COMPUTER AIDED DESIGN OF DEVELOPABLE SURFACES

By

Brian E. Konesky

B.A.Sc. (Mechanical Engineering) University of British Columbia

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

December 1993

Mechanical Engineering
The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date: Dec. 23, 1993

Abstract

The design of objects employing developable surfaces is of engineering importance because of the relative ease with which developable surfaces can be manufactured. The problem of designing developable surfaces is not new. Two space curves, defining the edges of the surface, are first created, then a set of rulings are constructed between the space curves under the constraint of developability. A problem with existing algorithms for designing developable surfaces is the tendency to introduce non developable portions of the surface; areas of regression.

A more reliable solution to the problem of creating a developable surface is proposed. The key to the method is to define the developable surface in terms of a normal directrix. The shape of the normal directrix defines the shape of the developable surface. Algorithms are defined to compute the shape of a normal directrix from a pair of space curves. A non-linear optimization technique was implemented to further refine the shape of the developable surface, but failed to yield satisfactory results. Other algorithms were also created to intersect adjacent developable surfaces and to generate the flat plate layouts. The algorithms were implemented using the C++ programming language and the AutoCAD CAD package. Recommendations for further work are given.

Table of Contents

List of Tables

List of Figures

Nomenclature

$\delta_{i,j}$	Dirac Delta Function.
θ	Out of Plane Rotation of G' With Respect to G.
$\frac{d\theta}{da}$	Change in Angle of Generator With Respect to Unit Motion of a Parameter Describing Directrix.
$\frac{d^2\theta}{dadt}$	Rate of Change of Angle of Generator With Respect to Motion of Parameter Describing Directrix.
ϕ	In Plane Rotation of G' With Respect to G.
a	Parameter Describing Distance Along Generator.
$\frac{da}{dt}$	Incremental Position Along Generator.
C_i	A Control Vertex that is Being Solved For.
$f(u)$	Position Along Left Adjacent Space Curve.
$\overrightarrow{\frac{df}{du}}$	Tangent Vector Along Left Adjacent Space Curve.
$\overrightarrow{\frac{d^2f}{du^2}}$	Curvature Vector Along Left Adjacent Space Curve.
$g(v)$	Position Along Right Adjacent Space Curve.
$\overrightarrow{\frac{dg}{dv}}$	Tangent Vector Along Right Adjacent Space Curve.

$\overrightarrow{\dfrac{d^2g}{dv^2}}$	Curvature Vector Along Right Adjacent Space Curve.
$\overrightarrow{gen}$	The Developable Surface Generator Vector.
$\overrightarrow{\dfrac{dgen}{da}}$	Change of Position of Generator With Respect to Motion of Parameter Describing Directrix.
$\overrightarrow{\dfrac{d^2gen}{dadt}}$	Rate of Change of Generator With Respect to Motion of Parameter Describing Directrix.
$\overrightarrow{\dfrac{dgen}{dt}}$	The Generator Slope Vector.
$\overrightarrow{n_1}$	Normal Vector For Left Adjacent Space Curve.
$\overrightarrow{n_2}$	Normal Vector For Right Adjacent Space Curve.
$n(t)$	Normal Vector From Surface
n_t	Number of Control Vertices For Normal Directrix.
n_u	Number of Control Vertices For Left Adjacent Space Curve.
n_v	Number of Control Vertices For Right Adjacent Space Curve.
$\overrightarrow{N}$	Unit Normal Vector.
$p(t)$	Point On the Directrix.
$\overrightarrow{\dfrac{dp}{dt}}$	The Directrix Tangent Vector.
$\overrightarrow{\dfrac{d^2p}{dt^2}}$	The Directrix Curvature Vector.
s	Scalar Value Along Generator

$q(t, s)$	Point on Surface Determined by Parametric Values t and Scalar s
t	Independent Parametric Variable Describing Position Along Directrix.
$\overrightarrow{T}$	Unit Directrix Tangent Vector.
$\overrightarrow{\frac{dT}{da}}$	Change of Position of Unit Tangent With Respect To Motion of Parameter Describing Directrix.
u	Dependent Parameter Describing Position Along Left Adjacent Space Curve.
$\frac{du}{dt}$	Incremental Position Along Left Adjacent Space Curve.
v	Dependent Parameter Describing Position Along Right Adjacent Space Curve.
$\frac{dv}{dt}$	Incremental Position Along Right Adjacent Space Curve.

A. Einstein

Keep it simple:

as simple as possible,

but no simpler

ENGINEERING

The scientist analyzes what is.

The engineer creates what has never been.

The engineer scientist analyzes what is

Imagines what should be

Creates what has never been

Analyzes the results of the creation.

By Gunnar Schlenius

Acknowledgement

I would like to express my sincere thanks to Dr. A. B. Dunwoody for his enthusiastic support, supervision and direction on this project.

I would also like to acknowledge the financial support from the NSERC of Canada whose financial support made this project possible.

Chapter 1

Introduction

1.1 Areas of Application

Developable surfaces form a special class of surfaces which are very useful in many practical situations. Developable surfaces have many applications. A few applications are cited below.

Naval Architecture Up until recently in history ship hulls were made of various types of wood, which could be easily worked into any desired shape. "The hull of continuous, homogeneous, testable sheet material is inherently stronger and lighter than the structure of small pieces of wood. If a skin of sheet material can be designed for low labour cost in construction, simple tools, and economy in repair, its engineering superiority and eventual economic advantage make it at once preferable to planks" [15]. To lower the labour cost in construction and repair even further, the skin of sheet material must be developable. At the same time, the hydrodynamic performance of the hull must be competitive with that of the best possible in compound hull construction [15]. Making the construction surface developable was therefore a desired criterion if possible.

Figure 1.1 below shows a developable surface single chine hull of a fishing vessel.

Figure 1.2 demonstrates how some manufacturers today create developable surface hulls for fishing vessels.

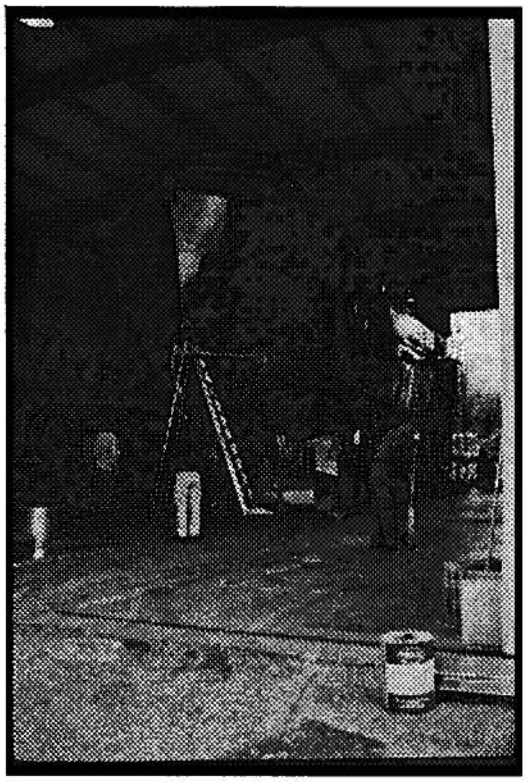


Figure 1.1: Developable Surface Single Chine Hull

Aerospace Today's modern aircraft are more complex in design and cost savings in manufacturing is crucial. Aircraft fuselages, wings and other smaller components can be produced by developable surfaces if considered in the design stage. One of the most recent "high-tech" declassified aircraft is the "wobblin-gobblin", ie. the "F117A", (see figure 1.3) which has a low radar profile signature by using flat plates, is yet another example of using developable surfaces in an ingenious manner.

Manufacturing In the area of manufacturing two new areas involving the developability criterion are in the application of peripheral milling and rolling. Each pass of an end mill cutting peripherally follows a developable surface.

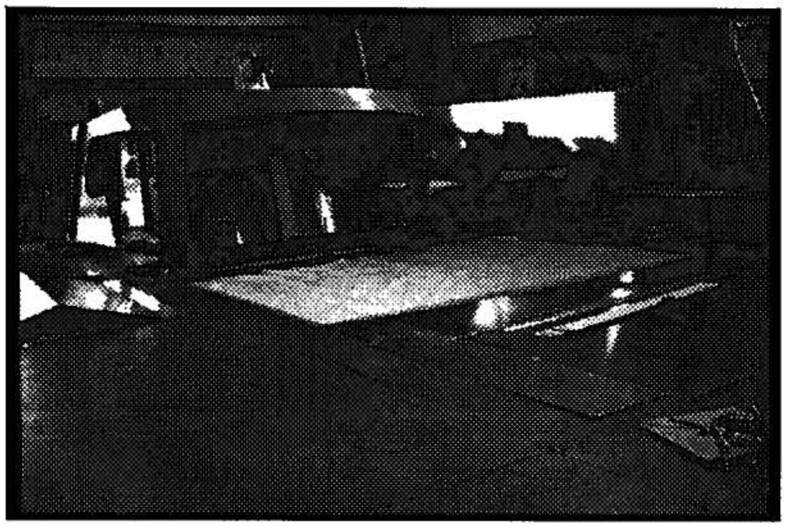


Figure 1.2: Manual method of generating developable surfaces

Another area is in the application of rolling. Developability must also hold for this process.

Both of these applications will not be discussed but only cited as other examples where developable surface criterion must hold due to the geometry of the application.

Textiles Another example of developability is in the textile industry. Clothing is manufactured from flat material and folded to conform to the appropriate geometry. Sails for sailing vessels is yet another application where developability must hold.

1.2 Definitions and Terminology

The developable surface is a subset of the class of ruled surfaces. The definitions of a ruled surface and a developable surface are as follows:

Ruled Surface: A ruled surface is defined as "The locus of a line, called a generator,

Figure 1.3: F117A stealth fighter

whose direction is determined by successive values of a parameter, moving continuously along a curve (a directrix) and intersecting that directrix at an angle other than zero."[15].

<u>Developable Surface:</u>

A developable surface,also defined by Kilgore and from Kreysig, is 'A ruled surface having the same tangential plane on one and the same generator"[15].

Figure 1.4 below illustrates the terms introduced similar to the definitions presented by G.D. Aguilar[2]. The directrix must lie in the surface and that each plane tangential to the surface must also be tangential to the directrix.

It must also be noted that with two space curves a ruled or compound surface may exist but no developable surface may be possible. Another theorem should also be noted that, "If two space curves lie in any developable surface, they lie in one and only one such surface"[15]. If the generators do not intersect anywhere, then the surface is developable.

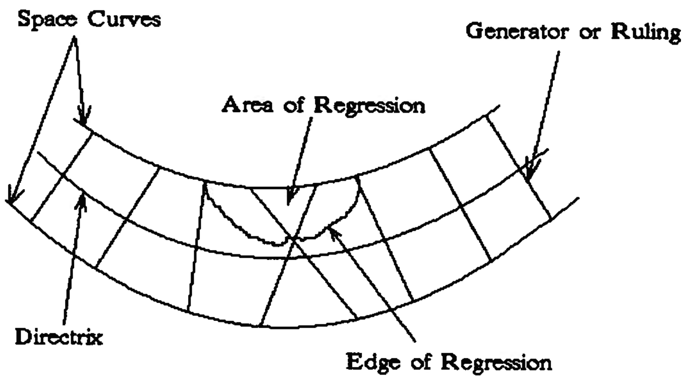


Figure 1.4: Definitions of Terminology of a Developable Surface

In figure 1.4 the areas where the generators overlap are known as areas of regression. The boundary of an area of regression outlined in figure 1.4 is called the edge of regression [7].

1.3 Methodologies

All existing methodologies for the design of developable surfaces start with the definition of two edges of the surface. Then, a set of generators is fit between those edges to define a developable surface.

1.3.1 Kilgore's Solution

The general method of matching developable surfaces to desired curves is an arduous task. The method assumes that the developable surface is either conical, cylindrical, or a combination of both. So, one tries a succession of surfaces until one is found to fit approximately. If the designer cannot find an exact solution his usual solution is to alter the original curve to fit the surfaces haphazardly [15]. See figure 1.5.

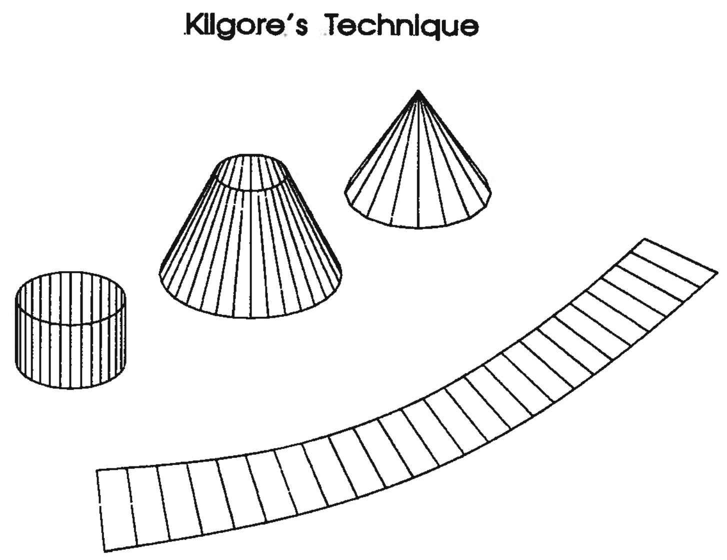


Figure 1.5: Kilgore's Method of Creating a Developable Surface

Kilgore examined this unique art and proposed a method for direct generation of developable surfaces from given beginnings. This method provides a manual graphical solution of surfaces to fit the curves, rather than to require alteration of the curves to fit the surfaces.

Kilgore's manual graphical solution is described in his paper[15] as well as a comprehensive description of the procedure is given in the Principles of Naval Architecture[11].

A sample manual graphical procedure is displayed below which was extracted from the Principles of Naval Architecture[11]. See Figure 1.6.

One can see that manual graphical solutions are only as accurate as the skills and precision of the naval architect. This poses problems in error of the final solution as to its accuracy.

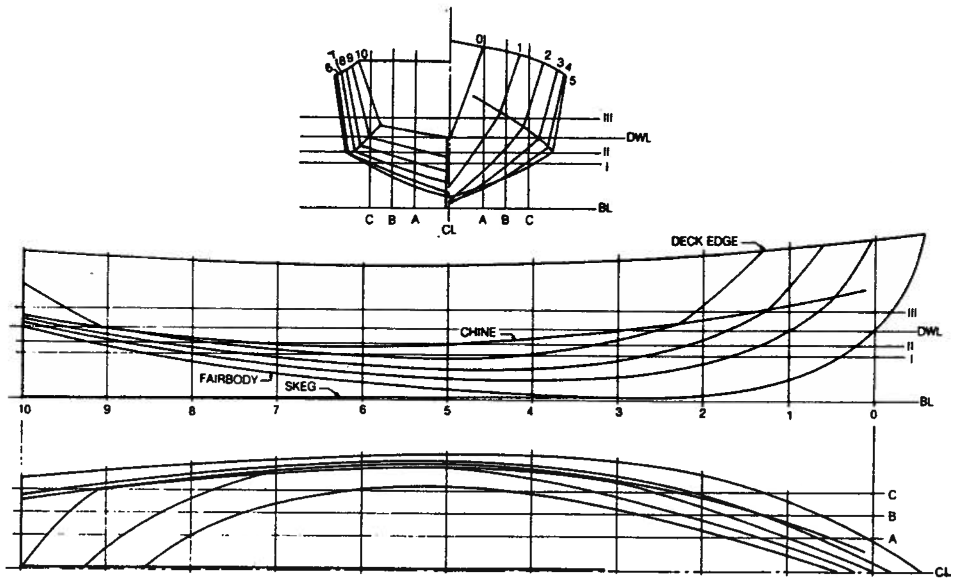


Fig. 7 Lines of small developable surface vessel

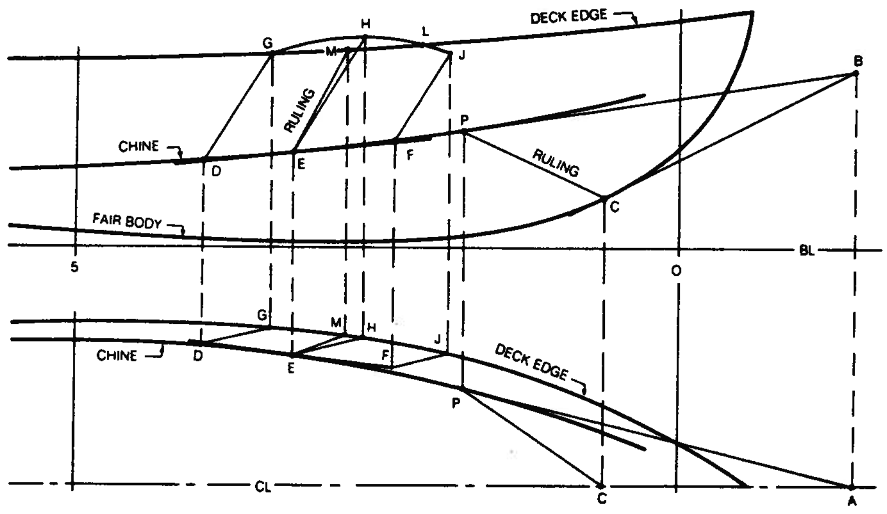


Fig. 8 Construction drawing for developable surface lines

To find ruling between deck edge and chine:

1. Draw $\overline{DF}$, tangent to chine at E.
2. Draw $\overline{DG}$, $\overline{EH}$ and $\overline{FJ}$ in half-breadth plan parallel to each other at arbitrary angle.
3. Project G to deck edge in profile.
4. Find points M and J in profile as projections from half-breadth where $\overline{DG}$, $\overline{EH}$ and $\overline{FJ}$ are parallel.
5. Draw curve GHJ in profile cutting deck edge at L.
6. M is midpoint of GL and is end of ruling $\overline{EM}$

   Plane DGJF is tangent to chine at E.

Curve GHLJ is the intersection of plane DGJF with a cylindical surface with vertical elements through deck edge.

To find ruling between stem profile and chine:

1. Draw $\overline{PA}$ and $\overline{PB}$ tangent to chine at P.
2. Project point A in half breadth at CL, to point B in profile.
3. Draw $\overline{BC}$ tangent to stem profile at C, giving end C of ruling $\overline{PC}$.

Figure 1.6: Kilgore's Manual Graphical Solution

This led to implementing a mathematical solution once computers had evolved to the point where it was a viable alternative. The first computer solutions which had a moderate success were those by Nolan [17] and Clements [7].

1.3.2 Nolan and Clement Computer Approach

One of the first well known published works involving a computer-aided approach to developable surface design was by T. J. Nolan [17]. In his paper he emphasized that a mathematical approach utilizing a computer proved to yield a substanitial increase in speed and precision for calculating a developable surface. Nolan noted that an infinite number of surfaces can be found to span the curves but the developable surface is unique in that it requires the minimum strain energy of flexture and that in a developable surface bending is restricted to nonintersecting axes lying in the surface so that section moduli and bending stresses are minimized for any given radius of curvature. As a result, a developable surface can be formed elastically from a plane sheet, while the surface fitting the same pair of curves but having compound curvature must undergo a costly plastic forming process. Nolan defines a developable surface as, "a developable surface spanning a pair of curves in space may be defined as the locus of straight lines or "rulings" which represent the line of contact of a plane which is tangent to the surface. The rulings are neutral axes of bending and must not intersect within the surface"[17].

Nolan's approach utilizes a Theilheimer spline interpolation and a vector representation to create a mathematical description for a developable surface. The approach is relatively simple, involving a representation of the tangents, normals and rulings in vector form. He iteratively solves his mathematical model to yield a zero angle for the cross product of the two normals of the space curves. See figure 1.7.

Nolan's vector approach calculates the normal of each space curve as $\overrightarrow{N_1} = \overrightarrow{R} \times \overrightarrow{T_1}$, and $\overrightarrow{N_2} = \overrightarrow{R} \times \overrightarrow{T_2}$, where R is the ruling and T_1 is the tangent calculated at that point

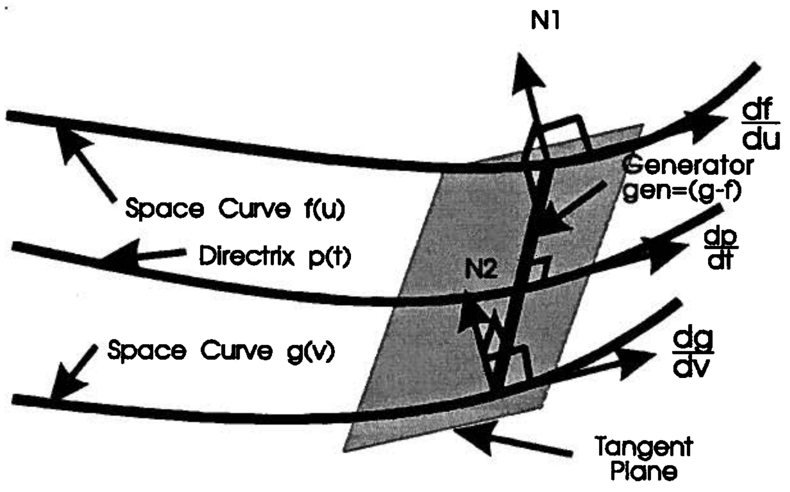


Figure 1.7: Nolan's Vector Description of Computer Solution

along space curve 1 and T_2 is the tangent calculated at a point on space curve 2. He then uses one of the constraints of a developable surface, namely that $\overrightarrow{N_1} \times \overrightarrow{N_2} = 0$.

This approach is moderately successful in that for very simple surfaces it may yield a resulting developable surface. However, all too often the rulings either cross or "fan" yielding an unrealistic or nonusable surface. Also, the Theilheimer spline interpolation is not of parametric form resulting in singularities which may occur in a three dimensional representation of the surface.

Clement's Solution to the problem was published approximately ten years later utilizing cubic spline functions, again in non-parametric form[7]. He states, "Between each pair of chine lines that ruled surface generated which has the same tangent plane at all points of each generator or ruling line. A procedure based on the multiconic development of a surface is used to modify the given chine lines to ensure that no ruling lines intersect at a point within the surface. The result is a developable surface"[7]. In addition, Clement's approach generated tables of offsets. This approach, like Nolan's, also had problems arising at singularities and generators crossing in areas of regression on the surfaces.

1.3.3 Normal Directrix Approach by Dunwoody

The approach taken by Dunwoody is unique in that it defines a developable surface by means of a normal directrix and an initial generator. The directrix is modelled by a parametric cubic spline; initially a uniform non-rational B-spline was used. Information about the spline in the parametric form is the position in space at a parametric value t, its tangent and its curvature. A start generator direction is also needed.

The differential equation derived must retain the constraints which define a developable surface. Refering to figure 1.8 to show the geometry, the following constraints are shown and the resulting differential equation is formed. Details of the proof can be seen in appendix B of this thesis.

From figure 1.8 the following vector definitions are in order:

$$\overrightarrow{g} = \text{generator vector} \tag{1.1}$$

$$\overrightarrow{\frac{dg}{dt}} = \text{derivative generator vector} \tag{1.2}$$

$$\overrightarrow{\frac{dp}{dt}} = \text{directrix tangent vector} \tag{1.3}$$

$$\overrightarrow{\frac{d^2p}{dt^2}} = \text{directrix curvature vector} \tag{1.4}$$

$$\Delta T = \text{step increment} \tag{1.5}$$

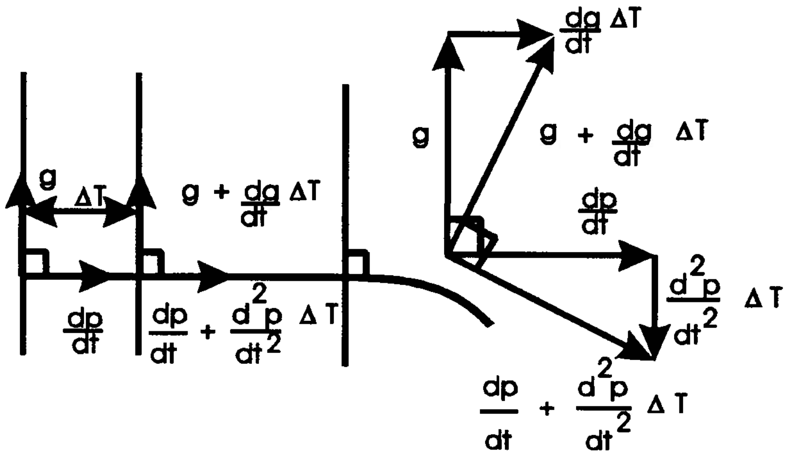


Figure 1.8: Normal Directrix Approach by Dunwoody

Three constraints are necessary for a surface to be developable. They are stated as follows using the above nomenclature:

1. The normal directrix and generator vectors must be perpendicular.

$$\vec{g} \cdot \overrightarrow{\frac{dp}{dt}} = 0.$$

Differentiation with respect to t yields

$$\overrightarrow{\frac{dg}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} = -\overrightarrow{\frac{d^2p}{dt^2}} \cdot \vec{g}$$

2. The vector $\vec{g}$ is of unit length.

$$\vec{g} \cdot \vec{g} = 1.$$

Differentiation with respect to t yields

$$\vec{g} \cdot \frac{\overrightarrow{dg}}{dt} = 0$$

3. The normal is invariant along a generator.

$$\left(\frac{\overrightarrow{dp}}{dt} \times \vec{g} \right) \cdot \frac{\overrightarrow{dg}}{dt} = 0$$

Combining the constraints yields the following differential equation describing the next consecutive generator:

$$\frac{\overrightarrow{dg}}{dt} = -\frac{\left(\frac{\overrightarrow{d^2p}}{dt^2} \cdot \vec{g} \right)}{\left(\frac{\overrightarrow{dp}}{dt} \cdot \frac{\overrightarrow{dp}}{dt} \right)} \frac{\overrightarrow{dp}}{dt} \tag{1.6}$$

Integrating this differential equation using such integration routines as Runge-Kutta fourth order forces a solution to be output. This approach will yield a particular solution. From this stage of the analysis, given a directrix and a starting generator of unit length, a unique developable surface results from the differential equation described above.

There are limitations with this technique. It is not yet in a form which would prove to be of any practical use. Further constraining is necessary in order to control the behavoir of the developable surface.

This approach is where the present research project started and expanded implementing new terminology and concepts which will be presented in detail.

1.4 New Approach Research Objectives

The new approach research objectives were based on expanding the work initiated by Dunwoody utilizing the normal directrix approach. This approach is unique in that it will always yield a solution. In this state, however, it is not very useful from a practical engineering view point since it does not match two space curves. This leads to the research objectives presented in this thesis to further refine this technique.

- The first research objective was to develop an algorithm in order to find a normal directrix such that the resulting developable surface lay close to two space curves, representing desired edges of the developable surface.
- Another research objective was to create an algorithm to intersect developable surfaces and to generate the flat plate layouts and angles.
- The final objective was to implement these algorithms using a modern computer language and a popular CAD package in order to assess the practicality of the approach.

Chapter 2

Splines

This chapter is included because the material covered on splines contains the necessary background in order to derive the additional tools and equations for developable surfaces. The two types of splines discussed in this chapter were selected because of their particular characteristics which proved to be useful for a designer.

2.1 Types of Splines Chosen

When considering using a mathematical representation for space curves one can classify them as of either parametric or non-parametric form. Non-parametric forms are used extensively in various fields of mathematics and engineering. Non- parametric curves can be further categorized as either explicit or implicit. The explicit non parametric form is usually expressed in the following form:

$$y = f(x)$$

where,

x	= independent variable
f(x)	= function of independent variable
y	= dependent variable

In this form multiple-valued or closed curves cannot be expressed. To overcome this form, one usually uses the implicit form of the non parametric curve in the following typical form:

$$f(x, y) = 0$$

where,

f(x,y) = A function of both x and y

One typically calculates a point on the curve by calculating the roots of the equation. The approach can sometimes prove to be fairly computationally expensive. Implicit formulation is a very common form of non-parametric polynomials. Many formulations used in engineering require higher than third order thereby making computations even more expensive when solving for roots of the equation.

The non-parametric implicit formulation presents difficulties when being applied in defining such three dimensional objects as ship hull curves and surfaces. One typical problem that arises is when a vertical slope is encountered along the curve or surface resulting in an infinite numerical value. An infinite number cannot be used in a numerical calculation when using a computer. Another problem arising in non parametric implicit form is that the positions are not distributed evenly along the curve or surface. This poses a problem when trying to present the curves or surfaces graphically on computers [19].

Parametric curves solve the problems presented above and are suitable for representing closed curves and curves with multiple values of an independent variable. Parametric curves are also axis independent. Parametric curves replace the use of geometric slopes (which may be infinite) with parametric tangent and curvature vectors, which are never infinite. In parametric form a curve is usually represented by a *piecewise polynomial*. Each segment of the curve is given by three functions x(t), y(t), z(t), which are polynomials in the parameter t. [12] For example:

$$P(t) = [x(t), y(t), z(t)]$$

After determining that parametric curves would be used in this work, the order and desired characteristics of the polynomials were investigated. Considering the types of applications and desired flexibility of the types and characteristics of curves desired, a fairly exhaustive investigation of various types of splines was conducted. Special cubic polynomials derived in the format pioneered by Barsky[3], DeRose[9], Forrest, Coons, Bezier and furthered by others were decided upon.

The initial stages of development of the Basis functions to taylor the desired behaviour of the splines yield cubic polynomials. Cubic polynomials are most often used because lower-degree polynomials give too little flexibility in controlling the shape of the curve, and higher-degree polynomials can introduce unwanted "wiggles" and also require more computation. No lower-degree representation allows a curve segement to interpolate (pass through) two specified end points with specified derivatives at each endpoint. Given a cubic polynomial with its four coefficients,

eg.

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

four knowns are used to solve for the unknown coefficients. For example, the four knowns might be the two endpoints and the derivatives at the endpoints. Other knowns might be slopes or additional points[12]. It should also be noted that parametric cubics are the lowest-degree curves that are nonplanar in 3-D (three dimensions). You can see this by recognizing that a second-order polynomial's three coefficients can be completely specified by three points and that three points define a plane in which the polynomial lies. Higher-degree curves require more conditions to determine the coefficients and can "wiggle" back and forth in ways that are difficult to control. The parametric curves used in this thesis are given in terms of their degree n, which is fixed at 3.

Much research has been done by such modern pioneers as Barsky[3], who developed *Basis* functions, and appropriate nomenclature on the various levels and types of curve continuities. No detailed analyses of the derivation of the splines used in this thesis will be discussed in substantial detail since this work has already been done by such authors as those previously cited. Only enough explanation of the nomenclature used in this thesis to familiarize the reader with the concepts and characteristics of the various forms of the splines used will be discussed.

Another point to mention is that local control of the 3-D curves was desired so that a curve segment is completely controlled by only four control vertices; therefore, a point on a curve segment can be regarded as a weighted average of these four control vertices.

The parametric splines used in this thesis are presented in the following form:

$$P(i+t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} \tag{2.1}$$

where,

$$P(t) = [x(t), y(t), z(t)] \tag{2.2}$$

and

$$\begin{aligned} Point &= P(t) = & \left[t^3\ t^2\ t\ 1\right][\ \ ][\ \ ] \qquad (2.3)\\ Tangent &= T(t) = \tfrac{dP(t)}{dt} = & \left[3t^2\ 2t\ 1\ 0\right][\ \ ][\ \ ] \qquad (2.4)\\ Curvature &= C(t) = \tfrac{d^2P(t)}{dt^2} = & [6t\ 2\ 0\ 0][\ \ ][\ \ ] \qquad (2.5) \end{aligned}$$

2.2 Uniform Parametric, Geometric and Non-Rational Continuity Requirements

One of the important properties discussed in such fields as finite elements and computer aided geometric design is of the mathematical techniques of shape representation. It is termed *continuity.* Continuity can be described as the highest level of differentiation which is continuous [3]. The types of continuity can be further categorized. Four types of continuity are considered in this analysis. Each type of continuity will be explained briefly. The types of continuity chosen can be expanded but only two of what was thought to be generally the most useful were selected at this stage of the research.

The first type of continuity requirement considered is whether or not the splines used in the analysis are either uniform or non-uniform. Since the splines are parameterized, and uniform parametric splines can be expressed in a pseudo standard format, these types of splines appeared to be a likely choice. Non- uniform splines are not able to be expressed in the format that was adopted in this reseach at this time.

Another type of continuity requirement which was desired was parametric continuity [18]. Parametric continuity can be explain quite briefly with the aid of Figure 2.1. If the n^{th} derivative vector of two cubic curve segments are equal (ie. their direction and magnitudes are equal) at the segments' join point, the curve has n^{th}-degree continuity in the parameter t, or *parametric continuity*[12]. One would then state that if the direction and magnitude of $\frac{dt}{dt^n}[P(t)]$ through the n^{th} derivative are equal at the join point, the curve is called C^n *continuous.* Figure 2.1 shows a curve segment S joined to three different curves with three different degrees of continuity ascertained by the superscript above the C. One should note that a parametric curve segment is itself everywhere continuous; the continuity of concern here is at the join points[12].

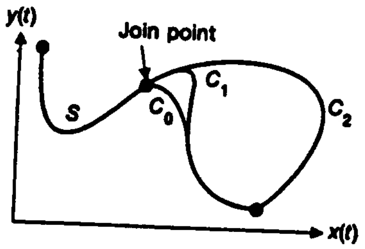


Figure 2.1: First Three Levels of Parametric Continuity

If two curve segments join together, the curve has G^0 *geometric continuity* . If the directions (but not necessarily the magnitudes) of the two segments' tangent vectors are equal at the joint point, point the curve has G^1 *geometric continuity.* In computer- aided design of objects, G^1 continuity between curve segments is often required. G^1 continuity means that the geometric slopes of the segments are equal at the join point. For two tangent vectors TV_1 and TV_2 to have the same direction, it is necessary that one be a scalar multiple of the other. We then state the relationship that $TV_1 = k \cdot TV_2$ with $k > 0$ [3][12]

One should note that in general, C^1 continuity implies G^1, but the converse is generally not true. G^1 continuity is generally less restrictive than is C^1, so curves can be G^1 but not necessarily C^1. However, visually, join points with G^1 continuity will appear just as smooth as those with C^1 continuity as can be seen in Figure 2.2.[12].

In Figure 2.2 curve segments Q_1, Q_2, Q_3 join at the point P_2 and are identical except for their tangent vectors at P_2. Q_1 and Q_2 have equal tangent vectors , and hence are both G^1 and C^1 continuous at P^2. Q_1 and Q_3 have tangent vectors in the same direction but Q_3 has twice the magnitude , so they are only G^1 continuous at P_2[12]

Another type of continuity which one may desire is Rational Continuity [13]. Rational continuity can be simply defined for "general rational cubic curve segments as ratios of polynomials:

$$x(t) = \frac{X(t)}{W(t)}; y(t) \;=\; \frac{Y(t)}{W(t)}; z(t) = \frac{Z(t)}{W(t)} \tag{2.6}$$

where $X(t)$, $Y(t)$, $Z(t)$ and $W(t)$ are all cubic polynomial curves whose control points are defined in homogeneous coordinates. We can also think of the curve as existing in homogenous space as:

$$Q(t) \;=\; [X(t)Y(t)Z(t)W(t)] \tag{2.7}$$

As always, moving from homogeneous space to 3 space involves dividing by $W(t)$. Any non rational curve can be transformed to a rational curve by adding $W(t) = 1$ as a fourth element" [12].

No splines were used in this thesis at this stage which included Rational continuity. In future work this type of continuity requirement may be implemented if requested. For further reading one may refer to either Barsky and Hohmebar [13] or Foley [12]

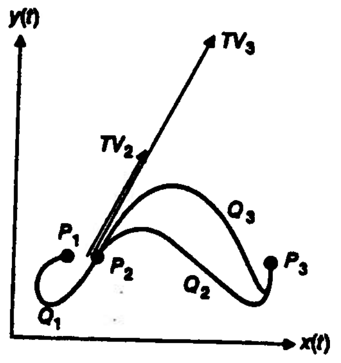


Figure 2.2: Comparing Geometric and Parametric Continuity

2.3 Uniform Non-Rational Tension Catmull-Rom Spline

The uniform non-rational Tension Catmull-Rom spline was chosen because it exhibits several useful features the designer may require [8] [9]. First, it is an interpolating spline, meaning that the curve passes through the points (control vertices). Second, it is in parametric form, meaning that it does not encounter singularities, only variations of vector magnitudes. Third, it exhibits desired parametric and geometric continuity requirements. Fourth, it has a global tension parameter which can further control the shape of the desired curve.

The uniform non-rational Tension Catmull-Rom spline is easiest described in vector-matrix form. In this form it is a relatively simple task to imbed into a program. The vector-matrix format is in a form in which not only the position along the curve can be calculated but also the tangent, curvature and other vector specific relations desired. This vector-matrix format is now almost a standard form in which these types of splines are presented.

This pseudo-standard form shows that the spline has a local influence of the control vertices as any chosen position. At any one particular location along the curve the control vertices are only influenced by the previous one and the next two. This is shown in the "[P]" vector of the vector-matrix form. The "[P]" vector shows that at a particular position along the curve the only infuence is from $P_{i-1}, P_i, P_{i+1} and P_{i+2}$.

Taking into account the end conditions of the splines also had to be considered. The approach taken was to create *Phantom* points. Given the control vertices vector below, *Phantom* end conditions were formulated. Detailed analysis of the derivation can be referred to in the appendices D.1 D.2. The vectors located below show the indexing of the control vertices and how end conditions are treated (phantom points).

$$\begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

For the initial condition P_0 the control vertices vector has the following form D.1:

$$\begin{bmatrix} P_i \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

For the end condition P_{n-1} the control vertices vector is in the following form D.2:

$$\begin{bmatrix} P_i \\ P_i \\ P_{i+1} \\ P_{i+1} \end{bmatrix}$$

The following page shows the vector-matrix form of the ***uniform non-rational Tension Catmull-Rom spline.*** The spline is parameterized with respect to the parametric variable t and has a global tension variable, β.

$$P(t) = \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix} \cdot \frac{1}{2} \begin{bmatrix} -2.0\beta & 4.0 - 2.0\beta & 2.0\beta - 4.0 & 2.0\beta \\ 4.0\beta & 2.0\beta - 6.0 & 6.0 - 4.0\beta & -2.0\beta \\ -2.0\beta & 0.0 & 2.0\beta & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

$$\beta = Tension\,parameter$$

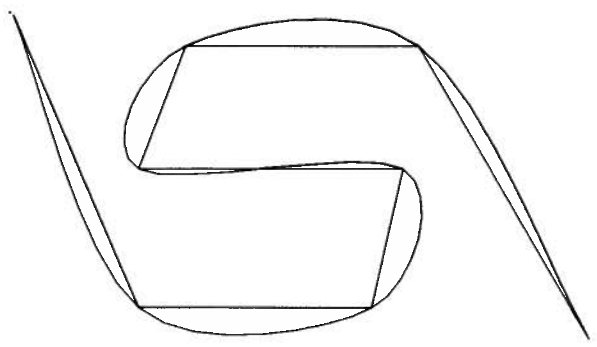
To give the reader a good comparison as to the behaviour of the Tension Catmull-Rom spline the following figures are included. The first figure, figure 2.3(a) shows the spline with a tension value of 0.5 shown relative to interconnected line segments. For the tension Catmull-Rom spline with a value of 0.5 this defaults to a traditional cardinal spline.

The next figure, Figure 2.3(b) shows little difference but is relaxing the spline tension when given a tension value of 1.0.

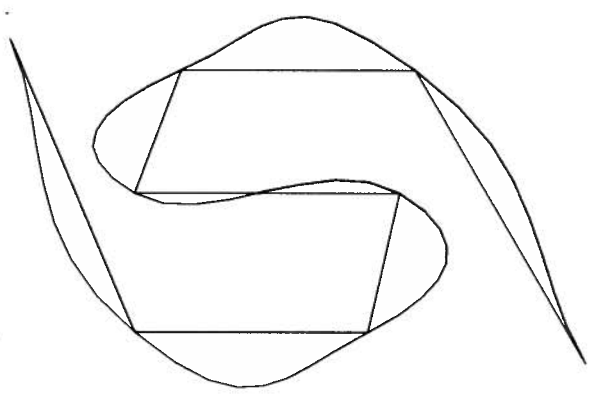
Finally the Tension Catmull-Rom spline in Figure 2.3(c) shows how it nearly contours to the inter-connected line segments shown in the figure. If the tension parameter is given a value of 0.0 then it becomes line segments.

2.4 Uniform Non-Rational Beta-Spline

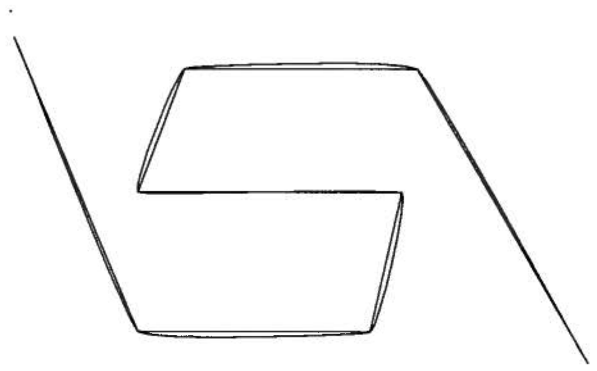
The uniform non-rational Beta-spline was chosen to provide other useful features. First, it is an approximating spline meaning that the curve passes near, not through, the control vertices. It also exhibits the convex-hull property which is also shared by the B-spline [18] and the Bezier [4] spline. Second, the Beta-spline is derived parametrically so it too does



(a) Tension at 0.5 Catmull-Rom



(b) Tension at 1.0 Catmull-Rom



(c) Tension at 0.1 Catmull-Rom

Figure 2.3: Catmull-Rom Splines

not have any singularities occur. Third, it also retains desired parametric and geometric continuity requirements. And, fourth, it has global *bias* and *tension* parameters which further enable the designer to better adjust the spline [3].

The vector-matrix form of the Beta-spline is shown on the following page:

$$P(t) = \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix}$$

$$\cdot\frac{1}{\delta}\begin{bmatrix} -2.0\beta_1^3 & 2.0(\beta_2+\beta_1^3+\beta_1^2+\beta_1) & -2.0(\beta_2+\beta_1^2+\beta_1+1.0) & 2.0 \\ 6.0\beta_1^3 & -3.0(\beta_2+2.0\beta_1^3+2.0\beta_1^2) & 3.0(\beta_2+2.0\beta_1^2) & 0.0 \\ -6.0\beta_1^3 & 6.0(\beta_1^3-\beta_1) & 6.0\beta_1 & 0.0 \\ 2.0\beta_1^3 & (\beta_2+4.0\beta_1^2+4.0\beta_1) & 2.0 & 0.0 \end{bmatrix}$$

$$\cdot\begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

$$\begin{aligned} \delta &= \beta_2+2.0\beta_1^3+4.0\beta_1^2+4.0\beta_1+2.0 \\ &\neq 0 \end{aligned}$$

$$\begin{aligned} \beta_1 &= Bias \\ \beta_2 &= Tension \end{aligned}$$

Like the Tension Catmull-Rom spline the Beta-spline exhibits the same localized control vertices influence. Again, if one were to choose a particular position along the curve the closest control vertex would only be influenced by the preceding one and the next two, eg. $[P_{i-1}, P_i, P_{i+1}, P_{i+2}]$.

Taking into account the end conditions of the spline also had to be considered in the same fashion as the Catmull-Rom spline. The same approach was taken to create the *Phantom* points. Given the control vertices vector below, *Phantom* end conditions were formulated. Detailed analysis of the derivation can be referred to in the appendices E.1 E.2.

The initial condition control vertices vector for the Beta-spline at P(0) is:

$$\begin{bmatrix} \frac{(\beta_2+4.0\beta_1^2+4.0\beta_1)P_i+2.0P_{i+1}}{\delta-2.0\beta_1^3} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

The end condition control vertices vector for the Beta-spline at P(n-1) is:

$$\begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ \frac{2.0\beta_1^3 P_i+(4.0+4.0\beta_1+\beta_2)P_{i+1}}{\delta-2.0} \end{bmatrix}$$

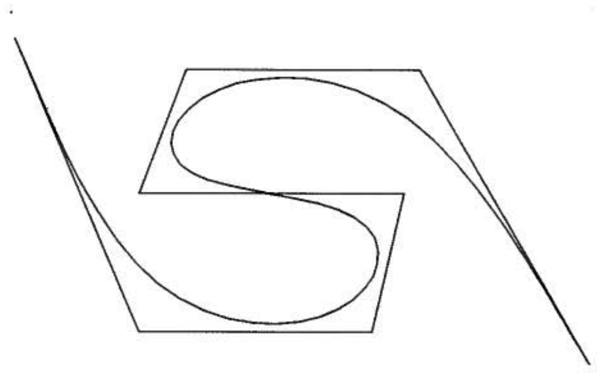
The first of the Beta-spline figures shown below reveals the spline compared to inter-connected line segments. The first figure, figure 2.4(a) shows the curve near the control vertices, exhibiting the characteristics of an approximating spline. The bias is at 1.0 and the tension is at 0.0. With these values the Beta-spline degenerates to a B-spline, which exhibits first and second order parametric continuity.

The next figure, figure 2.4(b), shows the Beta-spline with a Bias of 1.0 and a tension at 25.0. At these values one can see how this spline can also resemble inter-connected line segments if the tension value is increased substantially more.

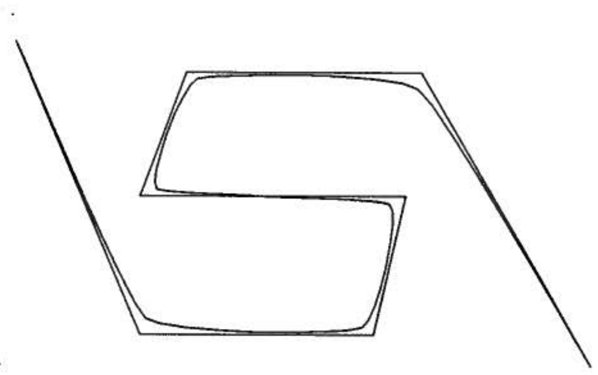
In figure 2.4(c) one can see how the curve behaves if the tension parameter is given a negative value of about -0.05.

Finally, if the bias is changed, as in figure 2.4(d), to a value of 1.5 and the Tension is left at a value such as 0.0 the curve exhibits the following behaviour

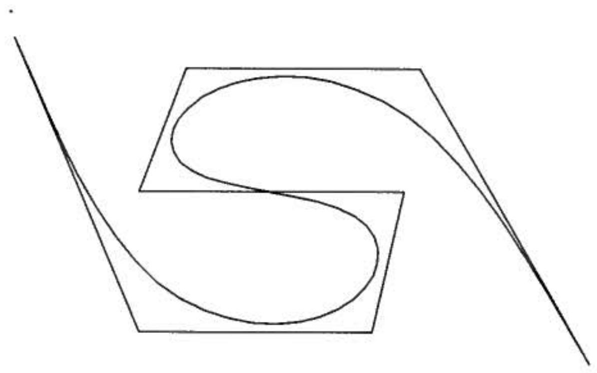
These figures shown above try to give the reader some idea of the capabilities of these type of splines and how one can use the features each spline possesses. These are but a few of the types of splines which are now being developed. Each of these types of splines has features which the reader should be aware of in order to maximize the benefits they have to offer.



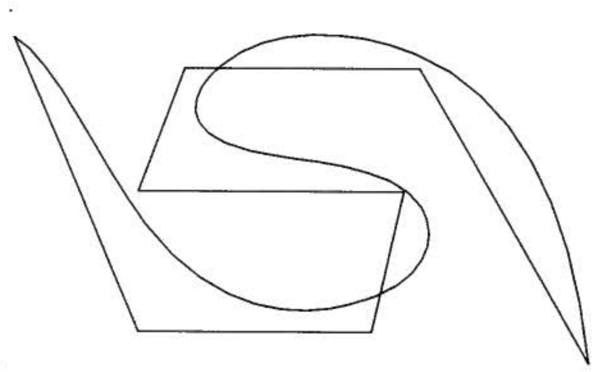
(a) Bias 1.0 Tension 0.0 Beta



(b) Bias 1.0 Tension 25.0 Beta



(c) Bias 1.0 Tension -0.05 Beta



(d) Bias 1.5 Tension 0.0 Beta

Figure 2.4: Beta Splines

Chapter 3

Creation of a Normal Directrix from Two Space Curves

The normal directrix approach will always yield a smooth developable surface in the vicinity of the normal directrix, so long as the normal directrix is itself smooth. Unfortunately, it is not always clear what shape the normal directrix should take in order that the resulting surface should meet the requirements of the designer. With respect to the requirements of the designer, the definition of a developable surface from two edges is superior. The objective of the present work is to create an algorithm which will shape a developable surface to lie close to a pair of edge curves. The developable surface will be specified in terms of a normal directrix in order to ensure that the surface is smooth. It is not expected that the developable surface will contain the two edge curves, only that it will be close to the two curves.

The creation of a normal directrix from two space curves follows from these criterion:

1. A normal directrix can be computed for any developable surface by starting at one point on the first generator, then constructing a curve which lies within the surface and is perpendicular to all generators. In addition, extra construction tools, in the form of differential equations, were created in order to better control the normal directrix solution.

2. Once a normal directrix has been computed, it can presumably be smoothed out to yield a smoother developable surface without departing greatly from the original equation.

3. Nolan's approach of matching the cross products between the generator and the tangents to each of the edge curves can be expressed in terms of a differential equation.

4. The curve of the normal directrix can also be expressed as a differential equation, to be solved in conjunction with the differential equation for the set of generators.

5. If the normal directrix is to be described by a spline with n control points, then the values for those n points can be computed to yield a spline which lies close to the normal directrix derived from the differential equations.

The approach taken to try to match a developable surface to two edge curves resulted in formulae modelled by differential equations. The differential equation version of the conventional method, named the modified conventional approach, created by Dunwoody and Konesky was used as an initial guess in order to utilize additional differential equations to solve for directrix control vertices.

3.1 Modern Approach Utilizing a Single Normal Directrix

A normal directrix can be computed for any developable surface by starting at one point on the first generator, then constructing a curve which lies within the surface and is perpendicular to all generators. This very powerful technique was created by Dunwoody which is termed in this thesis as *the modern approach.* In addition, extra construction tools, in the form of differential equations, were created in order to better control the normal directrix solution.

This modern approach, given a normal directrix and a start generator position, will always force a developable surface to be created. This solution is underconstrained, however, and further refinement was deemed necessary in order to better control the behaviour of the function.

The formulation is in vector differential equation form and uses the ODE class integration routine which implements Runge-Kutta 4th order. Constraints for this developable surface differential equation are given in the next subsection. For a more thorough analysis one can refer to Appendix B for the derivation.

3.1.1 Constraints Governing Modern Approach

The differential equation developed by Dunwoody [10] termed the *modern approach* involves three constraints which define a developable surface. They are as follows:

1. The generator must be of unit length

   ie.
   $$g \cdot \frac{dg}{dt} = 0$$

2. The vector normal is invariant along a generator

   ie.
   $$(g \times \frac{dp}{dt}) \cdot \frac{dg}{dt} = 0$$

3. Vectors must be perpendicular

   ie.
   $$\frac{dg}{dt} \cdot \frac{dp}{dt} = -\frac{d^2p}{dt^2} \cdot g$$

The three constraints can be seen in Figure 3.1 below in vector form.

Using the three constraints which define a developable surface yields the differential equation in simplified form:

$$\frac{dg}{dt} = -\frac{(\frac{d^2p}{dt^2} \cdot g)}{(\frac{dp}{dt} \cdot \frac{dp}{dt})}\frac{dp}{dt} \tag{3.1}$$

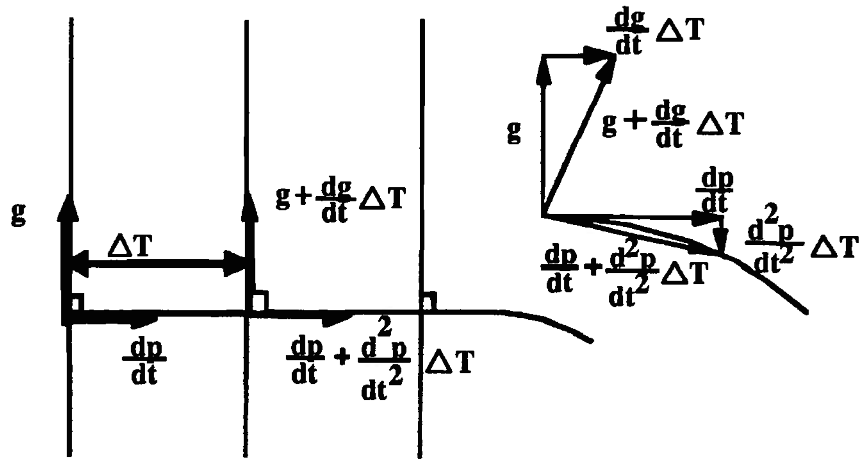

g
ΔT
$g+\frac{dg}{dt}\Delta T$
$\frac{dp}{dt}$
$\frac{dp}{dt}+\frac{d^2p}{dt^2}\Delta T$
$\frac{dg}{dt}\Delta T$
g
$g+\frac{dg}{dt}\Delta T$
$\frac{dp}{dt}$
$\frac{d^2p}{dt^2}\Delta T$
$\frac{dp}{dt}+\frac{d^2p}{dt^2}\Delta T$

Figure 3.1: Derivation of Developable Surface

This equation works fine as is but further constraining is necessary in order to make this solution practical. For example, given the theory presented so far we can generate a developable surface along a normal directrix given an initial generator position. However, more realistically, one would also want the surface to end up in alignment with a desired final generator position. We now move to the next stage in the development of alignment of the end generators.

3.1.2 Alignment of End Generators

Once a normal directrix has been computed, it can presumably be smoothed out to yield a smoother developable surface without departing greatly from the original equation. In addition, a much more realistic and desireable condition is where the user gives a normal directrix, a start generator position and a final generator position and the configuration adjusts itself to conform to the newly added constraints.

Change in Angle With Respect to Unit Motion of a Control Vertex

The problem was approached by looking at the problem using the perspective of observing the location of the directrix, the position of the generator and separate the vectors into in-plane and out-of-plane components. Another differential equation was formulated which accumulated information of the rate-of-rotation of a generator with respect to motion of the control vertices along the normal directrix . This formulation, as will be shown, proves to be very useful in storing information about the surface and how to correct accordingly to match the end generator. Figure 3.2 shows the relation of the original directrix and the *corrected* directrix as well as the amount of change that is necessary.

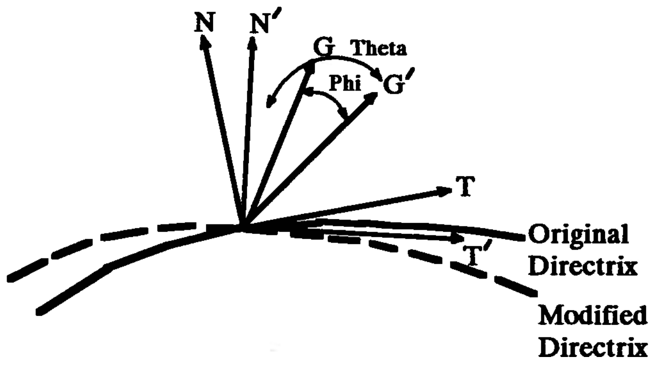


Figure 3.2: Vector Locations and Corresponding Angles

A detailed analysis of the derivation of the differential equation can be referenced in Appendix C if the reader wishes to look further. A brief summary of the highlights of the derivation is needed here in order to familiarize the reader with new concepts which are being introduced with the theory and the nomenclature of the user controllable design variables.

The angle Theta, herein referred to as θ, is the out-of-plane rotation of G' with respect to G. The angle Phi, is herin referred to as ϕ, and is the in-plane rotation of G' with respect to G. The variable "a" is a parameter describing a directrix control vertex component.

The rate of rotation of a generator with respect to changes in one of the parameters of the directrix curve is written as:

$$\frac{d^2\theta}{dadt}$$

The desired expression is the rotation of a generator with respect to changes in one of the parameters of the directrix, which is written as:

$$\frac{d\theta}{da}$$

The desired expression can be defined in terms we can derive, namely:

$$\frac{d\theta}{da} = \frac{dg}{da} \cdot N \tag{3.2}$$

where, N is the unit normal defined as: (3.3)

$$N = T \times g \tag{3.4}$$

but, (3.5)

$$\frac{d}{dt}\frac{d\theta}{da} = \frac{d}{dt}\left(\frac{dg}{da} \cdot N\right) \tag{3.6}$$

rewriting gives, (3.7)

$$\frac{d^2\theta}{dadt} = \frac{d^2g}{dadt} \cdot N + \frac{dg}{dt} \cdot \frac{dN}{dt} \tag{3.8}$$

Using numerous identities and proofs, the user can look at the derivation in detail in Appendix C, the resulting differential equation which contains both the in and out-of-plane components result in the following formulation:

$$\frac{d^2\theta}{dadt} = -\frac{(\frac{d^2p}{dt^2} \cdot N)(\frac{d^2p}{dadt} \cdot g)}{(\frac{dp}{dt} \cdot \frac{dp}{dt})} + \frac{(\frac{d^2p}{dadt} \cdot g)(\frac{d^2p}{dt^2} \cdot N)}{(\frac{dp}{dt} \cdot \frac{dp}{dt})} \tag{3.9}$$

Using several identities and constraints that have already been presented the final form simplifies down to the following relation:

$$\boxed{\frac{d^2\theta}{dadt} = \frac{(\frac{d^2p}{dt^2} \times \frac{dp}{dt})}{(\frac{dp}{dt} \cdot \frac{dp}{dt})^{\frac{3}{2}}} \cdot \frac{d^2p}{dadt}} \tag{3.10}$$

The Alignment Process and the Concept of Mobility

The alignment process involves a summation of the $\frac{d^2\theta}{dadt}$ term from t = 0.0 to t = N-1. The summation can be written in equation form as follows:

$$\int_0^{N-1} \frac{d^2\theta}{dadt} dt = \int_0^{N-1} \frac{(\frac{d^2p}{dt^2} \times \frac{dp}{dt})}{(\frac{dp}{dt} \cdot \frac{dp}{dt})^{\frac{3}{2}}} \cdot \begin{bmatrix} \frac{d^2p}{dadt}\{\delta - 1.0\} \\ \frac{d^2p}{dadt}\{\delta\} \\ \frac{d^2p}{dadt}\{\delta + 1.0\} \\ \frac{d^2p}{dadt}\{\delta + 2.0\} \end{bmatrix} dt \tag{3.11}$$

where, $\frac{d^2p}{dadt}$ is defined as:

$$\frac{d^2p}{dadt} = \begin{bmatrix} 3.0t^2 & 2.0t & 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & w \end{bmatrix} \begin{bmatrix} \delta - 1.0 \\ \delta \\ \delta + 1.0 \\ \delta + 2.0 \end{bmatrix} \tag{3.12}$$

The alignment process may take several passes, ie. from 0 to N-1, a correction, then from 0 to N-1, etc.

The procedure will be described and the successive passes usually reduce the error by one decimal place per pass (ie. iteration).

After the first accumulation of information along the spline from t = 0 to t = N-1, a few constraints are added. The first desired constraint of the movement of the control vertices is that the end control vertices can only move in the direction along each corresponding end generator. The second from the end control vertices are constrained to move both in the direction of the end generators and in the direction of the start and end tangent vectors respectively. Both ends are calculated in the same way, so for a better understanding only the t=0.0 and t=1.0 end conditions will be explained.

At t = 0.0 the control vertex located here only has the freedom to move in the direction of the starting generator. The end condition constraints are also influenced by the phantom end conditions in addition to depending upon the type of spline being used. For this analysis the degenerated version of the Beta spline to a B-spline will be used.

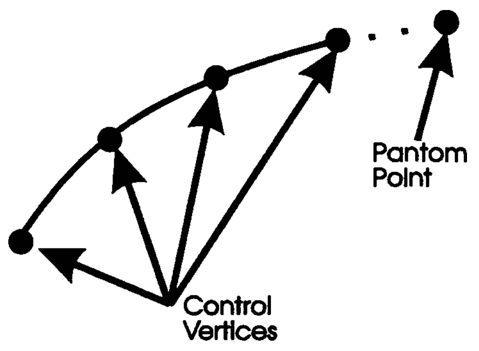


Figure 3.3: Control Vertices and End Phantom Point

At the end of the spline the end two control vertices of the $\frac{d^2\theta}{dadt}$ summed terms have to be modified because of the phantom end condition constraints. The end control vertices are influenced by the following condition which describes only the B-spline criterion: $2P_i - P_{i+1}$. From this relation we have to add $\frac{d^2\theta}{dadt}$ again to itself for the P_i control vertex of information and subtract $\frac{d^2\theta}{dadt}$ of the P_{i+1} component from itself. This is in essence the technique which is applied to the end conditions of the $\frac{d^2\theta}{dadt}$ control vertices for each type of spline being used.

The concept of mobility is quite simple and provides a further constraining on the behaviour of the alignment process. Each control vertex of $\frac{d^2\theta}{dadt}$ are assigned a mobility value, which is a constant. A mobility value of 1.0 means that the corresponding $\frac{d^2\theta}{dadt}$ vertex has full mobility and that it is free to adjust that control vertex as governed by the equation. At the other extreme a mobility value of 0.0 indicates that the $\frac{d^2\theta}{dadt}$ vertex is not to be moved during each iteration of the alignment process.

3.1.3 Analysis of Results

Many tests were done with the *modern approach* and then it was incorporated as a foundation to build upon. The approach was found to be very robust but specifically controlling its behaviour became the dominant problem. It was also noted to have one major instability. This occurred when the plate was very close to being perfectly flat. This was not deemed to be a very major problem since if the plate was flat, a solution existed already, thereby, no need of the *modern approach* would be required.

Mobius Strip Demonstrating Flexibility

As an example which demonstrates both the flexibility of the *modern approach* ie. the control vertices were not permanently fixed in position, (NB. only the end conditions were given a hard constraint). One challenge was to create a *developable* mobius strip. The mobius strip is a geometric anomally in that its edges are infinitely continuous, ie.

if one was to follow an edge it would continue infinitely along the surface travelling along both sides. For clarity, the reader should refer to Figure 3.4 in this chapter.

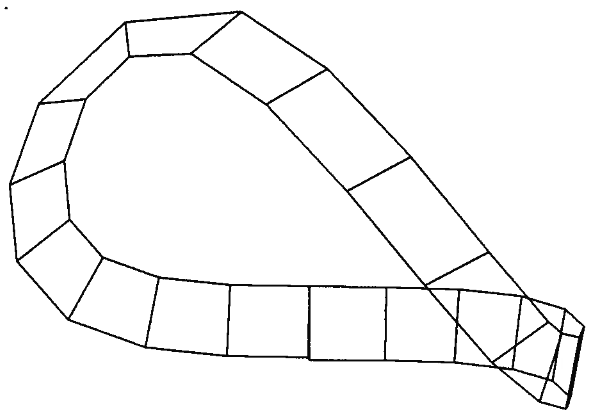


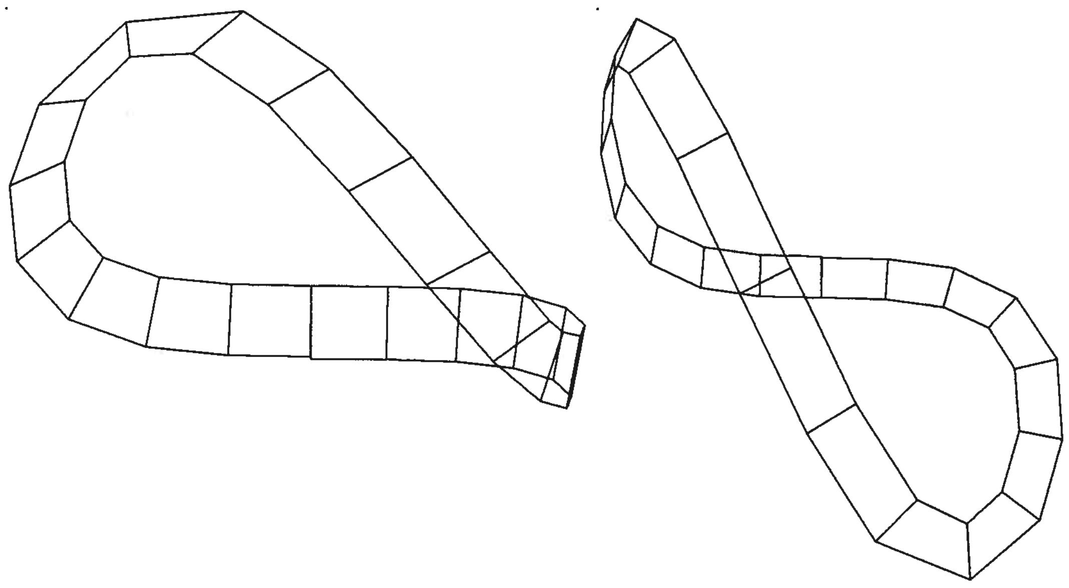
Figure 3.4: Robustness of Modern Approach Showing Mobius Strip

In order to create the ***developable*** mobius strip the very end control vertices were fixed. As mentioned earlier a new concept was introducted. This concept was termed ***mobility*** and is explained in the next section below with figures included to help clarify the explanation.

Controlling Alignment with Mobility

The concept of mobility was developed as a first tool to control the behaviour of the modern approach. It can simply be defined as local weighting of the change in angle with respect to unit motion of each control vertex. Each control vertex has information recorded about it by the function described in equation 3.10. When a mobility value of 1.0, unity, is assigned the class entity which retains the information about a particular vertex has 100% freedom to reposition the location of that particular vertex. In contrast, a mobility of 0.0, indicates that the vertex is not permitted to be moved at all. We can use a range of values for each vertex ranging from 0.0 to 1.0 if we wish to "fine tune" or more accurately control the behaviour of the iterative solving procedure in order to align with the end generator.

The example of the developable mobius strip shown previously and now shown in two views in figure 3.5 was constructed with 6 control vertices. Each control vertex was assigned a mobility. The first and last vertex were given a mobility of 0.0, ie. no movement allowed, and the other four were given full mobility of 1.0, ie. full freedom to be corrected.



(a) Developable mobius strip view 1 (b) Developable mobius strip view 2

Figure 3.5: Mobius Strip

The *developable* mobius strip was one of the first examples where mobility was necessary to create a specific type of shape. The same 6 control vertices used to construct the mobius strip were all initially given mobility of 1.0, and run to see what type of solution would result. The resulting figure 3.6, demonstrates what type of solution results when no constraining is implemented.

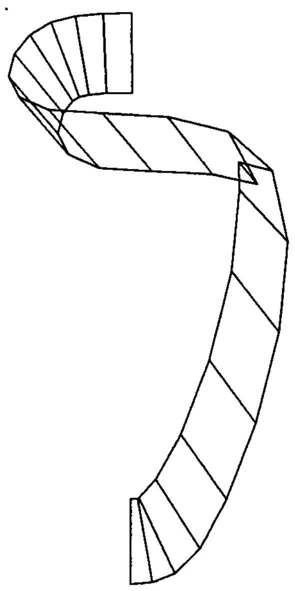


Figure 3.6: Modern Approach Showing Full Mobility of All Points

One should note that figure 3.6 is a valid solution. The end generators do align with the same line that passes through both the start and end generator positions. Clearly one can see that if "hard" constraints are not given, more than one solution can result. This provided us with insight in furthering the design analysis in that more than one solution could result; an entire family of solutions is possible with the single directrix approach unless further constraining is included.

Problems arising when Surface has Small In and Out-of-plane Curvature

In the *modern approach* a problem arises when the surface has small in and out-of-plane curvature , ie. the surface is almost flat. This problem is located where the alignment process takes place. When the accumulated values of the change in angle with respect to motion of the control vertices is very near zero, a subsequent calculation involves dividing the present angle that the end generator makes with the desired end generator. This results in division of a floating point value with one which is very near to zero. If the plate is very close to being flat the computer being used to do the analysis will yield a floating point error due to division by zero.

A tolerance or threshold value was implemented which would make the resulting floating point operation equal to zero if the total in and out-of-plane curvature was less than 1.0×10^{-10}. This was deemed necessary in order to improve the robustness of the algorithm. This results in the *modern approach* to accomodate when the surface is very near flat when initialized. A sample is shown in figure 3.7 where the surface is flat and a corresponding correct solution results.

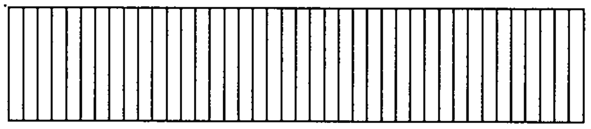


Figure 3.7: Provision Made for When Surface Very Near Flat

3.2 Constraints Defining Modified Conventional Approach (Modified Nolan's Approach) in Order to Create a Normal Directrix

Nolan's approach of matching the cross products between the generator and the tangents to each of the edge curves can be expressed in terms of a differential equation. Nolan's approach can give good approximations of developable surfaces. This approach expressed in differential equation form and with additional constraints could be used as an approximation for creating a close fitting normal directrix.

Before a normal directrix can be computed, it must have information as to where it should be located relative to two space curves that contain a developable surface. Information in the form of modelling the *Conventional Approach* by differential equations lead to the formulation of Normal Directrix Control Vertices. This material is presented with the *Modern Approach Utilizing a Single Normal Directrix* because it is used to derive the normal directrix control vertices.

In figure 3.8 the two outer space curves, referred to as $f(u)$ and $g(v)$, are used in defining the modified conventional approach. In this approach the first two of the three constraints stated by Nolan [17] are exactly the same. The third definition is also used but is modified to include an additional term is formulated in order to determine where along the generator the normal directrix should lie. These steps are now explained in more detail.

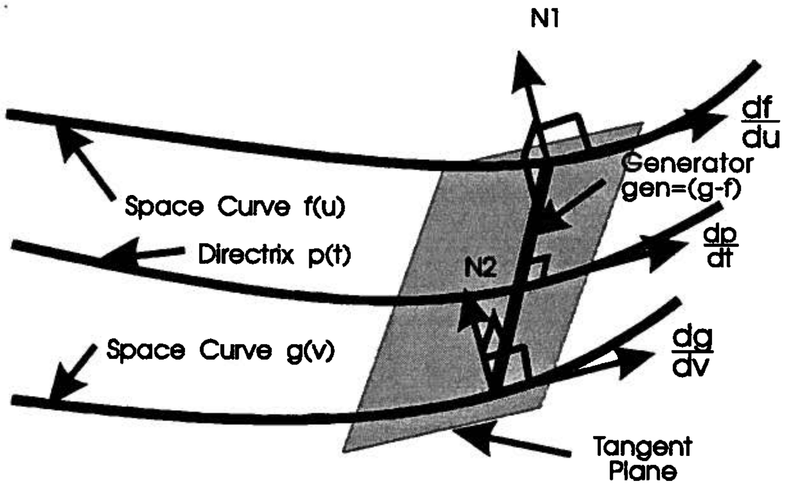


Figure 3.8: Relating Modified Conventional Approach Space Curves and Normal Directrix

In figure 3.8 the first two constraints are graphically shown stating that each space curve must have each tangent vector perpendicular to its normal. Also, the cross-products between the generator and the tangents to each space curve must be parallel to each other. We can express the first two constraints for each space curve as follows:

$$\overrightarrow{n_1} = \overrightarrow{\frac{df}{du}} \times \overrightarrow{gen} \tag{3.13}$$

$$\text{where } \overrightarrow{gen} = (g(v) - f(u)) = (g - f) \tag{3.14}$$

$$\text{and } \overrightarrow{n_2} = \overrightarrow{\frac{dg}{dv}} \times \overrightarrow{gen} \tag{3.15}$$

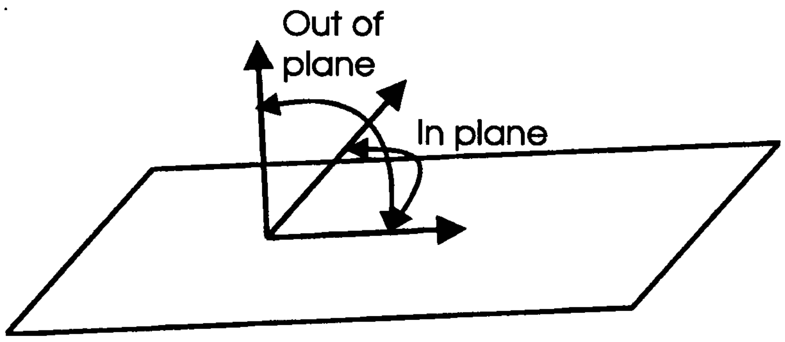


Figure 3.9: In-plane and out-of-plane components

The third relationship is derived from the out-of-plane components and the in-plane components leading to their next respective locations along the space curves. The third relationship is stated in equation 3.16 and is explained here below.

$$\left(\frac{\overrightarrow{n_1} \cdot \overrightarrow{\frac{d^2 f}{du^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_1}) \cdot \overrightarrow{\frac{df}{du}}} \right) \frac{du}{dt} = \left(\frac{\overrightarrow{n_2} \cdot \overrightarrow{\frac{d^2 g}{dv^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_2}) \cdot \overrightarrow{\frac{dg}{dv}}} \right) \frac{dv}{dt} \tag{3.16}$$

The numerator dot product relation in equation 3.16 represents the out-of-plane component. The denominator relation represents the in-plane component. Only magnitudes are needed since direction is constrained to being along each space curve's tangent and normal vectors. At a particular position along the space curve the out-of-plane direction is located along the normal vector and curvature vector. For example, on the space curve f(u), $\overrightarrow{n_1}$ and $\overrightarrow{\frac{d^2f}{du^2}}$ are in the directions of out-of-plane curvature. For in-plane curvature one can relate the resulting cross product of the generator, $\overrightarrow{gen}$, with the normal, $\overrightarrow{n_1}$, to get a vector in the appropriate in-plane direction. The tangent vector of curve f(u), $\overrightarrow{\frac{df}{du}}$, and the vector, $\overrightarrow{gen}$, lie in-plane. From these relations a ratio of just the magnitudes would be simplest to equate with respect to the next parametric calculation along the space curve since the directions are constraints in the formulations. These relations are then equated to two space curves. By equating these relations to two space curves a known developable set of generators (or rulings) can be first approximated. The equating of the ratio of out-of-plane curvature to in-plane direction for two space curves f(u) and g(v) in equaton 3.16 is written as the definition just described. One should note that equation 3.16 is close to the equivalent approach described by Nolan.

We now take into consideration a different number of control vertices for all of the space curves,and an approximated position of where the normal directrix should lie.

We now show the key equations which yield the final result in the modified conventional approach. For a more comprehensive derivation of this refer to appendix G.

Initially we need to set up the three space curve relationships:

$$p(t) \quad = \quad (1-a)f(u) + ag(v) \tag{3.17}$$

From the two space curves, f(u) and g(v), the generator is found as follows:

$$\overrightarrow{gen} = (g - f) \tag{3.18}$$

Equation 3.17 is then differentiated with respect to t in order to establish a differential equation for the normal directrix intersection with the generator. Differentiating yields:

$$\overrightarrow{\frac{dp}{dt}} = (1-a)\overrightarrow{\frac{df}{du}}\frac{du}{dt} + a\overrightarrow{\frac{dg}{dv}}\frac{dv}{dt} + \overrightarrow{gen}\frac{da}{dt} \tag{3.19}$$

Using the definition that, $\overrightarrow{gen} \cdot \overrightarrow{\frac{dp}{dt}} = 0$ and using this on equation 3.19 results in the following:

$$\overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{gen} = (1-a)\overrightarrow{\frac{df}{du}}\frac{du}{dt} \cdot \overrightarrow{gen} + a\overrightarrow{\frac{dg}{dv}}\frac{dv}{dt} \cdot \overrightarrow{gen} + \overrightarrow{gen}\frac{da}{dt} \cdot \overrightarrow{gen} \tag{3.20}$$

The normals of each space curve is also calculated using the following relationships:

$$n_1 = \overrightarrow{\frac{df}{du}} \times \overrightarrow{gen} \tag{3.21}$$

$$n_2 = \overrightarrow{\frac{dg}{dv}} \times \overrightarrow{gen} \tag{3.22}$$

Relating the out-of-plane curvature with the in-plane curvature of the two space curves as mentioned previously we get:

$$\left(\frac{\overrightarrow{n_1} \cdot \overrightarrow{\frac{d^2f}{du^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_1}) \cdot \overrightarrow{\frac{df}{du}}} \right) \frac{du}{dt} = \left(\frac{\overrightarrow{n_2} \cdot \overrightarrow{\frac{d^2g}{dv^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_2}) \cdot \overrightarrow{\frac{dg}{dv}}} \right) \frac{dv}{dt} \tag{3.23}$$

Including the relationship of non-equal number of control vertices for the space curves:

$$\frac{dv}{dt} = \frac{2n_v}{n_t}\left(1.0 - \frac{n_t}{2n_n}\frac{du}{dt}\right) \tag{3.24}$$

Rearranging these relationships to solve for $\frac{du}{dt}$, $\frac{dv}{dt}$ and $\frac{da}{dt}$ gives the following relation:

$$\frac{du}{dt} = \left(\frac{n_t}{2n_u} + \frac{n_t}{2n_v}\left(\frac{\left(\frac{\vec{n_1}\cdot\overrightarrow{\frac{d^2f}{du^2}}}{(\overrightarrow{gen}\times\vec{n_1})\cdot\overrightarrow{\frac{df}{du}}}\right)}{\left(\frac{\vec{n_2}\cdot\overrightarrow{\frac{d^2g}{dv^2}}}{(\overrightarrow{gen}\times\vec{n_2})\cdot\overrightarrow{\frac{dg}{dv}}}\right)}\right)\right)^{-1} \tag{3.25}$$

$$\frac{dv}{dt} = \frac{2n_v}{n_t} - \frac{n_v}{n_u}\frac{du}{dt} \tag{3.26}$$

$$\frac{da}{dt} = -\frac{(1-a)\overrightarrow{\frac{df}{du}}\cdot\overrightarrow{gen}\frac{du}{dt} + a\overrightarrow{\frac{dg}{dv}}\cdot\overrightarrow{gen}\frac{dv}{dt}}{\overrightarrow{gen}\cdot\overrightarrow{gen}} \tag{3.27}$$

3.2.1 Offset of Normal Directrix

One feature which was found useful for the designer to have as a variable to adjust is the initial positional offset of the normal directrix. This relation is shown again here as equation 3.28:

$$p(t) = (1-a)f(u) + ag(v) \tag{3.28}$$

By allowing the initial position offset, a, of the normal directrix to be adjusted the user may be able to further fine tune the surface to desired specifications. If this is not of any concern it defaults to the parametric value, 0.5, which is the midpoint between the two space curves.

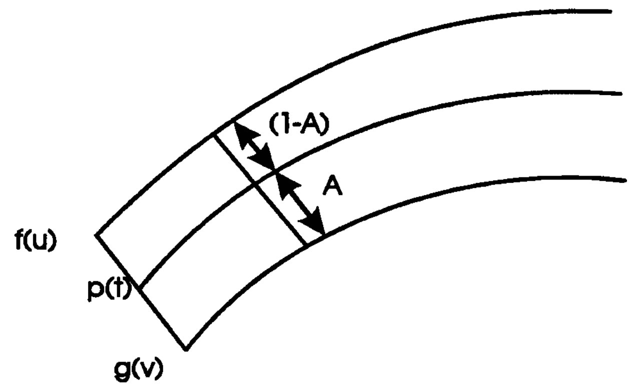


Figure 3.10: Positional Offset of Normal Directrix

3.2.2 Allowment of Different Number of Control Vertices

During the initial design stages one criterion was noted, namely that the number of control vertices for each space curve and the normal directrix may be desired to be different from each other. This was noted and a further relationship was established which allows for different control vertices for any of the curves. The relationship is presented as follows:

n_t = The number of control vertices of the normal directrix.

n_u = The number of control vertices of one space curve.

n_v = The number of control vertices of another space curve.

Given the variables shown above and the two differential parametric index parameters, $\frac{du}{dt}$ and $\frac{dv}{dt}$, a very simple equation can be stated as follows:

$$\frac{1.0}{2n_u}\frac{du}{dt} + \frac{1.0}{2n_v}\frac{dv}{dt} = \frac{1.0}{n_t} \tag{3.29}$$

Equation 3.29 states that each parametric differental variable, $\frac{du}{dt}$ and $\frac{dv}{dt}$, contributes one-half times its total number of control vertices. Ignoring the total number of control vertices for every curve results in the equation being written as:

$$\frac{1}{2} + \frac{1}{2} = 1.0 \tag{3.30}$$

The resulting relationship for the space curves is as follows:

$$\frac{dv}{dt} = \frac{2n_v}{n_t} - \frac{n_v}{n_u}\frac{du}{dt} \tag{3.31}$$

Further reading detailing the derivation can be referred to in G.

3.2.3 Present Problems with Current Solution and Improvisation Implemented

Tests including equation 3.16 proved promising. One problem noted, is shown in Figure 3.11 below when the surface is close to becoming flat. A phenomenon which we termed "fanning" occurs where the out-of- plane component becomes very small relative to the in-plane-component yielding undesirable characteristics. This is shown below in Figure 3.11. One can see from Figure 3.11 that intuitively if a plate is close to becoming flat, the generators should lie nearly perpendicular to the tangent vectors of each space curve, resulting in square flat plates for a flat surface. This problem was noted and addressed by incorporating an out-of-plane tolerance which would default to parallel lines when the surface fell below the user-specified tolerance. This is a temporary solution and further research is being directed to address this problem in future work.

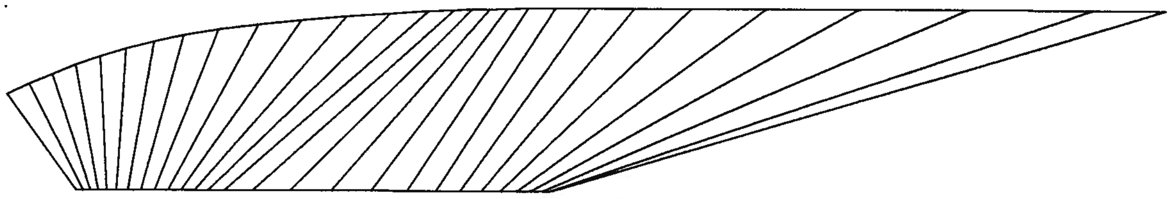


Figure 3.11: Fanning Occurring When Developable Surface Nearly Flat

For a more detailed example showing where this phenomenon occurs in practice see the examples in Chapter 7, entitled "Demonstration Examples". The three examples cited were a simple conical developable, an arctic series fishing vessel, and the UBC series fishing vessel.

3.3 Utilizing Modified Conventional Approach to Approximate a Single Directrix

If the normal directrix is to be described by a spline with n control points, then the values for those n points can be computed to yield a spline which lies close to the normal directrix derived from the differential equations. From the equations 3.25, 3.26 and 3.27, the intersection location along the generators can be calculated. The desired number of control vertices for the normal directrix is still to be determined and may vary. The number of desired normal directrix control vertices is specified by the variable n_t.

From the above mentioned relationships another differential equation is needed in order to solve for the *location* of the normal directrix control vertices . The relationship was found to be in the form of finding the minimum for an equation relating the normal directrix in terms of an error. This relation relates $p(t)$, the known spline, and $r(t)$, the spline control vertices we wish to solve for, is as follows:

$$\text{error} = \int_0^{N-1} |p(t) - r(t)|^2 \, dt \tag{3.32}$$

We then differentiate equation 3.32 with respect to the control vertices and solve for the relation to determine a minimum. This is shown as follows:

$$\begin{aligned}\frac{\partial \text{error}}{\partial C_i} &= 2\int_0^{N-1} \frac{dp}{dC_i}\,(p(t) - r(t))\,dt && (3.33)\\ &= \sum_{j=0}^{N-2}\int_0^1 \frac{dp(j+s)}{dC_i}\,(p(j+s) - r(j+s))\,dS && (3.34)\\ &= 0 && (3.35)\end{aligned}$$

Details of the analysis can be seen in Appendix F. The initial relation showing the equation is as follows:

$$\frac{\partial error}{\partial C_i} = \sum_{j=0}^{N-2}\int_0^1 \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix}^T [A]^T \begin{bmatrix} s^3 \\ s^2 \\ s \\ 1 \end{bmatrix} \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} C_{j-1} \\ C_j \\ C_{j+1} \\ C_{j+2} \end{bmatrix} dS \quad (3.36)$$

$$\begin{bmatrix} C_{j-1} \\ C_j \\ C_{j+1} \\ C_{j+2} \end{bmatrix} = \left[\sum_{j=0}^{N-2} \Phi(i,j)\right]^{-1} \sum_{j=0}^{N-2}\int_0^1 r(j+s) \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix} dS \quad (3.37)$$

3.3.1 Equations Yielding Approximated Normal Directrix

Collecting equations 3.25, 3.26, 3.27, equation 3.37 and presenting them below shows how all of the variables are sequentially coupled and integrated using the class ODE (Ordinary Differential Equation) solver.

$$\frac{du}{dt} = \left(\frac{n_t}{2n_u} + \frac{n_t}{2n_v} \left(\frac{\left(\frac{\overrightarrow{n_1} \cdot \overrightarrow{\frac{d^2 f}{du^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_1}) \cdot \overrightarrow{\frac{df}{du}}} \right)}{\left(\frac{\overrightarrow{n_2} \cdot \overrightarrow{\frac{d^2 g}{dv^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_2}) \cdot \overrightarrow{\frac{dg}{dv}}} \right)} \right) \right)^{-1} \tag{3.38}$$

$$\frac{dv}{dt} = \frac{2n_v}{n_t} - \frac{n_v}{n_u}\frac{du}{dt} \tag{3.39}$$

$$\frac{da}{dt} = -\frac{(1-a)\overrightarrow{\frac{df}{du}} \cdot \overrightarrow{gen}\frac{du}{dt} + a\overrightarrow{\frac{dg}{dv}} \cdot \overrightarrow{gen}\frac{dv}{dt}}{\overrightarrow{gen} \cdot \overrightarrow{gen}} \tag{3.40}$$

$$\begin{bmatrix} C_{j-1} \\ C_j \\ C_{j+1} \\ C_{j+2} \end{bmatrix} = \left[\sum_{j=0}^{N-2} \Phi(i,j) \right]^{-1} \sum_{j=0}^{N-2} \int_0^1 r(j+s) \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix} ds \tag{3.41}$$

In the computer program all of the terms are collected as the ODE solver integrates from 0.0 to N-1. The algorithm tested concluded to be quite robust. No singularities would occur. There are still, a few problems that have to be addressed which still affect the solution.

3.3.2 Results and Present Problems

Combining the equations above and integrating using the ODE class to generate the generators for the developable surface and the control vertices for the normal directrix yields the developable conical-like shape shown as an example in Figure 3.12. The directrix shown in the figure has actually two directrices. Both happen to coincide exactly for this example. One directrix follows $\frac{dp}{dt}$ exactly as the surface is being integrated. The other directrix results from solving for the normal directrix control vertices.

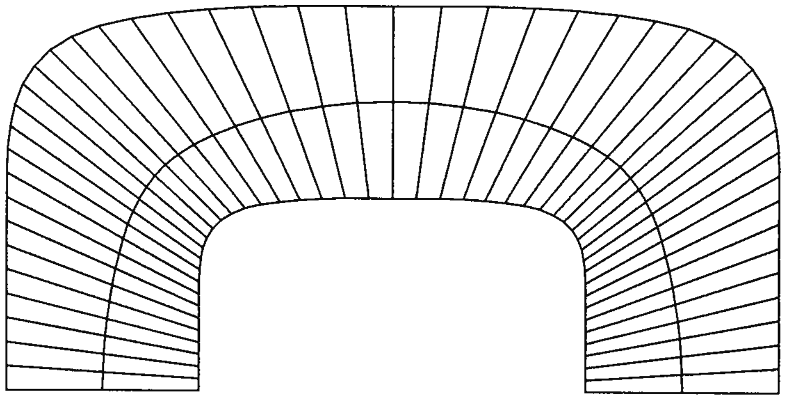


Figure 3.12: Conic-Like Section with Flatness Tolerance Specified at 0.001

The next figure shows the same conical-like shape without a flatness tolerance specified. The problem of fanning is apparent at both ends. Note how the two directrices are different. The one which looks invariant is the control vertices which have been solved for and have an additional constraint of having to be perpendicuar to the end generators. The directrix which appears to follow the surface more exactly is the one which relates the apparent tangent with the current generators.

The following figures display the modified conventional approach used to approximate

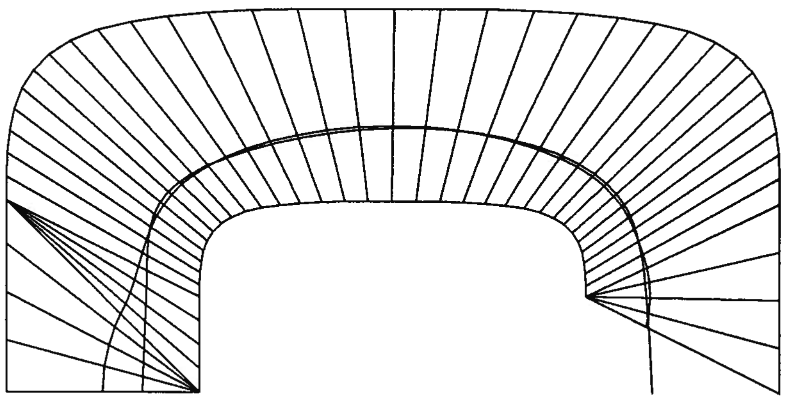


Figure 3.13: Conic-Like Section With No Flatness Tolerance Specified

the normal directrix control vertices which is then used by the modern approach. The examples cited are hull sections of the UBC Series fishing vessel.

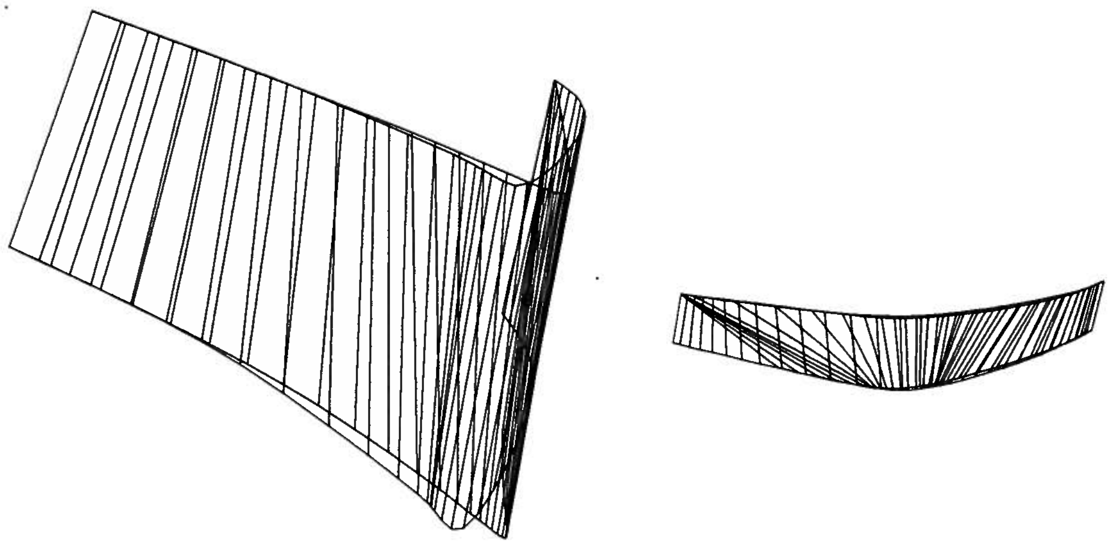
In figure 3.14 is an overlayed closer comparison of the two methods with the current configurations.

In Figure 3.15 both versions appear to give better results. The *modified approach* has the tolerance set higher , 0.01, and displays the *fanning* problem near the stern being reduced.

In figure 3.16 is an overlayed closer comparison of the two methods with the current configurations.

In Figure 3.17 both versions appear to give even better results. The *modified approach* has the tolerance set higher , 0.1, and displays the *fanning* problem near the stern is almost eliminated.

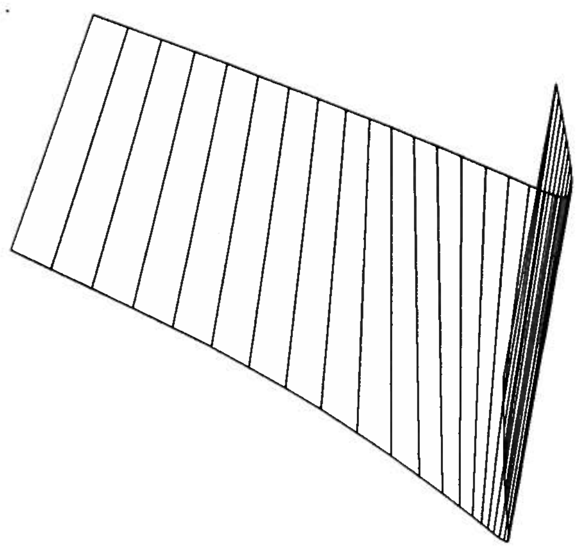
In figure 3.18 an overlayed closer comparison of the two methods with the current configurations is shown.

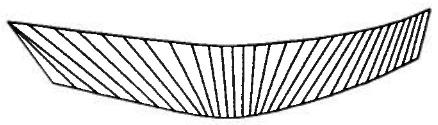


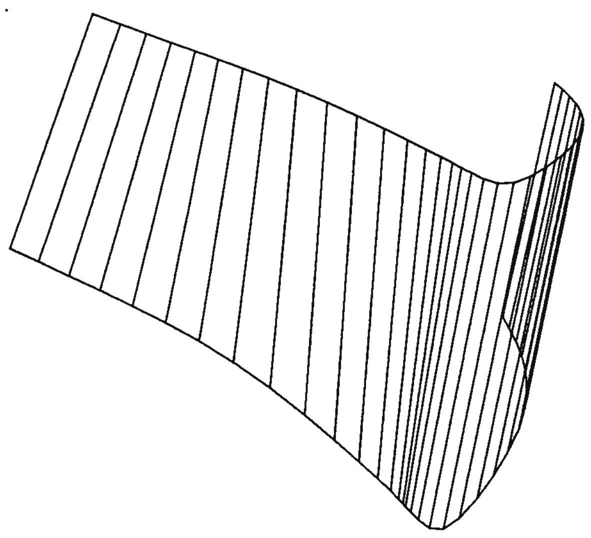
(a) UBC Series Surface (main deck chine and chine 1) body plan

(b) UBC Series Surface (main deck chine and chine 2) profile

Figure 3.14: Developable Surface Procedure Comparison, tolerance 0.001


(a) UBC Series Surface (main deck chine and chine 1) modified approach body plan


(b) UBC Series Surface (main deck chine and chine 2) modified approach profile


(c) UBC Series Surface (main deck chine and chine 1) non-optimized body plan

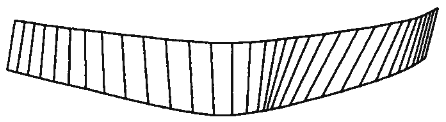
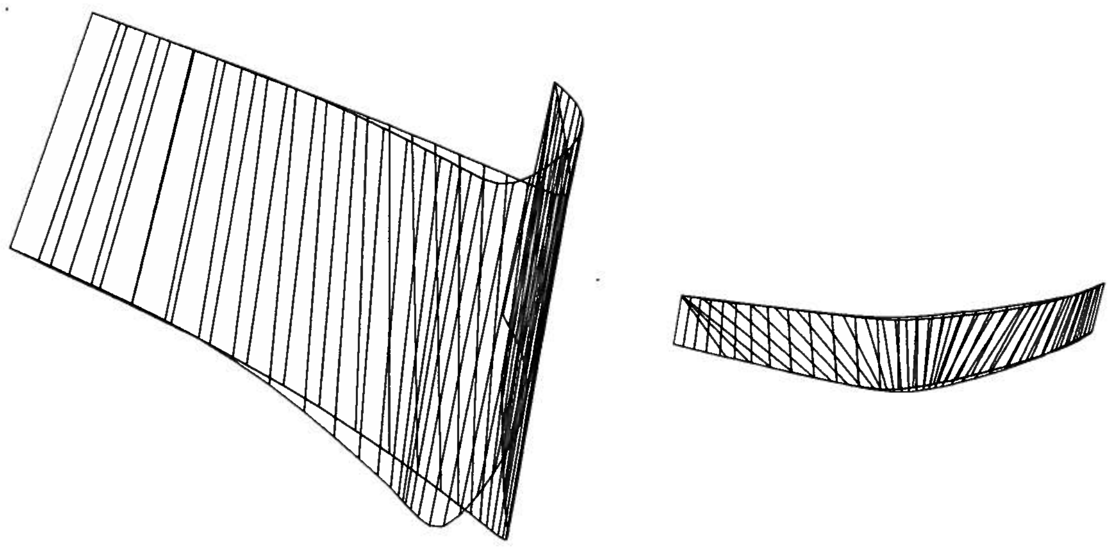

(d) UBC Series Surface (main deck chine and chine 2) non-optimized profile

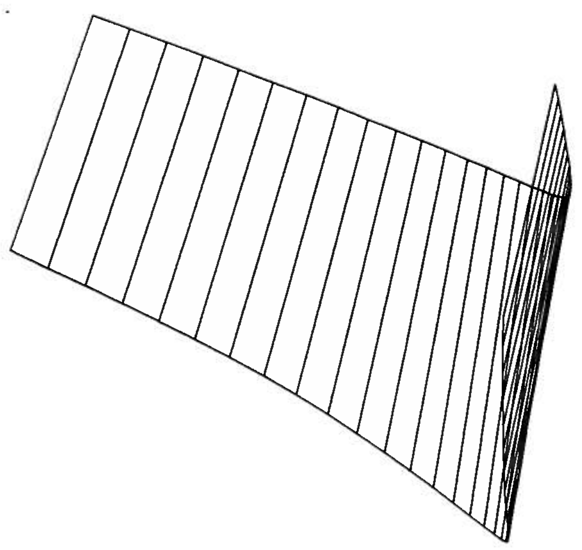
Figure 3.15: Developable Surface success, tolerance 0.01



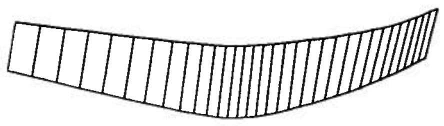
(a) UBC Series Surface (main deck chine and chine 1) body plan

(b) UBC Series Surface (main deck chine and chine 2) profile

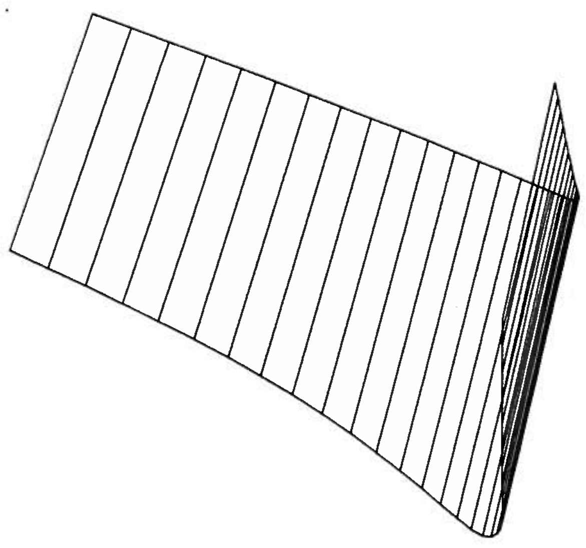
Figure 3.16: Developable Surface Procedure Comparison, tolerance 0.01



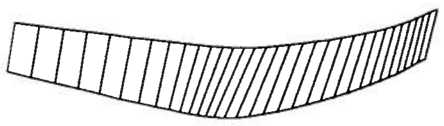
(a) UBC Series Surface (main deck chine and chine 1) modified approach body plan



(b) UBC Series Surface (main deck chine and chine 2) modified approach profile

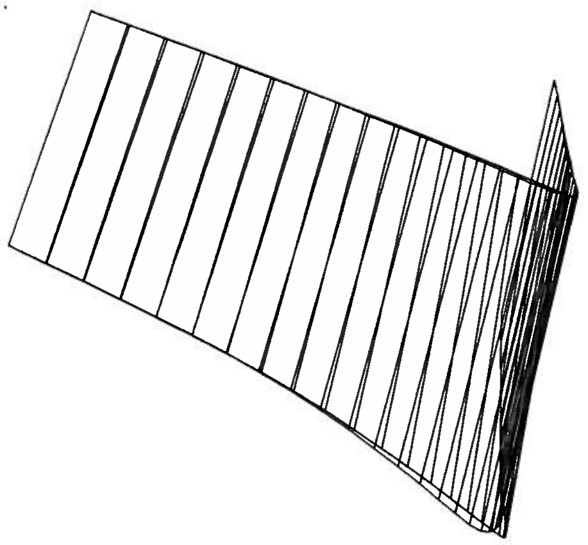


(c) UBC Series Surface (main deck chine and chine 1) non-optimized body plan

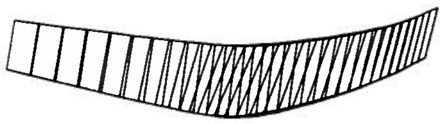


(d) UBC Series Surface (main deck chine and chine 2) non-optimized profile

Figure 3.17: Developable Surface success, tolerance 0.1



(a) UBC Series Surface (main deck chine and chine 1) body plan



(b) UBC Series Surface (main deck chine and chine 2) profile

Figure 3.18: Developable Surface Procedure Comparison, tolerance 0.1

Chapter 4

Optimization of a Normal Directrix

4.1 Objective

Once the *modern approach* was deemed fairly stable the next criterion were then approached and listed as follows:

- To better match a developable surface defined by a normal directrix to a pair of space curves.
- To create an optimization function which is the mean square distance from each space curve to the nearest point on the surface.
- To derive an equation to evaluate the mean square distance.
- To utilize a robust non-linear optimization technique to minimize the integrated mean square distance.

4.2 Optimization Function

Calculation of the normal distance between a point on a space curve and the developable surface, see Figure 4.1, can be obtained by three definitive relationships. This will be described and derivation of the first form of solution will be shown below:

- The first relation is that the normal, $\overrightarrow{n(t)}$, of the surface is equal to the cross product

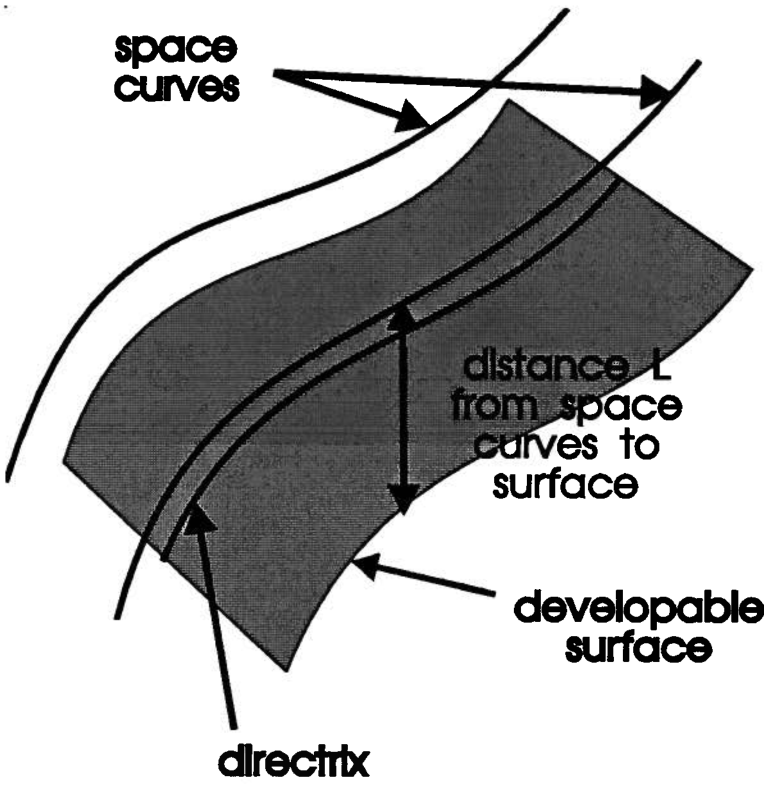


Figure 4.1: Relating the Distance Between Space Curves and Surface

of the tangent, $\overrightarrow{\frac{dp}{dt}}$ and the generator, $\overrightarrow{g(t)}$.

$$\overrightarrow{n(t)} = \overrightarrow{\frac{dp}{dt}} \times \overrightarrow{g(t)}$$

- The second relation states that any position on the surface can be calculated, $q(t, s)$ by moving along the directrix, $p(t)$ and then along the generator, $\overrightarrow{g(t)}$:

$$q(t, s) = p(t) + s\overrightarrow{g(t)}$$

- The third relation builds on the previous statement by further stating that the closest point from the surface to a point in space would be a specified distance, l, normal to the surface along the normal vector, $\overrightarrow{n(t)}$:

$$r(u) = q(t,s) + \overrightarrow{ln(t)}$$

Substituting the second relation into the third yields:

$$r(u) = p(t) + \overrightarrow{sg(t)} + \overrightarrow{ln(t)}$$

The third relation above is the combined total of the previous two and can now be differentiated with respect to the independent variable t in order to create a differential equation which can solved.

Differentiating yields the following:

$$\frac{\overrightarrow{dr}}{du}\frac{du}{dt} - \overrightarrow{g(t)}\frac{ds}{dt} - \overrightarrow{n(t)}\frac{dl}{dt} = \frac{\overrightarrow{dp}}{dt} + s\frac{\overrightarrow{dg}}{dt} + l\frac{\overrightarrow{dn}}{dt} \tag{4.1}$$

We also have two other known relations which can be substituted, namely:

$$\frac{\overrightarrow{dg}}{dt} = -\frac{(\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{g})}{\frac{\overrightarrow{dp}}{dt} \cdot \frac{\overrightarrow{dp}}{dt}}\frac{\overrightarrow{dp}}{dt} \tag{4.2}$$

$$\frac{\overrightarrow{dn}}{dt} = \frac{\overrightarrow{d^2p}}{dt^2} \times \overrightarrow{g} + \frac{\overrightarrow{dp}}{dt} \times \frac{\overrightarrow{dg}}{dt} \tag{4.3}$$

From these three relations we can solve for three equations and three unknowns, with the unknowns being the following dependent differentials:

- The differential dependent parametric position along a space curve

$$\frac{du}{dt}$$

- The differential scalar dependent position along the generator $\overrightarrow{g(t)}$

$$\frac{ds}{dt}$$

- The differential scalar dependent position along the normal $\overrightarrow{n(t)}$

$$\frac{dl}{dt}$$

We then solve for the following relation:

$$\begin{bmatrix} \frac{dr}{du}_x & -g_x & -n_x \\ \frac{dr}{du}_y & -g_y & -n_y \\ \frac{dr}{du}_z & -g_z & -n_z \end{bmatrix} \begin{bmatrix} \frac{du}{dt} \\ \frac{ds}{dt} \\ \frac{dl}{dt} \end{bmatrix} = \begin{bmatrix} \frac{dp}{dt}_x + s\frac{dg}{dt}_x + l\frac{dn}{dt}_x \\ \frac{dp}{dt}_y + s\frac{dg}{dt}_y + l\frac{dn}{dt}_y \\ \frac{dp}{dt}_z + s\frac{dg}{dt}_z + l\frac{dn}{dt}_z \end{bmatrix} \tag{4.4}$$

The second form involves using the three definitive relations and continuing to find relations yielding direct solutions without solving simultaneous equations.

A more detailed derivation for both of the forms can be found in Appendix H.

Starting with the differential relation:

$$\frac{\overrightarrow{dr}}{du}\frac{du}{dt} - \vec{g}\frac{ds}{dt} - \vec{n}\frac{dl}{dt} = \frac{\overrightarrow{dp}}{dt} + s\frac{\overrightarrow{dg}}{dt} + l\frac{\overrightarrow{dn}}{dt} \tag{4.5}$$

Taking the dot product with vector g: (4.6)

$$\left(\frac{\overrightarrow{dr}}{du} \cdot \vec{g}\right)\frac{du}{dt} - \frac{ds}{dt}\vec{g} \cdot \vec{g} - \frac{dl}{dt}\vec{n} \cdot \vec{g} = \frac{\overrightarrow{dp}}{dt} \cdot \vec{g} + s\frac{\overrightarrow{dg}}{dt} \cdot \vec{g} + l\frac{\overrightarrow{dn}}{dt} \cdot \vec{g} \tag{4.7}$$

This simplifies to the following: (4.8)

$$\frac{ds}{dt} = \left(\frac{\overrightarrow{dr}}{du} \cdot \vec{g}\right)\frac{du}{dt} \tag{4.9}$$

$$\text{Similarly, dot product with n,} \tag{4.10}$$

$$\text{and simplifying yields:} \tag{4.11}$$

$$\frac{dl}{dt} = \left(\frac{\overrightarrow{dr}}{du} \cdot \overrightarrow{n}\right)\frac{du}{dt} \tag{4.12}$$

$$\text{Similarly, dot product with} \frac{\overrightarrow{dp}}{dt} \tag{4.13}$$

$$\left(\frac{\overrightarrow{dr}}{du} \cdot \frac{\overrightarrow{dp}}{dt}\right)\frac{du}{dt} = \frac{\overrightarrow{dp}}{dt} \cdot \frac{\overrightarrow{dp}}{dt} + s\frac{\overrightarrow{dg}}{dt} \cdot \frac{\overrightarrow{dp}}{dt} + l\frac{\overrightarrow{dn}}{dt} \cdot \frac{\overrightarrow{dp}}{dt} \tag{4.14}$$

$$\text{Simplifying yields:} \tag{4.15}$$

$$\frac{du}{dt} = \frac{\left(\frac{\overrightarrow{dp}}{dt} \cdot \frac{\overrightarrow{dp}}{dt} - \frac{\overrightarrow{d^2p}}{dt^2} \cdot (s\overrightarrow{g} + l\overrightarrow{n})\right)}{\left(\frac{\overrightarrow{dr}}{du} \cdot \frac{\overrightarrow{dp}}{dt}\right)} \tag{4.16}$$

4.2.1 Integral Chosen to Minimize and Simplex Parameters Used

The key parameter chosen to calculate, the distance from a point on a space curve to the closest point on a surface, has to be further related to a function which describes the entire surface. Calculation of a least squares approximation for the area under the curves was chosen. One can refer to figure 4.2 for a visual representation of the area under the curves. By collecting a history of distances between the space curves and the surface, one can calculate the total area. Ideally, this would be zero. An equation in this form would work with the *Downhill Simplex Method of Multidimension.*

The distance along the curve, the arc length, was calculated as follows: The arclength, S, was calculated for both curves in calculations.

$$S = \int_a^b \left|\frac{\overrightarrow{dr}}{dt}\right| dt \tag{4.17}$$

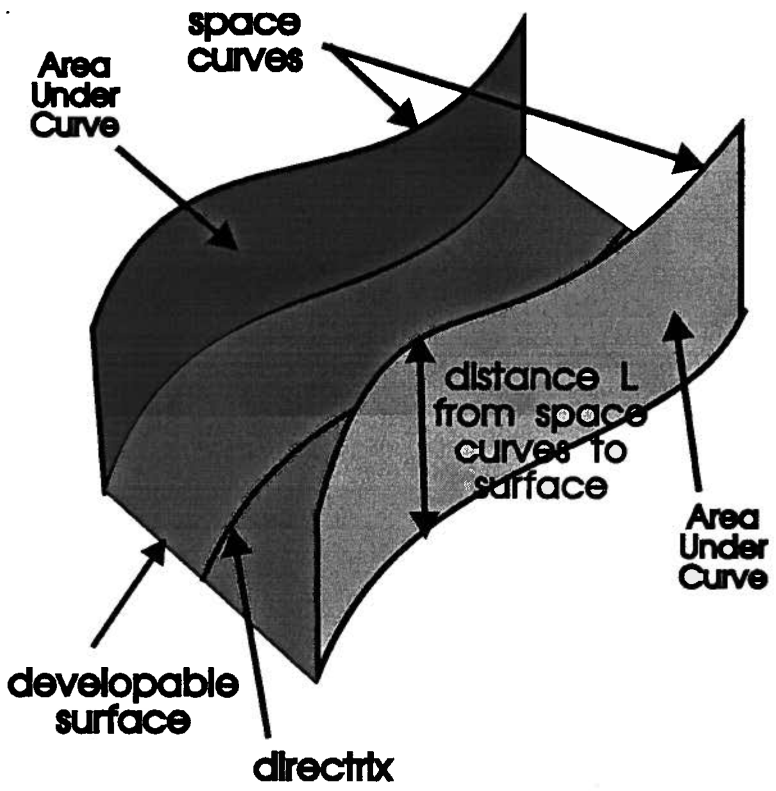


Figure 4.2: Area calculated under space curves

$$
\begin{aligned}
&= \int_a^b |v(t)|\, dt && (4.18)\\
&= \int_a^b v(t) dt && (4.19)\\
\frac{dS}{dt} &= \frac{d}{dt}\int_a^t v(\tau) d\tau = v(t) && (4.20)\\
&= \frac{d}{dt}\int_a^t |v(\tau)|\, d\tau = |v(t)| && (4.21)\\
dS &= v(t)dt && (4.22)
\end{aligned}
$$

Let h_1 and h_2 be arc lengths for the two space curves r_1 and r_2 respectively. The resulting least squares type approximation for the total area would be:

$$Area = \int \left[l_1^2 \left| \frac{dh_1}{dt} \right| + l_2^2 \left| \frac{dh_2}{dt} \right| \right] dt \qquad (4.23)$$

From equation 4.23 the area between the space curves and the developable surface is the function to be minimized. This is accomplished by using the equations for $\frac{dl}{dt}, \frac{du}{dt}, \frac{ds}{dt} and \frac{dq}{dt}$ for each space curve. These totalled together sum to be the degrees of freedom for the non-linear optimization method model. The non-linear optimization method used is the Downhill Simplex method. This is discussed later in this chapter.

By solving all the differential equations' independent and dependent variables from the start of the developable surface to its end, the non-linear optimization function can attempt to minimize equation 4.23.

Another key point is that the dependent and independent variables sum to give ten degrees of freedom. The nonlinear optimization method also needs ten different solutions to start with. This problem was solved by moving one control vertex degree of freedom from the directrix by unit length. From these initialization procedures the Downhill Simplex method was able to start solving these equations.

4.2.2 Present Problems

Both approaches stated above describing solving for the dependent variables will produce solutions for surfaces which do not have major changes in the rate of change of the curvature and the direction of the curvature. If the rate of change of the out-of - plane curvature with or without contribution by the in-plane curvature is very large, and changing in direction, the *modern approach* may have difficulty in the alignment process when trying to conform to the surface. Cases may arise, as shown in figure 4.3, where a corresponding normal on the surface which should match a position on the space curve will not occur. This may be due to the constraint that both the space curves and the surface are of finite length and the corresponding normal on the surface will point to an out-of-bounds position. One other possibility is that a localized change in curvature orients itself midway during the iteration process and another normal of infinite length occurs.

The first method of matching the *modern approach* to two space curves can also fail at a prior stage when calculating the inverse of the matrix. Reasons for a non - solution are generally for the same reasons.

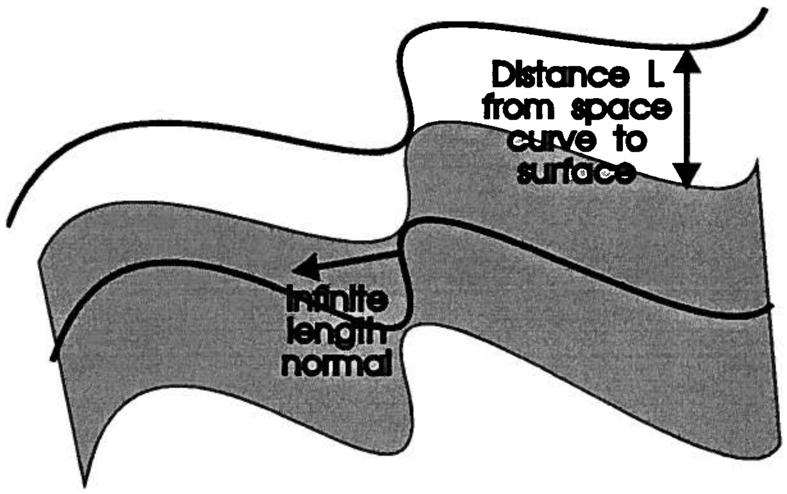


Figure 4.3: Possible inherent failure due to certain geometry

4.3 Downhill Simplex Method

Many non-linear optimization methods as cited by Jacoby [14] would be possible to implement, but, a more suitable and generally more stable and robust method, the Simplex method [16], was chosen. A major reason for this preference was the description cited in Press [20], which briefly described its multidimensional capablities.

4.3.1 Explanation of The Downhill Simplex Method

A brief description of the *Downhill Simplex Method in Multidimensions* is presented here to help clarify its use. A simplex is the geometrical figure consisting, in N dimensions, of N + 1 points (or vertices) and all their interconnecting line segments, polygonal faces, etc. [20] In three dimensions, it is a tetrahedron, as can be seen in figure 4.4.

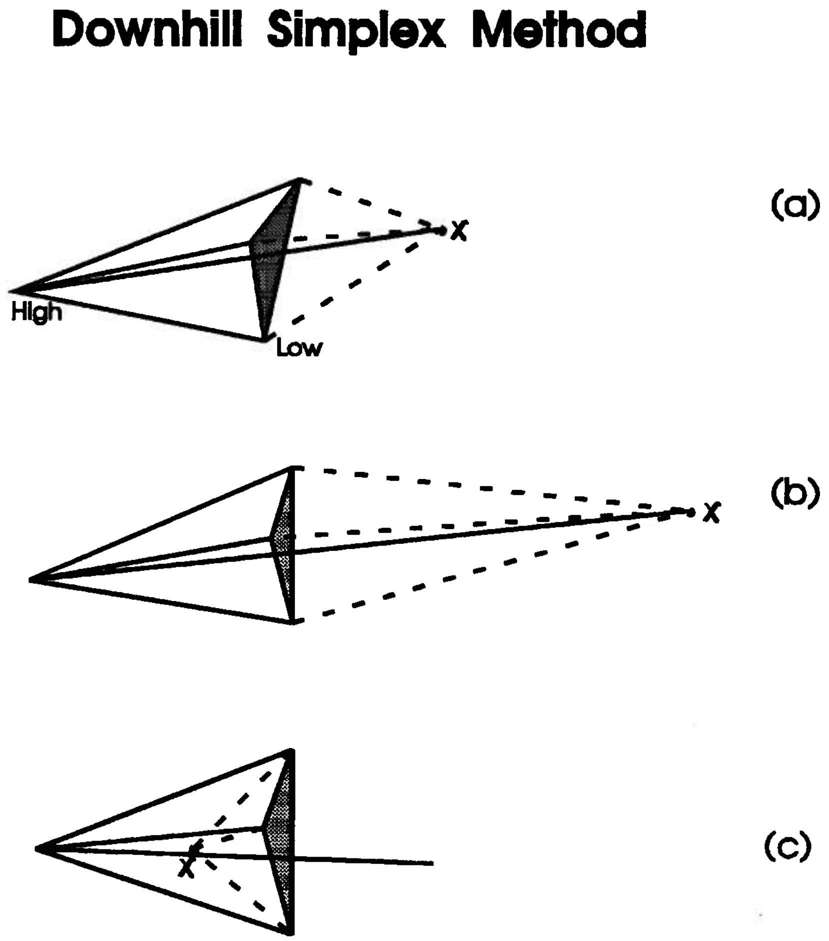


Figure 4.4: Analogy of a Simplex For The Downhill Simplex Method

The downhill simplex method has three parameters which define the moving behaviour of the simplex. These parameters are mentioned here because changes in their values may be necessary. The three parameters are defined by Nelder [16] as α, β, and γ. These parameters correspond to: reflection, contraction and expansion. The reflection parameter, α, is a positive constant, which is a scalar multiplication constant which mirrors the point through the simplex. The contraction parameter, β, is a constant which values lie between 0 and 1. It is a ratio of the distance of the point relative to the simplex centroid. The expansion coefficient, γ, is greater than unity and it is a ratio of the current point to the centroid with a point along the line joining the point to the centroid. For a more detailed explanation refer to Nelder [16]. For detailed coded form of the *Downhill Simplex Method*, programming and tests performed, refer to Appendix L.

4.3.2 Results and Present Problems

As stated previously, surfaces which resemble close to developable surfaces to begin with will be more likely to have a solution that closely matches the original space curves. Below are two examples which demonstrate both simple and difficult solutions encountered.

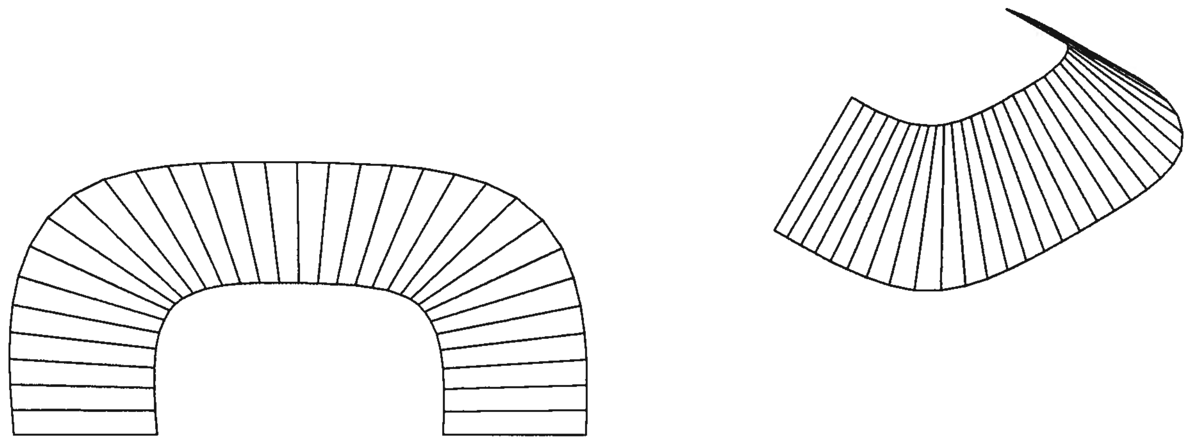
4.4 Examples

A few examples are shown in this section to help explain the various phenomenon of the results of the theory created this far in the research.

4.4.1 Example of Single Surface with Conical Properties

The first example demonstrates how the non-linear optimization can successfully solve a free-form surface with conical properties. Shown below in figure 4.5 are two views of a conical like surface.

Note that due to the symmetry of the object in figure 4.5 the generators also follow symmetrically along the surface.



(a) Conical-like developable surface view 1 (b) Conical-like developable surface view 2

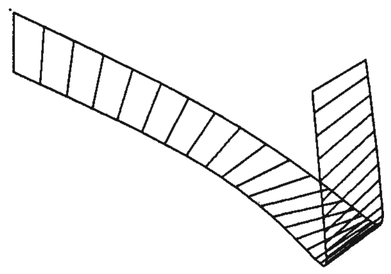
Figure 4.5: Conical-like developable surface

4.4.2 Example of UBC Series Demonstrating Present Problems

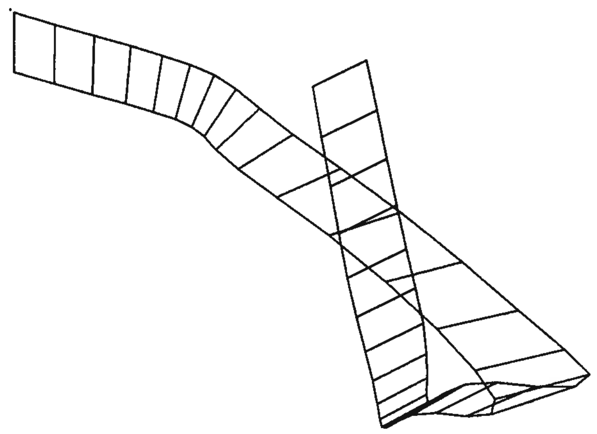
The UBC Series vessel was posed as the immediate goal to make into a developable ship series. In figure 4.6 the surface tested was using the 1st and 2nd chine as the space curves.

As seen in figure 4.6 the optimized version had difficulty with the adjustments due to the in-and-out-of-plane curvature that results from this type of surface.

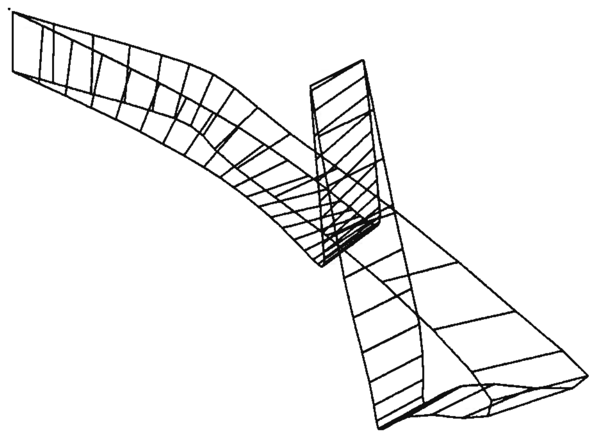
Referring to figure 4.7 both versions appear to give moderately successful results. The *modified approach* has the tolerance set low , 0.001, and displays the *fanning* problem near the stern.



(a) UBC Series Surface (Chine 1 and 2) non-optimized

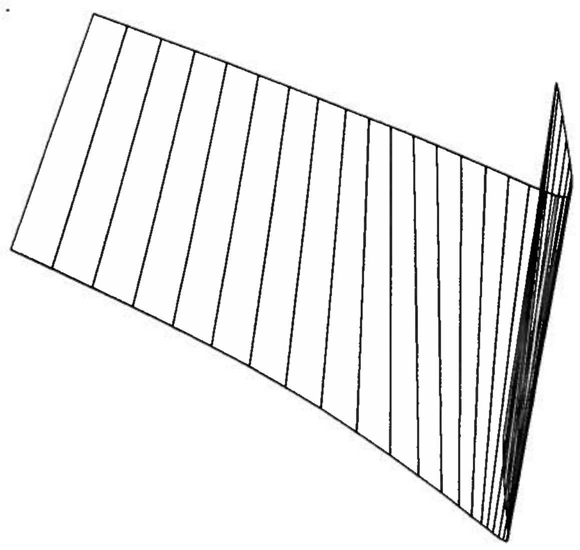


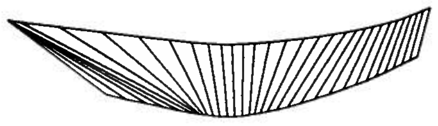
(b) UBC Series Surface (Chine 1 and 2) optimized

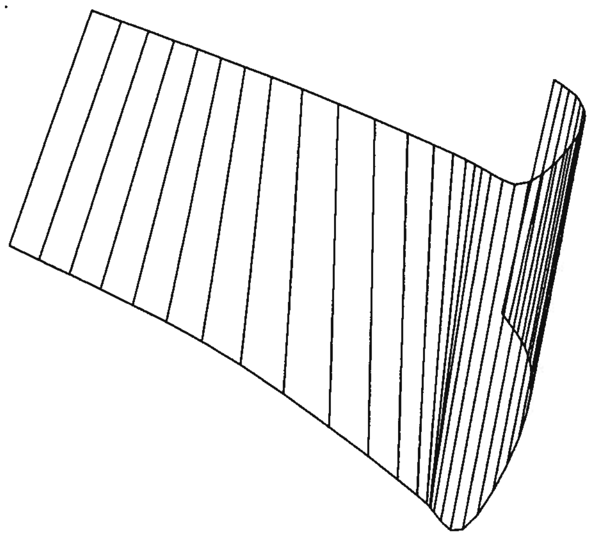


(c) UBC Series Surface (Chine 1 and 2) comparison

Figure 4.6: Developable Surface Failure


(a) UBC Series Surface (main and chine 1) modified approach body plan


(b) UBC Series Surface (main and chine 2) modified approach profile


(c) UBC Series Surface (main and chine 1) optimized body plan

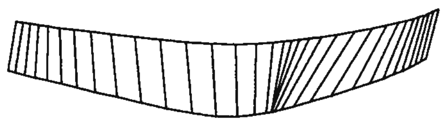

(d) UBC Series Surface (main and chine 2) optimized profile

Figure 4.7: Developable Surface success, tolerance 0.001

Chapter 5

Intersection of Developable Surfaces and Flat Plate Layout

Once developable surfaces have been created, it is necessary to define the edges of those surfaces by intersection with adjacent surfaces.

An additional consideration is once the bounded developable surface has been created a *flat plate layout* is necessary. A derivation of the solution is also presented here.

5.1 Intersection of Developable Surfaces

Once the developable surfaces have been created there is the requirement that they must intersect without gaps or spaces between them. Shown below is the vector representation for initial conditions of the surfaces and their orientation relative to each other. In figure 5.1 we see the various vectors that are dependent upon each other. We can show the vectors dependence on each other as follows:

Figure 5.1 presents the independent parametric variable t. The two dependent parametric variables are u and v. The independent space curve is $p(t)$; one dependent space curve which is a function of u is $f(u)$; the other dependent space curve is $g(v)$, which is a function of v. The curves $p(t)$, $f(u)$ and $g(v)$ are normal directrices representing the developable surface of interest, and the adjacent surfaces on each side. The intersection curves joining two developable surfaces are r_1 which is relating $f(u)$ and $p(t)$ and the other intersection curve r_2 relates $g(v)$ and $p(t)$.

A detailed derivation of the equation used to calculate the intersection of the developable surfaces are located in Appendix I

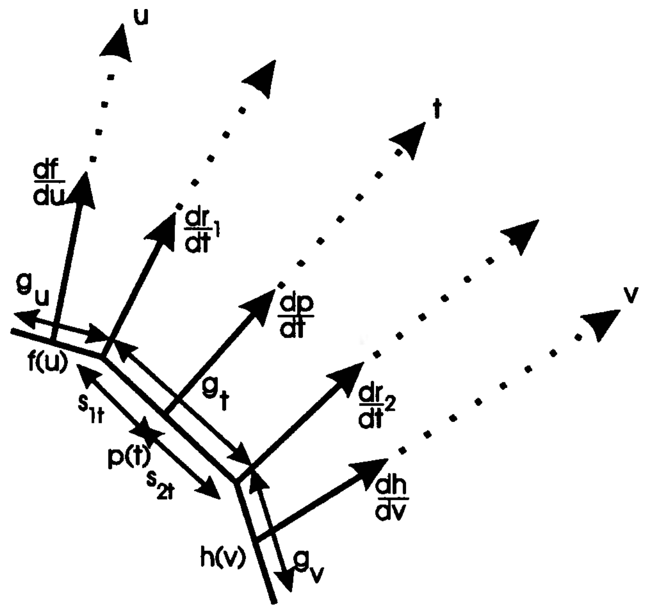


Figure 5.1: Intersection of Three Developable Surfaces

5.1.1 Derived Equations Yielding Intersections of Surfaces

Derivation of the equations used to calculate the intersection of the developable surfaces are as follows:

$$r_1 = p_t + s_{1t}\overrightarrow{g_t} \tag{5.1}$$

$$\overrightarrow{\frac{dr_1}{dt}} = \overrightarrow{\frac{dp}{dt}} + s_{1t}\overrightarrow{\frac{dg_t}{dt}} + \frac{ds_{1t}}{dt}\overrightarrow{g_t} \tag{5.2}$$

$$\frac{ds_{1t}}{dt} = -\frac{\overrightarrow{\frac{dp}{dt}} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right) + s_{1t}\overrightarrow{\frac{dg_t}{dt}} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right)}{\overrightarrow{g_t} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right)} \tag{5.3}$$

$$\overrightarrow{\frac{dg_t}{dt}} = -\left(\frac{\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{g_t}}{\overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{\frac{dp}{dt}}}\right)\overrightarrow{\frac{dp}{dt}} \tag{5.4}$$

$$\overrightarrow{\frac{dg_u}{du}} = -\left(\frac{\overrightarrow{\frac{d^2f}{du^2}} \cdot \overrightarrow{g_u}}{\overrightarrow{\frac{df_u}{dt}} \cdot \overrightarrow{\frac{df_u}{du}}}\right)\overrightarrow{\frac{df_u}{du}} \tag{5.5}$$

$$r_1 = f_u + s_{2u}g_u \tag{5.6}$$

$$\overrightarrow{\frac{dr_1}{dt}} = \overrightarrow{\frac{dr_1}{du}} \cdot \frac{du}{dt} \tag{5.7}$$

$$\overrightarrow{\frac{dr_1}{dt}} = \frac{du}{dt}\left(\overrightarrow{\frac{df_u}{du}} + s_{2u}\overrightarrow{\frac{dg_u}{du}} + \overrightarrow{g_u}\frac{ds_u}{du}\right) \tag{5.8}$$

But, (5.9)

$$r_1 = f_u + s_{2u}g_u, \ \textit{and} \tag{5.10}$$

$$s_{2u}g_u = r_1 - f_u \tag{5.11}$$

$$\textit{and}, \ r_1 = p_t + s_{1t}g_t \tag{5.12}$$

Substituting gives, (5.13)

$$\overrightarrow{\frac{df_1}{dt}} \cdot \overrightarrow{\frac{df_u}{du}} = \frac{du}{dt}\left(\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} - s_{2u}\overrightarrow{\frac{d^2f_u}{du^2}} \cdot \overrightarrow{g_u}\right) \tag{5.14}$$

$$\frac{du}{dt} = \frac{\left(\overrightarrow{\frac{dp}{dt}} + s_{1t}\overrightarrow{\frac{dg}{dt}} + \frac{ds_{1t}}{dt}\overrightarrow{g_t}\right) \cdot \overrightarrow{\frac{df_u}{du}}}{\left(\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} - \overrightarrow{\frac{d^2 f_u}{du^2}} \cdot (p_t + s_{1t}\overrightarrow{g_t} - f_u)\right)} \tag{5.15}$$

$$\overrightarrow{\frac{dg_u}{dt}} = \overrightarrow{\frac{dg_u}{du}}\frac{du}{dt} \tag{5.16}$$

$$r_2 = p_t + s_{2t}\overrightarrow{g_t} \tag{5.17}$$

$$\overrightarrow{\frac{dr_2}{dt}} = \overrightarrow{\frac{dp}{dt}} + s_{2t}\overrightarrow{\frac{dg_t}{dt}} + \frac{ds_{2t}}{dt}\overrightarrow{g_t} \tag{5.18}$$

$$\frac{ds_{2t}}{dt} = -\frac{\overrightarrow{\frac{dp}{dt}} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right) + s_{2t}\overrightarrow{\frac{dg_t}{dt}} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right)}{\overrightarrow{g_v} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right)} \tag{5.19}$$

$$\overrightarrow{\frac{dg_t}{dt}} = -\left(\frac{\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{g_t}}{\overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{\frac{dp}{dt}}}\right)\overrightarrow{\frac{dp}{dt}} \tag{5.20}$$

$$\overrightarrow{\frac{dg_v}{dv}} = -\left(\frac{\overrightarrow{\frac{d^2h}{dv^2}} \cdot \overrightarrow{g_v}}{\overrightarrow{\frac{dh}{dv}} \cdot \overrightarrow{\frac{dh}{dv}}}\right)\overrightarrow{\frac{dh}{dv}} \tag{5.21}$$

$$r_2 = h_v + s_{1v}\overrightarrow{g_v} \tag{5.22}$$

$$\overrightarrow{\frac{dr_2}{dt}} = \overrightarrow{\frac{dr_2}{dv}}\frac{dv}{dt} \tag{5.23}$$

$$\overrightarrow{\frac{dr_2}{dt}} = \frac{dv}{dt}\left(\overrightarrow{\frac{dh_v}{dv}} + s_{1v}\overrightarrow{\frac{dg_v}{dv}} + \overrightarrow{g_v}\frac{ds_{1v}}{dv}\right) \tag{5.24}$$

$$r_2 = h_v + s_{1v}\overrightarrow{g_v} \tag{5.25}$$

$$s_{1v}\overrightarrow{g_v} = r_2 - h_v \tag{5.26}$$

$$r_2 = p_t + s_{2t}\overrightarrow{g_t} \tag{5.27}$$

$$\overrightarrow{\frac{dr_2}{dt}} \cdot \overrightarrow{\frac{dh_v}{dv}} = \frac{dv}{dt}\left(\overrightarrow{\frac{dh_v}{dv}} \cdot \overrightarrow{\frac{dh_v}{dv}} - s_{1v}\overrightarrow{\frac{d^2h_v}{dv^2}} \cdot \overrightarrow{g_v}\right) \tag{5.28}$$

$$\frac{dv}{dt} = \frac{\left(\overrightarrow{\frac{dp}{dt}} + s_{2t}\overrightarrow{\frac{dg_t}{dt}} + \frac{ds_{2t}}{dt}\overrightarrow{g_t}\right) \cdot \overrightarrow{\frac{dh_v}{dv}}}{\left(\overrightarrow{\frac{dh_v}{dv}} \cdot \overrightarrow{\frac{dh_v}{dv}} - \overrightarrow{\frac{d^2h_v}{dv^2}} \cdot (p_t + s_{2t}\overrightarrow{g_t} - h_v)\right)} \tag{5.29}$$

$$\overrightarrow{\frac{dg_v}{dt}} = \overrightarrow{\frac{dg_v}{dv}}\frac{dv}{dt} \tag{5.30}$$

5.1.2 Results and Present Problems

The intersection of developable surfaces was tested and proved to be fairly stable when well behaved surfaces were created.

Conical Type Surface Tested

An example shown in figure 5.2 displays very tightly intersecting surface without any problems. The intersection algorithm needs N + 2 surfaces when designing for N surfaces. In the next subsection the algorithm is shown using what we term *phantom* surfaces .

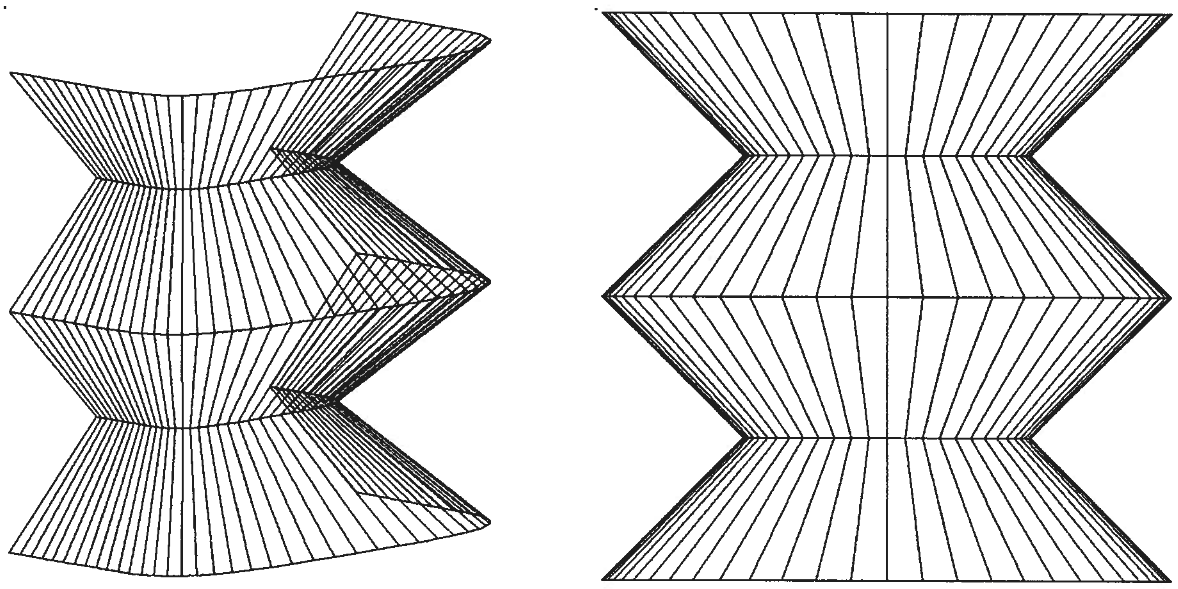

(a) Conical-type surfaces view point (1,1,1) (b) Conical-type surfaces front view

Figure 5.2: Conical-type Surfaces Intersections

Criterion of Phantom Surfaces

The criterion of phantom surfaces was introduced to provide a secure mechanism for preventing interconnecting surfaces from having "cracks" or "leaks". The conical-type surfaces presented in figure 5.2 show surfaces which have been created using the *modified approach*. These all appear to be developable but only the inner two have been confirmed as developable using the modern approach. The two outside surfaces were used as *phantom* surfaces in order to provide an intersecting edge that the inner two could use. Looking at the figure 5.3c) and figure 5.3d), these two surfaces use the *modern approach* and yield aligned surfaces with the intersection algorithms. The cracks will be more evident when looking at the *ARCTIC* series of fishing vessel or the *UBC* series vessel. Figure 5.3c) and figure 5.3d) are shown to display the *phantom* surfaces

5.2 Flat Plate Layout

In order to convey the information of a developable surface into a useful form, the flat-plate-layout was created so the user can essentially use this output as a template.

5.2.1 Derived Equations giving Flat Plates Dimensions and Interplate Angles

Once a developable surface has been created, the shape and position of the consecutive segments that make up the developable surface need to be known. Below is the derivation of the algorithm used to provide the information of the shape of the segments in the x-y plane and the angle and rate of change of bending of the segments relative to each previous one. For a more detailed explanation of the derivation one should refer to Appendix J

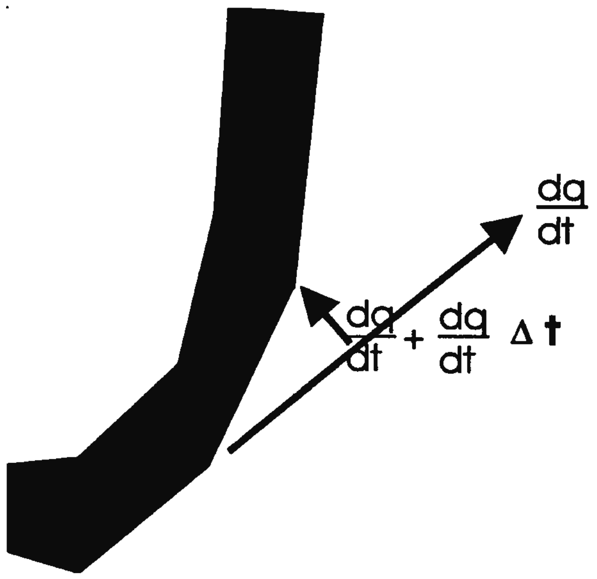


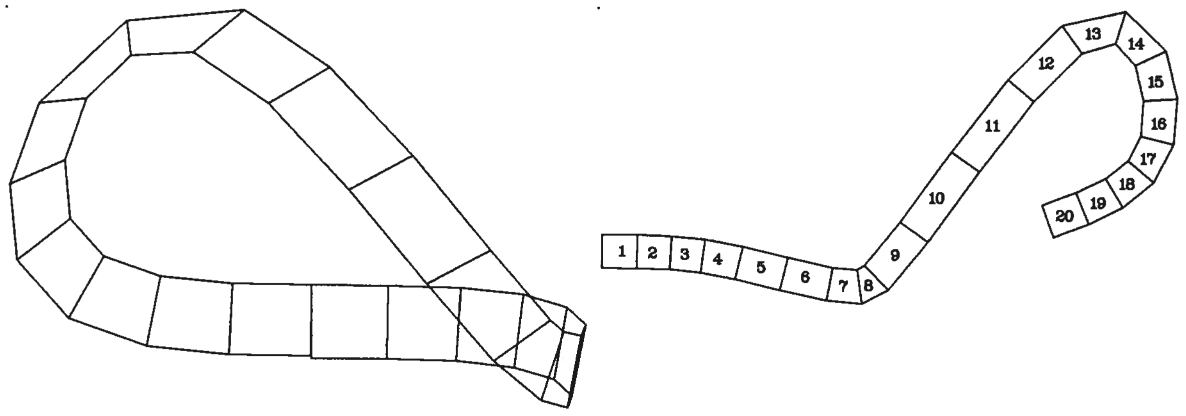
Figure 5.4: Flat plate layout derivation

The resulting differential equation is used to find the new tangent and corresponding point on the x-y plane where the next point of the plate is located:

$$\frac{d^2q}{dt^2} = \frac{\left(\frac{d^2p}{dt^2} \cdot \frac{dp}{dt}\right)\frac{dq}{dt}}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)} + \left(\frac{d^2p}{dt^2} \cdot g_{3d}\right) g_{2d} \tag{5.31}$$

5.2.2 Format of Output

The example of the flat plate layout for the developable mobius strip can be seen in figure 5.5



(a) 3-D view of developable mobius strip (b) Corresponding flat plate layout

Figure 5.5: Developable surface and flat plate layout

The flat plate layout corresponding to the numbered plates in figure 5.5 has data listed in table 5.1.

Plate no.	Parametric value	out of plane	angle (deg)
0	0.00	4.641389	0.975138
1	0.25	13.924170	8.451800
2	0.50	23.206951	19.508427
3	0.75	32.489716	27.889669
4	1.00	32.115269	21.843246
5	1.25	29.928911	13.160947
6	1.50	38.130260	19.531147
7	1.75	56.255440	60.359379
8	2.00	55.513470	41.122555
9	2.25	34.989170	8.168997
10	2.50	21.341158	3.404501
11	2.75	14.529543	2.682558
12	3.00	12.068573	6.007172
13	3.25	9.672798	13.691539
14	3.50	10.510071	9.681314
15	3.75	15.940498	9.440858
16	4.00	18.051476	19.695953
17	4.25	12.895594	21.226387
18	4.50	7.738149	11.441552
19	4.75	2.579468	4.393825
20	5.00	-0.016314	0.398113

Table 5.1: Flat plate data

Chapter 6

Choice of Computer Language and CAD Program

When this thesis work was initiated, one of the objectives was to initially create generic tools in a very portable modular language and then apply them to a well established open ended CAD package that is used throughout industry. At the present stage of the work both objectives seem to have been met.

6.1 Computer Language Chosen

The choice of which computer language to program was based on a number of design constraints. The language chosen should be flexible in case the method of design had to be altered. The three designs considered were:

- Top-down structured design
- Data-driven design
- Object-oriented design

Of these three types of design, data-driven design was initially dropped because of the nature of the design tool. Because of the financial constraints on the project and the software languages available on the platform chosen, the top-down structured design was initiated. This was then taylored into a modular top-down structured design. After the release of a popularly supported compiler for the platform was chosen, the design was then switched to an object-oriented design. The language initially chosen was ANSI C, best supported at the initial time of development by Microsoft for the PC platform.

Later, after an initial attempt with Walter Bright's, Zortech C++, the switch was made to Borland's C++, which proved to have excellent support tools, a good development environment and better technical support.

6.1.1 OOP - Object Oriented Programming

When it was realized that a true portable object-oriented language was available, the design was taylored using influences from OOP, object oriented programming, OOD, object oriented design, and OOA, object oriented analysis. First a definition of each is in order:

- OOP *Object-Oriented Programming*

  Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships [5].

- OOD *Object-Oriented Design*

  Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design [5].

- OOA *Object-Oriented Analysis*

  Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain [5]

6.1.2 Portability to Different Platforms

One major consideration when doing software development for an application is the standardization and portability of the code being developed. Up until a few years ago, one of the most standardized and portable languages was ANSI C. Now with more features and better type checking, ANSI C++ appears to be the successor.

Porting some of the sample computer code mentioned below to different compilers and to different computer platforms proved to be a very simple and straight forward task. Very little problems were encountered. The sample compilers tested were Borland C++ 3.1 and Zortech C++ 3.0. The platforms tested were DOS based PC platforms and the SUN SPARC UNIX platform. No major problems were encountered.

6.1.3 C++: Classes and Features

The language chosen to best implement the work being done was the C++ language. This language proved to be very much more than just a superset to the C language. Many of the class features, when used in its entirety, better shape the programming environment into a true object oriented program and design. A little explanation is presented in the next sections as to how the tools were used to better inform the reader as to the approach taken.

Class vector

```
#include <iostream.h>

   #ifndef _VECTOR_
   #define _VECTOR_

   enum direction { X, Y, Z };

   class matrix;
class vector {
int n;
static int default_length;
float *element;
public:
vector(){n=default_length;element=new float[n];};
vector(int dim) {n=dim;element=new float[n];};
vector(int dim, float x);
vector( const vector& v );
~vector();
static void set_default(int n){default_length=n;};
static void set_default(){default_length=3;};
float& operator[]( int i) {return element[i];};
float operator[](int i) const {return element[i];};
friend int size(const vector& v) { return v.n;};
vector& operator=(float x);
vector& operator=(const vector& v);
vector& operator=(const matrix& a);
friend vector operator+(const vector& a, const vector& b);
vector& operator+=( const vector& a);
friend vector operator-(const vector& a, const vector& b);
vector& operator-=(const vector& a);
friend vector operator*(const vector& v, float a);
friend vector operator*(float a, const vector& v);
friend float operator*(const vector& a, const vector& b); // dot product operator
friend vector operator*(const matrix& a, const vector& b);
vector& operator*=(float a);
vector& operator*=(const vector& a); // element-by-element multiplication
friend vector operator/(const vector& v, float a);
friend istream& operator>>(istream&, vector&);
friend ostream& operator<<(ostream&, const vector&);
};
#endif
```

The class *vector* is the name of a class which enables the user to use N dimensional vectors for calculations. A C++ feature, termed *operator overloading* enables the user to write equations almost exactly as one would write them by hand.

Included below is the *vector.h* class header file which shows how the class is initially designed. Some explanation is in order at this stage to clarify to the reader these basic features which are used throughout the development of other classes used in developing the various design tools.

Looking at the listing of the class *vector.h*, the compiler directives, #ifndef _VECTOR_ and #define _VECTOR_ are compiler directives which will not enable *redeclarations*. Below is an optional *enum direction* { *X, Y, Z* }; which enables the user to declare *X* as a *0*, *Y* as *1* and *Z* as *2*.

The next line *class matrix;* declares the matrix class before class *vector.h* can be declared, thereby, enabling *class matrix* to be used inside the *vector.h* class.

On the next line below is the class declaration *class vector* {, which assigns a class name. The next line, since it does not have any *member access control* stated as *private:*, *protected:*, or *public:*, then the members of a class declared with the keyword *class* are *private:* by default.

The next line, whose *member access control* is *public:* shows some very interesting features of *C++*. The first *constructor* , depicted by *vector()(n=3; element=new double[n]);*, which shows a default size, *n*, for the class to allocate memory. *vector()* is an operation which is empty of a *type*. This case is where the *type* is hidden and *some mechanism must be provided for a user to initialize variables of that* type. Located on the next line is, *vector(int dim)n=dim; element=new double[n]*, which is a constructor of type *double*. Moving down one line, *vector(const vector & v)*, is a copy constructor. "A *copy constructor* for a class *vector* is a constructor that can be called to copy an object of class *vector*; that is, one that can be called with a single argument of type *vector*. A *copy constructor* is generated only if no *copy constructor* is declared". The next line, *ṽector() delete element;;*, is contains what termed a destructor. A destructor automatically deallocates dynamic memory when a class goes out of scope. Moving to the next line contains one possible case of operator overloading. The line, *double& operator[](int i)return element[i];;*, enables the programmer to use the [] operator with the *vector* class. For example, the following code fragment would look like:

```
main()
{
    vector x;

    x[0] = 0.0;
    x[1] = 1.0;
    x[2] = 2.0;

    cout<<x[0] <<" "<< x[1] << " " << x[2] <<'\n';

}
```

There is also another form of the operator [] which implements the named *const.* There is also the following code fragment which resides inside of *class vector.* It is in the following form, *float operator[](int i) const{return element[i];};.* This is necessary since the compiler complains when several operations are done successively and invoke temporary storage variables. For example:

```
    vector x,y,z,a,b,c,d;

    ...x,y,z,a,b,c,d ...get assigned values
```

```
...then
x= (y*z)*a*(b*c)*d;
cout<<x;
```

The above list of code shows how the operator [] is used as well as the << stream operator is overloaded to standard output for the class *vector.* Syntactically this code is much simpler and far more legible to the reader. This code is simple and easy to follow thereby leaving the details of implementation to the compiler.

Another note one should make is that the declarations of *int n;* and *double *element;* are **private** inside class *vector.* Therfore, any queries of these variables must be done by a member function within class*vector.* So, for example, in order to find the size *n* of a declared class of type *vector*, one has to invoke either *friend int size(vector & v)return v.n;;* or *int size()return n;;* in order to access items from the *private* section of the class.

The next three lines of code depict three types of operator overloading one could encounter when developing classes. The first operator, *vector& operator=(vector& v);*, equates two classes of type vector. this member function also utilizes the **this* pointer, which is a special reserved pointer used in C++. The details of how the information is passed is left for the reader to investigate. The second operator, *vector& operator=(const matrix& a);*, equates two objects of class type vector and matrix. This enables an equating of classes of different types. The next example operator,*friend vector operator*(const vector& a,const vector& b);*, requires a friend to be declared. This code is optional since the same feature could be invoked by the following, *vector& operator*(vector& a);*.

Class matrix

In the listing below, the *matrix.h* class header file implements the *vector.h* class.

```
#include <vector.h>
#include <iostream.h>

#ifndef _MATRIX_
#define _MATRIX_

class matrix {
int nr, nc;
float **element;
public:
matrix(int rows, int columns );
matrix(const matrix& a);
matrix(const vector& a);
~matrix();
matrix& operator=(float x);
matrix& operator=(const matrix&);
float* operator[](int i) {return element[i];};
const float* operator[](int i) const {return element[i];};
vector row(int i);
friend int n_rows(const matrix& a){return a.nr;};
friend int n_columns(const matrix& a){return a.nc;};
vector column(int i);
friend matrix exp(const matrix& a);
friend matrix element_mult(const matrix& a, const matrix& b);
friend vector diagonal(const matrix& a);
friend matrix operator+(const matrix& a, const matrix &b);
friend matrix operator-(const matrix& a, const matrix& b);
friend matrix operator*(const matrix& a, const matrix& b);
friend vector operator*(const matrix& a, const vector& b);
friend matrix operator*(const matrix& a, float b);
friend matrix operator*(float a, const matrix& b) {return b*a;};
friend matrix transpose(const matrix& a);
friend matrix solve(const matrix& a, const matrix& b, float *det =NULL);
friend matrix identity(int);
friend matrix inverse(const matrix& a, float* det=NULL);
friend istream& operator>>(istream&, matrix&);
```

```
    friend ostream& operator<<(ostream&, const matrix&);
    friend int operator<=(const matrix&, const matrix&);
    friend matrix abs(const matrix&);
    };

    #endif
```

One can see the similarity between *class vector* and *class matrix* and how they can interact.

Class Curve and Surface

```
#include "matrix.h"
    #include <fstream.hpp>

    #ifndef _CURVE_
    #define _CURVE_

    class Curve{
       int n;
       char splinetype;
       vector *point;
       static matrix BB;
    public:
       void sptype(char what){splinetype=what;};
       char typeofspline(){return splinetype;};
       int size(){return n;};
       Curve(){n=5; splinetype= '1'; point = new vector[5];};
       Curve(int num){n=num; splinetype='1'; point = new vector[n];};
       Curve(Curve& a);
       ~Curve(){delete[n] point;};
       void realoc(int num);
       vector& operator[](int i){return point[i];};
       Curve& operator=(Curve& c);
       vector operator()(double t);
       vector tangent(double t);
       vector curvature(double t);
       double deriv_2c(double t, int i);
       double deriv_c(double t, int i);
    static void initbasis(char sptype='2', double b1=1.0, double b2=0.0);
    static matrix BBB;
};
#endif

#ifndef _SURF_
#define _SURF_

class Surface{
private:
    Curve *leftedge, *rightedge;
    Curve *directrix;
    Surface *leftsurf, *rightsurf;
public:
    Surface( Curve *leftedge, Curve *rightedge, Surface *leftsurf,
       Surface *rightsurf,int ncontrolpnts=5,double step=7.0,
       double outp=0.05,int surfno=1);
    void Rightsinit(Surface *rights);
    void Leftsinit(Surface *lefts);
    double Optimize(double tolerance=0.2, int itermax = 250);
    void Develop(void);
    void Developt(void);
    void Developt1(void);
    void Draw_3d(int genno=50,double step=7.0,int surfno=1);
    void Draw_33d(int genno=50,double step=7.0,int surfno=1);
    void Draw_3id(int genno=50,double step=7.0,int surfno=1);
    void Draw_2d(int genno=50,double steop=7.0,int surfno=1);
    ~Surface();
    Surface(Surface& s);
    Surface& operator=(Surface& s);
};
#endif

#ifndef _FILELISTS_
#define _FILELISTS_
#include <fstream.hpp>

class Filelisto{
public:
      ofstream file;
      Filelisto *next;
      ofstream& operator[](int n);
   };

   class Filelisti{
   public:
      ifstream file;
      Filelisti *next;
      ifstream& operator[](int n);
   };
   #endif
```

The following classes, *Curve and Surface* also implement *vector* and *Curve.* These sort of implementations were able to make the code more legible and apply the testing of *Developable Surface* modules much earlier in the design cycle.

Class ODE

```
#ifndef _ODE_
   #define _ODE_

   #include "vector.h"

   class dynamic_system {
   private:
   double time, step_size;
   vector state;
   vector error_scale;
   vector (*derivative)(double, vector&);
   public:
   dynamic_system(vector (*)(double, vector&), double start_time,
                     vector& initial_state, vector *error_scale=0);
   double& when();
   void reset();
   double& operator[](int i);
   vector operator()(double t);
   vector step(double delta);
```

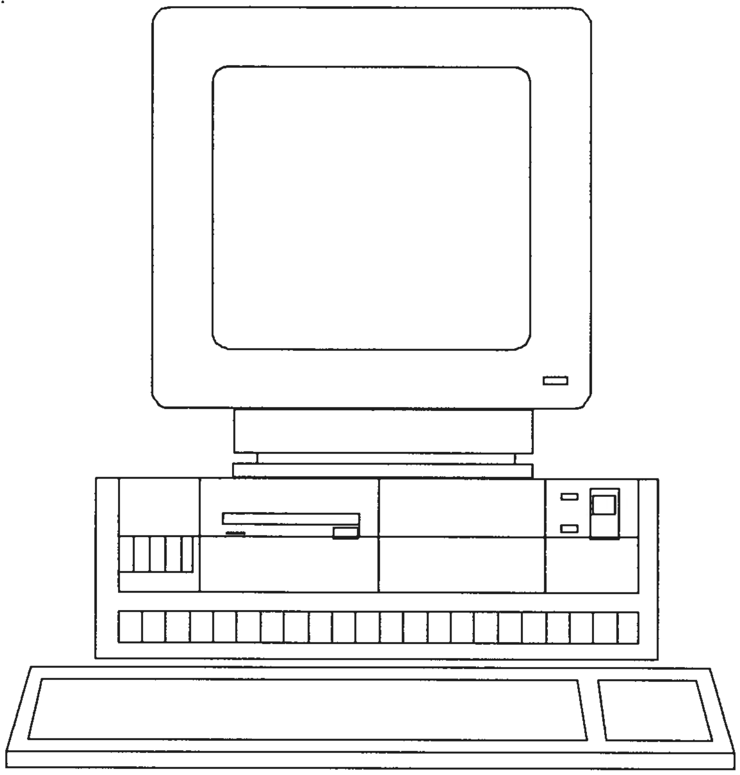


Figure 6.1: Computer Platform Selected Initially

```
};

void rk(double t0, double& del_t, double t1, vector& x,
        vector (*f)(double t, vector& x), vector& err_scale);

#endif
```

The above code class *ode*, ordinary differential equation solver, also shows another useful combination of class building to contribute to a library of useful class tools.

6.2 Computer Platform Selected

6.2.1 PC 386/486 Environment

The development of the theory for this thesis and the validation via programming was implemented on a PC. The PC, personal computer, platform was decided upon because it is the most popular and inexpensive platform in use for almost every application today. The PC is primarily an opened ended architecture in which a wide variety of hardware can be interfaced. With PCs becoming faster and less expensive every year most of the speed constraints of the thesis programs were relaxed. At the present state of the thesis research and design, the programs execute fast enough to become interactive. Commercial viability should be considered.

6.2.2 Computer Language Selection in this Platform

At the start of this thesis the computer language chosen was the C programming language. This language proved to be sufficient for initial implementations. Further development of computer languages, such as C++, proved to offer much more sophisticated features and modularity of code that would offer more efficient development cycles. This proved correct and the theory, code, test, development cycle time was greatly reduced.

Many other features of the C+++ language also proved invaluable towards implementing the theory. One of the most used features of the C++ language was the use of *operator overloading*. This enabled vector calculus equations to be written in code in almost the same convention one would use when writing by hand. Other features which were discussed above all contributed to more legible code and as a collection of useful tools which other applications would ensue.

6.3 CAD Program Selected

During the course of development of the theory a decision was made as to whether or not writing a computer platform independent CAD package should be undertaken. After assessing the existing CAD packages used on several computer platforms, the conclusion was that one would be wiser to develop the special code to an existing CAD package. The CAD package chosen was assessed as the most popular and efficient to develop upon. This CAD package was AutoCAD. During the course of development of the theory, ongoing assessment was done on AutoCAD's market position and development tools. This proved to be benefitial to this project as well as it reflected upon us how strong a commercial product this was in the CAD market as well as their desire to move to other hardware platforms.

AutoCAD was reviewed and concluded to be the most popular CAD package in industry and contained the best development design tools out of all of the CAD packages surveyed. Initial development started with AutoCAD release 10 on the 386 personal computer. This proved to be frustrating and fell short of our expectations and requirements. The release of AutoCAD release 11 and 12 proved to answer most of our questions and solve most of our earlier problems. Now pending publishing of this thesis ongoing development is being done towards commercialization of *developable surfaces* as a third party developer package. Much effort is needed in order to create user interactive tools sophisticated enough to ensure useful interactive CAD tools that the user would find intuitive to use.

6.3.1 CAD Program Environment and Open Architecture

The CAD program environment using AutoCAD ADS development tools proved to be what was needed for *developable surfaces*. The ADS development tools are C language calls which are registered and loaded as applications for AutoCAD. The *ADS development tools* have proved to be of high quality and as an *open architecture* to build and expand upon the existing tools contained in AutoCAD. This *open architecture* has enabled AutoCAD features and newer releases to grow very rapidly. The writing of *ADS development tools* in ANSI C language has also enabled AutoCAD to be ported to other platforms very quickly and easily with minimal effort when compared to other languages.

6.3.2 Present Limitations

So far the present limitations of AutoCAD and *ADS development tools* are few. Some awkwardness has occured when implemented AutoCAD ADS applications tools with some of the C++ modules. True C++ OOD, OOP techniques can be affected by some of the C functions and how the ADS environment is structured which limits some of the C++ design. Future releases of AutoCAD and ADS tools will probably evolve into C++ classes and methods. Existing design changes are being implemented quite easily into the AutoCAD ADS environment. Autodesk has been very supportive and provided encouragement for our ongoing research and development implementing C++ to ADS.

Chapter 7

Demonstration Examples

In this chapter there are four example sections which display the benefits and problems still facing the present design. The first section describes the power and flexibility of the base module which, when unconstrained, can create more difficult geometric anomalies in computational geometry.

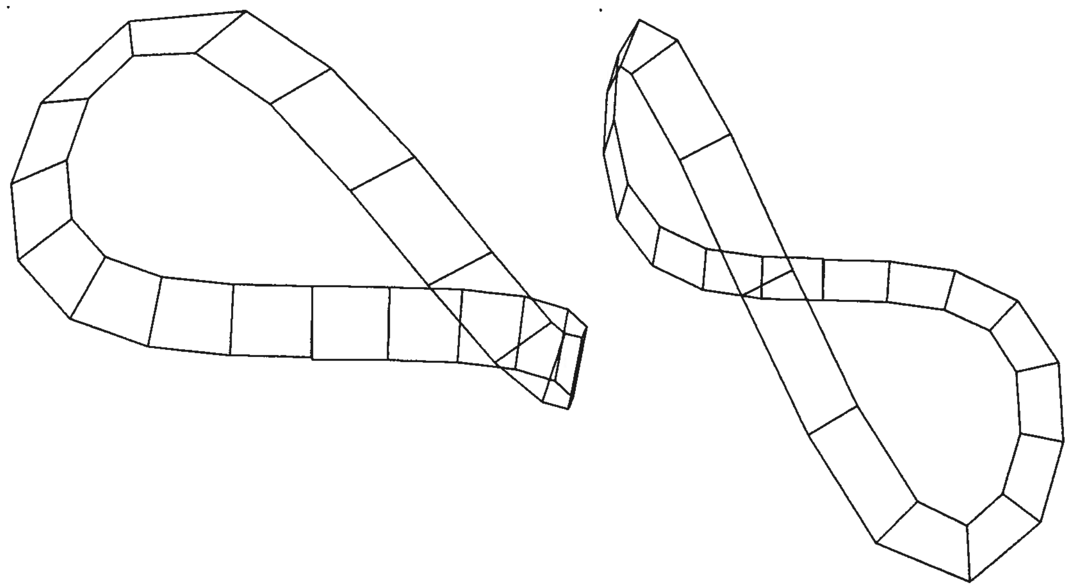
The other sections show more realistic applications and the difficulties that arise trying to "best fit" two or more space curves to the exact geometry desired. One must be made aware that from any two or more space curves, a developable surface may not exist; only a closest fitting one may be a solution. The reader should be made aware that this is a design tool, and it should also provide the user with feedback as to what type of geometries are potentially developable.

7.1 Developable Mobius Strip

The first example figures shown below in figure 7.1, which were presented in chapter 3, show the flexibility of the base module, given freedom to solve the geometric problem when working from a *directrix* and two generators only.

This geometric anomaly is interesting to view and cite as an example and perhaps as mathematical art, but provides no potential commercial benefit to the industrial designer.

The next few sections exemplify more practical examples in commercial areas such as manufacturing. When "harder" constraints are involved new problems arise. Examples could have been cited where these design algorithms would prove to the reader that they



(a) Developable mobius strip view 1 (b) Developable mobius strip view 2

Figure 7.1: Developable mobius Strip

work very well, but, they would also prove to be misleading. This thesis was presented in a very objective engineering manner which shows both the positive and negative sides inherent in design.

7.2 Simple Conical Developable

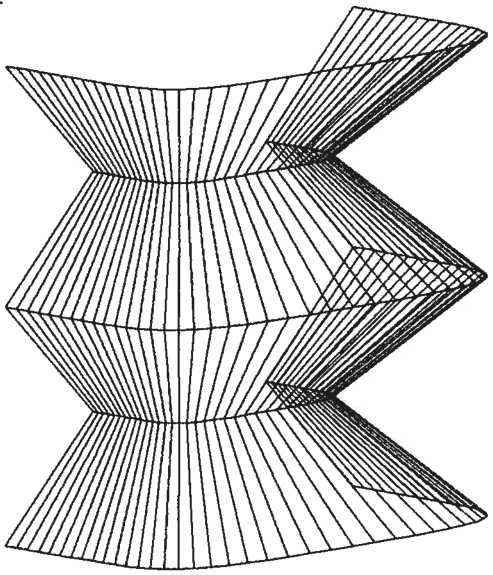
A simple conical developable surface was created in order to verify and check compliance with known developable surfaces, fanning and intersection of surfaces.

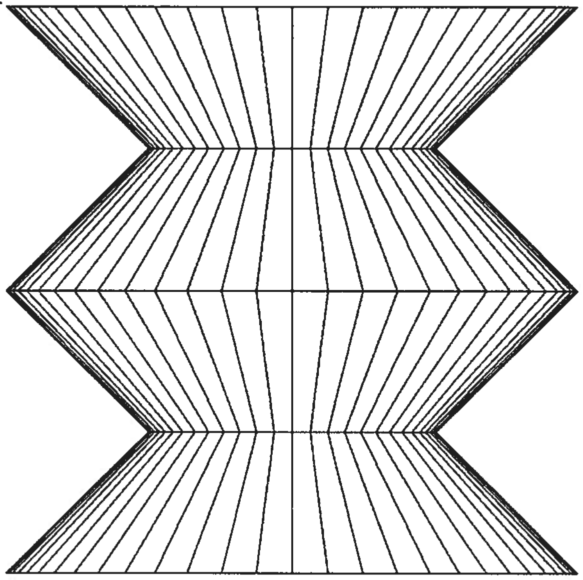
Shown above 7.2 are surfaces which intersect and verify results.

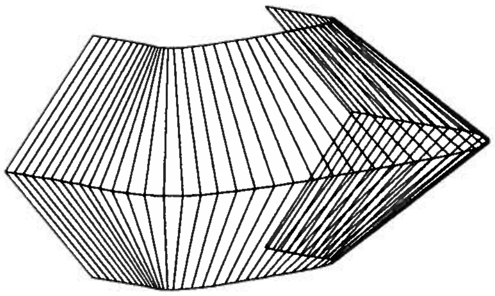
7.3 Arctic Fishing Vessel

7.4 UBC Series Fishing Vessel

Referring to figure 7.5 the ubc series was found to be the most difficult to model since the bow contains compound curvature hull form in certain regions from station 0 to station 2.0.


(a) Conical-type surfaces view point (1,1,1)


(b) Conical-type surfaces front view


(c) Conical-type surfaces view point (1,1,1)

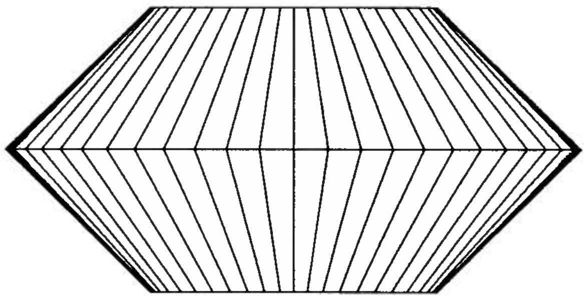
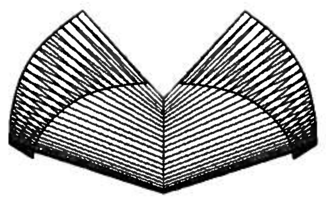
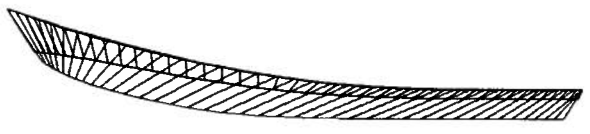

(d) Conical-type surfaces front view

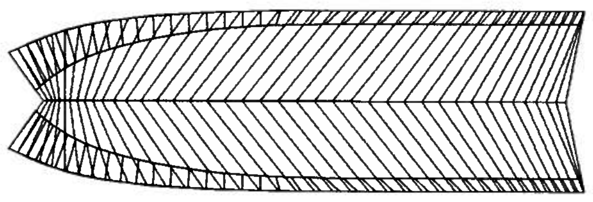
Figure 7.2: Conical-type Surfaces Intersections



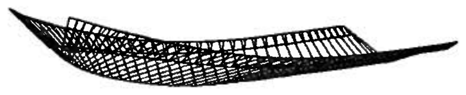
(a) ARCTIC vessel view point (1,0,0)



(b) ARCTIC vessel view point (0,-1,0)

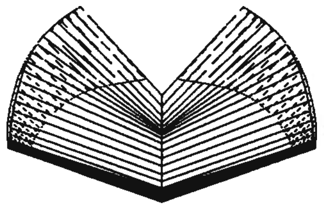


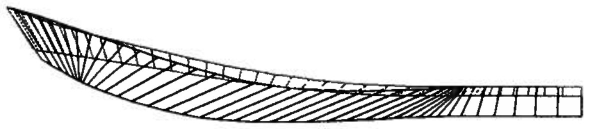
(c) ARCTIC vessel view point (0,0,1)

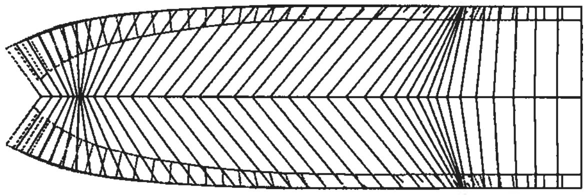


(d) ARCTIC vessel view point (-1,-1,0.2)

Figure 7.3: Arctic Vessel Conventional Approach


(a) ARCTIC vessel view point (1,0,0)


(b) ARCTIC vessel view point (0,-1,0)


(c) ARCTIC vessel view point (0,0,1)

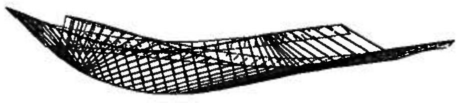
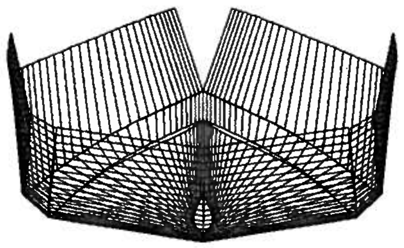
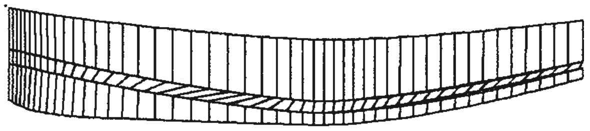
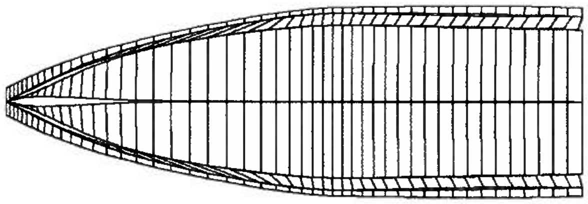

(d) ARCTIC vessel view point (-1,-1,0.2)

Figure 7.4: Arctic Vessel Modern Approach


(a) UBC series vessel view point (1,0,0)


(b) UBC series vessel view point (0,-1,0)


(c) UBC series vessel view point (0,0,1)

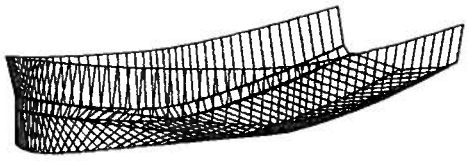

(d) UBC series vessel view point (-1,-1,0.2)

Figure 7.5: UBC series Vessel Conventional Approach

Chapter 8

Conclusions and Recommendations

8.1 Conclusions

The research objectives were

- to develop an algorithm in order to find a normal directrix such that the resulting developable surface lay close to two space curves representing desired edges of the developable surface.
- to create an algorithm to intersect developable surfaces and to generate the flat plate layouts and angles.
- to implement these algorithms using a modern computer language and a popular CAD package in order to assess the practicality of the approach.

Normal Directrix from Two Space Curves

An algorithm was created to compute a normal directrix for a developable surface from a pair of space curves. Differential equations were derived for defining a set of generators lying between the space curves, similar to the approaches taken by Nolan and Clements. The set of generators were used to compute a normal directrix, which was approximated by a parametric spline.

To ensure that the rulings between the ends of the space curves were also generators of the developable surface, the tangents at the ends of the space curves were aligned so that the cross-products between the end rulings and the two tangents at each end were co-linear. The control verticies of the normal directrix were adjusted so that the developable surface started from one end of the normal directrix aligned with the opposite end generator.

The algorithm created yielded undesireable fluctuations in generator direction when the surface was nearly flat. A threshold limit for the curvature was used, below which the generator direction was held constant. Practical examples with hard-chine ship hulls demonstrated the robustness of the approach.

A non-linear optimization technique, the downhill simplex method, was implemented to further refine the shape of the developable surface. Limited success was achieved.

Intersection of Developable Surfaces and Flat Plate Layout

An algorithm was derived to define the edges of a developable surface by intersection with adjacent developable surfaces. The method proved to be robust, with the exception when two adjacent developable surfaces are nearly co-planar.

Differential equations were also derived for creating a flat plate layout of a developable surface, including axes of bending and plate curvatures.

Implementation

The C++ programming language was used to implement the algorithms. This enabled an object-oriented and modular approach, for example 3-D space curves were implemented as a class of objects with member functions such as position, tangent and curvature as a function of the independent parameter. As a class of objects, details of the implementation of curves were transparent to or isolated from the rest of the program. Other features of this programming language enabled the main program segments to be written clearly and concisely in a mathematical format.

AutoCAD was selected as the host CAD package because of its wide acceptance, multiple platforms and development tools. Only preliminary steps have been taken to date to incorporate the algorithms and code developed directly into the CAD package.

8.2 Recommendations

- Alternate approaches to the threshold limit on curvature should be considered.
- Drop non-linear optimization for adjustment of the shape of the normal directrix.
- Implement a threshold limit for the shape of the intersection between adjacent surfaces when nearly co-planar.
- Implement non-uniform rational beta spline and non-uniform rational tension Catmull-Rom spline when they are available.
- A basic set of algorithms should be implemented including
  - creating a normal directrix from two space curves,
  - intersection of adjacent developable surfaces,
  - generation of flat plate layout and
  - direct interactive control of the shape of a normal directrix.

Bibliography

[1] R. A. Adams. *Calculus of Several Variables.* Addison-Wesley Publishers, 1987.

[2] Glenn D. Aguilar. Definition of Developable Surfaces with High Level Computer Graphics. In *Proceedings at the Pacific Northwest Section of the Society of Naval Architects and Marine Engineers*, pages 1 – 21, January 1987.

[3] B.A. Barsky. *Computer Graphics and Geometric Modeling Using Beta-Splines.* Springer-Verlag, 1988.

[4] P. E. Bezier. *Numerical Control-Mathematics and Applications.* John Wiley & Sons, 1972.

[5] G. Booch. *Object Oriented Design with Applications.* Benjamin Cummings, 1991.

[6] W.E. Boyce and R.C. DiPrima. *Elementary Differential Equations and Boundary Value Problems.* John Wiley & Sons, 1977.

[7] J.C. Clemens. A Computer System to Derive Developable Hull Surfaces and Tables of Offsets. *Marine Technology*, 18(3):227 – 233, July 1981.

[8] T.D. DeRose and B.A. Barsky. Geometric Continuity and Shape Parameters for Catmull-Rom Splines (Extended Abstract. *Proceedings of Graphics Interface*, (27):57 – 64, May - June 1984.

[9] T.D. DeRose and B.A. Barsky. Geometric Continuity, Shape Parameters, and Geometric Constructions for Catmull-Rom Splines. *ACM Transactions on Graphics*, 7(1):1 – 41, January 1988.

[10] A. B. Dunwoody. Computer Aided Design of Developable Surfaces. *not yet published*, 10, 1989.

[11] editor E.V. Lewis. *Principles of Naval Architecture, 2nd ed.* Volume 1,2,3, Society of Naval Architects and Marine Engineers, 1989.

[12] J.D. Foley, A. VanDam, Steven K. Feiner, and John F. Hughes. *Computer Graphics Principles and Practice.* Addison-Wesley Publishing Company, 1990.

[13] M.E. Hohmeyer and B.A. Barsky. Rational Continuity: Parametric, Geometric, and Frenet Frame Continuity of Rational Curves. *ACM Transactions on Graphics*, 8(4):335 – 359, October 1989.

[14] S.L.S. Jacoby. *Iterative Methods for Nonlinear Optimization Problems.* Prentice-Hall, 1972.

[15] U. Kilgore. *Developable Hull Surfaces.* Fishing News (Books) Ltd., 1967.

[16] J.A. Nelder and R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308 – 313, 1965.

[17] T.J. Nolan. Computer-Aided Design of Developable Hull Surfaces. *Marine Technology*, 233 – 242, April 1971.

[18] J.C. Beatty R.H. Bartels and B.A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Design.* Morgan Kaufmann Publishers, 1987.

[19] D. F. Rogers. B-Spline Curves and Surfaces for Ship Hull Definition. *Society of Naval Architects and Marine Engineers*, 1977.

[20] S.A. Teukolsky W.H. Press, B.P. Flannery and W.T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1988.

Appendix A

Mathematical Notation for Partial Differentiation

Notation used throughout is that which is defined in the text by Adams[1].

Notation example:

$$p = f(x, y, z) \;\; where \;\; x = x(t) \; y = y(t) \; z = z(t)$$

We can then say:

$$p = F(t) = f(x(t), y(t), z(t))$$

where we can regard p as a function of the single variable t.

Since p depends on t through both of the variables of f, the chain rule for $\frac{dp}{dt}$ has three terms:

$$\frac{dp}{dt} = \frac{\partial p}{\partial x}\frac{dx}{dt} + \frac{\partial p}{\partial y}\frac{dy}{dt} + \frac{\partial p}{\partial z}\frac{dz}{dt}$$

The use of the straight "d" denotes derivatives of only one variable.

For example:

$$The \;\; Derivative \;\; \frac{dp}{dt} \;\; means \;\; F'(t)$$
$$while \;\; \frac{\partial p}{\partial x} \;\; means \;\; f_1(x, y, z) = \frac{\partial}{\partial x} f(x, y, z)$$

Appendix B

Derivation of Developable Surface

B.1 Constraints which Define the Developable Surface

The definitions presented below were from previous work and on going research.

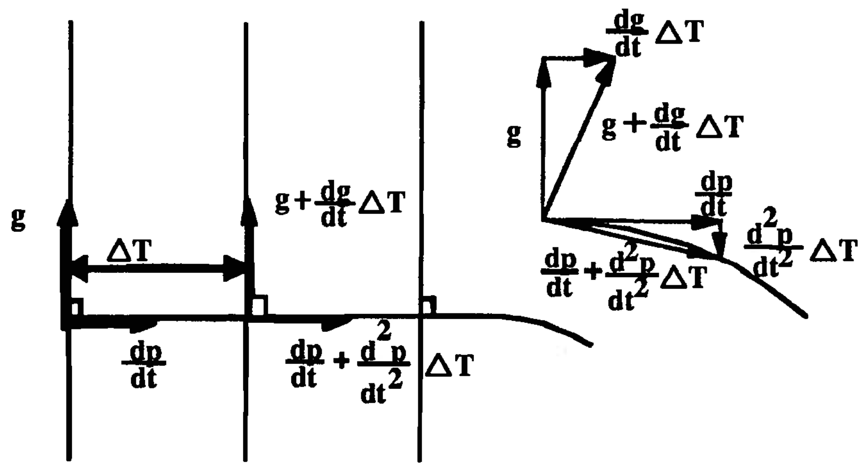


Figure B.1: Derivation of Developable Surface

Vectors Must Be Perpendicular

$$
\begin{aligned}
(g+\frac{dg}{dt}\Delta T)\cdot(\frac{dp}{dt}+\frac{d^2p}{dt^2}\Delta T) &= 0\\
(g\cdot\frac{dp}{dt}+\frac{dg}{dt}\cdot\frac{dp}{dt}\Delta T+g\cdot\frac{d^2p}{dt^2}\Delta T+\frac{dg}{dt}\cdot\frac{d^2p}{dt^2}\Delta T^2 &= 0)\\
(\frac{dg}{dt}\cdot\frac{dp}{dt}+g\cdot\frac{d^2p}{dt^2} &= 0)\\
\frac{dg}{dt}\cdot\frac{dp}{dt} &= -\frac{d^2p}{dt^2}\cdot g
\end{aligned}
$$

Vector g is of Constant Length

$$g \cdot \frac{dg}{dt} = 0$$

The Normal is Invariant Along a Generator

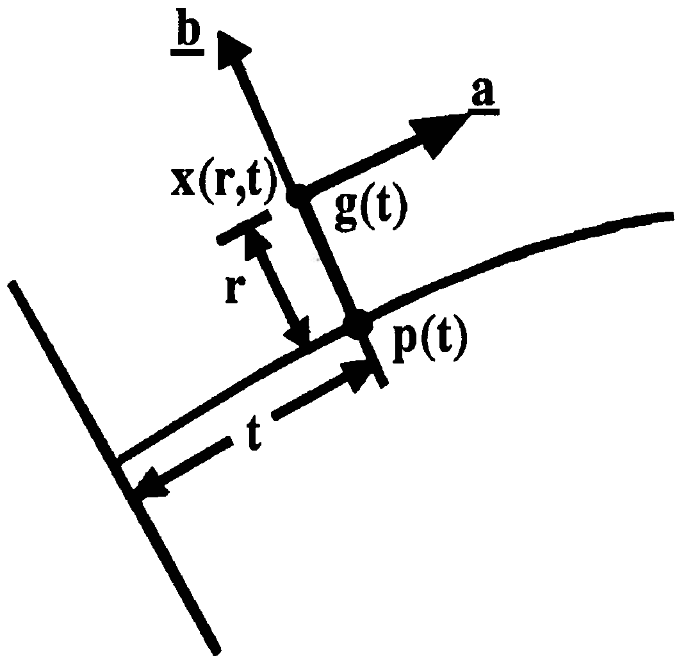


Figure B.2: Derivation showing normal is invariant along a generator

$$\begin{aligned}
x(r,t) &= p(t) + rg(t) \; where \; r \; is \; a \; scalar \\
\underline{a} &= \frac{dx}{dt} = \frac{dp}{dt} + r\frac{dg}{dt} \\
\underline{b} &= \frac{dx}{dr} = g \\
\underline{n} &= normal = \underline{a} \times \underline{b} = (\frac{dp}{dt} + r\frac{dg}{dt}) \times g \\
\underline{n} &= (\frac{dp}{dt} \times g) + (r\frac{dg}{dt} \times g)
\end{aligned}$$

$$\begin{aligned} But\ (\frac{dp}{dt} \times g) \parallel (r\frac{dg}{dt} \times g)\ ie. \quad are \quad parallel \\ therefore\ (\frac{dp}{dt} \times g) \cdot \frac{dg}{dt} \quad = \quad 0 \end{aligned}$$

B.2 Proof Using Constraints

$$\frac{dg}{dt} = -\frac{(\frac{d^2p}{dt^2} \cdot g)}{(\frac{dp}{dt} \cdot \frac{dp}{dt})}\frac{dp}{dt}$$

Let **B** be the constant we need to satisfy:

$$\frac{dg}{dt} = \mathbf{B}\frac{dp}{dt}$$

<u>CONSTRAINTS</u>

1. Vectors must be of unit lentgth.

$$g \cdot \frac{dg}{dt} = 0 \Longrightarrow g \cdot \mathbf{B}\frac{dp}{dt} = 0$$

2. Normal is invariant along a generator.

$$\begin{aligned} (g \times \frac{dp}{dt}) \cdot \frac{dg}{dt} &= 0 \\ \Longrightarrow (g \times \frac{dp}{dt}) \cdot \mathbf{B}\frac{dp}{dt} &= 0 \\ Identity:\ A \cdot (B \times C) &= B \cdot (C \times A) \\ \mathbf{B}\frac{dp}{dt} \cdot (g \times \frac{dp}{dt}) &= 0 \\ \Longrightarrow \mathbf{B}g \cdot (\frac{dp}{dt} \times \frac{dp}{dt}) &= 0 \end{aligned}$$

3. Vectors must be Perpendicular.

$$\begin{aligned}
\frac{dg}{dt}\cdot\frac{dp}{dt} &= -\frac{d^2p}{dt^2}\cdot g \\
\mathbf{B}(\frac{dp}{dt}\cdot\frac{dp}{dt}) &= -\frac{d^2p}{dt^2}\cdot g \\
\mathbf{B} &= -\frac{(\frac{d^2p}{dt^2}\cdot g)}{(\frac{dp}{dt}\cdot\frac{dp}{dt})} \\
therefore:\ \frac{dg}{dt} &= -\frac{(\frac{d^2p}{dt^2}\cdot g)}{(\frac{dp}{dt}\cdot\frac{dp}{dt})}\frac{dp}{dt}
\end{aligned}$$

Appendix C

Derivation of Rate of Rotation of Generator Differential Equation

This derivation also includes some new terminology.

A differential equation is derived which calculates the rate of rotation of a generator with respect to changes in one of the parameters of the directrix curve.

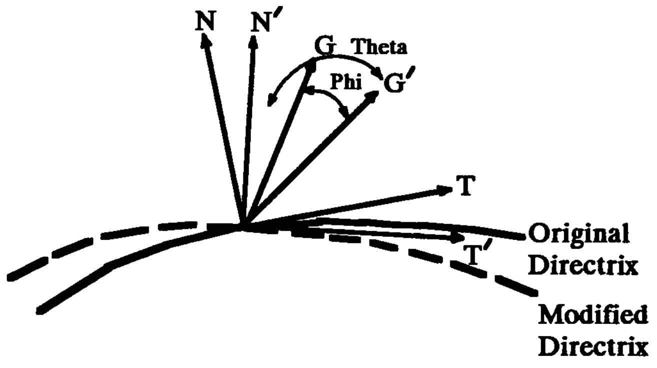


Figure C.1: Vector locations and corresponding angles

The angle Theta, herein referred to as θ, is the out of plane rotation of G' with respect to G.

The angle Phi, is herein referred to as ϕ, and is the in plane rotation of G' with respect to G.

The variable a is a parameter describing directrix.

The rate of rotation of a generator with respect to changes in one of the parameters of the directrix curve is written as:

$$\frac{d^2\theta}{dadt}$$

The desired expression is the rotation of a generator with respect to changes in one of the parameters of the directrix, which is written as:

$$\frac{d\theta}{da}$$

This desired expression can be defined as:

$$\begin{aligned}
\frac{d\theta}{da} &= \frac{dg}{da}\cdot N \\
\textit{where, } N \textit{ is} &\quad \textit{the} \quad \textit{unit normal defined as :} \\
N &= T \times g \\
\textit{therefore,} & \\
\frac{d}{dt}\frac{d\theta}{da} &= *\tfrac{d}{dt}\left(\tfrac{dg}{da}\cdot N\right)
\end{aligned}$$

giving,

$$\frac{d^2\theta}{dadt} = \frac{d^2g}{dadt}\cdot N + \frac{dg}{dt}\cdot\frac{dN}{dt} \tag{C.1}$$

From the derivation of a developable surface:

$$\begin{aligned}
\frac{dg}{dt} &= -\frac{\left(\frac{d^2p}{dt^2}\cdot g\right)}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}T \\
\textit{where } T \textit{ is of unit length :} & \\
T &= \frac{\frac{dp}{dt}}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}} \\
\textit{therefore,} & \\
\frac{d^2g}{dadt} &= -\frac{\left(\frac{d^2p}{dt^2}\cdot g\right)}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}\frac{dT}{da} - T\frac{d}{da}\left(\frac{\left(\frac{d^2p}{dt^2}\cdot g\right)}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}\right)
\end{aligned}$$

The second term in the above equation disappears because the derivative of a constant is zero.

therefore,

$$\frac{d^2g}{dadt} = -\frac{\left(\frac{d^2p}{dt^2} \cdot g\right)}{\sqrt{\frac{dp}{dt} \cdot \frac{dp}{dt}}} \frac{dT}{da}$$

$$\frac{d^2g}{dadt} \cdot N = -\frac{\left(\frac{d^2p}{dt^2} \cdot g\right)}{\sqrt{\frac{dp}{dt} \cdot \frac{dp}{dt}}} \left(\frac{dT}{da} \cdot N\right) \tag{C.2}$$

One of the next terms to define is the first term of the second part of

$$\frac{dg}{da} = \phi T + \theta N$$

where,

$$\phi = -\frac{dT}{da} \cdot g$$

combining gives,

$$\frac{dg}{da} = -\left(\frac{dT}{da} \cdot g\right) T + \theta N \tag{C.3}$$

One of the next terms to define is the second term of the second part of

Using the differentiation product rule on one of the definitions:

$$N = T \times g$$

yields,

$$\frac{dN}{dt} = \left(\frac{dT}{dt} \times g\right) + \left(T \times \frac{dg}{dt}\right) \tag{C.4}$$

Combining *Equation C.3 and Equation C.4* to form the second part of equation C.1,

$$\begin{aligned}
\tfrac{dg}{da}\cdot\tfrac{dN}{dt} &= \left(-\left(\tfrac{dT}{da}\cdot g\right)T+\theta N\right)\cdot\left(\left(\tfrac{dT}{dt}\times g\right)+\left(T\times\tfrac{dg}{dt}\right)\right)\\
&= \left(-\tfrac{dT}{da}\cdot g\right)\left(T\cdot\left(\tfrac{dT}{dt}\times g\right)\right)+\left(\left(-\tfrac{dT}{da}\cdot g\right)\left(T\cdot\left(T\times\tfrac{dg}{dt}\right)\right)\right)+ \\
&\quad \theta\left(N\cdot\left(\tfrac{dT}{dt}\times g\right)\right)+\theta\left(N\cdot\left(T\times\tfrac{dg}{dt}\right)\right)
\end{aligned} \tag{C.5}$$

several terms drop out of *Equation C.5*:

$$\begin{aligned}
\theta\left(N\cdot\left(T\times\frac{dg}{dt}\right)\right) &= 0\ \textit{since,}\\
\frac{dg}{dt}=\mathbf{B}T,\ \textit{hence,}\ T\times T &= 0\\
\textit{Another term also drops out}:&\\
\theta\left(N\cdot\left(\frac{dT}{dt}\times g\right)\right)&\\
\textit{Using Identity}\ A\cdot(B\times C) &= B\cdot(C\times A)\\
\left(\frac{dT}{dt}\cdot(g\times N)\right),\ \textit{but}\ T &= g\times N\\
\textit{resulting in}\ \left(\frac{dT}{dt}\cdot T\right) &= 0,\\
\textit{since, T remains} &\quad \textit{unit} \quad \textit{length}\\
\textit{One other term drops out}:&\\
\left(-\frac{dT}{da}\cdot g\right)\left(T\cdot\left(T\times\frac{dg}{dt}\right)\right)&\\
\textit{but}\frac{dg}{dt} &= \mathbf{B}T,\ \textit{resulting in}\ (T\times T)=0\\
\textit{Therefore}:&\\
\frac{dg}{da}\cdot\frac{dN}{dt} &= -\left(\frac{dT}{da}\cdot g\right)\left(T\cdot\left(\frac{dT}{dt}\times g\right)\right)\\
\textit{Using a vector triple products dot} &\quad \textit{and} \quad \textit{cross identity},
\end{aligned}$$

$$\begin{aligned}
&= -\left(\frac{dT}{da}\cdot g\right)\left(\frac{dT}{dt}\cdot (g\times T)\right)\\
\textit{using the fact that } N &= T\times g\\
&= \left(\frac{dT}{da}\cdot g\right)\left(\frac{dT}{dt}\cdot N\right)
\end{aligned}$$

Then,

$$\begin{aligned}
\frac{d^2\theta}{dadt} &= -\frac{\left(\frac{d^2p}{dt^2}\cdot g\right)}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}\left(\frac{dT}{da}\cdot N\right)+\left(\frac{dT}{da}\cdot g\right)\left(\frac{dT}{dt}\cdot N\right)\\
\frac{dT}{da} &= \frac{d}{da}(T)=\frac{d}{da}\left(\frac{\frac{dp}{dt}}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}\right)
\end{aligned}$$

Using the quotient rule for derivatives:

$$\begin{aligned}
\frac{dT}{da} &= \frac{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}\left(\frac{d^2p}{dadt}\right)-\frac{dp}{dt}\left(\frac{dp}{dt}\cdot\frac{dp}{dt}\right)^{-\frac{1}{2}}\frac{d^2p}{dadt}\cdot\frac{dp}{dt}}{\left(\frac{dp}{dt}\cdot\frac{dp}{dt}\right)}\\
&= \left(\frac{\frac{d^2p}{dadt}}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}-\frac{\left(\frac{d^2p}{dadt}\cdot\frac{dp}{dt}\right)}{\left(\frac{dp}{dt}\cdot\frac{dp}{dt}\right)^{\frac{3}{2}}}\frac{dp}{dt}\right)\\
\frac{dT}{da}\cdot N &= \left(\frac{\frac{d^2p}{dadt}\cdot N}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}-\frac{\left(\frac{d^2p}{dadt}\cdot\frac{dp}{dt}\right)}{\left(\frac{dp}{dt}\cdot\frac{dp}{dt}\right)^{\frac{3}{2}}}\frac{dp}{dt}\cdot N\right)\\
&\textit{The second term drops out because } \frac{dp}{dt}\cdot N=0\\
&\textit{Therefore:}\\
\frac{dT}{da}\cdot N &= \frac{\left(\frac{d^2p}{dadt}\cdot N\right)}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}\\
N\cdot\frac{dT}{dt} &= N\cdot\frac{d}{dt}\left(\frac{\frac{dp}{dt}}{\sqrt{\frac{dp}{dt}\cdot\frac{dp}{dt}}}\right)\\
\textit{but} &\ \textit{Using quotient rule}
\end{aligned}$$

$$= N \cdot \left(\frac{\sqrt{\frac{dp}{dt} \cdot \frac{dp}{dt}} \left(\frac{d^2p}{dt^2}\right) - \frac{dp}{dt}\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)^{-\frac{1}{2}} \left(\frac{d^2p}{dt^2} \cdot \frac{dp}{dt}\right)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)} \right)$$

$$\frac{dT}{dt} \cdot N = \frac{\frac{d^2}{dt^2} \cdot N}{\sqrt{\frac{dp}{dt} \cdot \frac{dp}{dt}}} - \frac{\left(\frac{d^2p}{dt^2} \cdot \frac{dp}{dt}\right)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)} \left(\frac{dp}{dt} \cdot N\right)$$

Again, the second term drops out because $\frac{dp}{dt} \cdot N = 0$

$$\frac{dT}{dt} \cdot N = \frac{\left(\frac{d^2p}{dt^2} \cdot N\right)}{\sqrt{\frac{dp}{dt} \cdot \frac{dp}{dt}}}$$

$$\frac{dT}{da} \cdot g = \frac{\left(\frac{d^2p}{dadt} \cdot g\right)}{\sqrt{\frac{dp}{dt} \cdot \frac{dp}{dt}}} - \frac{\left(\frac{d^2p}{dadt} \cdot \frac{dp}{dt}\right)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)^{\frac{3}{2}}} \left(\frac{dp}{dt} \cdot g\right)$$

$$\frac{dT}{da} \cdot g = \frac{\left(\frac{d^2p}{dadt} \cdot g\right)}{\sqrt{\frac{dp}{dt} \cdot \frac{dp}{dt}}}$$

Combining terms yields,

$$\frac{d^2\theta}{dadt} = -\frac{\left(\frac{d^2p}{dt^2} \cdot N\right)\left(\frac{d^2p}{dadt} \cdot g\right)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)} + \frac{\left(\frac{d^2p}{dadt} \cdot g\right)\left(\frac{d^2p}{dt^2} \cdot N\right)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)}$$

Using Identity: $(A \cdot C)(B \cdot D) - (A \cdot D)(B \cdot C)$

$= (A \times B) \cdot (C \times D)$ *yields,*

$$\frac{d^2\theta}{dadt} = \frac{\left(\frac{d^2p}{dt^2} \times \frac{d^2p}{dadt}\right) \cdot (N \times g)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)}$$

$$= \frac{\left(\frac{d^2p}{dadt} \times \frac{d^2p}{dt^2}\right)(g \times N)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)}$$

$$= \frac{\left(\frac{d^2p}{dadt} \times \frac{d^2p}{dt^2}\right) \cdot T}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)}$$

But, $A \cdot (B \times C) = (C \times A) \cdot B$

$$= \frac{\left(\frac{d^2p}{dt^2} \times T\right) \cdot \frac{d^2p}{dadt}}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)}$$

$$= \frac{\left(\frac{d^2p}{dt^2} \times \frac{dp}{dt}\right)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)^{\frac{3}{2}}} \cdot \frac{d^2p}{dadt}$$

$$\boxed{\frac{d^2\theta}{dadt} = \frac{\left(\frac{d^2p}{dt^2} \times \frac{dp}{dt}\right)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)^{\frac{3}{2}}} \cdot \frac{d^2p}{dadt}} \tag{C.6}$$

Appendix D

Tension Catmull-Rom Spline

The Catmull-Rom Spline matrix with a tension parameter, β, is shown below. The phantom point end conditions are shown on the following pages.

$$P(t) = \begin{bmatrix} u^3 & u^2 & u^1 & 1 \end{bmatrix} \cdot \frac{1}{2} \begin{bmatrix} -2.0\beta & 4.0-2.0\beta & 2.0\beta-4.0 & 2.0\beta \\ 4.0\beta & 2.0\beta-6.0 & 6.0-4.0\beta & -2.0\beta \\ -2.0\beta & 0.0 & 2.0\beta & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

$$\beta = Tension\ parameter$$

D.1 Phantom point, P(0), at 0

$$P(t) = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \frac{1}{2} \begin{bmatrix} -2.0\beta & 4.0-2.0\beta & 2.0\beta-4.0 & 2.0\beta \\ 4.0\beta & 2.0\beta-6.0 & 6.0-4.0\beta & -2.0\beta \\ -2.0\beta & 0.0 & 2.0\beta & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} = P_{i-1}$$

$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix} \left\{ \begin{array}{c} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{array} \right\} = P_{i-1}$$

$$P(0) = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \frac{1}{2} \begin{bmatrix} -2.0\beta & 4.0 - 2.0\beta & 2.0\beta - 4.0 & 2.0\beta \\ 4.0\beta & 2.0\beta - 6.0 & 6.0 - 4.0\beta & -2.0\beta \\ -2.0\beta & 0.0 & 2.0\beta & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} P_i \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

D.2 Phantom point, P(1), at n

$$P(t) = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} \frac{1}{2} \begin{bmatrix} -2.0\beta & 4.0 - 2.0\beta & 2.0\beta - 4.0 & 2.0\beta \\ 4.0\beta & 2.0\beta - 6.0 & 6.0 - 4.0\beta & -2.0\beta \\ -2.0\beta & 0.0 & 2.0\beta & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} = P_{i+2}$$

$$\begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix} \left\{ \begin{matrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{matrix} \right\} = P_{i+2}$$

$$P(1) = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} \frac{1}{2} \begin{bmatrix} -2.0\beta & 4.0 - 2.0\beta & 2.0\beta - 4.0 & 2.0\beta \\ 4.0\beta & 2.0\beta - 6.0 & 6.0 - 4.0\beta & -2.0\beta \\ -2.0\beta & 0.0 & 2.0\beta & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+1} \end{bmatrix}$$

Appendix E

Beta-Spline

$$P(t) = \begin{bmatrix} u^3 & u^2 & u^1 & 1 \end{bmatrix}$$

$$\cdot \frac{1}{\delta} \begin{bmatrix} -2.0\beta_1^3 & 2.0(\beta_2 + \beta_1^3 + \beta_1^2 + \beta_1) & -2.0(\beta_2 + \beta_1^2 + \beta_1 + 1.0) & 2.0 \\ 6.0\beta_1^3 & -3.0(\beta_2 + 2.0\beta_1^3 + 2.0\beta_1^2) & 3.0(\beta_2 + 2.0\beta_1^2) & 0.0 \\ -6.0\beta_1^3 & 6.0(\beta_1^3 - \beta_1) & 6.0\beta_1 & 0.0 \\ 2.0\beta_1^3 & (\beta_2 + 4.0\beta_1^2 + 4.0\beta_1) & 2.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

$$\begin{aligned} \delta &= \beta_2 + 2.0\beta_1^3 + 4.0\beta_1^2 + 4.0\beta_1 + 2.0 \\ &\neq 0 \end{aligned}$$

$$\begin{aligned} \beta_1 &= Bias \\ \beta_2 &= Tension \end{aligned}$$

E.1 Phantom point, P(0), at 0

$$P(t) = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$\cdot \frac{1}{\delta} \begin{bmatrix} -2.0\beta_1^3 & 2.0(\beta_2 + \beta_1^3 + \beta_1^2 + \beta_1) & -2.0(\beta_2 + \beta_1^2 + \beta_1 + 1.0) & 2.0 \\ 6.0\beta_1^3 & -3.0(\beta_2 + 2.0\beta_1^3 + 2.0\beta_1^2) & 3.0(\beta_2 + 2.0\beta_1^2) & 0.0 \\ -6.0\beta_1^3 & 6.0(\beta_1^3 - \beta_1) & 6.0\beta_1 & 0.0 \\ 2.0\beta_1^3 & (\beta_2 + 4.0\beta_1^2 + 4.0\beta_1) & 2.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} = P_{i-1}$$

$$\begin{aligned} \delta &= \beta_2 + 2.0\beta_1^3 + 4.0\beta_1^2 + 4.0\beta_1 + 2.0 \\ &\neq 0 \end{aligned}$$

$$\frac{1.0}{\delta}\begin{bmatrix} 2.0\beta_1^3 & (\beta_2 + 4.0\beta_1^2 + 4.0\beta_1) & 2.0 & 0.0 \end{bmatrix}\begin{Bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{Bmatrix} = P_{i-1}$$

$$P_{i-1} = \frac{(\beta_2 + 4.0\beta_1^2 + 4.0\beta_1)P_i + 2.0P_{i+1}}{(\delta - 2.0\beta_1^3)}$$

$$P(0) = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$\cdot \frac{1}{\delta} \begin{bmatrix} -2.0\beta_1^3 & 2.0(\beta_2 + \beta_1^3 + \beta_1^2 + \beta_1) & -2.0(\beta_2 + \beta_1^2 + \beta_1 + 1.0) & 2.0 \\ 6.0\beta_1^3 & -3.0(\beta_2 + 2.0\beta_1^3 + 2.0\beta_1^2) & 3.0(\beta_2 + 2.0\beta_1^2) & 0.0 \\ -6.0\beta_1^3 & 6.0(\beta_1^3 - \beta_1) & 6.0\beta_1 & 0.0 \\ 2.0\beta_1^3 & (\beta_2 + 4.0\beta_1^2 + 4.0\beta_1) & 2.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} \frac{(\beta_2+4.0\beta_1^2+4.0\beta_1)P_i+2.0P_{i+1}}{\delta-2.0\beta_1^3} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

E.2 Phantom point, P(1), at n

$$P(t) = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$$

$$\cdot\frac{1}{\delta}\begin{bmatrix} -2.0\beta_1^3 & 2.0(\beta_2+\beta_1^3+\beta_1^2+\beta_1) & -2.0(\beta_2+\beta_1^2+\beta_1+1.0) & 2.0 \\ 6.0\beta_1^3 & -3.0(\beta_2+2.0\beta_1^3+2.0\beta_1^2) & 3.0(\beta_2+2.0\beta_1^2) & 0.0 \\ -6.0\beta_1^3 & 6.0(\beta_1^3-\beta_1) & 6.0\beta_1 & 0.0 \\ 2.0\beta_1^3 & (\beta_2+4.0\beta_1^2+4.0\beta_1) & 2.0 & 0.0 \end{bmatrix}$$

$$\cdot\begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} = P_{i+2}$$

$$\begin{aligned} \delta &= \beta_2+2.0\beta_1^3+4.0\beta_1^2+4.0\beta_1+2.0 \\ &\neq 0 \end{aligned}$$

$$\frac{1.0}{\delta}\begin{bmatrix} 0.0 & 2.0\beta_1^3 & (4.0\beta_1^2 + 4.0\beta_1 + \beta_2) & 2.0 \end{bmatrix} \begin{Bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{Bmatrix} = P_{i+2}$$

$$P_{i+2} = \frac{2.0\beta_1^3 P_i + (4.0\beta_1^2 + 4.0\beta_1 + \beta_2) P_{i+1}}{\delta - 2.0}$$

$$P(1) = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$$

$$\cdot \frac{1}{\delta} \begin{bmatrix} -2.0\beta_1^3 & 2.0(\beta_2 + \beta_1^3 + \beta_1^2 + \beta_1) & -2.0(\beta_2 + \beta_1^2 + \beta_1 + 1.0) & 2.0 \\ 6.0\beta_1^3 & -3.0(\beta_2 + 2.0\beta_1^3 + 2.0\beta_1^2) & 3.0(\beta_2 + 2.0\beta_1^2) & 0.0 \\ -6.0\beta_1^3 & 6.0(\beta_1^3 - \beta_1) & 6.0\beta_1 & 0.0 \\ 2.0\beta_1^3 & (\beta_2 + 4.0\beta_1^2 + 4.0\beta_1) & 2.0 & 0.0 \end{bmatrix}$$

$$\cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ \frac{2.0\beta_1^3 P_i + (4.0 + 4.0\beta_1 + \beta_2) P_{i+1}}{\delta - 2.0} \end{bmatrix}$$

Appendix F

Derivation of Normal Directrix Control Vertices

It was found necessary to solve for a desired number of normal directrix control vertices from two space curves. The two space curves must be the same type of splines as the normal directrix in order to minimize the complexity of calculations. The number of control vertices of each space curve can be different from each other as well as both numbers can be different than the desired number of normal directrix control vertices.

In order to solve for the normal directrix control vertices we solve a relation involving p(t), our known spline, r(t), the spline we wish to solve for, in the following error equation:

$$\text{error} = \int_0^{N-1} |p(t) - r(t)|^2 \, dt \tag{F.1}$$

where we wish to minimize the integral and solve for the Contro Vertices, C_i:

$$\begin{aligned}
\frac{\partial \text{error}}{\partial C_i} &= 2\int_0^{N-1} \frac{dp}{dC_i}(p(t) - r(t))\, dt && \text{(F.2)}\\
&= \sum_{j=0}^{N-2} \int_0^1 \frac{dp(j+s)}{dC_i}(p(j+s) - r(j+s))\, dS && \text{(F.3)}\\
&= 0 && \text{(F.4)}
\end{aligned}$$

$$\sum_{j=0}^{N-2} \int_0^1 \frac{dp(j+s)}{dC_i} \cdot p(j+s) dS = \sum_{j=0}^{N-2} \int_0^1 \frac{dp(j+s)}{dC_i} \cdot r(j+s) dS \tag{F.5}$$

where the the above terms are defined below in vector form:

$$p(j+s) = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} C_{j-1} \\ C_j \\ C_{j+1} \\ C_{j+2} \end{bmatrix} \tag{F.6}$$

$$\frac{\partial p(j+s)}{\partial C_i} = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix} \tag{F.7}$$

Where A refers to the 4×4 spline basis matrix and δ is the Dirac delta function. Using the identity $(BA)^T = A^T B^T$ expanding for triple products $(ABC)^T =$

$C^T (AB)^T = C^T B^T A^T$ we get the following relation:

$$\frac{\partial error}{\partial C_i} = \sum_{j=0}^{N-2} \int_0^1 \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix}^T [A]^T \begin{bmatrix} s^3 \\ s^2 \\ s \\ 1 \end{bmatrix} \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} C_{j-1} \\ C_j \\ C_{j+1} \\ C_{j+2} \end{bmatrix} dS \tag{F.8}$$

$$= \sum_{j=0}^{N-2} \int_0^1 r(j+s) \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix} dS \tag{F.9}$$

Simplifying some of the terms yields the following form

$$\frac{\partial error}{\partial C_i} = \sum_{j=0}^{N-2} \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix}^T [A] \int_0^1 \begin{bmatrix} s^3 \\ s^2 \\ s \\ 1 \end{bmatrix} \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} dS\, [A] \begin{bmatrix} C_0 \\ \vdots \\ C_{N-1} \end{bmatrix} \tag{F.10}$$

$$= \sum_{j=0}^{N-2} \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix}^T [A]^T \begin{bmatrix} \frac{1}{7} & \frac{1}{6} & \frac{1}{5} & \frac{1}{4} \\ \frac{1}{6} & \frac{1}{5} & \frac{1}{4} & \frac{1}{3} \\ \frac{1}{5} & \frac{1}{4} & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{3} & \frac{1}{2} & 1 \end{bmatrix} [A] \begin{bmatrix} C_0 \\ \vdots \\ C_{N-1} \end{bmatrix} \tag{F.11}$$

$$= \sum_{j=0}^{N-2} \Phi(i,j)\Gamma \tag{F.12}$$

$$\text{where } \Gamma = \begin{bmatrix} C_0 \\ \vdots \\ C_{N-1} \end{bmatrix} \tag{F.13}$$

$$= \left[\sum_{j=0}^{N-2} \Phi(i,j)\right] [\Gamma] \tag{F.14}$$

$$= \sum_{j=0}^{N-2} \int_0^1 r(j+s) \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix} dS \tag{F.15}$$

$$\tag{F.16}$$

$$[\Gamma] = \left[\sum_{j=0}^{N-2} \Phi(i,j)\right]^{-1} \left\{ \sum_{j=0}^{N-2} \int_0^1 r(j+s) \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} [A] \begin{bmatrix} \delta_{ij-1} \\ \delta_{ij} \\ \delta_{ij+1} \\ \delta_{ij+2} \end{bmatrix} dS \right\} \tag{F.17}$$

Once the control vertices have been solved for, the desired end condition constraints must be invoked. Namely:

$$C_0 = P_0 + (C_1 - P_0) \cdot g * g \tag{F.18}$$

$$C_N = P_N + (C_{N-1} - P_N) \cdot g * g \tag{F.19}$$

where P are space curve control vertices and C are solved for control vertices. Also,

$$g_0 = \frac{(P_1 - P_0)}{|P_1 - P_0|} \tag{F.20}$$

$$g_N = \frac{(P_M - P_N)}{|P_M - P_0|} \tag{F.21}$$

Appendix G

Modified Conventional Approach Derivation

A few trials of the normal directrix method yielded results which were very dependent upon the initial position of the normal directrix when trying to match the normal directrix method developable surface to two space curves. Further testing revealed that if a good initial guess of a normal directrix could be achieved to match to two space curves, then little optimization or correction is necessary and a closest developable surface could be found.

Below is a figure relating the variables used in the derivation of solving for an initial normal directrix control vertices given the control vertices of two space curves.

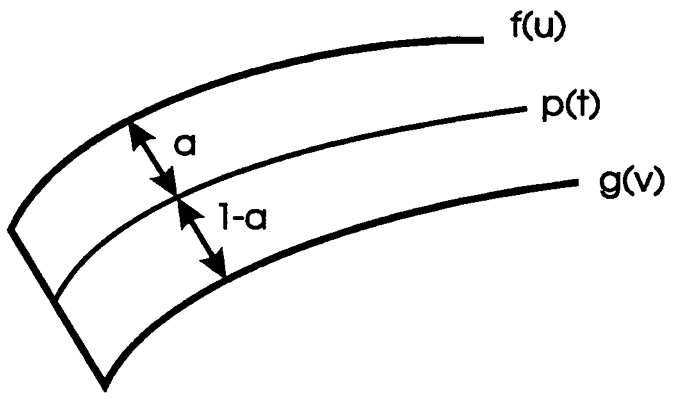


Figure G.1: Orientation of Space Curves and Directrix

From *Figure G.1* f(u) and g(v) refers to the space curves and p(t) refers to the normal directrix. We relate these as follows:

$$p(t) = (1-a)f(u) + ag(v) \tag{G.1}$$

Using partial fraction expansion from *equation G.1* we get:

$$\overrightarrow{gen} = (g-f) \tag{G.2}$$

$$\frac{\overrightarrow{dp}}{dt} = (1-a)\frac{\overrightarrow{df}}{du}\frac{du}{dt} + a\frac{\overrightarrow{dg}}{dv}\frac{dv}{dt} + \overrightarrow{gen}\frac{da}{dt} \tag{G.3}$$

From *Figure G.1* we also calculate the normals at these points, namely:

$$n_1 = \frac{\overrightarrow{df}}{du} \times \overrightarrow{gen} \tag{G.4}$$

$$n_2 = \frac{\overrightarrow{dg}}{dv} \times \overrightarrow{gen} \tag{G.5}$$

Now, relating the out-of-plane curvature with the in-plane curvature of the two space curves we get the following:

$$\left(\frac{\overrightarrow{n_1} \cdot \overrightarrow{\frac{d^2f}{du^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_1}) \cdot \frac{\overrightarrow{df}}{du}} \right) \frac{du}{dt} = \left(\frac{\overrightarrow{n_2} \cdot \overrightarrow{\frac{d^2g}{dv^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_2}) \cdot \frac{\overrightarrow{dg}}{dv}} \right) \frac{dv}{dt} \tag{G.6}$$

Also, relating the number of control vertices between the three curves and how $\frac{du}{dt}$ and $\frac{dv}{dt}$ are related we have:

$$\frac{1.0}{2n_u}\frac{du}{dt} + \frac{1.0}{2n_v}\frac{dv}{dt} = \frac{1.0}{n_t} \tag{G.7}$$

Rearranging, gives:

$$\frac{dv}{dt} = \frac{2n_v}{n_t}\left(1.0 - \frac{n_t}{2n_n}\frac{du}{dt}\right) \tag{G.8}$$

Rearranging, Equation G.6 to solve for $\frac{du}{dt}$ and $\frac{dv}{dt}$ and $\frac{da}{dt}$ gives:

$$\frac{du}{dt} = \left(\frac{n_t}{2n_u} + \frac{n_t}{2n_v} \left(\frac{\frac{\overrightarrow{n_1} \cdot \overrightarrow{\frac{d^2 f}{du^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_1}) \cdot \overrightarrow{\frac{df}{du}}}}{\frac{\overrightarrow{n_2} \cdot \overrightarrow{\frac{d^2 g}{dv^2}}}{(\overrightarrow{gen} \times \overrightarrow{n_2}) \cdot \overrightarrow{\frac{dg}{dv}}}} \right) \right)^{-1} \tag{G.9}$$

$$\frac{dv}{dt} = \frac{2n_v}{n_t} - \frac{n_v}{n_u}\frac{du}{dt} \tag{G.10}$$

$$\frac{da}{dt} = -\frac{(1-a)\overrightarrow{\frac{df}{du}} \cdot \overrightarrow{gen}\frac{du}{dt} + a\overrightarrow{\frac{dg}{dv}} \cdot \overrightarrow{gen}\frac{dv}{dt}}{\overrightarrow{gen} \cdot \overrightarrow{gen}} \tag{G.11}$$

Appendix H

Relating the Space Curves to the Developable Surface

Two forms were developed:

1. The first sets up the relations and solves for the equations
2. The second calculates the parameters directly

The First Approach

There are primarily three constraints which determine the distance from a point on the developable surface to the closest point on a space curve.

They are as follows:

$$\begin{aligned} n(t) &= \frac{dp(t)}{dt} \times g(t) \\ q(t,s) &= p(t) + sg(t) \\ r(u) &= q(t,s) + ln(t) \end{aligned}$$

where,

n(t) is the normal of the directrix,

q(t,s) is a determinable point on the surface,

resulting in the determination of the corresponding location along the space curve.

There are three variables in which we need to solve for for a corresponding value of t, which is the key parameterized variable. In order to solve for these we try to solve the problem in terms of a differential equation relating the variables u,s,l, to t. This is done by partial differentiation of the function r(u). The differential equation is with respect to t is as follows:

$$\begin{aligned}
\frac{dr}{du}\frac{du}{dt} &= \frac{dp}{dt} + s\frac{dg}{dt} + g\frac{ds}{dt} + l\frac{dn}{dt} + n\frac{dl}{dt} \\
&\quad Rearranging\ into\ a\ useful\ form, \\
\frac{dr}{du}\frac{du}{dt} - g\frac{ds}{dt} - n\frac{dl}{dt} &= \frac{dp}{dt} + s\frac{dg}{dt} + l\frac{dn}{dt}
\end{aligned}$$

There are only three unknowns to solve for given the initial conditions relating each space curve to the developable surface:

1) At t = 0.0, l1 = 0.0 and l2 = 0.0

2) At t = 0.0, u1 = 0.0 and u2 = 0.0

3) At t = 0.0, q(0.0,s1) = r1(0.0) and q(0.0,s2) = r2(0.0)

4) $s1 = (q(0.0, s1) - p1(0.0)) \cdot g(t)$

$s2 = (q(0.0, s2) - p2(0.0)) \cdot g(t)$

and,

$\frac{dg}{dt}$ *and* $\frac{dn}{dt}$ can be evaluated,

$$\begin{aligned}
\frac{dg}{dt} &= -\frac{\left(\frac{d^2p}{dt^2} \cdot g\right)}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)}\frac{dp}{dt} \\
\frac{dn}{dt} &= \left(\frac{d^2p}{dt^2} \times g\right) + \left(\frac{dp}{dt} \times \frac{dg}{dt}\right)
\end{aligned}$$

Three unkowns are then solved for each space curve and the developable surface.

$$\begin{bmatrix} \frac{dr}{du}_x & -g_x & -n_x \\ \frac{dr}{du}_y & -g_y & -n_y \\ \frac{dr}{du}_z & -g_z & -n_z \end{bmatrix} \begin{bmatrix} \frac{du}{dt} \\ \frac{ds}{dt} \\ \frac{dl}{dt} \end{bmatrix} = \begin{bmatrix} \frac{dp}{dt}_x + s\frac{dg}{dt}_x + l\frac{dn}{dt}_x \\ \frac{dp}{dt}_y + s\frac{dg}{dt}_y + l\frac{dn}{dt}_y \\ \frac{dp}{dt}_z + s\frac{dg}{dt}_z + l\frac{dn}{dt}_x \end{bmatrix}$$

The Second Approach

This approach uses the same definition but only differs after the follow differential equation has been derived.

$$\overrightarrow{\frac{dr}{du}}\frac{du}{dt} - \overrightarrow{g}\frac{ds}{dt} - \overrightarrow{n}\frac{dl}{dt} = \overrightarrow{\frac{dp}{dt}} + s\overrightarrow{\frac{dg}{dt}} + l\overrightarrow{\frac{dn}{dt}} \tag{H.1}$$

Taking the dot product with vector g: (H.2)

$$\left(\overrightarrow{\frac{dr}{du}} \cdot \overrightarrow{g}\right)\frac{du}{dt} - \frac{ds}{dt}\overrightarrow{g} \cdot \overrightarrow{g} - \frac{dl}{dt}\overrightarrow{n} \cdot \overrightarrow{g} = \overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{g} + s\overrightarrow{\frac{dg}{dt}} \cdot \overrightarrow{g} + l\overrightarrow{\frac{dn}{dt}} \cdot \overrightarrow{g} \tag{H.3}$$

This simplifies to the following: (H.4)

$$\frac{ds}{dt} = \left(\overrightarrow{\frac{dr}{du}} \cdot \overrightarrow{g}\right)\frac{du}{dt} \tag{H.5}$$

Similarly, dot product with n, (H.6)

and simplifying yields: (H.7)

$$\frac{dl}{dt} = \left(\overrightarrow{\frac{dr}{du}} \cdot \overrightarrow{n}\right)\frac{du}{dt} \tag{H.8}$$

Similarly, (H.9)

dot product with $\overrightarrow{\frac{dp}{dt}}$ (H.10)

$$\left(\overrightarrow{\frac{dr}{du}} \cdot \overrightarrow{\frac{dp}{dt}}\right)\frac{du}{dt} = \overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} + s\overrightarrow{\frac{dg}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} + l\overrightarrow{\frac{dn}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} \tag{H.11}$$

$$\text{Note perpendicular} \tag{H.12}$$

$$\text{vector relationships} \tag{H.13}$$

$$\text{For n}\overrightarrow{n} = \overrightarrow{\frac{dp}{dt}} \times \overrightarrow{g} \tag{H.14}$$

$$\overrightarrow{\frac{dg}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} = -\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{g} \tag{H.15}$$

$$\overrightarrow{\frac{dn}{dt}} = \overrightarrow{\frac{d^2p}{dt^2}} \times \overrightarrow{g} + \overrightarrow{\frac{dp}{dt}} \times \overrightarrow{\frac{dg}{dt}} \tag{H.16}$$

$$\overrightarrow{\frac{dn}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} = \left(\overrightarrow{\frac{d^2p}{dt^2}} \times \overrightarrow{g}\right) \cdot \overrightarrow{\frac{dp}{dt}} + \left(\overrightarrow{\frac{dp}{dt}} \times \overrightarrow{\frac{dg}{dt}}\right) \cdot \overrightarrow{\frac{dp}{dt}} \tag{H.17}$$

$$\text{but, identity:}(A \times B) \cdot C = (C \times A) \cdot B \tag{H.18}$$

$$\text{therefore, } \left(\overrightarrow{\frac{dp}{dt}} \times \overrightarrow{\frac{dg}{dt}}\right) \cdot \overrightarrow{\frac{dp}{dt}} = \left(\overrightarrow{\frac{dp}{dt}} \times \overrightarrow{\frac{dp}{dt}}\right) \cdot \overrightarrow{\frac{dg}{dt}} \tag{H.19}$$

$$\text{Simplifying:}\overrightarrow{\frac{dn}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} = \left(\overrightarrow{\frac{d^2p}{dt^2}} \times \overrightarrow{g}\right) \cdot \overrightarrow{\frac{dp}{dt}} \tag{H.20}$$

$$\left(\overrightarrow{\frac{dr}{du}} \cdot \overrightarrow{\frac{dp}{dt}}\right) \frac{du}{dt} = \overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} - s\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{g} + l\left(\overrightarrow{\frac{d^2p}{dt^2}} \times \overrightarrow{g}\right) \cdot \overrightarrow{\frac{dp}{dt}} \tag{H.21}$$

$$\text{but Identity:}(A \times B) \cdot C = (B \times C) \cdot A \tag{H.22}$$

$$\text{therefore, } -\left(\overrightarrow{\frac{d^2p}{dt^2}} \times \overrightarrow{g}\right) \cdot \overrightarrow{\frac{dp}{dt}} = -\left(\overrightarrow{\frac{dp}{dt}} \times \overrightarrow{g}\right) \cdot \overrightarrow{\frac{d^2p}{dt^2}} = -\overrightarrow{n} \cdot \overrightarrow{\frac{d^2p}{dt^2}} \tag{H.23}$$

$$\left(\overrightarrow{\frac{dr}{du}} \cdot \overrightarrow{\frac{dp}{dt}}\right) \frac{du}{dt} = \overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} - s\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{g} - l\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{n} \tag{H.24}$$

$$\text{yielding:} \tag{H.25}$$

$$\frac{du}{dt} = \frac{\left(\overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{\frac{dp}{dt}} - \overrightarrow{\frac{d^2p}{dt^2}} \cdot (s\overrightarrow{g} + l\overrightarrow{n})\right)}{\left(\overrightarrow{\frac{dr}{du}} \cdot \overrightarrow{\frac{dp}{dt}}\right)} \tag{H.26}$$

Appendix I

Intersection of Developable Surfaces

Once the developable surfaces have been created there is then the requirement that they must intersect without gaps or spaces between them. Shown below is the vector representation of the initial conditions and their orientation relative to each other. In Figure I.1 we see the various vectors in which their dependence to each other will be shown as follows:

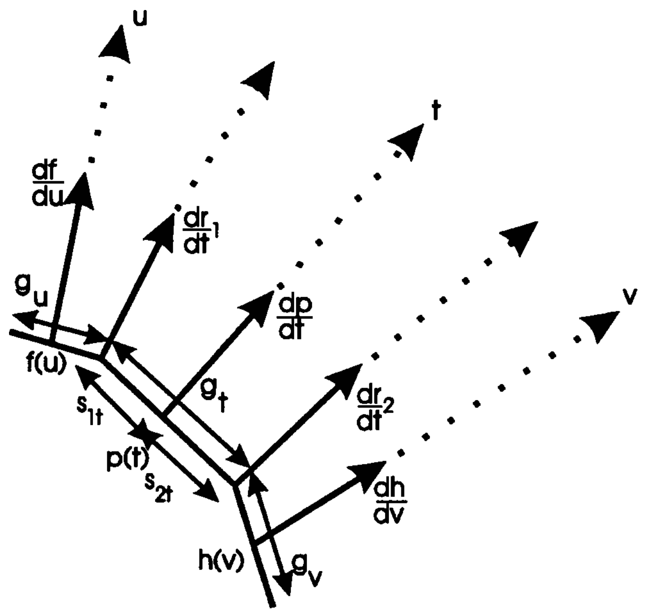


Figure I.1: Intersection of Three Developable Surfaces

As shown in Figure I.1 the independent parametric variable is t. The two dependent variables are u and v. There are three developable surface control vertices with all functions of the independent variable t. The independent space curve is $p(t)$; one dependent space curve which is a function of u is $f(u)$; the other dependent space curve is $g(v)$, which is a function of v. The intersection curves joining two developable surfaces are r_1 which is relating $f(u)$ and $p(t)$ and the other intersection curve r_2 which relates $g(v)$ and $p(t)$.

Derivation of the equations used to calculate the intersection of the developable surfaces are as follows:

$$r_1 = p_t + s_{1t}\overrightarrow{g_t} \tag{I.1}$$

$$\overrightarrow{\frac{dr_1}{dt}} = \overrightarrow{\frac{dp}{dt}} + s_{1t}\overrightarrow{\frac{dg_t}{dt}} + \frac{ds_{1t}}{dt}\overrightarrow{g_t} \tag{I.2}$$

$$\overrightarrow{\frac{dr_1}{dt}} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right) = 0 \tag{I.3}$$

$$\overrightarrow{\frac{dr_1}{dt}} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right) = \overrightarrow{\frac{dp}{dt}} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right) + s_{1t}\overrightarrow{\frac{dg_t}{dt}} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right) + \frac{ds_{1t}}{dt}\overrightarrow{g_t} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right) \tag{I.4}$$

$$\frac{ds_{1t}}{dt} = -\frac{\overrightarrow{\frac{dp}{dt}} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right) + s_{1t}\overrightarrow{\frac{dg_t}{dt}} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right)}{\overrightarrow{g_t} \cdot \left(\overrightarrow{\frac{df_u}{du}} \times \overrightarrow{g_u}\right)} \tag{I.5}$$

$$\overrightarrow{\frac{dg_t}{dt}} = -\left(\frac{\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{g_t}}{\overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{\frac{dp}{dt}}}\right)\overrightarrow{\frac{dp}{dt}} \tag{I.6}$$

$$\overrightarrow{\frac{dg_u}{du}} = -\left(\frac{\overrightarrow{\frac{d^2f}{du^2}} \cdot \overrightarrow{g_u}}{\overrightarrow{\frac{df_u}{dt}} \cdot \overrightarrow{\frac{df_u}{du}}}\right)\overrightarrow{\frac{df_u}{du}} \tag{I.7}$$

$$r_1 = f_u + s_{2u}g_u \tag{I.8}$$

$$\overrightarrow{\frac{dr_1}{dt}} = \overrightarrow{\frac{dr_1}{du}} \cdot \frac{du}{dt} \tag{I.9}$$

$$\overrightarrow{\frac{dr_1}{dt}} = \frac{du}{dt}\left(\overrightarrow{\frac{df_u}{du}} + s_{2u}\overrightarrow{\frac{dg_u}{du}} + \overrightarrow{g_u}\frac{ds_u}{du}\right) \tag{I.10}$$

$$\overrightarrow{\frac{dr_1}{dt}} \cdot \overrightarrow{\frac{df_u}{du}} = \frac{du}{dt}\left(\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} + s_{2u}\overrightarrow{\frac{dg_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} + \frac{ds_{2u}}{du}\overrightarrow{g_u} \cdot \overrightarrow{\frac{df_u}{du}}\right) \tag{I.11}$$

Since $\overrightarrow{g_u}$ is perpendicular to $\overrightarrow{\frac{df_u}{du}}$ the last term drops out (I.12)

Substituting gives, (I.13)

$$\overrightarrow{\frac{dr_1}{dt}} \cdot \overrightarrow{\frac{df_u}{du}} = \frac{du}{dt}\left(\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} - s_{2u}\left(\frac{\overrightarrow{\frac{d^2 f_u}{du^2}} \cdot \overrightarrow{g_u}}{\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}}}\right)\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}}\right) \tag{I.14}$$

But, (I.15)

$$r_1 = f_u + s_{2u}g_u, \quad and \tag{I.16}$$

$$s_{2u}g_u = r_1 - f_u \tag{I.17}$$

$$and, \; r_1 = p_t + s_{1t}g_t \tag{I.18}$$

Substituting gives, (I.19)

$$\overrightarrow{\frac{df_1}{dt}} \cdot \overrightarrow{\frac{df_u}{du}} = \frac{du}{dt}\left(\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} - s_{2u}\overrightarrow{\frac{d^2 f_u}{du^2}} \cdot \overrightarrow{g_u}\right) \tag{I.20}$$

$$= \frac{du}{dt}\left(\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} - \overrightarrow{\frac{d^2 f_u}{du^2}} \cdot (r_1 - f_u)\right) \tag{I.21}$$

$$= \frac{du}{dt}\left(\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} - \overrightarrow{\frac{d^2 f_u}{du^2}} \cdot (p_t + s_{1t}\overrightarrow{g_t} - f_u)\right) \tag{I.22}$$

$$\frac{du}{dt} = \frac{\left(\overrightarrow{\frac{dp}{dt}} + s_{1t}\overrightarrow{\frac{dg}{dt}} + \frac{ds_{1t}}{dt}\overrightarrow{g_t}\right) \cdot \overrightarrow{\frac{df_u}{du}}}{\left(\overrightarrow{\frac{df_u}{du}} \cdot \overrightarrow{\frac{df_u}{du}} - \overrightarrow{\frac{d^2 f_u}{du^2}} \cdot (p_t + s_{1t}\overrightarrow{g_t} - f_u)\right)} \tag{I.23}$$

$$\overrightarrow{\frac{dg_u}{dt}} = \overrightarrow{\frac{dg_u}{du}}\frac{du}{dt} \tag{I.24}$$

$$r_2 = p_t + s_{2t}\overrightarrow{g_t} \tag{I.25}$$

$$\overrightarrow{\frac{dr_2}{dt}} = \overrightarrow{\frac{dp}{dt}} + s_{2t}\overrightarrow{\frac{dg_t}{dt}} + \frac{ds_{2t}}{dt}\overrightarrow{g_t} \tag{I.26}$$

$$\overrightarrow{\frac{dr_1}{dt}} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times g_v\right) = 0 \tag{I.27}$$

$$= \overrightarrow{\frac{dp}{dt}} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right) + s_{2t}\overrightarrow{\frac{dg_t}{dt}} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right) + \frac{ds_{2t}}{dt}\overrightarrow{g_t} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right) \tag{I.28}$$

$$\frac{ds_{2t}}{dt} = -\frac{\overrightarrow{\frac{dp}{dt}} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right) + s_{2t}\overrightarrow{\frac{dg_t}{dt}} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right)}{\overrightarrow{g_v} \cdot \left(\overrightarrow{\frac{dh_v}{dv}} \times \overrightarrow{g_v}\right)} \tag{I.29}$$

$$\overrightarrow{\frac{dg_t}{dt}} = -\left(\frac{\overrightarrow{\frac{d^2p}{dt^2}} \cdot \overrightarrow{g_t}}{\overrightarrow{\frac{dp}{dt}} \cdot \overrightarrow{\frac{dp}{dt}}}\right)\overrightarrow{\frac{dp}{dt}} \tag{I.30}$$

$$\overrightarrow{\frac{dg_v}{dt}} = -\left(\frac{\overrightarrow{\frac{d^2h}{dv^2}} \cdot \overrightarrow{g_v}}{\overrightarrow{\frac{dh}{dv}} \cdot \overrightarrow{\frac{dh}{dv}}}\right)\overrightarrow{\frac{dh}{dv}} \tag{I.31}$$

$$r_2 = h_v + s_{1v}\overrightarrow{g_v} \tag{I.32}$$

$$\overrightarrow{\frac{dr_2}{dt}} = \overrightarrow{\frac{dr_2}{dv}}\frac{dv}{dt} \tag{I.33}$$

$$\overrightarrow{\frac{dr_2}{dt}} = \frac{dv}{dt}\left(\overrightarrow{\frac{dh_v}{dv}} + s_{1v}\overrightarrow{\frac{dg_v}{dv}} + \overrightarrow{g_v}\frac{ds_{1v}}{dv}\right) \tag{I.34}$$

$$\overrightarrow{\frac{dr_2}{dt}} \cdot \overrightarrow{\frac{dh_v}{dv}} = \frac{dv}{dt}\left(\overrightarrow{\frac{dh_v}{dv}} \cdot \overrightarrow{\frac{dh_v}{dv}} + s_{1v}\overrightarrow{\frac{dg_v}{dv}} \cdot \overrightarrow{\frac{dh_v}{dv}} + \frac{ds_{2u}}{dv}\overrightarrow{g_v} \cdot \overrightarrow{\frac{dh_v}{dv}}\right) \tag{I.35}$$

Since $\overrightarrow{\frac{dh_v}{dv}} \cdot \overrightarrow{g_v} = 0$ the last term drops out (I.36)

Substituting for $\overrightarrow{\frac{dg_v}{dv}}$ gives, (I.37)

$$r_2 = h_v + s_{1v}\overrightarrow{g_v} \tag{I.38}$$

$$s_{1v}\overrightarrow{g_v} = r_2 - h_v \tag{I.39}$$

$$r_2 = p_t + s_{2t}\overrightarrow{g_t} \tag{I.40}$$

$$\overrightarrow{\frac{dr_2}{dt}} \cdot \overrightarrow{\frac{dh_v}{dv}} = \frac{dv}{dt}\left(\overrightarrow{\frac{dh_v}{dv}} \cdot \overrightarrow{\frac{dh_v}{dv}} - s_{1v}\overrightarrow{\frac{d^2h_v}{dv^2}} \cdot \overrightarrow{g_v}\right) \tag{I.41}$$

$$= \frac{dv}{dt}\left(\overrightarrow{\frac{dh_v}{dv}} \cdot \overrightarrow{\frac{dh_v}{dv}} - \overrightarrow{\frac{d^2h_v}{dv^2}} \cdot (r_2 - h_v)\right) \tag{I.42}$$

$$= \frac{dv}{dt}\left(\overrightarrow{\frac{dh_v}{dv}} \cdot \overrightarrow{\frac{dh_v}{dv}} - \overrightarrow{\frac{d^2h_v}{dv^2}} \cdot (p_t + s_{2t}\overrightarrow{g_t} - h_v)\right) \tag{I.43}$$

$$\frac{dv}{dt} = \frac{\left(\overrightarrow{\frac{dp}{dt}} + s_{2t}\overrightarrow{\frac{dg_t}{dt}} + \frac{ds_{2t}}{dt}\overrightarrow{g_t}\right) \cdot \overrightarrow{\frac{dh_v}{dv}}}{\left(\overrightarrow{\frac{dh_v}{dv}} \cdot \overrightarrow{\frac{dh_v}{dv}} - \overrightarrow{\frac{d^2h_v}{dv^2}} \cdot (p_t + s_{2t}\overrightarrow{g_t} - h_v)\right)} \tag{I.44}$$

$$\overrightarrow{\frac{dg_v}{dt}} = \overrightarrow{\frac{dg_v}{dv}}\frac{dv}{dt} \tag{I.45}$$

Appendix J

Derivation of Flat Plates

Once a developable surface has been created, the shape and position of the consecutive flat plates that make up the developable surface need to be known. Below is a simple derivation of the algorithms used to provide the information of the shape of the plates in the x-y plane and the angle and rate of change of bending of the plates relative to each previous one.

In order to show the relations we need to define the various vectors in the two different spaces.

In the first space we are in a three dimensional system with the following termns:

The vector tangent at a point along the directrix is defined as $\frac{dp}{dt}$. The vector curvature at a point along the directrix is defined as $\frac{d^2p}{dt^2}$. The vector generator at a point along the 3 dimensional directrix is defined as g_{3d}.

In the second space we are in a two dimensional system with the following terms:

The vector tangent at a point along the directrix is defined as $\frac{dq}{dt}$. The vector curvature, to be integrated to find the next tangent, is defined as $\frac{d^2q}{dt^2}$. The vector generator at a point along the 2 dimensional directrix is defined as g_{2d}.

Since we are resolving the desired information in the 2 dimensional x-y plane, the normal, n, is obviously $n = (0,0,1)$.

Setting up the differential equation we intend to solve, we resolve the vector relationships into either in-plane or out-of-plane components. The following differential equation uses the existing definition of the differential equation defining the calculation of a generator and relates that as the in-plane component which defines the first term of the differential equation. The out-of-plane component is just resolving the out of plane curvature of the directrix and the three dimensional generator from the three dimensional system.

The resulting differential equation is used to find the new tangent and corresponding point on the x-y plane where the next point of the plate is located:

$$\frac{d^2q}{dt^2} = \frac{\left(\frac{d^2p}{dt^2} \cdot \frac{dp}{dt}\right) \frac{dq}{dt}}{\left(\frac{dp}{dt} \cdot \frac{dp}{dt}\right)} + \left(\frac{d^2p}{dt^2} \cdot g_{3d}\right) g_{2d}$$

Appendix K

O.D.E. Class, Adaptive Step Size and Runge-Kutta Method

Previous attempts at using different numerical integration methods proved that a fair amount of processing overhead takes place when making a function call to a numerical integration subroutine. Arguments must be pushed onto the stack, a call is then executed, a return is then implemented concluded with the parameters of the stack being removed.

Instead of this traditional approach, the numerical integration function was placed "inline". This technique is used in such languages as C++ and declared as an "inline" function. One may argue that this is sacrificing "modularity". It is, but when more than one function call is being made, like in a loop, then using an "inline" function shows considerable gains in speed, smaller executable files, and less functional overhead.

Evaluating the Initial Value Problem (I.V.P.):

$$y' = f(x,y) \;\; where \;\; y(x_0) = y_0$$

Runge-Kutta formula involves a weighted average of values of f(x,y) taken at different points in the interval:

$$x_n \leq x \leq x_{n+1}$$

The fourth-order Runge-Kutta is written in one of the classical forms as follows:

$$y_{n+1} = y_n + \frac{h}{6}(m_1 + 2(m_2 + m_3) + m_4)$$

$$\begin{aligned} where\ m_1 &= f(x_n, y_n) \\ m_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hm_1) \\ m_3 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hm_2) \\ m_4 &= f(x_n + h, y_n + hm_3) \end{aligned}$$

The sum $\frac{(m_1+2m_2+2m_3+m_4)}{6}$ can be interpreted as an average slope. m_1 is the slope at the left-hand of the interval, m_2 is the slope at the midpoint using the Euler formula to go from x_n to $x_n + \frac{h}{2}$, m_3 is a second approximation to the slope at the midpoint. Finally, m_4 is the slope at $x_n + h$ using the Euler formula and the slope m_3 to go from x_n to $x_n + h$.

This is a very accurate formula (halving the step size reduces the local formula error by the factor $\frac{1}{32}$); it is not necessary to compute any partial derivatives of f.

Another note should be mentioned, that if f does <u>not</u> depend on y, then:

$$\begin{aligned} m_1 &= f(x_n) \\ m_2 = m_3 &= f(x_n + \frac{1}{2}h) \\ m_4 &= f(x_n + h) \end{aligned}$$

and the equation reduces to that of Simpson's rule when evaluating the integral of y' $= f(x)$:

$$y_{n+1} - y_n = \frac{h}{6}[f(x_n) + 4f(x_n + \frac{1}{2}h) + f(x_n + h)]$$

Simpson's rule has an error proportional to h^5 which is in agreement with error in Runge-Kutta formula [6].

Appendix L

Non-Linear Optimization Downhill Simplex Method

The *Downhill Simplex Method* is a non-linear optimization tool for multidimensional minimization problems, ie. finding the minimum of a function of more than one independent variable. The original paper citing this method can be found by the authors Nelder and Mead [16]. A more recent overview with code samples can be found in Press [20]. A quick overview of the method and code implemented is presented here and mentioned briefly in Section 4.3.1 in this thesis.

A *simplex* is the geometrical figure consisting, in N dimensions, of N + 1 points (or vertices) and all their interconnecting line segments, polygonal faces, etc. In three dimensions it is a testrahedron, not necessarily the regular tetrahedron. The algorithm is supposed to then make its own way downhill through the geometrical configuration of an N - dimensional topography, until it encounters an (at least local) minimum [20].

The downhill simplex method must be started with N + 1 points defining an initial simplex. It then starts and takes a series of reflections, contractions and expansions. Each one of these is assigned a variable: α, reflection; β, contraction; γ, expansion. The first variable, α, is a positive constant, which is a scalar multiplication constant which mirrors the point through the simplex. The second variable, β, is a constant which values lie between 0 and 1. It is a ratio of the distance of the point relative to the simplex centroid. The third variable, γ, is greater than unity and it is a ratio of the current point to the centroid with a point along the line joining the point to the centroid.

In figure L.1 illustration *a)* shows the variable α being implemented in the algorithm.

Illustration *b)* shows both reflection and expansion. Illustration *c)* shows a contraction.

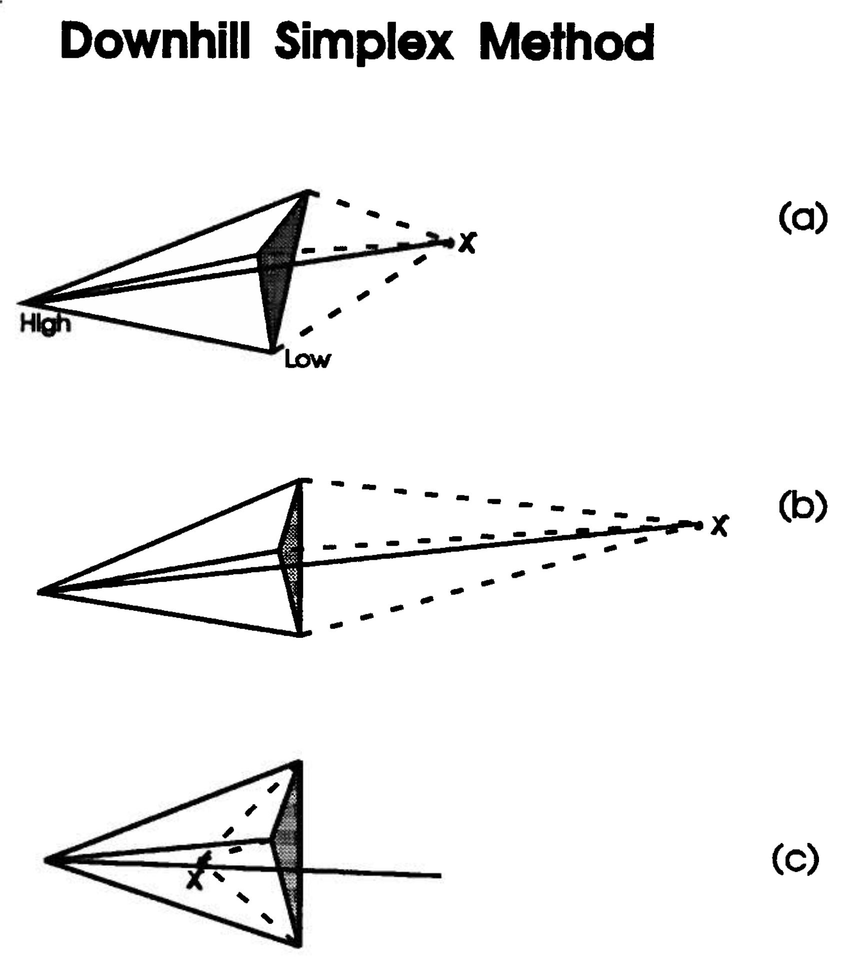


Figure L.1: Analogy of a Simplex for the Downhill Simplex Method

A few trial runs were done varying the three variables, α, β, γ and are listed in the table L.1 below.

```
#define ALPHA 0.9
#define BETA 0.4
#define GAMMA 1.9


#define GET_PSUM for (j=0;j < ndim;j++) { for (i=0,sum=0.0;i<mpts;i++)\
                          sum += p[i][j]; psum[j]=sum;}
```

α	β	γ
$\frac{1}{2}$	$\frac{1}{4}$	2.0
$\frac{2}{3}$	$\frac{1}{3}$	3.0
1.0	$\frac{1}{2}$	4.0
1.0	$\frac{1}{2}$	2.0
0.9	0.4	1.9

Table L.1: Values of variables used for Simplex Method

```
void simplex(double **p,double *y,int ndim,double ftol,double (*funk)(double *),
            int *nfunk, int nmax)
{

    int i,j,k,ilo,ihi,inhi,mpts=ndim+1,runout;
    int count=0;
    double ytry,ysave,sum,rtol,*psum;
    psum = new double[ndim];
    for(i=0; i < ndim; i++)
        psum[i]=0.0;
    double amotry(double ** ,double * ,double * ,int,double (*)(double *),
                int,int *,double);

    *nfunk=0;
    GET_PSUM
    for (;;) {
        ilo=0;
        ihi = y[0]>y[1] ? (inhi=1,0) : (inhi=0,1);
        for (i=0;i<mpts;i++) {
            if (y[i] < y[ilo]) ilo=i;
            if (y[i] > y[ihi]) {
                inhi=ihi;
                ihi=i;
            }
            else if (y[i] > y[inhi])
        if (i != ihi) inhi=i;
    }
    rtol=2.0*fabs(y[ihi]-y[ilo])/(fabs(y[ihi])+fabs(y[ilo]));
    if (rtol < ftol){
            break;
    }
    if (*nfunk >= nmax){
        cout<<"\nToo many iterations in SIMPLEX\n";
        goto runout;
    }
    ytry=amotry(p,y,psum,ndim,funk,ihi,nfunk,-ALPHA);
    if(ytry < 0.001) tol = 0.0000000000001;
    cout<<"\n1:"<<ihi<<"\n";
    if( kbhit()) goto runout;
    if (ytry <= y[ilo]){
        ytry=amotry(p,y,psum,ndim,funk,ihi,nfunk,GAMMA);
        if(ytry < -.001) tol = 0.0000000000001;
        cout<<"\n2:"<<ihi<<"\n";
        if( kbhit() ) goto runout;
    }
    else if (ytry >= y[inhi]) {
        ysave=y[ihi];
        ytry=amotry(p,y,psum,ndim,funk,ihi,nfunk,BETA);
        if(ytry < 0.001) tol = 0.0000000000001;
        cout<<"\n3:"<<ihi<<"\n";
        if( kbhit() ) goto runout;
        if (ytry > ysave) {
            for (i=0;i<mpts;i++) {
                if (i != ilo) {
                    for (j=0;j<ndim;j++){
                        psum[j]=0.5*(p[i][j]+p[ilo][j]);
                        p[i][j]=psum[j];
                    }
                    y[i]=(*funk)(psum);
                    if(kbhit()) goto runout;
                }
            }
            *nfunk += ndim;
            GET_PSUM
            }
        }
    }
    runout: runout=0;
    delete psum;
}

double amotry(double **p,double *y,double *psum,int ndim,
    double (*funk)(double *),int ihi,int *nfunk,double fac)
{
    int j;
    double fac1,fac2,ytry,*ptry;
    ptry = new double[ndim];
    fac1=(1.0-fac)/ndim;
    fac2=fac1-fac;
    for (j=0;j<ndim;j++) ptry[j]=psum[j]*fac1-p[ihi][j]*fac2;
    ytry=(*funk)(ptry);
    ++(*nfunk);
    if (ytry < y[ihi]) {
        y[ihi]=ytry;
        for (j=0;j<ndim;j++) {
            psum[j] += ptry[j]-p[ihi][j];
            p[ihi][j]=ptry[j];
        }
    }
    delete ptry;
    return ytry;
}


#undef ALPHA
#undef BETA
#undef GAMMA
```

Appendix M

Codelisting

M.1 Class Tools Used

M.1.1 vector.h

```
#include <iostream.h>

    #ifndef _VECTOR_
    #define _VECTOR_

    enum direction { X, Y, Z };

    class matrix;

    class vector {
    int n;
    static int default_length;
    float *element;
    public:
    vector(){n=default_length;element=new float[n];};
    vector(int dim) {n=dim;element=new float[n];};
    vector(int dim, float x);
    vector( const vector& v );
    ~vector();
    static void set_default(int n){default_length=n;};
    static void set_default(){default_length=3;};
    float& operator[]( int i) {return element[i];};
    float operator[](int i) const {return element[i];};
    friend int size(const vector& v) { return v.n;};
    vector& operator=(float x);
    vector& operator=(const vector& v);
    vector& operator=(const matrix& a);
    friend vector operator+(const vector& a, const vector& b);
    vector& operator+=( const vector& a);
    friend vector operator-(const vector& a, const vector& b);
    vector& operator-=(const vector& a);
    friend vector operator*(const vector& v, float a);
    friend vector operator*(float a, const vector& v);
    friend float operator*(const vector& a, const vector& b); // dot product operator
    friend vector operator*(const matrix& a, const vector& b);
    vector& operator*=(float a);
    vector& operator*=(const vector& a); // element-by-element multiplication
    friend vector operator/(const vector& v, float a);
    friend istream& operator>>(istream&, vector&);
    friend ostream& operator<<(ostream&, const vector&);
    };
    #endif
```

M.1.2 matrix.h

```
#include <vector.h>
    #include <iostream.h>

    #ifndef _MATRIX_
    #define _MATRIX_

    class matrix {
    int nr, nc;
    float **element;
    public:
    matrix(int rows, int columns );
    matrix(const matrix& a);
    matrix(const vector& a);
    ~matrix();
    matrix& operator=(float x);
    matrix& operator=(const matrix&);
    float* operator[](int i) {return element[i];};
    const float* operator[](int i) const {return element[i];};
    vector row(int i);
    friend int n_rows(const matrix& a){return a.nr;};
friend int n_columns(const matrix& a){return a.nc;};
vector column(int i);
friend matrix exp(const matrix& a);
friend matrix element_mult(const matrix& a, const matrix& b);
friend vector diagonal(const matrix& a);
friend matrix operator+(const matrix& a, const matrix &b);
friend matrix operator-(const matrix& a, const matrix& b);
friend matrix operator*(const matrix& a, const matrix& b);
friend vector operator*(const matrix& a, const vector& b);
friend matrix operator*(const matrix& a, float b);
friend matrix operator*(float a, const matrix& b) {return b*a;};
friend matrix transpose(const matrix& a);
friend matrix solve(const matrix& a, const matrix& b, float *det =NULL);
friend matrix identity(int);
friend matrix inverse(const matrix& a, float* det=NULL);
friend istream& operator>>(istream&, matrix&);
friend ostream& operator<<(ostream&, const matrix&);
friend int operator<=(const matrix&, const matrix&);
friend matrix abs(const matrix&);
};

#endif
```

M.1.3 develop.h

```
#include "matrix.h"
    #include <fstream.hpp>

    #ifndef _CURVE_
    #define _CURVE_

    class Curve{
       int n;
       char splinetype;
       vector *point;
       static matrix BB;
    public:
       void sptype(char what){splinetype=what;};
       char typeofspline(){return splinetype;};
    int size(){return n;};
    Curve(){n=5; splinetype= '1'; point = new vector[5];};
    Curve(int num){n=num; splinetype='1'; point = new vector[n];};
    Curve(Curve& a);
    ~Curve(){delete[n] point;};
    void realoc(int numm);
    vector& operator[](int i){return point[i];};
    Curve& operator=(Curve& c);
    vector operator()(double t);
    vector tangent(double t);
    vector curvature(double t);
    double deriv_2c(double t, int i);
    double deriv_c(double t, int i);
    static void initbasis(char sptype='2', double b1=1.0, double b2=0.0);
    static matrix BBB;
};
#endif


#ifndef _SURF_
#define _SURF_


class Surface{
private:
    Curve *leftedge, *rightedge;
    Curve *directrix;
    Surface *leftsurf, *rightsurf;
public:
    Surface( Curve *leftedge, Curve *rightedge, Surface *leftsurf,
        Surface *rightsurf,int ncontrolpnts=5,double step=7.0,
        double outp=0.05,int surfno=1);
    void Rightsinit(Surface *rights);
    void Leftsinit(Surface *lefts);
    double Optimize(double tolerance=0.2, int itermax = 250);
    void Develop(void);
    void Developt(void);
    void Develpt1(void);
    void Draw_3d(int genno=50,double step=7.0,int surfno=1);
    void Draw_33d(int genno=50,double step=7.0,int surfno=1);
    void Draw_3id(int genno=50,double step=7.0,int surfno=1);
```

```
   void Draw_2d(int genno=50,double steop=7.0,int surfno=1);
   ~Surface();
   Surface(Surface& s);
   Surface& operator=(Surface& s);
};
#endif

#ifndef _FILELISTS_
#define _FILELISTS_
#include <fstream.hpp>

class Filelisto{
public:
   ofstream file;
   Filelisto *next;
   ofstream& operator[](int n);
};

class Filelisti{
public:
   ifstream file;
   Filelisti *next;
   ifstream& operator[](int n);
};
#endif
```

M.1.4 ode.h

```
#ifndef _ODE_
    #define _ODE_

    #include "vector.h"

    class dynamic_system {
    private:
    double time, step_size;
    vector state;
    vector error_scale;
    vector (*derivative)(double, vector&);
public:
dynamic_system(vector (*)(double, vector&), double start_time,
                    vector& initial_state, vector *error_scale=0);
double& when();
void reset();
double& operator[](int i);
vector operator()(double t);
vector step(double delta);
};

void rk(double t0, double& del_t, double t1, vector& x,
        vector (*f)(double t, vector& x), vector& err_scale);

#endif
```

M.1.5 vector.cpp

```
#include <iostream.h>
    #include <process.h>
    #include <matrix.h>

    const char SP = ' ';

    static inline int min(int a,int b)
    {
    return a>=b?a:b;
    }

    int vector::default_length = 3;

    vector::vector(int dim, float initial_value)
    {
    n=dim;
    element=new float[n];
    for(int i=0;i<n;i++)
    element[i] = initial_value;
    }

    vector::vector(const vector& a)
```

```
{
n = a.n;
element = new float[n];
for(int i=0;i<n;i++)
element[i]=a.element[i];
}

vector ::~vector()
{
delete element;
}

vector& vector::operator=(float x)
{
for(float* temp=element+n-1;temp>=element;temp--)
*temp=x;
return *this;
}

vector& vector::operator=(const vector& v)
{
if(n!=v.n){
delete element;
n=v.n;
element=new float[n];
}
for(int i=0;i<n;i++)
element[i]=v.element[i];
return *this;
}

vector& vector::operator=(const matrix& a)
{
if(n!=n_rows(a)){
delete element;
n=n_rows(a);
element = new float[n];
}
for(int i=0;i<n;i++)
```

```
element[i] = a.element[i][0];
return *this;
}

vector operator+(const vector& a, const vector& b)
{
vector temp(a);
for(int i=0;i<min(a.n,b.n);i++)
temp.element[i] += b.element[i];
return temp;
}

vector operator-(const vector& a, const vector& b)
{
vector temp(a);
for(int i=0;i<min(a.n,b.n);i++)
temp.element[i] -= b.element[i];
return temp;
}

vector operator*(const vector &v, float a)
{
vector b(v);
for(int i=v.n-1;i>=0;i--)
b.element[i] *= a;
return b;
}

vector operator*(float a, const vector& v)
{
vector b(v);
for(int i=v.n-1;i>=0;i--)
b.element[i] *= a;
return b;
}

float operator*(const vector& a, const vector& b)
{
float temp=0.0;
```

```
for(int i=min(a.n,b.n)-1;i>=0;i--)
temp += a.element[i]*b.element[i];

return temp;
}

vector operator/(const vector& v, float a)
{
vector b(v);
for(int i=v.n-1;i>=0;i--)
b[i] /= a;
return b;
}

vector& vector::operator+=(const vector& a)
{
for(int i=min(n,a.n)-1;i>=0;i--)
element[i] += a.element[i];
return *this;
}

vector& vector::operator-=(const vector& a)
{
for(int i=min(n,a.n)-1;i>=0;i--)
element[i] -= a.element[i];
return *this;
}

vector& vector::operator*=(float a)
{
for(int i=n-1;i>=0;i--)
element[i] *= a;
return *this;
}

vector& vector::operator*=(const vector& a)
{
for(int i=min(n,a.n)-1;i>=0;i--)
element[i] *= a.element[i];
```

```
    return *this;
    }

    istream& operator>>(istream& input, vector& a)
    {
    for(int i=0;i<a.n;i++)
    input>>a.element[i];
    return input;
    }

    ostream& operator<<(ostream& output, const vector& a)
    {
    for(int i=0;i<a.n-1;i++)
    output << a.element[i] << SP;
    output << a.element[a.n-1];
    return output;
    }
```

M.1.6 matrix.cpp

```
#include <matrix.h>
    #include <iostream.h>
    #include <process.h>

    matrix::matrix(int rows,int columns)
    {
    nc=columns;
    nr=rows;
    element=new float *[nr];
    for(int i=0;i<nr;i++)
    element[i] = new float[nc];
    }

    matrix::matrix(const matrix& a)
    {
    int i,j;
    nc = a.nc;
    nr = a.nr;
```

```
element = new float *[nr];
for(i=0;i<nr;i++){
element[i] = new float[nc];
for(j=0; j<nc; j++)
element[i][j]=a[i][j];
}
}

matrix::matrix(const vector& a)
{
int i;
nc = 1;
nr = size(a);
element = new float *[nr];
for(i=0;i<nr;i++){
element[i] = new float[nc];
element[i][0] = a[i];
}
}

matrix::~matrix()
{
for(int i=0;i<nr;i++)
delete element[i];
delete element;
}

matrix& matrix::operator=(float x)
{
for(float** temp1=element+nr-1;temp1>=element;temp1--)
for(float* temp2=*temp1+nc-1;temp2>=*temp1;temp2--)
*temp2=x;
return *this;
}

matrix& matrix::operator=(const matrix& a)
{
for(int i=0;i<nr;i++){
for(int j=0;j<nc;j++)
element[i][j] = a.element[i][j];
}
return *this;
}

vector matrix::row(int i)
{
vector out(nc);
for(int j=0; j<nc; j++)
out[j] = element[i][j];
return out;
}

vector matrix::column(int i)
{
vector out(nr);
for(int j=0;j<nr;j++)
out[j] = element[j][i];
return out;
}

vector diagonal(const matrix &a)
{
vector out(a.nr<a.nc?a.nr:a.nc);
for(int i=0; i<size(out); i++)
out[i] = a[i][i];
return out;
}

matrix element_mult(const matrix &a, const matrix& b)
{
matrix result(a.nr,a.nc);
for(int i=0; i<a.nr; i++)
for(int j=0;j<a.nc;j++)
result[i][j] = a[i][j]*b[i][j];
return result;
}
matrix operator+(const matrix& a, const matrix& b)
{
```

```
matrix temp(a.nr,a.nc);
for(int i=0;i<a.nr;i++){
for(int j=0;j<a.nc;j++)
temp.element[i][j] = a.element[i][j]+b.element[i][j];
}
return temp;
}

matrix operator-(const matrix& a, const matrix& b)
{
matrix temp(a.nr,a.nc);
for(int i=0;i<a.nr;i++){
for(int j=0;j<a.nc;j++)
temp.element[i][j] = a.element[i][j]-b.element[i][j];
}
return temp;
}

matrix operator*(const matrix& a, const matrix& b)
{
matrix temp(a.nr,b.nc);
for(int i=0;i<a.nr;i++){
for(int j=0;j<b.nc;j++){
float total = 0.0;
for(int k=0;k<a.nc;k++)
total += a.element[i][k]*b.element[k][j];
temp.element[i][j]=total;
}
}
return temp;
}

vector operator*(const matrix& a, const vector& b)
{
vector result(n_rows(a));
result=0.0;
for(int i=0;i<size(result);i++){
for(const float *temp_a=&(a[i][0]), *temp_b=b.element;temp_b<b.element+b.n;temp_a++,temp_b++)
result[i]+=*temp_a**temp_b;
```

```
}
return result;
}

istream& operator>>(istream& input, matrix& a)
{
for(int i=0;i<a.nr;i++){
for(int j=0;j<a.nc;j++)
input >> a[i][j];
}
return input;
}

ostream& operator<<(ostream& output, const matrix& a)
{
for(int i=0;i<a.nr;i++){
for(int j=0;j<a.nc-1;j++)
output << a[i][j] << ' ';
output << a[i][a.nc-1] << '\n';
}
return output;
}

matrix transpose(const matrix& a)
{
matrix temp(a.nc,a.nr);
for(int i=0;i<a.nc;i++){
for(int j=0;j<a.nr;j++)
temp.element[i][j]=a.element[j][i];
}
return temp;
}

matrix identity(int a)
{
matrix temp(a, a);
for(int i=0;i<a;i++){
for(int j=0;j<a;j++){
temp[i][j]=(i==j)?1.0:0.0;
}
}
return temp;
}

matrix operator*(const matrix& a, float b)
{
int i,j;
matrix temp(a);
for(i=0;i<a.nr;i++){
for(j=0;j<a.nc;j++)
temp.element[i][j] *= b;
}
return temp;
}

int operator<=(const matrix& a, const matrix& b)
{
int i,j;
for(i=0;i<a.nr;i++){
for(j=0;j<a.nc;j++){
if(a[i][j]>b[i][j])
return(0);
}
}
return(1);
}

matrix abs(const matrix& a)
{
matrix temp(a);
int i,j;
for(i=0;i<temp.nr;i++){
for(j=0;j<temp.nc;j++){
if(temp[i][j]<0.0)
temp[i][j] = -temp[i][j];
}
}
return temp;
```

```
}
```

M.1.7 solve.cpp

```
#include <fstream.hpp>
#include <stdlib.h>
#include "matrix.h"

const double TINY = 1.0e-35;
inline void error(char * message) {cerr << message;exit(1);};
inline double abs(double & a) {
return a<0.0?-a:a;
};

matrix solve(matrix&a, matrix& b, double *det)
{
double *ptemp;
if(a.nr!=a.nc||a.nc!=b.nr)
error("matrix::solve Attempted operation on incompatible matricies\n");

matrix c(a);
double *scale;
if(det!=NULL)*det=1.0;
scale = new double[a.nr];

for(int i=0;i<a.nr;i++){ //loop over all rows, finding the largest
double largest=0.0; //element in each row for implicit scaling.
for(int j=0;j<a.nr;j++){
double temp=abs(c.element[i][j]);
if(temp>largest)
largest=temp;
}
if(largest==0.0)
error("matrix::solve singular matrix\n");
scale[i]=1.0/largest;
}

for(int j=0;j<a.nr;j++){ //loop over all columns
for(i=0;i<j;i++){ //solve for the elements of the upper triangular
double sum = c.element[i][j]; // matrix.
for(int k=0;k<i;k++)
sum -= c.element[i][k]*c.element[k][j];
c.element[i][j] = sum;
}
double largest = 0.0;
int imax=j;
for(i=j;i<a.nr;i++){ //solve for the elements of the lower triangular
double sum = c.element[i][j]; //matrix.
for(int k=0;k<j;k++)
sum -= c.element[i][k]*c.element[k][j];
c.element[i][j] = sum;
double temp=scale[i]*abs(sum); //keep track of the largest element.
if(temp>largest){
largest = temp;
imax=i;
}
}
if(imax!=j){    //if necessary, interchange rows.
ptemp = c.element[j]; //NOTE: this row interchange method depends on
c.element[j] = c.element[imax]; //the method of storing the matrix.
c.element[imax] = ptemp;
ptemp = b.element[j];   //Interchanging rows of the RHS matrix now
b.element[j] = b.element[imax]; //relieves the necessity of tracking the
b.element[imax] = ptemp; //row interchanges for later use.
if(det!=NULL)*det = (*det==1.0)?-1.0:1.0;
scale[imax]=scale[j];
}
if(c.element[j][j]==0.0)
c.element[j][j]=TINY;
double temp = 1.0/c.element[j][j];
for(i=j+1;i<a.nr;i++)
c.element[i][j] *= temp;
} //The LU decomposition in now complete.

if(det!=NULL){
for(i=0;i<a.nr;i++) //Calculate the determinant of the matrix.
*det *= c.element[i][i];
}
```

```
for(j=0;j<b.nc;j++){ //Solve for each column of the b matrix.
int ii = -1;
for(i=0;i<a.nr;i++){ //Forward substitution.
double sum = b[i][j];
int k;
if(ii>-1)
for(k=ii;k<i;k++)
sum -= c.element[i][k]*b[k][j];
else if(sum!=0.0)
ii=i;
b[i][j]=sum;
}
for(i=a.nr-1;i>=0;i--){ //Back substitution.
double sum = b[i][j];
for(int k=i+1;k<a.nr;k++)
sum -= c.element[i][k]*b[k][j];
b[i][j] = sum/c.element[i][i];
}
}
delete scale;
return b;
}

matrix inverse(matrix& a, double* det)
{
matrix b = solve(a, identity(a.nr), det);
return b;
}

double det(matrix&a)
{
double *ptemp;
double deter = 1.0;

matrix c(a);
double *scale;
scale = new double[a.nr];
```

```
for(int i=0;i<a.nr;i++){ //loop over all rows, finding the largest
double largest=0.0; //element in each row for implicit scaling.
for(int j=0;j<a.nr;j++){
double temp=abs(c.element[i][j]);
if(temp>largest)
largest=temp;
}
if(largest==0.0)
return 0.0;
scale[i]=1.0/largest;
}

for(int j=0;j<a.nr;j++){ //loop over all columns
for(i=0;i<j;i++){ //solve for the elements of the upper triangular
double sum = c.element[i][j]; // matrix.
for(int k=0;k<i;k++)
sum -= c.element[i][k]*c.element[k][j];
c.element[i][j] = sum;
}
double largest = 0.0;
int imax=j;
for(i=j;i<a.nr;i++){ //solve for the elements of the lower triangular
double sum = c.element[i][j]; //matrix.
for(int k=0;k<j;k++)
sum -= c.element[i][k]*c.element[k][j];
c.element[i][j] = sum;
double temp=scale[i]*abs(sum); //keep track of the largest element.
if(temp>largest){
largest = temp;
imax=i;
}
}
if(imax!=j){    //if necessary, interchange rows.
ptemp = c.element[j]; //NOTE: this row interchange method depends on
c.element[j] = c.element[imax]; //the method of storing the matrix.
c.element[imax] = ptemp;
deter=(deter==1.0)?-1.0:1.0;
scale[imax]=scale[j];
}
```

```
    if(c.element[j][j]==0.0)
    c.element[j][j]=TINY;
    double temp = 1.0/c.element[j][j];
    for(i=j+1;i<a.nr;i++)
    c.element[i][j] *= temp;
    } //The LU decomposition in now complete.
    for(i=0;i<a.nr;i++) //Calculate the determinant of the matrix.
    deter *= c.element[i][i];
    return deter;
    }
```

M.1.8 develop.cpp

```
////////////////////////////////////////////////////////////////////////////////
    // Class Curve C++      //
    // Modified Aug 8, 1992 by Brian Konesky, error noted deriv_2c      //
    ////////////////////////////////////////////////////////////////////////////////
    #include <math.h>
    #ifdef __ZTC__
    #include <fstream.hpp>
    #else
    #include <fstream.h>
    #endif
    #include "vector.h"
    #include "matrix.h"
    #include "develop.h"

    matrix Curve::BB(4,4);
    matrix Curve::BBB(4,4);

    void Curve::realoc(int numm)
    {
        delete[n] point;
        point = new vector[numm];
        n=numm;
    }

    Curve::Curve(Curve& a)
    {
        n=a.n;
        splinetype=a.splinetype;
        point= new vector[n];
        for(int i=0; i<n; i++) point[i]=a.point[i];
    }

    Curve& Curve::operator=(Curve& c)
    {
        if(n != c.n){
            delete[n] point;
            n=c.n;
            splinetype=c.splinetype;
            point = new vector[n];
        }
        splinetype=c.splinetype;
        for(int i=0; i<n; i++)
        point[i]=c[i];
        return *this;
    }

    vector Curve::operator()(double t)
    {
        int t2;
        double t3,bb0,bb1,bb2,bb3;
        vector p;
        matrix pi(4,3),tt(1,4),c2(1,3),cc(1,4);

        bb0 = BB[0][0]+BB[1][0]+BB[2][0]+BB[3][0];
        bb1 = BB[0][1]+BB[1][1]+BB[2][1]+BB[3][1];
        bb2 = BB[0][2]+BB[1][2]+BB[2][2]+BB[3][2];
        bb3 = BB[0][3]+BB[1][3]+BB[2][3]+BB[3][3];
        t2=(int)floor(t);
        t3=t-(double)t2;
        if(t == n - 1){
            t3 = 1.0;
            t2 = n - 2;
        }
        tt[0][0]=t3*t3*t3;
        tt[0][1]=t3*t3;
```

```
tt[0][2]=t3;
tt[0][3]=1.0;
if(splinetype == '1'){
    if(t2 == 0){
        pi[0][0]=point[t2][0];
        pi[0][1]=point[t2][1];
        pi[0][2]=point[t2][2];
    }
    else    {
        pi[0][0]=point[t2-1][0];
        pi[0][1]=point[t2-1][1];
        pi[0][2]=point[t2-1][2];
    }
    if(t2 == n - 2){
        pi[3][0]=point[t2+1][0];
        pi[3][1]=point[t2+1][1];
        pi[3][2]=point[t2+1][2];
    }
    else {
        pi[3][0]=point[t2+2][0];
        pi[3][1]=point[t2+2][1];
        pi[3][2]=point[t2+2][2];
    }
}
else if (splinetype == '2')   {
    if(t2 == 0){
        pi[0][0] = ((point[t2][0])*(1.0 - BB[3][1])
        -(point[t2+1][0])*BB[3][2])/BB[3][0];
        pi[0][1] = ((point[t2][1])*(1.0 - BB[3][1])
        -(point[t2+1][1])*BB[3][2])/BB[3][0];
        pi[0][2] = ((point[t2][2])*(1.0 - BB[3][1])
        -(point[t2+1][2])*BB[3][2])/BB[3][0];
    }
    else{
        pi[0][0] = point[t2-1][0];
        pi[0][1] = point[t2-1][1];
        pi[0][2] = point[t2-1][2];
    }
    if(t2 == n - 2){
            pi[3][0] = ((point[t2+1][0])*(1.0 - bb2)
            - (point[t2][0])*bb1
            - (point[t2-1][0])*bb0)/bb3;
        pi[3][1] = ((point[t2+1][1])*(1.0 - bb2)
            - (point[t2][1])*bb1
            - (point[t2-1][1])*bb0)/bb3;
            pi[3][2] = ((point[t2+1][2])*(1.0 - bb2)
            - (point[t2][2])*bb1
            - (point[t2-1][2])*bb0)/bb3;
        }
        else{
            pi[3][0] = point[t2+2][0];
            pi[3][1] = point[t2+2][1];
            pi[3][2] = point[t2+2][2];
        }
    }
    else   {
        if(t2 == 0){
            pi[0][0]=((point[t2][0])*2-(point[t2+1][0]));
            pi[0][1]=((point[t2][1])*2-(point[t2+1][1]));
            pi[0][2]=((point[t2][2])*2-(point[t2+1][2]));
        }
        else     {
            pi[0][0]=point[t2-1][0];
            pi[0][1]=point[t2-1][1];
            pi[0][2]=point[t2-1][2];
        }
        if(t2 == n - 2){
            pi[3][0]=((point[t2+1][0])*2-(point[t2][0]));
            pi[3][1]=((point[t2+1][1])*2-(point[t2][1]));
            pi[3][2]=((point[t2+1][2])*2-(point[t2][2]));
        }
        else {
            pi[3][0]=point[t2+2][0];
            pi[3][1]=point[t2+2][1];
            pi[3][2]=point[t2+2][2];
        }
    }
    pi[1][0]=point[t2][0];
```

```
        pi[2][0]=point[t2+1][0];
        pi[1][1]=point[t2][1];
        pi[2][1]=point[t2+1][1];
        pi[1][2]=point[t2][2];
        pi[2][2]=point[t2+1][2];
        cc = tt*BB;
        c2 = cc*pi;
        p[0]=c2[0][0];
        p[1]=c2[0][1];
        p[2]=c2[0][2];
        return p;
    }

    vector Curve::tangent(double t)
    {
        int t2;
        double t3,bb0,bb1,bb2,bb3;
        vector tang;
        matrix pi(4,3),tt(1,4),c2(1,3),cc(1,4);

        bb0 = BB[0][0]+BB[1][0]+BB[2][0]+BB[3][0];
        bb1 = BB[0][1]+BB[1][1]+BB[2][1]+BB[3][1];
        bb2 = BB[0][2]+BB[1][2]+BB[2][2]+BB[3][2];
        bb3 = BB[0][3]+BB[1][3]+BB[2][3]+BB[3][3];
        t2=(int)floor(t);
        t3=t-(double)t2;
        if(t == n - 1){
            t3 = 1.0;
            t2 = n - 2;
        }
        tt[0][0]=3.0*t3*t3;
        tt[0][1]=2.0*t3;
        tt[0][2]=1.0;
        tt[0][3]=0.0;
        if(splinetype == '1'){
            if(t2 == 0){
                pi[0][0]=point[t2][0];
                pi[0][1]=point[t2][1];
                pi[0][2]=point[t2][2];
        }
        else{
            pi[0][0]=point[t2-1][0];
            pi[0][1]=point[t2-1][1];
            pi[0][2]=point[t2-1][2];
        }
        if(t2 == n - 2){
            pi[3][0]=point[t2+1][0];
            pi[3][1]=point[t2+1][1];
            pi[3][2]=point[t2+1][2];
        }
        else {
            pi[3][0]=point[t2+2][0];
            pi[3][1]=point[t2+2][1];
            pi[3][2]=point[t2+2][2];
        }
    }
    else if (splinetype == '2'){
        if(t2 == 0){
            pi[0][0] = ((point[t2][0])*(1.0 - BB[3][1])
                -(point[t2+1][0])*BB[3][2])/BB[3][0];
            pi[0][1] = ((point[t2][1])*(1.0 - BB[3][1])
                -(point[t2+1][1])*BB[3][2])/BB[3][0];
            pi[0][2] = ((point[t2][2])*(1.0 - BB[3][1])
                -(point[t2+1][2])*BB[3][2])/BB[3][0];
        }
        else{
            pi[0][0] = point[t2-1][0];
            pi[0][1] = point[t2-1][1];
            pi[0][2] = point[t2-1][2];
        }
        if(t2 == n - 2) {
            pi[3][0] = ((point[t2+1][0])*(1.0 - bb2)
                - (point[t2][0])*bb1
                - (point[t2-1][0])*bb0)/bb3;
            pi[3][1] = ((point[t2+1][1])*(1.0 - bb2)
                - (point[t2][1])*bb1
                - (point[t2-1][1])*bb0)/bb3;
            pi[3][2] = ((point[t2+1][2])*(1.0 - bb2)
            - (point[t2][2])*bb1
            - (point[t2-1][2])*bb0)/bb3;
    }
    else{
        pi[3][0] = point[t2+2][0];
        pi[3][1] = point[t2+2][1];
        pi[3][2] = point[t2+2][2];
    }
}
else{
    if(t2 == 0){
        pi[0][0]=((point[t2][0])*2-(point[t2+1][0]));
        pi[0][1]=((point[t2][1])*2-(point[t2+1][1]));
        pi[0][2]=((point[t2][2])*2-(point[t2+1][2]));
    }
    else {
        pi[0][0]=point[t2-1][0];
        pi[0][1]=point[t2-1][1];
        pi[0][2]=point[t2-1][2];
    }
    if(t2 == n - 2){
        pi[3][0]=((point[t2+1][0])*2-(point[t2][0]));
        pi[3][1]=((point[t2+1][1])*2-(point[t2][1]));
        pi[3][2]=((point[t2+1][2])*2-(point[t2][2]));
    }
    else {
        pi[3][0]=point[t2+2][0];
        pi[3][1]=point[t2+2][1];
        pi[3][2]=point[t2+2][2];
    }
}
pi[1][0]=point[t2][0];
pi[2][0]=point[t2+1][0];
pi[1][1]=point[t2][1];
pi[2][1]=point[t2+1][1];
pi[1][2]=point[t2][2];
pi[2][2]=point[t2+1][2];
cc = tt*BB;
c2 = cc*pi;
```

```
    tang[0]=c2[0][0];
    tang[1]=c2[0][1];
    tang[2]=c2[0][2];
    return tang;
}

vector Curve::curvature(double t)
    {
    int t2;
    double t3,bb0,bb1,bb2,bb3;
    vector curv;
    matrix pi(4,3),tt(1,4),c2(1,3),cc(1,4);

    bb0 = BB[0][0]+BB[1][0]+BB[2][0]+BB[3][0];
    bb1 = BB[0][1]+BB[1][1]+BB[2][1]+BB[3][1];
    bb2 = BB[0][2]+BB[1][2]+BB[2][2]+BB[3][2];
    bb3 = BB[0][3]+BB[1][3]+BB[2][3]+BB[3][3];
    t2=(int)floor(t);
    t3=t-(double)t2;
    if(t == n - 1){
t3 = 1.0;
t2 = n - 2;
}
    tt[0][0]=6.0*t3;
    tt[0][1]=2.0;
    tt[0][2]=0.0;
    tt[0][3]=0.0;
    if(splinetype == '1')
    {
    if(t2 == 0){
     pi[0][0]=point[t2][0];
     pi[0][1]=point[t2][1];
     pi[0][2]=point[t2][2];
     }
    else     {
     pi[0][0]=point[t2-1][0];
     pi[0][1]=point[t2-1][1];
     pi[0][2]=point[t2-1][2];
     }
     if(t2 == n - 2){
      pi[3][0]=point[t2+1][0];
      pi[3][1]=point[t2+1][1];
      pi[3][2]=point[t2+1][2];
      }
     else {
      pi[3][0]=point[t2+2][0];
      pi[3][1]=point[t2+2][1];
      pi[3][2]=point[t2+2][2];
      }
     }
     else if (splinetype == '2')
     {
     if(t2 == 0)
{
pi[0][0] = ((point[t2][0])*(1.0 - BB[3][1])
     -(point[t2+1][0])*BB[3][2])/BB[3][0];
pi[0][1] = ((point[t2][1])*(1.0 - BB[3][1])
     -(point[t2+1][1])*BB[3][2])/BB[3][0];
pi[0][2] = ((point[t2][2])*(1.0 - BB[3][1])
     -(point[t2+1][2])*BB[3][2])/BB[3][0];
}
     else
{
pi[0][0] = point[t2-1][0];
pi[0][1] = point[t2-1][1];
pi[0][2] = point[t2-1][2];
}
     if(t2 == n - 2)
{
pi[3][0] = ((point[t2+1][0])*(1.0 - bb2)
 - (point[t2][0])*bb1
 - (point[t2-1][0])*bb0)/bb3;

pi[3][1] = ((point[t2+1][1])*(1.0 - bb2)
 - (point[t2][1])*bb1
 - (point[t2-1][1])*bb0)/bb3;
pi[3][2] = ((point[t2+1][2])*(1.0 - bb2)
 - (point[t2][2])*bb1
 - (point[t2-1][2])*bb0)/bb3;
}
else
{
pi[3][0] = point[t2+2][0];
pi[3][1] = point[t2+2][1];
pi[3][2] = point[t2+2][2];
}

    }
    else
    {
    if(t2 == 0){
     pi[0][0]=((point[t2][0])*2-(point[t2+1][0]));
     pi[0][1]=((point[t2][1])*2-(point[t2+1][1]));
     pi[0][2]=((point[t2][2])*2-(point[t2+1][2]));
     }
    else     {
     pi[0][0]=point[t2-1][0];
     pi[0][1]=point[t2-1][1];
     pi[0][2]=point[t2-1][2];
     }

    if(t2 == n - 2){
     pi[3][0]=((point[t2+1][0])*2-(point[t2][0]));
     pi[3][1]=((point[t2+1][1])*2-(point[t2][1]));
     pi[3][2]=((point[t2+1][2])*2-(point[t2][2]));
     }
    else {
     pi[3][0]=point[t2+2][0];
     pi[3][1]=point[t2+2][1];
     pi[3][2]=point[t2+2][2];
     }
    }


    pi[1][0]=point[t2][0];
    pi[2][0]=point[t2+1][0];
    pi[1][1]=point[t2][1];
    pi[2][1]=point[t2+1][1];
    pi[1][2]=point[t2][2];
    pi[2][2]=point[t2+1][2];
    cc = tt*BB;
    c2 = cc*pi;
    curv[0]=c2[0][0];
    curv[1]=c2[0][1];
    curv[2]=c2[0][2];

    return curv;
    }



double Curve::deriv_2c(double t, int i)
    {
    int t2;
    double t3,deriv;
    matrix tt(1,4),cc(1,4);

    t2=(int)floor(t);
    t3=t-(double)t2;
    if(t == n - 1){
t3 = 1.0;
t2 = n - 2;
}
    if( i < (t2-1) || i > (t2+2) ) return 0.0;
    tt[0][0]=3.0*t3*t3;
    tt[0][1]=2.0*t3;
    tt[0][2]=1.0;
    tt[0][3]=0.0;
    cc = tt*BB;
    if( splinetype == '2')
    {
  if(t == t2 && t != 0.0 && t != n-2)
      {
      cc[0][0]=cc[0][1];
      cc[0][2]=cc[0][3];
      }
    if(i == t2 - 1) deriv = cc[0][0];
    else if(i == t2) deriv = cc[0][1]+(t2==0.0?2.0*cc[0][0]:0.0)+
        (t2==n-2?-cc[0][3]:0.0); //Was t2=n-2, Bug caught Aug 8,1992
    else if(i == t2 + 1) deriv = cc[0][2]+(t2==0.0?-cc[0][0]:0.0)+
        (t2==n-2?2.0*cc[0][3]:0.0); //Was t2=n-2, Bug caught Aug 8, 1992
    else if(i == t2 + 2) deriv = cc[0][3];
    else deriv = 0.0;
    }
    else if( splinetype == '1')
    {
    if( i == t2 - 1) deriv = cc[0][0];
    else if(i == t2) deriv = cc[0][1] +(t2==0.0?cc[0][0]:0.0);
    else if(i == t2 + 1) deriv = cc[0][2] + (t2==n-2?cc[0][3]:0.0);
    else if(i == t2 + 2) deriv = cc[0][3];
    else deriv = 0.0;
    }
    else
    {
  if(t == t2 && t != 0.0 && t != n-2)
      {
      cc[0][0]=cc[0][1];
      cc[0][2]=cc[0][3];
      }
    if( i == t2 - 1) deriv = cc[0][0];
    else if(i == t2) deriv = cc[0][1]+(t2==0.0?2.0*cc[0][0]:0.0)
            +(t2==n-2?-cc[0][3]:0.0);
    else if(i == t2 + 1) deriv = cc[0][2]+ (t2==0.0?-cc[0][0]:0.0)
            +(t2==n-2?2.0*cc[0][3]:0.0);
    else if(i == t2 + 2) deriv = cc[0][3];
    else deriv = 0.0;
    }

    return deriv;
    }



double Curve::deriv_c(double t, int i)
    {
    int t2;
```

```
        double t3,deriv;
        matrix tt(1,4),cc(1,4);

        t2=(int)floor(t);
        t3=t-(double)t2;
        if(t == n - 1){
    t3 = 1.0;
    t2 = n - 2;
    }
        if( i < (t2-1) || i > (t2+2) ) return 0.0;
        tt[0][0]=t3*t3*t3;
        tt[0][1]=t3*t3;
        tt[0][2]=t3;
        tt[0][3]=1.0;
        cc = tt*BB;
        if( splinetype == '2')
        {
        if(i == t2 - 1) deriv = cc[0][0];
        else if(i == t2) deriv = cc[0][1] +(t2==0.0?2.0*cc[0][0]:0.0)+(t2==n-2?-cc[0][3]:0.0);
        else if(i == t2 + 1) deriv = cc[0][2] + (t2==0.0?-cc[0][0]:0.0)+(t2==n-2?2.0*cc[0][3]:0.0);
        else if(i == t2 + 2) deriv = cc[0][3];
        else deriv = 0.0;
        }
        else if( splinetype == '1')
        {
        if( i == t2 - 1) deriv = cc[0][0];
        else if(i == t2) deriv = cc[0][1] +(t2==0.0?cc[0][0]:0.0);
        else if(i == t2 + 1) deriv = cc[0][2] + (t2==n-2?cc[0][3]:0.0);
        else if(i == t2 + 2) deriv = cc[0][3];
        else deriv = 0.0;
        }
        else
        {
        if( i == t2 - 1) deriv = cc[0][0];
        else if(i == t2) deriv = cc[0][1] +(t2==0.0?2.0*cc[0][0]:0.0)+(t2==n-2?-cc[0][3]:0.0);
        else if(i == t2 + 1) deriv = cc[0][2] + (t2==0.0?-cc[0][0]:0.0)+(t2==n-2?2.0*cc[0][3]:0.0);
        else if(i == t2 + 2) deriv = cc[0][3];
        else deriv = 0.0;
        }
    return deriv;
    }


void Curve::initbasis(char spltype, double b1, double b2)
{
double del;
if(spltype == '1')
    {
    BB[0][0] = -b1;
    BB[0][1] = 2.0-b1;
    BB[0][2] = b1-2.0;
    BB[0][3] = b1;
    BB[1][0] = 2.0*b1;
    BB[1][1] = b1-3.0;
    BB[1][2] = 3.0-2.0*b1;
    BB[1][3] = -b1;
    BB[2][0] = -b1;
    BB[2][1] = 0.0;
    BB[2][2] = b1;
    BB[2][3] = 0.0;
    BB[3][0] = 0.0;
    BB[3][1] = 1.0;
    BB[3][2] = 0.0;
    BB[3][3] = 0.0;
    }
else if(spltype == '2')
    {
    del = (b2)+(2.0*(b1)*(b1))+(4.0*(b1)*(b1))+(4.0*(b1))+2.0;
    BB[0][0] = -2.0*(b1)*(b1)*(b1)/del;
    BB[0][1] = 2.0*((b2)+((b1)*(b1)*(b1))+((b1)*(b1))+(b1))/del;
    BB[0][2] = -2.0*((b2)+(b1)*(b1)+(b1)+1.0)/del;
    BB[0][3] = 2.0/del;
    BB[1][0] = 6.0*((b1)*(b1)*(b1))/del;
    BB[1][1] = -3.0*((b2)+2.0*((b1)*(b1)*(b1))+2.0*((b1)*(b1)))/del;
    BB[1][2] = 3.0*((b2)+2.0*((b1)*(b1)))/del;
    BB[1][3] = 0.0;
    BB[2][0] = -6.0*((b1)*(b1)*(b1))/del;
```

```
        BB[2][1] = 6.0*((b1)*(b1)*(b1)-(b1))/del;
        BB[2][2] = 6.0*(b1)/del;
        BB[2][3] = 0.0;
        BB[3][0] = 2.0*(b1)*(b1)*(b1)/del;
        BB[3][1] = ((b2)+4.0*((b1)*(b1))+4.0*(b1))/del;
        BB[3][2] = 2.0/del;
        BB[3][3] = 0.0;
        }
    else
        {
        del = (b2)+(2.0*(b1)*(b1))+(4.0*(b1)*(b1))+(4.0*(b1))+2.0;
        BB[0][0] = -2.0*(b1)*(b1)*(b1)/del;
        BB[0][1] = 2.0*((b2)+((b1)*(b1)*(b1))+((b1)*(b1))+(b1))/del;
        BB[0][2] = -2.0*((b2)+(b1)*(b1)+(b1)+1.0)/del;
        BB[0][3] = 2.0/del;
        BB[1][0] = 6.0*((b1)*(b1)*(b1))/del;
        BB[1][1] = -3.0*((b2)+2.0*((b1)*(b1)*(b1))+2.0*((b1)*(b1)))/del;
        BB[1][2] = 3.0*((b2)+2.0*((b1)*(b1)))/del;
        BB[1][3] = 0.0;
        BB[2][0] = -6.0*((b1)*(b1)*(b1))/del;
        BB[2][1] = 6.0*((b1)*(b1)*(b1)-(b1))/del;
        BB[2][2] = 6.0*(b1)/del;
        BB[2][3] = 0.0;
        BB[3][0] = 2.0*(b1)*(b1)*(b1)/del;
        BB[3][1] = ((b2)+4.0*((b1)*(b1))+4.0*(b1))/del;
        BB[3][2] = 2.0/del;
        BB[3][3] = 0.0;
        }
    BBB=BB;
    }


    ofstream& Filelisto::operator[](int n)
    {
    Filelisto *temp=this;
    for(; n>0; n--) temp = temp->next;
    return temp->file;
    }
    ifstream& Filelisti::operator[](int n)
    {
    Filelisti *temp=this;
    for(; n>0; n--) temp = temp->next;
    return temp->file;
    }
```

M.1.9 ode.cpp

```
#include "ode.h"


    inline double abs(double a){
    return a<0.0?-a:a;
    }


    dynamic_system::dynamic_system(vector (*f)(double, vector&), double start_time,
                vector& initial_state, vector *errors)
            :state(initial_state),error_scale(errors==0?initial_state:*errors)
    {
    time=start_time;
    step_size=0.0;
    if (errors==0){
    for (int i=size(state)-1;i>=0;i--)
    error_scale[i] = 1.0e-4;
    }
    derivative=f;
    }


    double& dynamic_system::when()
    {
    return time;
    }


    void dynamic_system::reset()
        {
        time=0.0;
        }


    double& dynamic_system::operator[](int i)
```

```
{
return state[i];
}

vector dynamic_system::operator()(double new_time)
{
if(step_size==0.0)
step_size=abs(new_time-time)/2.0;
rk(time,step_size,new_time,state,derivative,error_scale);
time=new_time;
return(state);
}
vector dynamic_system::step(double delta)
{
if(step_size==0.0)
step_size=delta/2.0;
rk(time,step_size,time+delta,state,derivative,error_scale);
time += delta;
return state;
}
```

Index