at the same time to a subdivision and refinement of the polygons. This is done until a fine polygon structure is obtained and the curves can be approximated well by the polygons defined by these subdivisions. This procedure reduces the problem to a number of straight line intersections that can be handled easily. Because subdividing the whole curves in each algorithm step is relatively time-consuming and needs a lot of storage space in addition an estimation of the intersection region is done.

The algorithm consists of four main parts (Figure 6), they are described now.

- First, the intersection area is estimated. Using a coarse but very quick estimate of the possible intersection regions of the two curves those parts of the curves that do not participate in the intersection will be eliminated as early as possible in the algorithm.
- Second, refinement occurs by subdividing the Bezier polygons. Except at the beginning, the algorithm subdivides not the whole Bezier polygons, but only those parts whose corresponding curve portions might participate in the intersection. An adaptive subdivision is done to detect the separation of regions of the two curves that do not intersect readily.
- Third, the intersection points are calculated by intersecting the Bezier polygons of the curve subsegments of possible intersection. Part three also calculates the parameter values of the intersection points.
- Fourth, error values are calculated, tolerances are checked, this part of the algorithm is the controlling unit of the algorithm and is very important for dealing with difficult and complicate cases.

Beside drawing parameters for creating the plot output, the input of the algorithm consists of the polynomial degrees (M and m) and of the Bezier points of the two Bezier curves ($\mathbf{B}(T)$ and $\mathbf{b}(t)$), furthermore of an error tolerance value to determine the accuracy needed. Pre-settings for controlling the algorithm can be specified in the program too.

The first step of the algorithm is to subdivide the two curves simultaneously forming two new subsegments on each curve. A *'min-max box'* defined by the maximum and minimum x and y coordinates of the curve segments defining Bezier points is built for each segment. The boxes of the two curves are then compared with each other (a comparison using min-max boxes instead of the convex hulls is rougher, but much more easily handled and quickly practised). Those subsegments

whose boxes do not intersect any box of the other curve will no longer be considered. Only those subsegments whose boxes can not be separated from that of their rivals will be dealt with further (Figure 7). For this, Bezier points of pairs of interfering subsegments of different curves will be provided with an subscript, called *'interference index'*. By this a list of pairs of segments of different curves which might interfere is created. In the following, Bezier points, i.e. segments of the same interference index, will all go through the algorithm subroutines.
The de Casteljau subdivision process, the min-max box formation and the separability test are connected by an algorithm loop, which will be done as often as is required by the level of accuracy needed. After each subdivision, two new subsegments are formed, each corresponding to a smaller convex hull. When more and more subdivisions are done each convex hull becomes smaller and smaller, while the curve topology near the intersection is reasonably closely approximated by the polygons of the subsegments.

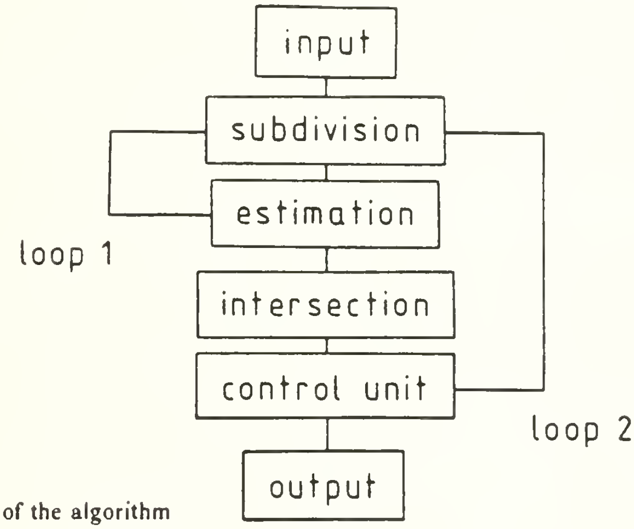


Figure 6. principle structure of the algorithm

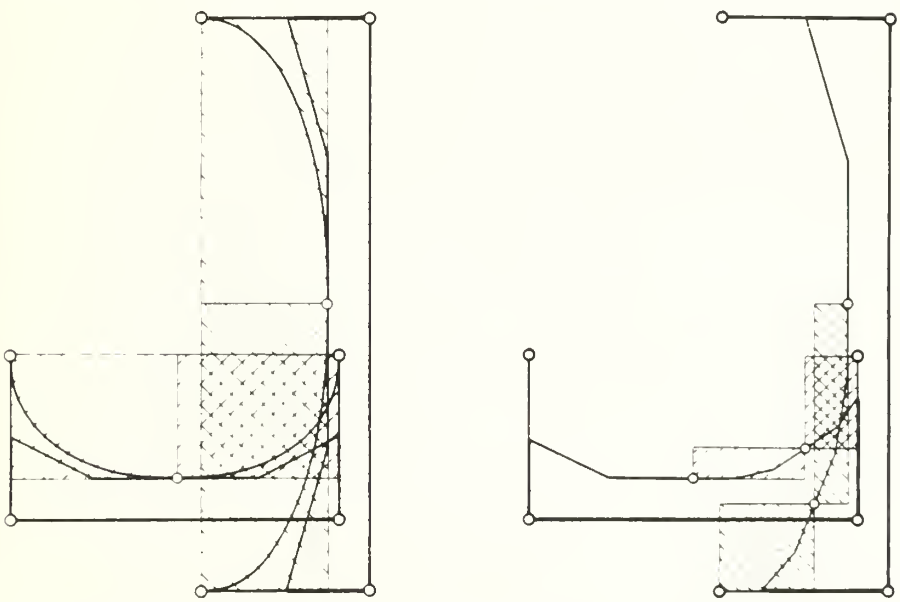


Figure 7. estimating the intersection region using min-max boxes

All subsegments which might participate in the intersection go through the third part of the algorithm: the section that computes the intersection points and the parameter values of the intersection points what is be done in the following way.

Let $\mathbf{B}_J(\tau)$ a subsegment of the first curve $\mathbf{B}(T)$ of degree M and

$$\mathbf{B}_j = (BX_j\,,\,BY_j) \qquad j = 0, \ldots, M$$

be the Bezier points of $\mathbf{B}_J(\tau)$ and let $\mathbf{b}_K(\tau)$ a subsegment of the second curve $\mathbf{b}(t)$ of degree m and

$$\mathbf{b}_k = (bx_k\,,\,by_k) \qquad k = 0, \ldots, m$$

be the Bezier points of $\mathbf{b}_K(\tau)$.
The polygon legs defined by the Bezier points are given by

$$\mathbf{G}_j = \mathbf{B}_j + \overline{T}_j\,\mathbf{S}_j \qquad j = 0, \ldots, M-1$$

where $\mathbf{G}_j = (GX_j\,,GY_j)$, $\mathbf{S}_j = (SX_j\,,SY_j)$, $\mathbf{S}_j = \mathbf{B}_{j-1} - \mathbf{B}_j$ and $\overline{T}_j \in [0,1]$ and similar for $\mathbf{g}_k$.
If $\mathbf{G}_j$ and $\mathbf{g}_k$ intersect in $\mathbf{P}$ (Figure 8) i.e.

$$\mathbf{G}_j(\overline{T}_j = \overline{T}_j(\mathbf{P})) = \mathbf{P} = \mathbf{g}_k(\bar{t}_k = \bar{t}_k(\mathbf{P}))$$

we have for the parameter values

$$\overline{T}_j(\mathbf{P}) = \frac{sy_k\,(BX_j - bx_k) - sx_k\,(BY_j - by_k)}{N_{jk}}$$

and

$$\bar{t}_k(\mathbf{P}) = \frac{SY_j\,(BX_j - bx_k) - SX_j\,(BY_j - by_k)}{N_{jk}}$$

where

$$N_{jk} = SY_j\,sx_k - sy_k\,SX_j$$

$\overline{T}_j(\mathbf{P})$ resp. $\bar{t}_k(\mathbf{P})$ are parameter values with respect to the polygon legs $\mathbf{G}_j$ resp. $\mathbf{g}_k$ but because the de Casteljau refinement is always done for 0.5 we also know the parameter value T_J of $\mathbf{B}_0 = \mathbf{B}_J(0)$ and the parameter value t_K of $\mathbf{b}_0 = \mathbf{b}_K(0)$ so that the parameter values $T(\mathbf{P})$ and $t(\mathbf{P})$ of the intersection point $\mathbf{P}$ with respect to the parameter intervals of the originally given Bezier curves can be calculated by (Figure 7)

$$T(\mathbf{P}) = T_J + \frac{T_j + (T_{j+1} - T_j)\,\overline{T}_j(\mathbf{P})}{2^s}$$

and similar for $t(\mathbf{P})$, where s is the number of subdivisions and T_j are the parameter values given to the Bezier points $\mathbf{B}_j$ of the Bezier polygon of $\mathbf{B}_J(\tau)$. The T_j (and so the t_k given to the $\mathbf{b}_k$) can be defined in different ways for example

by an **equidistant measure**

$$T_j^e = \frac{j}{M}$$

by an **chord length measure**

$$T_j^c = \frac{1}{L}\sum_{i=0}^{j} \|\,\mathbf{B}_{i+1} - \mathbf{B}_i\,\| \qquad where \quad L = \sum_{j=0}^{M-1} \|\,\mathbf{B}_{j+1} - \mathbf{B}_j\,\|$$

by an **geometric average measure** of T_j^e and T_j^c

$$T_j^g = \sqrt{T_j^e\,T_j^c}$$

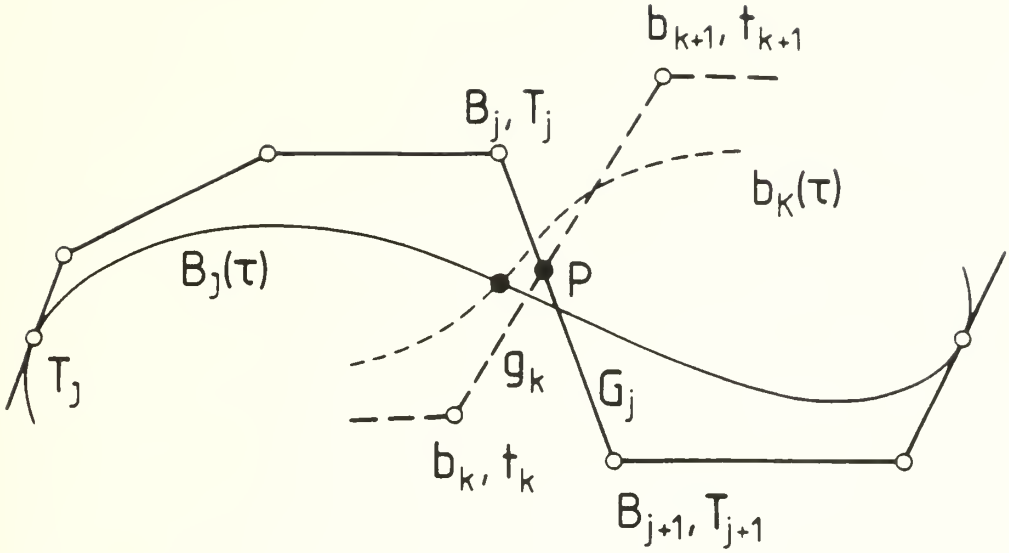


Figure 8. calculating of parameter values of the intersection points

As a **measure of error** we can use the distances

$$R_{Bb} = \| \mathbf{B}(T(\mathbf{P})) - \mathbf{b}(t(\mathbf{P})) \|$$

$$R_{BP} = \| \mathbf{B}(T(\mathbf{P})) - \mathbf{P} \|$$

$$R_{bP} = \| \mathbf{b}(t(\mathbf{P})) - \mathbf{P} \|$$

Per default a minimum number of de Casteljau subdivisions will be done before part three will be started (loop 1). If the accuracy needed is 0.002 for example the pre-setting has to be 6 (see table 1) an this will yield in almost every example to an accuracy of about 0.002, if in some complicated case not, the control unit will effect to do as many additional subdivisions as needed for the specified accuracy (loop 2).
When the two curves intersect in a very small angle or do not intersect, but come very close together part three might calculate more intersection points as two curves of degree M and m can produce or might calculate (pseudo-)intersection points lying very close together in parameter space which has to be checked (the statement of the parameter space criterion is stronger than an statement of an analog coordinate space criterion). In both cases the control unit will also effect to do as many additional subdivisions as needed for clarifying the situation.
The repeatedly done polygon refinement initialised by these criterions will be stopped in different ways: first, if the result has the accuracy needed, second, there is a default of an upper boundary for the number of de Casteljau subdivisions and third, there is a default of an maximal (possible) accuracy. This default value is dependent on the initialization of the variables, e.g. real or double precision real and of the machine accuracy for each kind of initialization.
Finally the control unit checks if the distance between intersection points in coordinate space is less than a specified tolerance. If yes, an intersection point is defined by the arithmetic average of these points.

Examples

Table 1 lists the maximal error

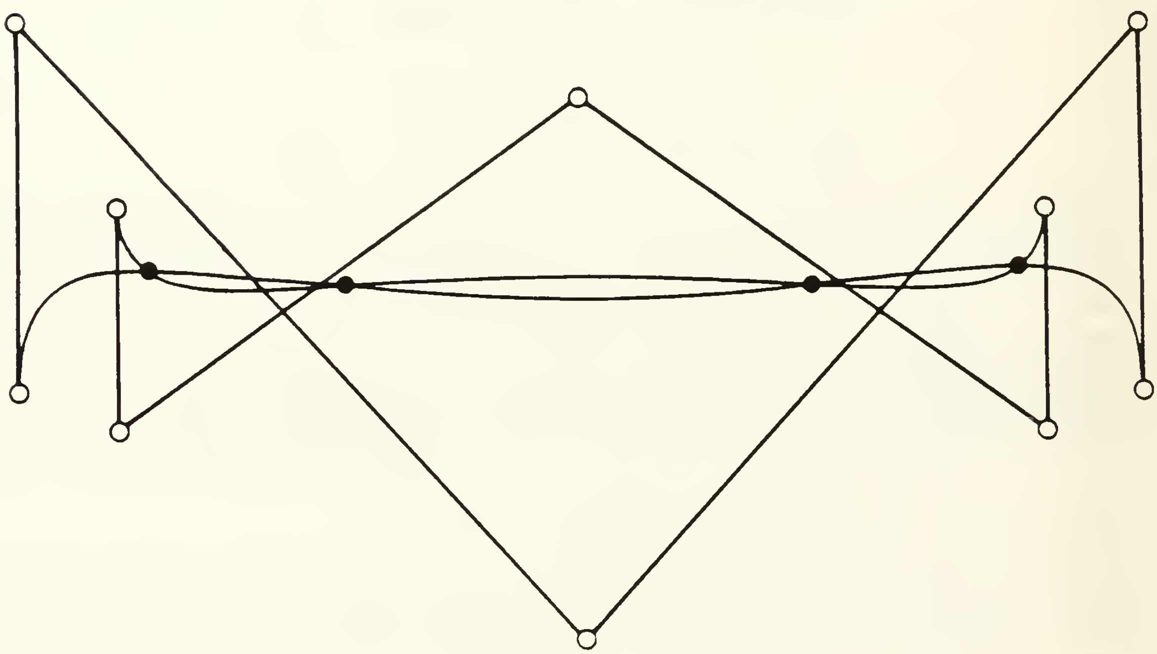
$$R = \max_{\forall \mathbf{P}} \{ R_{Bb}, R_{BP}, R_{bP} \} \qquad \otimes$$

as it depend upon an increasing subdivision factor for the examples 1 to 6 for equidistant parameterization for which we got the best results.

subdivision factor	4	5	6	7	8
Example 1	0.00904	0.00130	0.00039	0.00012	0.00003
Example 2	0.04563	0.00791	0.00210	0.00051	0.00014
Example 3	0.00273	0.00071	0.00014	0.00005	0.00001
Example 4	0.00450	0.00271	0.00121	0.00047	0.00001
Example 5	0.03817	0.01037	0.00260	0.00062	0.00019
Example 6	0.05235	0.00856	0.00177	0.00057	0.00012

Table 1. R for equidistant parameterization

Example 1



$P.X$	$P.Y$	$T(\mathbf{P})$	$t(\mathbf{P})$
-3.12109	0.76362	0.09834	0.20604
-1.67341	0.60298	0.32366	0.35662
1.67341	0.60298	0.67634	0.64338
3.12109	0.76362	0.90166	0.79396

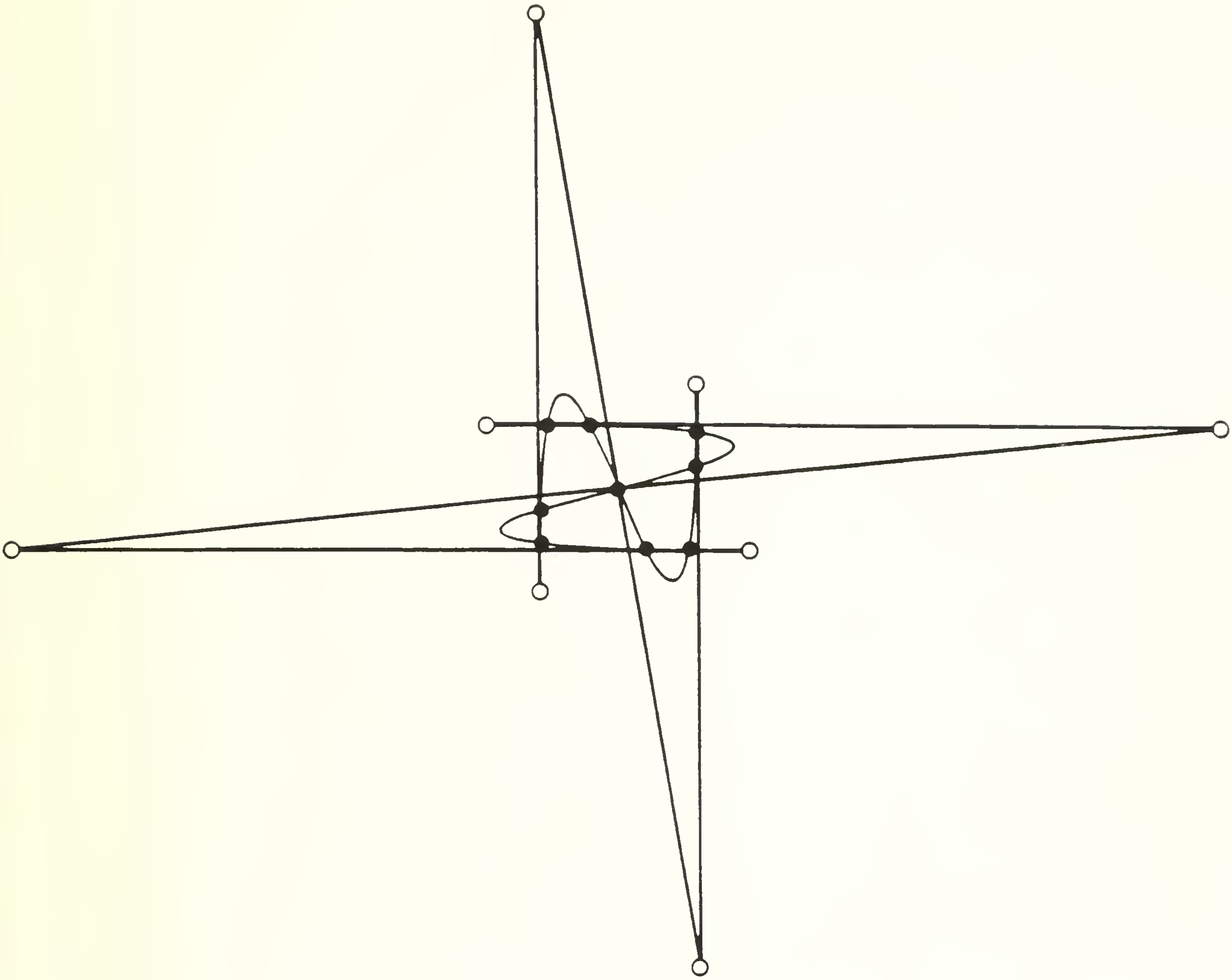
parameter values and x-y-coordinates of the intersection points

bx	by
-3.3	1.3
-3.3	-0.7
0.0	2.3
3.3	-0.7
3.3	1.3

BX	BY
-4.0	-0.35
-4.0	3.0
0.0	-2.6
4.0	3.0
4.0	-0.35

Bezier points of $\mathbf{b}(t)$ and of $\mathbf{B}(T)$

Example 2



PX	*PY*	*T*(**P**)	*t*(**P**)
0.00809	1.17249	0.03029	0.85430
0.02596	1.97778	0.05471	0.61825
0.17250	3.99191	0.14570	0.03029
0.97778	3.97404	0.38175	0.05471
1.50000	2.50000	0.50000	0.50000
2.02221	1.02596	0.61825	0.94529
2.82750	1.00809	0.85430	0.96971
2.97404	3.02221	0.94529	0.38175
2.99191	3.82750	0.96971	0.14570

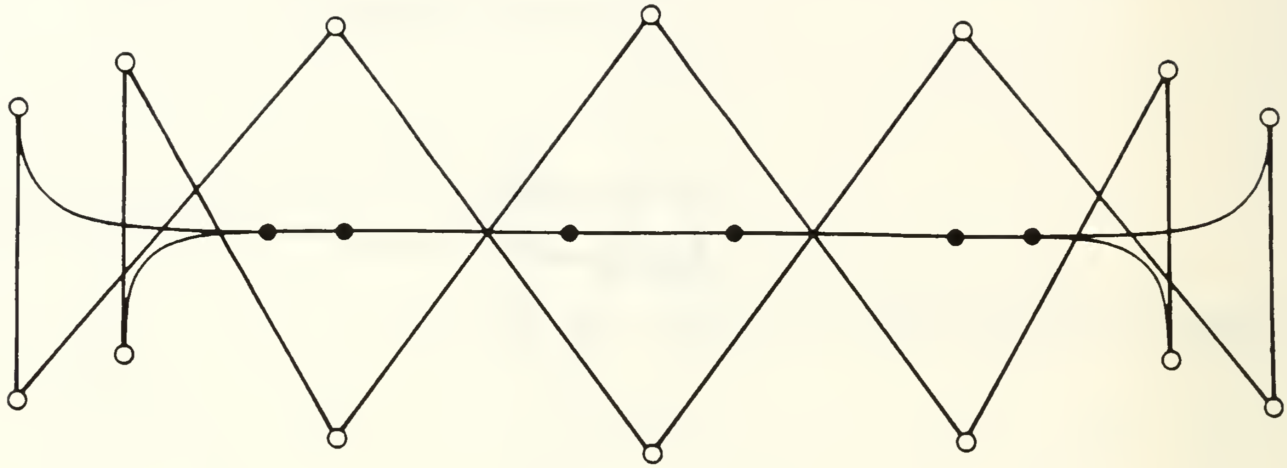
parameter values and x-y-coordinates of the intersection points

bx	*by*
0.0	0.0
0.0	14.0
3.0	-9.0
3.0	5.0

BX	*BY*
-1.0	4.0
13.0	4.0
-10.0	1.0
4.0	1.0

Bezier points of **b**(t) and of **B**(T)

Example 3



PX	PY	$T(\mathbf{P})$	$t(\mathbf{P})$
-3.64353	1.49822	0.23120	0.27305
-2.92393	1.50086	0.29330	0.32148
-0.77325	1.49989	0.44827	0.45409
0.77325	1.49989	0.55173	0.54591
2.92393	1.50086	0.70670	0.67852
3.64353	1.49822	0.76880	0.72695

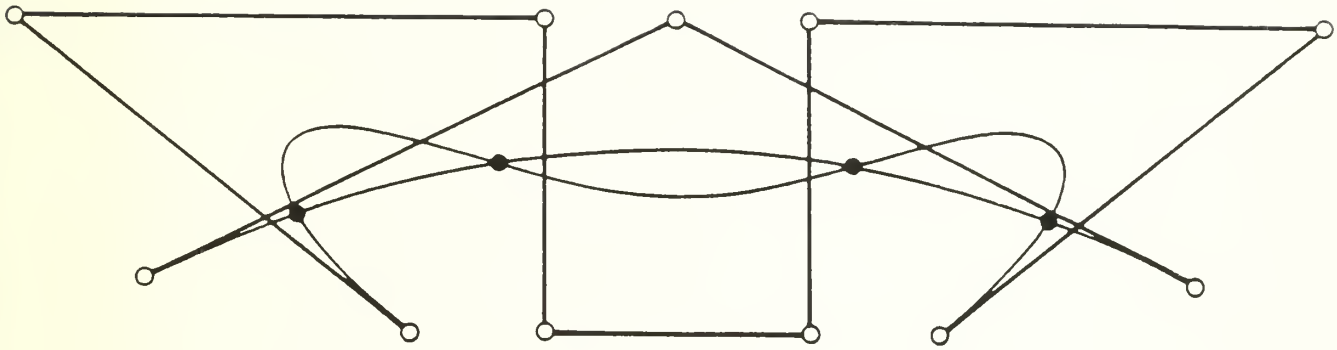
parameter values and x-y-coordinates of the intersection points

bx	by
-5.0	0.0
-5.0	3.555
-3.0	-1.0
0.0	4.17
3.0	-1.0
5.0	3.555
5.0	0.0

BX	BY
-6.0	3.0
-6.0	-0.555
-3.0	4.0
0.0	-1.17
3.0	4.0
6.0	-0.555
6.0	3.0

Bezier points of $\mathbf{b}(t)$ and of $\mathbf{B}(T)$

Example 4



PX	PY	$T(\mathbf{P})$	$t(\mathbf{P})$
-5.69310	2.23393	0.14418	0.06613
-2.68113	3.21920	0.33243	0.35152
2.68113	3.21920	0.66757	0.64848
5.69310	2.23393	0.85582	0.93387

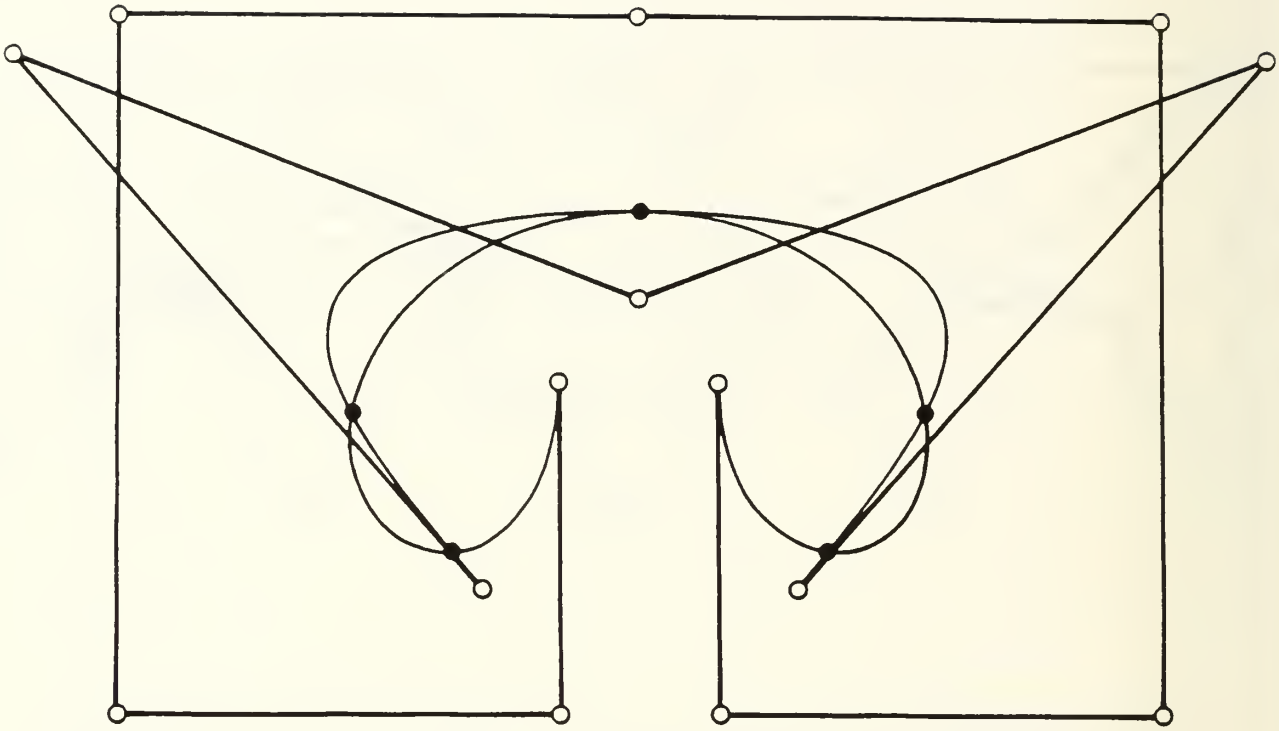
parameter values and x-y-coordinates of the intersectuion points

bx	by
-4.0	0.0
-10.0	6.0
-2.0	6.0
-2.0	0.0
2.0	0.0
2.0	6.0
10.0	6.0
4.0	0.0

BX	BY
-8.0	1.0
0.0	6.0
8.0	1.0

Bezier points of $\mathbf{b}(t)$ and of $\mathbf{B}(T)$

Example 5



PX	PY	$T(\mathbf{P})$	$t(\mathbf{P})$
-3.60359	-4.10631	0.01787	0.12443
-5.44653	-0.76332	0.10171	0.28110
0.00000	4.14844	0.50000	0.50000
5.44653	-0.76332	0.89829	0.71890
3.60359	-4.10631	0.98213	0.87557

parameter values and x-y-coordinates of the intersection points

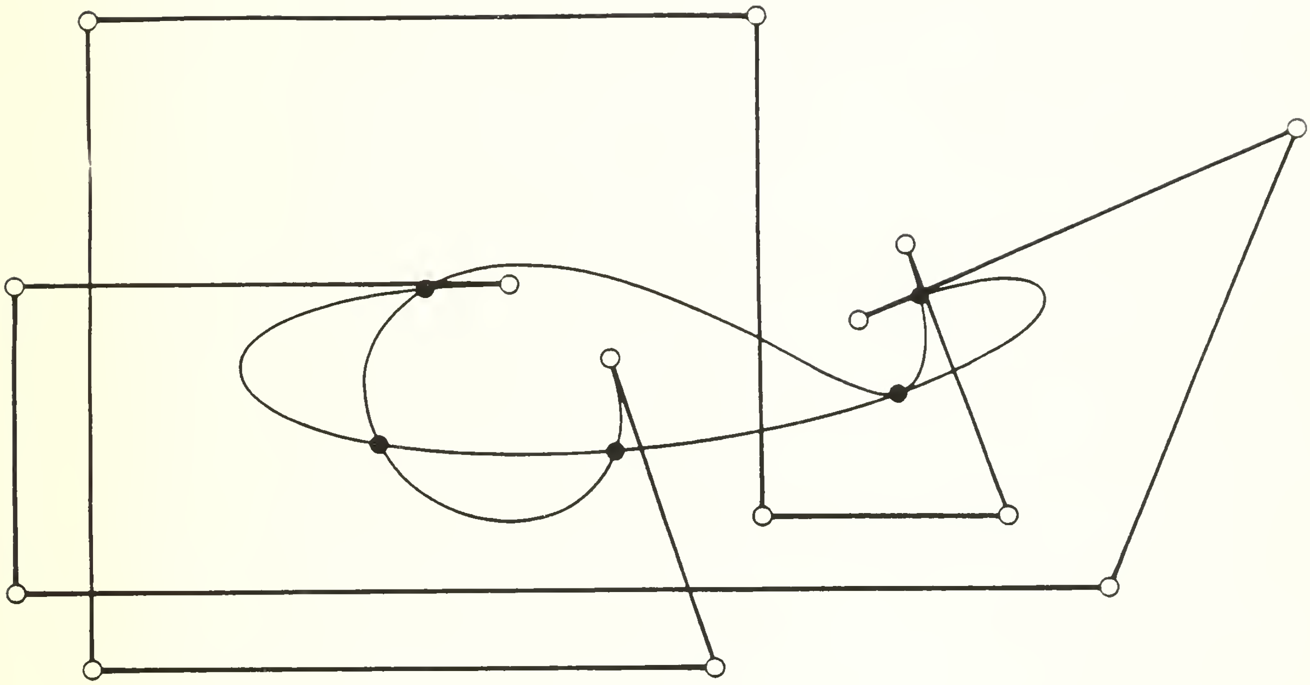
bx	by
-1.5	0.0
-1.5	-8.0
-10.0	-8.0
-10.0	9.0
0.0	9.0
10.0	9.0
10.0	-8.0
1.5	-8.0
1.5	0.0

BX	BY
-3.0	-5.0
-12.0	8.0
0.0	BY_2
12.0	8.0
3.0	-5.0

$BY_2 = 2.062507$

Bezier points of $\mathbf{b}(t)$ and of $\mathbf{B}(T)$

Example 6



PX	PY	$T(\mathbf{P})$	$t(\mathbf{P})$
6.29966	1.63288	0.03184	0.96977
5.87601	-0.86192	0.33990	0.85797
0.04246	-2.38219	0.49353	0.05087
-4.67397	-2.17973	0.62148	0.28232
-3.57214	1.91463	0.96618	0.46102

parameter values and x-y-coordinates of the intersection points

bx	by
5.0	1.0
14.0	6.0
10.0	-6.0
-12.0	-6.0
-12.0	2.0
-2.0	2.0

BX	BY
0.0	0.0
2.0	-8.0
-10.5	-8.0
-10.5	9.0
3.5	9.0
3.0	BY_5
8.0	BY_6
6.0	3.0

Bezier points of $\mathbf{b}(t)$ and of $\mathbf{B}(T)$

$(BY_5 = BY_6 = -4.129807)$

III. Self-Intersection Algorithm

It is not possible to calculate the self-intersection of a Bezier curve by the curve-curve algorithm of part II by doing the curve input twice because in this case the separability test of min-max boxes will always be positive so that no elimination of curve parts that do not participate in the self-intersection is possible. Furthermore part three will fail by doing the same input twice, so that an additional criterion is necessary.
What we would like to have is a geometric criterion based on a relation between the curve and its defining Bezier points i.e. its Bezier polygon which is as simple and at the same time as strong as the convex hull property. This turns out to be more difficult than it looks like first, because the situation is complicated by the fact that

- it is possible that the Bezier polygon has a self-intersection but the Bezier curve has no self-intersection (see Figure 9)

but on the other side even

- if the Bezier curve has a self-intersection the Bezier polygon does not have to have a self-intersection (see Figure 10).

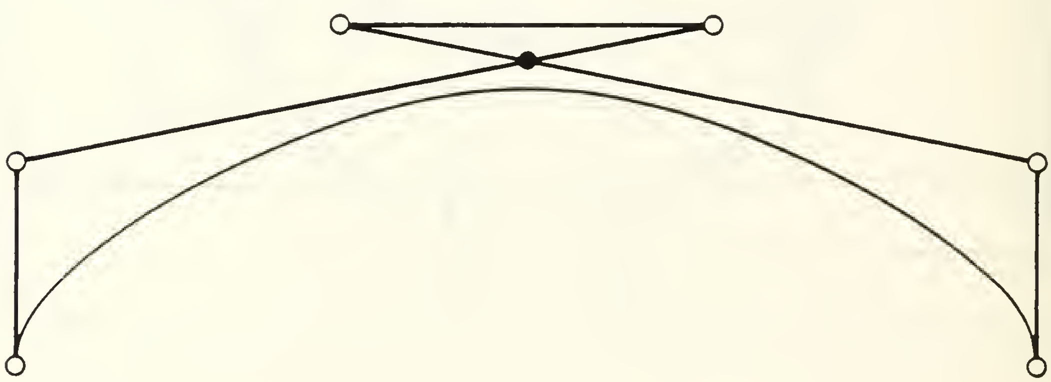


Figure 9. polygon self-intersection

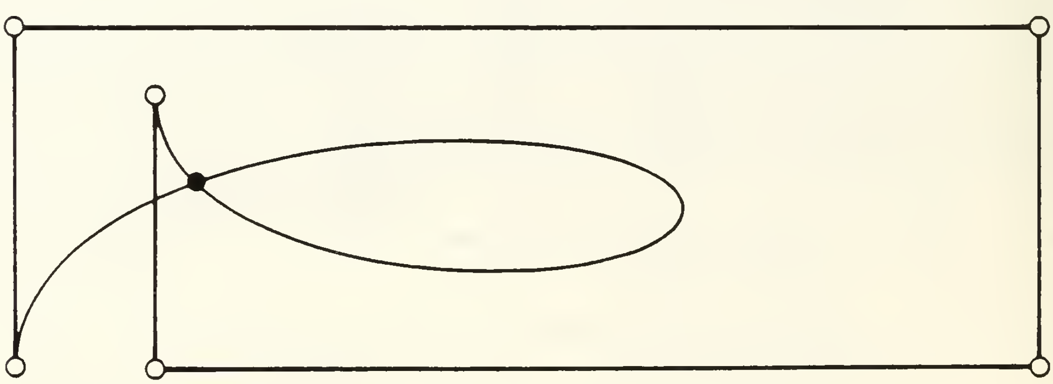


Figure 10. curve self-intersection

Furthermore,

- if $\sum |\alpha_k|$, i.e. the sum of the amounts of the rotation angles α_k of the Bezier polygon legs, is greater than π the Bezier curve does not have to have a self-intersection (see Figure 11) and even
- if the sum (from $u = 0$ to $u = 1$) of the amount of the rotation angle of the tangent vector $B'(u)$ of the Bezier curve is greater than π the Bezier curve does not have to have a self-intersection (see Figure 12).

But,

- the sum of the amount of the rotation angle of the tangent vector of the Bezier curve is greater than π if the Bezier curve has a self-intersection (see Figure 13).

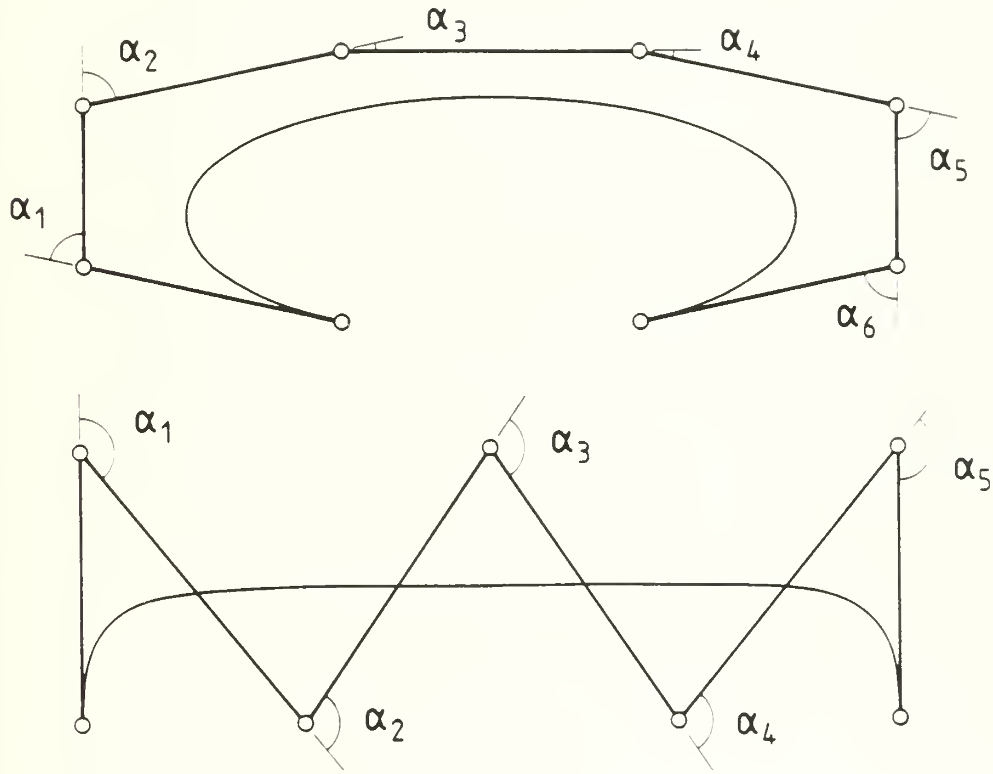


Figure 11. $\sum |\alpha_k| > \pi$, no self-intersection

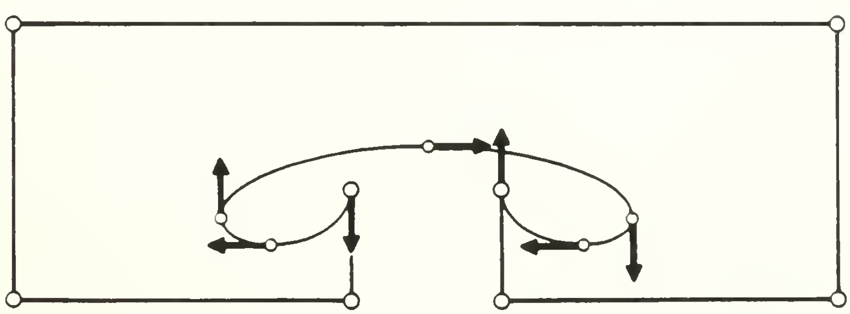


Figure 12. no self-intersection

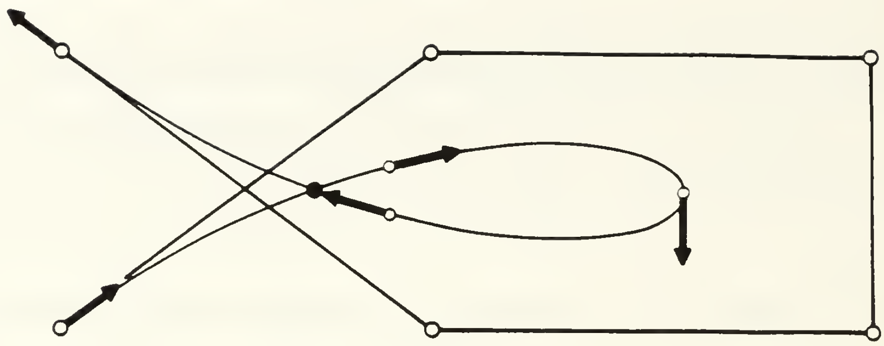


Figure 13. Bezier curve with self-intersection

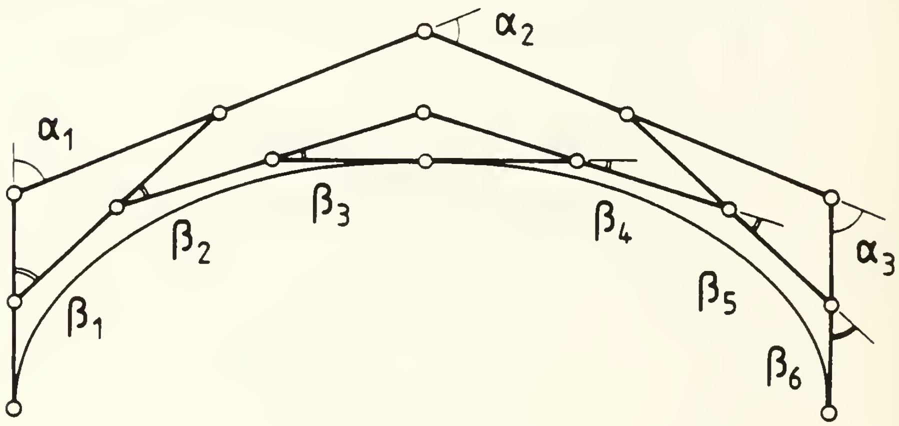


Figure 14. all α_k with same orientation $\Rightarrow \sum |\alpha_k| = \sum |\beta_k| = \ldots$

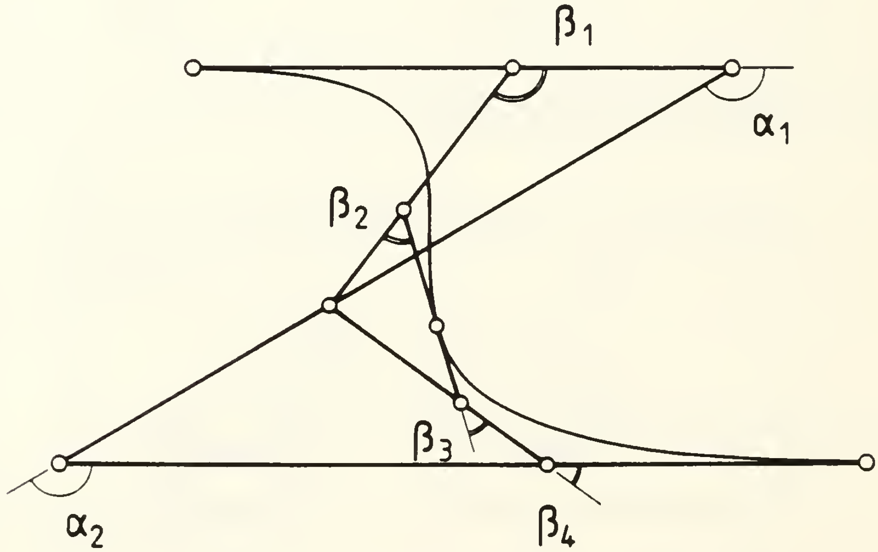


Figure 15. α_k with different orientation $\Rightarrow \sum |\alpha_k| > \sum |\beta_k| > \ldots$

Because of the de Casteljau construction which creates in every step a convex combination of the $\mathbf{b}_i^0$ and because of the approximation property mentioned in part I, the sum $\sum |\alpha_k|$ is equal to the sum of the amount of the rotation angle of the tangent vector of the Bezier curve if the orientation of the rotation angles of the Bezier polygon legs is the same in every inner Bezier point (see Figure 14). But the sum of the amount of the rotation angle of the tangent vector of the Bezier curve is smaller than $\sum |\alpha_k|$ if the orientation of the rotation angles of the Bezier polygon legs is not the same in every inner Bezier point (see Figure 15), that follows from the smoothing property of the de Casteljau subdivision process together with the approximation property mentioned in part I.
So we have the statement that

- the sum of the amount of the rotation angle of the tangent vector of the Bezier curve is always smaller or equal the sum $\sum |\alpha_k|$ of the amounts of the rotation angles of the Bezier polygon legs.

By combining the two ▪ statements we get the

- *angle criterion:* The sum $\sum |\alpha_k|$ of the amounts of the rotation angles of the Bezier polygon legs is greater than π if the Bezier curve has a self-intersection.

For the algorithm we will use the contraposition of the criterion.

- *angle criterion:* A Bezier curve has no self-intersection if the sum $\sum |\alpha_k|$ of the amounts of the rotation angles of the Bezier polygon legs is smaller or equal than π.

By this we have a very simple geometric criterion for deciding whether a curve has a self-intersection or not and for the elimination of curve parts that do not participate in a self-intersection. What we have to do is to calculate the sum $\sum |\alpha_k|$ of the polygon angles α_k and compare with π. If we have $\sum |\alpha_k| \leq \pi$ we know that there is no self-intersection of the curve (Figure 16.1 and 16.2), but if $\sum |\alpha_k| > \pi$ the curve might have a self-intersection (Figure 16.3 and 16.4). For clarifying we subdivide using de Casteljau and check the smaller parts again against the angle criterion.

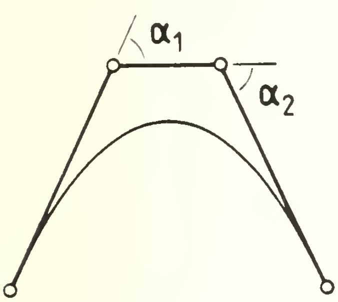


Figure 16.1

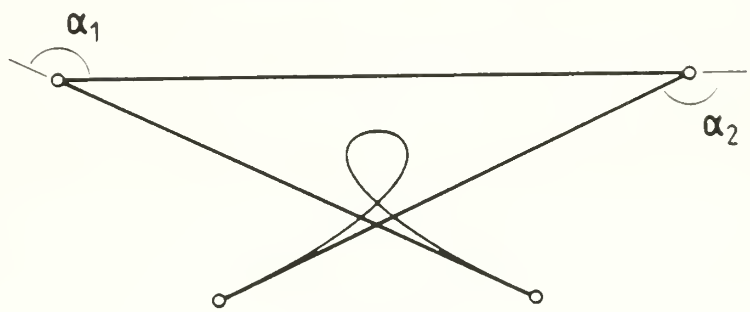


Figure 16.3

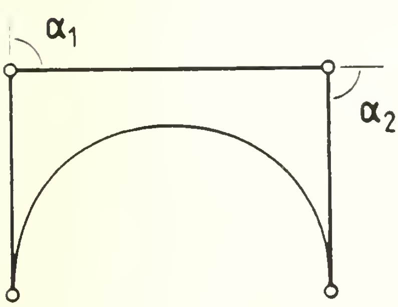


Figure 16.2

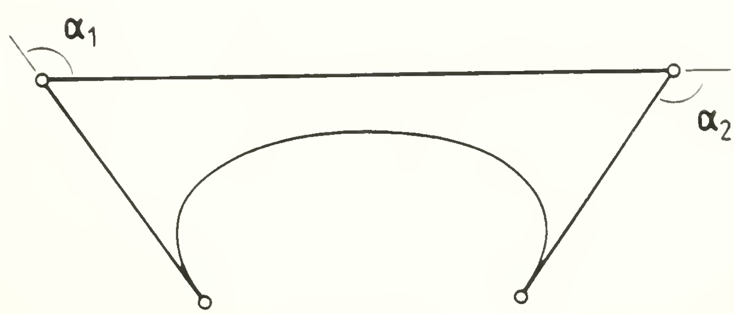


Figure 16.4

Figure 16. the angle criterion

To build up a self-intersection algorithm the idea of the angle test has to be combined with the idea of the min-max box test. This is done in the following way.
The algorithm consists again of the four main parts of Figure 6. But part one of the algorithm for estimating the self-intersection region of the curve consists now of two different tests. the min-max box test and the angle test. Figure 16 gives the example of an subdivided Bezier curve having several self-intersections. As we can see, there is a subsegment (subsegment $\mathbf{B}_5$) with self-intersection (point $\mathbf{P}_1$). there are two subsegments with common boundary point (subsegments $\mathbf{B}_2$ and $\mathbf{B}_3$) creating the self-intersection point $\mathbf{P}_2$ and there are subsegments (subsegments $\mathbf{B}_1$ and $\mathbf{B}_6$) which are not connected to each other but create the self-intersection point $\mathbf{P}_3$ of the Bezier curve $\mathbf{B}(T)$. To distinguish between these three different cases and for controlling the algorithm we introduce a so called *'genus index'*.
The self-intersection of a segment of genus one that means a segment of case one has to be checked by using the angle criterion. If the angle test is positive e.g. $\sum |\alpha_i| > \pi$ a refinement has to be done to clarify the situation. The refinement of a genus one segment produces two subsegments of genus one and one pair of subsegments of genus two.
A pair of subsegments of genus two that means subsegments with a common boundary point have also to be checked against the angle criterion but now the angle sum of both polygons has to be calculated. The min-max box criterion can not be used because of the common boundary point of the two segments. If the angle test is positive a refinement has to be done for both segments, it produces one pair of subsegments of genus two and three pairs of subsegments of genus three.
Subsegments of genus three can be dealt with as in the curve-curve algorithm of part II i.e. for calculating the self-intersection point $\mathbf{P}_3$ of Figure 16 we do need only the min-max box test not the angle test because the refinement of pairs of segments of genus three can produce pairs of subsegments of genus three only and no (pairs of) subsegments of genus one or two.

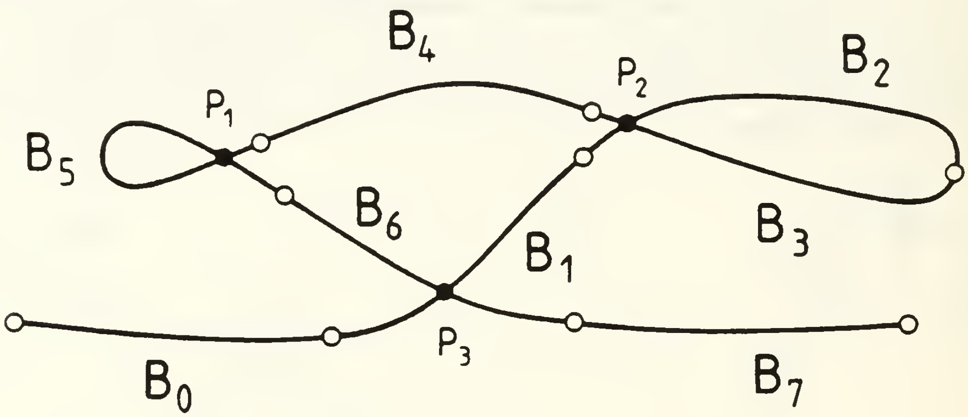


Figure 17. possible subsegment configurations contributing to the self-intersection

In the first step the algorithm has to deal only with one curve segment of genus one - the Bezier curve segment which has to be checked for self-intersections. If the angle test is positive a refinement has to be done, so that in the second step the algorithm has to deal with two subsegments of genus one and one pair of subsegments of genus two and the result of this step might be subsegments of genus one and pairs of subsegments of genus two or three. When more and more subdivisions are done not only each convex hull becomes smaller and smaller but because of the approximation property of the Bezier polygon also the angle sume of each subsegment becomes smaller and smaller so that after an initial increase of (pairs of) subsegments of genus one and two the number of (pairs of) subsegments of genus one and two decreases very fast until there are only pairs of subsegments of genus three. From this moment on the self-intersection algorithm works in the same way as the curve-curve algorithm described in part II of the paper. That also means that part two and part three of the algorithm - the subdivision of the curve in the aim of refinement and the calculation of the self-intersection points and parameter values - is done in exactly the same way as described in part II.

The control unit also works as in part II describted except that it checks in addition if the subdivided control polygon turns through 180 degrees at a subdivision point which implies a cusp at this point (Example 4).

Examples

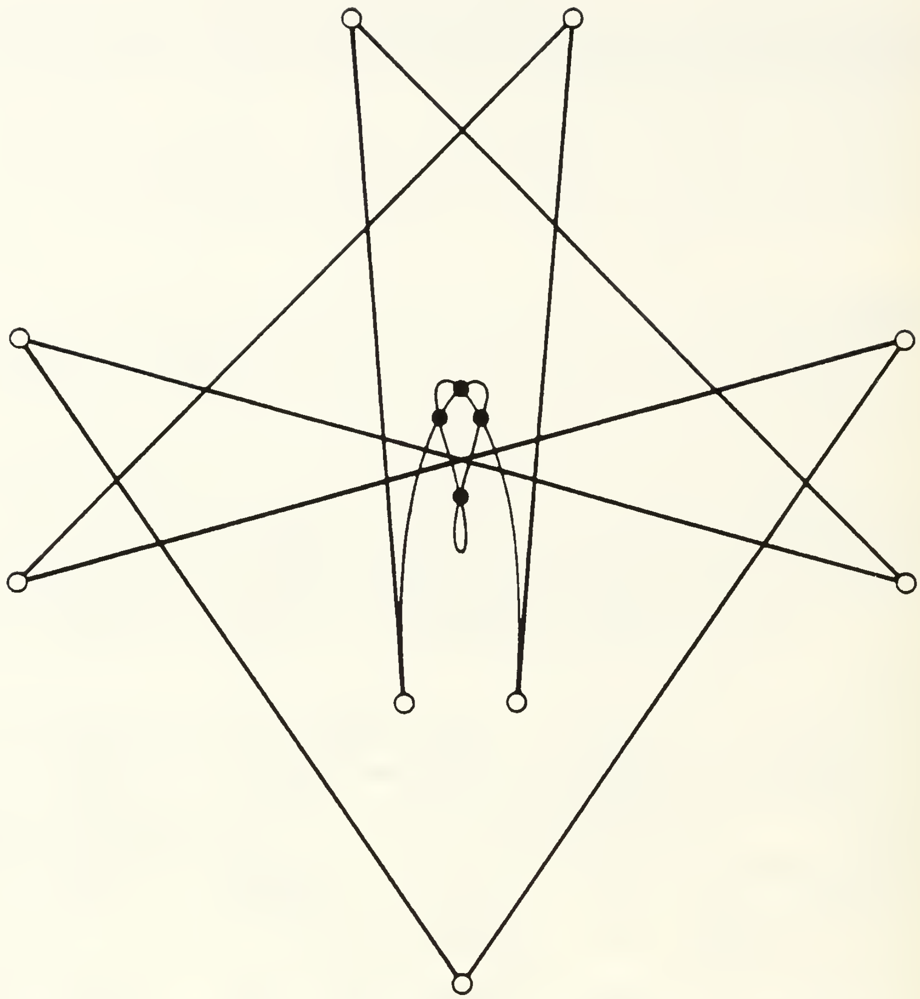
Table 2 lists the maximal error given by ⊗ part II as it depend upon an increasing subdivision factor for the examples 1 to 12 for equidistant parameterization.

subdivision factor	4	5	6	7	8
Example 1	0.28064	0.01093	0.00568	0.00269	0.00110
Example 2	0.08635	0.03628	0.01297	0.00189	0.00075
Example 3	0.20799	0.09101	0.02185	0.00619	0.00101
Example 4					
Example 5	0.02390	0.00445	0.00156	0.00007	0.00003
Example 6	0.03326	0.01207	0.00236	0.00080	0.00000
Example 7	0.04659	0.01579	0.00055	0.00027	0.00012
Example 8	0.06235	0.02459	0.00620	0.00150	0.00040
Example 9	0.10368	0.01132	0.00434	0.00162	0.00032
Example 10	0.10184	0.01123	0.00427	0.00159	0.00032
Example 11					
Example 12	0.07962	0.00794	0.00332	0.00124	0.00025

Table 2. **R** for equidistant parameterization

Because of the *'bad character'* of the two cusps appearing in **Example 11**, this example requires more than 8 subdivisions for the decision if the curve has self-intersections or cusps.

Example 1



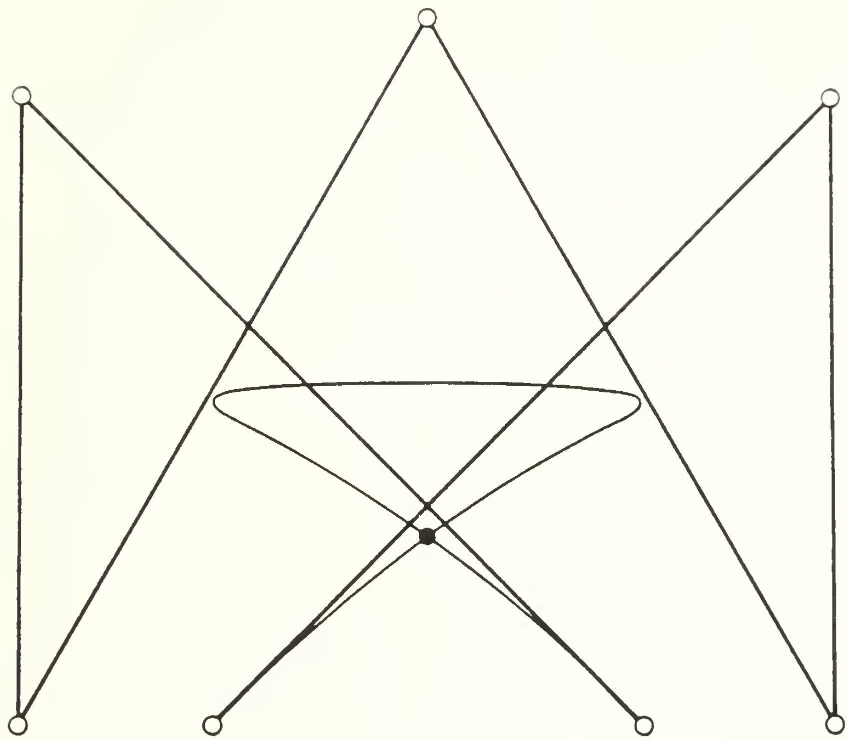
PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.00000	8.06782	0.35510	0.64490
-0.35685	9.99926	0.09241	0.75224
0.00000	10.70820	0.12485	0.87515
0.35685	9.99926	0.24776	0.90759

parameter values and x-y-coordinates of the intersection points

bx	by
-1.0	3.0
-2.0	20.0
8.0	6.0
-8.0	12.0
0.0	-4.0
8.0	12.0
-8.0	6.0
2.0	20.0
1.0	3.0

Bezier points of $\mathbf{b}(t)$

Example 2



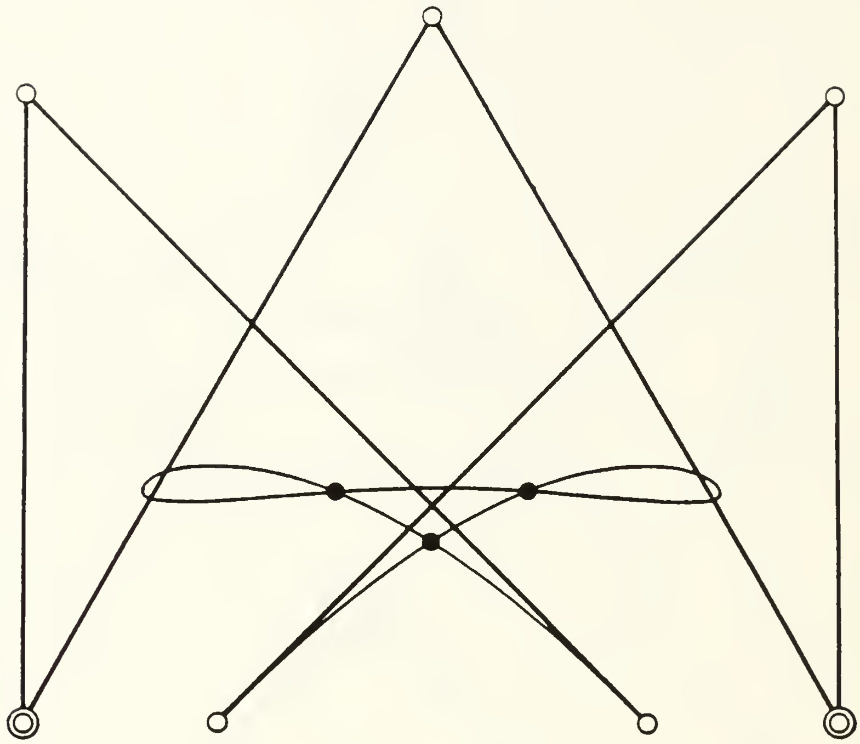
$P.X$	$P.Y$	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.00000	5.63639	0.06897	0.93103

parameter values and x-y-coordinates of the intersection point

bx	by
-2.0	8.0
3.8	0.0
3.8	8.0
0.0	-1.0
-3.8	8.0
-3.8	0.0
2.0	8.0

Bezier points of $\mathbf{b}(t)$

Example 3



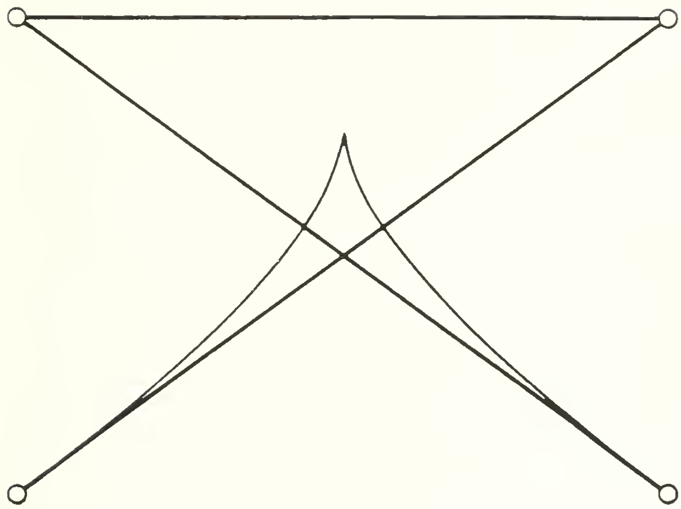
PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.90426	5.07460	0.08358	0.44207
-0.90426	5.07460	0.55793	0.91642
0.00000	5.71900	0.05154	0.94846

parameter values and x-y-coordinates of the intersection points

bx	by
-2.0	8.0
3.8	0.0
3.8	8.0
3.8	8.0
0.0	-1.0
-3.8	8.0
-3.8	8.0
-3.8	0.0
2.0	8.0

Bezier points of $\mathbf{b}(t)$

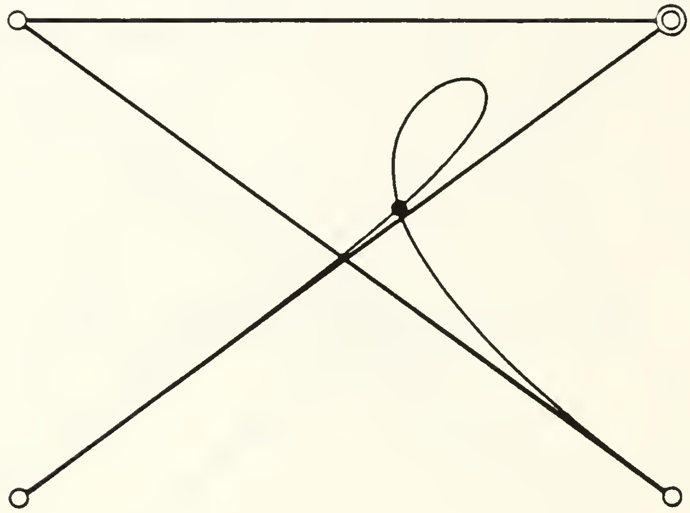
Example 4



bx	by
-2.0	0.0
2.0	4.0
-2.0	4.0
2.0	0.0

Bezier points of $\mathbf{b}(t)$

Example 5



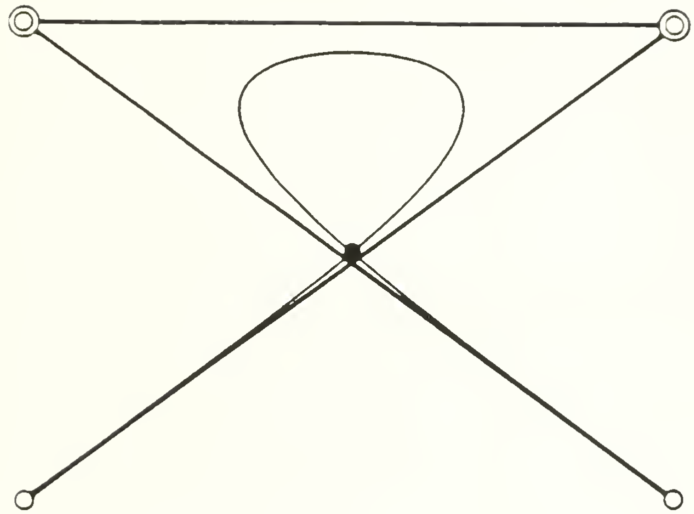
PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.33333	2.44444	0.21133	0.78867

parameter values and x-y-coordinates of the intersection point

bx	by
-2.0	0.0
2.0	4.0
2.0	4.0
-2.0	4.0
2.0	0.0

Bezier points of $\mathbf{b}(t)$

Example 6



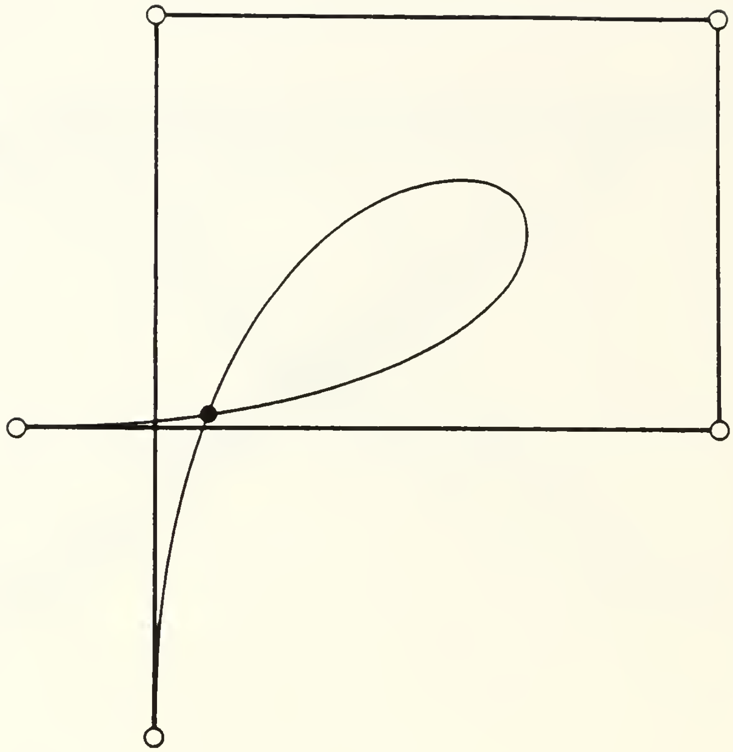
PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.00000	2.08204	0.13673	0.86327

parameter values and x-y-coordinates of the intersection point

bx	by
-2.0	0.0
2.0	4.0
2.0	4.0
-2.0	4.0
-2.0	4.0
2.0	0.0

Bezier points of $\mathrm{b}(t)$

Example 7



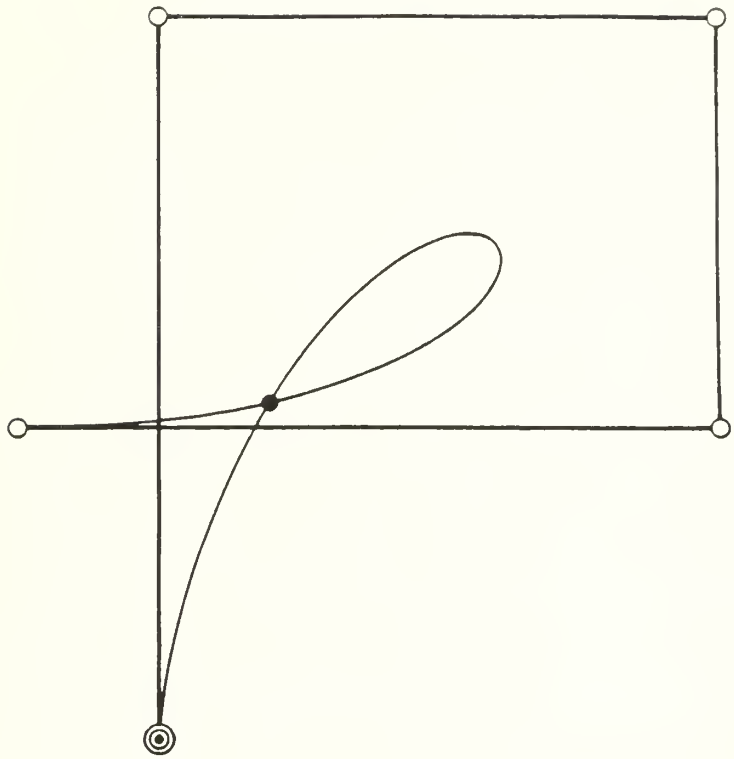
PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.77089	6.26442	0.14003	0.92162

parameter values and x-y-coordinates of the intersection point

bx	by
0.0	0.0
0.0	14.0
8.0	14.0
8.0	6.0
-2.0	6.0

Bezier points of $\mathbf{b}(t)$

Example 8



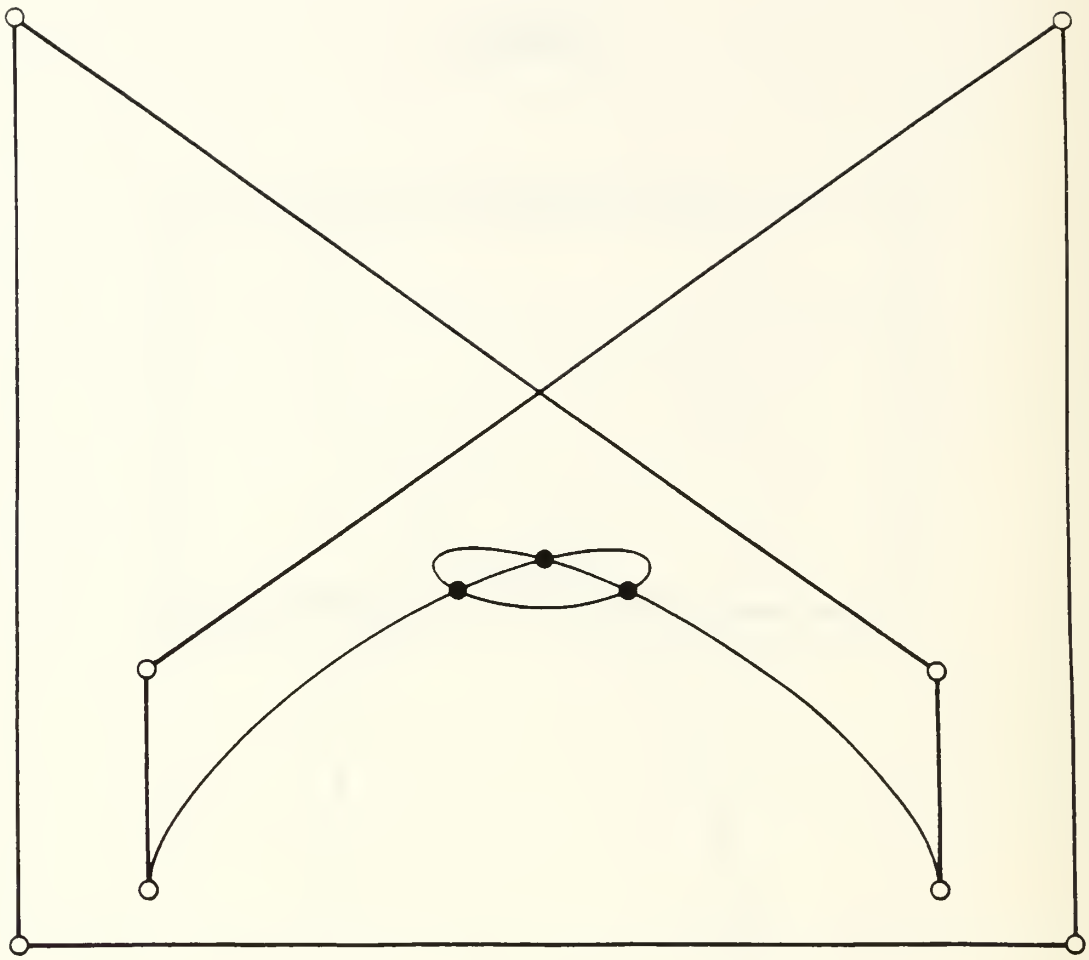
PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
1.59322	6.51678	0.41859	0.92722

parameter values and x-y-coordinates of the intersection point

bx	by
0.0	0.0
0.0	0.0
0.0	0.0
0.0	14.0
8.0	14.0
8.0	6.0
-2.0	6.0

Bezier points of $\mathbf{b}(t)$

Example 9



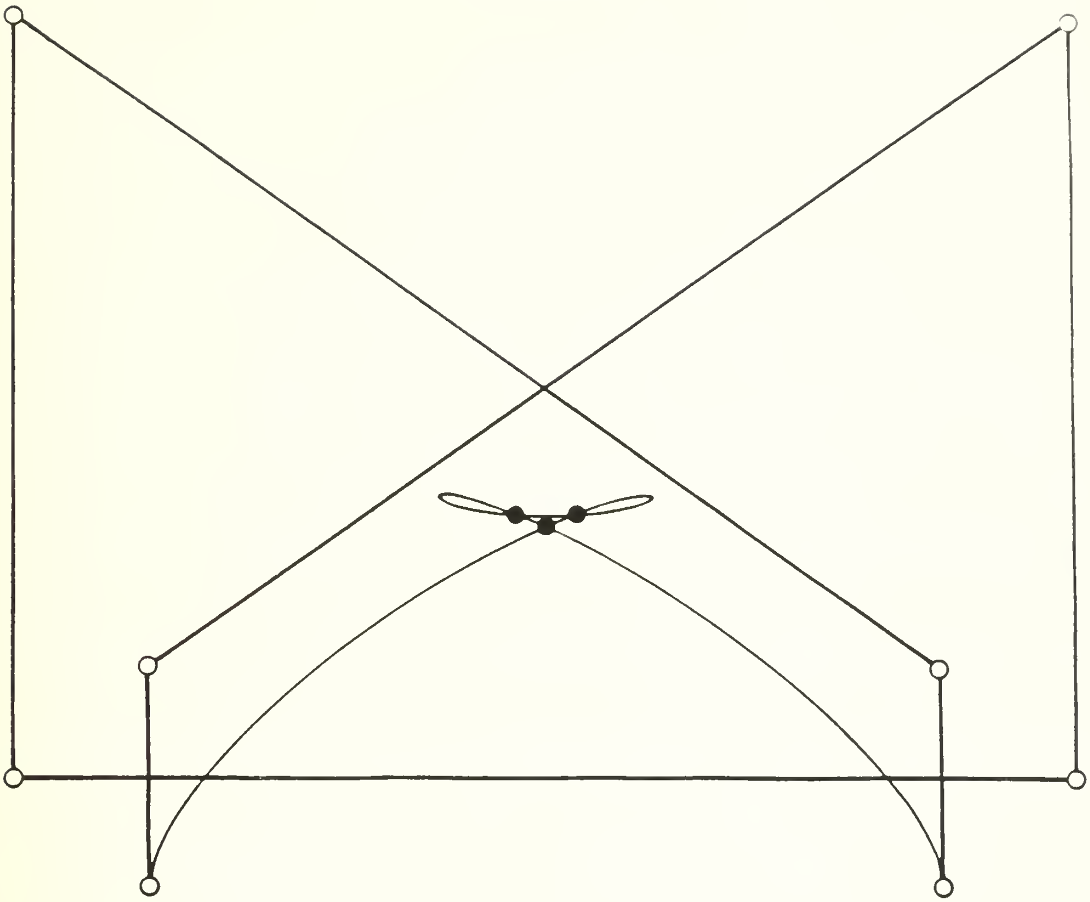
PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
-0.65030	2.72773	0.17750	0.58924
0.00000	3.01382	0.22167	0.77833
0.65030	2.72773	0.41076	0.82250

parameter values and x-y-coordinates of the intersection points

bx	by
-3.0	0.0
-3.0	2.0
4.0	8.0
4.0	-0.5
-4.0	-0.5
-4.0	8.0
3.0	2.0
3.0	0.0

Bezier points of $\mathbf{b}(t)$

Example 10



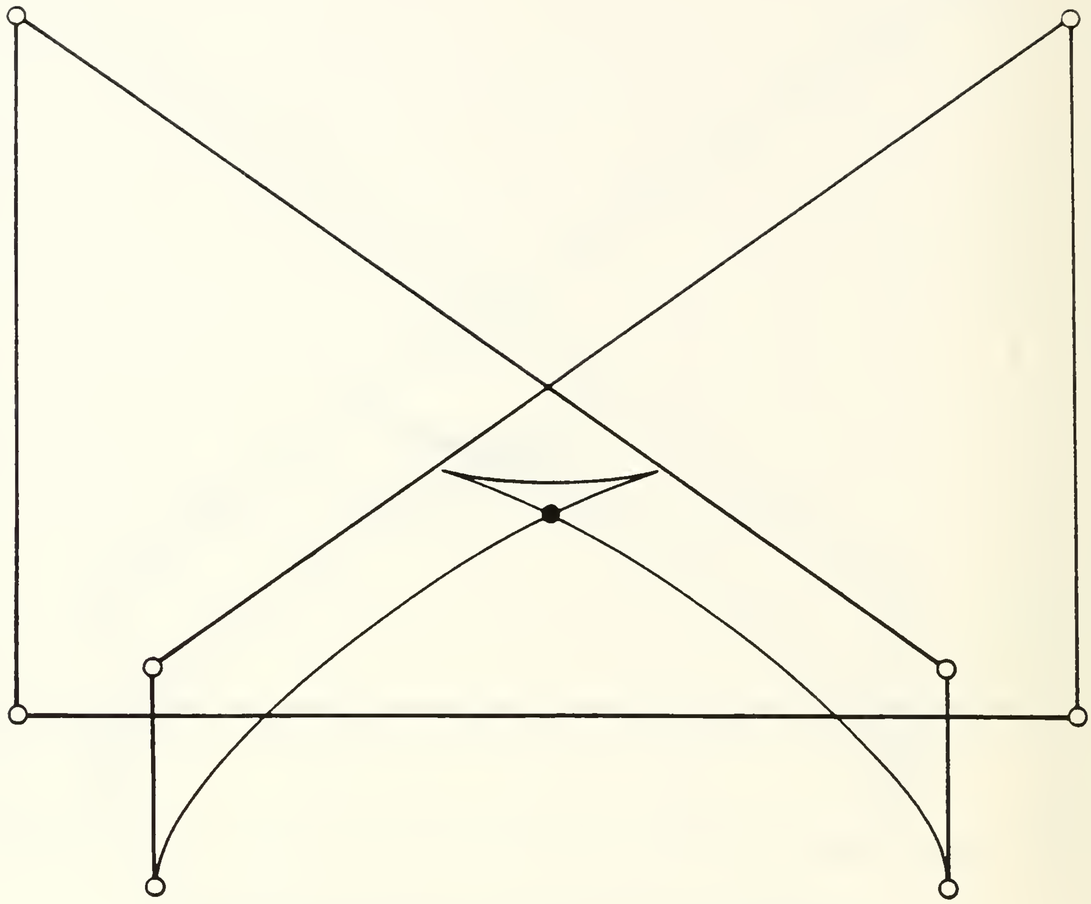
$P.X$	$P.Y$	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.23977	3.40057	0.24154	0.47076
-0.23977	3.40057	0.52924	0.75846
0.00000	3.28345	0.22167	0.77833

parameter values and x-y-coordinates of the intersection points

bx	by
-3.0	0.0
-3.0	2.0
4.0	8.0
4.0	1.0
-4.0	1.0
-4.0	8.0
3.0	2.0
3.0	0.0

Bezier points of $\mathbf{b}(t)$

Example 11



PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.00000	3.38682	0.22167	0.77833

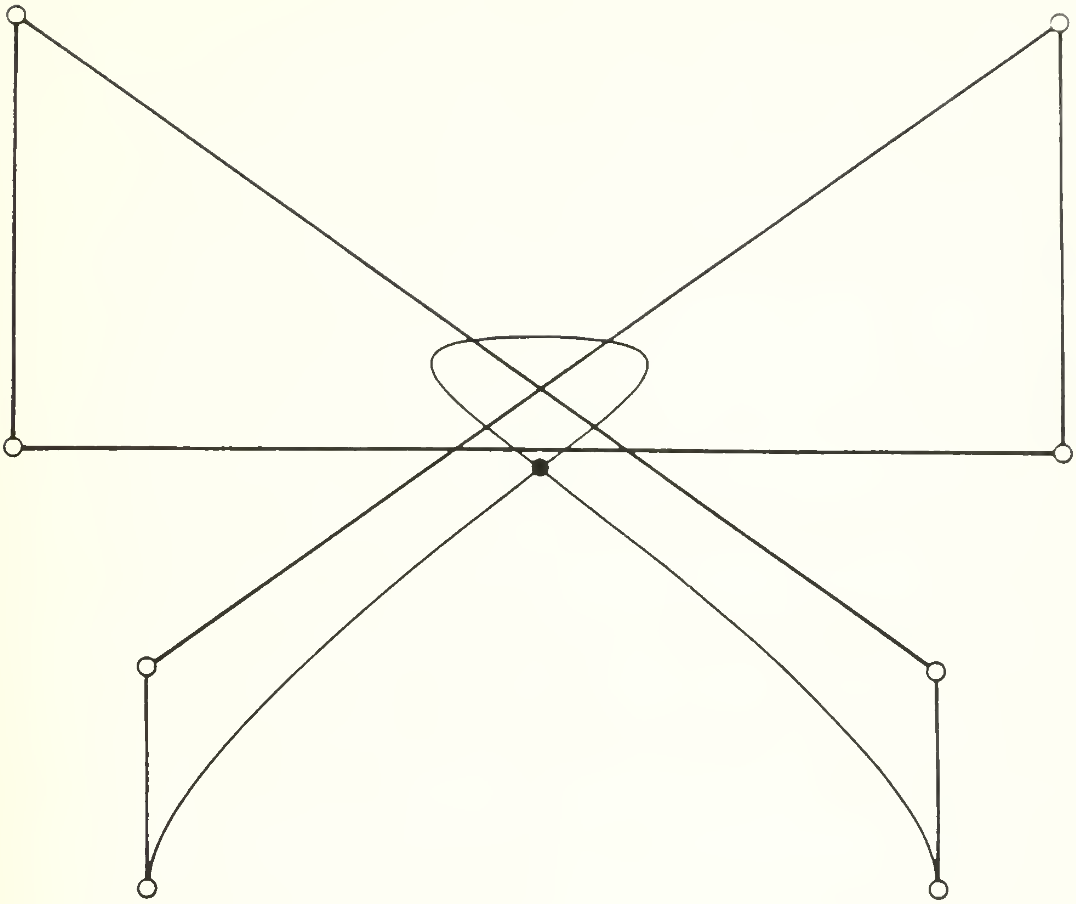
parameter values and x-y-coordinates of the intersection point

bx	by
-3.0	0.0
-3.0	2.0
4.0	8.0
4.0	by_3
-4.0	by_4
-4.0	8.0
3.0	2.0
3.0	0.0

Bezier points of $\mathbf{b}(t)$

$(by_3 = by_4 = 1.575039)$

Example 12



PX	PY	$t_1(\mathbf{P})$	$t_2(\mathbf{P})$
0.00000	3.82273	0.22167	0.77833

parameter values and x-y-coordinates of the intersection point

bx	by
-3.0	0.0
-3.0	2.0
4.0	8.0
4.0	4.0
-4.0	4.0
-4.0	8.0
3.0	2.0
3.0	0.0

Bezier points of $\mathbf{b}(t)$

IV. Self-Intersections of Spline Curves

Normally we are not really interested in single Bezier curve segments but in B-spline resp. Bezier spline curves consisting of several curve segments. Because a B-spline curve can be redefined in a Bezier form by using the Oslo algorithm adding multiple knots in one pass [*Cohen et al. '80*] or by using the computationally more efficient Boehm algorithm adding the multiple knots one by one [*Boehm '80, '82*], self-intersections of B-spline and of Bezier spline curves can be calculated using the algorithms of part II and III.

The segments $\mathbf{B}_K(u)$ of the Bezier representation of the spline curve might be given by

$$\mathbf{B}_K(u) = \sum_{k=0}^{m} \mathbf{b}_{mK+k} B_k^m(u)$$

where

$$\lambda = (1-u)\lambda_K + u\lambda_{K+1}, \quad 0 \le u \le 1, \quad K = 0, \dots, M$$

i.e. the spline curve is defined with respect to a partition of the domain space by *'knots'*

$$\lambda_0 < \lambda_1 < \dots \lambda_M .$$

The self-intersection points of a spline curve can be calculated by doing the curve-curve intersection algorithm for all pairs of segments $\mathbf{B}_K$ and $\mathbf{B}_{\overline{K}}$ with $K \neq \overline{K}$ and by doing the curve self-intersection algorithm for all segments $\mathbf{B}_K$. While the algorithms of part II and III calculate the parameter values of the self-intersection points with respect to the local coordinate domain [0,1] we also know - because of the linear relation between λ and u - the λ parameter values of the self-intersection points.

Remark

This study was done as a pre-study for the creating of a surface self-intersection algorithm for parameterized surfaces in Bezier representation. The surface algorithm is described in the paper **Self-Intersections of Parametric Surfaces** , Technical Report # NPS-53-88-002, Naval Postgraduate School, Monterey (1988).

Acknowledgment

This paper was written while the author was supported by an one year award of the *National Research Council, Washington, DC* visiting during this time at the *Naval Postgraduate School, Monterey, California*.
The author would like to thank *Professor Richard Franke* whose effort made this visit possible. And also like to thank *Professor Richard Franke* for all his help during the time in *Montery*.

References

Arnold, R: Quadratische und Kubische Offset-Bezierkurven. Diss. Dortmund (1986)

Bezier, P: Numerical control - mathematics and applications. J. Wiley & Sons, Chichester (1972)

Bezier, P: The mathematical basis of the UNISURF CAD system. Butterworths & Co Ltd (1986)

Boehm, W: Inserting new knots into B-spline curves. Computer Aided Design Vol 12 No 4 (1980)

Boehm, W: On cubics, a survey. Computer Graphics and Image Processing Vol 19, 201-226 (1982)

Boehm, W, Farin, G, Kahmann, J: A survey of curve and surface methods in CAGD. Computer Aided Geometric Design Vol 1 No 1 (1984)

Chandru, V, Kochar, B S: Analytic Techniques for Geometric Intersection Problems. in **Farin, G** (ed.) Geometric Modeling: Algorithms and New Trends. 305-318 SIAM (1987)

Cohen, E, Lyche, T, Riesenfeld, R F: Discrete B-Splines and subdivision techniques in computer-aided geometric design and computer graphics. Computer Graphics and Image Processing Vol 14 (1980)

Cohen, E, Schumaker, L L: Rates of convergence of control polygons. Computer Aided Geometric Design Vol 2 No 1-3 (1985)

Dahmen, W: Subdivision algorithms converge quadratically. Jour of Computational and Applied Mathematics 16, 145-158 (1986)

Dannenberg, L, Nowacki, H. Approximate conversion of surface representations with polynomial bases. Computer Aided Geometric Design Vol 2 No 1-3 (1985)

Farin, G: Algorithms for rational Bezier curves. Computer Aided Design Vol 15 No 2 (1983)

Farouki, R T: Exact offset procedures for simple solids. Computer Aided Geometric Design Vol 2 No 4 (1985)

Farouki, R T, Rajan, V T: On the numerical condition of polynomials in Bernstein form. Computer Aided Geometric Design Vol 4 No 3 (1987)

Faux, I D, Pratt, M J: Computational geometry for design and manufacture. Ellis Horwood, Chichester (1983)

Forrest, A R: Interactive interpolation and approximation by Bezier polynomials. Computer Jour. Vol 15, 71-79 (1972)

Goldman, R N: The method of resolvents: A method for the implicitization, inversion, and intersection of non-planar, parametric, rational cubic curves. Computer Aided Geometric Design Vol 2 No 4 (1985)

Goldman, R N, Sederberg, W T, Anderson, D C: Vector elimination: A technique for the implicization, inversion and intersection of planar parametric polynomial curves. Computer Aided Geometric Design Vol 1 No 4 (1984)

Gordon, W, Riesenfeld, F R: Bernstein-Bezier Methods for the Computer Aided Design of Free-Form Curves and Surfaces. Jour. of Assoc. for Computing Machinery Vol 21, 293-310 (1974)

Griffiths, J G: A data structure for the elimination of hidden surfaces by patch subdivision. Computer Aided Design Vol 7 No 3 (1975)

Hilton, H: Plane Algebraic Curves. Oxford University Press, London: Humphrey Milford (1932)

Hoschek, J: Offset curves in the plane. Computer Aided Design Vol 17 No 2 (1985)

Hoschek, J: Approximate conversion of spline curves. Computer Aided Geometric Design Vol 3 No 4 (1986)

Hoschek, J: Spline Approximation of Offset Curves. Computer Aided Geometric Design Vol 4 (1987)

Klass, R: An offset spline approximation for plane cubic splines. Computer Aided Design Vol 15 No 5 (1983)

Koparkar, P A, Mudur, S P: A new class of algorithms for the processing of parametric curves. Computer Aided Design Vol 15, 41-45 (1983)

Lane, J M, Riesenfeld, R F: A theoretical development for the computer generation of piecewise polynomial surfaces. IEEE Transaction on Pattern Analysis and Machine Intelligence PAMI 2 (1980)

Mortenson, M E: Geometric Modeling. John Wiley, New York (1985)

Piegl, L: A Geometric Investigation of the Rational Bezier Scheme of Computer Aided Design. Computers in Industry 7, 401-410 (1986)

Salmon, G: A Treatise on the HIGHER PLANE CURVES. Photographic reprint of the Third Edition of 1879, New York, G. E. Stechert & C0. (1934)

Sederberg, T W: Planar piecewise algebraic curves. Computer Aided Geometric Design Vol 1 No 3 (1984)

Sederberg, T W, Anderson, D C, Goldman, R N: Implicitization, Inversion and Intersection of Planar Rational Cubic Curves. Computer Vision, Graphics, and Image Processing 31, 89-102 (1985)

Sederberg, T W, Parry, S R: Comparison of three curve intersection algorithms. Computer Aided Design Vol 18 (1986)

Tiller, W: Rational B-Splines for Curve and Surface Representations. IEEE Computer Graphics & Applications Vol 3 No 6 (1983)

Tiller, W, Hanson, E G: Offsets of Two-Dimensional Profiles. IEEE Computer Graphics & Applications Vol 4 No 9 (1984)

Walker, R J: Algebraic Curves. Princeton University Press, Princeton, New Jersy (1950)

Yang, M C K, Kim, C-K, Cheng, K-Y, Yang, C-C, Liu, S S: Automatic Curve Fitting with Quadratic B-Spline Functions and its Applications to Computer-Assisted Animation. Computer Vision, Graphics, and Image Processing 33, 346-363 (1986)

Initial Distribution List

Associateship Programms JH608
National Research Council
2101 Constitution Avenue, NW
Washington, DC 20418 USA

Director (2)
Defense Tech. Information Center
Cameron Station
Alexandria, VA 22314 USA

Director of Research Admin.
Code 012
Naval Postgraduate School
Monterey, CA 03043 USA

Peter Alfeld
Department of Mathematics
University of Utha
Salt Lake City, UT 84112 USA

Randolf Arnold
Fachbereich Mathematik
Universitat Dortmund
4600 Dortmund FRG

Robert E. Barnhill
Department of Computer Science
Arizona State University
Tempe, AZ 85287 USA

Brian Barsky
Dept. of Electrical Engineering
and Computer Engineering
University of California
Berkeley, CA 94720 USA

Wolfgang Boehm
Institut fuer Math. Stochastik
Technische Universitat Braunschweig
3300 Braunschweig FRG

Genz-Zhe Chang
Department of Mathematics
University of Science and Technology
Hefei, Anhui
The People's Republic of China

Carl de Boor
Mathematics Research Center
610 Walnut Street
Madison, WI 53705 USA

Wendelin Degen
Mathematisches Institut B
Universitat Stuttgart
7000 Stuttgart 80 FRG

Library (2)
Naval Postgraduate School
Monterey, CA 93943 USA

Department of Mathematics
Naval Postgraduate School
Monterey, CA 93943 USA

Landesbibliothek Darmstadt (2)
6100 Darmstadt FRG

Fachbereichsbibliothek (2)
Fachbereich Mathematik
Technische Hochschule Darmstadt
6100 Darmstadt FRG

Franz Jurgen Delvos
Fachbereich Mathematik
Universitat Siegen
5900 Siegen FRG

Anthony D. DeRose
Department of Computer Science, FR-35
Universitaet of Washington
Seattle, WA 98195 USA

Gerald Farin
Department of Computer Science
Arizona State University
Tempe, AZ 85287 USA

Thomas A. Foley
Department of Computer Science
Arizona State University
Tempe, AZ 85287 USA

Rida T. Farouki
Manufacturing Research Department
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598 USA

Herbert Frank
Fachbereich Mathematik
Universitaet Dortmund
4600 Dortmund FRG

Richard Franke (5)
Department of Mathematics
Naval Postgraduate School
Monterey, CA 93943 USA

Harold M. Fredricksen
Chairman, Dept. of Mathematics
Naval Postgraduate School
Monterey, CA 93943 USA

Frederick N. Fritsch
Numerical Mathematics Group
Lawrence Livermore National Laboratory
Livermore, CA 94550 USA

Gerhard Geise
Technische Universitat Dresden
Sektion Mathematik
8027 Dresden DDR

Oswald Giering
Mathematisches Institut
Technische Universitat Munchen
8000 Munchen 2 FRG

Ronald Goldman
Control Data Corporation
4201 N Lexington Ave, AHS 251
Arden Hills, MN 55126 USA

William J. Gordon
Center for Scientific
Computation & Interactive Graphics
Drexel University
Philadelphia, PA 19104 USA

John A. Gregory
Dept. of Mathematics & Statistics
Bonnel University
Uxbridge UB8 3PH England

Hans Hagen
Institut fur Graphische Datenverarbeitung
und Computergeometrie
Universitat Kaiserslautern
6750 Kaiserslautern FRG

Josef Hoschek (10)
Fachbereich Mathematik, AG3
Technische Hochschule Darmstadt
6100 Darmstadt FRG

Reinhold Klass
Daimler-Benz AG
Abt. AIDK
7032 Sindelfingen FRG

Dieter Lasser (15)
Fachbereich Mathematik, AG3
Technische Hochschule Darmstadt
6100 Darmstadt FRG

Norbert Luscher
Angewandte Geometrie und
Geometrische Datenverarbeitung
Technische Universitat Braunschweig
3300 Braunschweig FRG

Harry W. McLauchlan
Dept. of Mathematical Sciences
Rensselaer Polytech. Inst.
Troy, NY 12181 USA

Gregory M Nielson
Department of Computer Science
Arizona State University
Tempe, AZ 85287 USA

Horst Nowacki
Institut fuer Schiffstechnik
Technische Universitat Berlin
Salzufer 17-19
1000 Berlin FRG

Malcom Sabin
Oakington
Cambridge CB4 5BA England

Paul Sablonniere
UER IEEA Informatique
Universite de Lille L
59655 Villeneuve d'Ascq. Cedex France

Ramon Sarraga
General Motors Research Laboratories
Warren,Michigan 48090-9055 USA

Thomas W. Sederberg
Department of Civil Engineering
Brigham Young University 368B CB
Provo, Utah 84602 USA

Larry Schumaker
Center for Approximation Theory
Texas A & M Unive
College Station, Tex A

Wolfgang Schwarz
Jakob-Stefan-Str. 12
6500 Mainz FRG

W. Strasser
Heinrich Fabri Institut
Universitat Tubingen
7400 Tubingen FRG

Andrew J. Worsey
Dept. of Mathematical Sciences
University of North Carolina
Wilmington, NC 28403-3297 USA