On the Capture and Representation of Fonts

Fiaz Hussain, B.Sc., M.Sc.

A thesis submitted for
the degree
of

Doctor of Philosophy

Department of Computer Science, Brunel University, England.
December 1991

Abstract

The commercial need to capture, process and represent the shape and form of an outline has lead to the development of a number of spline routines. These use a mathematical curve format that approximates the contours of a given shape. The modelled outline lends itself to be used on, and for, a variety of purposes. These include graphic screens, laser printers and numerically controlled machines. The latter can be employed for cutting foil, metal, plastic and stone.

One of the most widely used software design packages has been the IKARUS system. This, developed by URW of Hamburg (Germany), employs a number of mathematical descriptions that facilitate the process of both modelling and representation of font characters. It uses a variety of curve formats, including Bezier cubics, general conics and parabolics.

The work reported in this dissertation focuses on developing improved techniques, primarily, for the IKARUS system. This includes two algorithms which allow a Bezier cubic description, two for a general conic representation and, yet another, two for the parabolic case. In addition, a number of algorithms are presented which promote conversions between these mathematical forms; for example, Bezier cubics to a general conic form. Furthermore, algorithms are developed to assist the process of rasterising both cubic and quadratic arcs.

Acknowledgements

I would like to thank my supervisor, Professor Michael L V Pitteway, for providing an exciting research environment throughout my time at Brunel. Through him, opportunities have been presented to me that might otherwise have been missed.

My thanks goes to both Professor Pitteway and Professor Wright for providing financial assistance through the Science and Education Research Council (SERC), for the academic year 1988.

I am indebted to Dr Peter Karow (of URW, Hamburg, Germany) for sponsoring most of the work reported in this manuscript. Discussions with him and his staff, noticeably with Dr Juergen Willrodt, have assisted in giving me a better understanding of the commercial factors relating to the research field.

My appreciation goes out to the personnel belonging to the Support Team. They were always ready to assist (and remove!) any problems that concerned the network. Thanks, also, to the various members of staff, working within the Computer Science Department, for being supportive and helpful. Special thanks to my fellow researchers, who made my stay at Brunel socially acceptable.

Finally, I am grateful to my family for giving both financial and moral support throughout the course of this work.

Declaration

No part of the material presented in this thesis has been submitted in support of an application for another degree at Brunel or any other institution.

Contents

1.0 Introduction

The quest for an accurate and aesthetically pleasing representation of outlines of shape has traditionally been undertaken by proficient artisans. They produced desired results by using a skilled hand, having an eye for fine detail and an appreciation of the type and form of the shape itself. This process of drawing outlines found popularity in the field of typography, where an exact production (and reproduction) of contours of characters is required. The task for the typographer was not just to draw a given outline, but also to capture the distinct features of a particular font type. As thousands of different font types exist (each consisting of over a hundred characters), the manual capture of character outlines becomes rather cumbersome and time-consuming.

With the advent of computers, attempts have been made to automate the design process. This has resulted in software packages under the headings of computer-aided-design (CAD), manufacture (CAM) and geometric-design (CAGD) being developed. The modern typographer uses a particular design package to capture the font outlines which meet some predefined specifications. The aim of these conditions is to constrain the design process to producing an output which exhibits the unique features of the given font.

The goal of a design system, therefore, is to efficiently model and represent a desired shape in a digital form. This form facilitates the captured outlines to be processed using fast processors, and their output to be displayed on graphic devices such as screen displays and laser printers. The benefits of the digital form are also extended to cater for numerically controlled machines. These are employed to produce outlines (either by drawing, cutting or engraving) on paper, metal, plastic, stone or wood.

In view of the modern commercial demands for a fast and accurate means of modelling a given outline, the design system is required to employ an effective approach for representing contours of shape. This, in practice, is best achieved

through quantifying the given outline in terms of a mathematical description. The attraction of such an approach, when compared to the manual construction, is its speed of capture and its resulting portability within, and between, design systems. The described outline is also in a form which allows it to be rotated, scaled and translated through applying a simple and appropriate transformation matrix.

Mathematical descriptions come in various forms: The simplest uses straight lines from point to point around the outline. Each line requiring two integer parameters, a new end point relative to the current mesh point. A more sophisticated alternative allows for circular arcs, with the radius forming an additional parameter. The extra storage required for each arc should be compensated for by requiring fewer segments for the same accuracy in rendering a given shape outline. A fourth parameter is needed for a mathematical description involving parabolic arcs. The additional point forms the control element of a three-point Bezier curve which defines the starting and finishing directions for each arc [BEZI 72, PAVL 82]. To mathematically model outlines using the general conic (quadratic) sections, a fifth parameter is necessary; namely the sharpness value [PRAT 85]. For a Bezier cubic description [FORR 71, BEZI 72], six parameters are required, ie two control points and the endpoint with respect to the start point.

The application of a particular mathematical description is normally oriented at the type of data fit required. Using line segments to model a curve outline, for example, would require a considerable number of segments to meet some design specification. Even then, it is not certain whether the resulting representation would be acceptable for aesthetic reasons. Clearly, in such circumstances a capturing process that employs a curve description is more suitable. The question then is whether to use a quadratic or a cubic, or even a higher ordered modeller. This consideration is usually limited to deciding how two adjacent arcs will join together. In other words, what form of continuity will reside amongst the resulting description?

Increasing the order of the mathematical description (moving from a quadratic to a cubic form, for example) has the effect of offering a higher degree of continuity. Employing such continuity in the capturing process results in outlines which appear "smoother" at the joints. Although this is as desired, it is gained through increasing the amount of computation necessary for a description, highlighting the fact that when it comes to choosing a mathematical description, the computational efficiency of the capturing process is an important factor. For reasons of simplicity, therefore, curve descriptions above cubic order are normally not used to model outlines of shape.

This dissertation addresses the demand for mathematical curve descriptions that embody both efficiency and accuracy. Efficiency is generally measured in terms of the capturing speed, and the number of arcs used, for a complete description. Accuracy is quantified in terms of a specified tolerance that the modelling process needs to work within. Together, these two criteria allow the performance of a particular description to be assessed.

The work reported here concentrates on developing techniques that are applicable to the two most widely accepted descriptions, the Bezier cubic and the general conic. Both representations have unique features, and embody capturing capabilities that are explored and developed. The techniques presented in this thesis offer the designer the ability to model a given outline, make conversions between the two descriptions, and facilitate a rasterised output on a digital display. Although most of these developments are constructed to assist the typographic designer using the IKARUS system (see section 2.4.3), the concepts, as well as the approaches, can be applied on an equal merit to representing outlines of any shape.

In the light of some commercial requirements, capturing techniques are also developed for the parabolic case. Although this is a curve description which belongs to the conic family (as well as the fact that an exact Bezier cubic representation for it exists), some design systems (including IKARUS) have been attracted to it because it requires one less parameter than a general conic for a

solution. This does not, however, make the task of capturing any easier, and leads to a capturing process which takes longer to yield a solution than does the general conic (see chapter six).

Apart from chapter two, which gives an introduction to some of the concepts and techniques that has stimulated the development of mathematical descriptions for modelling outlines of shape, Fig 1.1 gives a graphical illustration of the work reported in each chapter of this dissertation. This can be summarised as follows:

Chapter three looks at the Bezier cubic curve. It presents some of the characteristics and features which have made this description popular with designers. Two approaches for capturing a given set of data points, of the type described in section 3.3, are then developed. The first technique uses the parametric form to represent the given data. This leads to an iterative approach which is costly in terms of time. An implicit form for the Bezier cubic is then developed, and an instability in the representation is highlighted. The process of displaying the described outline onto a screen of some resolution is then considered. Two algorithms for this purpose are developed. The first translates the curved outline into line segments, using a novel way of evaluating how many lines to employ. The second approach tracks the curved outline, using the nearest integer mesh point at each stage. Although returning a superior digitised output than the first approach, it is shown that in certain cases the algorithm fails to track the desired curve outline. Reasons for this occurrence and its possible solutions are given in section 3.6.3.

Chapter four concentrates on developing techniques for the curves belonging to the conic family. After a detailed introduction to the family, two algorithms for modelling a given set of data points, of the form outlined in section 4.3, are presented. The first technique has the feature of returning an outline which maintains tangent continuity (see section 2.2.2 for its definition) between joining arcs. The second approach works within a more relaxed environment to yield a conic description which adjusts the control point so that the resulting point-to-curve deviation is a minimum. The performance of both

algorithms is then assessed through their modelling of a given character outline. For the purpose of digitising the quadratic outlines, an algorithm is presented which follows the given curve and returns the closest integer pen position at each stage. The approach is made robust to cater for all conic arcs lying within a quadrant.

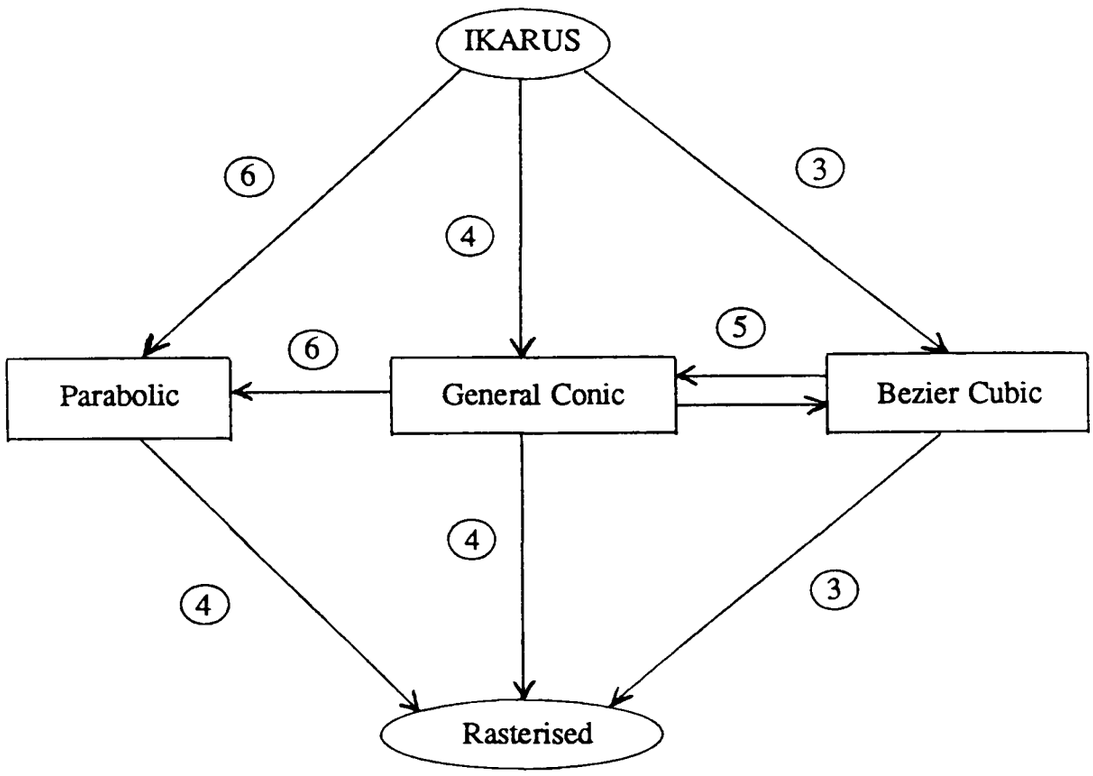


Fig 1.1 Gives an illustration of the various algorithms presented in this dissertation, in terms of their respective chapters.

Chapter five describes algorithms for conversion between the Bezier cubic and the general conic. After a detailed comparison of the two representations, two approaches for translating cubic curves into quadratic arcs are presented. Both employ an integral approach for gaining a solution. The first method returns a gradient continuous conversion, whilst the second method gives a conic approximation that deviates the least from the supplied Bezier cubic

curve. Their conversion capabilities are assessed in section 5.3.5. Two algorithms are then given which translate a conic arc into a corresponding Bezier cubic form. The first approach is based on the sharpness value, whilst the alternative method gains a conversion through matching curvature at the end (joining) points. Performance of both algorithms is analysed in section 5.4.3.

Chapter six evaluates the modelling capabilities of the parabolic curve. Some of its attractions are highlighted through a graphical illustration that shows how a manual construction for the curve can be realised. Two algorithms for capturing the given set of data points, of the type discussed in section 6.3, are presented. The first approach attempts to gain a "direct" description. This leads to a recursive method being applied for a solution. The second approach uses the capturing techniques of the general conic description as its basis. This is shown, in section 6.4.3, to return a better rate of capture than the first approach a result that is somewhat surprising as the parabolic arc uses one less parameter for its description than the general conic.

The seventh chapter gives an outline of the various algorithms presented in this thesis. Although the performance of each algorithm is assessed within its relevant chapter, the discussion focuses on highlighting some of the important results attained, and how these may effect their employment in a particular design system. In addition, possible areas of further research are emphasised.

2.0 Elements of Outline Capture

2.1 Introduction

The problem of finding a mathematical curve description, which models a desired shape, occurs in various situations and, its application can be found in a number of fields including design and manufacture, graphics, image processing, pattern recognition and typography. One of the earliest uses of mathematical modelling, for the purposes of manufacture, occurred in representing the surfaces of a car body. This was done by specifying the surfaces as a set of discrete points that met some design and aesthetic conditions. Other applications have included the representation of experimental data, either for display purposes or for automatic pattern recognition. In addition, the mathematical description of the contour of an object may give indications about the class to which the object belongs. This is a frequent requirement in the field of pattern recognition. In the field of typography, the representation of font outlines needs to be both accurate and aesthetically acceptable. Although different fields and applications use mathematical modelling of outlines for different purposes, the solution to finding an acceptable representation in each case does depend on having an effective method for capturing and characterising the given curves and surfaces.

This chapter looks at the fundamental concepts and techniques that have stimulated some of the work in the area of mathematical description of outlines. It introduces the spline, its historical importance and its modern practical relevance. The two forms (through interpolation and approximation) of modelling a set of data points are discussed. The chapter also looks at design systems, and introduces two internationally accepted packages; namely, the IKARUS and METAFONT systems. Finally, some of the characteristics and difficulties of displaying outlines on screen displays are highlighted. In short, this chapter gives a general introduction to capture methods and techniques, some of which are discussed further in the subsequent chapters.

2.2 Introduction to Splines

Probably the most important aspect of mathematical modelling, most definitely the main component in capturing techniques, is the spline. With the application of splines, outline and contours of objects can be mathematically modelled without too much difficulty. So, what exactly is a spline and how does it assist the designer in capturing outlines?

The term spline originated well before the days of computer graphics and the commercial use of numerical machines, when an outline was constructed by means of some weights and a flexible material. The weights (also called ducks) were located to shape the flexible material (such as wood) to yield a desired outline [BEZI 72, ROGE 76]. The process is illustrated in Fig 2.1, where a thin elastic material, called the spline, is bent around some weights. The weights were designed to hold the spline in position, while allowing the spline to turn and bend as necessary.

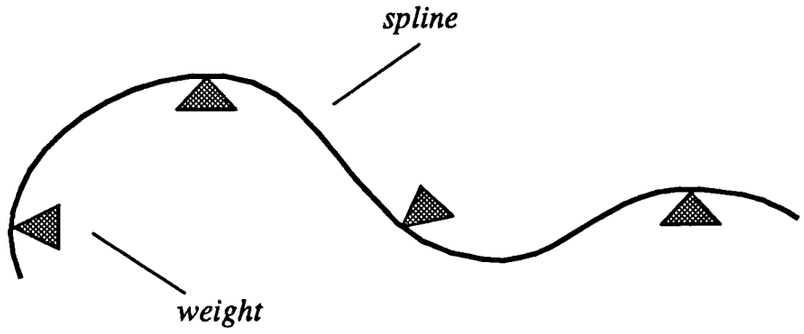


Fig 2.1 Illustrates the construction of a curved outline by using a spline and some weights.

Traditionally, the splines were used in the design and construction of contours for large objects such as ship hulls and aircraft fuselages. This manual process of drafting outlines was often performed in grand attics or lofts (hence the term lofting frequently associated with splines) [LIMI 44, CHAS 78]. As these splines (sometimes called the physical or natural splines) are recognised and have been

accepted as reliable means of representing contours of objects, attempts have been made to develop mathematically analogous splines.

The physical spline has an interesting and desirable property of exhibiting minimum total internal strain (bending) energy. This means that along the entire length of the spline, the integral of the square of the curvature has a minimum value amongst all the possible curves passing through the weights. Various authors have discussed this aspect, including: [BEZI 72, FAUX 79, ROGE 76, YAMA 88].

The mathematical spline therefore requires to encompass the physical spline's property of yielding minimum bending energy so that it returns the "smoothest" curve which passes through the weights. It turns out that the mathematical spline which best resembles the properties of the physical spline is a series of cubic curves joined together in a piecewise manner at the weights (more commonly known as the 'knot' points). It is found that these curves are continuous in position, slope and curvature (see section 2.2.2) at the knot points.

Although the mathematical spline, when compared to its physical counterpart, gives adequate approximations in most cases, it needs to include a procedure for representing difficult cases such as large or infinite slopes [BEZI 72]. If the slopes are large, a situation which is quite common in practice, two solutions could be employed: The first, consists of introducing local axes for each arc in the spline. This is normally done in a manner that leaves the new coordinate system having the x axis along and the y axis perpendicular to the chord of the arc. An illustration of this approach is given by Faux and Pratt [Faux 79]. The second method is to use a parametric spline, where an additional parameter is used to yield the points on the arc. This method has the advantage of being axis-independent. Parametric curves are covered in chapter three.

2.2.1 Control of Shape

The physical spline, as explained in section 2.2, is constrained to go through the fixed weight positions. Changing the position of one of the weights results in a corresponding change in the shape of the physical spline. This way, the weights are exerting some form of control over the shape, be it more of a global control than a local control.

The shape of the mathematical spline, in comparison, is determined by the positions of the knot points. The location of these points sets the shape of the spline. Although, this approach works well in most cases, there are occasions where a loop appears in the spline even though there is no such occurrence in the original data points [FAUX 79, YAMA 88].

Schweikert proposed a method to improve the control features of the mathematical spline [SCHW 66]. He introduced a new parameter that corresponded to tension (hence the term 'spline in tension'). The method was based on the concept of "pulling out" unwanted loops and inflection points by increasing tension (see Brodlie [BROD 80] for an illustration). This then resulted in mathematical splines being more in-line with the given outline. The computational aspects of these type of curves has been looked at by Cline [CLIN 74]. A generalised approach is given by Pilcher [PILC 74]. Nielson [NIEL 74] and Barsky [BARS 84] present a polynomial related solution to the spline in tension concept.

Bezier introduced the concept of defining a given (curve) outline within a single polygon [BEZI 72]. The modelled outline can be taken as being the defining polygon whose corners have been smoothed out. Bezier's polygonal definition assisted the designer in determining and controlling the overall outline of the shape. Much of Bezier's work was put to practical use in Renault's car body design system called UNISURF [BEZI 71, BEZI 74, BEZI 86].

Splines defined in terms of the Bezier polygonal set-up, provided a much more flexible way of controlling and changing the outline shape. The shape could be represented by a single spline or by a number of splines, connected together in a piecewise manner (see section 2.2.2). The designer has the option of making changes that only effects a localised section. By using a trial-and-error approach, a mathematical model of the given outline could then be converged to (see section 2.2.3).

2.2.2 Continuity of Joining Arcs

The outlines of some objects are normally too complex to merit a single spline representation. These outlines are then modelled by a series of arcs (splines) patched together, in a piecewise manner, to resemble the given object shapes. Each arc is defined within two knot points, one at each end.

The way two arcs join together depends, primarily, on the type and order of the spline being used. If, for example, cubic splines were employed, then there is the possibility of making the two arcs continuous in position, slope and curvature (as is the case for the mathematical spline in section 2.2). Fig 2.2 illustrates the three possibilities. By position, the arcs are guaranteed to touch. The arcs are said to possess tangential continuity if their slopes are the same at the joining knot. For the arcs to exhibit curvature continuity, the first arc needs to have the same curvature as the second arc at the joining knot.

Rogers and Adams [ROGE 76] state that: "In general the mathematical spline is a piecewise polynomial of degree K with continuity of derivatives of order K-1 at the common joints between segments. Thus the cubic spline has second-order continuity at the joints."

This implies that higher ordered splines return better continuity at the joining knots, resulting in a more aesthetically pleasing outlines. These splines, however, are found not to be commercially viable as they are computationally expensive and tend to introduce unwanted oscillations and wiggles between knot points

[ROGE 76, BART 87]. In addition, recent developments have meant that quadratic splines can (in addition to cubic splines) be fashioned to yield second-order (curvature) continuity [PRAT 85]; this will be discussed further in chapters four and five.

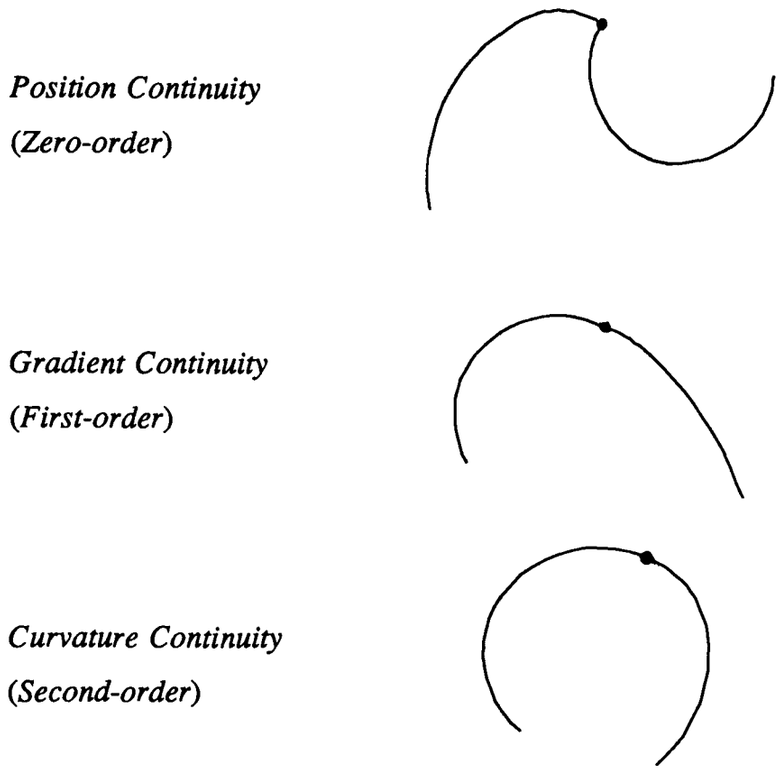


Fig 2.2 Shows three ways of connecting two arcs.

2.2.3 Subdividing Splines

In the process of interactive design, a frequent requirement is to edit and modify parts of an outline such that the overall drawing does not change significantly. If the outline shape is described by a spline, then it is necessary to make small adjustments to this in order to gain a desired representation. The definition of splines according to Bezier tend to be ideally suited for this purpose: An outline described by a Bezier spline, of a specified degree, can in fact be represented exactly by a series of splines having the same degree (see Farin [FARI 90], for

example). This process is called *subdivision*, and is performed by splitting the original outline at a desired point and then representing the outline by two, smaller, Bezier splines of similar degree: Fig 2.3 depicts a situation where the defining (Bezier curve) polygon is subdivided into two polygons (polygon A and B). This subdivision process does not change the shape outline. The important point here is that the designer is free to change the shape of polygon A without necessary effecting the shape described by polygon B. The problem of shape control is, therefore, being localised. (The process of subdivision is further discussed and applied in section 3.6.1).

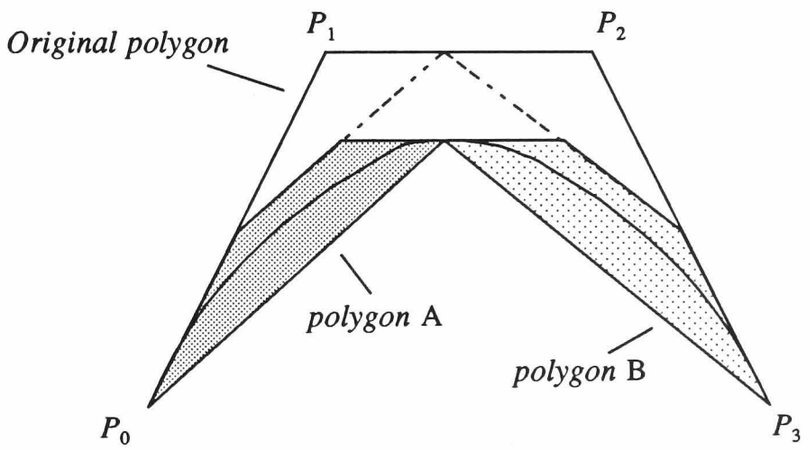


Fig 2.3 Highlights the fact that the subdivision process for a Bezier cubic results in two polygons of cubic order.

The process of spline subdivision is not just limited to the above mentioned application, but has been employed for a variety of tasks: The evaluation of intersection point(s), for example, of a high degree Bezier spline with a line or curve section is a common problem that occurs in computer graphics, especially in the area of hidden-line removal. An algorithm based on the subdivision technique could be employed for this purpose. A number of other applications, and approaches, can be found in [COHE 80].

2.2.4 Choosing the Spline Degree

Although it is common to work with a spline of a desired degree for interactive design purposes, it often happens that a design system requires a spline of a degree higher than the outline has been described with. This means that if the design system needs quartic splines, for example, then if the spline input is in the form of a (Bezier) cubic, a method for upgrading the degree of the input spline is required. This process is called *degree elevation*, and has been looked at by various authors including Boehm et al [BOEH 84], Farin [FARI 90], Forrest [FORR 72, FORR 90] and Yamaguchi [YAMA 88]. Fig 2.4 gives an illustration of the process for Bezier splines, and highlights how repeated degree elevations results in convergence of higher ordered polygons towards the modelled outline.

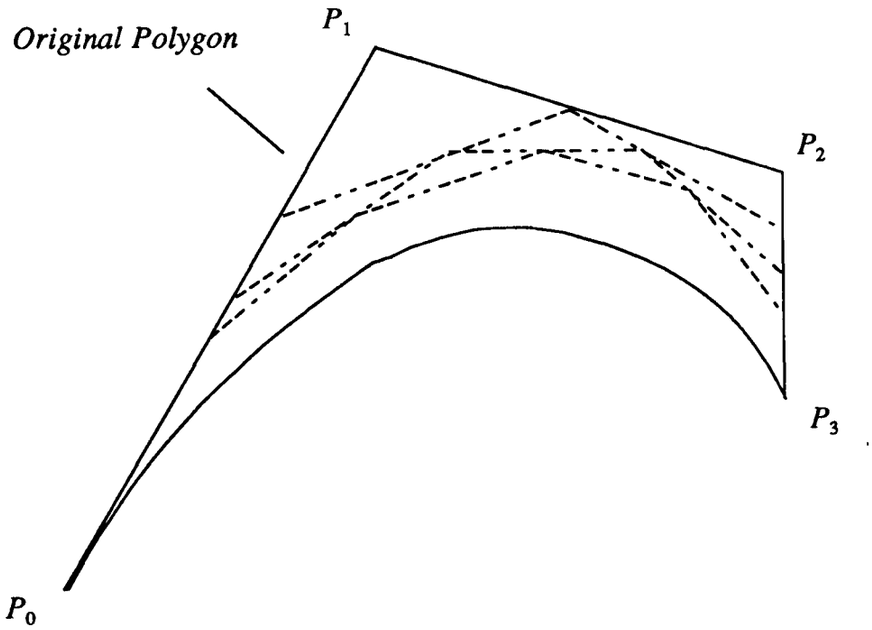


Fig 2.4 Gives an illustration of how the degree of the describing Bezier spline can be increased.

Degree elevation can be viewed as a process that introduces redundancy; an outline is modelled by more information than is actually necessary. The inverse of this process, called *degree reduction*, attempts to reduce (normally by one) the degree of a given spline. Although this is possible, any reduction in the degree

of a spline can only result in a set-up which approximates the original outline. This fact is highlighted by Watkins and Worsey [WATK 88] who present an algorithm that attempts to generate (n-1)th degree approximation to an nth degree Bezier spline. (Chapter five presents techniques for translating Bezier cubic splines to a general quadratic form and, also, for converting general quadratic descriptions to a Bezier cubic form).

2.3 Forms of Outline Capture

The physical spline, as described in section 2.2, uses the technique of interpolation to model the given set of data points (ie weights). The mathematical spline can be employed to represent an outline either by means of *interpolation* or by a process known as *approximation.* In the former case, the spline is constrained to pass through all the given set of data points, whilst in the latter case, the spline is allowed to represent each data point within a (pre-defined) least-squares distance criteria.

2.3.1 Capture by Interpolation

When considering how to model an outline mathematically by means of interpolation, the assumption is always made that the data points describing the outline are sufficiently accurate and have a smooth distribution that warrants the process of interpolation rather than approximation. Given a set of data points, it is not a difficult task for a proficient designer to draw an aesthetically pleasing outline that passes through all the data points. Interpolation attempts to develop mathematical tools that correlate well with the designer's experience and intuition so that the resulting representation is as acceptable as that drawn by hand.

This process of constructing a mathematical description that passes through a given set of data points is in fact not a new problem. J Lagrange (1736-1813) and C Hermite (1822-1901) pioneered and developed much of the mathematical tools that are employed in modern design systems to interpolate data.

Lagrange's approach was to use the x and y coordinates of each of the given set of data points to construct a polynomial. This polynomial then interpolated the data so that it returned the correct y value for the respected x data point. A full description and methodology of Lagrange's approach can be found in most mathematical and geometrical design literature including Faux and Pratt who give a full account with examples [Faux 79]; Yamaguchi gives a more mathematical description, including the effect of having the data points equidistant [YAMA 88].

Although Lagrange's method is intuitively and computationally simple to implement, it does have one major drawback. It tends to oscillate. Even for quite reasonable data points, this method generates wild wiggles that are not inherent in the data. This predicament is not due to numerical effects, but is inherent in the construction process for the interpolating polynomial, a problem that is discussed by Farin [FARI 90].

Mathematical interpolation is not just restricted to interpolating a given set of data points; it can include other information such as derivative data. This leads to an interpolation scheme that is considered to be more useful than Lagrange's method, namely Hermite interpolation. Although high-ordered derivatives can be employed, its construction is normally limited to the cubic case where data points are specified in terms of position and their first derivatives (tangents). The implementation of such a system is discussed by both Farin [FARI 90] and Yamaguchi [YAMA 88]. An interesting property of this method is that the shape of the interpolating cubic Hermite curve depends not only on the position and direction of the tangent vectors at these points, but also on the magnitudes of the tangent vectors.

One consequence of using Hermite interpolation is that it is not invariant under affine domain transformations. This will mean that the shape of an interpolating curve will change after it is transformed. In order to maintain the same curve as before the transformation, the defining tangent vectors (that is, their lengths) will

need to be adjusted [FARI 90]. This situation is rather unpleasant and thus makes this approach less attractive.

Lagrange and Hermite polynomials provide much of the basis of recent interpolating techniques. The use of tangential information is attractive as this can be used to develop piecewise interpolating splines with some degree of continuity. Ferguson, in the early sixties, introduced the idea of using cubic interpolating splines to model mathematically a given set of data points [FERG 64]. He employed these concepts in constructing a model for representing surfaces, a programme known as FMILL [FARI 90, FAUX 79, YAMA 88].

Methods that embody the use of piecewise interpolating splines of degree three (cubic) or lower have been proposed by various authors: Akima [AKIM 70] presents a technique for interpolating a given outline (in a piecewise manner) so that unwanted wiggles and loops are eliminated. He compares his technique with other methods, and extends his approach to cater for multi-valued functions. The use of Bezier splines has been considered by Piegl [PIEG 87], who utilises the rational Bezier form (see chapter three) to interpolate a given outline in a localised way. Harada and Nakamae also employ Bezier cubic splines in a localise sense [HARA 82]. They propose a method that produces an aesthetically pleasing representation. Pavlidis [PAVL 82] outlines the use of B-splines for this purpose. He illustrates, with an example, the effectiveness of utilising uniform B-splines for interpolating data. (B-splines form a topic of their own, and their introduction and development can be found in the literature: [BARS 83, BOOR 78, COX 72, GORD 74, LEE 85, RIES 73, SCHU 81]).

2.3.2 Capture by Approximation

When the given set of data points contain uncertainties, such as noise, we require a process that will return a "smooth" and "fair" representation of these points. By interpolating, the only certainty is that the resulting outline will pass through all the data points, leaving the desired shape unchecked. What is really required is a mathematical function that will pass close to the given data points, but not

necessary through them. This should then ensure that the overall outline (as described by the data points) is mostly preserved.

The process of approximation provides a means of finding a mathematical description that captures data points within some distance criteria. The use of a distance criterion gives the designer the flexibility of defining a modelling representation that meets some design specifications. The distance (curve to a given data point) is commonly measured along a coordinate or along a normal to the capturing curve. Chasen [CHAS 78] gives an introduction to the concept of mathematical approximation. He highlights some practical applications for such methods, and discusses the use of polynomials (linear, quadratic and higher ordered) for the purposes of least-squares data fitting.

The distance value can be used in a variety of ways to gain a "best-fitting" approximation to a given set of data points: It may be employed in an application that requires all the data points to be within a set tolerance. This approach is used by the IKARUS system (introduced in section 2.4.3). Alternatively, an acceptable approximation may be where the average of all the squared distance values is below some desired level. Pavlidis [PAVL 82] discusses these approximation procedures in relation to splines with fixed and variable knots. He uses the B-spline formation to further illustrate the concepts.

Splines used for the purposes of constructing an outline in a piecewise manner tend to employ both interpolating and approximating techniques. Each spline is constrained to interpolate the start and end points (of a given data set) and to approximate the remaining points based on some distance and continuity conditions. The designer's main concern is that the resulting spline representation "looks right"; whether it passes through all of them or just a few of them might be immaterial.

2.4 Design Considerations and Developments

This section develops further some of the concepts relating to splines. It gives the process of mathematical modelling of outlines in terms of the commercial needs and requirements. Through this a number of design considerations and factors are highlighted.

2.4.1 Fair and Smooth Curves

One of the primary tasks of a capturing system is to generate outlines that can be described as being "fair" and "smooth", and thus the modelled outline can be judged as "looking right". The problem is that, in industry, each designer tends to have his own method of defining and describing the terms fair and smooth. Forrest [FORR 68] elaborates on the various definitions: To some a fair curve or surface is one which does not contain any bumps or humps. Others judge the fairness of a curve by analysing the radius of curvature [FARI 89]. Draftsmen employed in the shipbuilding trade take a curve representation to be acceptable as long as it satisfies continuity requirements of the first and second derivative, with no inflection points that are not inherent in the given data.

Sometimes, as a result of digitising errors, the modelled outline is judged not to be fair or smooth: Data points describing an outline are commonly obtained by means of a digitising device (a tablet being the simplest), and an acceptable model is desired. In some cases, however, the digitised data points are found not be accurate, and thus this leads to unsatisfactory representations. The use of curvature plots first to remove the unhelpful data point and then to insert a new point has been discussed by Farin and Sapidis [FARI 89].

In the field of typography, the designer is interested in modelling outlines of font characters in a way that also captures the unique features of the font type. This means that the capturing system needs to preserve the aesthetic outlook of each outline (ie character). The IKARUS system produces mathematical descriptions of characters that are taken to be both fair and acceptable. It uses the criteria of approximating a given contour within a specified tolerance and that joining

curves have first order (tangent) continuity. This suggests that satisfactory character outlines can be gained without the use of curvature continuity. (Further discussion is left to the capturing algorithms presented in the subsequent chapters).

2.4.2 Production Systems

The employment of mathematical techniques, with computers, in designing commercial products has eliminated much of the difficulties encountered in the process of production: At each stage of product development, a large amount of data needs to be passed from the first to the next stage. Traditionally this was done manually. A person had to read the data and make calculations and decisions accordingly. This approach often led to mistakes being made, and the unsatisfactory outcome of a product that was either too big or found to be smaller than required.

With the development of computers and numerically controlled machines, mathematical methods were incorporated to assist not just the designer, but the whole process of production. Bezier [BEZI 90] outlines some of the thoughts and requirements of the personnel involved in making a product. His reflections are based on the prototype that was developed in the sixties to aid Renault's car manufacturing process. An in-depth account of the development and usage of these control machines can be found in Bezier's original (English) publication [BEZI 72]. He also supplies the necessary mathematics that may be used to describe contours of shape. Yamaguchi [YAMA 88] gives an historical account of the introduction of mathematics in representing outlines. In addition, he describes the development and application of various numerical control machines. Faux and Pratt [FAUX 79] give a comprehensive illustration of the usage of mathematics for the purposes of design and manufacture. They provide much of the geometric concepts that are necessary to build a mathematical or numerical model of a given object.

2.4.3 Typographic Systems

The numerical control of application machines, such as for milling and cutting, is just one situation where a mathematical model of a component or an object is effectively used. The model can be used to drive outputs on typesetting machines and to create images on film and/or paper. Furthermore, a mathematically modelled image can be displayed on a modern display screen.

One popular application of mathematical modelling occurs in the field of typography, where outlines of font characters need to be accurately represented. Characters, and thus font types, have an important role to perform: They are not just required in the commercial sector for design purposes, but are necessary in all aspects of life to formulate a channel of communication. The type and style of a font is normally chosen to suit a particular application and/or a selected group of people [GLEN 84].

The traditional approach to capturing outlines of characters (drawn by proficient artisans) was to represent its shape on a metallic block. Each contour, of a character, was carved onto the block with great care and accuracy. The modelled characters were then employed to produce prints as and when necessary. Lieberman [LIEB 67] gives a comprehensive historical background to the development of both font type and its technology.

Modern design systems attempt both to automate and to make the process of capturing outlines of font more versatile. The use of mathematics in capturing systems has meant that changing the size, position, shape, style or sense of orientation etc, of a modelled character, can easily be accomplished. Much of the work that traditionally took days to complete can now be performed in minutes or even seconds.

Several design systems have emerged within the last two decades. They offer some form of flexibility to the designer so that he is able to create, modify and change a desired shape without too much difficulty. Ruggles [RUGG 83]

provides a brief description of some design systems which have been developed specifically for the field of typography. He highlights the fact that although most design systems use a spline routine (to capture outlines of characters), the manner and method this is employed varies from system to system. To illustrate the point, two well known design systems (namely IKARUS and METAFONT) are described here:

The IKARUS system was developed by URW (Germany) in the early seventies. It was pioneered by Peter Karow, and follows the principle of first capturing the given outline in a "non-mathematical" form. This is done by quantifying the data points (describing the outline) as being either a corner, curve, start or a tangent point. Once this is accomplished, a second capturing stage is then activated where the outline is defined in terms of a mathematical description. The IKARUS system gives a simple but effective approach for modelling outlines of any shape. A full description of the IKARUS system can be found in Peter Karow's book [KARO 87].

The METAFONT system in contrast is much more of a mathematical programming language. It was pioneered by Donald Knuth in the late seventies [KNUT 79, KNUT 82, KNUT 85]. The outcome of its working is very much similar to that of 'join-the-dots' in pictures. A pen is programmed to draw an outline from one position to the next. The outline can be drawn in various sizes, and can also take the form of a line or curve. An output of a modelled outline is then produced by executing the program instructions.

Both design systems provide a means of mathematically modelling in a form desired by the designer. The fact that the METAFONT is a programming language leads itself to some drawbacks, the main one of which is that the system is declarative rather than interactive, requiring the designer firstly to create the program that includes a mathematical description and then to execute it, a process that can be fairly time-consuming.

2.4.4 Display Factors

Having mathematically modelled a character outline, the next stage is, normally, to display the outline on some graphics display terminal. The problem is that these display terminals have a finite resolution, resulting in a distorted (jagged) outlines. Various authors have done some research on assessing the factors that effect the quality of characters (text) on display screens, including: Kindersley and Wiseman [KIND 79] who give a comparison of modern techniques with established (printing) methods; Pringle, Robinson and Wiseman [PRIN 79] discuss the merits of text representation on raster-scan displays (such as televisions); Bigelow [BIGE 85] looks at the traditional factors governing the design of font characters. He suggests that because of the limitations of technology to improve screen resolution, it may be better to design new fonts to meet the digital environment. Pitteway and Banissi [PITT 87] suggest the use of soft-edging (greyscaling) to improve text representation.

Display screens, in fact, offer the most coarse resolution when compared to typesetting devices such as laser printers and flat bed drawing machines. The maximum resolution of display screens is around 100 lines per inch. Laser printers have resolutions that can range from 300 to 800 lines per inch. Flat bed drawing machines offer resolutions of 1000 to anything up to 5000 lines per inch [KARO 87]. Although modelled outlines displayed on screen terminals will appear less acceptable than those drawn on a high resolution device, they (screen displays) provide a suitable means for both interactive work and preparation stage for the high resolution devices.

Outputting modelled outlines on a low resolution device will, undoubtedly, lead to distortions of the type mentioned above. What makes the output look even worse is that the modelled character loses its proportionality. This may result in the three stem widths of the character 'm', for example, all being different. Distortions of this nature can make a character shape and type unrecognisable. To overcome some of these problems, special instruction (hints) are embodied as part of a character's outline description. They are then used by output devices to

ensure that the basic features of font characters are preserved. Karow [KARO 91] gives an account and some requirements of providing hints to improve the output quality of character outlines. He compares the type of hints being used by various established commercial packages, and elaborates on the type and form of each hint.

2.5 Summary

This chapter gives some of the aspects and concepts concerning the mathematical capture of shape outlines. It introduces the spline, probably the most important component in modelling contours. The use of piecewise splines to represent complex shapes, and control factors such as shape and continuity are also discussed. The features of spline subdivision, degree elevation and reduction are also highlighted.

The two approaches for capturing an outline, given as a set of data points, are presented. Both of these can be employed in an interactive system to achieve desired results. When the given data points are in-tune with the desired shape, the interpolation approach is usually employed. For data that contains noise factors, the approximation method of least-squares is normally preferred.

Finally, this chapter outlines some factors that are normally considered when attempting to model shape contours. The terms fair and smooth, when applied to shape appearance, are introduced. The application of mathematics to numerical machines and typographic systems are also discussed. Some of the considerations and difficulties relating to displaying mathematically described outlines on output devices (of, particularly, low resolution) are presented.

3.0 Capture by Bezier Splines

3.1 Introduction

One of the most popular and widely used technique for capturing contours of a given shape was invented by P Bezier. He managed to develop an approach that was powerful enough to represent any outline, yet simple enough for designers who thought in terms of drawings rather than equations. His contribution was warmly accepted and incorporated in the design stages of modelling motor vehicle bodies [HOCH 90].

This chapter firstly introduces the Bezier spline, its desirable properties and its mathematical forms. Through this, the parametric form is analysed, its benefits and possible drawbacks are highlighted. Two approaches are then presented for capturing a given set of data points in terms of Bezier *cubic* splines. The first method is in line with traditional approaches where a parametric form is used. A mathematical modelling algorithm is developed and its performance assessed. The second method focuses on a non-parametric approach. This consists of converting the parametric form to a non-parametric representation. The effectiveness of this method is then evaluated and an ill-condition illustrated. Finally, the techniques for rasterising (the process of choosing the "best" pixels on a digital display, for example) the Bezier cubic outlines are looked at. A new approach for tracking the contours of an outline is developed, and examples are given of its successes and failures.

3.2 Characteristics and Properties

This section outlines some of the features and properties that the formulation developed by Bezier offers. It therefore looks at some of the reasons that has attracted designers to employ Bezier splines to model outlines.

3.2.1 Polygonal Framework

The development of Bezier curves eased some of the difficulties which designers had encountered using either methods pioneered by Coons or Ferguson (see [COON 67] and [FERG 64] respectively). The main inconvenience imposed by these methods was in controlling the shape of a modelled surface, for example. In this case, large amount of input data such as tangent and twist vectors was required. Designers found it difficult to predict the effect of these parameters on the shape of a surface and, thus, evaluating suitable values for them was, in some cases, extremely hard. Yamaguchi [YAMA 88] gives a detailed discussion of the methods developed by Coons and Ferguson, and mentions some of their undesirable features as far as modelling shape outlines are concerned, whether in 2-D or in 3-D.

Bezier's description of shape curves and surfaces requires only the use of positional vectors. The defining polygon, in the case of a curve description, gives a coarse approximation regarding the form and outline of a given shape. The closeness of the defining polygon to the modelled shape, in practice, depends on the order (and thus the degree) of the Bezier spline being used. The polygon of any ordered spline tends to give some indication about the modelled shape, but the relationship between the polygon and the given outline becomes weaker as the order of the Bezier spline is increased [FAUX 79]. Yamaguchi [YAMA 88] illustrates with some eye-catching examples using high-ordered Bezier splines to construct some unusual shapes. In all cases, it is extremely difficult to predict from the polygonal set-up the resulting outline of shape.

The defining polygon, as mentioned in section 2.2.3, also provides a window for editing and modifying the shape. This is particularly useful when a given shape is modelled by a series of Bezier splines patched together. The shape modelled by one or more of these defining polygons can be changed without (necessarily) effecting the shape represented by adjacent polygons. This then localises the problem of capturing a given shape within a desired specification.

Bezier splines as described by a polygon do not, in general, pass through all the defining (n+1) positional vectors. As shown in Fig 3.1, they interpolate the two endpoint vertices P_0 and P_n (henceafter called the knot points), and "approximate" the remaining vertices (P_1 to P_{n-1}), henceafter referred to as control points). The defining polygon is characterised such that the tangents at the knot points only depend on the adjacent control points. This results in the tangent at the starting knot being governed by P_1-P_0, and that, at the finishing knot by P_n-P_{n-1}. Higher derivatives at the knot points employ additional control points: The second derivative at start knot P_0, for example, is obtained by a combination of P_0, P_1 and P_2. In general, therefore, the rth derivative at a knot point is determined by its r neighbouring control points. This characteristic of Bezier splines is used in applications to achieve a high-ordered continuity between joining arcs. Gorawara illustrates this point by joining two cubic arcs characterised by Bezier polygons [GORO 86]. He provides much of the mathematics required for this purpose.

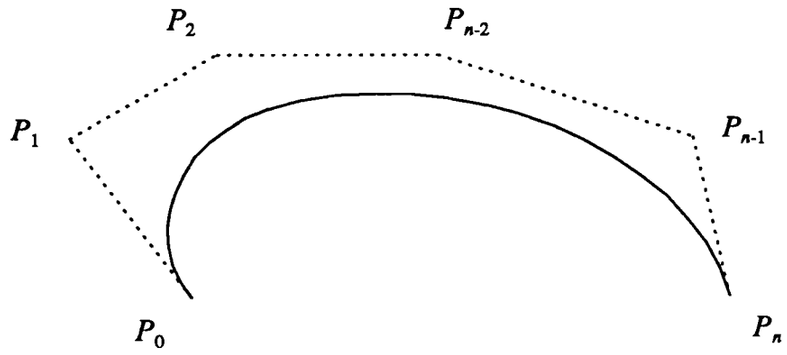


Fig 3.1 Highlights the relationship between the Bezier polygon (shown dashed, and without the line connecting the two knots, to aid clarity) and its resulting curve segment.

Once a Bezier polygon is established, the resulting curve is conditioned to lie within it. This is an important property that all Bezier curves exhibit. The curve is said to lie within the convex hull of the polygon (see [FARI 90, NEWM 79]). This has practical applications in evaluating, for example, whether two, or more, paths (projected by Bezier curves) overlap: Sederberg and Nishita [SEDE 90] use the convex hull property to perform "Bezier clipping". They locate points of

intersection by firstly identifying and then clipping away regions which do not include the solution.

Bezier originally characterised the polygon in terms of its sides [BEZI 72]: Given a set of defining vertices, as in Fig 3.1, a polygon was constructed by joining together the tangent vectors going from one vertex to the next. This meant that a chain of relative vectors was being employed to gain a Bezier description. Forrest developed a formulation which used the vertices (control and knot points) of the defining polygon rather than its sides [FORR 72, FORR 90]. As he mentions, using absolute (positional) vectors makes the formulation more compact and elegant, and has a greater intuitive appeal. Furthermore, he makes the point that the effects of rounding errors (particularly when performing transformations such as rotation) are less noticeable in a system which employs absolute vectors rather than relative vectors.

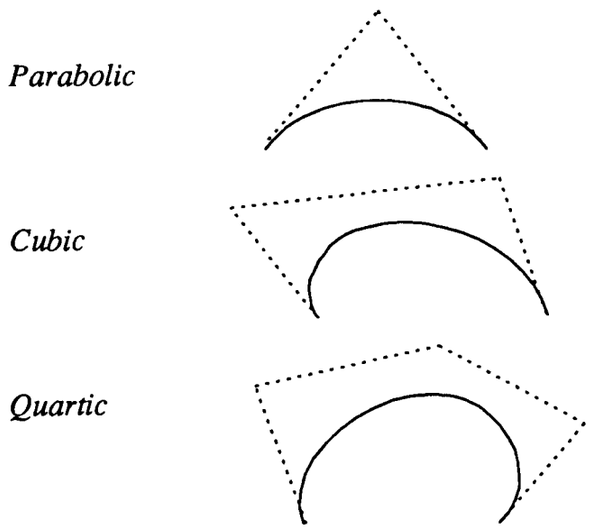


Fig 3.2 Shows how by increasing the number of polygon sides (shown dashed) also raises the degree of the Bezier spline.

The number of sides, and therefore control points, a Bezier polygon exhibits is directly related to the degree of the spline being used. As depicted by Fig 3.2, a quadratic (more precisely, parabolic) spline (in addition to the two knot points)

has one control, a cubic spline two control points and a quartic spline three control points. Thus in general a defining polygon is characterised to have one more point (knot plus control) then the desired degree of the spline.

3.2.2 Parametric Representation

Before examining the shape control parameters of Bezier curves, it is worth mentioning the role of parametric descriptions: Ferguson, in the early sixties, introduced the idea of employing parametric (cubic) curves and surfaces in the design of fuselages (contour of aircrafts) [FERG 63]. He included an additional parameter, sometimes called the parametric variable, to overcome some of the problems faced by a non-parametric mathematical form. He chose to employ, for each cubic curve segment, an equation of the form:

$$P(t) = a_0 + a_1\, t + a_2\, t^2 + a_3\, t^3 \,, \qquad \text{...(3.1)}$$

where: $P(t)$ is the positional curve vector,
a_0 to a_3 are vector coefficients, and
t is the parametric variable.

The points on a curve, expressed in terms of equation (3.1), were then readily available by stepping through the parametric variable t.

The attractive features of the parametric form has since meant that a number of mathematical techniques (including Bezier - see next section) employ this representation. Forrest devotes a whole chapter (chapter five in [FORR 68]) to discussing the advantages of using the parametric form. He mentions the appealing characteristic of this form in having a workable representation for curve outlines that possess infinite slopes. Faux and Pratt [FAUX 79], and Yamaguchi [YAMA 88], list this and other reasons for the popularity of the parametric form in describing outlines of shape.

Although the parametric form does have its benefits, it also has its limitations and drawbacks: When splines, whether cubic or otherwise, are used to model an

outline given as a set of data points, a relationship between the given data points and the parametric variable needs to be established in order to assess the goodness-of-fit [PAVL 83]. As it will shown in section 3.4, this process tends to be iterative, and therefore, time consuming.

3.2.3 Blending Functions

The benefits offered by employing parametric descriptions and functions are more intuitively appealing to the designer when their exact role is predictable. This, in the case of Bezier splines, means that a direct relationship between the parametric curve description and the polygonal (both control and knot) points is required. A common way of achieving this is through the application of blending functions. These are functions that "blend" the effects of the given geometric constraints, such as polygonal points, to generate a curve description.

Coons and Ferguson employed the use of Hermite blending functions in their modelling of curves and surfaces [YAMA 88]. Bezier, on the other hand, chose a family of functions called Bernstein polynomials to create a formulation for describing contours of shape [FAUX 79, FORR 90, MORT 85]. Given a set of polygonal points $P_0, P_1, \ldots, P_n$ (as in Fig 3.1), Bezier's general (non-rational) parametric equation takes the form:

$$P(t) = \sum_{i=0}^{n} P_i B_{i,n}(t), \qquad \ldots(3.2)$$

where: t is normalised to range from 0 to 1.

The $B_{i,n}(t)$ are a set of Bernstein blending functions, defined as follows:

$$B_{i,n}(t) = \frac{n!}{i!(n-i)!}\, t^i\, (1-t)^{n-i}.$$

It should be clear from equation (3.2) that the blending functions act as weights for each of the given polygonal points. These functions therefore are the key to the behaviour of Bezier curves. To analyse and understand their role further, the blending functions that the Bezier *cubic* curve exhibits are closely examined.

Equation (3.2) can be rewritten specifically for the cubic case, n=3, as follows:

$$P(t) = P_0 (1-t)^3 + 3P_1 (1-t)^2 t + 3P_2 (1-t) t^2 + P_3 t^3 . \qquad ...(3.3)$$

The first point to note is that there are four blending functions, one for each of the polygonal points. Each function has an influence on the shape of the generating curve. Fig 3.3 gives an illustration of the amount of influence each function exhibits as the parametric variable t is incremented [MORT 89]. The blending functions (as expected, and mentioned in section 3.2.1) facilitate the interpolation of (knot) points P_0 and P_3, at t=0 and t=1 respectively.

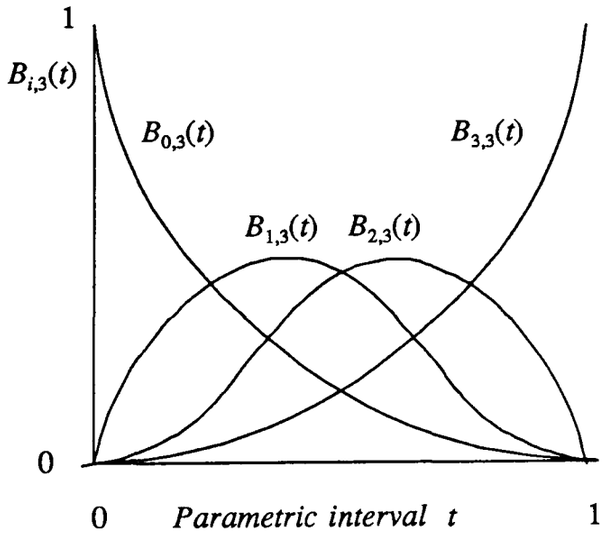


Fig 3.3 Depicts the behaviour of the blending functions associated with each of the respective polygonal points.

The amount of bias each blending function exerts varies as the value for the parametric variable is increased. The maximum "pull" of each function (ie knot

and control point) occurs for P_0, P_1, P_2 and P_3 at t=0, 1/3, 2/3 and 1 respectively. In general, the maximum influence of a Bezier control point P_i will occur when the parametric variable t becomes i/n, where n+1 are the number of polygonal points. This fact can be used to assist designers working with an interactive design system: Chasen [CHAS 78] assesses the influence of moving the control points of a Bezier cubic polygon. He gives examples of how an experienced designer may use the polygonal nature, with its blending functions, of Bezier cubics to achieve desired results.

The parametric variable t is normalised to lie in the interval of zero and one. This is done because it is convenient, not because it is necessary [FARI 90]. The variable can take on arbitrary values (as used in section 3.6.3), so that the shape of an entire curve can be sketched. In other words, the parametric interval (in conjunction with the defining polygonal points) is used to return a section of a complete curve.

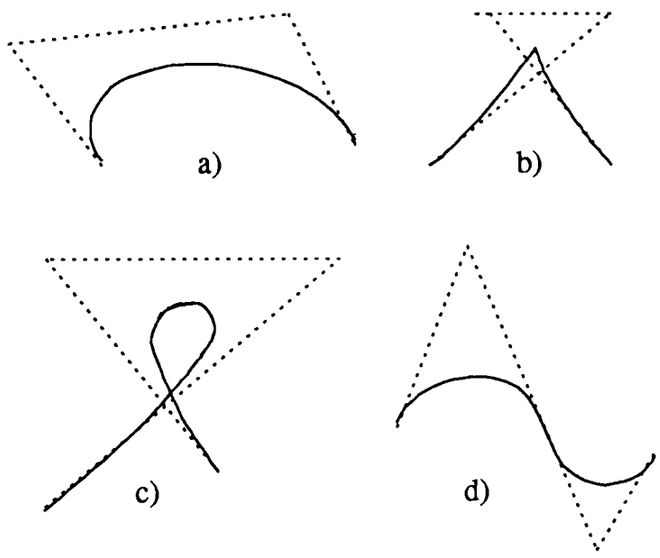


Fig 3.4 Demonstrates possible curve shapes a Bezier cubic spline can generate: a) an arc, b) a cusp, c) a loop, and d) an arc containing one inflection point.

The Bezier cubic spline, as Fig 3.4 manifests, can be set-up to return a number of different shape curves. A single spline can represent either a straight line, an arc with zero, one or two inflections, a cusp and a loop. Stone and DeRose [STON 89] show how by re-positioning a knot point of a given Bezier cubic polygon, the spline can be made to return any of the above curve shapes. As they mention, the knowledge of whether a cubic curve segment contains inflection points or a cusp or a loop can highlight unwanted features of a modelling scheme. Patterson [PATT 88] makes use of an algebraic form to determine various properties of the Bezier cubic curve. He extends his work beyond the parametric (variable) interval of zero and one.

In concluding, it should be mentioned that the use of Bernstein blending functions limits the flexibility of Bezier splines in two ways: Firstly, the number of polygonal points (knots plus control) fixes the order (degree) of the resulting Bezier spline. This means, for example, a cubic spline has to be defined by four polygonal points, a quartic by five, and so forth. It is therefore not possible to increase the order of a spline without increasing the number of polygonal points, and conversely, the only way to decrease the degree of a spline is to make a reduction in the number of polygonal points.

The second limiting factor is due to the global nature of Bernstein blending functions. As Fig 3.3 demonstrates, all of the blending functions within the parametric interval of zero to one, have values which are non-zero. This means, as indicated in sections 2.2.3 and 2.2.4, a small change in the location of the control points is "felt" throughout the entire modelled outline. This situation, as mentioned in section 2.2.3, can be catered for, but it involves the process of subdivision.

(In passing, it should be mentioned that by employing the rational parametric form for the Bezier cubic, the shape of the basis functions, and thus the amount of bias they exert, can be controlled. Each polygonal point has associated with it an additional parameter called a "weight". The value taken by this variable gives a measure of the "pull" that each polygonal point exerts. Rational Bezier

forms are discussed in the literature; a geometrical introduction is given by Piegl [PIEG 86].)

The two limitations, of the Bernstein basis, can be overcome by employing blending functions that are normally non-global within a curve segment, and whose order of curve is not set by the number of polygonal points. The types of mathematical curve descriptions which embody these properties are called B-splines. Discussion of these splines is left to the references listed in section 2.3.1.

3.2.4 Applications of Cubic Splines

Some of the features of the Bezier cubic spline have already been presented in this chapter. Because of its widespread use in industry, and the fact that it will be employed to solve the problem outlined in section 3.3, further elaboration on its applications and characteristics are given in this section.

In section 2.2, it was mentioned that the cubic spline is the mathematical analogous of the physical spline used to model outlines of ship hulls, aircrafts etc. It can provide second-order continuity between segments, and has discontinuities only in the third derivative. As it is difficult for the human eye to distinguish the latter, the resulting spline appears reasonably smooth [PAVL 82]. Furthermore, the cubic spline is the lowest degree space curve which has the ability to twist through space [ROGE 76]. These two properties have contributed mainly to their successful employment in a variety of applications.

Apart from their role in the motor vehicle manufacturing industries, cubic splines have been applied to model an extensive range of objects: Font characters used in laser printers often employ Postscript. The outlines of these characters are modelled by Bezier cubic splines [GROS 90]. Traditional chinese calligraphy and paintings require the finest skills in executing brushstrokes. Chua [CHUA 90] illustrates the use of Bezier cubic splines in modelling these brushstrokes. In the field of dentistry, a common requirement is to model the outlines of dental

arches. Various mathematical descriptions are normally used, including cubic splines [BEGO 79].

The use of cubic splines, as can be gathered from the diverse applications cited above, is a popular means of describing shape outlines. As it might be expected, various forms and types of cubic splines (including Bezier) exist. An extensive survey in respect of this has been carried out by Boehm [BOEH 82]. He introduces the various representations, their differences and relationships. For a more general look at the various mathematical descriptions and methods, Faux and Pratt [FAUX 79], Boehm *et al* [BOEH 84], Yamaguchi [YAMA 88] and Farin [FARI 90] give an instructive introduction.

3.3 Outline of Problem

Having presented some of the properties and features that have made the use of Bezier splines popular, this section describes the nature and type of approximation required:

The IKARUS design system, as mentioned in chapter one, requires to model mathematically outlines of font characters. It employs a number of descriptions, including the use of Bezier cubic sections. The problem therefore is to develop techniques that will efficiently capture the character contours. The efficiency is judged mainly by the speed of capture (conversion), although the number of Bezier curve segments used are also observed. The problem can be summarised as follows:

Given a set of data points (Q_i) describing an outline, whose directional tangents at each point are known, a mathematical model of the outline is required which uses the minimum number of Bezier cubic segments. The resulting model needs to represent the given data points within a predefined tolerance.

The tangents for each of the data points are *not* provided by the designer, but are evaluated in the preparation stage for all digitised contour points. As mentioned

by Karow [KARO 87], the best way of achieving this is to employ an algorithm which interpolates the given data points. They base this algorithm on the methods discussed by Spath [SPAT 74]. The use of tangents in the capturing scheme should ensure that the resulting Bezier cubic curve segments possess at least first order (tangent) continuity at the (curve) knots.

Two approaches for finding an acceptable solution to the given problem are developed: The first, in section 3.4, takes a look at the 'traditional' (non-rational) parametric approach. Some of the concepts involved are highlighted and a method is developed which meets the given specifications. The performance of this method is assessed and possible improvements investigated.

In section 3.5, a non-parametric solution to the problem is presented. The parametric form of the Bezier cubic formulation is transformed into a corresponding non-parametric form. An ill condition in the representation is highlighted and its possible consequences illustrated through an example.

3.4 Capture by Parametric Form

The use of parametric curves for modelling outlines can be split into three distinct areas: The first area provides the initial values for the parametric variable, the second evaluates the corresponding Bezier control points, whilst the third stage both assesses the goodness-of-fit of the resulting curve and, if necessary, updates the parametric values. The way these are employed to form a capturing algorithm is shown in Fig 3.5. Detailed discussion of each of the three primary stages is given in the subsequent sections. The performance of the algorithm is presented in section 3.4.4.

3.4.1 Parametrisation of Outline

The given data points (hereinafter referred to as IK points) reflect the outline of a character to be mathematically modelled. Each IK point has been digitised with a view to the type and form of the outline being captured. The various guidelines for digitising outlines to provide suitable IK points has been addressed by Karow

[KARO 87]. The important point here is that the IK points have been supplied with some insight to the nature of the character outline. The resulting values for the parametric variable t_i must therefore share, and be seen to reflect, this. The process which gives initial values for t_i is called *parametrisation.*

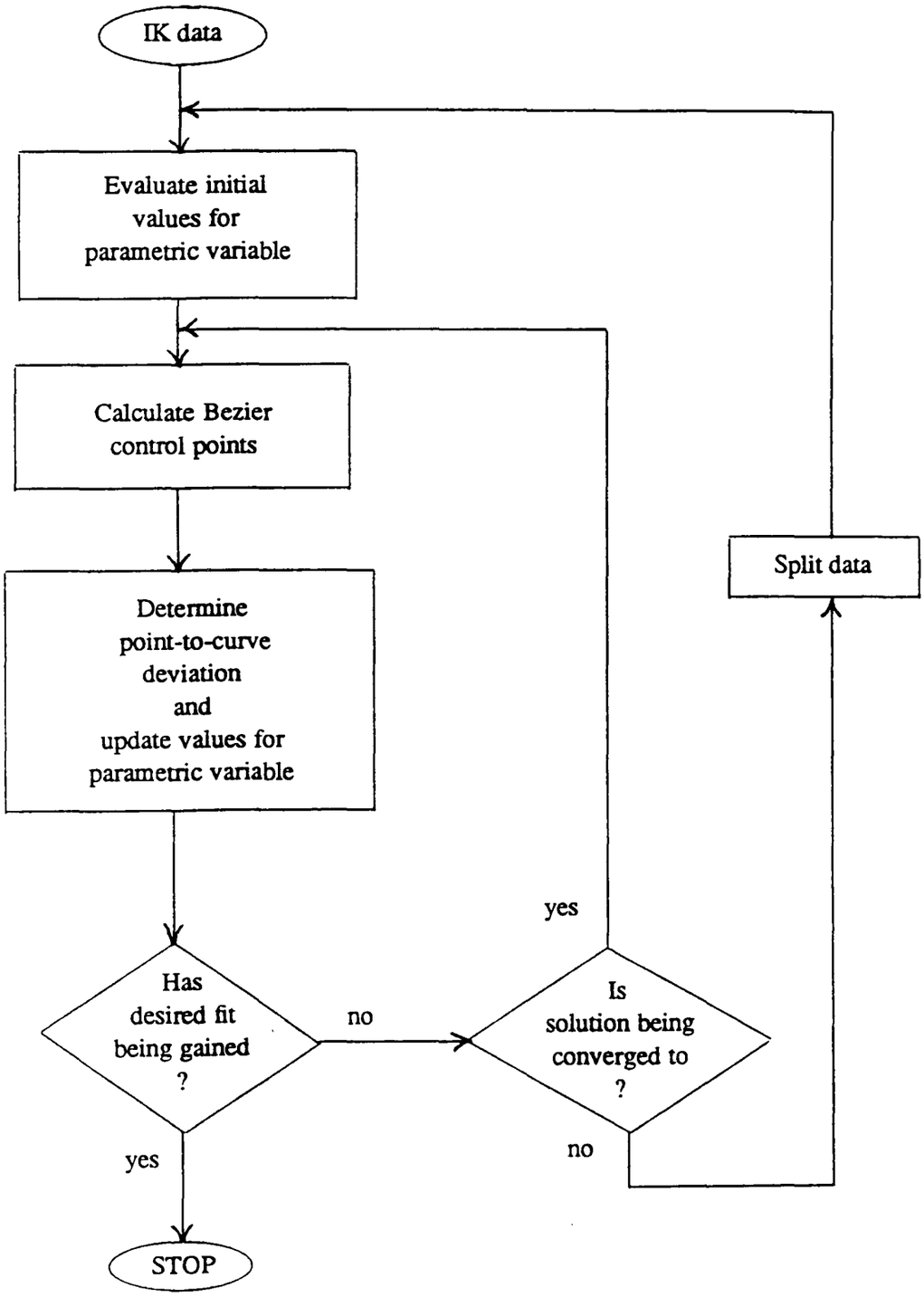


Fig 3.5 Shows the various steps necessary to perform an IK to Bezier cubic description using the parametric approach.

There are a number of methods that can be employed to return suitable parametrisations. Farin [FARI 90] discusses some of these and assesses their performance through examples. Like he points out: "There is probably no "best" parametrisation, since any method can be defeated by a suitably chosen data set". A common and popular approach is to use the relative distances between given points [FAUX 79]. This is known as the *chord length* method. Fig 3.6 illustrates the approach.

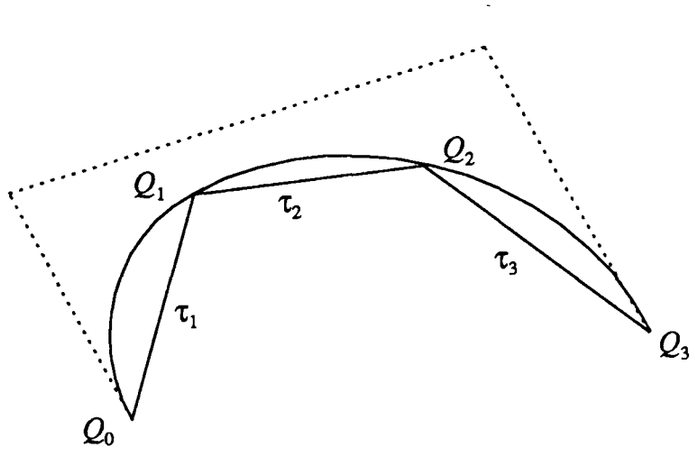


Fig 3.6 Gives an illustration of the chord length method for estimating initial values for the parametric interval t_i, for a set of IK points Q_i, using the τ approximations.

It is apparent from Fig 3.6, that the chord length approach consists of providing an initial approximation based on line segments. The amount of line segments used is directly related to the number of IK points supplied. As the parametric variable requires to range from zero to one, each chord length needs to be expressed in terms of the total chord length. Mathematically, the method can be described as follows:

$$t_i = \frac{\sum_{j=0}^{i-1} \tau_j}{\sum_{j=0}^{n-1} \tau_j} \qquad ...(3.4)$$

where: $\tau_j = | Q_{j+1} - Q_j |$,

n = total number of IK points.

Although the chord length method is an effective way of attaining initial values for the parametric variable t_i, an approach based on the actual curvature tends to give better results [COHE 89]. This, however, leads to the practical problem of how to determine the curvature information about a given set of IK points, whose tangents at each point are known. A solution which has been adopted at URW, is to compute circular arcs between two IK (curve) points. The mathematics for this approach are given by Karow [KARO 87], and further details and discussion can be found in the following references: [BOLT 75, RUTK 79, ALIA 87, THOM 89]. This technique of gaining curvature data allows the initial values for t_i to be based on arc lengths, rather than the chord approach.

The main purpose of this routine is to introduce additional (help) points that would make the resulting modelled (by Bezier cubics or otherwise) outlines aesthetically acceptable. In section 3.4.4 (results section), this routine has been employed for the following two purposes: Firstly, it has been used to analyse and assess the effects (if any) of evaluating the initial values for t_i through the arc lengths returned by the circular arcs and those calculated using the chord length approach. Secondly, the effects of introducing additional curve points at various locations along a given character outline are observed. In both cases, the performance of the algorithm is judged in terms of speed and accuracy.

3.4.2 Evaluation of Control Points

Having gained initial values for t_i which correspond "best" with the given IK points, a process for evaluating the control points is considered in this section. For this purpose, the given conditions, such as tangents, are utilised to develop a reasonably compact form for the unknown variables. A numerical approach, similar to that used by Engels [ENGE 86], is employed to solve for the control points. The process in detail is as follows:

Looking at the Bezier cubic form expressed in equation (3.3), it is clear that in general the solution to eight unknowns is required. As the knot points are given (ie P_0 and P_3) through which the resulting curve must pass, this leaves the determination of the two control points, that is P_1 and P_2. Since the tangents at the knot points are also known, this reduces the overall problem to calculating the 'control lengths' l_1 and l_2. In other words, the two control points take on the following expressions:

$$\begin{aligned} P_1 &= P_0 + T_1\, l_1 \,, \\ P_2 &= P_3 + T_2\, l_2 \,. \end{aligned} \qquad \text{...(3.5)}$$

where: T_1 is directional vector at start knot,

T_2 is directional vector at end knot, and

$|T_1| = |T_2| = 1$.

Substituting the expressions of equation (3.5) into (3.3), and by collecting common terms, equation (3.3) then takes form:

$$\begin{aligned} P(t_i) = {} & P_0\,[s_i^3 + 3s_i^2\, t_i\,] + P_3\,[t_i^3 + 3s_i\, t_i^2\,] \\ & 3T_1\, l_1\, s_i^2\, t_i - 3T_2\, l_2\, s_i\, t_i^2 \,, \end{aligned} \qquad \text{...(3.6)}$$

where: $s_i = 1\text{-}t_i$.

It is clear from equation (3.6) that the only unknowns are the control lengths l_1 and l_2. The solution to these has to be such that it minimises the distances (residues) of all the given IK data points Q_i (x_i,y_i) to their corresponding curve points $P(t_i)$. A simple way, in principle, of achieving this is to set-up initially a function that returns the square of the residue values at each IK point. That is:

$$\Gamma = \sum_{i=1}^{n-1} d_i^2 \,, \qquad \text{...(3.7)}$$

where: $d_i = Q_i - P(t_i)$.

The best values for the control lengths are then gained when Γ is a minimum. This, in mathematical terms, occurs when:

$$\frac{\partial \Gamma}{\partial l_1} = 0 \ ,$$

$$\frac{\partial \Gamma}{\partial l_2} = 0 \ . \qquad \text{...(3.8)}$$

An analytical approach to solve for the control lengths can be used by combining the terms generated via equation (3.8). Although this forms an approach, a better way is to apply a numerical method. This has a matrix form which attracts computation efficiency, and as section 3.4.4 shows, has a faster IK to Bezier conversion rate when compared to the analytical approach. The numerical method being employed is a frequently used technique (see [ENGE 86], for example), whose solution process can be summarised as follows:

Given a solution matrix of the form $[A]\,[x] = [b]$, where the size of A is n x m (n=total number of observations, m=number of unknowns). The corresponding least squares solution is obtained by reducing the matrix A to be of size m x m. This is achieved by multiplying each side of the solution matrix by the transpose of matrix A (A^T). That is:

$$[A^T]\,[A]\,[x] = [A^T]\,[b] \ . \qquad \text{...(3.9)}$$

To solve for the two unknowns, the expressions of equation (3.7) are formulated with those required by equation (3.9). The tangents at the start and end knots are expressed in terms of their components: tx_0, ty_0 and tx_n, ty_n respectively. After combining and simplifying terms, the solution matrix takes the following form:

$$\begin{bmatrix} f_1^2 + f_2^2 & f_1*g_1 + f_2*g_2 \\ g_1*f_1 + g_2*f_2 & g_1^2 + g_2^2 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} = \begin{bmatrix} f_1*h_1 + f_2*h_2 \\ g_1*h_1 + g_2*h_2 \end{bmatrix}, \qquad \text{...(3.10)}$$

where:

$$
\begin{aligned}
f_1 &= -3s_i^2\ t_i\ tx_0\ , \\
g_1 &= -3s_i^2\ t_i\ ty_0\ , \\
f_2 &= 3s_i\ t_i^2\ tx_n\ , \\
g_2 &= 3s_i\ t_i^2\ ty_n\ , \\
h_1 &= x_0\ [s_i^3 + 3s_i^2\ t_i\ ] + x_3\ [t_i^3 + 3s_i t_i^2\ ] - x_i\ , \\
h_2 &= y_0\ [s_i^3 + 3s_i^2\ t_i\ ] + y_3\ [t_i^3 + 3s_i t_i^2\ ] - y_i\ .
\end{aligned}
$$

As is apparent from equation (3.10), relatively little computation is required to solve for the functional expressions (ie f_1, f_2, g_1 etc). Indeed, as the two knot points are supplied, the given IK curve points can be translated to start from an origin (0,0). This will make the expressions for h_1 and h_2 (in equation (3.10)) simpler, and thus further enhance the conversion rate.

The solution for the control lengths that best fit the given IK points (with the assigned parametrisation), is then gained by employing a suitably fast numerical subroutine, such as that developed by Lewis [LEWI 86]. Once the control lengths have been evaluated, equation (3.5) can be employed to calculate the corresponding Bezier control points.

3.4.3 Calculation of Curve Deviation

With the computation of the control points, the given set of IK curve points have been modelled by a corresponding Bezier cubic spline. The next task is to evaluate whether this spline matches the desired goodness-of-fit. If this is in the affirmative, then a best-fitting mathematical description has been gained; otherwise the values of t_i require adjusting, and new control points need to be calculated.

To assess the goodness-of-fit, the shortest distance between the generated curve points $P(t_i)$ and the corresponding IK points Q_i needs to be measured and compared with the desired tolerance. Equation (3.7) can be employed to return

the corresponding deviations. Once these have been computed, it is necessary to calculate the worst deviation produced by the curve. If this is found to be greater than the desired tolerance, re-parametrisation of the given IK points needs to be performed with reference to the existing values of t_i and also to the resulting deviations.

For the purposes of successive improvements to the values of t_i, two techniques are employed: The first approach is the Newton-Raphson method. This a frequently used technique to locate roots of an equation of the form $f(x)=0$. Plass and Stone [PLAS 83] elaborate on its application in providing successive t_i values. As they point out, the Newton-Raphson approach can be formulated to solve for $f(t)=0$ as follows:

$$t_i = t_i - \frac{f(t_i)}{f'(t_i)} ,$$

where:

$f(t_i) = x'(t_i)[x(t_i)-x_i] + y'(t_i)[y(t_i)-y_i]$,

$f'(t_i) = x'(t_i)^2+y'(t_i)^2+x''(t_i)[x(t_i)-x_i]+y''(t_i)[y(t_i)-y_i]$,

$x'(t_i)$ and $y'(t_i)$ are respective first derivatives with respect to t of equation (3.3), and

$x''(t_i)$ and $y''(t_i)$ are respective second derivatives with respect to t of equation (3.3).

The second approach, which has been employed in the IKARUS system, makes use of tangent vectors [MEIE 89]. Fig 3.7 gives a graphic illustration of the technique, the basic concept of which is as follows: The distance d_i, as expressed in equation (3.7), yields the deviation of the curve point $P(t_i)$, based on the value of t_i, to the given IK point Q_i. Describing the deviation as a tangent vector D_i, we can gain its (local x and y) components; namely the tangent vectors Dx_i and Dy_i. The values for these components, as Fig 3.7 shows, are with reference to the

gradient of the curve at t_i. Adjustments to t_i are made via the vector Dy_i; resulting in a new value shown as t_i^* in Fig 3.7.

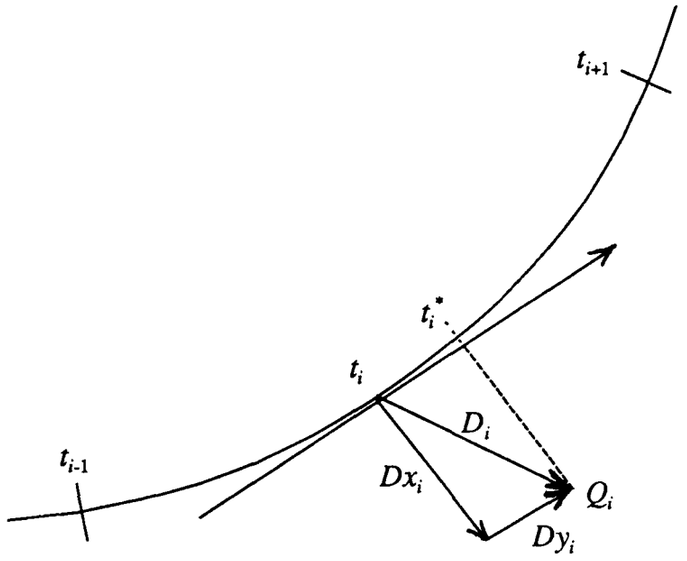


Fig 3.7 Depicts the vector approach for making successive improvements to the values of t_i.

Improvements to the values of t_i are made, using either approaches, until a desired fit results or when the adjustments to the t_i values are not significant. In the latter case (and also when the convergence is slow so that it demands a high number of iterations) the given IK data points are subdivided and represented by two Bezier cubic splines.

3.4.4 Analysis and Observations

Having described an approach for each of the primary stages of the Bezier conversion algorithm, its performance is analysed in this section. For this purpose, the 'S' character from URW's typeface codename an201215.ik (antiqua) is selected. As Fig 3.8 depicts, the body of this character is mainly described by two, rather lengthy, curve sections. Each contains two points of inflection, marked with an asterisk (*) in Fig 3.8. It is apparent from the location of these points that the minimum number of Bezier cubic segments required to capture the two given curve sections is four. Furthermore, the positioning of the knot points

needs to be such that only one of the inflection points appears within the solution.

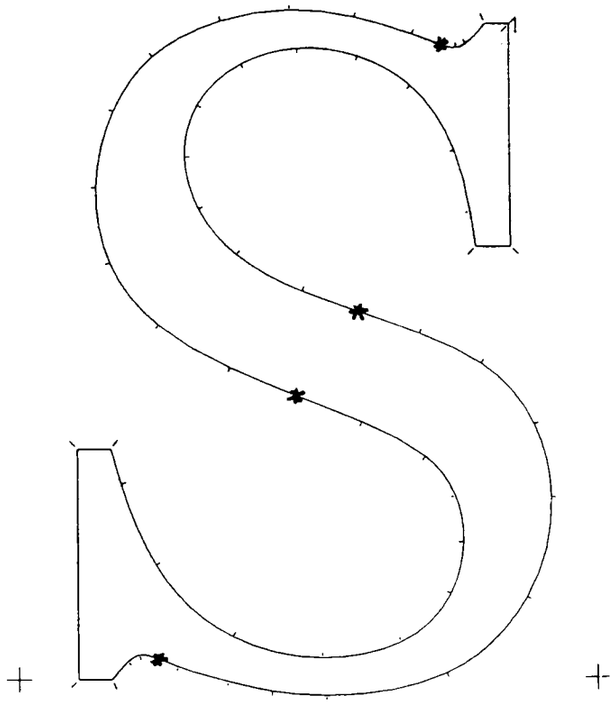


Fig 3.8 Gives the IK defined outline of the 'S' character, points of inflection shown with an asterisk (*).

In order to make a thorough assessment of the conversion process, each of the three stages shown in Fig 3.5 are realised through two respective approaches (as discussed in the relevant sections above). This means that the initial values for the parametric variable are either gained via the chord or arc length method, the control lengths are either analytically obtained or through the approach outlined in section 3.4.2, and the process of successive improvement is made using either the Newton-Raphson method or by the vector approach depicted in Fig 3.7. In addition, the help-point routine is used to examine further the effects, if any, of describing the character outline through a varied number of IK points.

To enhance the clarity of the results, the following abbreviations are used to represent the corresponding approach:

Stage	*Approach*	*Abbreviation*
Parametrisation:	Chord	PC
	Arc	PA
Control Lengths:	Analytic	CA
	Numerical	CN
Deviation:	Vector	DV
	Newton	DN

In addition, the amount of help-points introduced between the two given IK points varies from 1 to 4. These are denoted respectively as HP1, HP2, HP3 and HP4.

The first combination for a conversion algorithm is PA, CA, and DV. Utilising this, it is possible to develop the results table of Fig 3.9. The observations which are being noted include the number of iterations necessary to convert the IK defined character 'S' to a corresponding Bezier cubic description. Each iteration amounts to the employment of the three primary stages. It does not include the recursive process of evaluating the deviation and making successive improvements to t_i. In Fig 3.9, the worst deviation refers to the least acceptable value incurred by any of the arcs used in the representation. The desired accuracy ensures that the modelling arcs are within a tolerance of 15 units (where each unit corresponds to 1/100mm for a character bodysize of 15cm). The resulting average worst deviation of all these arcs is also listed.

It is clear from Fig 3.9 that the fastest conversion rate is returned by HP1, where only one help-point is introduced between two given IK points. This also uses the greatest amount of iterations to yield an acceptable representation. Although it appears that these two results are incompatible with each other, the fact that the conversion stages are working (in the case of HP2, HP3 and HP4) with greater amount of data points, accounts for this apparent disparity. This point is further highlighted in the case of HP3, where 84 less iterations are used than for HP1, yet it takes 36.6 seconds of additional CPU time to convert the given 'S' character.

PA,CA,DV	HP1	HP2	HP3	HP4
Total No of Bezier Arcs	15	14	13	15
Conversion Rate *CPU secs*	73.1	97.5	109.7	152.0
No of Iterations	377	345	293	353
Worst Deviation	14.2842	14.8591	14.6518	14.6566
Average Worst Deviation	10.7637	11.1445	11.4893	9.6564

Fig 3.9 Table of results for the IK to Bezier cubic conversion algorithm using the PA, CA, and DV combination.

Introducing three help-points, ie HP3, returns a Bezier description which employs the minimum number of curve segments. It uses one less than for HP2, and two less than for both HP1 and HP4. The placement of the knot and control points for all four cases are depicted in Fig 3.10. It is clear from this that the curve segments employed by HP1 and HP4, though the same in number, represent different parts of the given outline. This point is emphasised further by the average worst deviation value for both, as shown in Fig 3.9. Further analysis of this quantity highlights the fact that HP4 yields the most accurate approximation, with only an average of 9.6564 units of deviation between curve and the given set of IK points. Although this is desirable, it falls more than 5 units below the specified tolerance, resulting in more arcs being used than necessary. In other words, a Bezier description which has an average deviation close to the upper limit of the specified tolerance will employ the least number of curve segments (a fact indicated by HP2 and HP3).

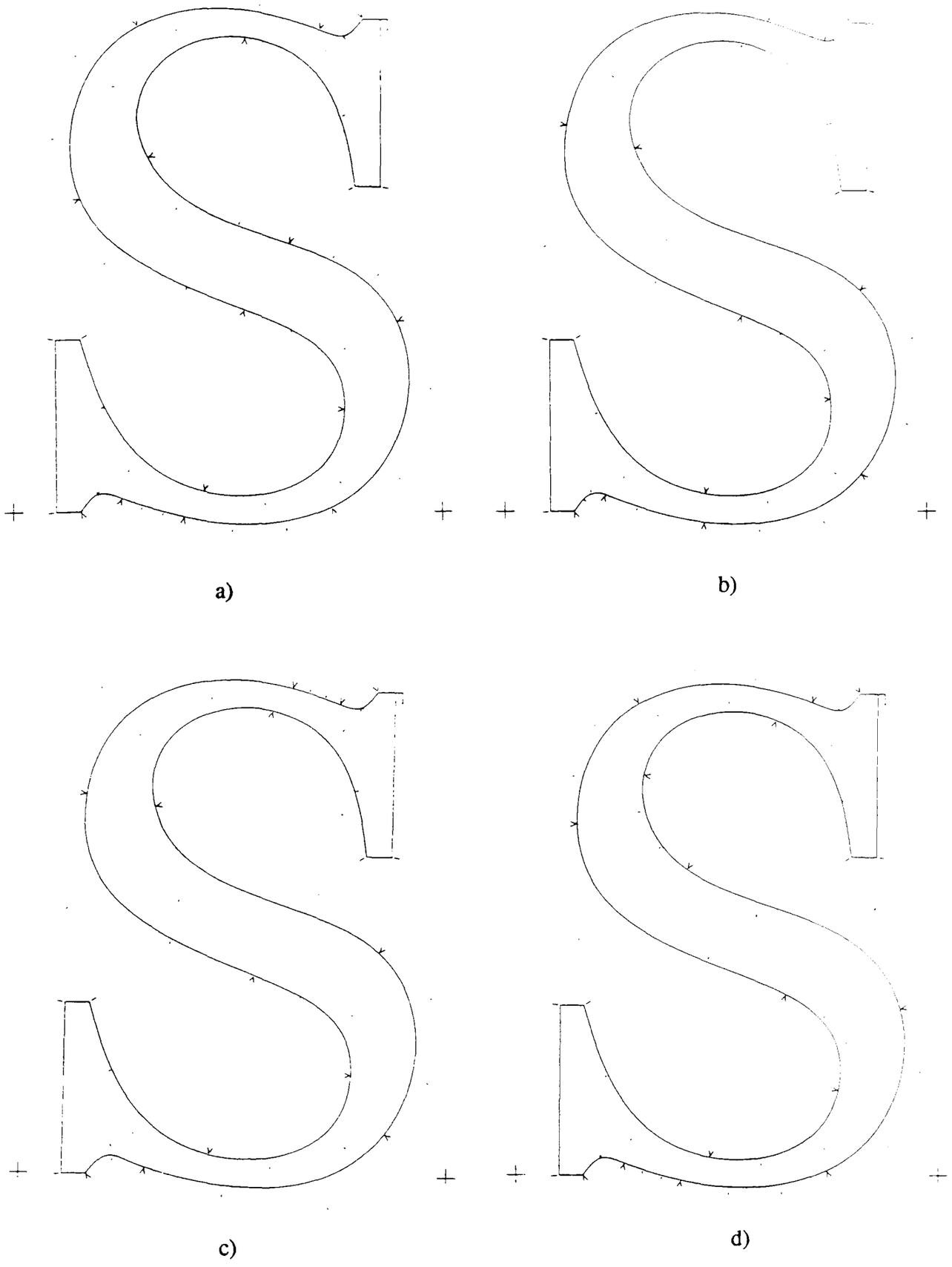


Fig 3.10 Depicts the outputs returned by the PA, CA, and DV conversion combination, employing a varied number of help-points:
a) 1, b) 2, c) 3, and d) 4.

The next conversion blend consists of PA, CN, and DV. Here, the analytical approach for evaluating the control positions is replaced by the numerical method discussed in section 3.4.2. The results obtained through this set-up for an IK to Bezier conversion are shown in Fig 3.11.

PA,CN,DV	HP1	HP2	HP3	HP4
Total No of Bezier Arcs	15	14	13	15
Conversion Rate *CPU secs*	65.4	89.1	97.6	138.9
No of Iterations	377	339	291	358
Worst Deviation	14.2839	14.8592	14.6517	14.6570
Average Worst Deviation	10.7637	11.1444	11.4893	9.6566

Fig 3.11 Table of results for the IK to Bezier cubic conversion algorithm using PA, CN, and DV combination.

As expected, the observations listed in Fig 3.11 are much similar to that gained through the analytical approach (Fig 3.9), the key difference being the rate of conversion. The numerical method returns a Bezier description in about 10 (CPU) seconds less than its analytical counterpart. Although this may not seem a considerable improvement, the fact is that if this was true for each of the characters in a font, then this would accumulate to a saving of about 15 to 20 minutes.

Changing the conversion combination to PA, CN, and DN, where the successive improvements to t_i are made using the Newton-Raphson method, leads to the observations tabulated in Fig 3.12.

PA,CN,DN	HP1	HP2	HP3	HP4
Total No of Bezier Arcs	15	14	13	15
Conversion Rate *CPU secs*	105.9	111.4	171.3	239.3
No of Iterations	449	369	352	413
Worst Deviation	14.5140	14.0847	14.7043	14.4712
Average Worst Deviation	10.5591	11.4139	11.1862	9.7357

Fig 3.12 Table of results for the IK to Bezier cubic conversion algorithm using PA, CN, and DN combination.

The Newton-Raphson method, as shown in Fig 3.12, does not effect the number of arcs employed in the Bezier description. As Fig 3.13 highlights, however, the location of the respective knot points are different from that shown in Fig 3.10. Further comparison of the tabulated results of Fig 3.9 and 3.11 shows that the solution rate of the Newton-Raphson is considerably slower than the vector approach. In the case of HP4, for example, it takes another 100 CPU seconds before a conversion is complete. Looking at the number of iterations necessary, the PA, CN, and DN combination requires on average 60 iterations more to yield a satisfactory outcome, a 15 per cent increase on both the previous combinations using the vector (DV) routine. Apart from these rather less desirable features, the use of the DN procedure does not seriously change the worst point-to-curve deviations, whether average or otherwise.

The next, and final, combination analyses the effect (if any) of using the chord length technique for evaluating the initial values for the parametric variable t_i. The conversion algorithm uses the procedures PC, CN, and DV. The results returned by this are recorded in Fig 3.14.

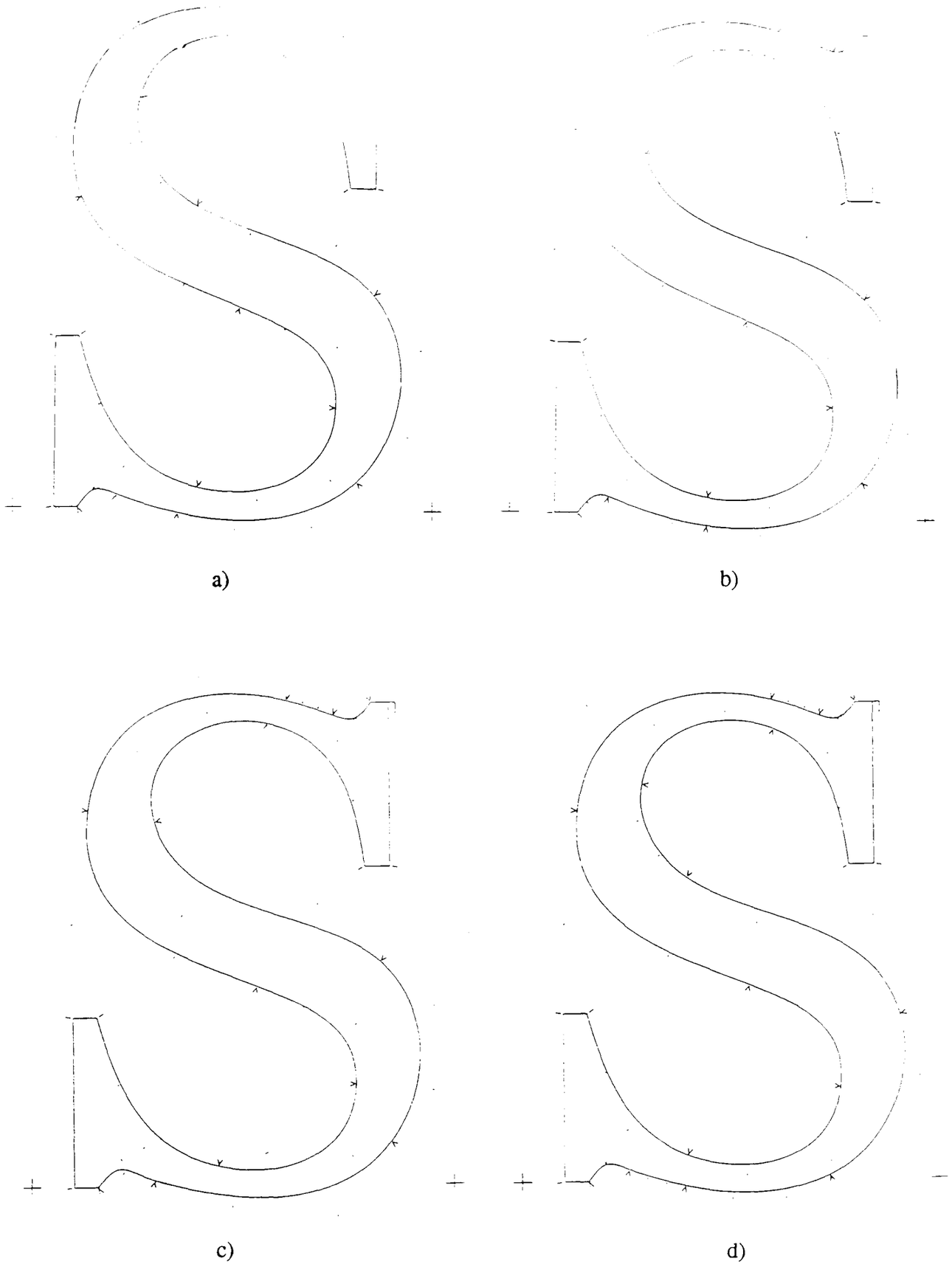


Fig 3.13 Shows the respective outputs gained through employing the PA, CN, and DN conversion combination for the following help-points:
a) 1, b) 2, c) 3, and d) 4.

PC,CN,DV	HP1	HP2	HP3	HP4
Total No of Bezier Arcs	15	14	13	15
Conversion Rate *CPU secs*	68.3	92.4	101.5	142.5
No of Iterations	371	338	289	362
Worst Deviation	14.3000	14.4706	14.7183	14.6634
Average Worst Deviation	10.1177	11.1543	11.5049	9.6526

Fig 3.14 Table of results for the IK to Bezier cubic conversion algorithm using PC, CN, and DV combination.

The results tabulated in Fig 3.14 when compared with those of Fig 3.9 show that it takes slightly longer for a solution to be realised; otherwise, there is little to choose from both set of results. This appears to indicate that the form of parametrisation (whether based on arc or chord length) does not effect the nature of the captured outline. Although this is valid in this case, for more diverse circumstances such as a high curvature contour (described in terms of a few data points), the results would have been drastically different (see [MANN 74]). In other words, the 'S' character (like all IKARUS fonts) is IK defined such that between two adjacent curve points the curvature does not vary significantly. This leads to the fact that any difference between the t_i values gained through the normalised chord length approach and, those acquired via the circular arcs, occurs only in the third decimal place. As the recorded results of Fig 3.13 show, this does not profoundly alter the solution parameters.

In order to summarise the many observations achieved through utilising the various conversion combinations, Fig 3.15 lists these with regards to possible

design specifications. Clearly, the tabulated results are in a form which would assist a designer in using a particular IK to Bezier algorithm.

Design Specification	Parameter Value	Number of help points	Conversion combination
Employs Least Number of Arcs	13 arcs	HP3	All
Returns the Fastest Conversion rate	65.4 CPU seconds	HP1	PA,CN,DV
Uses the minimum amount of iterations	289 iterations	HP3	PC,CN,DV
Exhibits the least IK to curve deviation	14.0847 units	HP2	PA,CN,DN
Yields the closest fit to given IK points	9.6526 units per arc (average)	HP4	PA,CA,DV
Utilises the most number of arcs	15 arcs	HP1 and HP4	All
Takes the longest time for a conversion	239.3 CPU seconds	HP4	PA,CN,DN
Requires the greatest number of iterations	449 iterations	HP1	PA,CN,DN
Produces the largest IK to curve deviation	11.5049 units per arc (average)	HP3	PC,CN,DV

Fig 3.15 lists the acquired observations in terms of possible design specifications.

With respect to the IKARUS requirements, discussed in section 3.3, it is apparent from Fig 3.15 that the most suitable conversion algorithm needs to employ the PA, CN, and DV procedures. This will ensure, depending on the number of help points used, a Bezier description which has the fastest rate of conversion and applies the minimum number of cubic splines.

3.5 Capture by Non-Parametric Form

In the previous section, the capturing algorithm employed a (non-rational) parametric form to model the given set of IK data points. Although this tends to be an acceptable way of gaining a Bezier cubic representation, working with a parametric variable t_i adds, as implied in section 3.4.3, to the computation costs. In addition, the iterative nature of the parametric approach (updating control positions and t_i values until a desired solution is gained) burdens further the rate of conversion. As an alternative to this approach, a method which incorporates the non-parametric form for the Bezier cubic is developed in this section.

The basic attraction of the non-parametric approach is that for a given set of data points, the control points are only evaluated once. The resulting Bezier spline would yield the "best" model for the data points. If it is found that the curve-to-point deviation is above a desired tolerance then the given IK points are subdivided and represented by two Bezier arcs.

In the following sections, the process of converting the non-rational parametric form for the Bezier cubic curve to the corresponding non-parametric is looked at: After an introduction, the process is utilised to produce a non-parametric representation for the Bezier cubic spline. The way this new form could be employed to capture mathematically the given IK points is also discussed. Finally, an ill-condition is highlighted which limits the application of the developed form. Possible solutions and consequences are presented through an illustrative example.

3.5.1 Introduction to the Form

A non-parametric formation can either be expressed as *explicit* or *implicit*. The explicit case takes the form $y=f(x)$. In this form, there exists only one y value for each given x value. For this reason, multi-valued curves (such as loops) cannot be represented by the explicit form. This limitation can be overcomed by employing the implicit form, which is expressed as $f(x,y)=0$. Although this

involves the calculation of roots to determine a point on a describing curve, it does provide a convenient way of evaluating whether a given point lies on a curve. Furthermore, the task of finding intersections, between curves and lines (a common requirement in the field of computer animation and graphics) can be simplified if implicit forms are available.

The process of transforming a parametric description into an implicit form is called *implicitisation* (and the process in reverse is termed *inversion*). Recent contributions to this field have been made by Sederberg: Some of the benefits of employing algebraic forms in describing curves and surfaces are given in [SEDE 83, 87]. He makes the point of utilising the implicit form as a tool for the Computer-Aided-Geometric-Design field. Together with Anderson, Sederberg elaborates on the way parametrically described curves and surfaces might be transformed into an implicit form [SEDE 84]. An example of converting the parametric form of a quadratic curve (ie a parabola) is given. An interesting article on the techniques of implicitisation and inversion with reference to rational cubic curves can be found in [SEDE 85]. An implicit representation which assists in locating double points of cubics is developed. (Double points are discussed in section 3.6.3).

The next section provides a derivation of an implicit representation for the non-rational parametrically describe Bezier cubic curve. Having produced this, a new (compact) implicit form is developed which yields a better insight to the geometrical nature of the defining Bezier polygon.

3.5.2 Process of Implicitisation

The mechanism for performing implicitisation makes use of elimination theory: The polynomials in t (parametric variable) are represented by a single equation $f(x,y)=0$, which is a polynomial in x and y. For the cubic case, the implicit equation, generally, consists of ten terms. However, with the application of scaling, there are in practice nine coefficients that need to be solved.

The first step in implicitisation is to formulate the parametric expression as a polynomial in t. That is, we require to translate the Bezier cubic expression given in equation (3.3) to the form used by Ferguson as in equation (3.1). By performing this translation, the coefficient vectors (a_0 to a_3) can be expressed in terms of the Bezier positional vectors (P_0 to P_3) as:

$$\begin{aligned} a_0 &= P_0 \ , \\ a_1 &= 3(P_1 - P_0) \ , \\ a_2 &= 3(P_2 - 2P_1 + P_0) \ , \ and \\ a_3 &= P_3 - 3P_2 + 3P_1 - P_0 \ . \end{aligned} \qquad ...(3.11)$$

To simplify the terms of equation (3.11), the Bezier curve can be translated such that its starting knot is at the origin (ie P_0=0). This will mean that the resulting values for the control points are gained with reference to this origin. The expressions for the parametric functions $x(t)$ and $y(t)$ describing the Bezier curve then take the form:

$$\begin{aligned} x(t) &= ax_1\, t + ax_2\, t^2 + ax_3\, t^3 \ , \\ y(t) &= ay_1\, t + ay_2\, t^2 + ay_3\, t^3 \ , \end{aligned} \qquad ...(3.12)$$

where: ax_1, ay_1, ax_2, ay_2, ax_3 and ay_3 are the respective (translated) x and y components of the a_1, a_2 and a_3 vectors expressed in equation (3.11).

Having prepared the parametric equations, as in (3.12), the following transformation determinant (see [SEDE 87] for a general description) is then established:

$$f(x_i,y_i) = \begin{vmatrix} ax_3 & ax_2 & ax_1 & -x_i & 0 & 0 \\ ay_3 & ay_2 & ay_1 & -y_i & 0 & 0 \\ 0 & ax_3 & ax_2 & ax_1 & -x_i & 0 \\ 0 & ay_3 & ay_2 & ay_1 & -y_i & 0 \\ 0 & 0 & ax_3 & ax_2 & ax_1 & -x_i \\ 0 & 0 & ay_3 & ay_2 & ay_1 & -y_i \end{vmatrix} = 0 \ . \qquad ...(3.13)$$

Computing the determinant given in equation (3.13) yields the following implicit form for the Bezier cubic:

$$f(x_i,y_i) = (w_1w_4 - w_2w_3)(w_3w_6 - w_4w_5) - (w_1w_6 - w_2w_5)^2 ,$$

...(3.14)

where:

$$\begin{aligned}
w_1 &= w_{12} - w_{13} ,\\
w_2 &= w_{13} - w_{23} ,\\
w_3 &= w_x y_i - w_y x_i - w_{12} ,\\
w_4 &= w_x y_i - w_y x_i - w_{13} ,\\
w_5 &= 3(x_2 - 2x_1)y_i - 3(y_2 - 2y_1)x_i ,\\
w_6 &= (x_3 - 3x_1)y_i - (y_3 - 3y_1)x_i ,\\
w_{12} &= 9(y_1x_2 - x_1y_2) ,\\
w_{13} &= 3(y_1x_3 - x_1y_3) ,\\
w_{23} &= 3(y_2x_3 - x_2y_3) ,\\
w_x &= x_3 - 3x_2 + 3x_1 ,\\
w_y &= y_3 - 3y_2 + 3y_1 .
\end{aligned}$$

(It is worth noting that if all the w terms are substituted in $f(x_i,y_i)$ then a factor of (w_1-w_2) emerges. Dividing equation (3.14) by this factor, therefore, simplifies the implicit equation for the Bezier cubic; a fact which is used to gain equation (3.16))

By examining the nature of the terms employed in equation (3.14), it is apparent that the coefficients of the implicit form can be computed from the coefficients of the parametric expressions by only using the arithmetic operators of addition, multiplication and subtraction. This, thus, ensures that the process of conversion is performed in exact integer arithmetic, with no numerical errors being introduced.

As mentioned earlier, the implicit form (as of equation (3.14)) can be used to compute whether a given data point (x_i,y_i) lies on the defining curve. If it does, then $f(x_i,y_i)$ will equal zero; otherwise $f(x_i,y_i)$ will return a non-zero value. The size of this residue gives an indication of how far the given data point is from

the curve. (This feature is made use of in chapter four for the general quadratic case where it is employed to return the curve-to-point deviation).

To calculate the "best-fitting" Bezier spline for a given set of data points (x_i,y_i), values for the two respective control points are required that will result in equation (3.14) returning the smallest residue values for each data point. In other words, we need values for the control points that make $\sum_{i=1}^{n} f(x_i,y_i)^2$ a minimum.

Rather than solving for the four unknowns (ie x_1, y_1, x_2 and y_2), the problem is reduced to finding solutions for two unknowns: With similarity to the parametric case (see section 3.4.2), the given tangents at the two knot points are used to set-up an implicit equation in terms of control lengths rather than control points. As illustrated in Fig 3.16, this then reduces the problem to finding the "best" values for the control parameters r and s. The control points for the Bezier cubic curve in terms of r and s can then be expressed as:

$$\begin{aligned} x_1 &= ru \ , \\ y_1 &= rv \ , \\ x_2 &= x_3 - s(x_3-u) \ , \\ y_2 &= y_3 - s(y_3-v) \ . \end{aligned} \qquad \ldots(3.15)$$

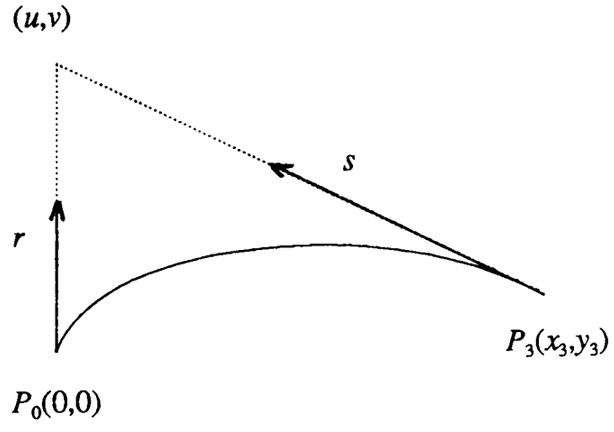


Fig 3.16 Depicts the role of the two control parameters, r and s.

The control parameters are normalised with respect to the intersection point (u,v) of their tangents. This results in them ranging from zero to one up to (and including) the intersection point. Within this range (see section 5.4.3 for illustrative examples), elliptic, parabolic and hyperbolic curve shapes can be produced. Taking values for both r and s beyond the intersection point will result in extreme hyperbolas, two inflection point curves, cusps and loops being generated. When one of these parameters has negative values with respect to the other, a Bezier curve containing one inflection point will result.

In order to work with the control parameters, the implicit equation for the Bezier curve is transformed to work with r and s rather than with the control points (x_1, y_1, and x_2,y_2). In undertaking this, it is found that the resulting (new) implicit form is more compact (than equation (3.14)) and, more importantly, it expresses the control parameters explicitly so that a better acquisition of the mathematical form in relation to the resulting Bezier curve can be made. If we denote e_i as the new implicit form which returns the residue for a given set of IK data points (x_i,y_i), then it can be shown that this takes the form:

$$e_i = \chi^3 + \lambda\,[3\Delta\,\Delta_i r^2 - \Delta_3^2(1-s) - 3\Delta_i r^2(\Delta_i-\Delta_3)] - \mu\,[\Delta_3(2-r-2s) + \Delta_i(s-r)]\ , \qquad ...(3.16)$$

where:

$$\chi = 2\Delta_3(2-3s) + 6\Delta_i(s-r)\ ,$$
$$\lambda = 72\Delta(3s-3s^2-r)\ ,$$
$$\mu = 216\Delta\,\Delta_i(s-r)^2\ ,$$
$$\Delta = \frac{vx_3-uy_3}{2}\ ,$$
$$\Delta_i = \frac{vx_i-uy_i}{2}\ ,$$
$$\Delta_3 = \frac{y_3x_i-x_3y_i}{2}\ .$$

The condition for r and s to acquire "best" values is then achieved through minimising the respective residues for the IK points. This is gained when:

$$\sum_{i=0}^{n} \frac{\partial (e_i)^2}{\partial r} = 0 \ , \ and$$

$$\sum_{i=0}^{n} \frac{\partial (e_i)^2}{\partial s} = 0 \ . \qquad \ldots(3.17)$$

Having performed and organised the expressions as required by equation (3.17), we are then in a position to solve for the two unknowns. As the resulting terms are non-linear in both r and s, a software routine which returns the roots of the non-linear expressions is needed. For this purpose, a standard NAG routine such as (coded) C05NBF, which applies a technique developed by Powell [POWE 70], is used.

3.5.3 Analysis and Observations

If an attempt is made to gain the "best" values for the control points using the approach described in the previous section, it will be found that the algorithm converges to different solutions depending on the starting values for r and s. This might be as expected, but for the fact that even for seemingly simple data, the solutions which are converged to tend to be out of range. Only after further investigations and tests with some experimental data, did it become apparent that the reasons the algorithm is "mis-behaving" should have been obvious all along, "though it never is":

The problem lies with the implicit form for the Bezier cubic curve. Although it is not obvious looking at equation (3.14), the compact form as given by equation (3.16) highlights an ill-condition for, when $r = s = 2/3$, the corresponding variables χ, λ and μ (of equation (3.16)) all become zero. This makes the entire expression, and thus the residue e_i, zero as well; implying therefore that a solution exists for this case. As chapter five (section 5.4.1) demonstrates, the resulting Bezier curve for $r = s = 2/3$ is a parabolic arc. The first conclusion which can be made, therefore, is that the implicit form for the Bezier cubic cannot represent the quadratic case of a parabolic curve.

The consequence of this ill-condition are rather severe. These are not just limited to the parabolic case, but affect *all* approximations which employ the implicit form. This is illustrated through an example: Given a Bezier polygonal (triangular) set-up as shown in Fig 3.17, a solution is desired that passes through the two knot points ((0,0) and (28,17) respectively) and approximates (or even interpolates) the two curve points, (9,7) and (19,13). It is expected, in this case, that the "best" values for both r and s will be positive.

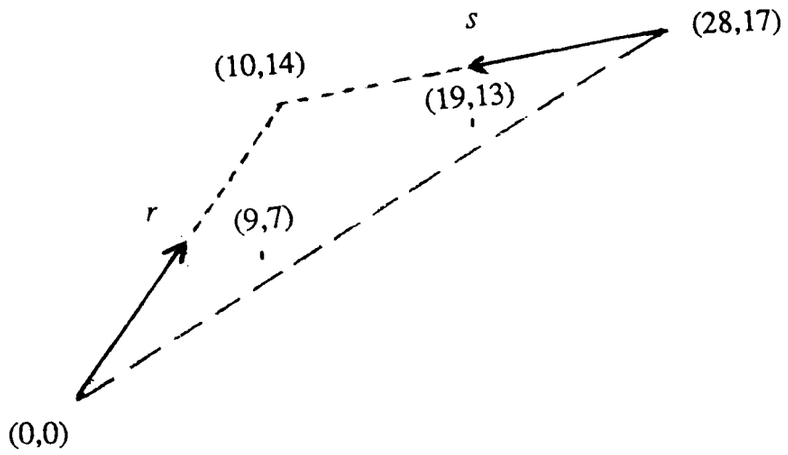


Fig 3.17 Shows the Bezier framework required for the given set of four data points.

If we trace-out the locus of the minimum values for the residue (e_i) for the two given curve points (as r and s are stepped from -0.6 through to 1.0), a graph similar to that shown in Fig 3.18 will result. Curves A and B provide the actual respective solutions; whilst curves C and D are due to the ill-condition and, therefore, amount to unwanted respective solutions. If a point on curve A is chosen, for example, this would ensure that the resulting Bezier curve will pass through the data point (9,7). Choosing a point on curve B, will guarantee the Bezier curve interpolates the given point (19,13).

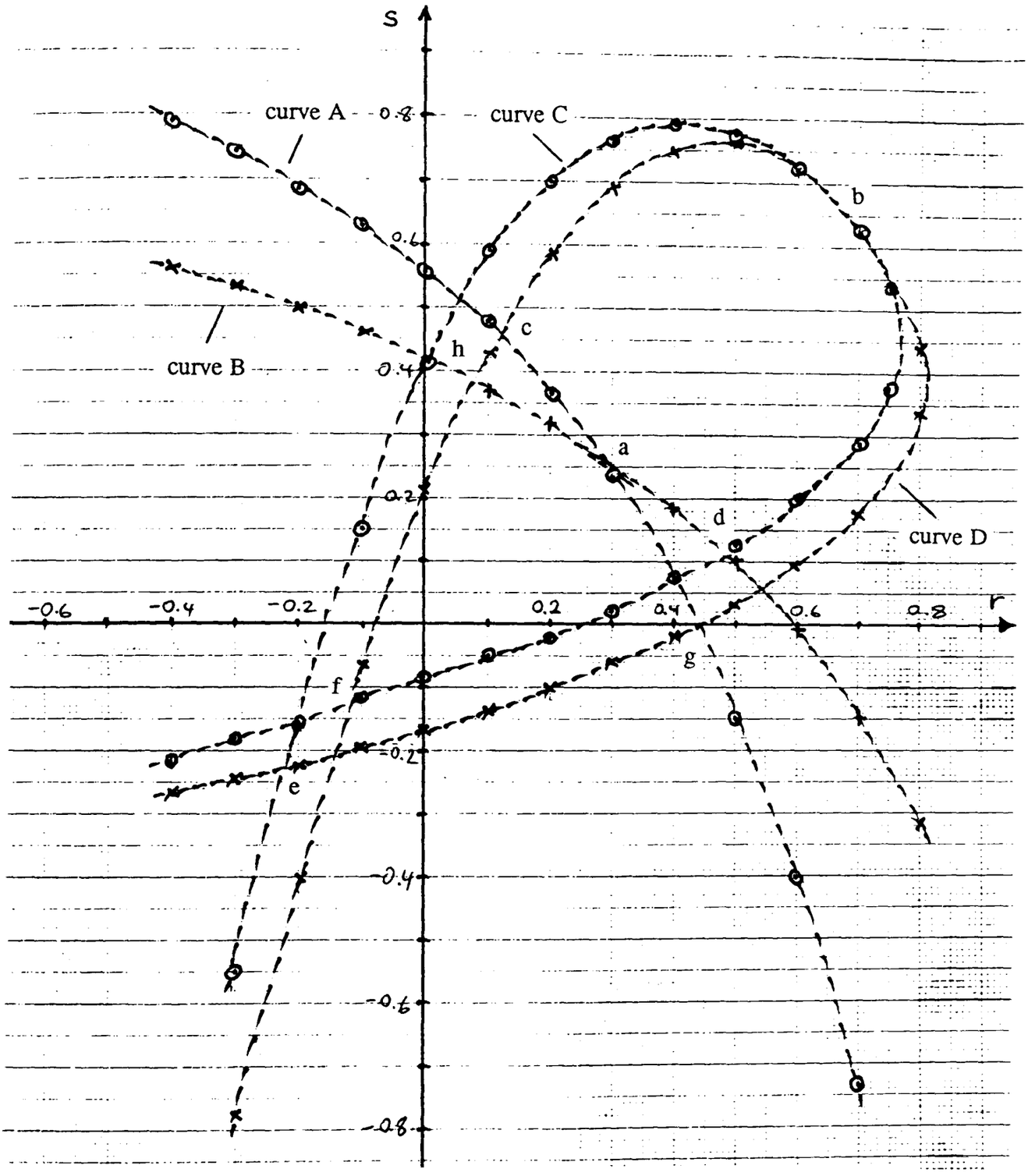


Fig 3.18 Highlights areas of minimum residue (potential solutions) for the given set of data points.

If we decide to select a point from curves C or D, this will result in the interpolation of the respective given data point, but not in the manner nor by the expected shape of the Bezier curve. Developing this subject matter a little further by requiring that the resulting Bezier arc interpolates the two given curve points, it is found that there are eight possible combinations for r and s that can be interpreted as a solution. These occur where the curves of point (9,7), A and C, intersect with the curves of data point (19,13), B and D. The eight possibilities are indicated in Fig 3.18, and the resulting solution values are tabled in Fig 3.19. The desired solution is found to be where curves A and B intersect; the remaining interpolate the data points, but not with the expected Bezier curve shape.

IK to Bezier cubic (non parametric)	Respective Values for the Control Parameters	
Solution Number	r	s
a	0.2696328	0.2799311
b	0.6666667	0.6666667
c	0.1172791	0.4562553
d	0.4793482	0.1193226
e	-0.2181235	-0.2285905
f	-0.1208993	-0.1281465
g	0.4362405	-0.0007040
h	0.0094888	0.4170311

Fig 3.19 Lists the numeric values for the eight possible solutions returned by the non-parametric Bezier algorithm, for two given data points.

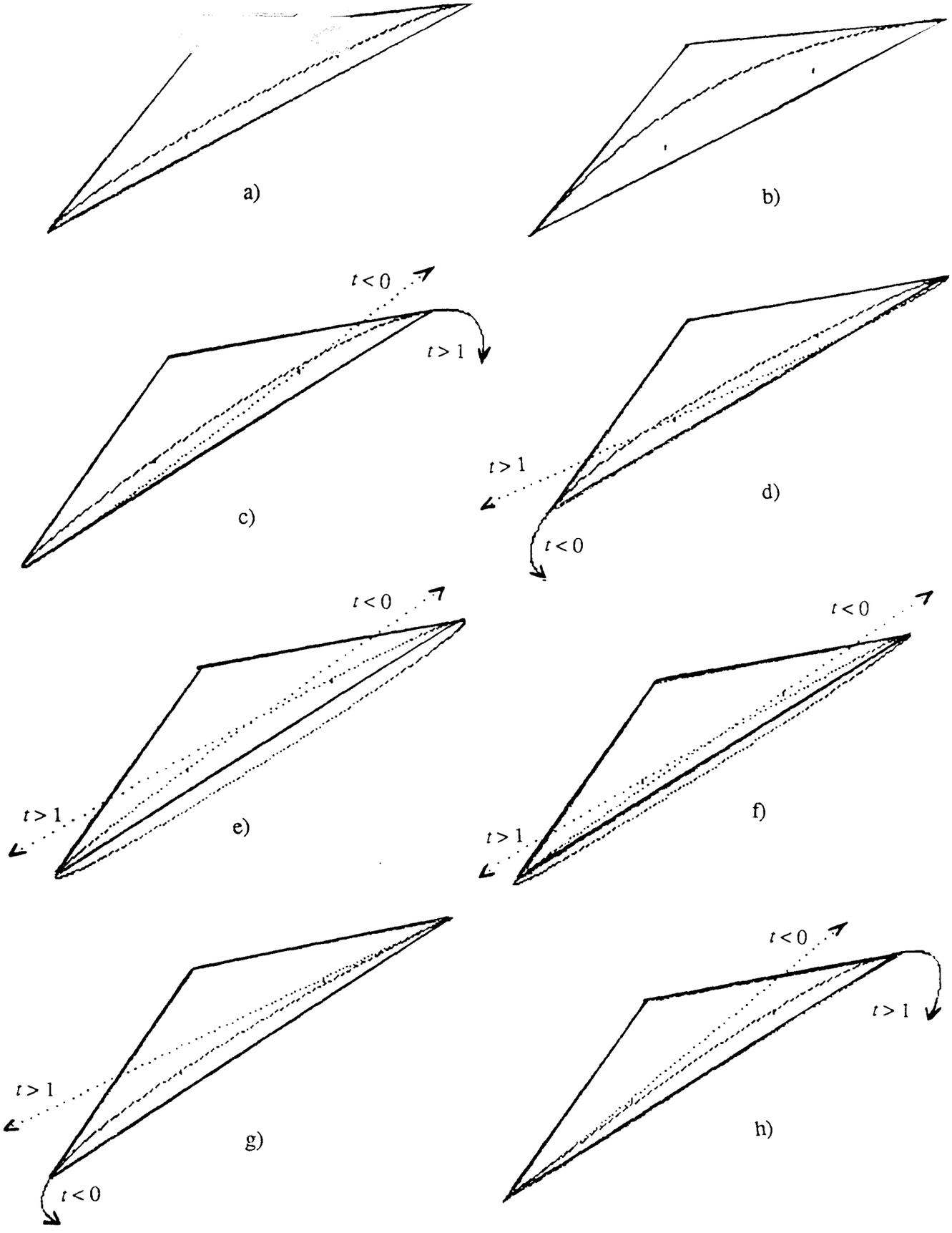


Fig 3.20 Gives an illustration (with reference to Fig 3.19) of how the eight solutions capture the given set of data points.

In Fig 3.20, a graphical illustration of the shapes listed in the table of Fig 3.19 is given. It is apparent from this that the shape of the Bezier curve depends directly on which of the solutions are converged to. If, for example, we look at the intersection points of curve A with curve C (Fig 3.20c and 3.20g), it is observed that the resulting Bezier curve (as expected) goes through the point (9,7), but interpolates the data point (19,13) either with very large negative or positive values for the parametric variable (that is, for $t<<0$ or $t>>1$). Comparing the graph of Fig 3.18 with the corresponding shapes for the Bezier curve given in Fig 3.20, it is possible to gain some insight into the form of the Bezier curve from the location of the various solutions. With reference to Fig 3.21, the shape of the possible solutions can be summarised as follows:

a) The desired solution is found at (1), where the given data is interpolated for parametric range $0<t<1$,

b) A solution at (2) will mean that the curve will pass through one of the points between $0<t<1$ and interpolate the other for $t<<0$,

c) A solution at (3) will result in the Bezier curve passing through one of the points for $0<t<1$ and interpolating the other for $t>>1$, and

d) A solution at (4) will interpolate one of the data points for $t<<0$ and the other for $t>>1$.

It is clear from the detailed analysis presented, that if we wanted to employ the implicit form for the Bezier cubic, the initial approximation for r and s needs to be quite near the final solution for an acceptable outcome. Possible initial values could be constrained to lie between the unwanted solutions, as in (1) in Fig 3.21. This will only cater for shapes that can be termed elliptic. Other shapes such as hyperbolic will normally require that r and s be greater than 2/3, implying that if the initial values are similar to the elliptic case then we are bound to get an

unwanted solution. As an alternative, the given set of data points could initially be described in terms of a general quadratic form in order to quantify the given outline, but this will limit the flexibility of the Bezier description to representing outlines which do not possess points of inflections.

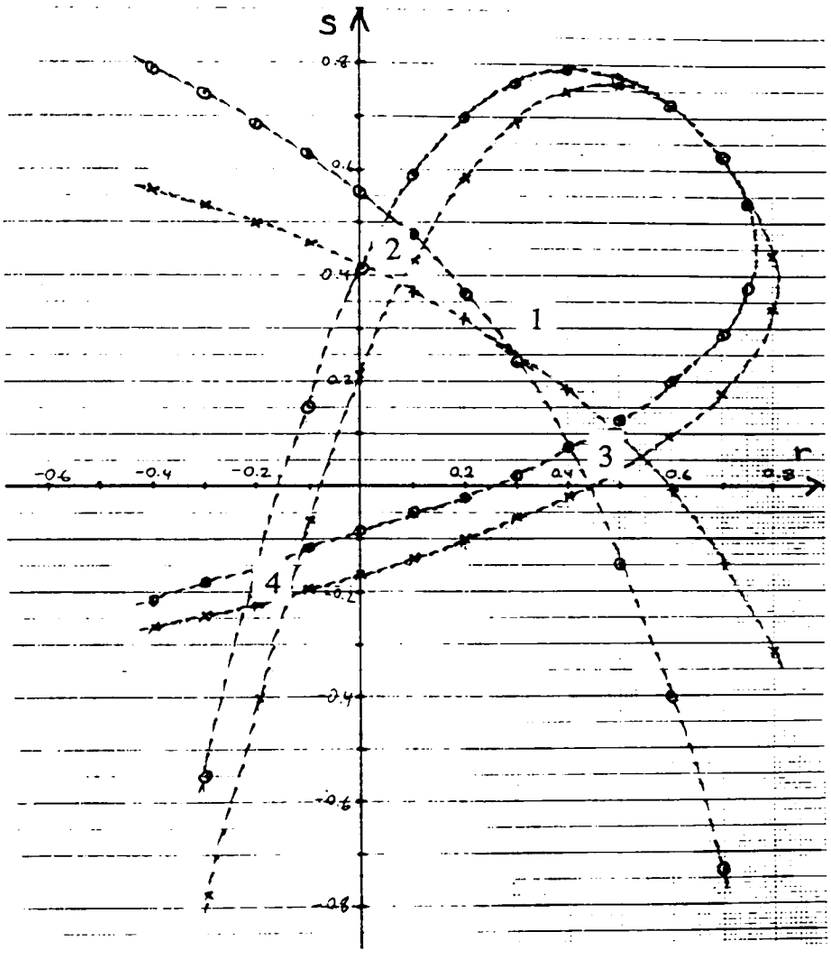


Fig 3.21 Highlights four areas of potential solutions.

In passing, it should be pointed out that this unwanted behaviour of the conversion algorithm is avoided if the two control parameters are constrained to be the same. This means that *r* and *s* take on similar values, resulting in the Bezier cubic being able to represent only conic-like shapes (see [PITT 91], a paper written by Pitteway and myself). Although this assists in that the conversion process is no longer inhibited by the parabolic ill condition, gaining a Bezier description which can only represent conic shapes is rather unsatisfactory. If the cubic spline is to be limited in such a manner, then clearly an approach which gives a Bezier cubic description through the general conic

could be employed (see section 5.4, where algorithms for converting general conic to Bezier cubic are presented). This at least has the advantage of removing some of the non-linearities encountered in the above approach.

Finally, it needs to be emphasised that the ideal solution to the problem is to develop an implicit form for the non-rational Bezier spline that does not contain the ill-condition. The approach discussed in this section gives a basis for such implementation. It requires further development to remove and/or limit the effects of the unwanted parabolic solutions.

3.6 Displaying Cubic Arcs

Thus far, most of the discussion has concerned itself to the process of capturing the given IK data points in an acceptable Bezier cubic form. Some general remarks in section 2.4.4 were made regarding the aspect of displaying contours of outlines. In this section, the case of representing Bezier cubic arcs on a digital display is examined. The aim is to have a working algorithm that could be employed in the IKARUS system. Discussion is confined to analysing two approaches, one based on the parametric form and the other on the non-parametric form. In each case, an algorithm is developed whose performance is analysed through given examples. A condition where the non-parametric algorithm "misfires" is highlighted, and possible solutions discussed.

In the next section, the parametric algorithm is presented. The non-parametric algorithm is discussed in section 3.6.2.

3.6.1 Parametric Algorithm

The parametric form has, as part of its characteristics, the ability to return a curve point $(x(t_i),y(t_i))$ for a given t_i value. An outline could simply be generated by computing the curve points for various values of t_i ranging from zero to one. This approach will not suffice as different types of outlines will require a different amount of curve points. For example, a low curvature outline (one approaching a straight line) will require few curve points, whilst on the other extreme, a high

curvature curve (such as a semi-circle whose governing radius is relatively small) will require a large number of curve points so that its accurately displayed.

Deciding on the precise number of curve points forms the bases of most of the techniques that are used: Pavlidis [PAVL 82] conjectures about employing a curvature dependent approach. He also discusses the effects of rounding errors that occur when translating the curve points, returned by the parametric variable, into integer (grid) values. Mortenson [MORT 89] proposes the use of line segments to approximate the curved outline. Again, how many lines to use is the critical choice. He suggests the amount of lines employed should be based on the relative change in slopes. This further emphasises the point that an approach which adapts to the changing nature of the given contours is required.

The technique which is presented here is based on representing the given curve outline in terms of line segments. These lines are then drawn (ie rasterised) by an algorithm pioneered by Bresenham [BRES 65]. The characteristics of the polygon defining the Bezier arc are incorporated in the rasterisation process. The number of lines employed is based on a distance criterion. The algorithm in detail is as follows:

To convert the curved outline into line segments, the process of subdivision is used on a recursive basis. Subdivision has been introduced in section 2.2.3. It was originally looked at by De Castaljau. He formulated a scheme for the defining polygon which used repeated linear interpolation to return (Bezier) curve points. These principles are applied (and those of more recent authors such as [COHE 80, PAVL 82 and YAMA 88]) to develop a subdivision algorithm. Fig 3.22 illustrates the technique. By using this geometric set-up, it can be shown (see the above cited references) that for a given parametric variable value for splitting t_s, the following expressions for the resulting two Bezier arcs can be derived:

$$
\begin{aligned}
P_1^1 &= P_0 + t_s(P_1 - P_0)\ , \\
P_2^1 &= P_1 + t_s(P_2 - P_1)\ , \\
P_3^1 &= P_2 + t_s(P_3 - P_2)\ , \\
P_2^2 &= P_1^1 + t_s(P_2^1 - P_1^1)\ , \\
P_3^2 &= P_2^1 + t_s(P_3^1 - P_2^1)\ , \\
P_3^3 &= P_2^2 + t_s(P_3^2 - P_2^2)\ ,
\end{aligned}
\qquad \ldots(3.18)
$$

where: P_0, P_1^1, P_2^2, P_3^3 forms 1st half of Bezier arc,

P_3^3, P_2^3, P_1^3, P_3 forms 2nd half of Bezier arc.

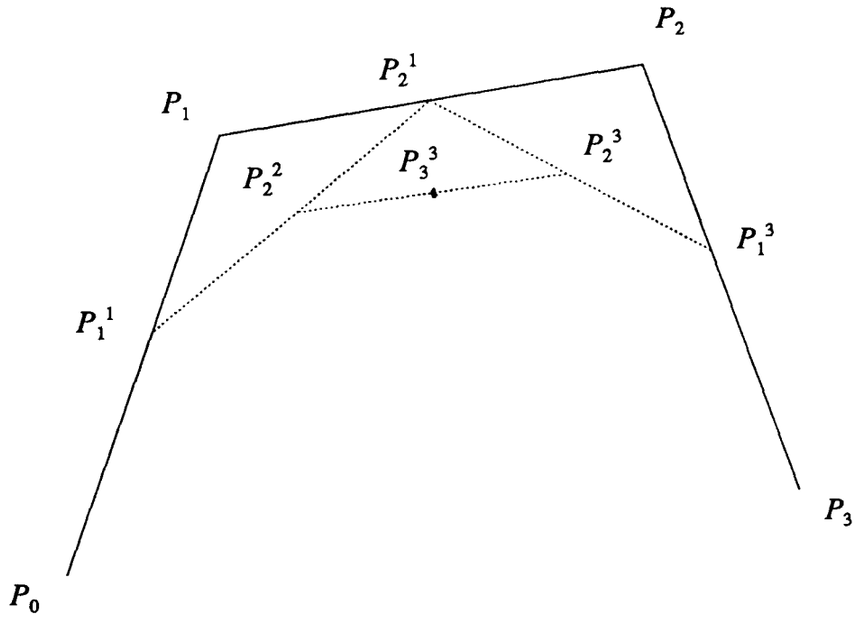


Fig 3.22 Illustrates De Casteljau's algorithm for subdividing a Bezier cubic spline.

It is apparent from equation (3.18), that the value of t_s can be chosen (ranging between zero and one) to split the defining curve at a desired point. The strategy, which the rasterising algorithm incorporates, is bisecting the Bezier curve at its mid-point (t_s=0.5). This process is repeated on a recursive basis (as and when necessary) until a desired approximation of the curve outline is achieved using line segments.

In order to have some control over the number of line segments being employed, a mechanism for determining when a desired approximation has been achieved needs to be developed. For this purpose, the polygonal set-up of the Bezier curve is used. The convex hull property of Bezier curves ensures that the generated arc always lies (for the parametric interval zero to one) within its defining polygon. It is, therefore, reasonable to assume that if the two control points of the Bezier curve are within a desired distance from the approximating line segment, then an acceptable fit has been gained. The main benefit offered by this approach is that it is relatively simple, but effective, and does not computationally burden the rasterisation process.

Although the performance of this algorithm is assessed together with that of the implicit approach (described in the next section), it is worth noting the fact that subdividing the curve into line segments results in non-integer knot positions. The algorithm is, therefore, liable to rounding error effects. Furthermore, the continuity between adjacent line segments is of zero order, that is, they are constrained only to touch each other. The consequence of these factors are analysed in section 3.6.3.

3.6.2 Implicit Algorithm

The approach described in the last section provides a solution to the problem of digitising an outline modelled by Bezier cubic arcs. The use of non-integer knot points (as mentioned) can lead itself to the effects of rounding errors. An algorithm is, therefore, required which will "track" the curve, returning the "best" pixel (integer) positions along the curve's outline. In this section, such an algorithm is developed that uses the implicit form to describe the defining Bezier cubic spline [HUSS 91]. The algorithm initially calculates the coefficient values for the nine (implicit) terms and then rasterises the curve outline based on similar principles as those employed by Bresenham for the case of digitising line segments [BRES 65]. The method in detail is as follows:

It is accepted that Bresenham's algorithm forms the best approach for drawing straight line segments of arbitrary gradients. The basis of this technique is to choose the closest mesh-point (ie pixel) to the line being generated. Pitteway applied the principles of this technique to develop algorithms to cater for rasterising outlines described by general quadratic splines [PITT 67, PITT 85]. He together with Botting, extended the approach to cater for cubic splines [BOTT 68]. Their algorithm at each stage chooses either to make a square-move or a diagonal move. By a square-move, it is understood that either the x or the y position is stepped by one; a diagonal-move requires both the x and y positions to be incremented by one. The algorithm is geared to work with eight octants. The quest for an effective rasterising algorithm for the cubic curve is re-addressed in this section. In order to reduce some of the computational overheads, the algorithm presented here works with four quadrants rather than the eight octants. Furthermore, by making square-moves only, the algorithm ensures that all the pixels intersected by the Bezier cubic curve are visited.

In line with the implicit forms used by both Bresenham and Pitteway, the Bezier cubic curve can be expressed as follows:

$$d(x,y) \equiv k + 2vx - 2uy - \alpha y^2 - \beta x^2 - 2\gamma xy - \frac{rx^3}{3} - \frac{sy^3}{3} - px^2y - qxy^2 = 0 \; . \qquad \ldots(3.19)$$

The coefficients of equation (3.19) can be found from either of the two implicit forms given by equations (3.14) and (3.16). Appendix A3.1 gives the expressions for the coefficients in terms of equation (3.14).

The concept of the rasterising algorithm can be summarised as follows: If we denote the current (pen) position as i and j respectively, then the value of d (of equation (3.19)) is evaluated at the mid-point of the next possible square-move, that is at $i+½$ and $j+½$. If $d < 0$ at this point the curve passes below, and a x-step is called for; otherwise $d > 0$ so that the curve passes above, and therefore a y-step is required. If, in the rare case, $d(i+½,j+½)$ happens to be zero, then either move will suffice.

Incorporating these basic concepts results in the working algorithm shown in Fig 3.23. As is apparent from this, the algorithm includes two further tests that enable it to track cubic curves which might require quadrant changes. The two tests, therefore, detect and take appropriate action to ensure that the algorithm continues to follow the contours of the given outline. If we take the example of a circle with its four extreme points (top, bottom, left and right), then the "a-test" detects the left and right occurrences; whilst the "b-test" caters for the top and bottom incidents (for a detailed discussion see [PITT 85]).

The cubic algorithm in its inner (main) loop requires just six add operations for each x or y move. The code for a quadrant change is made modest by carefully choosing (as shown in Fig 3.23) the place where it re-joins the main loop. The algorithm is geared to stop when the end knot positions (for a given Bezier cubic curve) have been reached. It can be seen that as a whole the algorithm has the attraction of employing simple algebra which should make it computationally efficient.

The initial conditions for the algorithm can be gained by means of finite differences. It can be shown that for the Bezier cubic case, these can be expressed as follows:

$$
\begin{aligned}
L_1 &= 2r \,, \\
L_2 &= 2p \,, \\
L_3 &= 2q \,, \\
L_4 &= 2s \,, \\
K_1 &= 2\beta - r + p \,, \\
K_2 &= 2\gamma \,, \\
K_3 &= 2\alpha - s + q \,, \\
a &= 2u + \gamma + \frac{s}{12} + \frac{p}{4} \,, \\
b &= 2v - \gamma - \frac{r}{12} - \frac{q}{4} \,, \\
d &= k - \frac{\alpha}{4} - \frac{\beta}{4} + \frac{\gamma}{2} - \frac{a}{2} + \frac{b}{2} \,.
\end{aligned}
$$

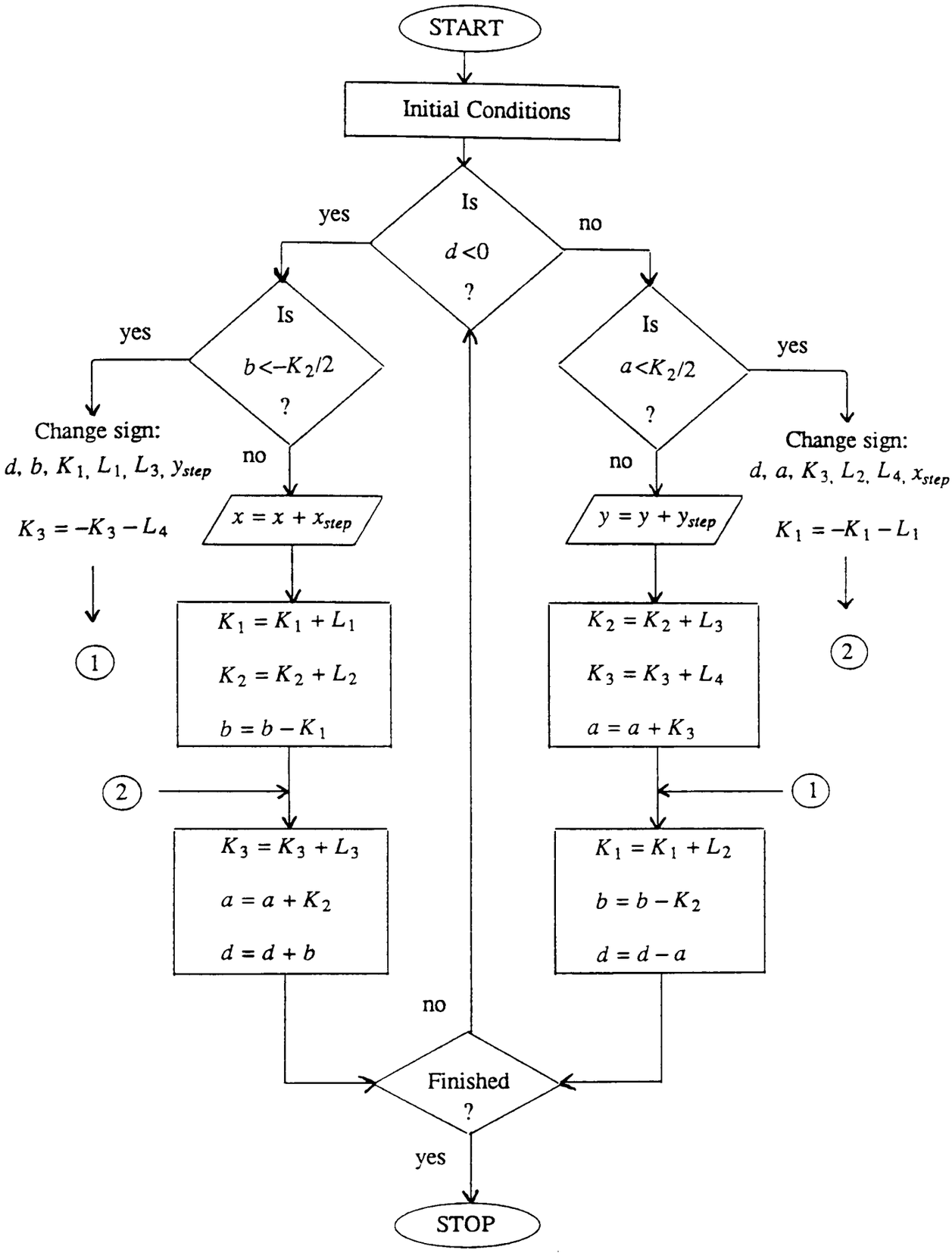


Fig 3.23 Depicts the rasterisation algorithm for the Bezier cubic, which permits square-moves only.

As mention in section 3.5.3, the implicit form of the Bezier cubic curve (as given by equation (3.16)) does not cater for the parabolic case. This, therefore, requires

detection *before* attempting to use the cubic rasterising algorithm. If, and when, a parabolic arc is detected then the cubic terms p, q, r and s need to be set to zero, and the values for u, v, α, β and γ will require calculation. The control point for the parabolic (ie u and v) can be gained via the tangents at the two knot points, and the other three variables by using the relevant expressions of equation (4.4) of section 4.2.3. Once these have been evaluated then the cubic rasterising algorithm can be employed. (Note: The general conic rasterising algorithm of Fig 4.9, section 4.4, could be used to gain exactly the same results).

Finally, it should be realised that the cubic rasterising algorithm is developed in-view of the demand for representing outlines of font characters. These, on average, use Bezier curves that are slow-moving (low curvature), and which normally do not employ sharp-cornered arcs that might lead to the cubic algorithm getting "confused" and losing its way. The algorithm, as given, forms the basis of rasterising cubic outlines, and as will be highlighted in the next section, requires further development for it to be commercially viable.

3.6.3 Analysis and Observations

To assess the performance of the two rasterising algorithms presented in the previous two sections, various curve shapes (generated using a Bezier cubic polygonal framework) are supplied as input. The effectiveness of each algorithm is evaluated in terms of the amount of (CPU) time consumed and, probably more importantly, the way the distinct features (smoothness and curvature) are digitally represented. A problem case for the implicit (tracking) algorithm is highlighted. A novel method for detecting such occurrences is presented, and possible ways of solving the problem are discussed.

Before discussing the results, it is worth noting that these were produced using a SUN 3/50 workstation, with a screen resolution of 900 by 1152 pixels. The corresponding hard-copies were obtained by means of a screen-to-postscript (suntops) routine, making the screen output suitable for a (APPLE) laser printer. The point to realise is that the laser printer exhibits a resolution which is about

three times greater than the screen display. This has a "damping" effect, therefore, on the respective hard-copies of the rasterised outlines.

The digitised outlines using the parametric (line) approach are depicted in Fig 3.24. These are produced using a control point to line tolerance (as discussed in section 3.6.1) of 0.1 units. This value appears to be most suitable in terms of conversion rate and effectiveness. The corresponding implicit (track) algorithm's performance is shown in Fig 3.25.

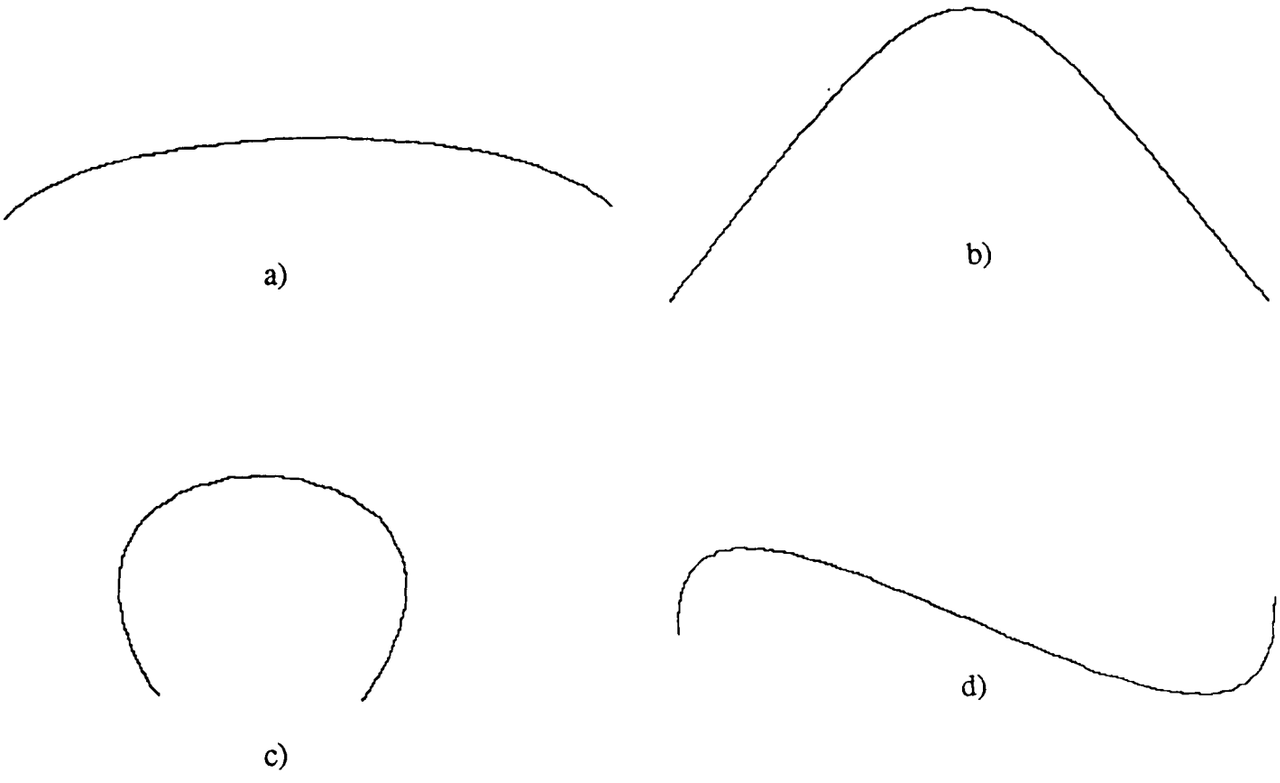


Fig 3.24 Shows given outlines digitised by the parametric (line) algorithm:
a) an elliptic curve, b) a hyperbolic arc, c) a "circular" curve, and
d) an arc containing an inflection point.

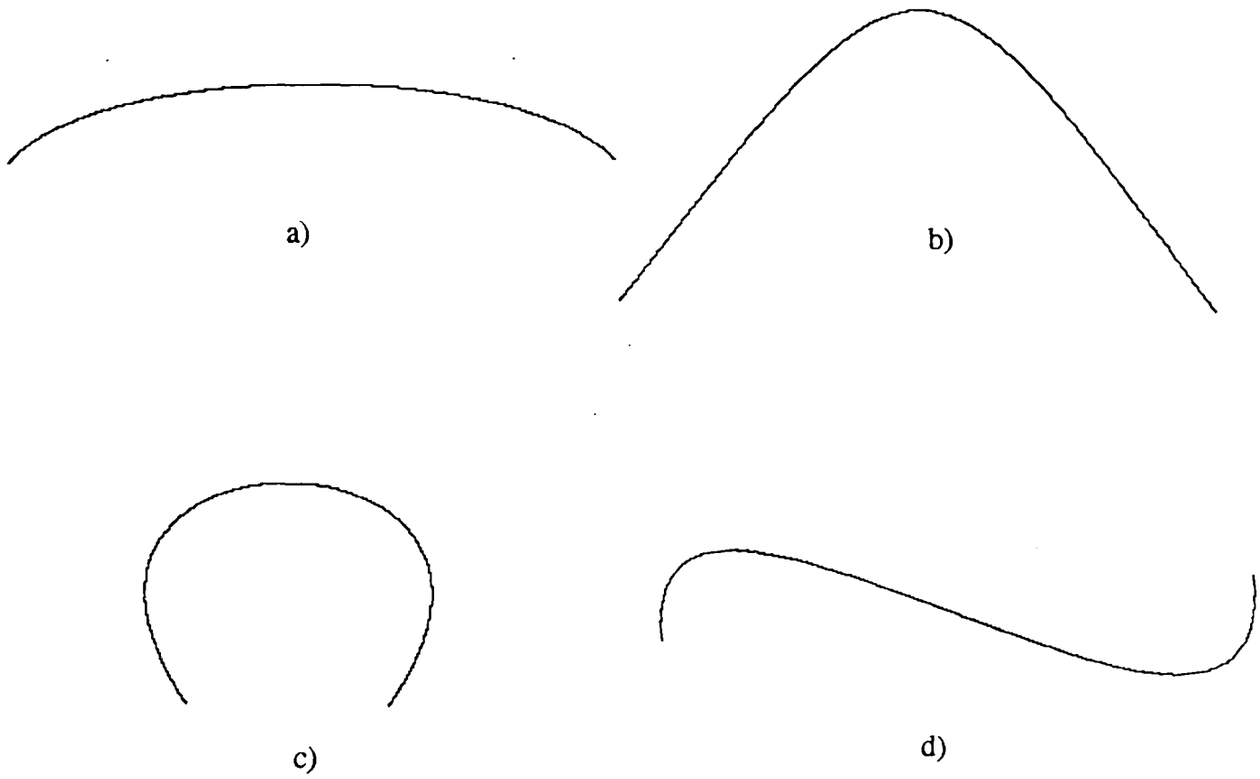


Fig 3.25 Depicts the rasterised output returned by the implicit (track) algorithm for the respective cases of Fig 3.24.

Comparing the two outputs, it is evident that the tracking approach, as far as representation is concerned, performs much better than the line method: The elliptic and hyperbolic arcs (Fig 3.24a and 3.24b respectively) show an apparent "uneasiness" of the parametric algorithm to render curves of both low and high curvature. This is highly visible when the hyperbolic output of Fig 3.24b is compared with that of Fig 3.25b. The implicit approach yields a "digitally smooth" outline, whilst the same Bezier curve appears much more rugged in the case of the line algorithm. Figs 3.24c and 3.24d (together with their counterparts of Fig 3.25) illustrate the efficiency of the algorithms in representing Bezier curves which are circular in appearance and, those containing a point of

inflection. Clearly, both approaches are able to rasterise these types of arcs, the track method, again, producing the most aesthetically acceptable output.

As far as conversion rates are concerned, Fig 3.26 lists the CPU times consumed by each of the examples depicted in Figs 3.24 and 3.25. Also tabulated are the number of line segments used by the parametric algorithm.

Respective curve of Fig 3.24 & 3.25	**Implicit Algorithm**	**Parametric Algorithm**	
	Conversion rate *CPU seconds*	Conversion rate *CPU seconds*	Number of line segments
a	28.9	28.9	42
b	29.2	29.0	50
c	28.9	29.5	74
d	29.4	29.3	52

Fig 3.26 Table of results for the two Bezier cubic rasterisation algorithms.

The observations recorded in Fig 3.26 clearly show that both approaches utilise, on average, similar conversion times. This verifies the fact, therefore, that for the parametric algorithm a distance of 0.1 units between the Bezier control points and the approximating line leads to conversion rates which are comparable to the implicit approach. Changing the distance value will alter both the speed of conversion and, also, the number of line segments employed. Although more line segments results in a closer approximation, it does not, necessarily, improve the appearance of the rasterised outline. As mentioned earlier, rounding errors make the digitised contour look rather coarse. For this reason, the tracking algorithm tends to be more acceptable, as it chooses the nearest mesh-point to the actual curve.

Although the implicit approach appears to be ideally suited for rasterising Bezier cubic arcs, it tends to get "lost" for what seems to be (as far as the Bezier

polygonal framework is concerned) a simple, straight forward, looking curve. The problem is illustrated through the example shown in Fig 3.27: Given the elliptic looking arc of Fig 3.27a, the algorithm starts following the curve in the direction of A. At a point C, it gets "confused" and loops back to give the illustration of Fig 3.27b. If the start and end knots are reversed, so that the algorithm now follows the direction of B, it again loses its way at point C. Fig 3.27c depicts this situation graphically. A closer examination of the cubic curve (Fig 3.27d) shows that this contains a self-intersecting point (point C, called a *crunode* in the field of mathematics). The algorithm, as given in section 3.6.2, fails to track the given Bezier curve properly, and performs an unwanted quadrant change at point C, resulting in the two cases shown in Fig 3.27b and 3.27c.

The reason for this failure can be easily understood: It is well known that a general cubic equation in an unknown in x has at least one and at most three real roots [PATT 88]. It follows, therefore, that the cubic curve in x and y described by equation (3.16) will cross any given straight line either once, or three times. The relevant straight line here extends from the start knot and finishes at the end knot (ie the baseline of the Bezier polygon). The cubic is known to cross this line at both knot points, so there has to be a third crossing. This, as shown in Fig 3.27d, occurs at a relative distance t_c measured along the baseline, where:

$$t_c = \frac{r^2(r + 3s^2 - 3s)}{(r-s)^3} . \qquad ...(3.20)$$

In the case shown in Fig 3.27d, r=0.19444, s=0.58333, so t_c=0.34374. Any values of t_c ranging between zero and one will result in the third crossing of the baseline confusing the cubic, in the manner described above.

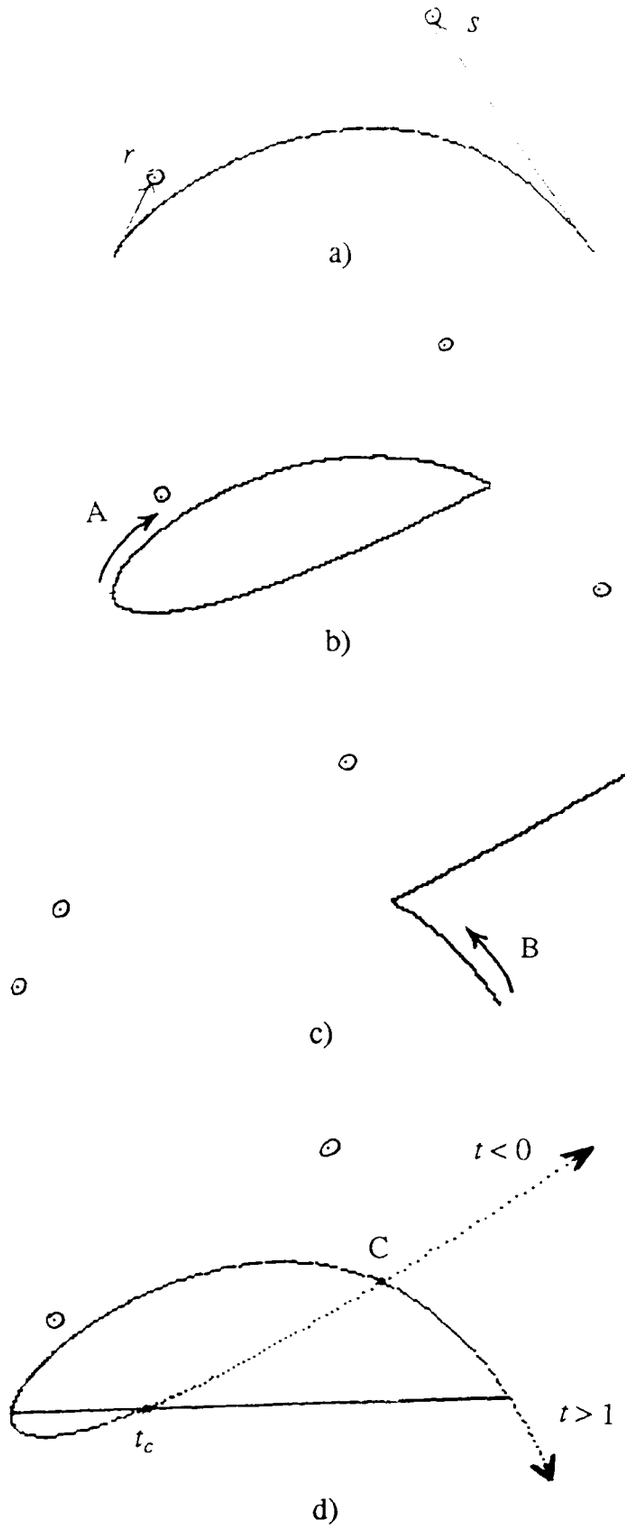


Fig 3.27 Gives an example of how the tracking algorithm gets confused and subsequently loses control at point C:

a) the given arc,

b) following direction of A,

c) following direction of B, and

d) shows that at point C, the cubic self-intersects.

To use the implicit algorithm of Fig 3.23, it is necessary, therefore, to scan all the given Bezier described outlines in order to detect (by using equation (3.20)) whether any arcs will confuse the tracking process. If such an arc exists, then the following possible remedies are available: The first consists of making small adjustments to r and s until the third crossing point occurs outside the harmful range for t_c of one and zero. This approach, however, leads to an iterative process for finding suitable values for the control parameters. Furthermore, as Fig 3.28 manifests, unless the r and s values are close to the unshaded (safe) areas, considerable adjustments have to be made to the values of either r, or s, or both; resulting in the Bezier spline changing its shape.

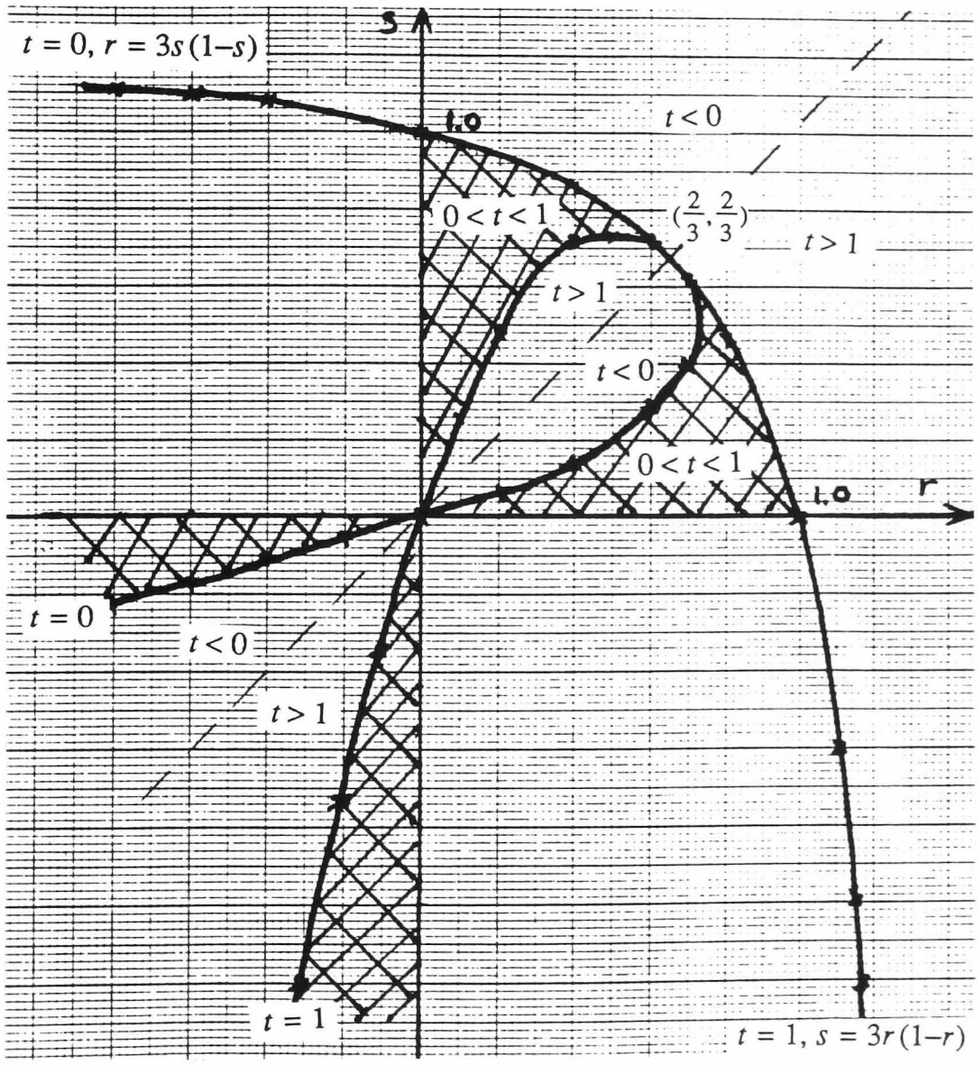


Fig 3.28 Illustrates safe areas (shown unshaded) for the implicit rasterisation algorithm, based on equation (3.20).

The second solution involves the employment of the line algorithm. This has the advantage of rasterising Bezier arcs based on the defining polygon. Any "troublesome" arcs could, therefore, be handled by the line algorithm.

The third, and probably the best, approach consists of subdividing the given Bezier curve at its self-intersecting point. By taking this point as a (new) knot point, the resulting two arcs will *not* contain any self-intersecting points within their respective parametric intervals, removing, therefore, the likely state of confusion for the rasterising algorithm. The calculation of the two values for the parametric variable which give the same curve point (double point), t_d, can be made through the following expression developed by Pitteway and myself [PITT 91]:

$$t_d = \frac{r(3s-2) \pm \sqrt{3rs(3rs-4r-4s+4)}}{2(3rs-r-s)} \quad . \qquad ...(3.21)$$

By choosing the value of t_d which lies between zero and one, it is possible to subdivide the supplied Bezier curve into two arcs. As this will effectively change the double point into a knot point, the algorithm would then be in a better position to perform the rasterisation without getting lost.

A closer examination of equation (3.21), in fact, leads to further classification of the type and form of the self-intersecting cubic. By using the expression $3rs$-$4r$-$4s$+4 of equation (3.21), a rectangular hyperbola can be incorporated in the r and s graph (of Fig 3.28) to quantify safe areas for the implicit rasterisation algorithm. This, as Fig 3.29 shows, has the effect of "fine-tuning" the existing boundaries and adding a new (cusp) region to the graph. In other words, the graph of Fig 3.29 represents all the possible Bezier cubic shapes, including that of the loop (a case not catered for by equation (3.20) as it does not intersect the baseline). This fact is verified by Fig 3.30, which depicts the corresponding self-intersecting cases for the cubic in relation to Fig 3.29.

In short, both the parametric and the non-parametric approaches can be employed to digitise contours, mathematically described by Bezier cubics. The

representation returned by the implicit form, however, has a number of benefits, including that of giving a superior rasterisation. Although this approach suffers from the self-intersecting property of a cubic spline, it is shown that through a simple process of quantification, this problem can be resolved.

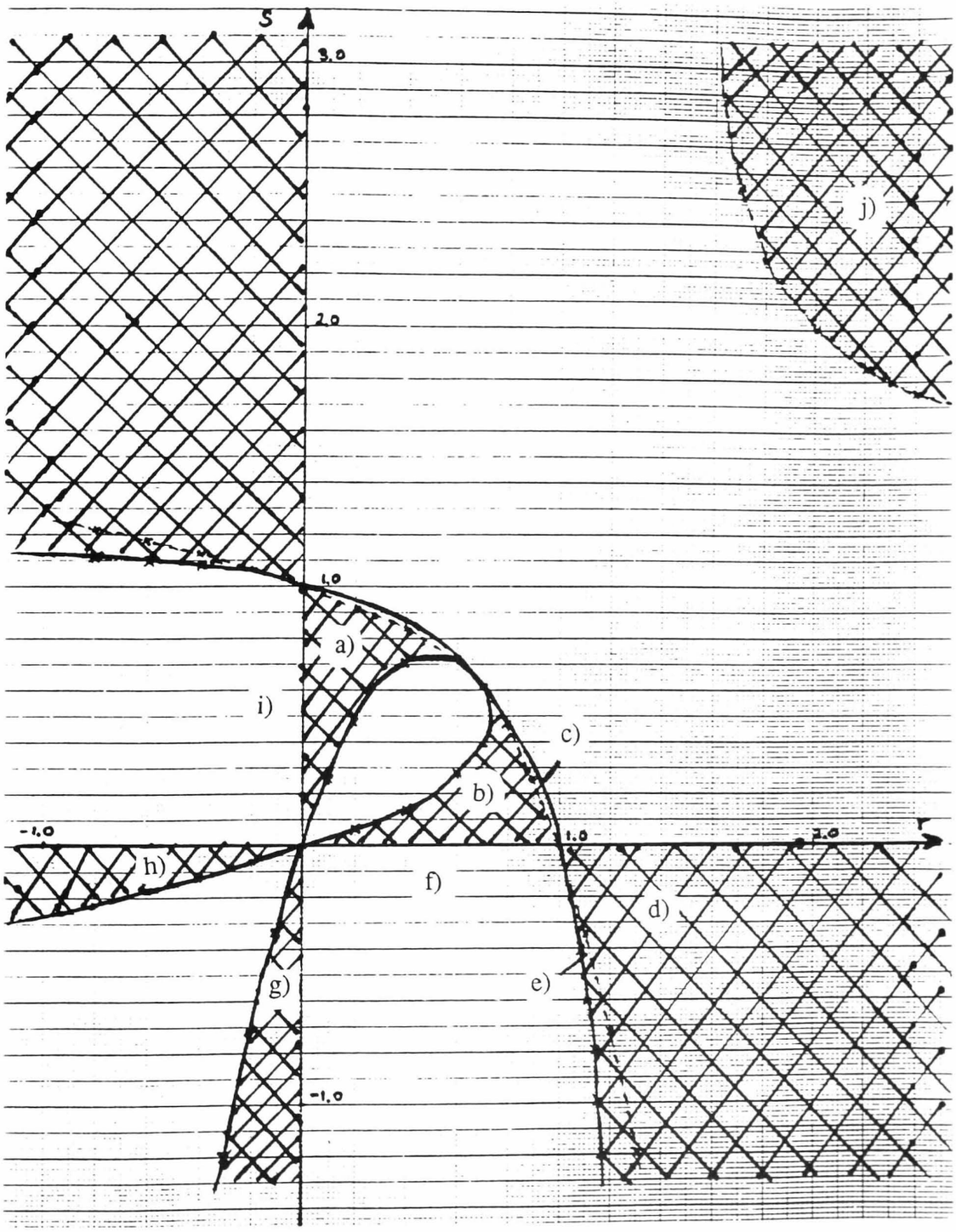


Fig 3.29 Highlights safe areas (shown unshaded) for all the Bezier cubic curve shapes.

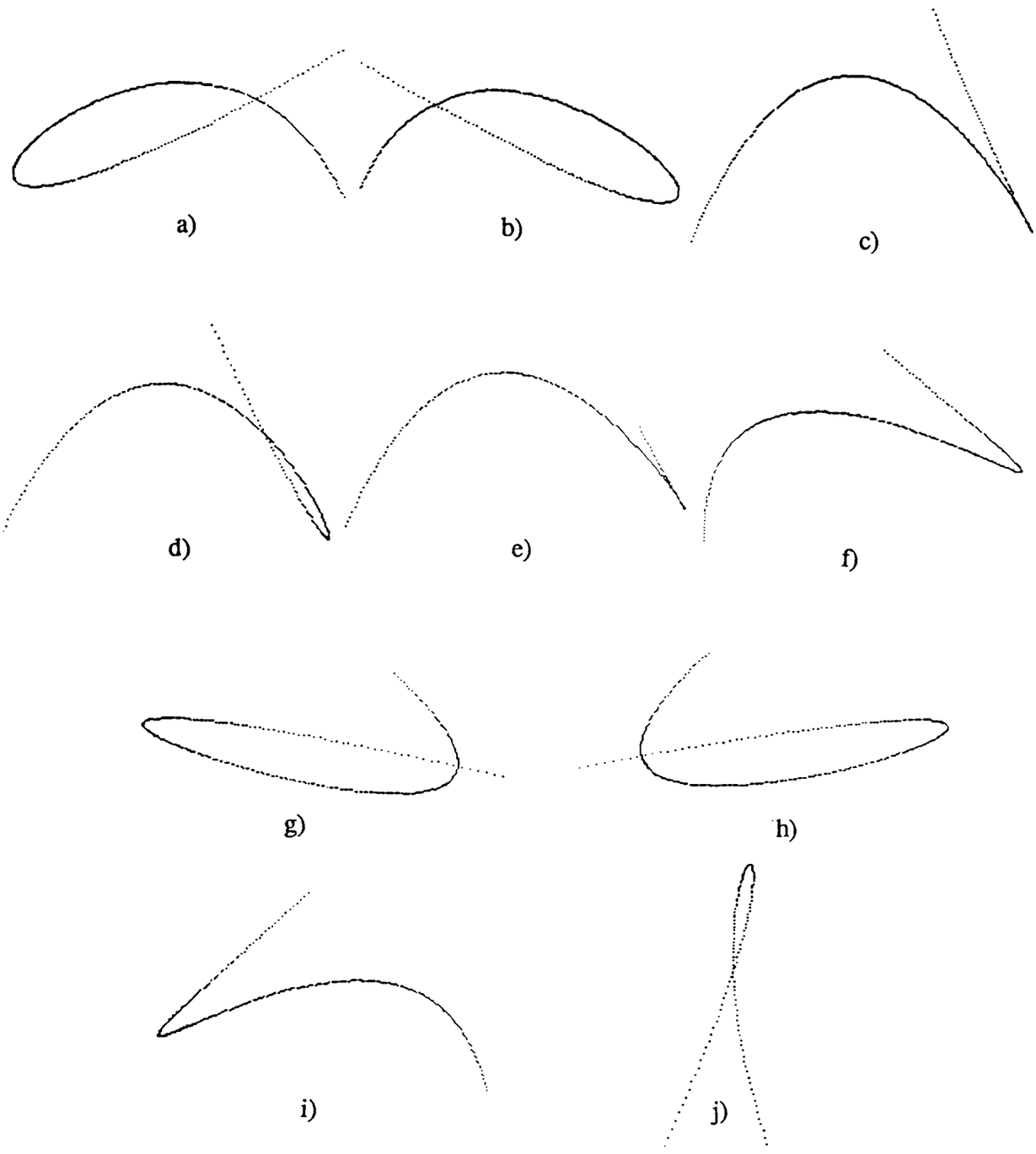


Fig 3.30 Depicts various shapes for a self-intersecting cubic, based on the points marked in Fig 3.29; splitting cases are:
a), b), d), e), g), h) and j).

3.7 Summary

The primary task of this chapter has been to present the Bezier cubic spline. In attempting to achieve this, a detailed discussion of the flexibility and versatility offered by Bezier splines is given. Some of the factors and reasons for its attraction to designers has also been presented.

Two approaches for fitting Bezier splines to a given set of constraints are looked at. The first employs the parametric form. Its performance is analysed in a number of ways, including the introduction of additional curve points and using parametrisation based on arc and chord lengths. A non-parametric form for the Bezier spline is then developed. Possible benefits resulting from this are discussed, and an ill-condition which "haunts" the technique is highlighted.

Finally, two ways for rasterising the Bezier described outlines are presented. The first approach uses a subdivision procedure to approximate the curved outline in terms of line segments. These are then rasterised using a method based on Bresenham's algorithm. The second approach makes use of the implicit form developed for the Bezier spline. It yields the "best" pen (integer) positions. It is compared with the first approach. A limiting case in the implicit approach is highlighted and suggestions to cater for such occurrences presented.

Appendix A3.1

Coefficients for equation (3.19)

In what follows, the '*w*' variables are as given by equation (3.14).

If we let:

$$
\begin{aligned}
x_{21} &= x_2 - 2x_1 \ , \\
x_{31} &= x_3 - 3x_1 \ , \\
y_{21} &= y_2 - 2y_1 \ , \\
y_{31} &= y_3 - 3y_1 \ .
\end{aligned}
$$

Then it can be shown that:

$$
\begin{aligned}
a &= w_y^2(w_1 - w_2)(3y_{21} - y_{31}) \ , \\
b &= -w_x^2(w_1 - w_2)(3x_{21} - x_{31}) \ , \\
v_1 &= w_1^2 y_{31}^2 - w_1 y_{31}(6w_2 y_{21} - w_y w_{12}) \ , \\
v_2 &= w_1 w_y w_{13}(y_{31} - 6y_{21}) + 9w_2^2 y_{21}^2 \ , \\
v_3 &= w_2 w_y(w_{12}(3y_{21} - 2y_{31}) + 3w_{13} y_{21}) \ , \\
c &= -(v_1 + v_2 + v_3) \ , \\
v_1 &= -w_1^2 x_{31}^2 + w_1 x_{31}(6w_2 x_{21} - w_x w_{12}) \ , \\
v_2 &= w_1 w_x w_{13}(6x_{21} - x_{31}) - 9w_2^2 x_{21}^2 \ , \\
v_3 &= w_2 w_x(w_{12}(2x_{31} - 3x_{21}) - 3w_{13} x_{21} \ , \\
d &= v_1 + v_2 + v_3 \ , \\
e &= -(w_1 w_{13} - w_2 w_{12})(w_{12} y_{31} - 3w_{13} y_{21}) \ , \\
f &= (w_1 w_{13} - w_2 w_{12})(w_{12} x_{31} - 3w_{13} x_{21}) \ , \\
g &= -(w_y(w_1 - w_2)(w_x(6y_{21} - 2y_{31}) + w_y(3x_{21} - x_{31}))) \\
h &= w_x(w_1 - w_2)(w_x(3y_{21} - y_{31}) + w_y(6x_{21} - 2x_{31})) \\
v_1 &= w_1(y_{31}(2w_1 x_{31} - 6w_2 x_{21}) - 6w_2 y_{21} x_{31}) \ , \\
v_2 &= w_1 w_x(w_{12} y_{31} - w_{13}(6y_{21} - y_{31})) \ , \\
v_3 &= w_1 w_y(w_{12} x_{31} - w_{13}(6x_{21} - x_{31})) \ , \\
v_4 &= w_2(18w_2 x_{21} y_{21} + w_x(w_{12}(3y_{21} - 2y_{31}) + 3w_{13} y_{21})) \ , \\
v_5 &= w_2 w_y(3w_{13} x_{21} + w_{12}(3x_{21} - 2x_{31})) \ , \\
i &= v_1 + v_2 + v_3 + v_4 + v_5 \ .
\end{aligned}
$$

Hence:

$$
\begin{aligned}
&v = \frac{e}{2} \ , \quad u = -\frac{f}{2} \ , \\
&\alpha = -d \ , \quad \beta = -c \ , \quad \gamma = -\frac{i}{2} \ , \\
&r = -3a \ , \quad s = -3b \ , \quad p = -g \ , \quad q = -h \ .
\end{aligned}
$$

4.0 Capture by Conic Sections

4.1 Introduction

The previous chapter has analysed, developed and explored some of the methods and techniques which the mathematical contemporary of the physical spline employs. In particular, the properties of the Bezier cubic curve (both parametric and non-parametric forms) has been investigated. One of the main observations regarding the cubic spline is its inherent need for iterative procedures. Whether attempting to gain a desired mathematical description or seeking to rasterise the modelled outline, the best and most suitable approach requires the use of some recursive routines. This need has a detrimental effect on the rate of converting IK-defined characters to Bezier described outlines. With there being thousands of fonts, each on average comprising of at least one hundred characters, there exists a demand for describing contours using a simpler modelling scheme based on the *quadratic* spline.

This chapter focuses on the capturing capabilities and shortcomings of the curves belonging to the conic (quadratic) family. After presenting their various characteristics and features, two approaches for mathematically describing a given set of data points using conic sections are given. The first method provides a desired fit based on the tangents at the knot points; whilst the second works within a more relaxed environment and yields the "best" possible quadratic section which will model the given outline based entirely on the location of data points supplied. The performance of both approaches are then analysed and compared. Through this, the advantages and limitations of the two capturing algorithms are highlighted. Finally, in section 4.5, a technique for rasterising the quadratic curve segments is presented.

4.2 Characteristics and Properties

Conics, and conic splines, have a number of features and characteristics which make them suitable for modelling contours of shape. Although conics have a rich background in terms of the literature available discussing their properties, this

section limits itself to presenting their capabilities for the purposes of design. This leads to the introduction of the curves belonging to the conic family. Through this, some of their properties are highlighted. Both the parametric (including rational and non-rational) and implicit forms for the quadratic spline are looked at. Some applications employing conic splines are also presented.

4.2.1 Historical Perspective

One of the main attractions of conic, quadratic, sections is that they have been studied for centuries and, therefore, there is a wealth of mathematical results about them. Of the relevant literature, the book by Sampson [SAMP 79] gives a classic introduction to conic sections and their properties. Coolidge [COOL 45] provides a historical input to the subject. A number of books give a comprehensive, although at times rather complex, treatment to the geometric aspects of conic sections [ASKW 47, BAIL 36, SMIT 26, TODD 47, TUCK 18].

The employment of conic sections for the purpose of design, and representing outlines of shape, was looked at in the early forties by Liming [LIMI 44]. His work in the aviation industry required mathematical modelling of contours forming components of mechanical parts, as well as accurate approximations of the cross-sections of aircraft fuselages. Although the methods developed were exclusively for the aviation industry, the principles which evolved were applicable to other manufacturing industries.

The benefit of having the possibility of algebraic and geometric solutions has played a part in the successful employment of conic sections for modelling outlines. As mentioned in the previous chapter, the use of cubic curves and splines did not really materialise until the advent of computers. With this, it was possible to develop computerised techniques which enabled the simulation of the physical spline. Even with the latest algorithms, the modelling of shape using cubic splines is still a demanding task. The use of conic sections for purposes of computer-aided-design, on the other hand, can be viewed as automating the

traditional approach; thereby, enhancing and improving an acceptable form of representation.

The fact that it is possible to manually construct conic sections from a given set of constraints has been dealt with best by Liming. Although the above cited reference highlights some of these techniques, his later publication gives a more comprehensive insight on the topic [LIMI 79]. This, apart from others, includes the (manual) construction of a conic section from three points and two tangents, four points and one tangent, and five points. (In chapter six, a procedure for manually constructing a parabolic arc is given).

With the development of parametric representation for formulating curve descriptions, the mathematical form for defining conic sections has been parameterised. Whereas, Liming [LIMI 44] described how the theory of pencils of conics can be used to define a general conic segment, Faux and Pratt [FAUX 79] have dealt with its parametrisation. Both Ball [BALL 74] and Forrest [FORR 68] link the use of a rational cubic form to gain a rational quadratic (parametric) form, Forrest evolving the parametric parabola to gain representation for various conic sections [FORR 68, FORR 80]. In section 4.2.3, the conic spline and its mathematical forms are discussed. The next section, gives an introduction to the family of curves which come under the heading of conic sections.

4.2.2 Family of Conic Curves

Before analysing and discussing the characteristics of the conic spline, this section gives a summarised introduction to the nature of the conic shapes and their classical definitions. The section focuses on properties which are relevant in understanding some of the concepts that are later developed relating to quadratic splines.

The term "conic sections" is derived from the fact that all the shapes can be obtained from intersecting a plane, at various orientations, with the surface of a cone. As Fig 4.1 shows, depending on the way the cone is intersected, it is

possible to gain a family of curves including circles, ellipses, hyperbolas and parabolas. All of these curves can be described by a second order (quadratic) equation. This, in general, takes the form:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \ . \qquad ...(4.1)$$

Equation (4.1) is expressed in terms of six coefficients. The shape and type of conic generated by such an expression does not depend on the absolute values of these coefficients, but is a function of their ratios to each other [SAMP 79]. This means we could divide equation (4.1) by any one of the six coefficients, leaving just five coefficients which uniquely define a conic shape. Furthermore, for the purposes of capture, five input constraints are required either in terms of data points or a mixture of points and derivatives.

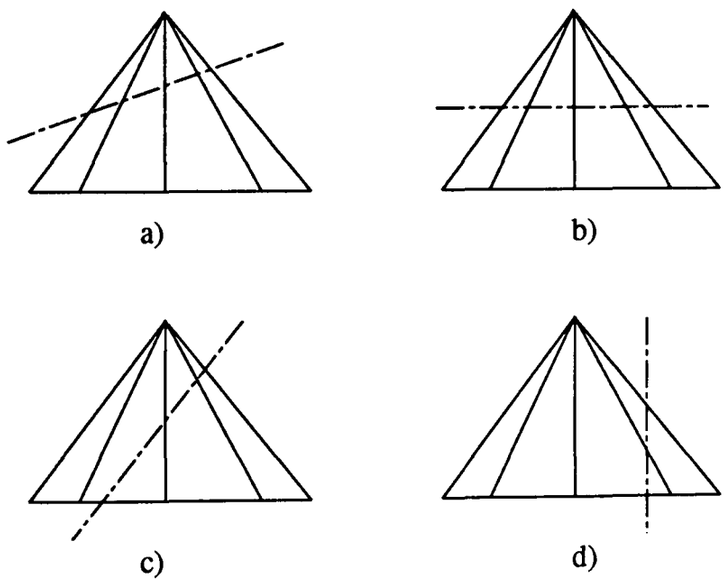


Fig 4.1 Illustrates how the intersection of a plane with a circular cone defines a particular conic section: a) an ellipse, b) a circle, c) a parabola, and d) a hyperbola.

The most frequently used "curved" shape is the circle. In just about every design system, the employment of circles and respective arcs is found in one form or another. The general conic equation (4.1) can be transformed to represent circles by realising that in this case the coefficients of x^2 and y^2 will be equal, and that the orientation coefficient B will be equal to zero (this is explained in the next paragraph). Thus, $A = C$ and $B=0$, and by completing the squares the well-known equation for circles can be derived (see [CHAS 78] for a detailed analysis).

The desire to have an elliptic curve tends to be a little more involved than the circular case, even though the circle is a particular type of ellipse. For conic equation (4.1) to represent an elliptic shape, it is necessary for coefficients A and C to have similar signs irrespective of the magnitudes. The ellipse is described in terms of a major and a minor axis. If these axes are at some angle to a reference axis system, then the term xy (sometimes called the cross product or orientation term) of equation (4.1) exists; otherwise, $B=0$. For design simplicity, the tendency is to work with the *standard* ellipse, where the axes of the ellipse are made parallel to those of the defining coordinate system [CHAS 78, ROGE 76].

The equation of the general conic can also be formulated to describe the shape of a parabola. For the standard case, where $B=0$, the requirement is that either A is zero or C is zero, but not both. This will result in either term x^2 or y^2, and by means of simple algebraic manipulation, the standard form for describing a parabolic shape can be gained. (Chapter six investigates, and presents, the parabolic spline as an alternative means for mathematically describing outlines).

In similar fashion to the elliptic case, the general conic equation of (4.1) can be transformed to return a hyperbolic shape. The coefficients of x^2 and y^2, in this case, require to have opposite signs irrespective of the magnitudes. Again, based on similar reasons as for the ellipse, a standard form for the hyperbola can be realised.

In passing, it is worth mentioning the fact that it is always possible to rotate the general conic in order to eliminate the B coefficient (of equation (4.1)). This can be done by rotating the given conic through a angle defined by $\cot(2\theta) = \frac{A-C}{B}$ [ROGE 76]. This combined with the option of translating the resulting general conic to eliminate the coefficients D and E, formulates a typical approach for gaining standard mathematical forms for the conic sections.

Based on some of the observations made in this section, it can be shown (see [YEFI 64] for a detailed proof) that the expression $B^2 - 4AC$ provides a means of quantifying the shape of a conic section. If this is less than zero, then the resulting curve is an ellipse (including the special case of a circle); if its greater than zero, then a hyperbolic curve shape is expected, and if it is equal to zero, then the resulting arc is a parabola. This is a useful property, and lends itself conveniently for purposes of interactive design where quadratic curves sometimes need to be classified for capture or rasterisation purposes.

Finally, it should be stressed that the material covered in this section yields only a brief insight to the family of conics. For thorough examination of these curves, some of the cited references give a good introduction, including the necessary mathematics. Two other books which provide a classical review of conic sections are by Protter and Morrey [PROT 75], and Vasilyev and Gutenmacher [VASI 80]. The first devotes a complete chapter (chapter six) to developing and working with standard forms, whilst the other gives an illustrative presentation of geometric properties including that of the quadratic sections.

4.2.3 Classification of Conics

The term "conic sections" is also frequently used to describe curved outlines which can be modelled by part of a defining conic shape. This in the case of an ellipse, for example, might result in less than half of its entire outline being used to describe a given set of data points. When only a portion of a conic is being employed, it will hereafter be referred to, in general, as an arc or a segment or

simply as a conic section. The expression "conic splines" then refers to a series of conic arcs patched together, with given constraints, to form a desired outline of shape. The conic spline, therefore, has similar properties to the Bezier spline in that it facilitates the construction of a desired shape using localised segments. This, as mentioned in sections 2.2.2 and 2.2.3, is an important and necessary property for purposes of interactive design work.

To capture outlines by means of conic splines, a suitable mathematical form is required where a conic arc can be generated between given knot points. A common approach to achieve this has been to develop a rational quadratic form for Bezier curves. A detailed derivation of this is given by Lee [LEE 87]. He uses the derived formulation to determine the characteristics (such as centres and foci) of the resulting conics. Faux and Pratt [FAUX 87] gain the same result through a derivation based on a specific parametrisation for the conic. The resulting rational form for the general conic case is expressed as follows:

$$P(t) = \frac{P_0 w_0 (1-t)^2 + 2P_1 w_1 t(1-t) + P_2 w_2 t^2}{w_0(1-t)^2 + 2w_1 t(1-t) + w_2 t^2}, \qquad \text{...(4.2)}$$

where: t is normalised to range from zero to one,
P_0, P_1 and P_2 form the defining polygon, and
w_0, w_1 and w_2 are their respective weights.

The resulting quadratic polygon takes the triangular set-up shown in Fig 4.2. It is clear from this that the control point, P_1, is gained from the tangents at the two knot points, respectively at P_0 and P_2. The curve that is generated, therefore, starts at the knot point P_0 and terminates at P_2, and it is tangential to P_0P_1 at P_0 and to P_1P_2 at P_2.

The weights of equation (4.2) are similar to those employed in the rational Bezier form in that they pull the curve towards them. The greater the value, the greater the pull (see [FARI 90] for an illustrative example). The weights, thus, control the type of quadratic arc to be generated. Although values for the weights can

range between zero and infinity, the ratio $\frac{w_0 w_2}{4w_1^2}$ remains a constant for a conic arc [PAVL 83, LEE 87]. The tendency, therefore, is to set the weights of the two knot points (that is w_0 and w_2) to equal one. The weight of the control point, w_1, is then varied to gain the complete spectrum of conic arcs.

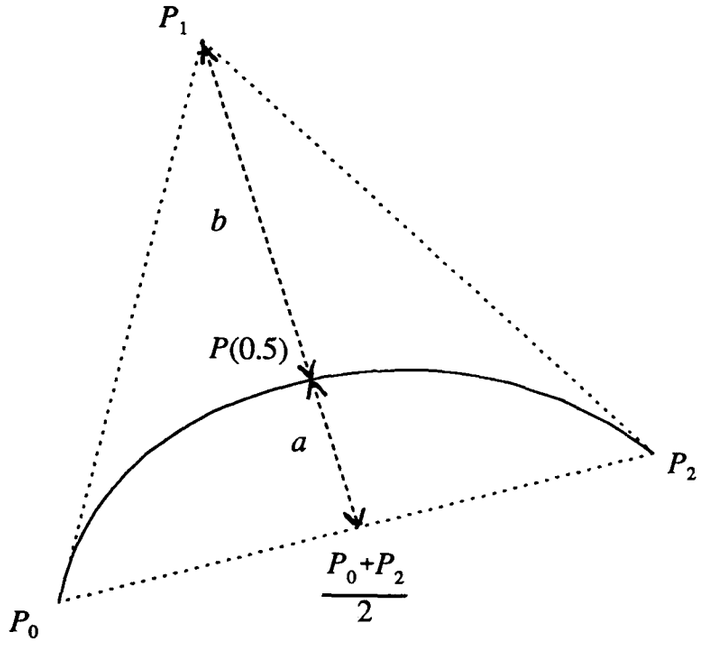


Fig 4.2 Depicts the guiding triangle set-up for describing conic sections, where the sharpness value $S = a/b$.

The control weight, w_1, is more commonly referred to as the stiffness or sharpness value for a conic arc, and denoted by S. In line with the above considerations, and as employed by Pratt [PRAT 85], the value of S determines both the type and the shape of a conic section. As Fig 4.2 illustrates, the value for the sharpness is based on the lengths a and b; where a is the length of the line extending from the point $\frac{P_0+P_2}{2}$ to the mid-point of the curve $P(0.5)$, and b is the corresponding length from this mid-point to the control point.

Referring to Fig 4.2, when S equals zero then the arc is the line connecting the two knot points, between zero and one it returns elliptic segments, at one a parabola results, and for values of S between one and infinity, hyperbolic arcs are realised. Close to infinity, or in practice for very large values of S, the arc returns the two line segments connecting the control point (ie lines P_0 --> P_1 and P_1 --> P_2 respectively). In addition, for positive sharpness values, the resulting arc is contained within the convex hull of the defining triangle. If S is allowed to have negative values, then the arc will lie outside the convex hull. This will in the case of S ranging between 0 and -1, for example, result in a curve which complements the elliptic shape contained within the convex hull.

Applying these considerations to equation (4.2) results in the following rational parametric form for describing quadratic splines:

$$P(t) = \frac{P_0(1-t)^2 + 2P_1t(1-t)S + P_2t^2}{1 + 2t(1-t)(S-1)} . \qquad ...(4.3)$$

It is clear from equation (4.3) that for the parabolic case, where S equals one, the term in the denominator equals unity, and the resulting parabola is generated by the expression in the numerator. In other words, when the weights of the three points (defining the triangle) are made equal, the non-rational parametric form for quadratics results, which is a parabola. This means that for a three-point guiding (Bezier) triangle, the generated curve segment is always a parabolic arc.

Having considered the parametric form to illustrate some of the conic splines properties, its implicit (algebraic) form is considered next. This mathematical scheme is employed in the two capturing approaches discussed in section 4.4. In order to use the implicit form of equation (4.1), its coefficients need to be expressed in terms of the conic spline considerations mentioned above. For this purpose, the workings of Pitteway and Banissi [PITT 88] are used to gain the following expressions in terms of control point $P_1(u,v)$ and end knot $P_2(C_x,C_y)$:

$$d \equiv 2vx - 2uy - \alpha y^2 - \beta x^2 - 2\gamma xy - k \,, \qquad \ldots(4.4)$$

where:

$$\alpha = \frac{(C_x-2u)^2 + C_x^2 S_p}{4\Delta} \,,$$

$$\beta = \frac{(C_y-2v)^2 + C_y^2 S_p}{4\Delta} \,,$$

$$\gamma = \frac{(C_y-2v)(C_x-2u) + C_x C_y S_p}{4\Delta} \,,$$

$$S_p = \frac{1-S^2}{S^2} \,,$$

$$\Delta = \frac{vC_x - uC_y}{2} \,, \text{ and is the area of the triangle.}$$

For convenience, the residue at the origin $d(0,0)$ is taken to be zero, so that the constant term k will also be zero. The expressions of equation (4.4) are such that apart from the conic curve starting at the origin, it is required to interpolate the end knot, resulting in zero residue, so that $d(C_x,C_y) = 0$. The control point P_1 defines the initial gradient $\frac{v}{u}$, so, by taking advantage of the arbitrary scale, the control positions u and v are made equal to the respective x and y coordinates of the control point P_1.

Although equation (4.4) gives an implicit form for the general conic, a more "visual" form can be realised by substituting the expressions for the coefficients α, β and γ into the main equation. This results in a form which is solely in terms of u, v, C_x and C_y. By collecting terms, a new expression results that is based on the triangular scheme as manifested in Fig 4.3. This expresses the implicit form for the general conic in terms of four triangles. By using this, it can be shown that equation (4.4) can be transformed to take on the following form:

$$d(x_i,y_i) = \frac{4\Delta_i\Delta_j - \frac{\Delta_k^2}{S^2}}{\Delta} \,, \qquad \ldots(4.5)$$

where:

$$\Delta = \textit{area of triangle } P_0P_1P_2 = \frac{vC_x - uC_y}{2},$$

$$\Delta_i = \textit{area of triangle } P_0Q_iP_1 = \frac{vx_i - uy_i}{2},$$

$$\Delta_k = \textit{area of triangle } P_0Q_iP_2 = \frac{C_xy_i - C_yx_i}{2},$$

$$\Delta_j = \textit{area of triangle } P_1Q_iP_2 = \Delta - \Delta_i - \Delta_k, \textit{ and}$$

$$S = 0 \textit{ to } \infty, \textit{ depending on conic shape.}$$

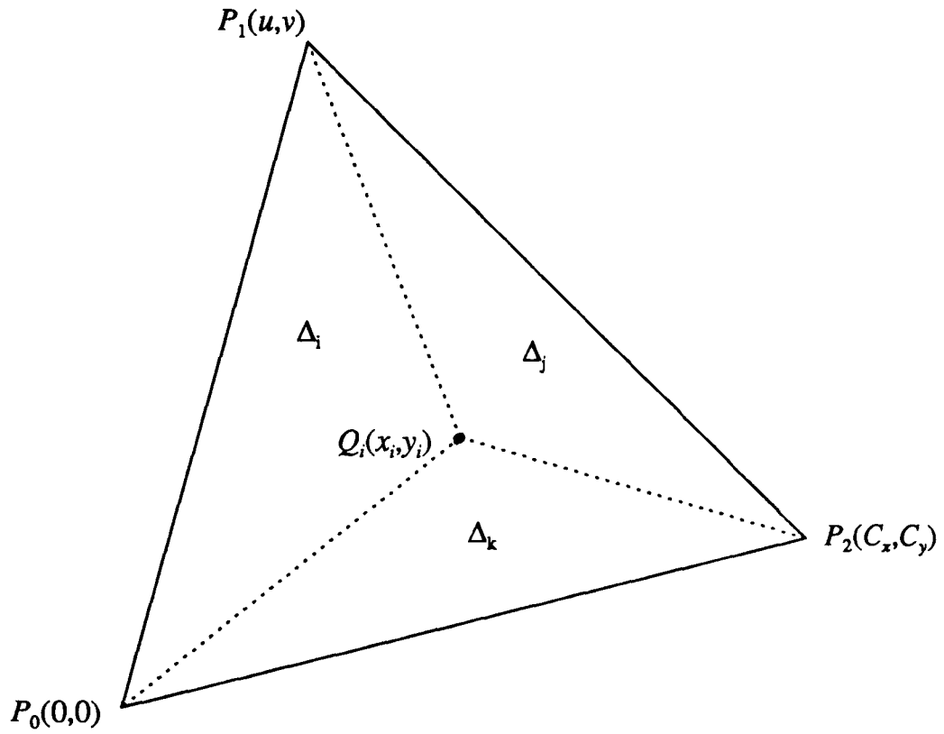


Fig 4.3 Shows how a given point $Q_i(x_i,y_i)$ results in a general conic description which uses four triangles.

The parametric formulation as given by equation (4.3), and the implicit forms of equation (4.4) and (4.5), are mathematical representations which can be used for purposes of conic splining. Both mathematical descriptions have their usefulness, and are applied when and where necessary. The implicit form as given by

equation (4.5) is employed in the first conic conversion algorithm discussed in section 4.4.1, whilst the other implicit form (equation (4.4)) is used in the method presented in section 4.4.2.

The next section looks at the various applications that have employed conic sections to describe contours of shape. Through these, some methods and techniques used to conic-fit a given set of data points are also discussed.

4.2.4 Applications and Techniques

As mentioned in section 4.2.1, conic sections have been employed in the aviation industry to model components, parts and sections of aircraft fuselages. This constitutes an important development in employing conics for designing shape outlines. The use of quadratic sections extends to other fields where a requirement to model shape outlines exists: In the field of human biology, a common need is to represent the shape of cells (and its contents) for purposes of analysis. Paton [PATO 70] proposed the use of conic sections for representing individual chromosomes. His development aided the investigation of properties and characteristics of individual chromosome.

Albano [ALBA 74] presents an algorithm that captures digitised contours in terms of conic arcs and line segments. He employs the method of weighted least-squares to gain "best-fitting" conic representation. He extends the approach to finding the minimum number of quadratic arcs that can be employed for a given set of data points, and which lie within a desired tolerance.

Like cubic curves, conic arcs have also be used to model dental arches: Biggerstaff [BIGG 72] provides a brief inclination towards employing the family of conic curves for representing shapes of human dental arches. A more rigorous and detailed method is given by Bookstein [BOOK 79]. He develops an approach that attempts to represent scattered data. His algorithm returns the optimum conic-fit by minimising the curve-to-point deviation expressed as a ratio of two squared distances along rays through the centre of a conic. An approach which

endeavours to make improvements to Bookstein's algorithm has been developed by Sampson [SAMP 82]. He employs an iterative subroutine which yields better approximations, and is based on minimising the sum of the squared orthogonal distances of data points from the fitted conic. (Two other references by this author, where he discusses the shape of dental arches in relation to conic sections, are [SAMP 81] and [SAMP 83]).

The employment of conic arcs and splines for modelling the outlines of font characters has been presented by both Pavlidis [PAVL 83] and Pratt [PRAT 85]. Both conjecture and develop techniques which use quadratic splines rather than cubic. The paper by Pavlidis [PAVL 85] discusses in detail the development of the guiding triangle for the conic arc (see section 4.2.3). Having proposed a criteria for evaluating the point-to-curve deviation, he develops an algorithm for modelling the contours of shape (including outlines of font characters). Pratt [PRAT 85], in contrast, looks at how it might be possible to use quadratic splines to capture shape outlines which, traditionally, have employed cubic curves. He also develops an approach for making Pitteway's conic generating algorithm [PITT 67] exact, and fixes an aliasing problem which has plagued it since its introduction.

It can be seen from this that conic arcs and splines are applied in a number of different, and diverse, fields where a given shape outline requires to be modelled. The fact that the conic family can mathematically be represented through either a parametric form or via the implicit formulation leads itself to greater applications.

In passing, it should be emphasised that when it comes to modelling a given shape outline (for a particular application), each capturing process is formulated to return a desired type of "best-fit". The process is termed quadratic normalisation and is used to "customise" the conic-fit to meet some specified objectives. Bookstein [BOOK 79], for example, uses the expression $A^2 + B^2/2 + C^2$ to gain conic approximations which are invariant under normal transformations. He develops the desired normalisation by observing that both the

terms $A + C$ and $B^2 - 4AC$ are invariant. His algorithm is, therefore, normalised to encompass this property. Different authors have applied different normalisations. Pratt [PRAT 87] gives a detailed discussion of some of the normalisations that have been used in capturing outlines, with regards to both curves and surfaces. He highlights their objectives and possible side-effects. Furthermore, he provides some useful concepts for fitting mathematical descriptions to data.

It is clear, therefore, that different applications may employ different normalisations in order to ensure that the resulting quadratic approximation meets some given design specifications. Further discussion of this concept is made in the relevant sections manifesting the solutions to the problem outlined in section 4.3.

4.2.5 Point-to-Conic Deviation

Before giving details of the problem to be solved, this section presents a method for evaluating the quality of any conic approximation. The approach is based on calculating the perpendicular distance of a point from a straight line. For this purpose, the implicit form for the conic as given by equation (4.5) is utilised. Unlike the parametric form, equation (4.5) has the attraction of returning residue values directly, rather than through an iterative process.

The approach makes the assumption that in the vicinity of a given IK point, the corresponding conic curve is slow moving (low curvature) so that it can be approximated by a line segment. This, for the defining IK curve points, tends be the norm rather than the exception.

If for clarity we let d_i denote the residue as given by equation (4.5), and Λ_i as being the corresponding point-to-curve deviation, then using the graphical illustration of Fig 4.4, it can be shown that:

$$\Lambda_i \approx \frac{d_i}{\sqrt{\Delta_x^2 + \Delta_y^2}}, \qquad \text{...(4.6)}$$

where:

$$\Delta_x = \frac{\partial d_i}{\partial x} = 2v - \frac{2S^2[\Delta_i(2v+C_y)-v\Delta_k]+C_y\Delta_k}{\Delta S^2},$$

$$\Delta_y = \frac{\partial d_i}{\partial y} = 2u - \frac{2S^2[\Delta_i(2u+C_x)-u\Delta_k]+C_x\Delta_k}{\Delta S^2},$$

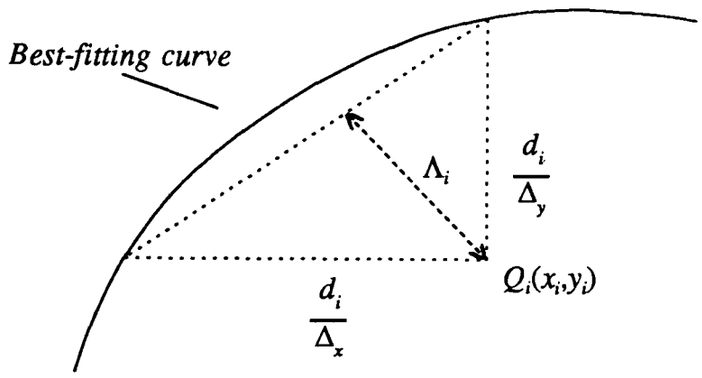


Fig 4.4 Highlights the method for evaluating the point-to-conic curve deviation Λ_i.

Equation (4.6) forms a means of assessing the quality-of-fit. Although this measurement is an approximation, it does in practice conform to being an acceptable way of gauging the resulting conic-fit.

4.3 Outline of Problem

Having described some of the characteristics and properties of conic sections and conic splines, this section gives an outline of the nature and type of approximation which is required:

As mentioned in section 3.3, the IKARUS system requires to model outlines of font characters. One of the mathematical descriptions it employs is based on

conic splines. The requirement, therefore, is to develop conversion approaches which can be used by IKARUS. It defines the problem as follows:

Given a set of data points (Q_i) describing a character outline, whose directional tangents at each point are known, a "best-fitting" conic approximation is required. The quality-of-fit is assessed in terms of a predefined tolerance. The number of conic segments used are also observed.

The "best" approximation is quantified in terms of two approaches: The first technique uses the given tangential information. The resulting conic-fit is ensured to exhibit at least first order (gradient) continuity between adjacent curve segments. The second technique is made more flexible in that it is not constrained to employ the given tangents. It returns the conic-fit which deviates the least from the given IK data points. Both approaches use techniques which are non-iterative and are expected, therefore, to yield acceptable conversion rates. Although the approaches have been developed with specification to the IKARUS system, both algorithms lend themselves to general curve-fitting applications.

4.4 Algorithms for Capture

The first method for gaining a general quadratic description is discussed in the following section. In section 4.4.2, the second approach is detailed. A comparison of both techniques is made in section 4.4.3.

4.4.1 Capture Through Knot Tangents

This method caters specifically for the case where the tangents of all the data points, and not just of the knot points, are supplied. Although the method only uses the knot tangents, tangential information for the curve points is necessary for the process of subdivision. This takes place when either the desired conic-fit is found to be unsatisfactory or when a given outline cannot be modelled by a single conic arc. The latter occurs (taking a circle outline for example) when the knot tangents span more than, or at least, 180 degrees so that the desired control point for a guiding triangle is not possible. The process of subdividing the given

outline is achieved through choosing a suitable data point which lies close to, or is, the mid-point of all supplied data points.

For the purposes of mathematically modelling the given set of IK points $Q(x_i,y_i)$, the implicit form as given by equation (4.5) is utilised. The conic arc as expressed by this equation is in an ideal form for capturing data with given tangent constraints. It is apparent from this equation that, in general, an approach is required to solve for the three unknowns: the control points u and v, and the sharpness S. The "best-fitting" control position is gained from the intersection point of the two knot tangents, leaving only the sharpness value to be determined.

In an attempt to gain a corresponding "best-fitting" value for the sharpness, equation (4.5) is employed. As this does not go through any further process of normalisation, the sharpness value endeavours to minimise the residue $d(x_i,y_i)$ for each given data point. Similar to the discussion for the implicit Bezier case outlined in section 3.5.2, for a perfect conic-fit, the residue will equal zero; otherwise it takes on a small none zero value which can be quantified (in the manner discussed in section 4.2.5) to assess the quality of conic fit.

The residue given by equation (4.5) is minimised by choosing the best values for S^2. This mathematically is achieved through evaluating the following:

$$\frac{\partial}{\partial(S^2)}\sum_{i=1}^{n} d(x_i,y_i)^2 = 0 \ .$$

If we take the partial derivatives as instructed, and collect common terms, then it is not too difficult to show that the sharpness value which "best-fits" the given conditions can be gained through the following:

$$S^2 = \frac{\sum_{i=1}^{n} \Delta_k^2}{\sum_{i=1}^{n} 4\Delta_i\Delta_j} \ . \qquad \ldots(4.7)$$

It is clear from equation (4.7) that a "best-fitting" conic approximation is gained without having to resort to iterative means. The approach, when compared with that for the Bezier cubic case, uses relatively modest amount of computation and, thus, highlights a feature which has attracted designers to employ conic splines to capture outlines.

Fig 4.5 gives a graphic illustration of the steps necessary to gain a conic description using this approach based on knot tangents. As it shows, if any of the given IK data points returns a deviation value (calculated through equation (4.6)) which is above an acceptable tolerance, then the IK points are subdivided and modelled by two or more conic sections.

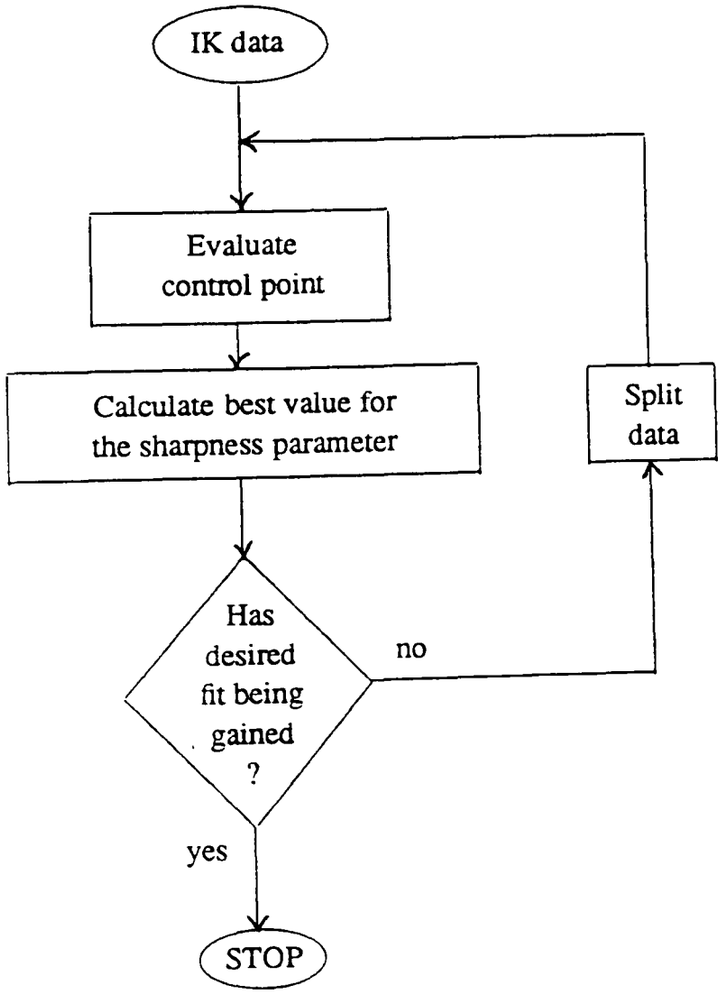


Fig 4.5 Depicts the steps necessary to gain a general conic description which ensures gradient continuity at joining knot points.

The conversion algorithm's performance is analysed in-relation to the second, and alternative, approach given in the next section. This, as manifested in section 4.4.3, is achieved through assessing their capturing capabilities for a given outline of character.

4.4.2 Capture with Least Deviation

In section 4.4.1, an approach for modelling outlines using conic sections is given which has the attraction of ensuring gradient continuity between joining arcs. In addition, the computation necessary for a solution is relatively little compared to that needed for the Bezier cubic case. The method employed, however, is restricted by the tangent constraints and does not return the conic curve which would deviate the least from the given IK data points. As the goodness-of-fit is assessed through the worst-case deviation, it gives additional incentive to develop an approach which reduces this and, therefore, minimises the number of conic arcs employed.

Before giving the mathematical insight, it is worth noting that this was developed whilst experimenting with the approach described in section 4.4.1. Having found a conic-fit, it was observed that if the three parameters (the u and v of the control point and the sharpness value S) were each iterated with small increments, then at times there was a significant reduction in the overall residue. In other words, a better conic-fit resulted. This iterative process lead to the following mathematical development, which offers a non-iterative solution:

The approach uses the implicit representation of equation (4.4), and is based on the guiding triangle scheme discussed in section 4.2.3. Looking at the implicit form, it is clear that there are (in general) five coefficients to solve: u, v, α, β and γ. A conversion procedure is required which will give the "best" values for all these coefficients. As the two knot points (the starting knot being at the origin) are given, a possible approach could substitute the expressions for α, β and γ, and gain an implicit form in terms of the desired three parameters u, v and S (as undertaken in the previous section). This approach, however, leads to a

non-linear equation with respect to the three unknowns; resulting in a iterative means for a solution.

A better approach (which leads to a linear set-up) is to develop an implicit equation based on α, β and γ, instead of u, v and S. In order to achieve this, equation (4.4) needs transforming such that it is expressed solely in terms of the three unknowns (α, β and γ). The requirement, therefore, is to cultivate expressions for the control coordinates (u and v) such that they are expressed in terms of the three unknowns. This process of transformation is initiated by formulating an expression for the gradient at any point on the conic curve. Using equation (4.4), this takes the form:

$$\frac{dy}{dx} = \frac{v - \beta x - \gamma y}{u + \gamma x + \alpha y} . \qquad ...(4.8)$$

The control point, apart from determining the initial gradient, also defines the gradient at the end knot. The corresponding gradient at this point (using equation (4.8)) takes the following form:

$$\frac{v - \beta C_x - \gamma C_y}{u + \gamma C_x + \alpha C_y} = \frac{C_y - v}{C_x - u} . \qquad ...(4.9)$$

The conic equation at the end knot is given by $d(C_x,C_y) = 0$. By utilising this fact and combining expressions with that of equation (4.9), it can be shown that the control coordinates can be expressed as follows:

$$u = \frac{C_x - \gamma C_x - \alpha C_y}{2} ,$$
$$v = \frac{C_y + \beta C_x + \gamma C_y}{2} . \qquad ...(4.10)$$

Substituting the expressions for u and v in equation (4.4), a new implicit form (describing all conic sections) results:

$$d(x_i,y_i) \equiv \alpha(yC_y - y^2) + \beta(xC_x - x^2) + \gamma(xC_y + yC_x - 2xy) + (xC_y - yC_x) . \qquad ...(4.11)$$

Again, the quantity $d(x_i,y_i)$ yields the residue of a given IK point to a resulting conic curve. Although the conversion algorithm minimises this residue by choosing the "best" values for α, β and γ, as an alternative approach, equation (4.11) could be normalised such that it is in a form which represents the shortest distance from the resulting conic curve to IK point. In other words, use the formulation of equation (4.6), where the corresponding parameters (such as d_i) are based on equation (4.11). If an attempt is made to develop such an approach then we will find that the resulting equation becomes, not just more complicated, but leads to non-linear expressions which require iterative methods for a solution to the three unknowns. Since the conversion process based solely on the equation (4.11) attempts to minimise the residue, looking at equation (4.6), this would also lower the corresponding point-to-curve deviation. In short, the residue formulation as given by equation (4.11) appears to be the most attractive both in terms of its effectiveness and, also, for its computation needs.

The "best" possible conic approximation to a IK defined character outline is gained by choosing the "best" values for the three unknowns. The solution required, therefore, needs to minimise the summation of the squared residue (as given by equation (4.11)) at each data point, that is:

$$\frac{\partial}{\partial\alpha}\sum_{i=0}^{n} d(x_i,y_i)^2 = 0 ,$$
$$\frac{\partial}{\partial\beta}\sum_{i=0}^{n} d(x_i,y_i)^2 = 0 ,$$
$$\frac{\partial}{\partial\gamma}\sum_{i=0}^{n} d(x_i,y_i)^2 = 0 . \qquad ...(4.12)$$

For a solution to equation (4.12), the computationally fast routine developed by Lewis [LEWI 86] is employed. The routine returns the corresponding values for α, β and γ, and also the total value of the squared residue. The latter could be used if, for example, we were interested in evaluating the average residue of a given outline.

Having gained values for α, β and γ, equation (4.10) is used to calculate the respective "best" values for the control coordinates. The sharpness value for the

"best-fitting" conic arc is acquired by the utilising the following expression (evaluated with reference to the forms of α, β and γ as given in equation (4.4)):

$$\alpha\beta-\gamma^2 = \frac{1}{S^2}-1 ,$$

so that

$$S = \frac{1}{\sqrt{1+\alpha\beta-\gamma^2}} . \qquad ...(4.13)$$

The quality of the resulting conic-fit is then assessed in the manner discussed in section 4.2.5. If it is found that this is not satisfactory, then the given IK defined outline is split near its centre and approximated by two conic curve segments.

This alternative approach for capturing outlines using conic sections is graphically illustrated in Fig 4.6. Again, if a single conic does not suffice, then two or more conic arcs are used so that the point-to-conic deviation (as given by equation (4.6)) is below a desired value.

4.4.3 Results and Observations

Having described two approaches for modelling a given set of IK defined outlines of characters, this section analyses their performance. For purposes of demonstration, the character 'S' of typeface an201215.ik is, again, chosen. This employs four additional points between each two of its digitised IK points (as discussed in section 3.4.4). Because this character contains four inflection points (in the manner shown in Fig 3.8), it is clear that at least six conic sections will be required for a quadratic description. It is expected that additional arcs will be necessary for parts of a given outline which do not contain an inflection point, but are shaped such that an appropriate guiding triangle cannot be formed (as explained in section 4.4.1).

The performance of each algorithm is tabulated in Fig 4.7. These results are gained with a maximum allowable point-to-conic deviation of 15 units. Analysing the recorded results of Fig 4.7, it is clear that the second approach, which is

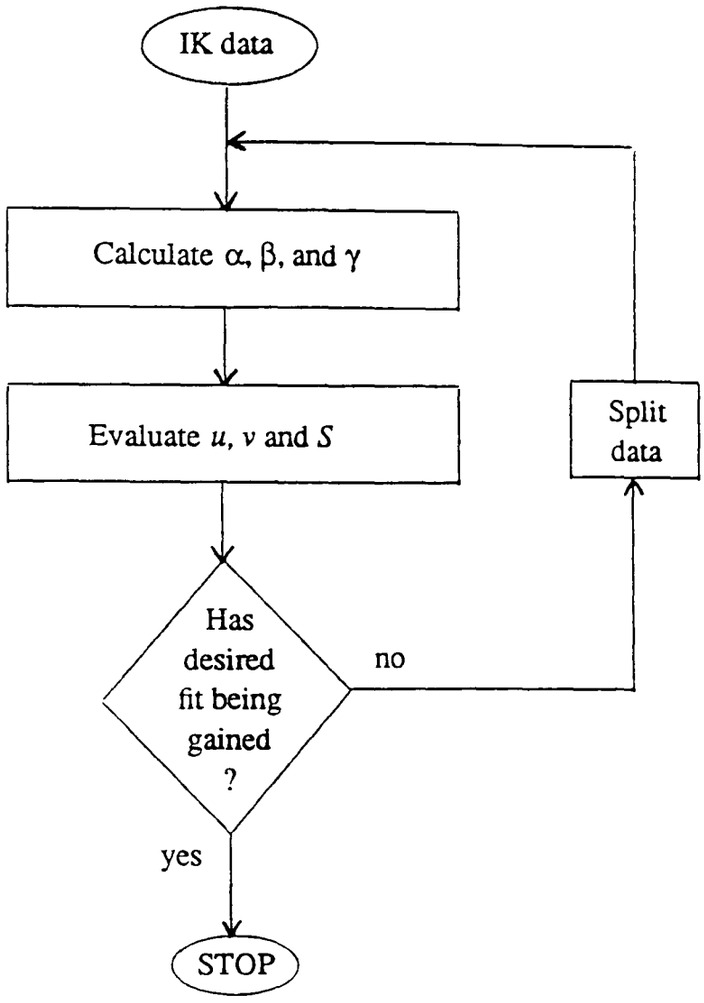


Fig 4.6 Shows the stages necessary to perform an IK to general conic conversion which deviates the least from the given set of data points.

allowed to take on "best" values for u, v and S, uses the least number of arcs. It employs seven less conic sections than that needed for the tangent approach. The amount of deviation exhibited by the modelling arcs is, on average, remarkably little. With an acceptable tolerance of 15 units (where each unit equals 1/100mm for a character bodysize of 15cm), the two algorithms both approximate the given 'S' character with a high degree of accuracy. Since the second approach uses a significantly lower number of arcs, it is not surprising to note therefore that its average deviation value is slightly higher. Fig 4.8 depicts the corresponding conic descriptions, returned by both approaches, for the given letter 'S'. It is worth

noting that (as expected) the four points of inflection have all been translated into respective knot points.

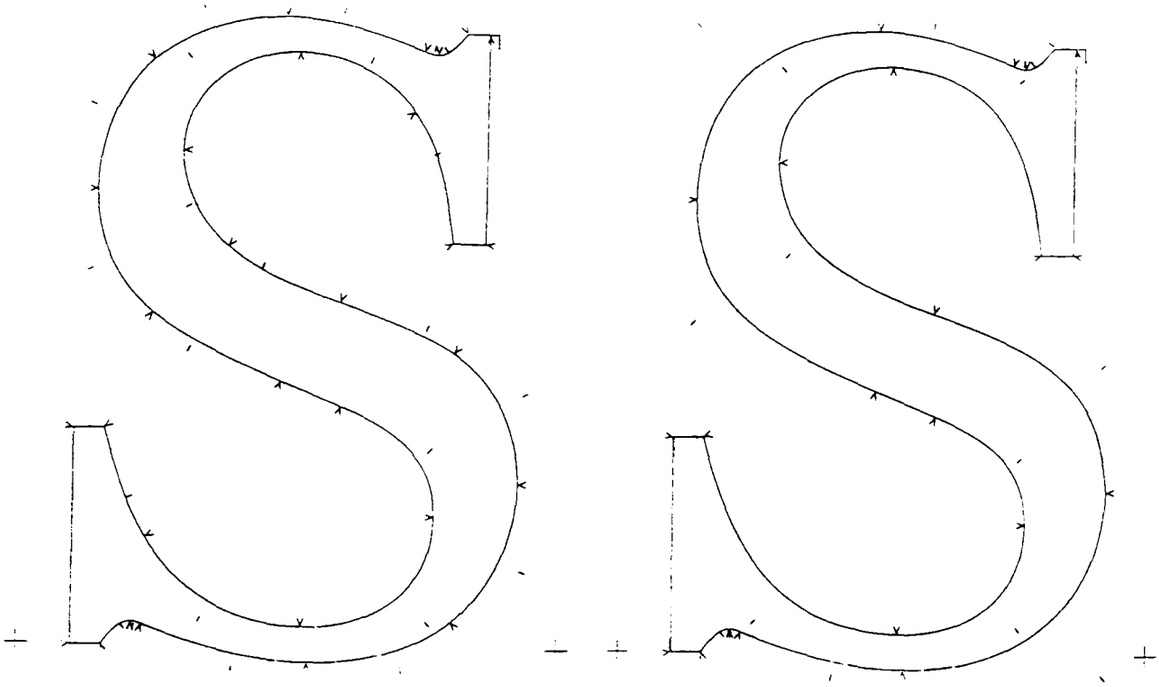
IK to General Conic Results	Conversion through tangents (Fig 4.5)	Conversion through deviation (Fig 4.6)
Number of Arcs	23	16
Rate of Conversion *CPU seconds*	9.3	8.7
Maximum IK-to-conic deviation	7.5307	13.3483
Average deviation per arc	1.3270	2.0169
Maximum variation of knot tangents *degrees*	0	10.1912
Average variation of knot tangents *degrees*	0	2.9933

Fig 4.7 Shows tabulated results for the two approaches used to convert the given IK data points to a corresponding general conic description.

The conversion rate of the two algorithms is considerably high, especially when compared to the results for the Bezier cubic case (section 3.4.4). Although the conic algorithm of Fig 4.6 yields a somewhat faster rate, the amount of processing necessary in each approach is relatively small. Both algorithms have the attraction, therefore, of efficiency in terms of speed.

The tabulated observations of Fig 4.7 highlight also the "price" for allowing the control point to take on the most optimum position, regardless of the given knot tangents. On average, the two respective tangents meeting at a knot point are

about 3° out. In other words, a correction of about 3°, on average, in the direction of the tangents is necessary to ensure a gradient continuous description. This, as Fig 4.8b shows when compared to Fig 4.8a, does not seriously "damage" the appearance of the captured outline. (Further thoughts and discussion of this point are deferred until chapter five, section 5.3.5, where a closer examination between the amount of deviation exhibited by two joining arcs and their corresponding tangents is presented).



a) b)

Fig 4.8 Shows conic outputs gained through the algorithms of a) Fig 4.5 and b) Fig 4.6.

In short, it can be concluded that both the algorithms output conic descriptions which are of acceptable quality. The fact that the conversion algorithm of Fig 4.6 (which chooses optimum values for S and, also, for the control point) uses considerably less conic sections gives it added appeal. This is gained, however, at the expense of first-order continuity, a feature that the conic algorithm of Fig 4.5 guarantees.

4.5 Rasterising Conic Sections

Having developed methods for modelling a given outline in terms of quadratic splines, this section focuses on representing the outline on to a digital display. With similarity to the Bezier cubic case (see section 3.6), the aim is to develop a working algorithm that could be employed in the IKARUS system. This section concentrates on producing a general conic algorithm, therefore, which works for all given (quadratic) curve segments. The discussion is limited in that it considers an approach based on the original work carried out by Pitteway [PITT 67]. This has attracted much attention since its introduction, and can be employed to yield the "best" possible grid positions for a given conic section.

4.5.1 Development of Algorithm

Pitteway originally chose to employ the mid-point criterion for sampling a point along the curve [PITT 67]. He extended the concepts used by Bresenham [BRES 65] for drawing lines to that required for displaying quadratic sections. The elegance of this algorithm lies in its simplicity of employing simple add operations to update the control and numerical components.

Although the algorithm works well in rasterising frequently used quadratic curves, it fails when the given curve, for example, is a thin ellipse. In this case, it is possible to cross both sides of the ellipse in one move and, therefore, miss the region in between. When this happens, Pitteway's algorithm generates either a horizontal or vertical line whilst attempting to track the given conic curve.

In order to simplify and "dampen" these effects, an algorithm which worked with four quadrants, rather than the eight octants, was developed [PITT 85]. This facilitates square-moves only, and operates in a similar manner to the implicit algorithm developed for the Bezier cubic case (section 3.6.2). Although this copes better than the original algorithm, there are cases where even this gets "lost", and as a consequence misses the end knot.

Banissi analyses both algorithms and suggests an "improved" version of these [BANI 90]. His technique utilises global control coupled with local control to ensure that the new algorithm does not get "lost". Although he highlights their performance through some chosen examples, the fact that his approach requires a number of algorithms to represent fully all cases, makes the whole process of rasterising conic sections rather complicated. Furthermore, the similarity of computational operations between the x and y branches (as presented by Pitteway) is at times not reflected in the resulting algorithm.

The improvements suggested by Pratt [PRAT 85] to Pitteway's original algorithm appear to be more in line with what's required here. He develops a "cure" for Pitteway's algorithm by constraining it to work within a quadrant. In other words, the given conic arc is assumed to require no quadrant changes and, therefore, problems of continuous quadrant changes do not arise.

In addition to the above restriction, the control component of the algorithm is modified to deal with special conic shapes such as thin and sharp-cornered arcs. For this purpose, two tests are introduced within the main loop of the algorithm. As Fig 4.9 depicts, the tests are located on only one branch. The branch which requires these depend on the shape of the conic arc and also on the direction (octant) in which the arc resides. If an elliptic curve is to be drawn in the first octant, for example, the two extra tests will be incorporated within the "y" branch. If it happens that either of these tests is found to be on the affirmative then a "safe" x-move would result rather than the "risky" y-move. In short, the tests are conditioned to identify and rectify potential getting "lost" situations.

Before giving details of the initial conditions, it is worth emphasising the fact that this algorithm (and those developed by Pitteway) can be conditioned to work with integer arithmetic. The conic spline has two knot points which are supplied as integers. Its control point is, normally, rounded to the nearest grid-position. The only other parameter left is the sharpness value. As mentioned by Pratt, this can be rationalised in terms of two integers [PRAT 85]. The implicit form (as

expressed by equation (4.4)) can be scaled to return integer coefficients for the conic arc.

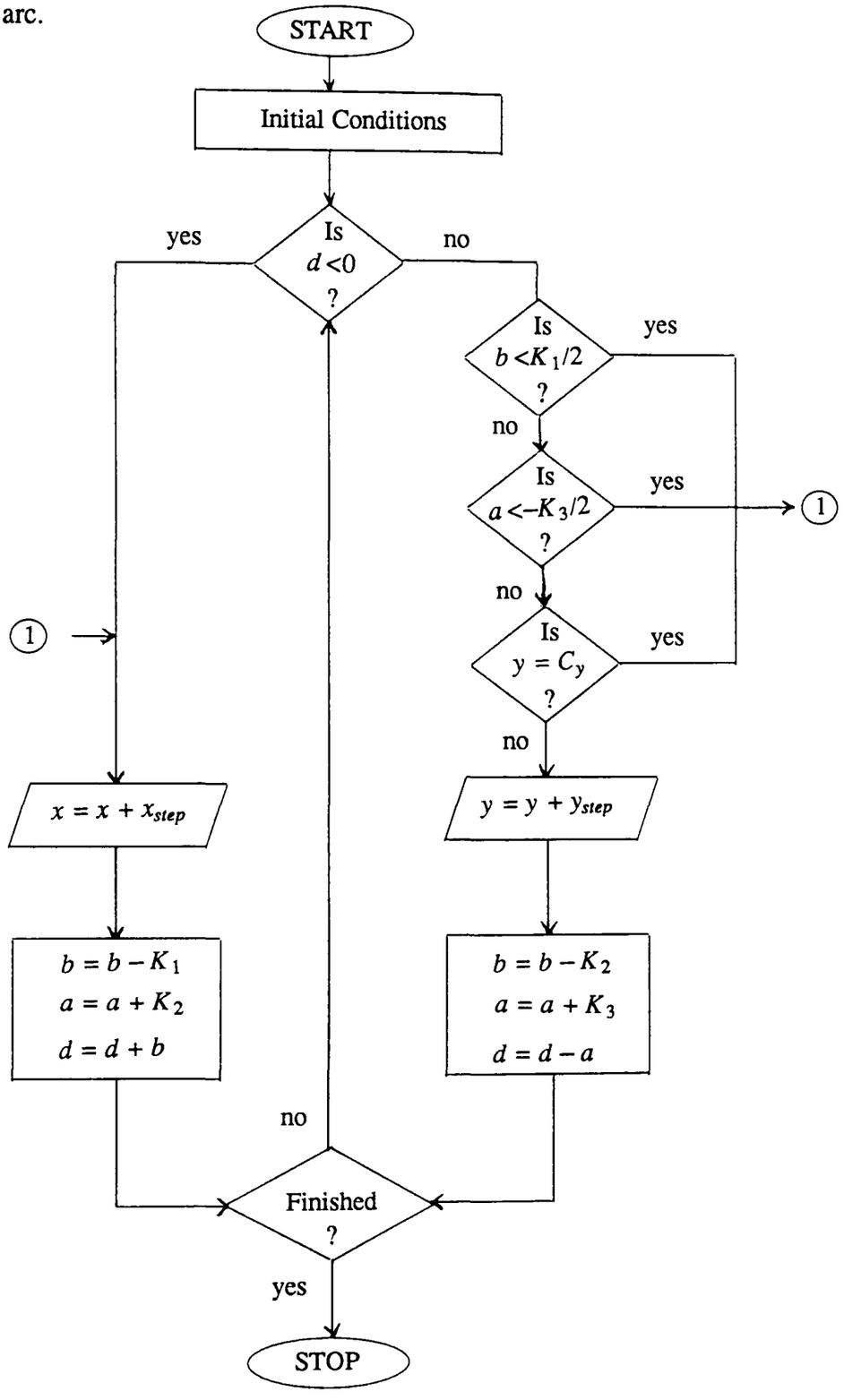


Fig 4.9 Shows the implicit rasterising algorithm for the first octant, where the additional tests are located in the y branch.

4.5.2 Initial Conditions

As mentioned in the previous section, the rasterising algorithm requires to identify "no lost" states in order to track the given conic spline correctly. These can be determined for the elliptic curve, for example, by evaluating the residue d_c at its centre. The algorithm, as given by Fig 4.9, will not get lost as long as the sign of its residue d corresponds with that of d_c. For the implicit equation (4.4), the residue at the centre of the ellipse can be expressed as follows:

$$d_c = \frac{\alpha v^2 + \beta u^2 + 2\gamma uv}{\alpha\beta - \gamma^2} . \qquad ...(4.14)$$

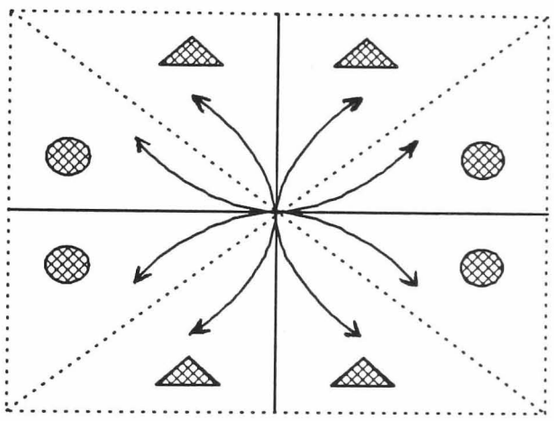


Fig 4.10 Illustrates octant symmetry which is employed in the implicit rasterising algorithm.

For the elliptic case, the denominator term of equation (4.14) is always positive [BANI 90]. This, therefore, means that the sign of the residue depends on the numerator term. If we take the eight octants as depicted by Fig 4.10, then the residue d_c is negative for the first octant and positive for the second. The sign for the other octants are gained by means of reflection, where the first quadrant is reflected onto the second, and both are reflected onto the third and fourth quadrants respectively. Fig 4.10 illustrates this feature through using the circle marker for negative d_c and a triangle for positive d_c.

In equation (4.14), the term in the denominator in fact yields the type of conic arc being rasterised. (Compare the denominator with the sharpness expression of equation (4.13)). This term vanishes (ie equals zero) for a parabolic arc and is negative for hyperbolic curves [BANI 90]. For the hyperbolic case, equation (4.14) returns the residue at its origin (that is, where the asymptotes intersect each other). Again, the rasterising algorithm is "not lost" whilst the sign of its residue d follows that of d_c. The residue d_c for parabolic splines is made equal to the corresponding value returned by the numerator of equation (4.14). This will generate, therefore, similar signs as those produced for the elliptic case. The "no lost" states for the various conic sections are summarised in the table of Fig 4.11.

No Lost States		Ellipse	Parabola	Hyperbola
1st, 4th, 5th and 8th Octant	d_{cn}	*negative*	*negative*	*negative*
	d_{cd}	*positive*	*zero*	*negative*
	d_c	*negative*	*negative*	*positive*
2nd, 3rd, 6th and 7th Octant	d_{cn}	*positive*	*positive*	*positive*
	d_{cd}	*positive*	*zero*	*negative*
	d_c	*positive*	*positive*	*negative*

Fig 4.11 Gives the signs of the parameters for detecting no lost states:
d_{cn} returns direction (octant) of flow,
d_{cd} yields type of conic section,
$d_c = d_{cn}/d_{cd}$, and highlights no lost states.

4.5.3 Results and Observations

To demonstrate that the algorithm of Fig 4.9 (employing the initial conditions as given in the previous section) can cater for all conic curves lying within a single quadrant, it was supplied with various sharpness values for a specified guiding triangle framework. In addition, the algorithm was adapted to work with integer values, so that each of the given sharpness value was translated into a rational

form. The chosen values for sharpness were 0.1 (1/10), 0.2 (1/5), 0.5 (1/2), 1 (1/1) and 3 (3/1), where numbers in brackets give respective rational form.

The guiding triangle, initially, used started at the origin (0,0), with a control point at (200,0), and an end knot at (200,-200). For this arrangement, the output produced by the rasterising algorithm is given in Fig 4.12. This illustrates that the algorithm can, as expected, generate pen positions for all the quadratic curves supplied. In addition, the rasterising process maintains the conic symmetry about its mid-point.

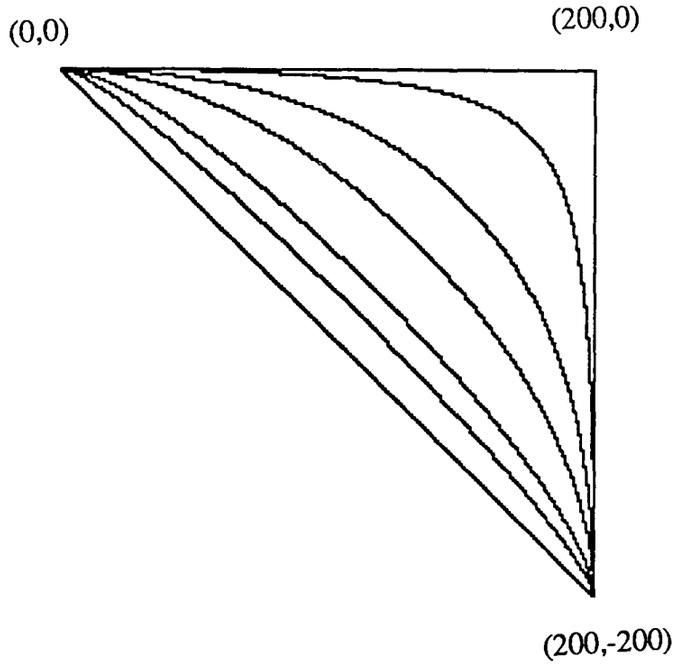


Fig 4.12 Shows the rasterised output as given by the conic rasterisation, for various values of sharpness.

To analyse the effectiveness of the algorithm, the guiding triangle is modified, whilst retaining the given set of values for the sharpness: In Fig 4.13, the position of the control point is changed; the u (x coordinate of the control point) is

subdivided in each case to result in Figs 4.13a, 4.13b and 4.13c respectively. As these diagrams depict, the algorithm does not get confused or lost, and generates an acceptable digitised output. Fig 4.14 highlights the case where if, instead of the control position, we decide to change the position of the end knot. Clearly, the rasterising algorithm performs on an equal merit to the other cases.

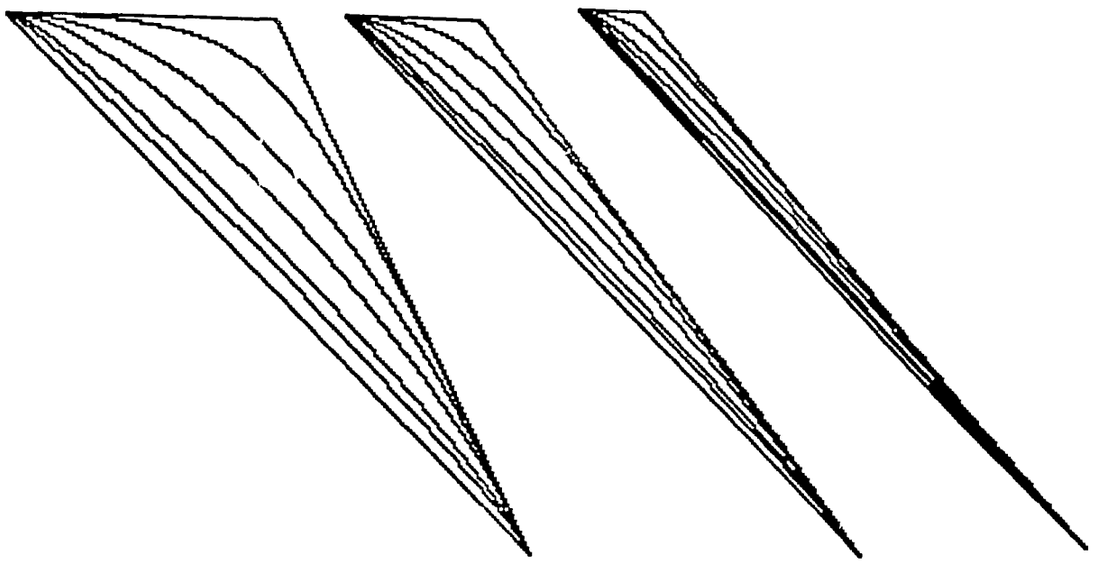


Fig 4.13 Depicts the performance of the algorithm when the given control point for the conic sections is varied.

In short, it can be concluded that the algorithm of Fig 4.9 is robust enough to cater for all conic arcs *not* requiring a quadrant change. Furthermore, the rasterising process has the attraction of capturing the conic's symmetry about its mid-point, as Fig 4.12 and 4.14b, clearly, demonstrate.

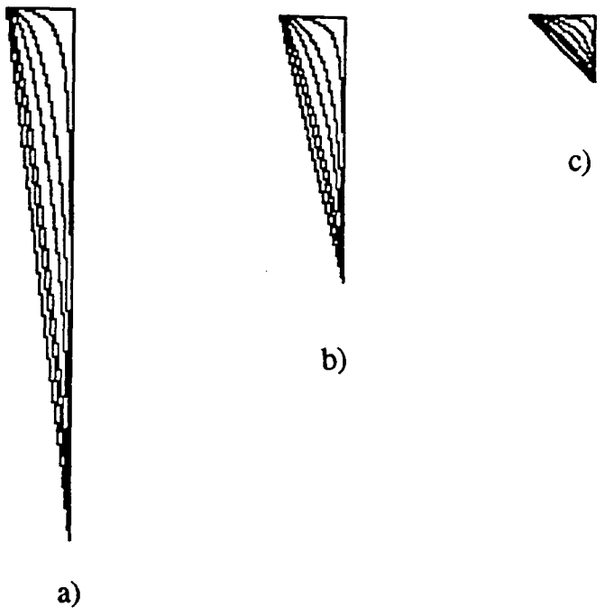


Fig 4.14 Shows outputs for the case when the end knot of the supplied conics is re-positioned.

4.6 Summary

This chapter begins by giving an historical background to the use of conic sections in mathematically representing contours of shape. After introducing the family of conics, their characteristics and features are then analysed. The mathematical form for representing conic sections is introduced. Both the rational form and the implicit form are given.

Having presented a detailed analysis, two methods for modelling outlines with conic splines are developed. The first approach guarantees gradient continuity between joining arcs. The second method works within a more relaxed environment and returns conic arcs which deviate the least from the given set of

data points. A technique for calculating the point-to-curve deviation is also presented.

Finally, an algorithm for rasterising conic curves is outlined. The method is based on the work carried out by both Pitteway [PITT 67, PITT 85] and Pratt [PRAT 85]. An approach for locating "no lost" states is also given. Through an example, the algorithm is shown to cater for all conic arcs that lie within a quadrant.

5.0 Algorithms for Bezier-Conic Conversions

5.1 Introduction

It is clear from the discussions and methods developed in chapter three and four that both the Bezier cubic and the general conic are capable of modelling contours of a given shape. Each mathematical form exhibits certain features and properties which are attractive for the purposes of interactive design work. In order to gain the maximum benefit from each modelling form, conversion routines are often employed which translate one mathematical description to another. These might take the task of converting Bezier cubic curve segments to a corresponding quadratic form or, on the other hand, transform a conic spline description to a respective cubic form. Whatever conversion is required, it is a fact of commercial life that these are needed often for no other reasons than for the sake of portability of design systems.

This chapter focuses on the conversion techniques required in relation to Bezier cubics and general conics. A detailed comparison of the two mathematical forms is presented. Both their attractions and restrictions are highlighted. Two algorithms for converting outlines, originally described in terms of Bezier cubic splines, to quadratic curve segments are developed. Their performance is analysed and assessed through a working example. Techniques for translating conic splines to a Bezier cubic representation are also presented. Their capacity for converting outlines is examined and possible limitations outlined.

5.2 Splines: Cubics versus Conics

When attempting to model mathematically a given character outline, the critical question of which description to employ often arises. Various factors such as the form of the supplied data, the type of continuity desired between joining arcs, the method of evaluating quality-of-fit and so forth, are common considerations made before incorporating a capturing routine in a design system. The decision to use Bezier cubic or conic splines is made with similar thoughts and observations. This section outlines the features and characteristics of both the Bezier cubic and

the conic spline. It attempts to highlight these with a view to practical and commercial requirements.

Many of the features of both representations have already been presented. In chapter two, the Bezier spline was presented as a versatile curve which exhibits inherent flexibility to represent difficult outlines of shape such as those containing points of inflection. Cusps and loops can be handled by Bezier splines without too much difficulty. This fact that the cubic spline can model curve outlines which can twist through space has greatly enhanced its popularity amongst designers from a variety of backgrounds. The application of this in the IKARUS system, for example, is restricted to modelling character outlines containing at most one inflection point within a defining Bezier cubic segment. Even in this situation, the designer is expected to supply the location of the inflection point as an input [KARO 87]. (In fact, the modelling stage frequently takes this as a knot point rather than a curve point.) Cusp, double inflection and loop exhibiting outlines are seldom captured by a single Bezier cubic spline; they are normally split and modelled by two or more splines.

Employing cubic splines in a design system offers the possibility of curvature continuity at the knot points of joining curve segments. Although this (and higher ordered continuity) might improve the aesthetic outlook of mathematically described shape contours, it also increases the computation necessary to gain an acceptable representation. As discussed in section 3.2.1, curvature continuity is a function of the two control points associated with each of the two Bezier polygons requiring second-order continuity. This, in practice, means that given a set of data points describing an outline, the "best-fitting" Bezier cubic spline will, not necessary, be the one which deviates the least from the given data. In other words, curvature (like gradient) continuity is gained at the expense of increasing the worst-case point-to-curve deviation. As the latter, normally, determines the quality-of-fit, more cubic segments may be needed to achieve a desired fit with second-order continuity as a constraint than otherwise.

Comparing the characteristics of a conic spline with that of the Bezier cubic, it is apparent that it (the quadratic arc) does not inherently exhibit the two properties discussed above, namely curvature continuity and points of zero curvature (ie inflection points). Pratt [PRAT 85] proposes a solution to this based on the sharpness value of a conic spline. His findings can be summarised as follows:

a) *Curvature continuity using conics.*
A formula for evaluating curvature at the knot points of a conic is derived by using the fact that, as the sharpness value increases from zero to infinity, the corresponding curvature at the knot points decreases from infinity to zero. This property could be exploited to return curvature continuous quadratic splines.

b) *Zero curvature for inflection points.*
As points of inflection are non-existent within a conic spline, an approach based on the curvature expression used in a) is developed. The principle of this is to make the curvature at the knot (where the inflection point is located) such that it is much smaller than any prevailing curvature at some distance away from this point. This can, therefore, enable piecewise conic curve segments to approximate zero curvature at a point of inflection.

If a design system required the application of curvature continuous conic splines then the method developed by Farin [FARI 89] can be incorporated. He uses the concepts outlined in a) to evolve an approach which uses the rational (parametric) form for describing conics. The method is based on a recursive technique that yields sharpness values which are "close" to the gained conic-fit, and which have been processed to ensure curvature continuity.

The above discussion again indicates the fact that curvature continuous splines are produced through compensation of the "best-fitting" mathematical description, which strays the least from the given set of data points. For the conic case, the

"best-fitting" sharpness value is adjusted to yield a desired conic-fit with second-order continuity.

In the field of typography, it is paramount that the modelled outline should preserve the unique features of a given font type. This, therefore, means that gaining a desired fit might not be enough if the resulting character outline fails to achieve aesthetic acceptability. Although the IKARUS system converts the given outlines of characters to various mathematical forms (including Bezier cubics), the modelling stage ensures that the continuity of arcs is only first order. This implies, therefore, that restricting the arcs to be only gradient continuous does not seriously hinder the quality of the captured font.

The IKARUS system is not alone when it comes to preferring computational simplicity for capturing outlines of fonts. The C curve, developed by Conographic Corporation (Irvine, USA), is based on the conic form rather than the cubic description. Villalobos [VILL 87] discusses the reasons for preferring the C curve as basis for a design system, and highlights computational practicality and efficiency as being the main attractions.

In passing, it should be mentioned that since a Bezier cubic curve segment can represent (at least) one inflection point within its description, it is able to capture more of a given outline than a corresponding quadratic arc. This might be important for reasons of storage compactness. In practical terms, however, the hardware memory available in modern design systems tends to make this a minor consideration rather than major.

The rate at which each character's outline can be mathematically described forms an integral component of a capturing process. It is clear from the discussion and methods developed for the Bezier cubic spline in chapter three that it employs recursive techniques for returning mathematically described outlines. These are rather time-consuming, especially, when there are thousands of fonts (with hundreds of characters) in the field of typography to convert. It takes approximately 30 minutes (of CPU time) to transform an IK font type to a

corresponding Bezier cubic form (see section 3.4.4). For a similar conic spline conversion, it takes only about 5 minutes. Clearly, one advantage of employing the quadratic description is the speed of conversion it offers.

The Bezier cubic, expressed parametrically, offers eight degrees of freedom. Its algebraic counterpart returns nine degrees of freedom [SEDE 85]. These are reflected in their mathematical forms where, in the case of the non-rational parametric form, the two knot and the corresponding two control points fully describe a Bezier cubic set-up. For the algebraic form, as expressed by equation (3.19), nine coefficients are required for a complete description. Conic splines, in general, offer seven degrees of freedom: two knot points, a control point and the sharpness parameter. It is apparent, therefore, that the non-rational parametric cubic spline offers one degree of freedom more than the general quadratic spline. This accounts for the fact that shapes such as a cusp and a loop can be modelled by a single Bezier cubic spline. These, as discussed in section 5.3.1, can be represented by employing two (or more) conic splines.

Although the polygonal set-up of the Bezier cubic arc gives some indication (as mentioned in section 3.2.1) of the resulting curve outline, in some cases (especially for loops and cusps) it is difficult to predict the nature of the shape from the polygonal framework. If we limit the locations of the two control points so that they do not cross (by setting both r and s less than one for the implicit form derived from Fig 3.12), we will then restrict the Bezier form to capturing "conic-like" shapes and those that contain a single point of inflection. This might assist in gaining a better appreciation about the form of the described outline, but this limitation also prevents the representation of extreme hyperbolic shapes (such as those possible for conic splines when the sharpness value tends towards infinity).

The triangular framework for the conic spline, on the other hand, appears to be a better set-up for gaining an insight to the type of shape being described. As cusps, inflection points and loops do not occur within a quadratic curve segment, the shape modelled by the guiding triangle is either a part of an ellipse, a

parabola or a hyperbola (depending on the value of the sharpness). Given a conic spline, with its respective sharpness parameter, the corresponding shape outline can be constructed through the evaluation of the shoulder point. Clearly, the triangular framework for the conic splines has attractions when it comes to the manual construction of curve shapes (a point illustrated through graphical examples by Liming [LIMI 79]).

When it comes to rasterising cubic and conic arcs (a topic discussed in sections 3.6 and 4.5 respectively), the best approach for both is via the corresponding tracking algorithms. In the case of the cubic algorithm, it requires an additional preparation stage to safeguard it from getting into a "state of confusion". These type of problems do not occur for the conic case since it does not exhibit the property of self-intersection. Problems of representing sharp ellipses, in the case of the latter, are resolved through employing the technique developed in section 4.5.

Pavlidis [PAVL 83] prefers to employ conic splines for the fact that they have a longer and solid historical background, and more importantly, they are much simpler to work with then cubics when it comes to finding the intersection of a line with a curve. The solution for this problem is required in many applications of computer graphics (for example, hidden surface removal) and its demand has led to the development of recursive subdivision techniques for cubic splines.

In summary, it can be seen that both the cubic and the conic spline offer features and properties that make them suitable for capturing and representing contours of shape. The fact that most of the benefits in using Bezier cubics can be approximated by conic splines, makes the latter a real and practical alternative to replacing the iterative and time-consuming methods of the Bezier cubic.

5.3 Bezier to Conic Algorithms

Having presented a comparison of the two most used mathematical descriptions, this section considers the transformation process necessary to approximate Bezier

defined outlines with general conic sections. The mechanism employed for a degree reduction of a Bezier spline (as discussed in section 2.2.4) is utilised here. Converting a Bezier cubic to a quadratic form, through the use of only control points, results in a parabolic curve. In other words, a Bezier quadratic (expressed solely in terms of its single control point) represents a parabola. Although this might be required in some cases (see chapter six), this section considers and develops techniques which convert the Bezier cubic outline to any of the curves belonging to the family of conics.

5.3.1 Concept of Conversion

Before presenting the techniques for conversion, it is worth considering the form and nature of the Bezier cubic spline in terms of its general quadratic representation. If a Bezier cubic curve was allowed, for example, to generate a cusp and a loop within its polygonal set-up, then these could only reasonably be approximated by using two or more conic sections. This would require splitting the Bezier shape to accommodate a quadratic representation. Possible conic modelling of such shapes is illustrated in Fig 5.1. (Note the zero-order (conic) continuity for the cusp).

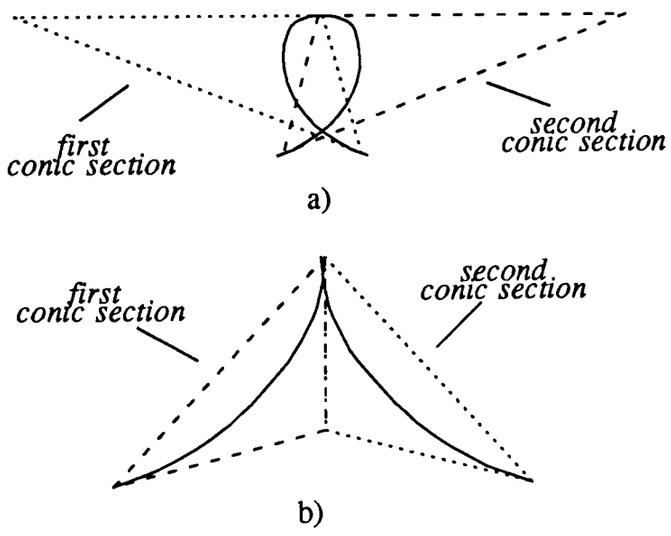


Fig 5.1 Gives an illustration of how a quadratic representation could be gained for the case of: a) a loop and b) a cusp.

As mentioned in the previous section, the Bezier cubic spline is normally used to represent at most one point of inflection. This reduces, therefore, the problem of conversion to three possibilities: The first case is where the Bezier spline contains an inflection point. This will need, at least, two conic splines, joined together at the occurring point of zero curvature. The second case is where the two knot tangents of the Bezier curve cannot return an acceptable control point for the approximating conic. This again will require two or more conic sections. In the third (and ideal) case is where a single conic-fit will suffice. This will occur for Bezier arcs whose tangents can directly yield the corresponding position for the conic control point. Fig 5.2 gives a graphical illustration of the three possibilities.

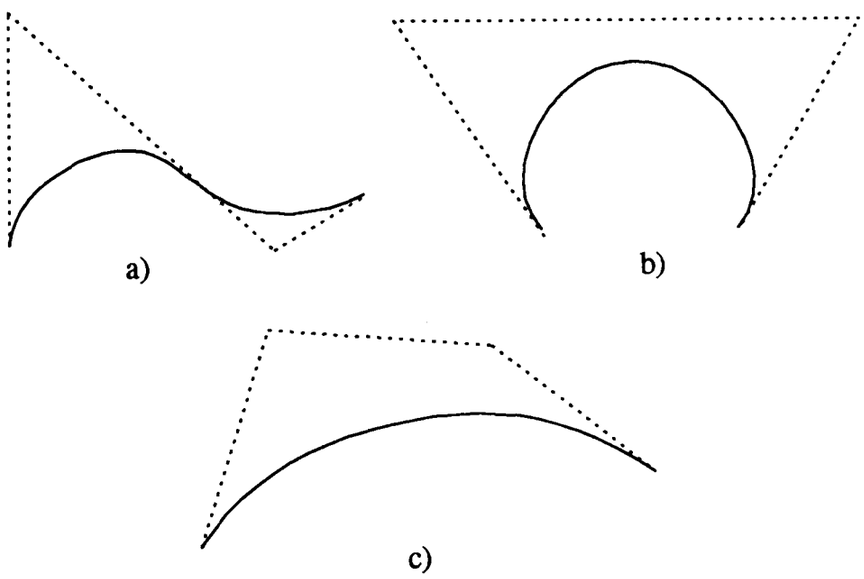


Fig 5.2 Shows three types of Bezier curves that will require a conic solution:
a) and b) will require two or more quadratic arcs, whilst
in c) a single conic approximation is possible.

5.3.2 Preparation for Conversion

In view of the discussion made in the previous section, the Bezier cubic curve needs to be processed such that its in a form which will enable a satisfactory

conic approximation. The Bezier spline is required, therefore, to have tangents which do intersect in the manner depicted in Fig 5.2c. This means a method for locating the point of inflection, and a strategy for finding a suitable point to subdivide the Bezier spline (when necessary) needs to be developed.

When deciding where to split the given Bezier cubic curve of the type shown in Fig 5.2b, two considerations are necessary: The first is that the given curve can be subdivided at its mid-point (t=0.5), using the approach presented in section 3.6.1. This will yield two Bezier curves which are in the required form of Fig 5.2c. Although this method is satisfactory and is employed in the algorithms presented later in this chapter, an alternative approach is to subdivide the given Bezier cubic curve at various points and, at each point, determine the corresponding conic-fit. By comparing the worst-case point-to-curve deviations (based on the method outlined in section 4.2.5), the "best" possible approximation can then be made. This approach leads to a graph similar to that shown in Fig 5.3. In practice, however, this approach tends to return the "best" splitting values which lie close to the given curve's mid-point, enforcing the opinion that the first (non-recursive) approach suffices for most situations.

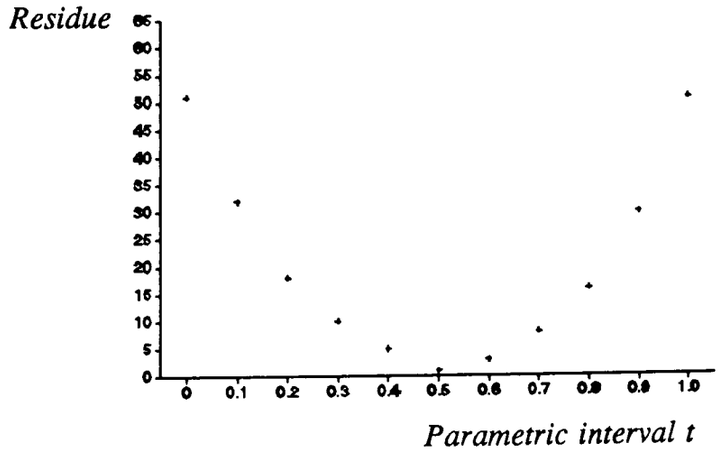


Fig 5.3 Depicts a typical graph of how the residue returned by a conic approximation varies for a Bezier curve split at specified values of t.

In passing, it should be emphasised that the situation as depicted in Fig 5.2b requires an efficient approach for its detection. If its allowed to be modelled by the method outlined in section 5.3.3, for example, then a "best" fitting conic spline will result which would be the complement of the given Bezier shape. The situation is shown in Fig 5.4. The detection process, therefore, should be based on the tangent directions of the knot points. The principles of such an approach are discussed by Hu and Pavlidis [HU 91].

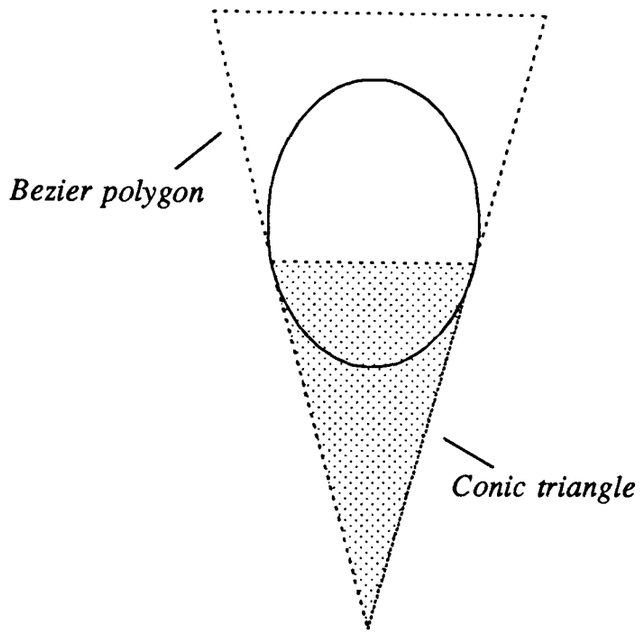


Fig 5.4 Highlights the case where the conic representation returns a shape which complements the given Bezier curve.

For purposes of finding a point of inflection in the given Bezier cubic curve segment, various approaches could be applied: The recursive approach for locating the most suitable place to subdivide the given curve (as discussed above) can be used for this purpose. Implementation of such a technique will yield that the "best" place to split the Bezier curve is at its point of inflection. The technique is thus being used indirectly to locate the point where the curvature of a Bezier cubic curve goes to zero. Although this approach can be used, it is neither the most elegant nor the fastest way of achieving the desired result.

A technique based on the work of Stone and DeRose [STON 89] could be developed to detect any points of inflection. As mentioned in section 3.2.3, they characterise the Bezier cubic to determine whether the resulting shape contains a cusp, a loop or any inflection points. By computing the corresponding characteristic point and testing it against the region boundaries, it is possible to detect whether a Bezier curve exhibits a point of inflection. Although it is not made clear in [STON 89] exactly how to use this to locate the point of inflection, a possible approach would be to subdivide the given Bezier curve and then to characterise the two Bezier arcs to evaluate which contained the point of inflection. This process would continue until for a specified t value, neither of the two resulting Bezier arcs displayed an inflection point; implying curve point $P(t)$ is the desired solution.

The technique employed for detecting and locating the point of inflection, for the Bezier to conic algorithms, is based on attaining curvature information through a frequently used mathematical expression. This returns the appropriate signed curvature value $k(t)$ for a curve defined parametrically by t. The equation for the curvature has been given by various authors including Faux and Pratt [FAUX 79] and Roulier [ROUL 88]. This is expressed in terms of the corresponding first and second parametric derivatives (that is, $x'(t)$, $y'(t)$, $x''(t)$ and $y''(t)$ respectively) for the curve:

$$k(t) = \frac{x'(t)y''(t) - x''(t)y'(t)}{[x'(t)^2 - y'(t)^2]^{3/2}} . \qquad ...(5.1)$$

The way equation (5.1) is used to detect an inflection point is by checking if the sign of the curvature value $k(t)$ changes within a Bezier curve segment. If there is no change of sign then the curve does not contain a point of zero curvature. If, on the other hand, a change of sign in $k(t)$ is discovered than there exists an inflection point between the points where the sign change has taken place. By means of successively subdividing this interval and testing for a sign change in equation (5.1), it is possible to converge quickly to the point where $k(t)$ will equal zero; returning the location for the "best" point to split the Bezier curve. (Alternatively, an analytical approach could be used to find the point of inflection, see Patterson [PATT 88] for a general discussion).

5.3.3 Conic Capture Through Tangents

With the Bezier cubic curve initially processed to have knot locations which make it possible to have a suitable conic approximation, this section presents the first conversion algorithm. The method has similarities with the approach used to model IK data using general conic arcs (section 4.4.1) in that it ensures that the quadratic approximation exhibits, at least, first-order continuity. From this, it is clear that the IK to conic algorithm could be employed to gain a conversion between Bezier cubic to conic sections. In this case, the Bezier described outline would then need to be expressed in terms of discrete data points in the manner highlighted in section 4.4.1. Although the parametric nature of Bezier arcs makes it relatively simple to attain data points describing the outline, each arc shape will require a different number of data points to fully represent it. This will lead to problems of either having too many, or not enough, points for gaining a conic-fit which is accepted aesthetically. Furthermore, it is desirable to have a direct conversion algorithm, based on the polygonal framework of Bezier splines, rather than on a process of summing the individual data points.

In line with the IK to conic algorithm (of section 4.4.1), the "best" location for the quadratic control point is evaluated using the knot point tangents. The intersection point of these returns the desired control coordinates u and v. This then leaves the calculation of the corresponding "best" value for the sharpness.

To gain an expression for the sharpness value S, the steps outlined in section 4.4.1 are utilised. The principal difference is the fact that the discrete summation of data points is replaced by an integral, which implies that a conic-fit is acquired for a mathematically described outline. Making use of the implicit form for the general conic as given by equation (4.5), a solution is desired which minimises this through yielding the "best" values for S^2. In other words, a solution to the following is required:

$$\frac{\partial}{\partial(S^2)} \int_0^1 d(x_i,y_i)^2 \, dt = 0 \, . \qquad \ldots(5.2)$$

The curve points x_i and y_i are expressed as a function of the Bezier cubic parametric variable t, as given by equation (3.13). If we perform the partial derivatives as instructed by equation (5.2), and compile similar terms, then it can be shown that the "best-fitting" sharpness value for the given constraints can be evaluated through the following:

$$S^2 = \frac{\int_0^1 \Delta_k^2 \, dt}{\int_0^1 4\Delta_i\Delta_j \, dt} \, . \qquad \ldots(5.3)$$

The integration, with respect to t, is computed just once, at the beginning of the conversion process. As the limits for the integral are zero and one, the resulting expressions for the numerator and denominator of equation (5.3) are found to be compact, and whose values are gained through the sole use of u, v and the given Bezier cubic polygon. Clearly, the "best-fitting" conic approximation is made without having to resort to any iterative or recursive procedures.

The goodness-of-fit is assessed through the employment of equation 4.6. This appeared to be the simplest way of gaining the worst-case point-to-curve deviation. The "best-fitting" conic curve was compared with some discrete data points acquired through the parametric form for the Bezier cubic as given by equation (3.1).

To summarise this conversion process, the complete algorithm is depicted in Fig 5.5. As is apparent from this, if the resulting conic-fit fails to satisfy either a desired accuracy value or is not accepted on grounds of aesthetic appeal, then the given Bezier arc is subdivided in the manner discussed in section 5.3.2 and represented by two conic sections. Fig 5.5 does not show the processing of line segments. These normally are not expressed in terms of a Bezier "arc" (where it is necessary to set the control points P_1 at $\frac{P_3}{3}$ and P_2 at $\frac{2P_3}{3}$ relative to start knot P_0), but as a line segment described completely by the two knot positions.

The conic representation of this is simply attained by setting the sharpness value equal to zero.

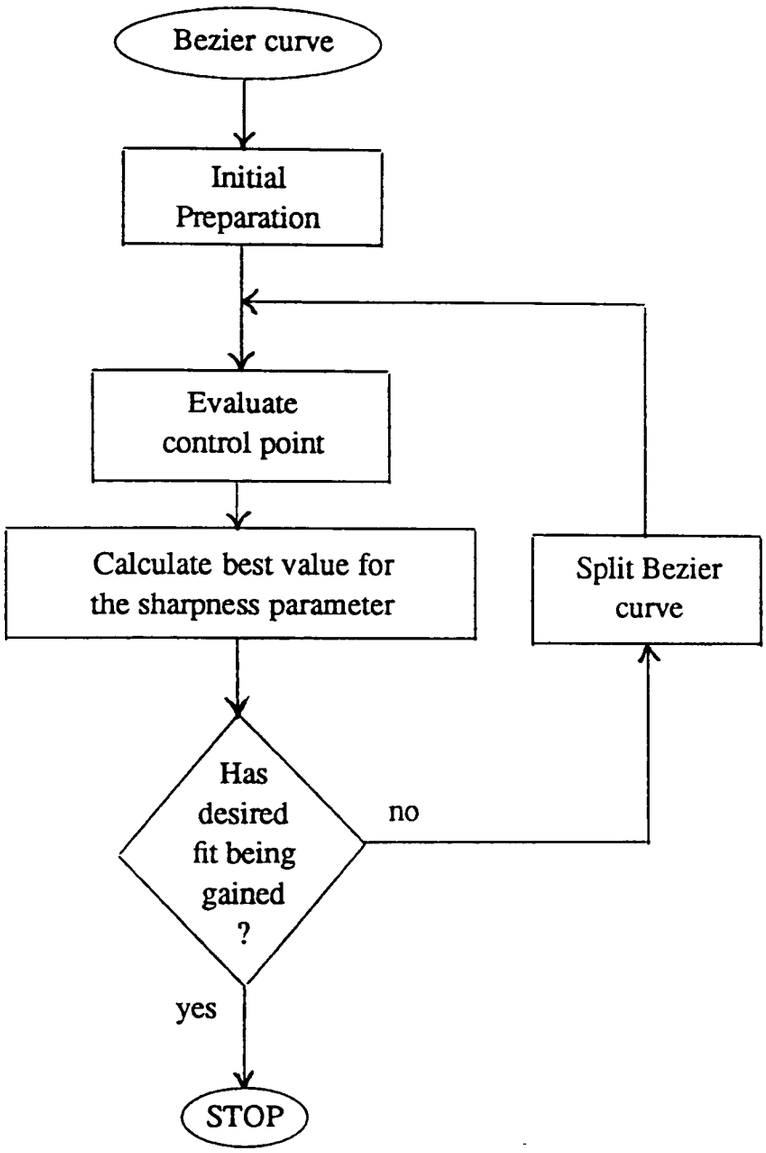


Fig 5.5 Shows the Bezier to general conic conversion algorithm which maintains first-order continuity.

The next section presents an alternative approach for converting Bezier cubic arcs to general quadratic splines. In section 5.3.5, the performance of both the conversion algorithms are analysed. A comparison is presented which leads to highlighting further the capabilities of both approaches.

5.3.4 Conic Capture with Least Deviation

The method outlined in the previous section has the attraction of safe-guarding, at least, first-order continuity between joining conic arcs. Because of this, the approach confines itself to evaluating the "best" value for the sharpness. As the Bezier described outline is initially processed to be conic-like, the respective conversion process should then return the "best" possible conic-fit. Using the knot point tangents as a basis for the control point, however, means that the resulting conic-fit will not necessarily be the one which deviates the least from the given Bezier curve. This section presents a conversion algorithm which has the attraction of providing general conic approximations that deviate the least from the supplied curve. Furthermore, the equations developed are of a linear, non-iterative, type.

With analogy to the previous section, the method developed for IK data to general conic representation (section 4.4.2) could be used for the purpose of converting Bezier cubic curves to conic splines. Details of its employment can be found in the paper written by the author [HUSS 89]. Although the approach is acceptable, it has the drawback (as mentioned in section 5.3.3) of either introducing too many, or not enough, points in describing the Bezier outline. In [HUSS 89], for example, each Bezier curve segment was represented through 1001 curve points. Clearly, this originates redundancy in the conversion process, and leads to an algorithm which cannot be described as being computational efficient. Even if fewer data points were employed, a decision (or better still, a criteria based on curvature) will be needed to evaluate the number of data points which will suffice. In short, the method of section 4.4.2 is ideally suited for capturing a given set of data points, and requires "improving" to represent a mathematically described outline.

In order to avoid working with non-linear expressions, equation (4.11) is utilised such that the solution to the three unknown α, β and γ is required. Unlike equation (4.12), however, the "best" values for these parameters is gained through

minimising the integral of the squared residue (as given by equation (4.11)). This can be expressed as follows:

$$\frac{\partial}{\partial\alpha}\int_0^1 d(x_i,y_i)^2\ dt = 0\ ,$$
$$\frac{\partial}{\partial\beta}\int_0^1 d(x_i,y_i)^2\ dt = 0\ ,$$
$$\frac{\partial}{\partial\gamma}\int_0^1 d(x_i,y_i)^2\ dt = 0\ . \qquad \text{...(5.4)}$$

With analogy to section 5.3.3, the x_i and y_i are expressed in terms of the parametric variable t of equation (3.13). The integration, again, is performed once in the beginning of the conversion algorithm. Although this, and the computation of the partial derivatives, can be carried-out manually, the expressions tend be rather lengthy and ideally require the use of an algebraic manipulation package (such as REDUCE). Having performed the evaluations as instructed by equation (5.4), a three by three (linear) matrix is set-up, which can be solved by using the process of Gaussian elimination.

The solution of the conic parameters α, β and γ makes it possible to use equation (4.10) to calculate the coordinates for the control point. The corresponding sharpness value and, therefore the type of conic section being employed, is obtained via equation (4.13).

The steps involved in performing the conversion are graphically outlined in Fig 5.6. If a single conic-fit does not suffice, then the given Bezier spline is subdivided and represented by two "best-fitting" conic splines. This process of subdivision continues until a desired conic approximation results.

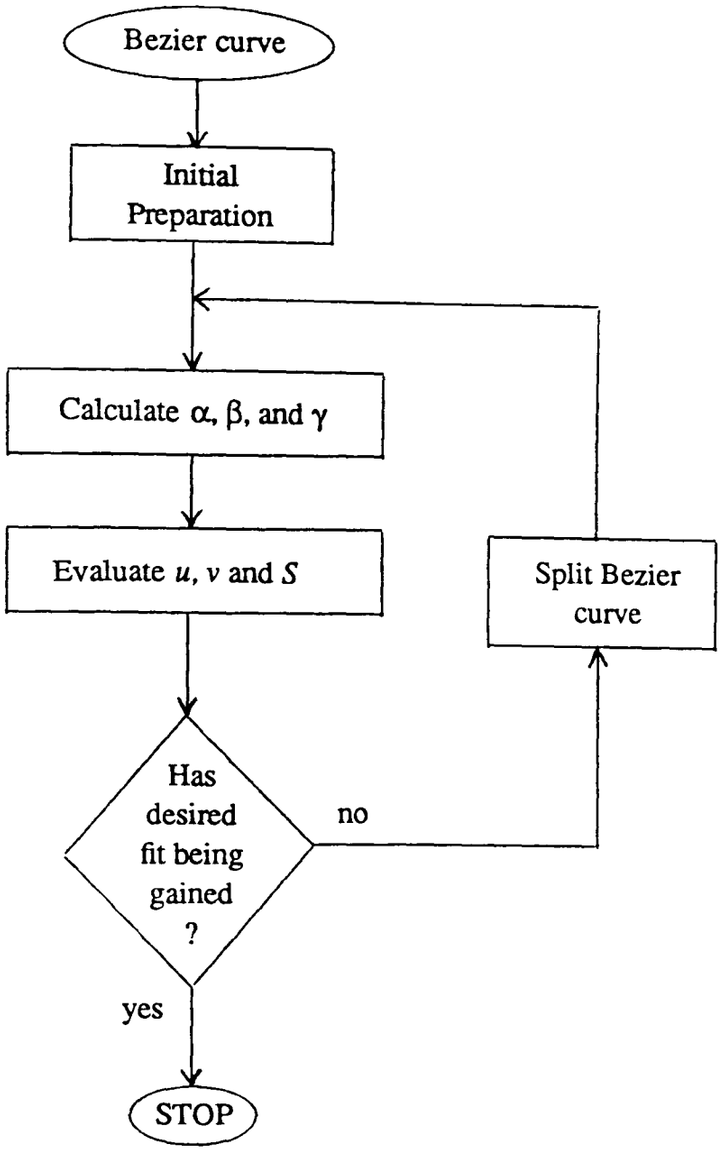


Fig 5.6 Gives the steps for a Bezier to general quadratic conversion which yields the closest approximation.

5.3.5 Analysis and Performance

The effectiveness of the two Bezier cubic to general conic conversion algorithms is assessed through their capture of a given 'S' character. Developed by Osland [OSLA 86], the character forms part of a font which uses Bezier cubic arcs for its description. The outline of the 'S' character is shown in Fig 5.7. This is produced by using the parametric form as given by equation (3.3). The output shown in Fig 5.7 is gained through rounding respective x and y values to their nearest mesh points. It is clear from the location of the knot points that, the

character consists of six (Bezier) curves, labelled A, B, C, D, E and F, in Fig 5.7. Although it is necessary to split curves C and F (because each contains a point of inflection), the remaining four curves were also subdivided to facilitate an acceptable conversion, resulting in arcs A1,A2 for curve A, B1,B2 for curve B,...., F1,F2 for curve F. Apart from curves C and F (which are split at their respective point of inflection), the other four curves are subdivided at their mid-point (t=0.5).

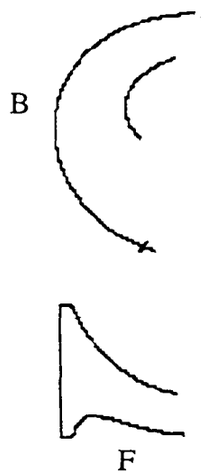


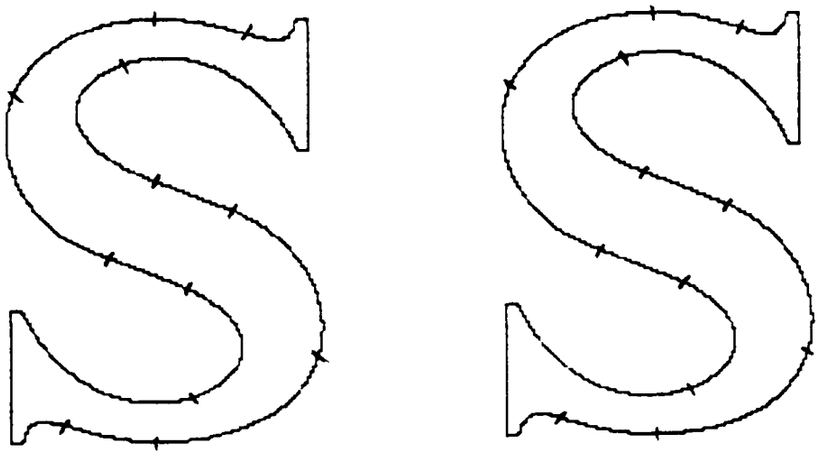
Fig 5.7 Depicts outline of the 'S' character, used as an input for the Bezier to conic conversion algorithms.

The Bezier described character is then processed by the two conversion algorithms. Their respective output is shown in Fig 5.8. This is gained by utilising the parametric form for the general conic as expressed by equation (4.3). In a similar fashion to the output of Fig 5.7, each x and y curve point returned by this equation is rounded to the nearest integer.

If a visual comparison is made between the given Bezier outline and the two generated conic approximations (that is, Fig 5.7 with Figs 5.8a and 5.8b, respectively), it can be seen that most of the distinct features have, to a good degree, been captured. The fact that all the Bezier curves had been split (resulting in non-integer mesh points that required rounding), and the conversion process

having to employ two conic arcs for each Bezier curve, has not seriously handicapped the quality of the two approximations.

A closer examination of the output is manifested in Fig 5.9. This illustrates the difference in terms of the respective integer versions being exclusively OR-ed. Although, in theory (that is, if floating-point values for the x and y positions were being used to both evaluate and represent the respective outputs) there would be negligible differences, the output of Fig 5.9 demonstrates that both algorithms do yield a few integer positions which are out of step with the given Bezier version. This shows the accumulative effect of rounding to integers the split points and the respective curve points, a effect which is most apparent for given curve E, where splitting at its mid-point results in a floating point value of 56.5 (for the y-axis).



a) b)

Fig 5.8 Shows the output from the two conversion algorithms:
a) tangent continuous and b) most accurate.

Comparing the difference output produced by the conversion algorithm of Fig 5.4 with that generated by the steps of Fig 5.5 (that is, Fig 5.9a with Fig 5.9b), it is clear that the closest fit, as expected, is returned by the latter process. This fact is verified by the tabulated results shown in Fig 5.10. Allowing the position of the control point to vary independently of the given knot tangents results in a

conic approximation which is a factor three to four times closer than otherwise. In other words, the observations noted in Fig 5.10 show that first-order continuity is maintained at the expense of increasing significantly the point-to-curve deviation. Since the quality-of-fit is measured in terms of this deviation, more quadratic curves will result for a gradient continuous description than an approach (such as of section 5.3.4) which is allowed to convert within a more relaxed constraints. (This point has already been verified in section 4.4.3 through the capturing process of IK data to general conic format).

a) b)

Fig 5.9 Highlights the difference between the given outline and the resulting outputs: a) tangent continuous and b) most accurate.

Fig 5.10 also shows the amount of "discontinuity" between joining arcs for the conversion algorithm described in section 5.3.4. The least acceptable value is 3.03313 degrees, occurring at the joining knot of curves A1 and A2 and, of curves D1 and D2. Although this appears to be relatively higher than desired, it is acquired through increasing the accuracy of the conic approximations; a point demonstrated by Fig 5.9b when compared to Fig 5.9a.

In short, it can be concluded that both the Bezier cubic to general conic conversion algorithms are suitable approaches, which return an acceptable output. The algorithm illustrated in Fig 5.4 has the attraction of yielding a first-order continuous description, whilst the second approach (Fig 5.5) dispenses with gradient continuity and returns conic approximations which deviate the least from the supplied outline.

Bezier to Conic Conversion	**Algorithm of Fig 5.4**	**Algorithm of Fig 5.5**	
Input Curve	Worst point-to-curve deviation	Worst point-to-curve deviation	Difference in joining tangents *degrees*
A1	0.19544	0.06206	3.03313
A2	0.32054	0.07757	0.73976
B1	0.03365	0.01125	0.64190
B2	0.05321	0.01697	0.05202
C1	0.03634	0.00804	0.66219
C2	0.16060	0.03446	
D1	0.19545	0.06206	3.03313
D2	0.32054	0.07757	0.76096
E1	0.03283	0.01043	0.67603
E2	0.05335	0.01735	0.05663
F1	0.03727	0.00856	0.65490
F2	0.14748	0.03247	

Fig 5.10 Shows tabulated results for the two Bezier to general conic conversion algorithms.

5.4 Conic to Bezier Conversion

The techniques developed, in chapter 3, for employing Bezier splines to describe contours of shape have highlighted the fact that these require considerable processing time to compute. Converting IK data to conic splines, on the other

hand, takes relatively little computation and is, therefore, performed at a much faster rate than its Bezier cubic counterpart. This section explores the possibility of converting general quadratic splines to Bezier cubic curve segments. Two approaches are developed. Their benefits and limitations are discussed in detail.

5.4.1 Concept of Conversion

The process of transforming the conic triangle to a respective Bezier cubic polygon involves replacing the single weight parameter (ie the sharpness value) by two control points. These are located on the corresponding tangents of the knot points. Unlike the general conic (except for the parabolic case), the positioning of the control points governs the resulting Bezier shape (see section 3.2.3). The process, therefore, can more precisely be described as attempting to position the control points on the guiding triangle (of the conic) such that it matches the shape and, thus, the influence of the sharpness value.

Obviously, the Bezier cubic spline is restricted to modelling shapes which are conic-like. If given outlines contain inflection points, cusps or loops, then these will require splitting in order to facilitate a conic approximation. Each resulting quadratic arc will then be captured by a corresponding Bezier curve segment.

The next section presents an algorithm which is based on the sharpness parameter for the general conic section, and employed as a subroutine in the IKARUS system. In section 5.4.3, an alternative method is developed. This uses the curvature at the knot points to gain a corresponding Bezier cubic approximation. Performance analysis of both approaches can be found section 5.4.4.

5.4.2 Approximation Through Sharpness

Since the requirement for the conversion process is, basically, to translate the sharpness value into the two control points for the Bezier cubic, this section presents a method for performing a direct translation. The method develops a conversion formulation which uses the mid-point of the conic arc to evaluate the corresponding "best-fitting" Bezier cubic control points.

Using the illustration of Fig 5.11, the values of r and s are required such that they yield a desired Bezier approximation to the given conic enclosed within the triangle P_0, P_1 and P_2. The parameters r and s have the same meaning and significance as discussed in section 3.5.2. For simplicity, it is assumed the given conic arc starts from an origin (0,0) so that P_0 can be taken as being equal to zero. The mid-point expression (where t=0.5) for a conic curve can be gained by using equation (4.3). This takes the form:

$$P(\tfrac{1}{2}) = \frac{P_1 S + \frac{P_2}{2}}{S + 1} . \qquad ...(5.5)$$

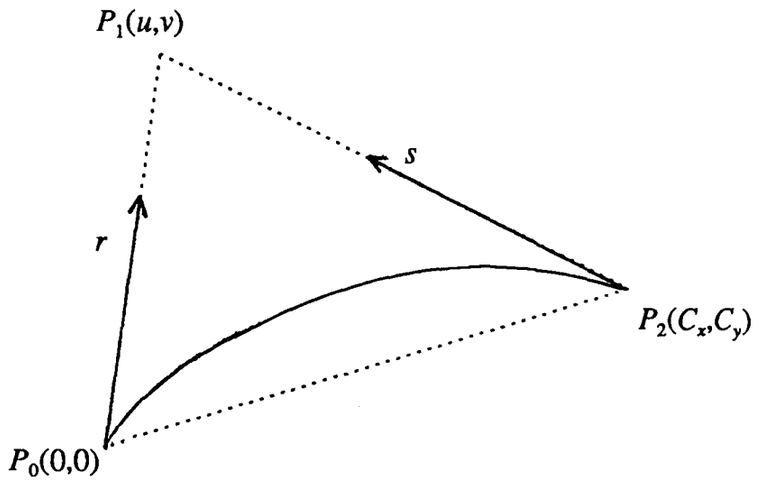


Fig 5.11 Depicts the relationship between the conic guiding triangle (P_0, P_1 and P_2) and the corresponding Bezier polygon (P_0, r, s and P_2).

In the case of the Bezier cubic curve, a corresponding mid-point formulation can be developed in terms of r, s, P_1 and P_2 (where these are as manifested in Fig 5.11) by utilising equation (3.3) with the respective substitutions of equation (3.16). Combining common terms and simplifying leads to the following expression:

$$P(\tfrac{1}{2}) = \frac{3P_1(r+s) + P_2(4-3s)}{8} . \qquad ...(5.6)$$

By letting r=s, equation (5.6) then takes the form:

$$P(\tfrac{1}{2}) = \frac{3r(2P_1 - P_2) + 4P_2}{8} \qquad ...(5.7)$$

By equating equations (5.5) and (5.7), the following conversion formulae evolves:

$$r = \frac{4S}{3(S+1)} \; . \qquad ...(5.8)$$

It is apparent from equation (5.8) that relatively little computation is required to translate a given conic description to a corresponding Bezier representation.

The positioning of the Bezier controls is such that for elliptic arcs, as the sharpness value S is increased from zero to one, r (as well as s) takes values from zero to 2/3. The conic spline yields a parabolic curve for S equals one. From equation (5.8), this for the Bezier case occurs at $r = s = 2/3$. Values of S greater than one and tending towards infinity generate hyperbolic arcs. These are converted to r (and s) values which range between 2/3 and one respectively.

Further discussion of this approach is deferred until section 5.4.4, where its performance is analysed together with the alternative approach. The next section outlines this alternative approach for converting general quadratic splines to a Bezier cubic curve format.

5.4.3 Conversion Through Curvature

As mentioned by Pratt [PRAT 85] and discussed in section 5.2, the curvature at the knots of a conic curve is directly related to its sharpness value. This section, as an alternative to the approach outlined in the previous section, presents a conversion method which matches the curvature at the knots of a conic arc with that for a Bezier cubic curve. The discussion here is limited to gaining appropriate values for the Bezier control parameters r and s. Development of implicit cubic forms resulting from these constraints are given in [PITT 91], a paper which is jointly written by myself and Pitteway.

With reference to [PRAT 85], the curvature at the two knots for a conic spline defined by a guiding triangle (such as that shown in Fig 5.11) can be expressed as follows:

$$\begin{aligned} \textit{at origin:}\quad k_{cs} &= \frac{\Delta}{S^2c^3}\ , \\ \textit{at end:}\quad k_{ce} &= \frac{\Delta}{S^2a^3}\ , \end{aligned} \qquad \text{...(5.9)}$$

where: Δ as given in equation (4.5),
c is length of line from P_0 to P_1, and
a is length of line from P_1 to P_2.

Corresponding curvature equations for the Bezier cubic curve can be gained in terms of the control parameters r and s. These take the form:

$$\begin{aligned} \textit{at origin:}\quad k_{bs} &= \frac{4\Delta(1-s)}{3r^2c^3}\ , \\ \textit{at end:}\quad k_{be} &= \frac{4\Delta(1-r)}{3s^2a^3}\ . \end{aligned} \qquad \text{...(5.10)}$$

Letting r=s, and matching curvatures at respective knots (by equating equations (5.9) and (5.10), results in the following conversion expression:

$$r = \frac{2S}{3}(\sqrt{S^2+3} - S)\ . \qquad \text{...(5.11)}$$

Compared with equation (5.8), the curvature conversion expression (equation (5.11)) requires a little more computation; the use of a square-root makes it unattractive. This, however, does ensure that the resulting Bezier spline captures more than just gradient continuity, the effect of which would be more apparent if the modelling process for the general conic is constrained to embody second order continuity.

As the sharpness values goes from zero towards infinity, the corresponding curvature at the knots goes from infinity to zero. Equation (5.11) is expected, therefore, to yield acceptable approximations for the elliptic, parabolic, and for hyperbolic arcs which have S values close to one. As the sharpness tends towards infinity, the resulting arc has more curvature at its centre than at its knot points; implying that conversions for such extreme quadratic curves may not be suitable through equation (5.11). This point, together with the approach of 5.4.1, are analysed and discussed further in the next section.

5.4.4 Analysis and Performance

Having presented two approaches for converting outlines employing general conic sections to a corresponding Bezier cubic description, this section assesses the merits of both techniques. Each is supplied with an array of conic sections. For this purpose, sharpness values of 0.25, 0.5, 1, 2, 4, 8, and 16 are used. The coordinates for the guiding triangle are (0,0), (100,200), and (280,30) representing the starting, control, and finishing points respectively. This results in the conic shapes depicted in Fig 5.12, which are supplied as input to the two converting algorithms.

The output from both approaches is depicted in Fig 5.13. A visual comparison of the two outputs shows that the curvature approach behaves reasonably for the elliptic and parabolic arcs, but fails to represent hyperbolic curves within an acceptable level. This fact is further highlighted by Fig 5.14, where the respective outputs have been exclusively OR-ed with that of the input (that is, Fig 5.13 with Fig 5.12). As Fig 5.14a manifests, the conversion process based on the sharpness (section 5.4.2) returns the best of the two outputs. Like the curvature approach, its major difficulty lies with the representation of the hyperbolic arcs. But, since it uses the mid-point of the conic as a basis for a conversion, the Bezier approximation maintains the overall shape of the given quadratic arcs (as Fig 5.14a clearly demonstrates when compared to Fig 5.14b).

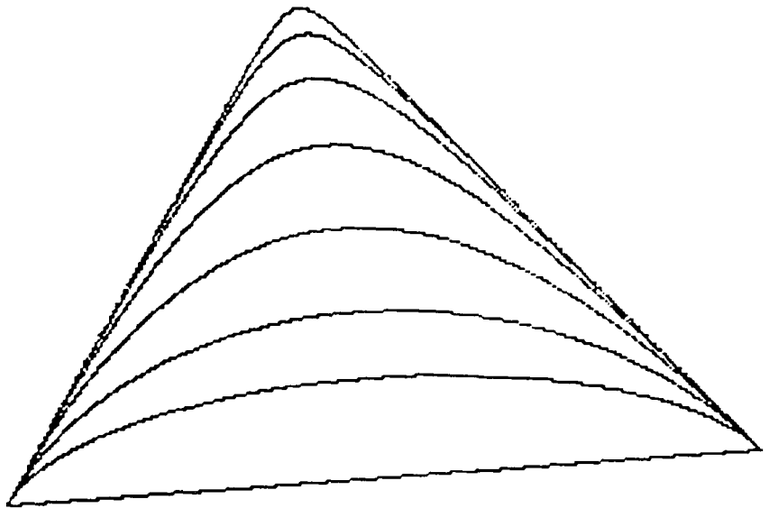
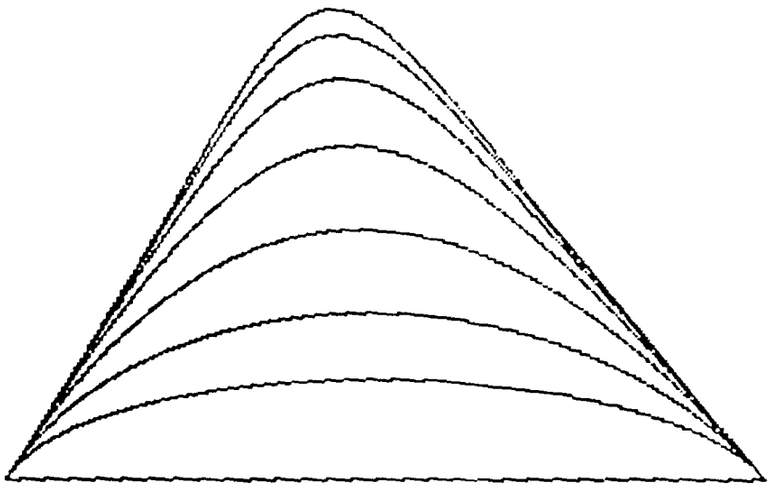


Fig 5.12 Gives an illustration of the quadratic shapes supplied as input to the conic to Bezier cubic conversion algorithms.

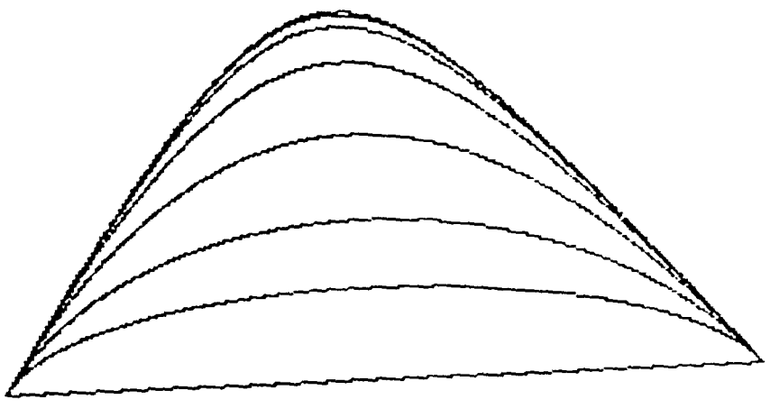
Fig 5.14 shows also the effectiveness of both approaches in capturing the curvatures at the two knots. In the case of the curvature matching approach of section 5.4.3, this is embodied in its conversion process, and the results at the knot points are satisfactory. Although the conversion method based on the sharpness is not constrained to match curvatures at the knots, it appears to have fulfilled this requirement through its accurate approximation of the given conic sections.

It is apparent from Fig 5.14 that the two approaches yield a perfect parabolic conversion. For sharpness value of one, both equations (5.8) and (5.11) give the corresponding value for r (=s) = 2/3.

In conclusion, therefore, it can be said that the best approach for converting outlines from a general conic form to a Bezier cubic description is through the sharpness approach (discussed in section 5.4.2). This returns acceptable approximations for all the conic sections, especially for the elliptic and parabolic arcs.

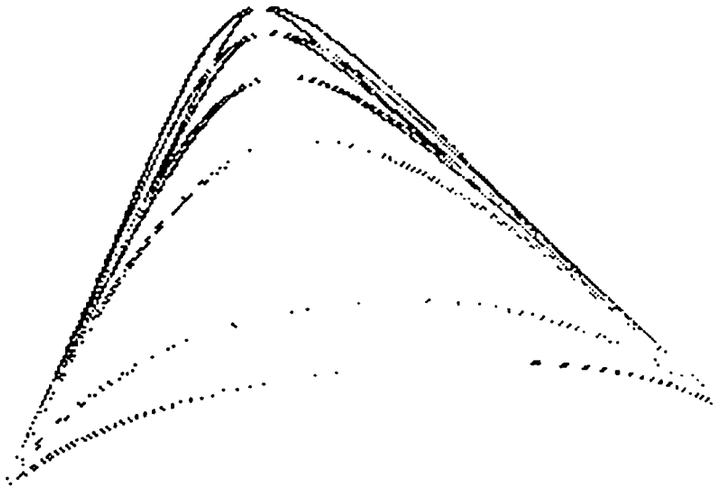


a)

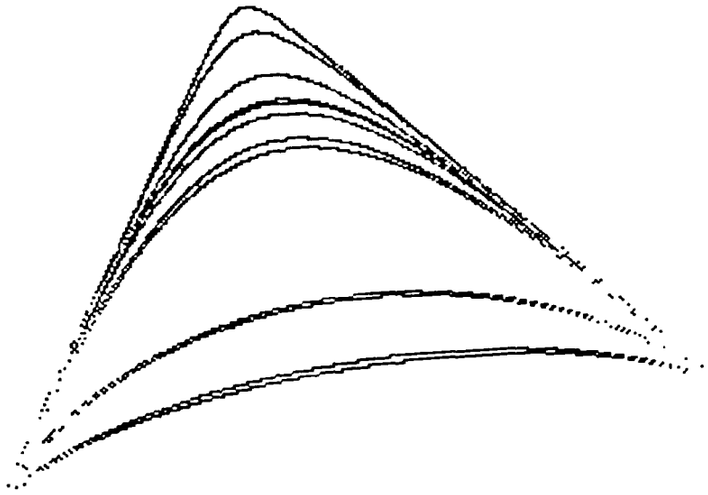


b)

Fig 5.13 Gives the output versions of both conversion algorithms: a) sharpness based and b) curvature based.



a)



b)

Fig 5.14 Shows the corresponding differences between the given outline and the Bezier converted outputs: a) sharpness based and b) curvature based.

5.5 Summary

This chapter gives a detailed comparison of the two most popular schemes for mathematically modelling outlines, namely the Bezier cubic and the conic spline. Both benefits and limitations are presented. It is acknowledged that most of the attractions of Bezier splines can, to a good degree, be approximated by a corresponding conic arc.

Two approaches for converting Bezier cubic curves to conic sections are presented. Both methods are based on an integral approach, which works with the control points rather than with discrete Bezier curve points. The first approach ensures the capturing process maintains the specified gradient continuity between conic arcs. As an alternative, the second method yields conic arc approximations which deviate the least from the given Bezier curve segment.

Finally, this chapter considers techniques for transforming a general quadratic spline to a respective Bezier cubic form. Two methods are developed. The first is based on the sharpness value of the conic. This is used to yield a direct conversion by placing the Bezier control points on the given guiding triangle. The second method matches the curvature at the knots to gain a corresponding conversion. It is shown that this works well for the elliptic and parabolic case, but not for the hyperbolic curve.

6.0 Capture by Parabolic Arcs

6.1 Introduction

Thus far, the discussion has concerned itself with either employing the Bezier cubic or the general conic form for mathematically modelling a given outline of shape. Although these are sufficient for representing contours efficiently, some design systems have incorporated the parabolic arc as part of their main spline routine. System 7.0 developed by Apple™, for example, uses this form to model character outlines for its True-Type font. This has lead to other systems, such as IKARUS, to include a mechanism for capturing outlines with the parabolic arc.

This chapter considers the parabolic spline in terms of its modelling capabilities. An introduction to the nature of the spline is given in section 6.2. Both the advantages and limitations it exhibits are highlighted. The chapter then describes two algorithms for capturing shape contours using the parabolic spline. Through this, a unique approach is developed which demonstrates that it is much easier to gain a parabolic solution via the general conic than directly. This finding is somewhat surprising as there are five parameters to solve for the general conic, compared to just four for the parabola.

6.2 Characteristics and Properties

This section presents some of the properties and features of the parabolic arc. As the parabola belongs to the conic family, some of its characteristics have already been presented, especially in section 4.2. The attributes discussed here, therefore, form an extension of this, and are presented in relation to the parabolic arc.

6.2.1 Manual Construction of a Parabolic Arc

In section 4.2.1, the point of being able to gain a conic arc through a manual construction is made. Much of its introduction to the design process is attributed to Liming [LIMI 44, LIMI 79]. To investigate and highlight some of the

properties of a parabolic curve, it is necessary to consider the way it is manually constructed:

Fig 6.1 illustrates one approach for gaining a parabolic arc. It is apparent from this, that the curve is defined through two knot points and their respective tangents. The guiding triangle formed by such an arrangement is, therefore, enough to result in the complete description of the curve. Points on this arc are gained by, initially, subdividing the lines resulting from the two tangents and their respective point of intersection. The subdivision process partitions the lines into a number of desired parts. By joining these lines, in the manner depicted in Fig 6.1, it is possible to generate a parabolic curve.

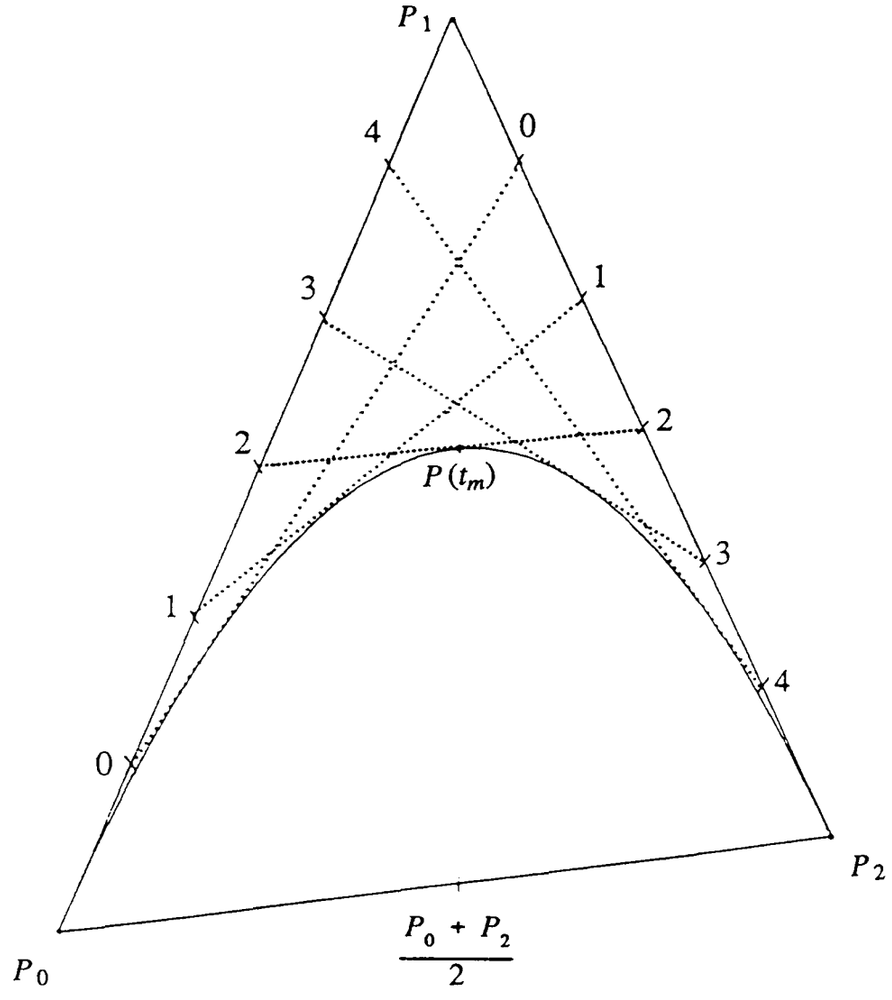


Fig 6.1 Gives an illustration of a manual construction for a parabolic arc.

Through this construction, it is possible to realise that the shoulder point for the parabolic arc lies half-way between the intersection (control) point P_1 and the point at $\frac{P_0 + P_2}{2}$. This means that, as expected, the resulting sharpness value equals one. The line 2-2 highlights the fact that the tangent at the shoulder (mid) point (t_m) is parallel to the line made by P_0 and P_2. This is an important property which simplifies the mathematics, and is applicable to all the conic sections described in terms of the guiding triangular framework as used by Pratt [PRAT 85].

6.2.2 Applications of Parabolic Form

Like most of the mathematical forms used for capturing shape outlines, the parabolic arc is employed in a variety of applications. These include modelling of dental arches, heels of shoes and contours of characters. The focal properties of the parabola are employed in transmission systems, where the "dish" is appropriately shaped to receive or transmit signals [VASI 80].

Since the parabola belongs to the conic family, it is not surprising to realise, therefore, that it is applied together with the other quadratic forms. The parabolic arc has the attraction, however, of requiring less information for a description than the general case. This is reflected in Fig 6.1, where a parabolic arc is defined through the two knots and the control point.

In the field of typography, piecewise parabolic arcs have been employed to model outlines of font characters. Their simplicity of description makes them an ideal candidate for such applications. Apple™ have capitalised on this by defining an entire font on the parabolic spline. This has lead to other design systems, such as IKARUS, developing suitable conversion routines.

Although the parabolic curve offers some benefits over the other quadratic curves (ie elliptic and hyperbolic), it does open itself to some practical limitations. This will be highlighted through the two algorithms developed for parabolic capture,

described in the latter sections of this chapter. The next section presents the mathematical forms used for describing parabolic curves.

6.2.3 Mathematical Forms

In similar fashion to the Bezier cubic and the general conic descriptions, the mathematical form for the parabolic case can either be in terms of a parametric variable or expressed solely in terms of a quadratic equation in x and y. In other words, both parametric and implicit forms exist for the parabola.

The three-point Bezier triangle always returns a parabolic arc [BEZI 72]. It can be shown, therefore, that for such an arrangement a parametric form for the curve point $P(t)$ can be gained through equation (3.2) by setting n equals to two. The corresponding coordinate expressions can readily be formulated (for a Bezier starting from its origin) as follows:

$$\begin{aligned} x(t) &= 2t(1-t)u + t^2C_x \,, \\ y(t) &= 2t(1-t)v + t^2C_y \,, \end{aligned} \qquad \ldots(6.1)$$

where: u and v denotes control position,
C_x and C_y denotes end knot position, and
t ranges between zero and one.

The respective implicit form for representing parabolic arcs can be gained using equation (4.5), where the sharpness value S is set appropriately to equal one. This then leads to the following formulation for the parabolic case:

$$d(x_i,y_i) = \frac{4\Delta_i\Delta_j - \Delta_k^2}{\Delta} \,. \qquad \ldots(6.2)$$

In passing, it is worth noting that the parametric form for the parabolic spline can also be gained through the rational representation for the general quadratic, as given by equation (4.3). This leads to equation (6.1) by setting S=1, highlighting the fact that the Bezier quadratic set-up yields a parabolic arc.

6.2.4 Point-to-Curve Deviation

With reference to the general conic case of section 4.2.5, a convenient way of evaluating the quality-of-fit of a parabolic arc to a given set of data points is through calculating the point-to-curve deviations. For this purpose, the implicit form, as given by equation (6.2), is used. By letting Λ_i denote this deviation for a given data point (x_i,y_i), then it can be shown that this takes the form:

$$\Lambda_i \approx \frac{d_i}{\sqrt{\Delta_x^2 + \Delta_y^2}} , \qquad ...(6.3)$$

where: d_i is as given by equation (6.2),

$$\Delta_x = \frac{\partial d_i}{\partial x} = 2v + \frac{(\Delta_k+2\Delta_i)(C_y-2v)}{\Delta} ,$$

$$\Delta_y = \frac{\partial d_i}{\partial y} = 2u + \frac{(\Delta_k+2\Delta_i)(C_x-2u)}{\Delta} ,$$

Δ, Δ_i and Δ_k are as given by equation (4.5).

For each parabolic approximation, equation (6.3) provides a means of estimating the goodness-of-fit. The deviation returned by each IK point is compared with a given, and desired, accuracy. If it happens that the worst-case deviation is above this, then two or more arcs are required for an acceptable fit.

6.3 Outline of Problem

Having presented some of the properties and features of the parabolic spline, this section gives a description of the type and form of the approximation that is required. Although the techniques developed can be employed for capturing any outline, the parabolic algorithms have evolved with view to the IKARUS system. The problem in this case can be summarised as follows:

Given a set of data points (Q_i) describing a character outline, whose directional tangents at each point are known, a parabolic description is required that yields the "best" possible approximation. The goodness-of-fit is measured in terms of

a predefined accuracy. The number of arcs employed and the rate of conversion are the other two factors which determine the efficiency of an algorithm.

The term "best" is characterised, in a similar manner to the general conic case (see section 4.3), to mean either an approximation which deviates the least from the given IK data points or one which maintains gradient continuity between joining arcs with acceptable accuracy. Three possible solutions to this problem are considered. The first, discussed in the next section, is shown to have characteristics that make it unattractive for commercial use. The other two approaches for a parabolic solution are given respectively in section 6.4.1 and 6.4.2.

6.4 Elements of Parabolic Conversion

When considering how to convert the given IK outline points to parabolic curve segments, a choice has to be made whether to use the parametric or the implicit form. Both forms have their advantages and limitations. Meier [MEIE 88], for example, chooses to employ the parametric form (as given by equation (6.1)). Although he does not elaborate why the parametric form was used, it is clear from the capturing process that his selection was influenced through his association with the Bezier cubic case. In other words, techniques developed (by him) for the cubic spline were translated to the quadratic case. This, obviously, resulted in an approach which embodied all the recursive, and time-consuming, stages linked with the Bezier cubic. In order to avoid unnecessary iterations in the capturing process, the implicit form of equation (6.2) is used here.

In view of the problem outlined in the previous section, a parabolic solution can be gained either directly through the specified tangents, or by allowing some flexibility in the location of the control points, or via another mathematical description which either maintains first-order continuity or is made less restrictive and returns control positions which give the least point-to-curve deviation.

The first possibility is easily realisable. The given tangents at the knot points can be used to construct a guiding triangle which in return will yield a parabolic arc. Although this approach guarantees first-order continuity between joining arcs, it does not take note of the IK curve points between the two (defining) knots. In other words, the process assumes a parabolic-fit without considering the given outline shape. This does not appear, therefore, to be the most suitable way of gaining the "best" possible parabolic-fit.

To "improve" the situation, two parabolic arcs can be used: The splitting point (ending knot for the first arc and starting knot for the second) is normally constrained to lie half-way between the two control points of the resulting arcs. The point of subdivision is either gained through "designating" one of the IK points as being the most suitable, or by applying a more complicated procedure where it is obtained through a recursive mechanism. This approach has the advantage that it maintains gradient continuity and its method of solution is based on the given IK points. It employs, however, at least two parabolic arcs (for a given outline) to sustain such continuity. Furthermore, and more importantly, its mechanism for capture is based entirely on a recursive set-up. This has a detrimental effect, therefore, on the rate of conversion.

The second and third possibilities for a parabolic solution are presented respectively in section 6.4.1 and 6.4.2. Their performance in capturing outlines is analysed and compared in section 6.4.3. Through this, it is shown that the fastest (and most convenient) way of gaining a parabolic-fit is via the general conic, a result which is somewhat surprising as the parabola requires fewer parameters for its description.

6.4.1 Capture With Least Deviation

Having discussed an approach that exhibits some form of continuity, this section presents an algorithm which yields a parabolic spline that deviates the least from the given set of IK points. The gradient continuity is compromised to enable the capturing process to take on the "best" value for the control point, and not just

the point where the knot tangents intersect. It is expected that this would lead to fewer curve segments being employed. The process in detail is as follows:

If we consider the capturing process of the general conic as manifested in section 4.4.2, it is clear that this leads to a linear solution for the three unknowns α, β and γ, resulting in the "best" possible values for the control point (u,v) and the sharpness parameter S. In the parabolic case, S equals one, implying that an additional constraint of $\alpha\beta-\gamma^2 = 1$ needs to be introduced in the capturing process. This leads to a non-linear solution, requiring iterative routines.

Employing equation (6.2), the implicit form for the parabolic spline, also leads to a formulation which is non-linear in both u and v. This highlights further the basic problem of fitting a parabola to a given set of data points. Although it requires four parameters (one less than the general conic), it is well known in the field of Mathematics that given four data points, two parabolic solutions can result [GRIE 34, LIMI 79]. The situation is depicted in Fig 6.2.

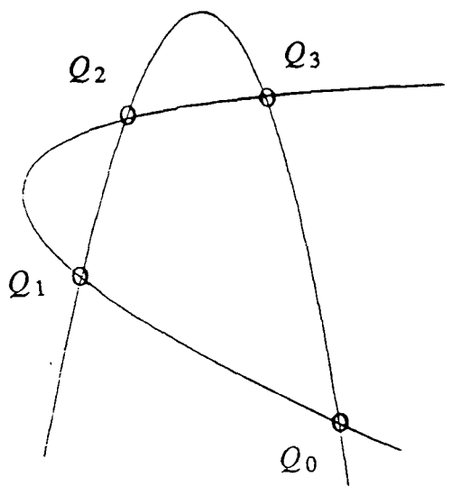


Fig 6.2 Shows that there are, in general, two parabolas which can be drawn through four given data points (Q_0, Q_1, Q_2 and Q_3).

The iterative solution, though not desirable, does form an approach for converting the given IK data points to a corresponding parabolic representation. Using equation (6.2) for this purpose, the "best" values for u and v can be gained through minimising the sum of the squared residue at each of the given data points. That is:

$$\frac{\partial}{\partial u}\sum_{i=1}^{n} d_i^2 = 0 \ ,$$

$$\frac{\partial}{\partial v}\sum_{i=1}^{n} d_i^2 = 0 \ . \qquad \ldots(6.4)$$

For a solution to the two unknowns, expressed in terms of equation (6.4), the NAG routine coded CO5NBF [POWE 70] is used. If this returns an unsatisfactory parabolic-fit, the given set of IK points are subdivided and approximated by two or more arcs.

The various stages of this approach are expressed graphically in Fig 6.3. Its performance is analysed in section 6.4.3. The next section presents a unique non-iterative algorithm.

6.4.2 Capture Through the General Conic

As an alternative to the algorithm presented in the previous section, an approach is described in this section which does not employ an iterative mechanism for a solution. Furthermore, this approach is more adaptable in that the resulting parabolic-fit can either be the one which deviates the least from the given set of data points or, be the most optimum to ensure gradient continuity between joining arcs. The method uses a two-pass capturing mechanism, where the first pass captures the given IK data points using general conic sections and the second pass translates these into parabolic arcs. The method in detail is as follows:

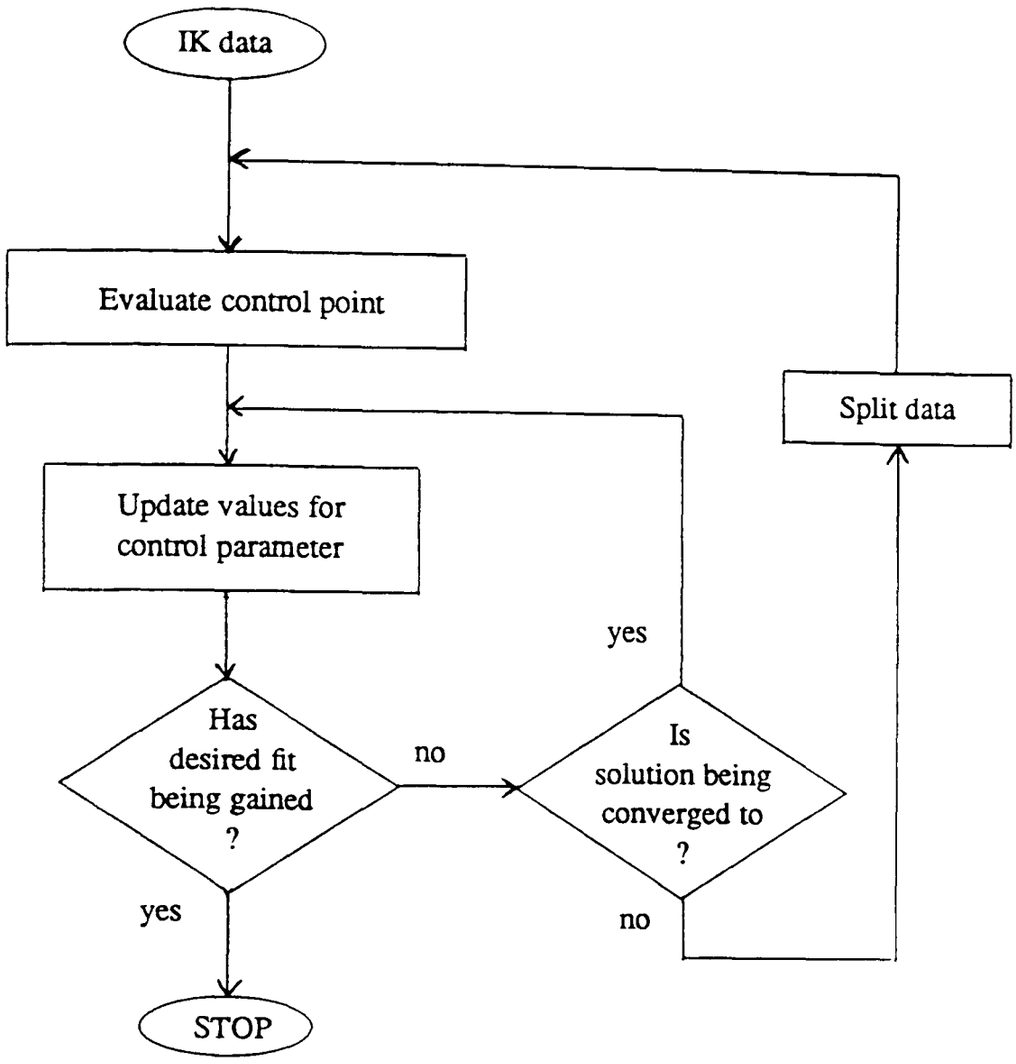


Fig 6.3 Depicts the technique for a parabolic solution which returns the closest approximation to the given outline.

As discussed and presented in chapter four, the process of mathematically modelling the outline of a shape using general quadratic curves can be accomplished without resorting to iterative and recursive techniques. Both the methods described respectively in sections 4.4.1 and 4.4.2 yield expressions which are linear in terms of their solution variables. Since the parabola forms a special curve within the conic family, an approach based on this relationship appears to be the best way to gain a parabolic solution.

The primary stages of such an approach are shown in Fig 6.4. After receiving a set of IK curve points, the algorithm yields an output using a two-pass conversion system. The first stage quantifies the data points in terms of a "best-

fitting" general conic section. This could be in the manner described in section 4.4.1, where gradient continuity is guaranteed, or be in the form of section 4.4.2, where a conic approximation is gained which deviates the least from the given set of data points. As far as the parabolic description is concerned, either approach could be employed, though its clear that for first-order continuous arcs, the conic quantification must maintain the given tangent information.

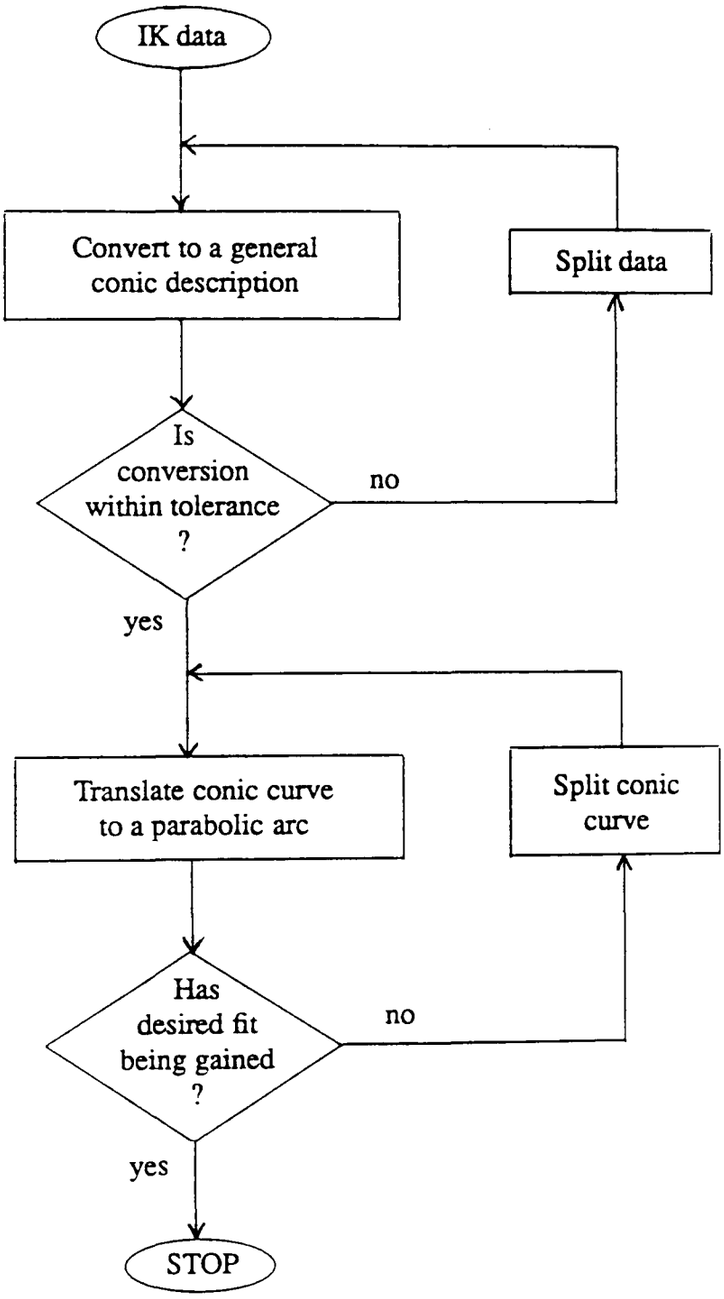


Fig 6.4 Shows the parabolic conversion process which uses the general conic representation.

The second pass, as depicted in Fig 6.4, takes the general conic section and translates it to a parabola. This is acquired by using the guiding triangle (as provided by pass one) and setting the sharpness value to equal one. In other words, a parabolic-fit is gained through the triangular framework for a general conic description. In the case of an elliptic or a hyperbolic spline, some deviation between these and the resulting parabolic arc will result. An expression for measuring this deviation can be derived using respectively equations (4.3) and (6.1) for t=0.5. With reference to the graphical illustration of Fig 6.5, where Λ_m denotes the corresponding deviation, and taking S as the "best-fitting" sharpness value for the general conic, the following formulation can be devolved for Λ_m:

$$\Lambda_m = \frac{(1-S)}{(1+S)}\sqrt{(0.5u-0.25C_x)^2 + (0.5v-0.25C_y)^2} \quad . \qquad ...(6.5)$$

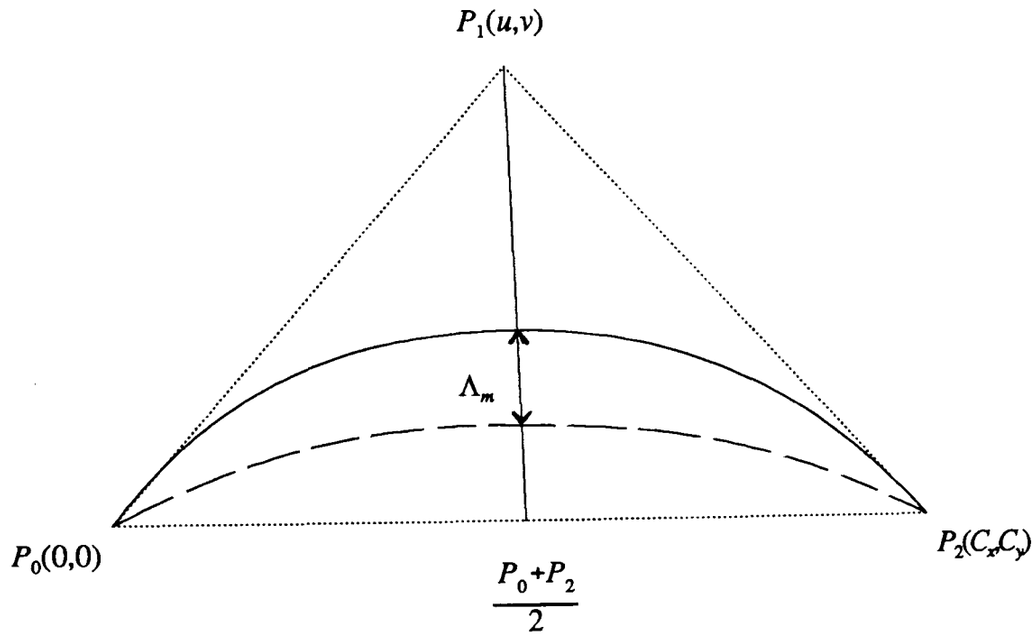
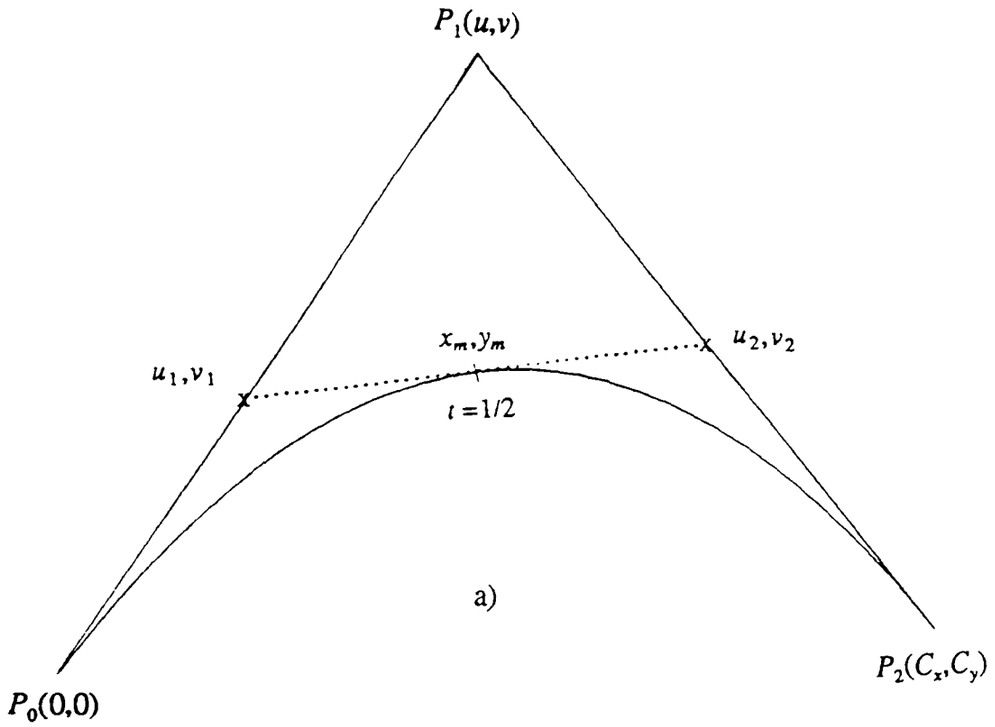


Fig 6.5 Gives a graphical representation for the mid-point deviation Λ_m for a general conic curve (shown with long dashes) and a parabolic arc.

Clearly, equation (6.5) is providing a means of evaluating how close the parabolic-fit is to the "best-fitting" conic description. An alternative way of assessing this could be based on the parameters α, β and γ for the general conic. The equality $\alpha\beta = \gamma^2$ manifests that the general quadratic section is a parabola; otherwise if $\alpha\beta>\gamma^2$ or $\alpha\beta<\gamma^2$ it is respectively an ellipse or a hyperbola. This fact could be used to replace the term $\frac{(1-S)}{(1+S)}$ with an appropriate proportional term in α, β and γ.

Equation (6.5) provides a useful way of determining whether, or not, the resulting parabolic curve will return a satisfactory approximation that meets a desired specification. If Λ_m is found to be above this, then the given set of IK data points would need two (or more) parabolic arcs for a solution. Otherwise, equation (6.3) will be employed to evaluate the worst-case deviation. If this is found to be larger than desired, the conic curve would need to be subdivided and represented by two (or more) parabolic splines.

As depicted by Fig 6.4, the second phase of the conversion algorithm is not activated until an acceptable conic-fit has been obtained. The given IK data is subsequently partitioned so that two or more general quadratic arcs describe it accurately. If it is then found that the resulting parabolic-fit is not satisfactory, the conic curve is subdivided in the manner shown in Fig 6.6. For the case where a single new knot point is sufficient (Fig 6.6a), the conic spline is split at its mid-point (ie at t=0.5). The new knot positions are calculated by applying the parametric form for the general conic as given by equation 4.3. The corresponding tangent at this new knot is parallel to the line from the (conic) start and end knots. Furthermore, the location for the resulting two control points of the parabolic splines is such that the new knot lies half-way between them (Liming gives an illustration of this general conic property [LIMI 79]), thus fulfilling the desire to minimise the amount of storage required (see previous section).



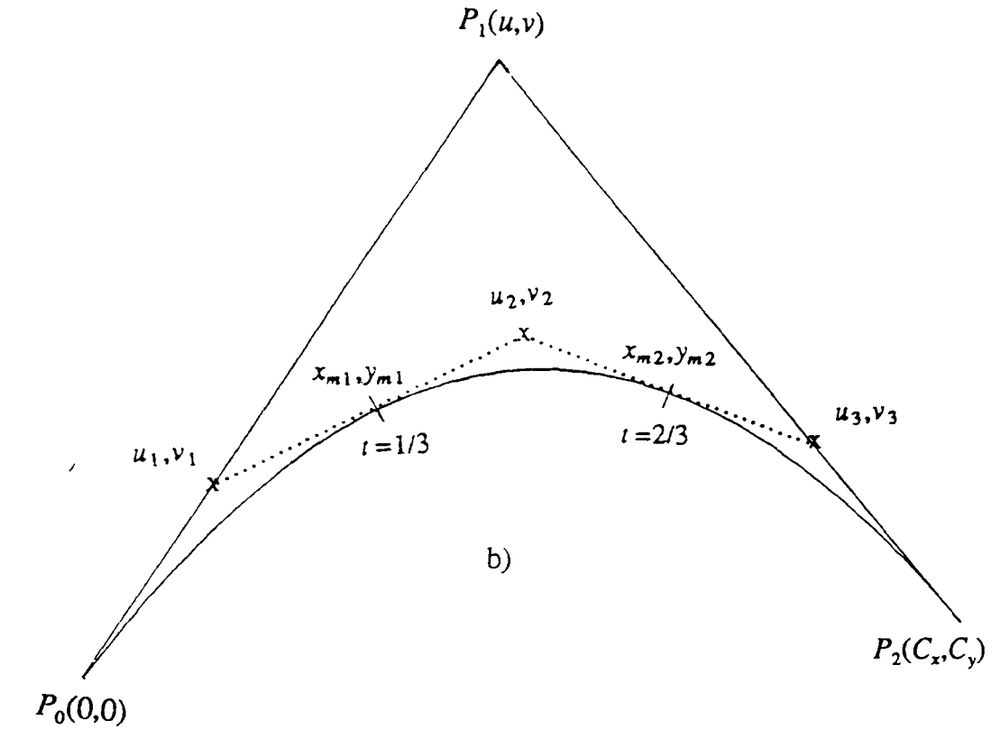


Fig 6.6 Demonstrates the subdivision of the general conic curve, to yield: a) two parabolic arcs, and b) three parabolic curves.

Although general expressions in terms of tangents can be derived, the formulation for the two control points (for the parabolic splines) can be expressed solely as a function of the defining guiding triangle for the conic:

$$
\begin{aligned}
u_1 &= \frac{uS}{S+1}\ , \\
v_1 &= \frac{vS}{S+1}\ , \\
u_2 &= \frac{uS+C_x}{S+1}\ , \\
v_2 &= \frac{vS+C_y}{S+1}\ ,
\end{aligned}
\qquad \ldots(6.6)
$$

where: S is the sharpness value for the "best-fitting" general conic,
u and v form the control point for the general conic,
u_1 and v_1 constitute the control point for the 1st parabolic arc, and
u_2 and v_2 constitute the control point for the 2nd parabolic arc.

If a single subdivision still does not yield an acceptable parabolic approximation, the conic curve can be further subdivided in the manner shown in Fig 6.6b and approximated by using three parabolic arcs. Two new knot points are placed, on the "best-fitting" conic section, at t = 1/3 and t = 2/3 respectively. The mathematics for solving for the three parabolic control points is based on similar principles used for the single split case. In order to simplify the computation, it is worth noting that the tangent at t = 1/3 is parallel to the line from the (conic) start knot to the second new knot point at t = 2/3. Similarly, the tangent at this point is parallel to the line from the first new knot point to the (conic) end knot.

In passing, it is worth mentioning the fact that the process of subdivision is greatly eased by the initial conic capture. Unlike the non-linear approach (of section 6.4.1) where each time the given IK defined outline is normally partitioned, this alternative approach makes use of the geometric properties of the general conic. Furthermore, the precise location of the point for subdivision can be selected and, as indicated above, an appropriate number of parabolic arcs employed.

Having described this linear approach for a parabolic solution, the next section assesses its performance through a specified example. The results are compared with that of the non-linear method (discussed in the previous section).

6.4.3 Analysis and Observations

To evaluate the performance of the two approaches adopted for describing outlines with parabolic arcs, the contour of the 'S' character shown in Fig 5.7 is used. Each curve segment is subdivided, in the manner discussed in section 5.3.5. The input to both algorithms is in the form of a "best-fitting" general conic, gained through the gradient continuous method of section 5.3.3.

Fig 6.7 depicts the capturing capabilities of using the non-linear approach (of section 6.4.1). For examination purposes, conversion is performed at various levels of fitness. Looking at the various outputs of Fig 6.7, it can be observed that most of the distinct features of the given 'S' outline have, to a good degree, being captured. Fig 6.8 demonstrates this fact by highlighting the differences in the integer positions between the supplied character shape and the corresponding parabolic fit (that is, between Fig 5.7 and Fig 6.7 respectively).

Applying the non-iterative procedure (of section 6.4.2) to the given outline results in the outputs shown in Fig 6.9. Again, these are gained through similar levels of fitness as for the non-linear approach. The respective differences in the integer positions between the given outline and the corresponding converted versions is depicted in Fig 6.10. The worst-case, shown in Fig 6.9a and corresponding difference in Fig 6.10a, gives an illustration of the margin between the (given) general conic sections and its (unique) parabolic arc. Like Fig 6.7a, all the curve segments (of Fig 6.9a) have been approximated using respective single parabolic arcs, but in this case by setting S equals to one rather than by an iterative least-squares fit. It is apparent from Fig 6.10a, that the two curves which return the greatest difference are A2 and D2 (both ellipses). Subdividing these curves (in the manner illustrated by Fig 6.6) results in the output shown in Fig 6.9b and their respective difference in Fig 6.10b. It is quite evident from this that the

subdivision procedure works effectively. Figs 6.9c, 6.9d and 6.9e highlight this point further, where curves A1-D1, B1-E1 and B2-C2-F2 have been respectively subdivided. The corresponding difference outlines are shown in Figs 6.10c, 6.10d and 6.10d.

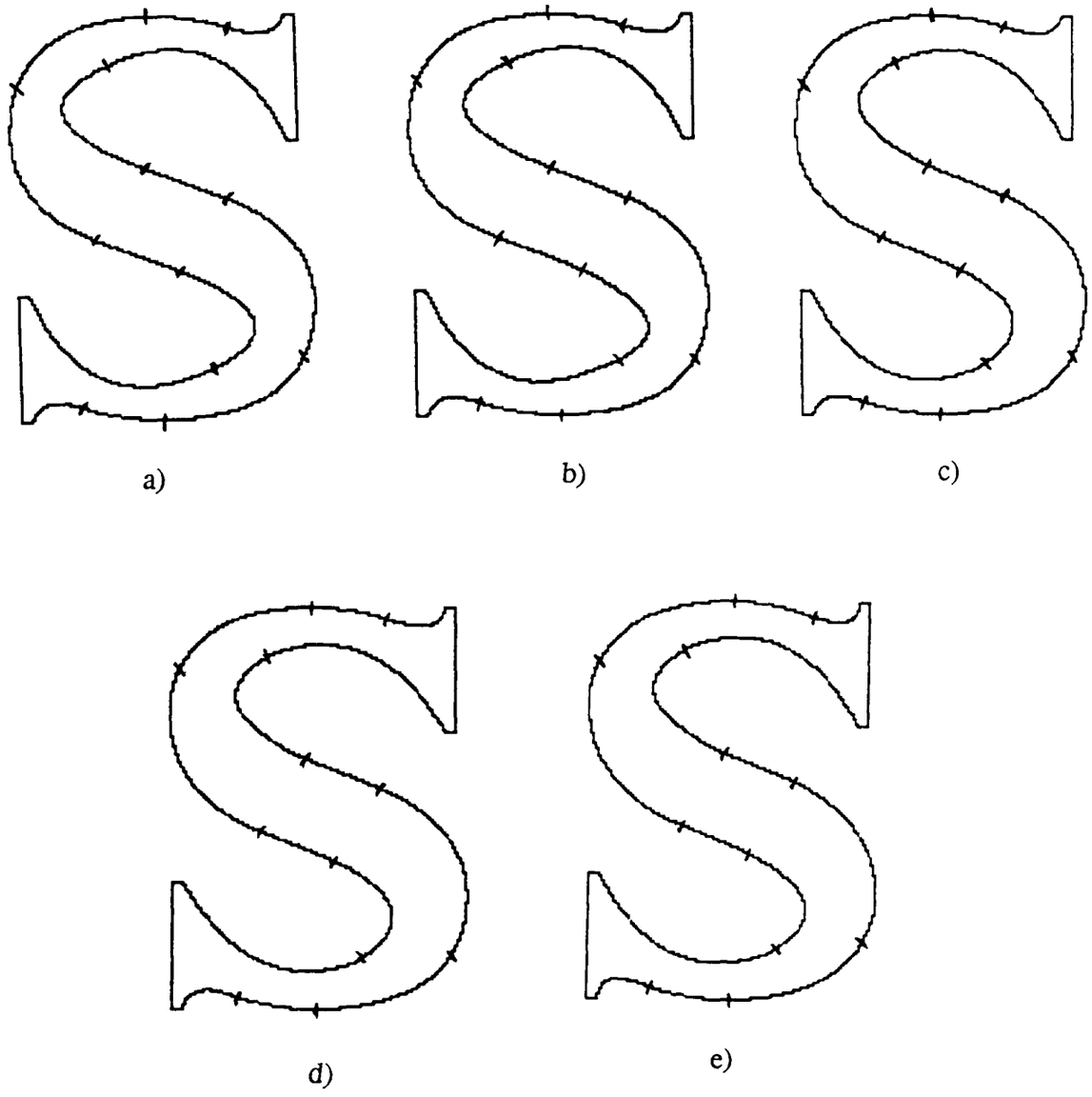


Fig 6.7 Shows the parabolic representation gained through the non-linear approach for various levels of accuracy (units):
a) 10, b) 5, c) 2, d) 1, and e) 0.5.

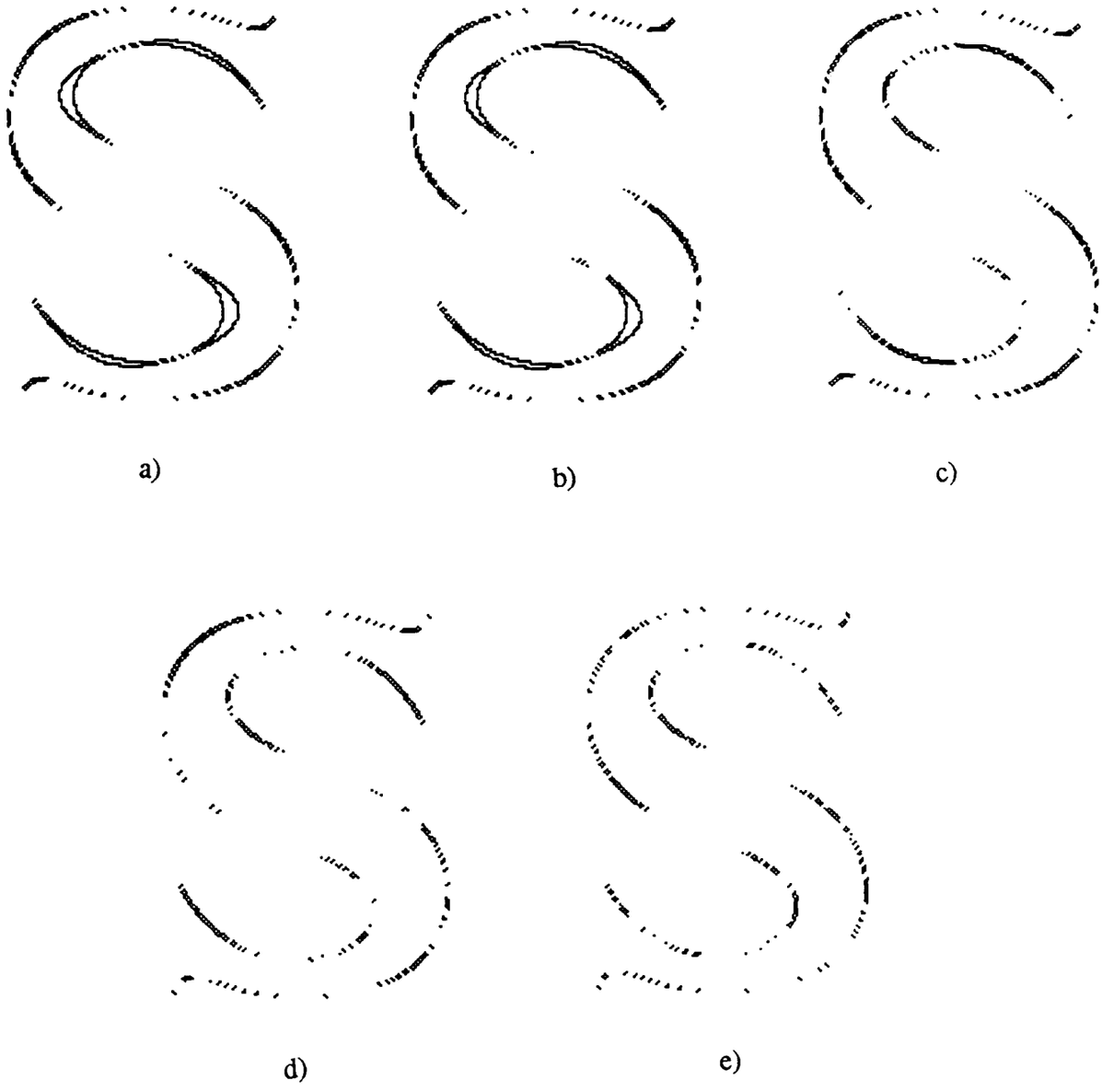
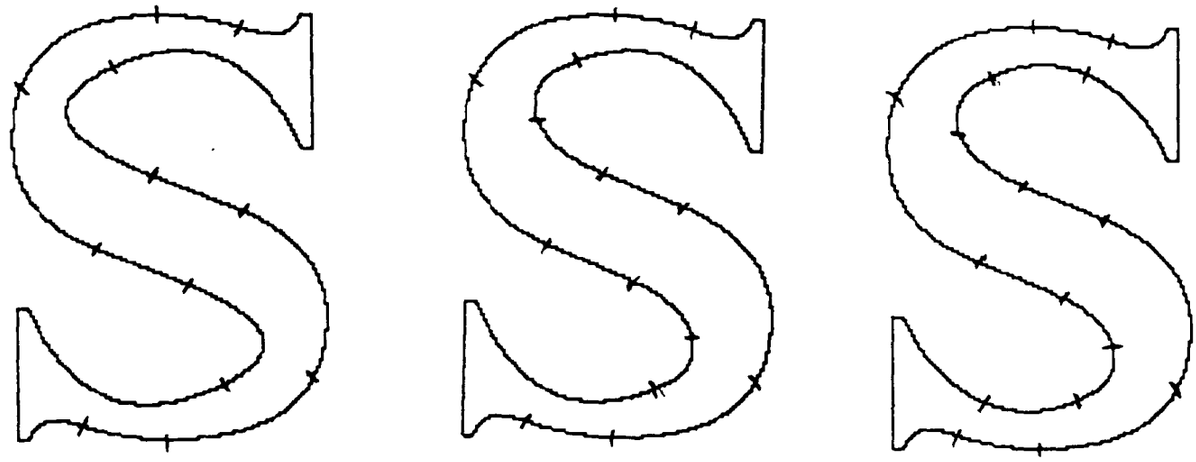


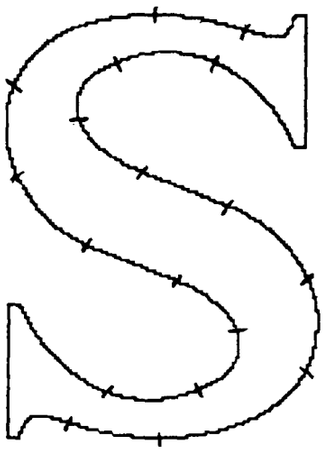
Fig 6.8 Highlights the corresponding differences in integer positions between the given outline and the resulting parabolic outputs of Fig 6.7, for various levels of tolerances (units): a) 10, b) 5, c) 2, d) 1, and e) 0.5.



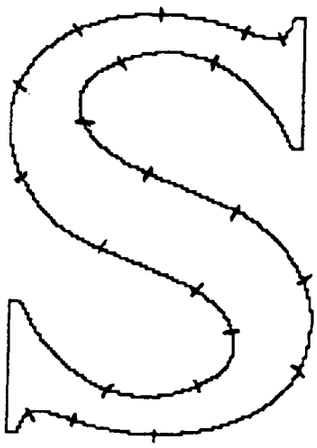
a)

b)

c)

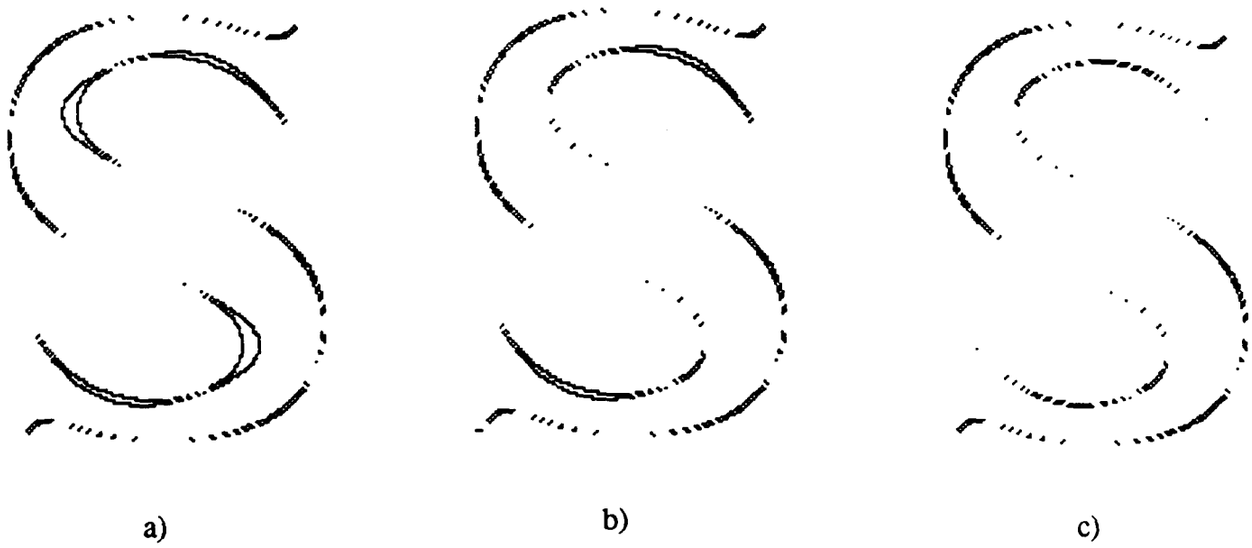


d)



e)

Fig 6.9 Depicts the outputs achieved through the non-iterative parabolic algorithm, for various levels of acceptability (units):
a) 10, b) 5, c) 2, d) 1, and e) 0.5.



a) b) c)

d) e)

Fig 6.10 Gives the difference in integer positions between the supplied outline and the outputs of Fig 6.9, for various levels of deviation (units): a) 10, b) 5, c) 2, d) 1, and 0.5.

To gain a better understanding of the effectiveness of the two conversion algorithms, their respective observations have been tabulated: Fig 6.11 shows the recordings for the non-linear approach, whilst the corresponding results from the non-iterative approach are listed in Fig 6.12.

IK to Parabolic, Non-Linear Method	**Point-to-Parabolic Deviation**				
	0.5	1.0	2.0	5.0	10.0
Total Number of Arcs	12	12	12	12	12
Total Number of Iterations	223	140	74	44	12
Rate of Conversion *CPU secs*	50.0	34.2	18.3	10.9	3.4
Average Deviation per Arc	0.1690	0.3359	0.4650	0.7760	0.8290

Fig 6.11 Shows tabulated results for the parabolic conversion using the non-linear approach of section 6.4.1.

IK to Parabolic, Non-Iterative Approach	**Point-to-Parabolic Deviation**				
	0.5	1.0	2.0	5.0	10.0
Total Number of Arcs	21	18	16	14	12
Rate of Conversion *CPU secs*	2.8	2.2	2.0	1.5	1.5
Average Deviation per Arc	0.0789	0.2745	0.4387	0.6783	0.9044

Fig 6.12 Shows tabulated observations for the parabolic conversion using the non-iterative approach of section 6.4.2.

Comparing the table of results of Fig 6.12 with that of Fig 6.11, it is clearly apparent that the non-iterative approach exhibits a decisive advantage when it comes to choosing the algorithm for reasons of conversion speeds. It takes, on average, about one fifth of the time to yield a parabolic solution. This factor decreases as the accuracy of the approximation is enhanced. In other words, the non-iterative yields a parabolic description at a faster rate, when compared with the non-linear method, as the point-to-curve deviation is decreased (highlighted by the respected CPU times for 0.5 and 10.0 in Fig 6.11 and 6.12 respectively).

As the two tabulated results highlight, speed of conversion for the non-iterative algorithm is gained at the expense of introducing more arcs for its description. Unlike the non-linear case, the non-iterative approach applied more than one parabolic arc (where necessary) to return an approximation within the given tolerance. This, as Fig 6.12 shows, results in the algorithm employing 21 arcs (about two parabolas for each given curve segment) for the case where the point-to-curve deviation falls below 0.5 units. The non-linear method returns a solution for this threshold by iterating until the desired outcome is reached. This, as Fig 6.11 shows, has the effect of increasing the time taken for a conversion.

As far as quality-of-fit is concerned, both methods give outputs which are acceptable. The recorded observations of the two approaches indicate that the most accurate conversion is performed by the non-iterative algorithm. This, however, is only true when more than one parabolic arc is being used to model each supplied curve segment. Indeed, when both methods employ a single parabola for an input curve (when the point-to-parabolic deviations is set at 10 units), the non-linear yields a better fit.

Although it was not necessary to employ three parabolic arcs for a supplied curve segment (as illustrated by Fig 6.6b), investigations on other character outlines highlighted a deficiency in this approach. The main problem is that once the three (parabolic) control points have been evaluated, the corresponding new knot points do not, as desired, lie half-way between respective control points. It turns out that for a parabolic solution, it is better to subdivide the given set of data

points and approximate them using two general conic sections before employing the non-iterative conversion algorithm.

In concluding, therefore, it can be said that both the approaches discussed respectively in sections 6.4.1 and 6.4.2 provide a means of capturing the given set of data points. The fact that it takes longer to gain a solution directly adds weight to the non-iterative approach. This yields an acceptable output at a much faster rate than the non-linear method. Furthermore, the process of gaining a solution through the general conic leads to an efficient subdivision process. The price for employing the non-iterative approach, however, is in the number of arcs used for its description.

6.5 Summary

This chapter looks at the possibilities of employing the parabolic spline as means of gaining a mathematical description. Its properties and features for representation are expressed, mainly, through its process of manual construction. The attraction that it requires one less parameter than a general conic description is highlighted.

Two algorithms for modelling outlines using parabolic arcs are given. The first attempts a "direct" approach, where the general quadratic equation is expressed in terms of the unknowns. This results in a non-linear expression, which requires recursive means for a solution.

An alternative approach is based on a two-pass conversion system. The first-pass converts the given IK data to a general conic format. This is taken as an input for the second phase, where it is translated into a corresponding "best-fitting" parabolic spline(s). This approach is shown to have benefits of simplicity and speed over the iterative "direct" approach.

7.0 Conclusions and Further Work

A number of algorithms are presented in this dissertation that assist the designer to process outlines of shape. Although consideration is generally limited to aid typographers (employing the IKARUS software package), most of the concepts and methods developed are applicable to stylists and proficient artisans working in other fields. The fact that many design systems incorporate a spline routine, which models a desired outline, enhances the possibility of including many of the described algorithms in such systems.

The algorithms, which are developed, can be categorised in terms of three, distinct, types: Firstly, there are those which capture a given set of data points. These take a supplied outline of shape and yield an appropriate mathematical description, resulting in either a Bezier cubic, or a general conic, or a parabolic representation. The second type of algorithms are those which enable conversion between the three mathematical descriptions. Translating a Bezier cubic described outline to a general quadratic form, for example, are undertaken by these algorithms. The third classification caters for the methods and techniques developed to facilitate the rasterisation of the mathematically described contours. These take the modelled outline and output a digitised version, where this conforms to the grid positions of the resulting display.

With reference to Fig 1.1, where the various types of algorithms are portrayed graphically, Fig 7.1 gives the number of algorithms developed in each case. It is clear that six methods facilitate a mathematical description. Each description is achieved through two respective approaches (the second approach for the parabolic case is via the general conic). Five methods are shown which allow conversion between the three mathematical descriptions. Four of these are used to convert between the Bezier cubic and the general quadratic. As far as algorithms for rasterisation are concerned, Fig 7.1 shows that four techniques have been presented for this purpose. The parabolic case, although depicted as a separate process, is catered for by the general conic. In short, about twelve

algorithms have been developed in total which enable the designer to mathematically capture a given outline (whether IK described or otherwise), allow conversion between these descriptions, and facilitate a digitised output.

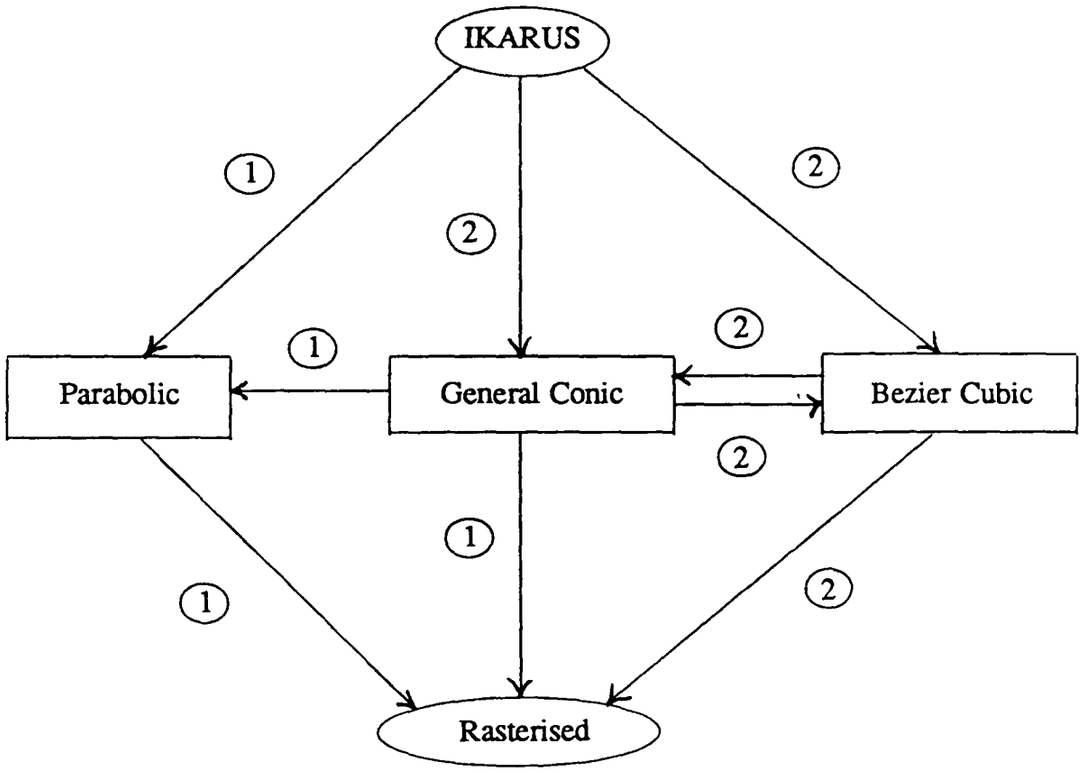


Fig 7.1 Depicts the number of algorithms developed for the cases shown.

In chapter three, the procedures and methods associated with modelling and rasterising outlines using the Bezier cubic are presented. It is apparent that this mathematical description has the attraction of representing a point of inflection within a curve segment and encompasses the necessary properties to allow a curvature continuous (piecewise connected) outline description. Although these features are similar to those exhibited by the physical spline, the Bezier cubic is shown to forfeit some of its appeal when it comes to using it for purposes of both capture and rasterisation: In section 3.4, a method for modelling outlines

using the parametric form (of the Bezier cubic) is given. This is shown to have an inherent tendency to employ recursive means for providing a solution, resulting in rates of conversion which are expensive in terms of time. For the example used in section 3.4.4, the best capturing rate is 65.4 CPU seconds. In other words, the least amount of time required to convert the given 'S' outline is over one minute. The fact that most fonts consist of more than a hundred characters gives an appreciation of how slow the process of gaining a Bezier cubic description can be. Clearly, employing such an approach for the commercial sector, where in the field of typography (for example) thousands of fonts regularly require such a conversion, is not viable as far as costs are concerned.

Although methods of improving the speed of conversion (as well as, enhancing the quality-of-fit) are assessed in section 3.4.4, it is apparent that the parametric approach will always lead to an iterative, and time consuming, solution. In order to remove some of the overheads of this approach, the implicit form for the Bezier cubic is employed. This has the attraction of being non-iterative (though still recursive as the unknown expressions are non-linear) and, more importantly, does not use the parametric variable. The implicit form has the advantage, therefore, of returning a residue value directly for each given data point, without having to relate it to the parametric variable (as in the case of the parametric form).

The algebraic form developed for the Bezier cubic, however, has an undesired and unwanted attribute. This leads to a number of solutions which, as addressed in section 3.5.3, interpolate the given data in a rather unacceptable way. The reason for this behaviour is due to the implicit form exhibiting an instability for the case of a parabolic arc. Possible ways of overcoming these problems are discussed in the above cited section. The best solution in this case is to limit the capturing performance of the algorithm to representing conic-like shapes. In this case, the methods for converting a general conic section to a corresponding Bezier cubic form, as discussed in section 5.4, can be used to gain an initial estimate for the Bezier control points. Clearly, the cubic spline employed in such

a manner would not be allowed to capture an outline with a cusp or a loop, or one which contains a point of inflection. Restricting the Bezier cubic in this way, therefore, does not appear to be the ideal solution. A better approach would be to limit, or remove, the effects of the instability by normalising it, an approach which certainly needs to be explored.

When it comes to rasterising the Bezier described outlines, the traditional approach has been to convert the curve segments in to a number of line segments. A method based on this technique, which uses a novel way of determining how many line segments to employ, is presented in section 3.6.1. This has the advantage of computational simplicity. The main drawback of this technique is its zero-order continuity between joining line segments. This leads to a rasterised outline that appears to lose its overall smoothness. An alternative, and aesthetically acceptable, method is developed in section 3.6.2. This uses the implicit form to describe the Bezier cubic spline. It has a clear advantage over the previous approach in that it tracks the given curve segment. In other words, it follows the curve by choosing the nearest integer grid point at each stage; maintaining, therefore, the general appearance of the described contour.

Although the tracking approach is better, it has one serious handicap: It tends to get confused and loses its way if it encounters a self-intersecting point (for the Bezier cubic) within the parametric interval of zero and one. The consequence of this are manifested in section 3.6.3. As discussed in this section, there are a number of ways to overcome this problem. Each solution requires that all given Bezier cubic splines be quantified in terms of whether or not an intersection point exists within the normal parametric interval. If it does, then it is necessary to split the Bezier point at its point of self-intersection. From the discussions made in section 3.6.3, it is apparent that a given Bezier cubic curve can be mapped on to an r and s graph. This will then give an indication of the form for the resulting curve. By subdividing the graph into regions, it is possible to distinguish the curves which require splitting and those that do not. The conditions for these regions need still to be developed.

In a quest for a simpler and more efficient mathematical description, techniques are developed which employ the general quadratic form. These, as discussed in chapters four and five, cannot represent within a single curve segment an outline containing points of inflection, a cusp or a loop. Furthermore, the conic spline is restricted to capturing outlines whose knot tangents can intersect in such a manner as to form a guiding triangle. Although it appears that the capturing capabilities of the general conic are rather limited when compared to the Bezier cubic, in fact the opposite tends to be the case in practical situations. This is especially the case in the field of typography, where a given outline is normally modelled through extrema points acting as knot points [KARO 87]. As far as outlines containing cusps, loops and points of inflection are concerned, these can easily be represented through employing two or more conic arcs.

One of the main attractions of employing conic splines is their speed of modelling a given set of data points. Compared with 65.4 CPU seconds for the Bezier cubic, a general conic description can be gained for the given 'S' character at about 9 CPU seconds. This is achieved through the non-iterative and non-recursive techniques developed for the general quadratic case. The two conic algorithms, in addition, require relatively small amount of computation to yield an approximation, enhancing further the rate of conversion.

For the given set of IK data points, both the conic capturing approaches return a satisfactory goodness-of-fit. Constraining the modelling process to maintain first-order continuity between joining conic arcs results in more curve segments being employed than otherwise. Indeed, as shown in section 4.4.3, up to seven more conic arcs are required (for the supplied 'S' icon) to retain the given gradient continuity. In other words, the conic approach of section 4.4.2 (which relaxes on the tangent constraints and chooses values for the control and sharpness parameters such that the resulting conic arc deviates the least from the given data) employs significantly less curve segments for its description. The cost for this achievement is that each two joining arcs exhibit a discontinuity of, on average, about three degrees. This does not appear to be too great, and opens up the possibility of devising a capturing scheme for the conic which approximates

the given data within a desired tolerance and which maintains, also, "gradient" continuity within an acceptable discontinuity. This way a compromised conic approximation will result that uses a minimum number of curve segments and which exhibits some form of first-order continuity.

Although the two conic algorithms appear to be ideally suited for modelling outlines of font characters, their performance on other types of data, such as the situation where the given points are scattered, still needs to be assessed. It is expected that the algorithms will return a suitable conic-fit as long as the given data can be described within a guiding triangle. In addition, the effects of normalising the conic expression to gain a particular type of "best-fitting" approximation needs further exploration.

When it comes to rasterising a contour described by conic sections, the algorithm presented in section 4.5 appears to suffice for all the cases considered. This has the attraction that it follows (that is, tracks) the given conic arc, returning the nearest mesh-point at each stage. Unlike the cubic case, the conic curve does not exhibit "states of confusion" except for extreme curves such as ellipses. Even for these rare cases, the algorithm developed in section 4.5 is made robust enough to perform satisfactory for all forms and shapes of conic arcs which lie within a quadrant.

The process of converting between Bezier cubic and the general quadratic splines is considered in chapter 5. With similarity to gaining a general conic description, two algorithms are presented for translating a Bezier curve into a corresponding conic arc. The difference in this case is that the discrete summation of data points is replaced by an integral representing the complete Bezier curve within its parametric interval of zero and one. Both algorithms use non-iterative and non-recursive methods for providing a conversion.

As demonstrated in section 5.3.5, the closest approximation is returned by the conversion algorithm which does not maintain gradient continuity. Again this is gained through relatively little discontinuity between joining arcs. With analogy

to the general conic case, the expressions for the residue could be normalised to yield a specific type of solution. The process of normalisation, however, will not necessarily lead to a more efficient algorithm as far its usage of computation (and rate of capture) is concerned.

Two algorithms are developed for the purposes of converting a general conic section into a respective Bezier cubic spline. Apart from the parabolic case (which is an exact solution), both the elliptic and hyperbolic curves can only be approximated by the cubic representation. It is clear from section 5.4.3, that the Bezier description finds it easier to model elliptic arcs than hyperbolic sections. Indeed, the algorithm which yields a conversion through matching curvatures fails to return a good approximation for the hyperbolic case. This clearly is due to fact that most of the curvature is near its mid-point; implying that an approach based on the sharpness value might be better. The second algorithm uses this fact to return the two control points for the Bezier cubic which best approximate the given conic at its mid-point. This appears to give better results, especially for the hyperbolic curve, when compared to the first algorithm.

Both the algorithms are useful for gaining initial values for r and s, the control parameters for the Bezier cubic. These can then be employed together with the non-parametric form (of section 3.5) to gain a corresponding best-fit. It would be interesting to evaluate how well the r and s values returned by the conversion algorithm (which is based on the sharpness value) compare with those resulting from the non-parametric Bezier algorithm constrained so $r = s$.

In chapter six, two algorithms are given for modelling a given outline via parabolic arcs. The first attempts to gain a direct solution through using the general conic equation, by setting the sharpness value to equal one. This leads to an algorithm, however, which becomes non-linear in the two unknown variables, requiring recursive means to gain values for the control parameters. To remove some of these time costly procedures, an alternative algorithm is developed which gives a parabolic description through using the capturing capabilities of the general conic. It is found that this returns a rate of conversion

which is, on average (depending on the accuracy of approximation), about ten times faster. Furthermore, this approach leads to a simpler process of subdivision.

The employment of parabolic arcs for modelling outlines of font characters (or of any shape) appears to have little basis when compared to the general conic: As discussed earlier, the general conic leads to capturing techniques which are linear in terms of the unknown variables, resulting in fast rates of capture. Attempting to gain a parabolic description, however, leads to either an approach which is recursive, or one that is based on another mathematical form (such as the general conic, see section 6.4). The simplicity of description for the parabolic when compared to the general conic, as highlighted in section 6.2.1, in requiring one less parameter, does not result in a similar reduction in the amount of computation necessary for capturing purposes. Furthermore, the general conic form appears more appropriate for cases where two curve segments are necessary to model accurately a given outline. It can apply any of the curves belonging to the quadratic family, and not just the parabolic arc. The effect of this, in general, would be a marked reduction in the number of curve segments required for a description.

In short, this thesis has presented a number of algorithms which are aimed at assisting the modern designer in mathematically modelling outlines of a given shape. The designer is at ease to choose either a Bezier cubic or a general conic or, even, a parabolic description. Each representation has its advantages, as well as some drawbacks. It is important, therefore, that these are considered before an attempt is made to incorporate a particular type of mathematical description in a design system.

8.0 References

[AKIM 70] H Akima,
A New Method of Interpolation and
Smooth Curve Fitting Based on Local Parameters,
ACM, vol 17, no 4, 1970, pp 589-602.

[ALBA 74] A Albano,
Representation of Digitised Contours
in Terms of Conic Arc and Straight
Line Segments,
Computer Graphics and Image Processing,
vol 3, 1974, pp 23-33.

[ALIA 87] G Alia, F Barsi, E Martinelli, N Tani,
Angular Spline: A New Approach to
the Interpolation Problem in
Computer graphics,
Computer Vision, Graphics and Image
Processing, vol 39, 1987, pp 56-72.

[ASKW 47] E H Askwith,
The Analytical Geometry of
the Conic Sections,
Cambridge University Press,
England, 1947.

[BAIL 36] J H S Bailey,
Elementary Analytical Conics,
Oxford University Press,
England, 1936.

[BALL 74] A A Ball,
CONSURF Part 1: Introduction of the
Conic Lofting Tile,
Computer Aided Design,
vol 6, no 4, 1974, pp 243-249.

[BARS 83] B A Barsky,
A study of the Parametric Uniform
B-spline Curve and Surface Representation,
Tech. Report No: UCB/CSD 83/118,
Electrical and Computer Sciences Dept;
University of California, Berkeley,
California, USA, 1983.

[BARS 84] B A Barsky,
Exponential and Polynomial Methods for
Applying Tension to an Interpolating
Spline Curve,
Computer Vision, Graphics, and Image
Processing, vol 21, no 1, 1984, pp 1-18.

[BART 87] R H Bartels, J C Beatty, B A Barsky,
An Introduction to Splines for use in
Computer Graphics and Geometric Modelling,
Morgan Kaufmann, USA, 1987.

[BEGO 79] E A BeGole,
Application of the Cubic Spline Function
in the Description of Dental Arch Form,
Journal of Dental Research,
vol 59, no 9, 1979, pp 1549-1556.

[BEZI 71] P Bezier,
Example of an Existing System in the Motor Industry: The UNISURF System,
Proceedings of Royal Society of London,
vol A321, 1971, pp 207-218.

[BEZI 72] P Bezier,
Numerical Control
Mathematics and Applications,
John Wileys & Sons, England, 1972.

[BEZI 74] P Bezier,
Mathematical and Practical
Possibilities of UNISURF,
Computer Aided Geometric Design,
Academic Press, 1974.

[BEZI 86] P Bezier,
The Mathematical Basis of
the UNISURF CAD system,
Butterworths, England, 1986.

[BEZI 90] P Bezier,
Style, Mathematics and NC,
Computer Aided Design,
vol 22, no 9, 1990, pp 524-526.

[BIGE 85] C Bigelow,
Font Design for Personal Workstations,
BYTE, January 1985, pp 255-270.

[BIGG 72] R H Biggerstaff,
Three Variations in Dental Arch Form
Estimated by a Quadratic Equation,
Journal of Dental Research,
vol 51, 1972, pp 1509.

[BOEH 82] W Boehm,
On Cubics: A Survey,
Computer Graphics and Image Processing,
vol 19, 1982, pp 201-226.

[BOEH 84] W Boehm, G Farin, J Kahmann,
A Survey of Curve and Surface
Methods in CAGD,
Computer Aided Geometric Design,
vol 1, 1984, pp 1-60.

[BOLT 75] K M Bolton,
Biarc Curves,
Private Communication.

[BOOK 79] F L Bookstein,
Fitting Conic Sections to Scattered Data,
Computer Graphics and Image Processing,
vol 9, 1979, pp 56-71.

[BOOR 78] C de Boor,
A Practical Guide to Splines,
Applied Mathematics Sciences vol 27,
Springer-Verlag, USA, 1978.

[BOTT 68] R J Botting, M L V Pitteway,
Cubic Rendering Algorithm,
Computer Journal, vol 11, 1968, pp 120.

[BRES 65] J E Bresenham,
Algorithm for Computer Control of
a Digital Plotter,
IBM Systems Journal,
vol 20, no 4, 1965, pp 25-30.

[CHAS 78] S H Chasen,
Geometric Principles and Procedures
for Computer Graphic Applications,
Prentice-Hall, USA, 1978.

[CHUA 90] Y S Chua,
Bezier Brushstrokes,
Computer Aided Design,
vol 22, no 9, 1990, pp 550-555.

[CLIN 74] A K Cline,
Scaler and Planer valued Curve Fitting
using Splines under Tension,
ACM Communications,
vol 17, 1974, pp 218-220.

[COHE 80] E Cohen, T Lyche, R Riesenfeld,
Discrete B-Splines and Subdivision
Techniques in Computer-Aided-Geometric-
Design and Computer Graphics,
Computer Graphics and Image Processing,
vol 14, 1980, pp 87-111.

[COHE 89] E Cohen, C L O'Dell,
A Data Dependent Parametrization
for Spline Approximation,
Mathematical Methods in Computer Aided
Geometric Design, ed: T Lyche, L Schumaker,
Academic Press, 1989, pp 155-166.

[COOL 45] J L Coolidge,
A History of the Conic Sections and
Quadric Surfaces,
Oxford University Press, England, 1945.

[COON 67] S A Coons,
Surfaces for Computer Aided Design
of Space Forms,
Report MAC-TR-41, Project MAC, MIT, 1963.

[COX 72] M G Cox,
The Numerical Evaluation of B-splines,
Journal of Institute of Maths and Applications,
vol 10, no 2, 1972, pp 134-149.

[ENGE 86] H Engels,
A Least Squares Method for Estimation of
Bezier Curves and Surfaces and its
Applicability to Multivariate Analysis,
Mathematical Biosciences,
vol 79, 1986, pp 155-170.

[FARI 89] G Farin, N Sapidis,
Curvature and the Fairness of Curves
and Surfaces,
IEEE Computer Graphics and Applications,
vol 9, no 2, 1989, pp 52-57.

[FARI 90] G Farin,
Curves and Surfaces for Computer Aided
Geometric Design,
Academic Press, USA, 1990.

[FAUX 79] I D Faux, M J Pratt,
Computational Geometry for Design
and Manufacture,
Ellis Horwood, England, 1979.

[FERG 63] J C Ferguson,
Multi-variable Curve Interpolation,
Report no: D2-22504, The Boeing Company,
Seattle, Washington, USA, 1963.

[FERG 64] J C Ferguson,
Multi-variable Curve Interpolation,
Journal ACM, vol 11, no 2, 1964, pp 221-228.

[FORR 68] A R Forrest,
Curves and Surfaces for
Computer Aided Design,
PhD Thesis, University of Cambridge,
Trinity College, Cambridge, England, 1968.

[FORR 72] A R Forrest,
Interactive Interpolation and Approximation
by Bezier Polynomials,
Computer Journal,
vol 15, no 1, 1972, pp 71-79.

[FORR 80] A R Forrest,
The Twisted Cubic Curve: A Computer-Aided
Geometric Design Approach,
Computer Aided Design,
vol 12, no 4, 1980, pp 165-172.

[FORR 90] A R Forrest,
Interactive Interpolation and Approximation
by Bezier Polynomials,
Computer Aided Design,
vol 22, no 9, 1990, pp 527-537.

[GLEN 84] B T Glenn,
Alphabet and Text in
Presentation and Graphics,
Computer Graphics World, 1984, pp 11-22.

[GORD 74] W J Gordon, R F Riesenfeld,
B-spline Curves and Surfaces,
Computer Aided Geometric Design,
Academic Press, USA, 1974, pp 95-126.

[GORO 86] K K Gorowara,
Representation of Two Bezier Cubic Curves,
Proceedings of National Aerospace
and Electronics Conference,
Dayton, Ohio, USA, vol 3, 1986, pp 733-734.

[GRIE 34] A B Grieve,
Analytical Geometry of Conic Sections
and Elementary Solid Figures,
Bell & Sons Ltd, England, 1934.

[GROS 90] J Grosvenor, K Morrison, A Pim,
The Postscript Font Handbook 1990,
Ivanoe Press, England, 1990.

[HARA 82] K Harada, E Nakamae,
Application of Bezier Curve
to Data Interpolation,
Computer Aided Design,
vol 14, no 1, 1982, pp 55-59.

[HOCH 90] H J Hochfeld, M Ahlers,
Role of Bezier Curves and Surfaces in the
Volkswagen CAD approach from 1967 to today,
Computer Aided Design,
vol 22, no 9, 1990, pp 598-607.

[HU 91] J Hu, T Pavlidis,
Function Plotting using Conic Splines,
To be published.

[HUSS 89] F Hussain,
Conic Rescue of Bezier Founts,
New Advances in Computer Graphics,
Ed: R A Earnshaw, B Wyvill,
Springer-Verlag, 1989, pp 97-120.

[HUSS 91] F Hussain, M L V Pitteway,
Rendering a Cubic with Square Moves Only,
Computer Journal, vol 34, 1991, pp 405.

[KARO 87] P Karow,
Digital Formats for Typefaces,
URW Verlag, Hamburg, 1987.

[KARO 91] P Karow,
Intelligent FontScaling,
To be published.

[KIND 79] D Kindersley, N Wiseman,
Computer Aided Letter Design,
Printing World, October 31, pp 12-17.

[KNUT 79] D E Knuth,
Tex and Metafont:
New Directions in Typesetting,
Digital Press/American Math Soc, 1979.

[KNUT 82] D E Knuth,
The Concept of a Metafont,
Visible Language,
vol 16, no 1, 1982, pp 3-27.

[KNUT 85] D E Knuth,
Lessons Learned from Metafont,
Visible Language,
vol 19, no 1, 1985, pp 35-54.

[LEE 85] E T Y Lee,
Some Remarks Concerning B-splines,
Computer Aided Geometric Design,
vol 2, no 4, 1985, pp 307-311.

[LEWI 86] D Lewis,
Gaussian Elimination Subroutine,
NOAA Research Laboratory,
Baulder, Colarado, USA.

[LIEB 67] J B Lieberman,
Types of Typefaces,
Sterling Publishers, USA, 1967.

[LIMI 44] R A Liming,
Practical Analytical Geometry
with Applications to Aircraft,
MacMillan Co, USA, 1944.

[LIMI 79] R A Liming,
Mathematics for Computer Graphics,
Aero Publishers, USA, 1979.

[MEIE 88] H Meier,
IKARUS to Parabolic Form,
URW Technical Report,
Hamburg, Germany.

[MEIE 89] H Meier,
IKARUS to Bezier Form,
URW Technical Report,
Hamburg, Germany.

[MORT 85] M E Mortenson,
Geometric Modelling,
John Wiley and sons, USA, 1985.

[MORT 89] M E Mortenson,
Computer Graphics: An Introduction
to the Mathematics and Geometry,
Heinemann Newnes, USA, 1989.

[NEWM 79] W M Newman, R Sproull,
Principles of Interactive Computer Graphics,
McGraw-Hill, 1979.

[NIEL 74] G M Nielson,
Some Piecewise Polynomial Alternatives
to Splines in Tension,
Computer Aided Geometric Design,
Academic Press, 1974.

[PATO 70] K Paton,
Conic Sections in Chromosome Analysis,
Pattern Recognition, vol 2, 1970, pp 39-51.

[PATT 88] R R Patterson,
Parametric Cubics as Algebraic Curves,
Computer Aided Geometric Design,
vol 5, 1988, pp 139-159.

[PAVL 82] T Pavlidis,
Algorithms for Graphics and
Image Processing,
Computer Science Press, 1982.

[PAVL 83] T Pavlidis,
Curve Fitting with Conic Splines,
ACM Transactions on Graphics,
vol 2, no 1, 1983, pp 1-31.

[PIEG 86] L Piegl,
A Geometric Investigation of the Rational
Bezier Scheme of Computer Aided Design,
Computers in Industry,
vol 7, 1986, pp 401-410.

[PIEG 87] L Piegl,
Interactive Data Interpolation by Rational
Bezier Curves,
IEEE Computer Graphics & Applications,
1987, pp 45-58.

[PILC 73] D T Pilcher,
Smooth Approximation of Parametric
Curves and Surfaces,
PhD Thesis, University of Utah,
Salt Lake City, Utah, USA, 1973.

[PITT 67] M L V Pitteway,
Algorithm for Drawing Ellipses or
Hyperbolae with a Digital Plotter,
Computer Journal, vol 4, 1967, pp 25-30.

[PITT 85] M L V Pitteway,
Algorithms of Conic Generation,
Fundamental Algorithms for
Computer Graphics,
Ed: R A Earnshaw, NATO ASI Series
vol F17, 1985, pp 219-237.

[PITT 87] M L V Pitteway, E K Banissi,
Hardware Aspects of Algorithm,
Fundamental Algorithms for
Computer Graphics,
NATO ASI series, vol 19, 1987.

[PITT 91] M L V Pitteway, F Hussain,
On the Rendering of Conic Sections
and Bezier Cubic Arcs,
"Graphics, Interaction and Visualization -
The Challenge for the 1990's",
British Computer Society's Seminar,
December 1991, London, England.

[PLAS 83] M Plass, M Stone,
Curve-Fitting with Piecewise
Parametric Cubics,
Computer Graphics,
vol 17, no 3, 1983, pp 229-239.

[PRAT 85] V Pratt,
Techniques for Conic Splines,
Computer Graphics (Siggraph85),
vol 19, no 3, 1985, pp 151-159.

[PRAT 87] V Pratt,
Direct Least-Squares Fitting of
Algebraic Surfaces, Computer Graphics,
vol 21, no 4, 1987, pp 145-151.

[PROT 75] M H Protter, C B Morrey,
Analytic Geometry, 2nd edition,
Addison-Wesley Publishers, USA, 1975.

[POWE 70] M J D Powell,
A Hybrid Method for
Non-Linear Algebraic Equations,
Numerical Methods for Non-Linear Algebraic
Equations, ed: P Rabibowitz,
Gordon and Breach, 1970.

[PRAT 85] V Pratt,
Techniques for Conic Splines,
Computer Graphics (ACM Siggraph),
vol 17, pp 229-239.

[PRIN 79] A Pringle, P Robinson, N Wiseman,
Aspects of Quality in the Design
and Production of Text,
Computer Graphics,
vol 13, no 2, 1979, pp 63-70.

[RIES 74] R F Riesenfeld,
Applications of B-spline Approximation to
Geometric Problems of Computer-Aided-Design,
PhD Thesis, Syracuse University,
Syracuse, USA, 1973.

[ROGE 76] D J Rogers, J A Adams,
Mathematical Elements for Computer Graphics,
McGraw-Hill, USA, 1976.

[RUTK 79] W S Rutkowski,
Shape Completion,
Computer Graphics and Image Processing,
vol 9, 1979, pp 89-101.

[SAMP 81] P D Sampson,
Dental Arch Shape: A Statistical
Analysis Using Conic Sections,
American Journal of Orthodontics,
vol 79, no 5, 1981, pp 535-548.

[SAMP 82] P D Sampson,
Fitting Conic Sections to "Very Scattered"
Data: An Iterative Refinement of
Bookstein Algorithm,
Computer Graphics and Image Processing,
vol 18, 1982, pp 97-108.

[SAMP 83] P D Sampson,
Statistical Analysis of Arch Shape
with Conic Sections,
Biometrics, vol 39, 1983, pp 411-413.

[SCHU 81] L L Schumaker,
Spline Functions: Basic Theory,
John Wiley & Sons, USA, 1981.

[SCHW 66] D G Schweikert,
An Interpolation Curve using
Spline in Tension,
Journal of Mathematics & Physics,
vol 45, 1966, pp 312-317.

[SEDE 83] T W Sederberg,
Implicit and Parametric Curves and Surfaces
for Computer Aided Geometric Design,
PhD Thesis, Purdue University, USA, 1983.

[SEDE 84] T W Sederberg, D C Anderson, R N Goldman,
Implicit Representation of Parametric
Curves and Surfaces,
Computer Vision, Graphics, and
Image Processing, vol 28, 1984, pp 72-84.

[SEDE 85] T W Sederberg, D C Anderson, R N Goldman,
Implicitization, Inversion, and Intersection
of Planar Cubic Curves,
Computer Vision, Graphics, and
Image Processing, vol 31, 1985, pp 89-102.

[SEDE 87] T W Sederberg,
Algebraic Geometry for Surface
and Solid Modelling,
Geometric Modelling: Algorithms and New Trends,
Ed: G Farin, SIAM publication, 1987.

[SEDE 90] T W Sederberg, T Nishita,
Curve Intersection using Bezier Clipping,
Computer Aided Design,
vol 22, no 9, 1990, pp 538-549.

[SMIT 26] C Smith,
Geometrical Conics,
MacMillan and Co, England, 1926.

[SPAT 74] H Spath,
Spline Algorithms for Curves and Surfaces,
Translated by: W D Hoskins, H W Sager,
Utilitas Mathematica Publishers,
Winnipeg, Canada, 1974.

[STON 89] M C Stone, T D DeRose,
A Geometric Characterization of
Parametric Cubic Curves,
ACM Transactions on Graphics,
vol 8, no 3, 1989, pp 147-163.

[THOM 89] S M Thomas, Y T Chan,
A Simple Approach for the Estimation of
Circular Arc Center and its Radius,
Computer Vision, Graphics, and Image
Processing, vol 45, 1989, pp 362-370.

[TODD 47] J A Todd,
Projective and Analytical Geometry,
Pitman, England, 1947.

[TUCK 18] C O Tuckey, W A Nayler,
Analytical Geometry,
Cambridge University Press,
England, 1918.

[VASI 80] N B Vasilyev, V L Gutenmacher,
Straight Lines and Curves,
Tr: A Kundu,
Mir Publishers, USSR, 1980.

[VILL 87] L Villalobos,
The Conic Curve,
Computer Graphics World,
vol 10, no 5, 1987, pp 91-94.

[WATK 88] M A Watkins, A J Worsey,
Degree Reduction of Bezier Curves,
Computer Aided Design,
vol 20, no 7, 1988, pp 398-405.

[YAMA 88] F Yamaguchi,
Curves and Surfaces in Computer Aided
Geometric Design,
Springer Verlag, Germany, 1988.

[YEFI 64] N V Yefimov,
Quadratic Forms and Matrices,
Tr: A Shenitzer,
Academic Press, England, 1964.