CRANFIELD UNIVERSITY

O MUNAUX

CAD INTERFACE AND FRAMEWORK
FOR CURVE OPTIMISATION APPLICATIONS

SIMS

PhD THESIS

CRANFIELD UNIVERSITY

SIMS

O MUNAUX

CAD INTERFACE AND FRAMEWORK FOR CURVE OPTIMISATION APPLICATIONS

Supervisor: G JARED

September 2004

This thesis is submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

Abstract

Computer Aided Design is currently expanding its boundaries to include more design features in its processes. Design is identified as an iterative process converging to solutions satisfying a set of constraints. Its close relation with optimisation indicate that there is strong potential for the integration of optimisation and CAD. The problem addressed in this thesis lies in interfacing the geometric representation of design with other non-geometric aspects. The example of free-form curve modelling is taken to investigate such relationships. Assumptions are made that Optimisation is powered by Evolutionary Computing algorithms like Genetic Algorithms (GA).

The geometric definition of curves is commonly supported by NURBS, whose construction constraints are defined locally at the data points. Here the NURBS formulation is used with GA in an attempt to provide complementary handles on the curves shape other than the usual data point coordinates and control points weights. Differential properties are used for optimising NURBS, Hermite interpolation allows for the definition of higher order constraints (tangent, normal, bi-normal) at data points. The assignment of parameter values at the data points, known as parameterisation also provides control of the curve's shape. Curve optimisation is also performed at the geometric modelling level. Old mathematical theorems established by Frénet and further developed by other mathematicians provide means of defining a curve's shape with it's intrinsic equations. Such representation is possible by using Function Representation (F-rep) algebra available in the ACIS software. F-rep allows more generic and exact means of interfacing with the curve's geometry and new functionality for curve inspection and optimisation are proposed in this thesis.

The integration of optimisation findings and CAD are documented in the definition of a framework. The framework architecture proposed reconstructs a new CAD environment from separate elements bolted together in a generic Application Programming Interface (API) named "Oli interface". Functionality created to interface optimisation and CAD makes a requirement list of the work that both sides should undertake to achieve design optimisation in the CAD environment.

Acknowledgements

Merci á mes parents Longue vie á mes grands parents. Bon rétablissement Thomas. Bienvenue au second, Karine et Sylvain. Plein de bonheur á tous les amis.

With this dissertation, I would like to take the opportunity to communicate my gratitude to Graham Jared for his precious help and generosity.

Part of the research presented in this thesis was conducted jointly with other researchers at Cranfield University. The research includes an industrial project entitled *"Flexible Optimisation within CAD/CAM Environment."* Also known as *"Flexo"*. The Flexo project aims to develop a framework for Flexible Optimisation within CAD/CAM Environment. A PhD thesis in the field of real life optimisation was submitted by co-researcher *Ashutosh Tiwari* at Cranfield University. In order to gain a holistic picture of the research presented here, the reader is also invited to read the volume mentioned above.

The author wish to acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC) – Grant No: GR/M 71473, Nissan Technical Centre-Europe (NTCE) and Structural Dynamics Research Corporation (SDRC) UK.

Contents

List of Figures

List of Tables

List of Equations

Introduction

For many companies, cost and built quality is no longer a competitive advantage; new innovative products must be launched regularly in an attempt to keep up with the market. In a competitive worldwide market, minimising time to market is essential to business performance: The focus is on 'creating new designs quicker'. In addition, product designs have become more complex with the advance of technology and with the growing importance of product styling. Manufacturing enterprises are introducing optimisation in their design processes to reduce design lead-time and investment cost.

At the moment the use of optimisation across industries is limited by its lack of integration in a design environment. In Appendix 1, a selection of industries using some form of optimisation in their processes are surveyed to examine the current state of the art in optimisation. Extensive internet-based search and numerous company visits have investigated the use of optimisation across engineering industries. The main source of product design data readily usable for numerical analysis and optimisation is undeniably the CAD/CAM environment. Paradoxically, the main conclusion drawn from this research is the lack of integration of optimisation capabilities within the CAD/CAM environment. An integrated optimisation framework would enable optimisation to interact with the geometric definition of product designs and subsequently optimise some of the design aspects described in paragraphs 1.5 and 3.8. The results of this survey, presented in Appendix 1, identify the need for a framework for design optimisation. In the remainder of this thesis, optimisation of free form curves is used as example. This choice is not arbitrary, because curves and surfaces also were also the centre of interest of the Flexo project and therefore some of the information gathered throughout the project could be used in this research.

The topic of this thesis is the integration of flexible optimisation within the CAD environment. It assembles knowledge from diverse topic areas such as design optimisation, evolutionary computing, geometric modelling, differential geometry. The thesis is therefore organised in three parts each constituting research in its own rights.

Part I of the thesis proposes a study of design, optimisation, evolutionary computing, and design and optimisation of free-form curves in order to establish some connectivity between these subject areas.

Chapter 1 exposes some of the theoretical aspects of design and a definition of design is established as a divergent-convergent system of finding solutions that satisfy constraints. It is thought that the design process described above can be mapped onto evolutionary computing techniques to automate its iteration loop. Hence for optimisation, the top priority is the representation of the constraints that characterise the design.

Chapter 2 examines the optimal problem formulation together with a review of some optimisation techniques available for engineering optimisation. Paragraph 2.2 directs the research towards adaptive search algorithms and evolutionary computing. The basics of Genetic Algorithms (GA) are reviewed in paragraph 2.3 and an implementation in object-oriented language C++ is documented in Appendix 2.

Chapter 3 deals with the design processes of free-form curves and surfaces. The study uses the example of the car manufacturing industry to highlight the difficulties found and the solutions this research could provide. The shape of finished goods and manufactured products exhibits a marked emphasis on smooth shapes and free flowing contours. Whether it is cars, audio equipment, cameras, ergonomic furniture or injection moulded plastic products, all have smooth contours in their design. However, the production of manufactured objects from such shapes is not an easy process. Specialised CAD software provide means to create free form contours for the design and manufacture of artefacts, but it is still a laborious process. One of the major bottlenecks in current CAD systems is the inefficiency of representation and manipulation tools for the design of free form, sculpted, three-dimensional shapes. This is forcing designers to use physical clay models and templates for generating curves and surfaces which are digitised with 3D coordinate-measuring machines (CMM) and 3D scanners to gain a computerised geometric definition.

Curve and surface optimisation depicted in paragraph 3.8, is an iterative process which consists of creating a CAD model, carrying out some analyses, which will give some indications on the surface quality, modifying the model, and so on, until satisfactory result is obtained. It is observed that 80% of the surface development time is spent on fine-tuning or optimising the geometry. Most commercial CAD/CAM systems have an integrated surfacing module that enables the user to create free form surfaces with 3D sweeps and lofts as well as NURBS patches. However free form surfaces are difficult to control because of their numerous control points and large number of degrees of freedom. As a result, most companies in the automotive sector use reverse engineering techniques to recreate geometry from clay or foam models built in styling. The discussion shows that its iterative nature is proven time consuming and does not always produce satisfactory results as designers are still faced with the problem of control points. The surface reconstruction process as it stands today strips geometry of its intrinsic properties like surface normal or curvature. At the same time, there is currently no possibility of storing surface interrogation results like shading or reflection lines.

In Part II the research aims to develop methodologies for the optimisation of free-form curves using geometric modelling, optimisation, and differential geometry.

Chapter 4 reports on theory of solid modelling to come to the topological definition of geometry. Complementary material is included in Appendix 3. As the focus of the research is on curve and surface optimisation, Chapter 5 reviews the mathematic foundations of spline curves. Interpolation methods of Lagrange and Hermite are detailed alongside the NURBS representation.

Some surface quality evaluation techniques such as curvature profile, reflection lines and offsetting are already available within CAD/CAM. The quality of the design relies on the designer’s skill to interpret the line pattern of the surface evaluation and modify the model. This research aims to develop a process that requires less intervention from the designers and that can be carried out within the CAD/CAM environment. It proposes to replace the traditional iterative loop of modification and subsequent evaluation by a reversed process, which enables the designer to define its design intent and let the application reverse engineer geometry starting from the designers’ prescribed properties.

Reflection lines, curvature plots, compound fins are applications that use the theorems of differential geometry laid by French mathematician Frénet. His propositions presented in

Chapter 6 give a general definition of shape in terms of differential equations. These relate to the fundamental concept of a curve that is its frame and the speed at which this frame travels along the curve. This formulation, more generic than NURBS, allows a more systematic approach to curve optimisation. The applications developed with NURBS in Chapter 7 show the limitations of NURBS for achieving optimisation, compared with the applications developed with Frénet in Chapter 8.

The Third part of this thesis aims to define a framework for flexible optimisation within the CAD environment. Open architecture is the term used in paragraph 9.1 to refer to the exposure of a systems' functionality to outside applications. Investigations in these show that commercial turn-key systems do not comply with the objectives of this research. Rather a reconstituted framework is produced. The flexible optimisation framework that is proposed combines functionality from a geometric modeller, ACIS, a geometric editor, Open GL, ACIS MFC, and an optimisation capability, Flexo toolbox. All of these all have their own API's, except the optimisation toolbox which was provided as code only by *Tiwari* [99]. Typically geometric modelling algorithms are implemented in geometric modelling kernel, which exposes it's functionality through a programming interface.

This thesis is a multi-disciplinary research program involving different areas such as Design, Geometric Modelling and Optimisation. Figure 1 shows the interactions between these different fields constituting the scope of this research and serves the purpose of mapping the research framework for the forthcoming thesis. The topics introduced here are briefly outlined below and are developed further in following chapters.

> Design can be described as the process of establishing requirements based on human needs, transforming then into performance specification and functions, which are then mapped and converted (subject to constraints) into design solutions (using creativity, scientific principles, and technical knowledge) that can be economically manufactured and produced.
>
> Optimisation can be defined as the act of obtaining the best result under given circumstances.
>
> Geometric modelling is the process of defining the geometry of an object.
>
> API is the application programming interface that exposes a set of procedures that an application can call to carry out operations.

CAD is a workspace in which product designs are created, modified and edited.

Design Optimisation is the process of improving a design by manipulating the design variables.

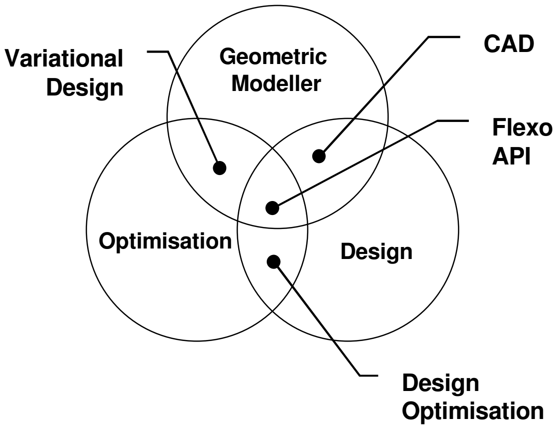


Figure 1: Research map [97]

Part I

Chapter 1 Design

Increasing global competition in the manufacturing environment is pushing companies to improve on product performance This is forcing manufacturing enterprises to seek more advanced technology for improving product specification at lower cost. Companies are introducing ‘design optimisation’ in their organisations as an attempt to improve their design process capability and thus stay competitive in the market place.

This chapter exposes some of the theoretical aspects of design science in order to establish some relationship between design and optimisation. To begin with, the investigations target a wide spectrum of product design and later narrow the scope down to curve and surface design.

1.1 Engineering Design

1.1.1 A Definition of Design

Design activities although performed for many centuries have not generally had any structure or organisation to them; or perhaps design wasn’t regarded as a discipline in its own right until after Second World War. With the advance of modern society and mass production, it became apparent that design as a process should be given more thought. Many attempts have been made to map its process with complex picture diagram models and even to establish some form of definition. This has proven a very difficult task since design is everything but an exact science and is left to each individual designer to interpret the philosophy of design within a specific context and from his or her own perspective. Several designers, engineers and researchers, from experience have expressed their views on the definition of or what they consider design to be. Some of these viewpoints are expressed by *Feilden* [47], *Finkelstein* [50], *Luckman* [71] *Archer* [5] and *Caldecote* [22] In general, certain key-words and phrases can be noted which have a strong bearing on design. These include: needs, requirements, solutions, creativity, constraints, scientific principles, technical information, functions, mapping, transformation, manufacture and economics. Taking into account all these words, design can be described as the process of establishing requirements based on human needs, transforming then into performance specification and functions, which are then mapped and converted (subject to constraints) into design solutions (using creativity, scientific principles, and technical knowledge) that can be economically manufactured and produced, *Evbuomwan* [42].

1.1.2 A Design Model

In the more specific context of engineering, design can be seen as a decision making process that involves an evolutionary process, where changes (improvements or refinements) are proposed to the current design in order to move to a better design, *Rzevski* [104]. As for every design problem there is an infinite number of options available (also known as design concepts), the aim of design optimisation is to provide solutions that satisfy best a given set of constraints, parameters and variables. In the design process, iterative in nature, a common line of thoughts emerges as designers move from an abstract problem definition to a fully specified product. These are the divergence, transformation and convergent stages of design, as formulated below, *Evbuomwan* [42] and pictured in Figure 2.

Divergence

This is the act of extending the boundary of a design in order to have a large enough search space. The divergent search approach aims to break the initial design concepts, while identifying alternative feasible designs. This chaotic phase is most productive in the initial stages of the design process.

Transformation

This is the stage of pattern making, high level creativity. The objective here is to re-structure the design thoughts into a more structured search pattern allowing convergence to a single design solution.

Convergence

The main objective of convergence is to progressively narrow down the design alternatives through a selection process. The end result of this phase is the reduction of the range of options to a single chosen design while avoiding setbacks and retreats.

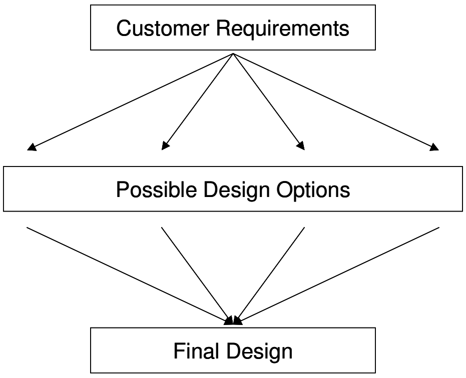


Figure 2: Divergent–Convergent design model, *Roy* [92]

1.1.3 Design Goals

Design goals can be defined as the purposes for design actions and decision taken in each step. They guide the choice of what to do at each point during the design process, *Mostow* [74]. Design goals represent one or more decision points from a problem solving point of view, and they define some of the dimensions of the design space, *Evbuomwan* [42].

1.2 Review of Design Problems

1.2.1 Design Problems Classification

There is in the world of engineering design an infinity of design problems, each one of them trying to achieve one singular goal. Engineers and designers face different situations; a classification of design problems is presented below, *Juster* [66], *Cagan* [21], *Sriram* [113], *Pahl* [81].

Routine Designs

These designs are derived from existing design prototypes with a common set of variables; the structure does not change. Here design plans and alternative solutions are known in advance.

Redesigns

These involve modifying an existing design to satisfy new requirements or improve their performances under new requirements. The end result of redesigns may also exhibit some aspects of creative designs or routine designs. Redesigns come in two categories, adaptive and variant designs, which are discussed below.

Adaptive, configurative or transitional designs are those that involve adapting a known system to perform a new task. They also involve improvements on a basic design by a series of ‘detail’ refinements.

Variant, extensional or parametric designs involve using a proven design as a basis for generating further geometrically similar designs of differing capabilities. Knowledge Based Engineering (KBE) systems are typical variant design applications. KBE uses library of predefined geometric entities or assemblies, which are automatically selected, and if necessary varied, accordingly to the design specification.

Conceptual designs

Non-routine designs, original or new designs are classified into innovative and creative designs.

With innovative design, new variables or features are introduced, which still bear some resemblance to existing variables or features. The decomposition of the problem is known but the alternatives are yet to be synthesised. In other situations, a recombination of the problems alternatives may produce a new design.

Creative design. In this case new variables or features are introduced, which bear no similarity to variables or features in the previous prototype and the resulting design has very little resemblance to existing designs. For creative design, no design plan is known.

1.2.2 Product Design Classification

The end result of any design process is a product or a system. Such product depending on the engineering discipline or domain, vary in one-way or another. Product variation also arises depending on the market segment, knowledge available, the design process and manufacturing capabilities. In the light of general constraints, products can be classified as either over constrained or under constrained. Depending on the customer demand and market competition, some products are considered as static or dynamic. These various forms and classifications are discussed below, *Clausing* [24], *Medland* [72]

Static Product Designs

Static products are those that demand remains stable and no changes to the product design are required. The design concept is already known from existing products, such products are considered as conceptually static.

Dynamic Product Designs

Dynamic products have a limited life cycle before the next generation supersedes them. Here development is focused on the product and the design process involves development of new radical and alternative designs.

Over constrained Product Designs

These products exist in the high technology markets. Here, the design process evolves around analysing alternative proposals until an acceptable solution is found. Over constrained products are subjected to several constraints. These include functionality, environment, performance, materials, manufacturing processes and cost, some of which can be conflicting one another adding further complication to the system.

Under constrained Product Designs

In the case of under constrained designs, the design activity is centred on bringing products in the market to satisfy market demands. There are usually not many constraints, and the designer has ample room for innovation. The focus here is usually on the product concept, materials and techniques, which are chosen to satisfy the required functions. Most industrial designs fall into this category, development is on aesthetics, ergonomics and functionality.

1.3 Constraint Based Design

Since the first CAD systems were introduced, there have been on-going attempts to put more ‘design’ into CAD including conceptual design, design embodiment, design for manufacturing, design for environment, etc. One way to characterise the process of mechanical design is to describe it as a process of constraint specification and satisfaction, *Thornton* [118]. Design can therefore be seen as a constrained optimisation problem that searches for the best acceptable solution. Consequently the rest of design can be seen as the search for a solution that best satisfies these constraints.

Numerous researchers also see constraints specification and satisfaction as a key issue in the design process. *Sriram et al.* [113] state “Design can be viewed as the process of specifying a description of an artefact that satisfies constraints arising from a number of sources by using diverse sources of knowledge”. *Serrano* [111] also refers to design as constraint object-oriented.

Researchers have studied a number of different approaches to constraint-based design. Most of these use of artificial intelligence techniques, which are particularly well suited to combinational problems defined with discrete variables.

In conceptual design, *Thornton et. al* [118] use a genetic algorithm in a software support tool for constraint processing in embodiment design (CADET). Here generic component libraries are used to automate the specification of design constraints.

In feature modelling, *Laakko et al.* [68] propose a method integrated with EXTDesign, an incremental design environment that combines solid and feature modelling, developed at Helsinki University. The proposed system aims to localise the problem of constraint solving and maintenance. The constraint algorithm used is based on local techniques to propagate changes in the constraint graph.

Yet another approach to constraint satisfaction is followed by *Buchanan et. al.* [20], who describe a constraint based modelling system called CDS Other applications use computer algebra techniques for geometric modelling in particular, the geometric algebra system developed at the university of Bath [15].

1.4 Evolutionary Design

Evolutionary design states that natural evolution is capable of generating new or evolved designs, evaluating these designs and optimising them as depicted in *Bentley et. al.* [9]. Only two aspects of evolutionary design are relevant to this thesis and are outlined below.

1.4.1 Optimisation of Existing Designs

The development of non-generic optimisation systems, capable of optimising selected aspects of existing designs is a common research area, *Parmee* [83]. Numerous examples of design optimisation exist in both academia and industry; many using GAs or other adaptive search techniques. Examples of such real life design optimisation problems can be found in *Rogero et. al.* [89]. The wide variety of optimisations problems tackled by evolutionary based techniques shows that algorithms could successfully perform optimisation tasks on many different type of designs. However such optimisation schemes all suffer the same drawbacks:

They can only optimise existing designs; none of these techniques would be capable of generating new designs.

They are all application specific; only optimising the type of design they are created for.

Further analysis of the type of optimisation described above is developed in Appendix 1, which details the use of optimisation in industry and their inhibitors *Roy et. al.* [95], [100].

1.4.2 Generic Optimisation of Designs

Generic design optimisation, aims to optimise more than one type of design with a single system. A generic optimisation toolbox consists of a collection of algorithms capable of optimising a different range of applications. Generic optimisation is not very common. However efforts to develop some form of generic optimisation toolbox include, *Culley* [29]. More recently, research at Cranfield University with *Tiwari* [99] has pushed in the area of generic optimisation and establishes definitions and classifications of optimisation algorithms. His effort has also attempted to define criteria for selection of algorithms. The selection criteria are based on the features of real optimisation problems. They define the typical scenarios in real-life optimisation, which in turn identify the ingredients of the 'tool box'.

1.5 Structural Optimisation

There are in industry many different techniques used to optimise mechanical product design; each optimisation technique looks at one particular aspect of the design. Design for assembly, design for manufacture and design for quality all look at the functionality and manufacturability of the product. Such optimisation techniques are popular in industry as they can be performed without complex mathematical analysis and are proven very effective. Also these optimisations are performed externally to CAD/CAM environment and often after the completion of the design definition.

Structural optimisation of mechanical components is an activity carried out within the CAD/CAM environment simultaneously with the product definition. In this paragraph, a classification of the different types of structural optimisation used for product development is presented, all of which optimise the geometry of components with various objectives. Structural optimisation is rapidly becoming an integral part of the product design process. Considering timing and budget constraints, structural optimisation yields a significantly superior design than the conventional trial-and-error approach. Structural Optimisation is the generic term to describe optimisation of geometric entities composing engineering products.

Structural design optimisation problems are classified into three main categories: Sizing, Shape, and Topology *Bremicker et. al.* [19]. Alternative classification, can be found in *Raasch* [88], here Sizing is described part of a more generic domain termed property optimisation. In general, the classification of structural optimisation is closely related to the choice of the design variables. Sizing variables such as cross-sectional area of a truss member, plate thickness or cross-sectional dimensions of a beam do not change the shape of a structure. However design variables that govern the shape or geometry of a structure are referred to as shape design variables. For example coordinates of nodes in solid models or curve/surface are typically shape design variables.

1.5.1 Property Optimisation

With this type of optimisation, some geometric items are transformed into a property, which can be changed independently with respect to the rest of the geometry. Examples of property include size or weight of a component. With property optimisation, the overall shape and structure of the component remain unchanged. This type of optimisation works on models such as beams and shells, which are reduced in one or more dimensions by using abstract mathematical models. For example, sheet metal structures are usually represented by surfaces, and thickness is a separate item. Due to this, the shell thickness can be changed independently to improve the functionality of the component. This optimisation is losing its popularity in industry; it is being replaced by more sophisticated CAD driven analyses like FEA.

1.5.2 Shape Optimisation

In shape optimisation, the geometry of components' structure changes from an initial shape to the optimum shape. Parameterised geometry variables are taken as design variables by the optimisation algorithm. The model is generally meshed with a FEA/CFD package for analysis.

1.5.3 Topology Optimisation.

In general, designs have fixed points, or “hard points”, which are defined by the engineering necessities. Topology optimisation aims to explore the different ways in which the hard points are linked. Topology defines the overall structure of a component or assembly. Such study in usually carried out at early in the design process.

1.6 Summary

A definition of design has been established as a divergent-convergent system of finding solutions that satisfy constraints. It is thought that the design process described above can be mapped onto evolutionary computing techniques to automate its iteration loop. Hence for optimisation, the top priority is the representation of the constraints that characterise the design. For the remaining of this thesis, the choice was made to look into the shape optimisation of free form curves. This choice is not arbitrary, because curves and surfaces also was also the centre of interest of the Flexo project and therefore some of the information gathered throughout the project could be used in this research.

Chapter 2 Optimisation

This chapter reviews some optimisation techniques available for engineering optimisation. An implementation of a GA (Genetic Algorithm) written in object-oriented language C++ is documented in Appendix 2. Reports on the current status of design optimisation in industry are presented in Appendix 1.

2.1 Optimisation for Engineering Design

In Chapter 1, design is described as an iterative and evolutionary process of changes leading to a better design for given specifications. This also applies to optimisation, which is described as the process of attaining a superior design, based on some pre-defined criteria, from a set of feasible alternative designs. However, as explained in *Mussa* [76], there is a precise difference between the process of improvement and the search for an optimum. Improvement involves subtle human decisions based on the subjective notions of goodness and badness, which is usually defined by a mathematical objective function. The search for optima is frequently seen as a convergence method to find the peaks (minimum or maximum) of a mathematical function representative of the objective, i.e. the best solution to a problem, Figure 3.

Thus, optimisation is defined as search process through the solution space driven by a human like decision-making process. The first aspect of optimisation is iterative in nature and can be carried out either manually or automatically using search algorithms. Furthermore it relates to the “Divergent and Transformations” phases of design discussed in paragraph 1.1. The second aspect of optimisation aims to map the designers’ intent in order to represent the design goals with in mathematical function, which is also called objective function. Also, proposed solutions are evaluated against the objective and assigned a fitness value, a criteria upon which the solution is selected or discarded. As for design, this is the converging agent in the optimisation process.

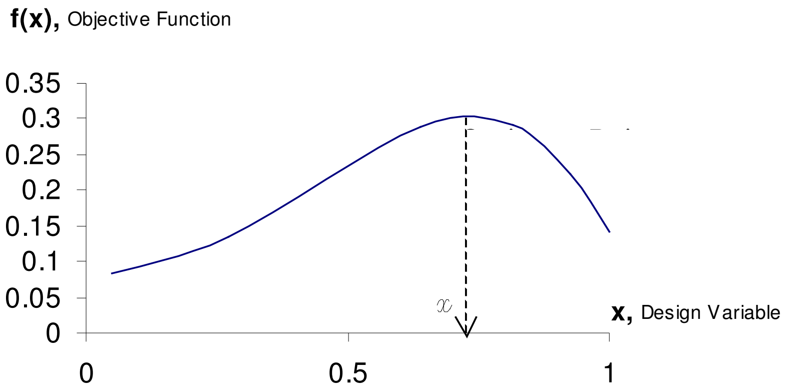


Figure 3: Objective function optima (peak)

2.1.1 Optimal Problem Formulation

As previously discussed, optimisation aims to explore and evaluate a variety of design solutions within the range of feasibility. Since an optimisation algorithm requires comparison of a number of design solutions, it is usually time consuming and computationally expensive, hence the need to formulate the design problem in a format suitable for an algorithm. The optimal problem formulation is the key to achieving competitive product design. Since the objective in a design problem and the associated design parameters vary with products, different techniques are used to define the mathematical model of the optimal design problem. An outline of the steps typically involved in an optimal design formulation process, Figure 4, is proposed by *Deb* [37]. Components of this format are discussed below.

The first step in the formulation is to realise the need for optimisation in a specific design problem. Next, the designer selects the associated design variables. The formulation involves other considerations such as constraints, objective function and variable bounds. These terms are developed in the subsequent paragraphs. The last step of the process deals with the selection, for a specific problem, of an appropriate optimisation algorithm. Literature on this particular aspect of optimisation includes *Roy et. al.* [94]. As shown in Figure 4, there is a hierarchy in the optimal design process, however each aspect may be influenced by the others.

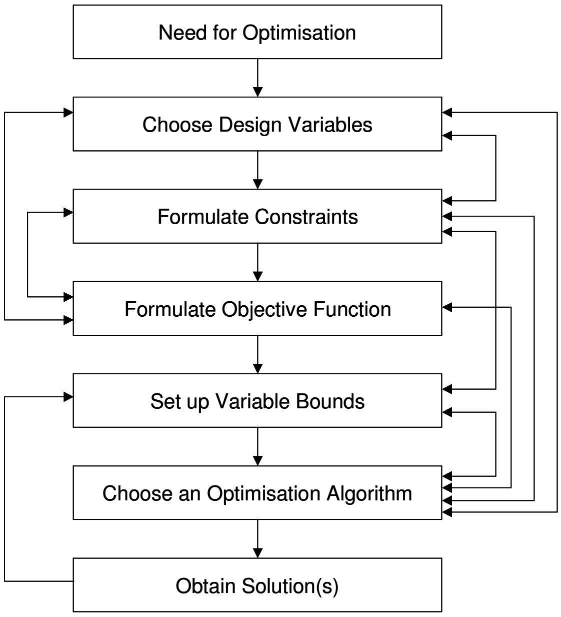


Figure 4: A flowchart of the optimal design procedure [37]

Design Variables

Identifying the design variables is the first step in the formulation of the optimal design. Design variables are numerical quantities, which define the design solution and values vary within bounds during the optimisation process. A design problem usually involves many design variables, some of which are highly sensitive to the solution output. The least influential variables are called design parameters, as they often remain constant during the optimisation process. Prior knowledge may dictate on the choice and ranking of the design variables. However, it is important to understand that the efficiency and speed of the algorithm depends to a large extent on the number of variables. The outcome of the optimisation procedure may indicate whether to include more design variables in a revised formulation or to replace some initially considered design variables with new variables or parameters. For a more systematic approach, performing sensitivity analysis on the input-output system gives accurate feedback information on an adequate choice of variables, Figure 5. Solution sensitivity analysis varies incrementally the design variables independently or a group of variables (other parameters remain constant) as input and returns response fitness values as output. Graphical presentation of the data as well as numerical analysis might ease interpretation. Sensitivity analysis is used later in this thesis to validate optimisation algorithms.

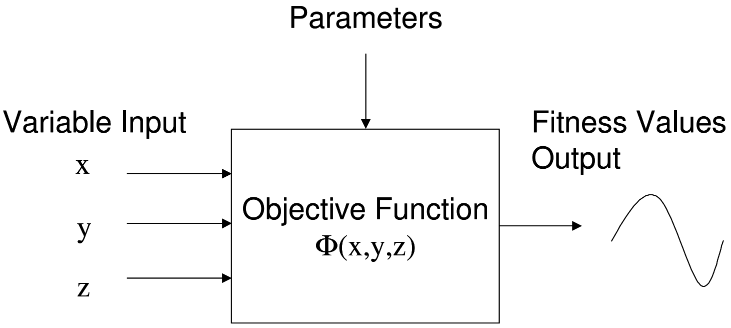


Figure 5: Optimisation system sensitivity analysis

Design Constraints

Constraints represent some functional relationships among the design variables and other design parameters. Design constraints are conditional restrictions that must be satisfied in order to produce an acceptable design solution. The nature and number of constraints to be included in the formulation depend on the user and the nature of the application. Constraints can range from simple conditional statement to complex mathematical definition; in which case some mechanism to calculate the constraints must be provided to the optimisation algorithm, *Deb* [37]. For example for sizing of complex mechanical structures, a finite element processor is often required to compute the maximum stress load in the structure. From an optimisation point of view, *Deb* [37], constraints are of two types: Either Inequality or equality constraints. Inequality constraints state that the functional relationships among the design variables are greater than, smaller than or equal to, a set value. These types of constraints cover most of the engineering problem range. On the contrary, equality constraints state that the functional relationships among the design variables are equal to a set value. The later is the most restrictive type of constraints and should be avoided as much as possible.

Objective Function

Optimisation requires evaluating every design solution produced. To do so, an evaluation function representative of the design, which is defined in terms of the design variables and other design parameters, needs to be formulated, *Deb* [37]. In engineering problems, most design objectives are quantitative (cost, weight, life cycle) and are mapped in a mathematical form. However some design aspects such as aesthetics are difficult to quantify since they are subjective criteria. Qualitative criteria are used to formulate the objective function where exact mathematical formulation is not available *Roy et. al.* [92].

Moreover, in real life applications, more than one objective needs optimising simultaneously, *Deb* [36] To date multi objective optimisation methods are still in their early stages of development but are the centre of attention of many research programs. An extensive literature survey in the field of multi-objective optimisation is available in *Roy et. al.* [94].

Fitness Function

The fitness function is used to accommodate the objective function with the optimisation algorithm own objective when there is a conflict of interest. For instance algorithms that are set to converge to a maximum conflict with optimisation problems that are solved by finding a minimum. In this situation , the invert of the objective function can will phase them if the division by zero if carefully avoided. Adding one to the denominator will just do that and set the fitness function definition in the zero-one range. Other operations can be made to the objective function, there is an example later in this chapter that illustrates that transformation of the objective value improves optimisation performances.

Fitness Function Mapping

Mapping the fitness function is showing fitness score against variable input in a graph. For one variable the map is a curve and for two variables it is a surface. For a greater number of variables, one need more dimensions that are not easy to represent graphically. The map is used for uncovering the nature of the objective function. Paragraph 2.1.2 reviews some optimisation problem characterised by their objective function.

Variable Bounds

Variable bounds concern the design variables definition range or feasibility range. In general, all design variables are restricted to lie within the minimum and the maximum bounds (upper and lower limit). Setting wide variable bounds expands the search space and consequently enables the algorithm to potentially find more optimum solutions, particularly in the case of a multi-modal objective function, which is defined in paragraph 2.1.2. The down side is that it is more computationally expensive. A compromised setting can be reached by gradually narrowing down variables bounds, while insuring that the optimum values always lay within that range. An alternative is to reinitialise the algorithm initial values with an intermediate solution. This will free the algorithm from variable bound restrictions.

Convergence Graphs

The purpose of convergence graphs is to examine the population's history in a chronological order. For each GA run, and each member of the population, the fitness function is recorded. The aim of such graphs is to judge on the optimisation performance in converging by simple visual check. A plot of fitness values against GA runs is presented in Figure 6, this example shows a population that is converging steadily to an optimum value. The drop is constant from the start to two thousand individuals. After that point the algorithm does not converge any more, which means that the optimisation loop can stop.

It is noticeable in the graph, Figure 6, that there is a concentration of individuals around at the lowest values; This band highlighted below the cloud is called the Pareto front. It presence signifies that the optimisation algorithm is doing well.

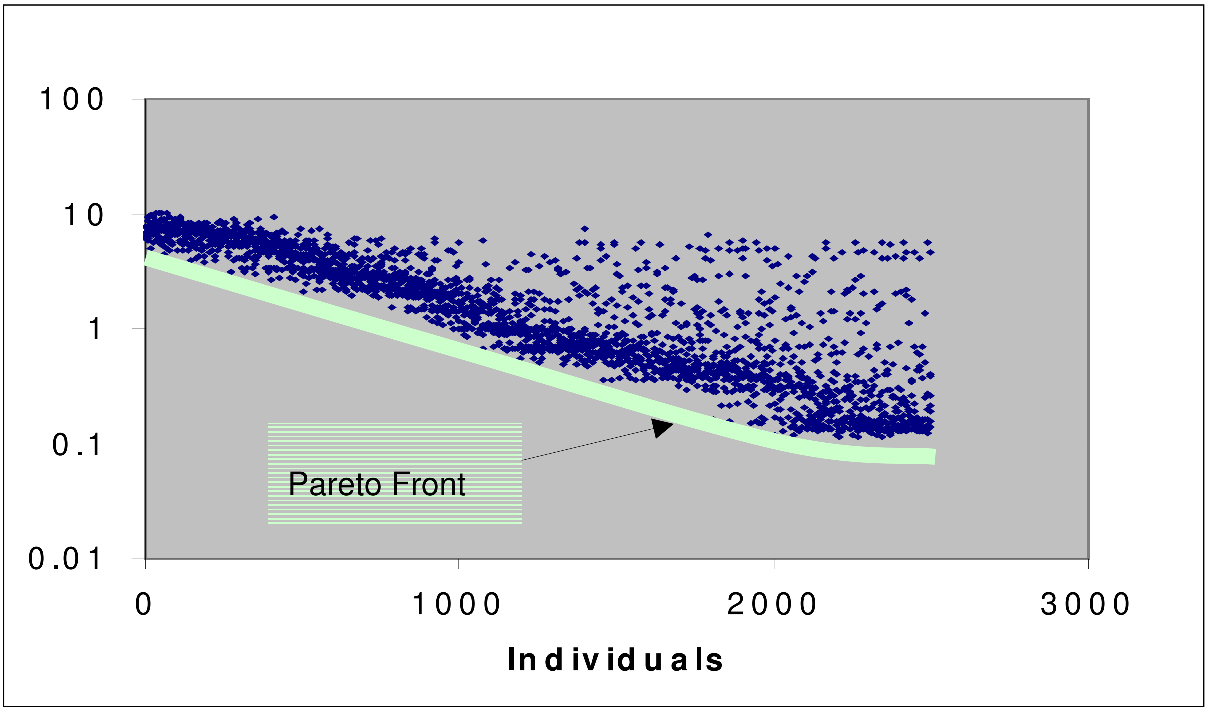


Figure 6: Population convergence graph

2.1.2 Optimisation Algorithms

The above discussion revealed that the formulation of the optimisation problem in a mathematical form and subsequently in computer code depends to a large extent on the nature of the design. This has pushed researchers and developers to create a multitude of algorithms capable of dealing with real life design problems. Features of real life optimisation involve multi-variable, multi-modal, multi-objective and constrained optimisation. For more details on features of real life optimisation, refer to *Roy et, al.*[94]

Single-variable

The simplest form of optimisation algorithm is single-variable. These types of algorithms work with one-dimensional problems (characterised by a unique design variable). Although the majority of engineering problems present more than one variable, single-variable algorithms are used to conduct a unidirectional search method in a multi-variable problem. This is particularly useful to carry out sensitivity analysis on design variables.

Multi-variable

Multi-variable algorithms work in a multi-dimensional space. Here more than one variable are varied simultaneously and solutions come in a set of multiple variables. The case study on curve optimisation reported in this thesis is an example of multi-variable optimisation design, it takes 3D point coordinates, 3D vectors input and output. In optimisation jargon, a so-called "individual" contains a set points which themselves contains the conventional three coordinate variables *x,y,z*. If the geometry has four points, the optimisation algorithm will input twelve variables.

Multi-modal

In the previous section, an example of a single peak function was presented in Figure 3. This type of function is called uni-modal. Wherever there is more than one peak, the function is called multi-modal. Multi-modal optimisation can be defined as the problem of locating several good solutions in the search space. An example of a multi-modal function is shown in Figure 7.

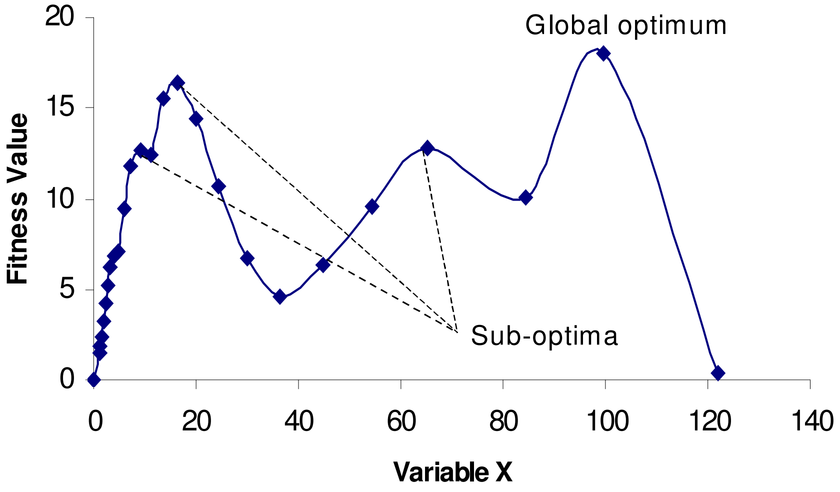


Figure 7: A multi-modal function

Optimum points can be localised by examining the values for which the gradient is equal to zero. In presence of a multi-modal objective function, we are not only interested in finding just the best solution (global optimum *Deb* [37]), but as many as possible (sub-optimum). The reason behind this is that sub-optimum solutions may present some interesting design properties that should always be taken into consideration. The sub-optimum solutions are considered as good solutions in a multi-modal function, *Roy et. al.* [91].

Multi-objective

Other optimisation problems include the optimisation of several objective functions simultaneously. This type of scenario is called multi-objective optimisation. At present, the industrial use of optimisation algorithms is limited mainly to problems involving maximisation or minimisation of a single measure of performance, or objective. This prevents them from handling many real-world problems since most of them involve multiple objectives, which should be optimised simultaneously, *Deb* [34]. This has encouraged the growth of research in the field of multi-objective optimisation using evolutionary algorithms *Roy et. al.* [94].

The principles of multi-objective optimisation differ widely from those of single objective optimisation. In a multi-objective optimisation problem, there is more than one objective function, each of which may have a different individual optimal solution, *Steuer* [115]. This gives rise to a set of optimal solutions called non-inferior, non-dominated or Pareto optimal solutions. These solutions are located on the boundary of the region containing the feasible solutions. In presence of multi-objective functions, the optimum point lies in the region of intersection of the functions Pareto fronts. Figure 8 below shows an example of a multi-objective optimisation problem with two objective functions.

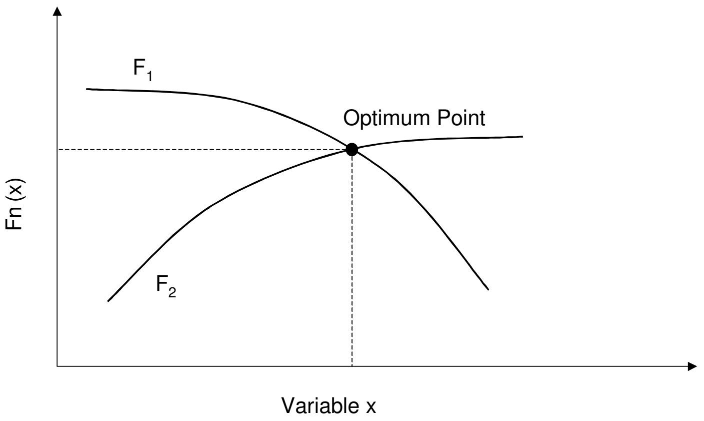


Figure 8: Multi-objective function

2.2 Adaptive Search Techniques

Efforts to model, algorithmically, the basic evolutionary principles (population, self-replication, variation, and selection) go back to the 1950s in the *Handbook of evolutionary Computation*, *Bäck* [6]. Since the 1980s, with the advance of computers, a new class of search techniques, called adaptive (or stochastic) search techniques, has been developed. The two best known are genetic algorithms (GAs) and simulated annealing (SA). Both techniques are loosely based on processes that happen in nature. These two algorithms have been applied to numerous fields as domain-independent optimisers. Both techniques are robust and good at finding a global minimum in large complex search spaces, *David* [31]. In the context of this research, the focus is on GAs only, due to resource limitations on the one hand, and also since development work in the area of GAs was made available for use in a case study involving integration of flexible optimisation within CAD/CAM environment.

As the name stochastic implies, adaptive search techniques have a random aspect to them. They have a reduced chance of converging to a local optimum because the searches are not restricted to finding a better solution for each iteration. This diverging property is particularly suited to design problems, which are, as discussed previously, divergent-convergent in nature. Adaptive search algorithms contrast with traditional optimisers that present the inconvenience of finding the optimum nearest to the starting point irrespective of whether it the global optimum or not, *Thornton* [118].

2.3 Genetic Algorithms

Genetic Algorithms (GA), were first developed by John Holland at the University of Michigan, USA, and were further developed by his colleagues and students. The goals of their research were firstly to abstract the adaptive process found with natural species, and secondly to implement the findings in computer software that replicates the phenomenon of evolution. The basic implementation of GAs is designed to mimic the theory of evolution propounded by *Charles Darwin* in *The Origin of Species*. He explains the evolution of species with a natural, but yet ruthless, process of selection and survival of the fittest. Based on this model of evolution, GAs are capable of finding good solutions to a problem bounded by constraints. The basis of a new branch in evolutionary computing was formed from the experiments conducted by *Holland*. Later, GA applications were used for engineering purposes such as machine learning *Goldberg* [55].

In general, GAs are robust techniques that can handle large numbers of parameters as well as large search spaces. They are often described as providing a good balance between exploitation and exploration of search spaces. Since its origin, GAs have been applied successfully in many fields of science. The prime reason for their success is that they consistently outperform traditional methods.

2.4 The GA Process

As previously explained, prior to performing optimisation, the problem has to be formulated in a format suitable for optimisation algorithms. This format is defined in terms of design variables, constraints, objective function(s), fitness function, and variable bounds. With the formulation of the optimal problem, the optimisation problem has moved from a high level abstract definition to a formal mathematical representation.

At this stage, the formulation is generic, i.e. algorithm independent. This paragraph presents an implementation of a simple GA using the optimisation problem formulation structure.

The tasks performed in a typical GA cycle loop, shown in Figure 9, are evaluation, selection, recombination and mutation. These terms are discussed later in this paragraph. It is important to notice the point of entry in the loop; an initial random population is generated and evaluated outside the GA loop. There is an issue over the generation of random numbers that holds potential for arguments. In the context of this research, in-built pseudo random generation number module *rand()* along with *srand() p*rovided with most C/C++ compilers is thought to be satisfactory. The initial generation is evaluated before entering the GA iteration loop. The cycle begins with selecting the mating parameters (chromosomes) for the GA operators (crossover, mutation) among the individuals constituting the initial population.

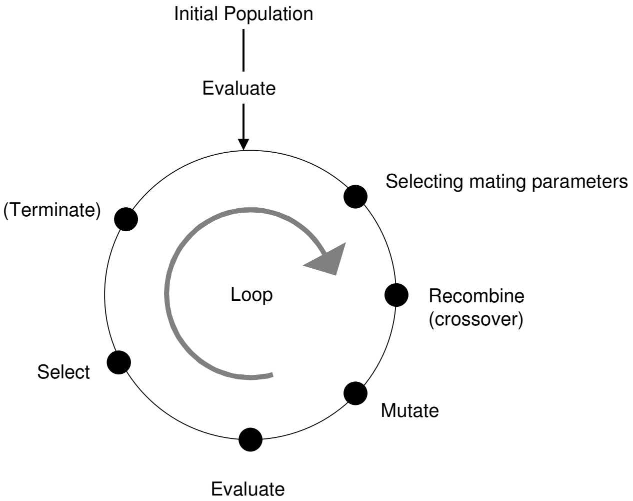


Figure 9: The basic loop of genetic algorithms

Binary Coding and Chromosomes

In most optimisation algorithms, the variables being optimised are maintained in an array labelled subjectively with the object it relates to. Also all objects should be given a tag to allow their identification.

$$X=\{X_1, X_2,\ldots X_n\}$$

In genetic algorithms, array of parameters, X, are often developed as binary strings (or list of real numbers) of length *nL,* where n is the number of parameters and *L* is the number of binary bits used the represent each value X_i. In GA terminology, an array of binary coded variables is called chromosome. The term was borrowed from genetics because of the analogy with nature where inherited characteristics of living things are encoded in their genes. An example of variable binary coding is shown in Table 1.

	Variable 1								Variable 2							
Decimal Value	45								78							
Binary Coding	1	2	4	8	16	32	64	128	1	2	4	8	16	32	64	128
BinaryValue	1	1	1	0	1	1	0	0	0	1	1	1	0	0	1	0
	Gene 1								Gene 2							
	Chromosome															

Table 1: Chromosome binary coding

For each individual, the fitness value is evaluated by firstly decoding the individual chromosomes into separate parameters and secondly feeding the evaluation function with the parameters values obtained from the GA

Binary coding (and decoding) involves computer-rounding errors and requires careful examination. Excessive coding precision is unnecessary and computationally expensive. Typically precision of a thousandth of the unit is adequate. This is also consistent with engineering applications that are in the worst case bounded with two hundredths of a millimetre tolerance. However, experiments on GAs in Appendix 6 show that special care must be taken in the presence of small numbers as rounding errors could mislead the algorithm into inaccurate solutions.

GA Operators

GA operators comprise of crossover and mutation, which operate on the individual chromosomes combination, (parents) to create new chromosomes (children). For this reason GA operators are also called a mating pool. These transforming operators enable the search algorithm to explore new points in the search space while converging towards solution peaks.

In simple terms, with crossover operator, portions of two individual strings are swapped. As a result the offspring inherits some characteristic of each parent making up a new individual. Not all chromosomes swap genes, typically, the crossover probability should range from 60% to 100%. In the case of zero crossover probability, the offspring are a strict replica of the parents.

In addition to crossover, some algorithms include mutation. This operator randomly changes values in the chromosome string. This is achieved by changing binary 0 to binary 1. This supplement operator, more random than crossover, is introduced to boost the exploring properties of the search algorithm. The mutation probability should be kept in the region of 0% to 10% depending on the nature of the search space. Any higher mutation probability would introduce too much diversity in the population, hence slow down convergence. The opposite would resume the algorithm to a systematic exploring of the search space. There are no standard values that guarantee GAs to perform best, both in exploring and converging. It is up to the user to try out different settings to reach a compromise. Experiments on GA operators are included in Appendix 6. On a grand scheme of things, it seems rather ironic to see a capability such as optimisation, whose sole purpose is removing trial and error from a given system, actually adds more to it; or perhaps one should imagine optimisation for optimisers.

Selection Wheel

Prior to the selection process, each individual is evaluated with the help of the objective function. Genes are decoded into function parameters and passed to the evaluation function for calculation. From the value returned by the evaluation function, the fitness value that characterises each individual value with respect to the design objective is obtained. Most GAs are coded in such way that the individual with the highest score is the fitness. In other words, GAs converge towards the maximal. Simple mathematical operations are applied to the fitness value, in order to match the algorithm scheme (minimising or maximising).

The roulette wheel selection method simulates a wheel on which the sectors size is proportionate to each individual's fitness score. An individual's selection is attained by spinning the wheel and picking the individual on which the selector pointer has stopped, *Mussa* [77]. Individuals with the highest fitness score take a bigger share of wheel therefore have a greater chance of being selected for reproduction. This method based on probability laws replicates the process of survival of the fittest described by *Darwin*.

Fitness Boosting

Since individuals are selected upon their fitness relatively to the other individuals among the population, there is competition between individuals to occupy the widest sector on the wheel. A non-sensitive objective function (low gradient), puts the GA in a difficult position, high variable variation translates into low fitness value variation. This means that two individuals with close fitness values would have very different variable values. This type of scenario is highly undesirable as no convergence to optima is possible. Applying mathematical functions to the fitness value is a possible answer to the problem of flat objective functions. The desired effect is boosting the fittest individuals while diminishing the others. This can be achieved by using a function with exponential gradient. An example of objective function boosting is presented in Figure 1. Here, a power function is applied to the objective function.

$$\text{fitness value} = \text{power}(1/(1 + \text{objective value}), n)$$

The resulting effects on the solution space are presented in Figure 10, where power indices are gradually increased form 1 to 10. As the power indices increase, the solution band becomes narrower with a distinct peak. This gives the GA better chances of converging to an optimal solution. The example provided here is a single-objective, uni-modal, and one-variable objective function used in a curve optimisation application. The technique described above has been developed and applied in the context of this research; further enhancements and results are discussed in Appendix 6.

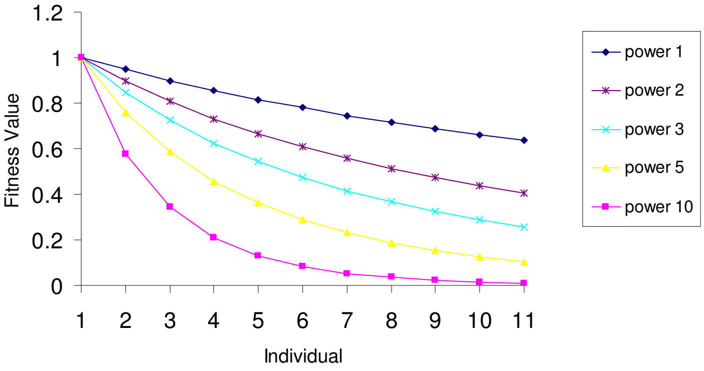


Figure 10: Objective function boosting

2.5 Summary

Optimisation has been defined as an iterative process of search of optimum through the solution space. Optimal problem formulation is characterised by the design variables, variable bounds, design constraints and one or several objective functions. Based on this description, algorithms are classified in the followings: Single or multi objective, uni or multi modal and constrained. Genetic algorithms are a kind of adaptive search technique, which mimics the evolution of species described by *Darwin.* Its process can be broken down in the following steps: Selection, mutation, crossover and evaluation.

Experiments on curve optimisation carried out in this research make a real-life test bench of GAs and evolutionary computing in general. The results will provide indications on the level of efficiency and accuracy of GAs. However, some predictions can already be made. GAs are used to give approximate answers to optimisation problems which cannot be solved by exact mathematical means. In other words there are the last option when every thing else has failed. Because heuristics return approximate answers, extreme care must be taken not falling into the inaccuracy trap. If the evaluation function formulation is not correct, the algorithm will run regardless and give an answer. The validity of the answer must be examined with most care; it can be tempting to blame inaccuracy to justify unexpected answers. So it utmost be ensured that the evaluation function is really the representation of the design constraints. One way of doing that is performing sensitivity analysis on problems which solution is known in advance.

Chapter 3 Reverse Engineering of Curves and Surfaces

The shape of finished goods and manufactured products exhibits a marked emphasis on smooth shapes and free flowing contours. Whether it is cars, audio equipment, cameras, ergonomic furniture or injection moulded plastic products, all have smooth contours in their design. However, the production of manufactured objects from such shapes is not an easy process. Most CAD software provide means to create free form contours for the design and manufacture of artefacts, but it is still a laborious process. One of the major bottlenecks in current CAD systems is the inefficiency of representation and manipulation tools for the design of free form, sculpted, three-dimensional shapes. This is forcing designers to use physical clay models and templates for generating curves and surfaces which are digitised with 3D coordinate-measuring machines (CMM) and 3D scanners to gain a computerised geometric definition. Curves and surfaces modelling specialised software such as *SURFACER* [108] provide adequate capabilities for transforming scanned data (point clouds) into CAD representation.

3.1 Design Cycle

The process of aesthetic design differs from company to company, depending on the styling job, resources constraints, equipment and tools. Nevertheless a general workflow can be drawn, in which all the individual steps can be identified as activity centres. There are three main activity centres involved in the design cycle shown in Figure 11; these are styling, designing and production. The styling department is in charge of creating a concept model in accordance with the requirements expressed by other departments such as marketing. The model is manually digitised and passed on to the design department where the data is cleaned before being passed on to the surface developer who produces a replicated CAD model. This model is then reported to the styling department for evaluation. The approved CAD model is then passed on to production department, which checks the model for manufacturability, compliance to regulation and tolerance. The final model is frozen and is used as a master copy for tooling.

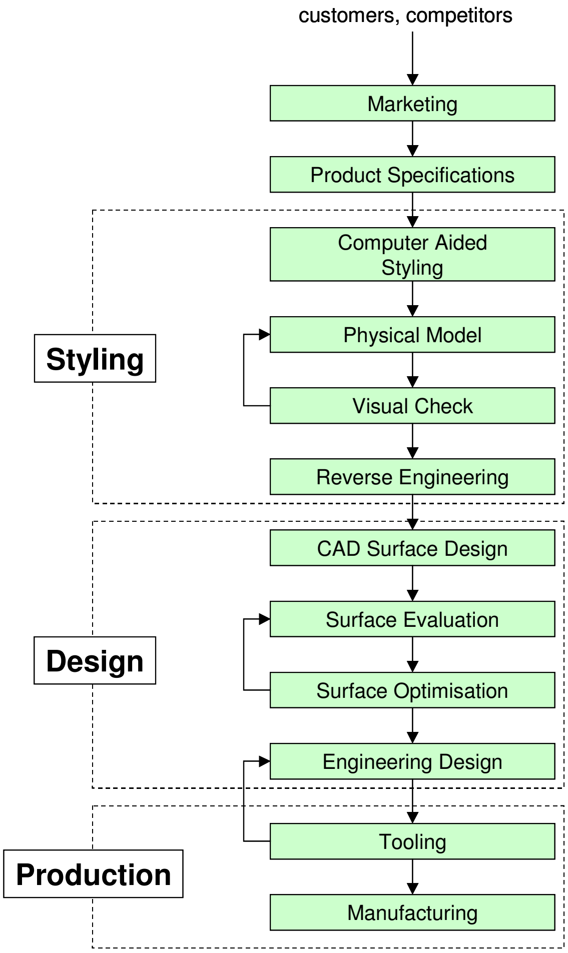


Figure 11: Surface design cycle

3.2 Styling

Concept models are first explored by means of free hand sketches showing general lines and volumes. More and more Computer Assisted Styling (CAS) systems such as ALIAS are used simultaneously in the styling process. They allow the designer to sketch a vehicle concept starting from the “hard points” and onto the outer skin definition. From the CAS model, a first scale plastic model is milled for down stream activities such as costing, digital mock-up, and aerodynamic evaluation.

Also because other departments such as body engineering design require design data early on, a full-scale model is simultaneously further developed. Traditionally, working materials are synthetic clay and hard foam, allowing direct sensory connection between stylist and model. Surface quality is assessed manually by running a flat hand on the surface or visually with aluminium foil to visualise the reflection lines under studio lights. Figure 12 shows a finished clay model from the Nissan Micra [79] under tube light. The bright area pointed by the arrow is a reflection line. The shape of this reflection is an indicator of the quality of the surface.

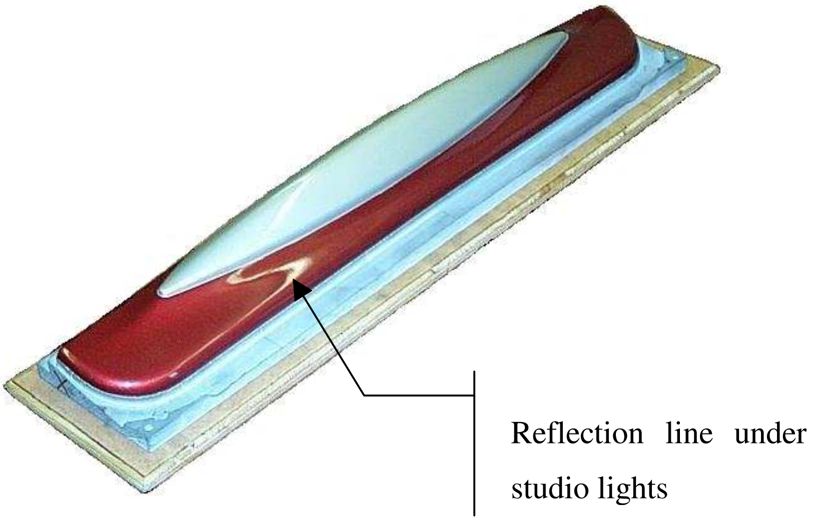


Figure 12: Clay model from Nissan Micra [79]

Manual shape definition is constantly receding because physical models are expensive and from a process point of view, CAS offers the possibility to obtain a model that is that is modifiable and that can be evaluated quickly both aesthetically and technically. Accordingly the designer can develop a greater number of design alternatives, thus enhancing the efficiency of making design decisions.

3.3 Reverse Engineering

Reverse engineering in its complete definition refers to the process of creating a completely engineered prototype that is a clone in form and function, from a physical part, *Sinha* [112]. In the context of this research, the definition of reverse engineering is to recreate a CAD model from a set of measurements. Such process is found in design studios where reverse engineering is used to translate the stylists' clay work into a computer accessible representation. It is generally composed of four activities, which are as follows:

Digitisation → Translation → Smoothing → Surface creation

3.4 Dimensional and Geometrical Metrology

Metrology is used in reverse engineering to digitise model by means of 3D measuring machine; which is undoubtedly part of manufacturing technology rather than design. 3D sensors are broadly classified into two types, contact sensors which touch the object in order to take measurement of it (coordinate measuring machines) and non-contact sensors which use laser light beams (3D scanners).

Coordinate Measuring Machine

Coordinate measuring machines are not only used for dimensional and geometrical accuracy inspection, it can also produce accurate measurements of the shape and position of any complex work piece features.

In brief, a coordinate measuring machine is a work bench mounted with three orthogonal axis gantry. These axis, termed X, Y, Z represent a 3D Cartesian coordinate system. The frame holds a finger probe that makes contact with the measured piece. The displacements of the probe are registered with a digital length measuring device and transferred to an electronic control cabinet (more likely to be a PC by today's standards). These contact sensors collect data by making contact with the part at every measurement location. Hence, data collection is a long and cumbersome process.

Light Based Sensors

Other coordinate measuring machine include 3D scanners, they provide a fast and easy way of acquiring 3D information about objects. These hardware equipment produce accurate, dense point measurement very quickly and are more suited to scanning large surfaces than touch-probe sensors.

Digitising Strategy

Digitisation of models is not a trivial task as the choice of strategy heavily depends on the down stream application. Some examples of digitisation applications are given with a brief description of the digitisation strategy.

> Copy Milling: A large number of points on the surface are acquired to facilitate direct linking to a CNC milling machine via a CAD module. In this case only the dimensional accuracy is required and is inherent of the CMM capability. Mostly, the strategy is to define different measuring areas scanned in lines or 2D sections.
>
> Reverse Engineering: In this case the objective is to gain the surface description of the model. The final accuracy is a combination of two elements.
>
> Geometrical Accuracy: Defined by the intrinsic characteristics of the surface such as the continuity order C(0), C(1), C(2), and smoothness. This play a crucial role in reverse engineering for A-class surfaces such as car bodies as it will ease the reconstruction of the surface for engineering design

An example of a model digitisation performed at NTCE [79] is shown in Figure 13. The digitisation is performed using a 3D measuring bench mounted with a probe. Each point is a 3D coordinate measurement. These measurements are the basis for producing the curves necessary for the skinning operations.

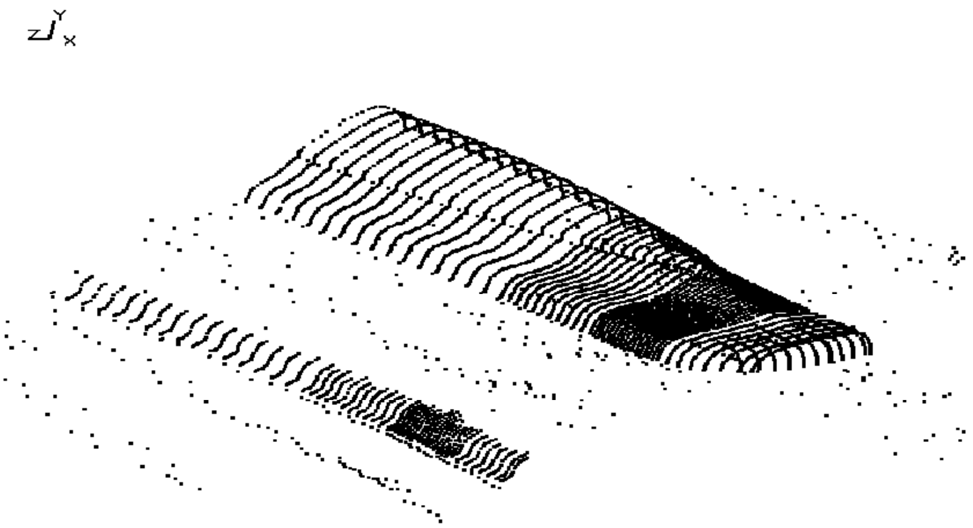


Figure 13: Point cloud from model digitising

3.5 Point Processing

As explained in the previous paragraph, the digitising quality is dependent on the clay surface finish, the CMM capability as well as the operator skills and experience. However, point-cloud data always contain undesirable irregularities. The first task consists of removing the stray points, i.e. the points that stand out from the series of measurements. Typically the following tolerances must be met: 0.3mm for external surfaces (A-Class) and 0.5mm for trim surfaces (B-Class).

There is a significant drop in data quality between the clay surfaces generated by stylists and computer reconstructed surfaces. This gap shown in Figure 14 is the result of digitising surface reconstruction because points do not carry any information on the internal structure of the part. The FIORES project, *Dankwort* [30] has addressed these issues and proposed a revised workflow for reverse engineering. The results suggest that physical models are avoided in favour of 3D screen visualisation or virtual reality models. This radical approach presents the advantage of retaining a CAD definition through out the entire design process. The limitations are that virtual reality can never match physical models properties.

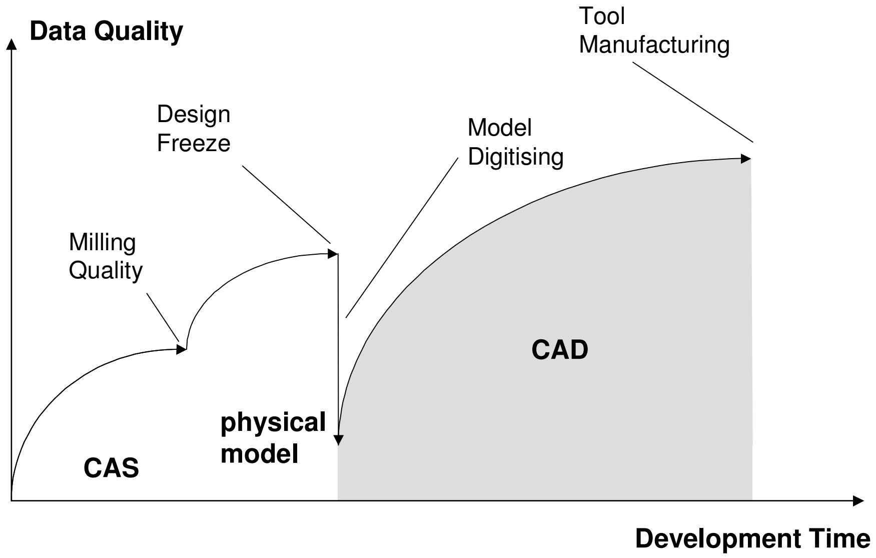


Figure 14: Loss of data quality in the design process, *Dankwort* [30]

3.6 Surface Generation

This activity consists of reconstructing model surfaces from the digitised points. Surface reconstruction, falls into Computer Aided Graphic Design (CAGD) and has vast literature and an exhaustive review would make this thesis look like an appendix. In basis terms, the main techniques include with Spline interpolation, Sweeping and Lofting, Triangulated Surfaces and NURBS.

Skinning

This is the most conventional surface reconstruction method. Surfaces are generated in three steps:

Curve generation

Smoothing curves

Skinning

First cross-section points are interpolated producing initial low quality curves that contain inflection points, peaks and troughs. The number of curves is reduced to the minimum number required to define the final profile. Sections of profiles between inflection points are initially approximated with known perfect geometries (segment, circle and conic). If that cannot be accomplished, the designer "splines" the particular profile section. Now begins the long surface optimisation process. Profiles are smoothed by freeing points, curvature and tangency. Most surfacing tools are provided with a curve smoothing capability. Quality is also assessed by looking at curvature profiles. After repeating the process for all the profiles that define the skeleton, surfaces are created with "lofts" or "Sweeps".

NURBS Patches

This process described above constitutes a traditional scenario in the automotive industry. However, the trend in surface design is moving towards a more flexible approach to deformable objects. The NURBS surface representation presents such characteristics. Instead of using Spline curves as guides for lofts and sweeps, a surface patch is directly applied using point cloud data as nodes. This approach is said to be more efficient than the traditional approach.

3.7 Surface Check

A number of quality checks are performed on the finished surface. Some of these are listed below.

Studio lights running along the surface

Offsetting, if no distortion appears within 1m offset, the surface is considered satisfactory

Reflection lines, see Figure 15

Colour code profile for curvature

Using these tests, the designer knows whether the surface needs further development. If so, modifications are applied to the skeleton curves, followed by skinning and so on.

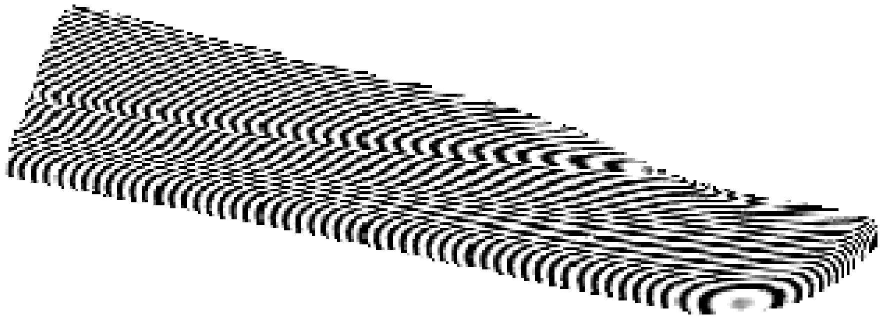


Figure 15: Surface quality inspection with highlight lines

3.8 Surface Optimisation

Curve and Surface optimisation can be classified in shape optimisation in the sense that it changes the surface of a structure. However it differs in many ways: Surface optimisation does not attempt to modify the internal structure of the component but rather focuses on the surface properties. The objectives of the optimisation are the aesthetic aspect of the design. We can distinguish three techniques used for surface development where optimisation algorithms are used. These techniques are surface reconstruction in reverse engineering, reflection line, and smoothing and fairing of curves and surfaces.

Surface reconstruction also referred to “reverse engineering” is the creation of a computer model from data points obtained from an existing object. Optimisation techniques such as Genetic Algorithms are then used to improve the model definition. Inspection of reflection line patterns is a standard way to check the quality of free form surfaces. Reflection lines or silhouette lines are created by the outline contour of the surface seen from a specific angle. Surface smoothing is a set of techniques used to evaluate the quality of surfaces in terms of curvature and oscillation. The design of surfaces involves multiple criteria, which are governed and restrained by a number of geometric constraints. Surface optimisation techniques are widely used in the automotive industry for the design of ‘A class’ surfaces including body shells and other exterior components.

CAD/CAM/CAE systems often obtain suitable designs through an iterative decision making process. The process is time consuming and involves lot of human intervention. This method of design improvement is dependent on the skill and experience of the designer, resulting in a need for a flexible optimisation approach.

Linden and *Westberg* [69] presented the FANGA (Formela ANGle Analysis) technique and analysis method developed by Saab-Scania Aircraft division to optimise and assess the quality of surfaces used in their aircraft and car body design. The fundamentals of which is a extension of Saab-Scania earlier development of FORMULA, given in *Einar* and *Skappel* [40]. This enables a surface to be refined by the response of a set of angles and directions, which are easily computed from the surface model. These are then used in a standard optimisation technique to compute the necessary changes to the parameters of the surface to produce the required qualities.

Earlier research is reported by *Kaufmann et al.* [67], with regards to car body design. They use an algorithm to define reflection lines and family of planes on the surface to represent surface irregularities. From which new reflection lines are produced and smoother spline and surface are obtained. This method has been used in the CAD/CAM system Syrko at Daimler-Benz, with successful results in the improved quality of body designs, coupled with the emphasis on time efficiency, this method is limited in simplicity and robustness that is required.

Watebe et al. [119] have introduced a methodology to generate a suitable shape automatically using genetic algorithms (GA), and by the application of Free-Form deformation (FFD) technique [110]. Related research is reported by *Weinert et al.* [121] at the university of Dortmund, Germany. Their research is aimed at generating optimal smooth surfaces from digitised point data using evolutionary algorithms. The work presents three solutions to the problem of reconstructing smooth surfaces using triangular tiles.

Examples of other methods that improve quality of surfaces (smoothness and fairness) are those proposed by *Higashi et al.* [59] for the Toyota technological institute, and *Szilvasi-Nagy* [116]. A detailed list of procedures for the assessment and analysis of the surface quality and the detection of undesirable curvature are explained in *Hoscheck* [61] and *Pottmann* [86].

Fergusson et al. [48] and *Anderson et al.* [4] illustrate two mathematical means of smooth surface-control and constrained optimisation. These provide a complex mathematical approach to controlling the surface definition profile, which conforms to the designers' qualitative idea and true representation of the object under consideration. Thus, leading to an automatic mechanism for shape control, and creating convex surfaces with prescribed smoothness.

3.9 Discussions

Major Problems

There are different types of model representation on different levels. Processing and translating the representations causes problems mainly because points do not carry any information beyond pure 3D position coordinates. The surface reconstruction process as it stands today strips geometry of its intrinsic properties like surface normal or curvature. At the same time, there is currently no possibility of storing surface interrogation results like shading or reflection lines. And of course points do not capture design history, functionality aesthetic aspects or the stylists' intent, which would dramatically ease the surface reconstruction. Up to now, semantic information that exists about the objects is not used in reverse engineering.

Moreover, questions hang over the parameterisation of 3D models as surfaces (Bézier, NURBS). Representation could use completely different surface parameterisation or even work directly on point clouds. Perhaps new mathematical approach could avoid the numerical problems of real-number computer algebra. It is also possible to think of some representation schemes, which do not need parameterisation as they are used today (for example: voxel techniques).

Other technical problems are the numerical accuracy of the algorithms used, the handling of error propagation, the surface representations, insufficient user interface and systems incompatibility.

In this thesis, an attempt is made to answer some of the problems highlighted above through a better representation scheme for curves.

Requirements and Future Trends

Future developments in reverse engineering should include far more aspects of product designs than it does today. Instead of just focussing on geometry, reverse engineering could be used to support the entire industrial development and production process from conceptual design to manufacturing by integrating more high-level semantics. Or on larger scale, reverse engineering should be supported by a more flexible and integrated development process. CAD models should be extended to store associated information. History, functionality, constraints, design intent and design process data could be recorded along side the geometric representation. As previously seen, this model representation problem is addressed by the "feature based" and "constraint based" approaches.

Curve developers are not interested in knowing the underlying mathematical issues like smoothness, curvature or different degrees of continuity invariably associated with modelling of curves. These issues should be transparent, thus providing designers with tools powerful enough to manipulate models interactively. The lack of efficiency in the design process is partly due to inappropriate software, and more importantly user interface. Commercial CAD systems surveyed by the author, which include SDRC Ideas [109], IMAGEWARE SURFACER [108], IBM CATIA, *Mussa* [76], show that tools allowing constructing curves from desired properties (target driven design) are not (always) present CAD packages. Designers are left with "tweaking" tangents and control points in an attempt to match the desired curve characteristics. In the author's view, this trial and error approach to curve design is clumsy and shows that CAD vendors have a long way to go before meeting designers' requirements. In the light of this, this current research will focus on further investigate in the curve design area and propose software solution to the lack of flexibility with currently available commercial CAD packages.

Cooperation

One question that needs addressing is the (lack of) collaboration between universities, CAD vendors and industry. These three bodies have diverging objectives in nature, which makes any collaboration plans difficult. Research institutes are eager to develop new tools and techniques but need support from CAD vendors to implement these within an existing system. Developing a system from scratch would simply be a colossal task outside any research scope. Unfortunately, most commercial CAD systems are so called “closed” systems, this for confidentiality reasons, making them unsuitable for research purposes. Researchers in the area of geometric modelling and more generally CAD suffer from the unavailability of suitable research tools and paradoxically, CAD vendors suffer from a general lack of innovation. At the other end, industries suffer from a lack of productive tools in their design processes. Recently, CAD vendors are undertaking a more transparent policy by “opening” their systems to the outside world. This new trend in software architecture might put an end to this “catch 22” situation and benefit the world of CAD as a whole.

Part II

Chapter 4 Geometric Modelling

This part of the research aims to develop methodologies for the optimisation of free-form curves using geometric modelling, optimisation, and differential geometry. Some fields and applications related to geometric modelling that are found in *Bowyer* [18] are presented here with a short definition. They introduce the tools and topics that are used through out the research.

Computer Aided Graphic Design

Computer Aided Graphic Design is a discipline dealing the approximation and representation of 'free form' curves and surfaces with computer. The major breakthrough in CAGD came with the polynomial formulation of curves and surfaces in the Bernstein form independently developed by pioneers *P. Bézier* at Renault, *P. de Casteljau* at Citroen.

Computational Geometry

"Computational geometry is a phrase mostly used to refer to the study of geometrical algorithms, and particularly their theoretical efficiency, or order", *Bowyer* [18].

Geometric Modelling Kernels

The code implementation of Computational Geometry is enclosed in a kernel. Geometrical algorithms are coded into functions capable of dealing with the geometry. The data structure of the kernel plays an important part in the way these functions are accessible by peripheral applications.

An example of this is ACIS 3D Geometric Modeller from Spatial Corporation [1]. ACIS is an object-oriented three-dimensional (3D) geometric modelling engine designed for use as the geometry foundation within virtually any end user 3D modelling application. Written in C++, ACIS provides an open architecture framework for wireframe, surface, and solid modelling from a common, unified data structure. Parasolid from EDS [82] is another geometric modelling kernel that offers a similar functionality to ACIS. Paragraph 9.1

expands further on the topics of kernel architecture and interface. In this research, all the geometric algorithms use ACIS for computation.

Computer Aided Design

Computer Aided Design (CAD) is a rather general engineering term used to describe computer systems for designing parts. Traditionally CAD systems comprise of a geometric modeller, a graphic display interface, and a user interface. The so-called ‘turnkey’ systems combine all these three elements, and some more, into one large system called CAD. In this research, computer-aided design is used in a more restricted scope, this in order to separate CAD from its components (application programs, user interfaces, geometric reasoning). CAD simply is a user interface for editing geometry. ACIS has been used in conjunction with the Microsoft Foundation Classes (MFC) to develop a user interface, namely HulaHoops, for the optimisation applications proposed in this thesis.

Optimisation

As previously seen, optimisation aims to obtain better design given a set of design variables and constraints. Although optimisation is generally not regarded as part of geometric modelling, there is however material in this thesis on interacting optimisation with geometry and the advantages it offers.

4.1 Implicit and Parametric Geometry

There are many ways in which geometry can be represented. The simplest, or perhaps the best-known equation for representing a straight line is its explicit form:

$$y = ax + b$$

Unfortunately, this is the least useful representation. It is suited for single valued functions that will not double back on themselves. The explicit form cannot describe vertical straight line, the gradient a would then be infinite.

A better formulation for geometric modelling is the implicit form of equations. Implicit equations classify points in space in two categories, so the curve or surface defined is at the boundary between the two sets. Given a point on the surface or curve, implicit equations return zero and more or less than zero when the given point lies on either side of the boundary. Because they divide space into two, they are called Half Spaces.

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

It is said that Half Spaces are of dimensionality one lower than the space in which they are embedded, *Bowyer* [18]. In the plane, implicit equations describe curves; in the three dimensional space, they describe surfaces and in the four dimensional space, they describe volumes.

For example, if the six sides of a 3D rectangle block, shown in Figure 16, aligned with the coordinate axes are:

$$\begin{array}{lll} x = x_0 & y = y_0 & z = z_0 \\ x = x_1 & y = y_1 & z = z_1 \\ x_0 \geq x_1 & y_0 \geq y_1 & z_0 \geq z_1 \end{array}$$

Six sets of points can be generated from the following inequalities:

$$\begin{array}{lll} x \geq x_0 & y \geq y_0 & z \geq z_0 \\ x \leq x_1 & y \leq y_1 & z \leq z_1 \end{array}$$

And combining them with the intersection Boolean operator $\cap$:

$$(x \geq x_0)\cap(x \leq x_1)\cap(y \geq y_0)\cap(y \leq y_1)\cap(z \geq z_0)\cap(z \leq z_1)$$

Any point that satisfies that inequality lies inside the block, or more precisely on the edges or corners of the rectangle. Combining implicit equations with Boolean operators is the foundation of set-theoretic (or constructive solid geometry – CSG), which is further developed later in this chapter.

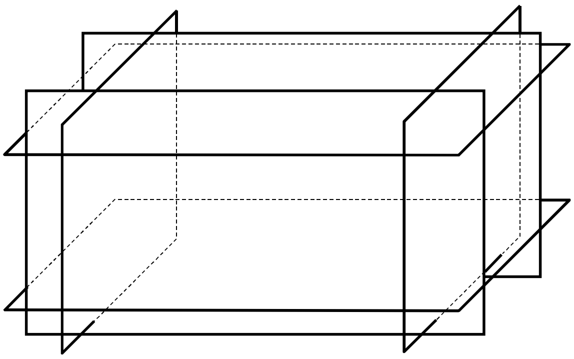


Figure 16: Six Half Spaces defining a rectangular block

In Parametric Equations, space coordinates *(x,y,z)* are functions of one or more variables or parameters. Parametric curves are defined in terms of one single independent variable often known as *t.* Similarly parametric surfaces are defined in terms of two variables often known as *u* and *v*.

Parametric equations are well suited for free-form curves and surfaces such as B-splines and NURBS. These will be further explored in subsequent chapters.

To summarise, *Bowyer* [18]:

Implicit Equations	*Classify points in the space*	Fixed dimensionality
Parametric equations	Generate points on the element	*Any dimensionality*

4.2 Geometric Solid Models

In the remaining of this chapter, geometric solid modelling schemes are presented and discussed. Even though solid modelling is not within the scope of this thesis, it cannot be omitted because solid modelling encapsulates the topology of models, which is common to curves and surfaces. In actual fact curve and surface modelling is a special case of geometry of solids, therefore an overview of solid modelling is needed.

4.2.1 Wire-Frame

This type of representation is probably the earliest used in CAD. 3D wire-frame models are defined by a set of 3D points (vertices) in the (x,y,z) orthogonal coordinate system linked together by lines or curves (edges). In wire frame models no surfaces or faces are defined, keeping the model representation to simplest form. Wire frame does not require difficult computation and 2D orthographic views are generated by projection of 3D data.

Only two types of information is conveyed with wire frame:

- Metric
- Geometric

Other deficiencies of wire frame representation are outlined below, *Jared* [63]:

- Ambiguity, one wire frame might have several interpretations
- Nonsense objects cannot be easily detected
- No automatic generation of view dependant information such as silhouette lines of curves surfaces
- More generally, lack of surface information

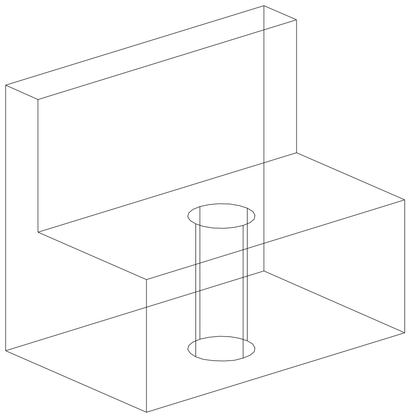


Figure 17: Wire frame model

4.2.2 Set-theoretic Modelling

Set-theoretic treats geometry as a three-dimensional Venn diagram. It takes simple shapes and puts them together to make more complicated ones by using the operators of set-theory. There are four operators, *Bowyer* [17]:

- Union gives solid where either or both of the two objects being unioned are solid, just as the OR operator gives you a 1 when either or both of the two bits being ORed are 1

$$A+B = A \cup B$$

- Intersection gives the part of space where the two objects being intersected are both solid, just as the AND of two bits of data only gives you a 1 when both bits are 1.

$$A \,\&\, B = A \cap B$$

- Difference subtracts one object from the other where the two objects intersect. In simple terms, Difference implies the set-complement of the subtracting object.

$$A-B = A \cap \overline{B}$$

- Symmetric difference is equivalent to the exclusive OR operator (XOR)

$$A \Delta B = (A \cap \overline{B}) \cup (B \cap \overline{A})$$

The usual two-dimensional Venn diagram for all these is shown in Figure 18. There is striking similarity between the operators of set-theory and those of Boolean logic. It is because of all this that set-theoretic geometric modellers are sometimes called Boolean modellers or Constructive Solid Geometry (CSG) in the literature.

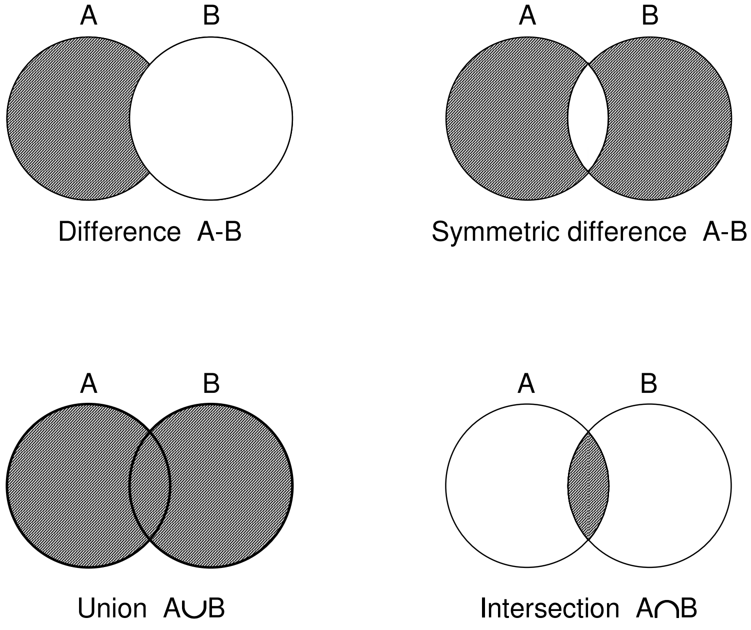


Figure 18: Set-theory operators

In Set-theoretic modelling, solids are defined in terms of Boolean operations on simple solid primitives. Primitives are either a combination of half spaces or bounded primitives such as blocks, spheres, cones. Thus, a model can be conveniently described by a tree data structure with its terminal nodes representing solid primitives and non-terminal nodes denoting Boolean operations. An example of a CSG model is presented Figure 19 where the object, lets say a toilet seat, is constructed by unioning two blocks and subtracting a cylinder

$$Toilet\ Seat\ =\ (A \cup B) - C$$

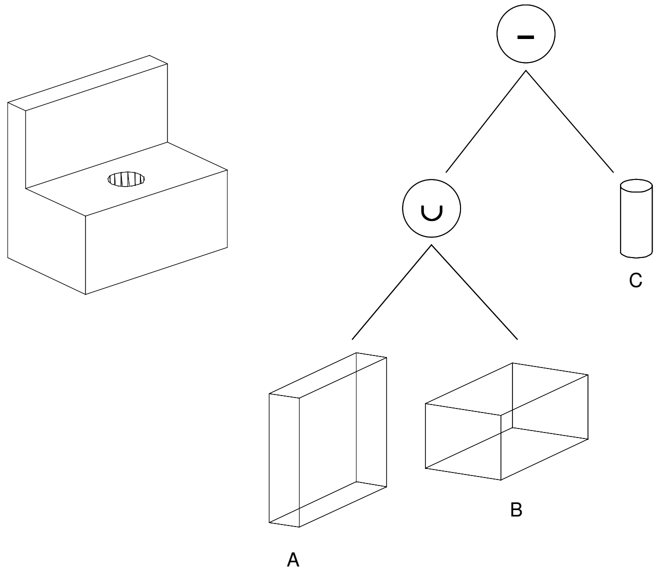


Figure 19: Set-theoretic model

Point-Set Membership Classification method allows classifying points inside or outside a solid object.

If an arbitrary point $\mathrm{P_i}(x_i, y_i, z_i)$ is tested against each element constituting the toilet seat:

$$\begin{aligned}
&\mathrm{P_i} \text{ vs Block A} \xrightarrow{lies} \text{Inside}\\
&\mathrm{P_i} \text{ vs Block B} \xrightarrow{lies} \text{Inside}\\
&\mathrm{P_i} \text{ vs Cylinder C} \xrightarrow{lies} \text{Outside}
\end{aligned}$$

The Boolean equations becomes: $(\text{Inside} \cup \text{Inside}) - \text{Outside} \xrightarrow{lies} \text{Inside}$

4.2.3 Boundary Representation (B-Rep)

The principal property of B-rep is to represent the geometry (detailed shape) and the topology (connectivity) of objects separately. This concept provides the ability to determine whether a position is inside, outside, or on the boundary of a volume, which distinguishes solid models from surface or wire frame models. Among Bounded Geometry and Topology that are included to this chapter, other topics related to boundary representation are presented in Appendix 3. These are Topological hierarchy, Euler-Poincaré formula, and some introductory material on Graph Theory.

Bounded Geometry

Solid models are composed of a collection of surfaces joined together as shown in Figure 20. Intersecting surfaces give curves, and intersecting curves give points. These three geometric entities, point, curve and surface, are the basic elements needed to define a solid object.

With the exception of individual points, circles and spheres, analytical geometry only represents geometry that extends to infinity (unbounded). In order to obtain finite segments of curves and portions of surfaces, B-rep has to explicitly bound the geometry:

- A curve is bounded by a pair of points
- A surface is bounded by a collection of curves lying on the surface
- A solid is bounded by a collection of surfaces

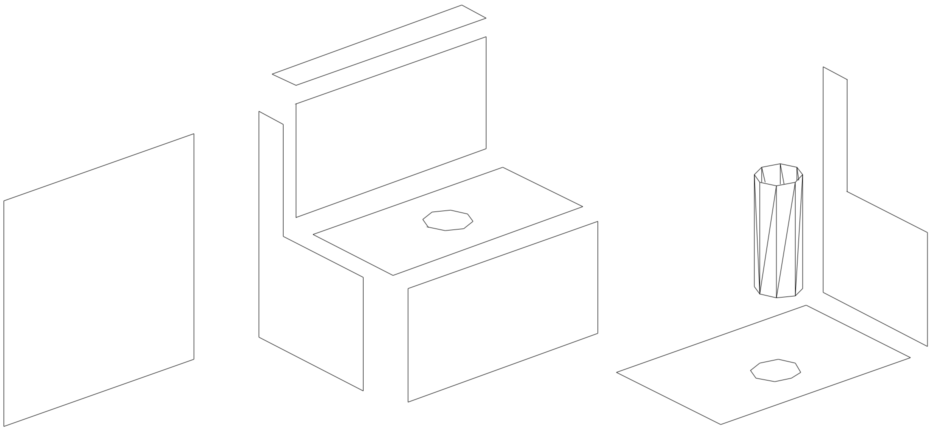


Figure 20: Solid object composed of surfaces

In order to distinguish between unbounded and bounded geometry, it is convenient to give the bounded elements different names.

- A connected portion of a surface is a face
- A connected segment of a curve is an edge
- A point at the boundary of an edge is a vertex

Faces, Edges, Vertices are composite entities known as topological entities because they define how things interconnect.

- The shape of a Face is defined by a Surface whose boundary is represented by a collection of Edges associated with it.
- The shape of an edge is defined by a curve whose boundary is represented by a pair of Vertices associated with it.
- The location of a vertex is defined by a Point.

Topology

The remaining task to complete the B-Rep model is to provide means of recording the arrangements of the points, curves and surfaces. This is known as the topology.

Topology refers to the spatial relationships between the various entities in a model. Topology describes how geometric entities are connected. On its own, topology defines a "rubber" model, whose position is not fixed in space. For example, a circular edge and an elliptical edge are topologically equivalent (but not geometrically). Likewise, a square face and a rhomboid face are topologically equivalent (but not geometrically). A topological entity's position is fixed in space when it is associated with a geometric entity. Topology can be bounded, unbounded, or semi-bounded, allowing for complete and incomplete bodies. A solid, for example, can have missing faces, and existing faces can have missing edges. Solids can have internal faces that divide the solid into cells. Bodies such as these are called non-manifold because they are not physically realizable. The topological information of a B-Rep model is stored in a graph data structure. Also to be stored in the B-Rep data structure is the adjacency relationships describing how the elements are connected. Such a network of relationships can be used to share bounding entities and prevent data duplicates. For example in a cube, each corner point bounds three edges and each edge bounds two faces.

The boundary representation (B-rep) of a model is a hierarchical decomposition of the model's topology. The model hierarchy with ACIS is presented in Appendix 3.1.

4.3 Function Representation

The representation of a geometric object by a single real continuous function of several variables as F(X) >= 0 is called Function Representation or F-rep. It provides symbolic representations of equations that are parsed in much the same way that equations are. They provide the ability to solve complex mathematical problems.

In ACIS geometric entities provide methods for querying their properties. However, the data structures do not explicitly store information such as a point on a curve where the radius of curvature is a minimum, the tangent vector from a given point on an edge, the normal vector from a given point on a surface, or the numerical values of the *n*th derivatives of points along a curve to determine continuity.

There are also the SvLis [17] and GAS (Geometric Algebra System) [15] systems from Bath University, that can provide Function Representation modelling capabilities. However the Svlis modeller is CSG and therefore not much use for dealing with free-form curves, which is the example used in this thesis. Actually, *Berchtold and Bowyer* [10] have published on the problem of supporting NURBS in CSG modellers, but have faced problems with implicitisation of polynomials.

4.4 Summary

In this chapter, different types of geometric solid schemes have been introduced. Table 2 summarises these in terms of their representation paradigms.

Type	Representation
Wire frame	Vertices and edges only, no surfaces or faces
Set-theoretic (CSG)	Surfaces (i.e. half spaces) only, no vertices or edges, implicit boundary
Boundary Representation (B-rep)	Vertices, edges, faces, explicit boundary
Polyhedral model	Only planar geometry, sometimes only triangular faces
Cellular decomposition (Voxel, Octree)	Thousands of cubes arranged in a hierarchy
F-rep	Continuous function representation

Table 2: Types of geometric modellers

Chapter 5 Spline Curve Modelling

Shape definition is a challenging task in engineering design. Traditionally aesthetic shapes are generated manually and subsequently translated into computer representation. Conventional methods for shape design and analysis are not necessarily the best methods for computer implementation, but have provided insight for new algorithms. In this chapter some free-form modelling algorithms are introduced.

Surface modelling differs from solid modelling in that surfaces do not have thickness; hence surfaces have no notion of volume. Free-Form Curves and Surfaces falls into the discipline called Computer Aided Graphic Design or CAGD, which is concerned with the representation of curves and surfaces. Recently CAGD has focused most attention and is a subject area that has more literature than any other subjects in Geometric Computing.

5.1 Literature

A complete reviewing of curves and surfaces from literature would simply be -a- totally pointless, -b- too long. Therefore, this thesis will be restricted to cover material necessary for the understanding of the following chapters avoiding to falling into the trap of obscurantism. Mainstream books, such as *Farin* [44] and *Faux* & *Pratt* [46], give comprehensive basements for the best-known curves and surfaces methods. Further reading, and more specialised are these. *Choi*'s book [23] is from a CAM point of view, but towards the end he makes some recommendations for the design of a neutral representation (unified) of curves and surfaces, which holds potential for arguments. *Sapidis* [106] is a collection of rather insightful articles, which addresses quality issues with surface design. On NURB curves and Surfaces, *Piegl* [84] is certainly the most complete volume available, others include *Farin*, [45].

5.2 Polynomials

One way to extend parametric equations is to make the function of parameter(s) an arbitrary polynomial. The polynomial form is more flexible than its monomial equivalents and is widely used for representing curves and surfaces.

$$\text{explicit polynomial}: y = f(x) = a + bx + cx^2 + ...;\ 2D\ only$$
$$\text{implicit polynomial}: f(x, y) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij}x^i y^j = 0; \quad \text{m, n integers}$$

The parametric form is particularly attractive since it is easy to create points on the on the element and to bound them to a particular range of parameter values.

$$P(u) = A + B(u) + C(u^2) + D(u^3) + ...$$

In theory it is possible to convert equations from implicit to the parametric form. This conversion process is called parameterisation; the inverse is called implicitisation.

The Power Basis

A simple parametric polynomial can be written as a weighted combination of a set of powers of the variable, *u.*

$$P(u) = [1 \quad u \quad u^2 \quad u^3 ...]\begin{bmatrix} A \\ B \\ C \\ D \\ ... \end{bmatrix}$$
$$P(u) = UA \text{ with } 0 \le u \le 1$$

In the above equation, $U = [1 \ \ u \ \ u^2 \ \ u^3 ...]$ is called the *power basis* vector and $A = [A \quad B \quad C \quad D ...]$ is the coefficient vector.

5.3 Interpolation

Parametric polynomials have limitations: Extending the definition of a curve implies adding more high degree terms to the equation. High degree polynomials are sensitive to inaccuracies and are computational inefficient. Also the coefficients in a polynomial equation do not bear any physical significance, which makes handling of curves difficult. This problem can be overcome by defining curves such that the curve satisfies a set of constraints. For general parametric equations, common constraints are point coordinates and tangent direction, but others such as higher derivatives or curvature can also be used. Constructing geometry as a constraint satisfaction problem is called interpolation. The Hermite and Lagrange interpolation are discussed here.

Going back to the essence of this thesis, Flexible Optimisation in CAD/CAM Environment, the concept of curve and surface definition as a set of constraints is an appealing one. As previously discussed, one of the main advantages of evolutionary algorithms is their ability to represent constraints as a function of variables. This enables an algorithm to perform operations on geometry without needing to know the underlying representation paradigm.

5.3.1 Lagrange Interpolation

The most obvious interpolation technique is to make the curve pass through a series of points. This is called Lagrange interpolation. Each point the curve passes through is a coefficient in the equation. To interpolate a parametric polynomial of degree *n* to the *n+1* data points $P(u_i)$, $0 \leq i \leq n$ *where* $\underline{u_i}$ is a fixed value of the parameter *u* at P_i,, *Farin* [44], [52]

$$P(u) = \sum_{i=0}^{n} P(u_i) L_{0,i}(u)$$

Where $L_{0,i}(u)$ are the Lagrange polynomials

$$L_{0,i}(u) = \frac{\prod_{\substack{j=0 \\ j \neq i}}^{n} (u - u_j)}{\prod_{\substack{j=0 \\ j \neq i}}^{n} (u_i - u_j)}$$

and $\prod$ is product with respect to the index *j*.

In simpler terms, the coefficients are evaluated by substitution of the coordinates at each point and the parameter value at the point in the power basis equation. From the following polynomial (2D for simplicity):

$$\begin{aligned}
&P(u) = A + B(u) + C\left(u^2\right) \\
&x = a_1 + b_1(u) + c_1\left(u^2\right) \\
&y = a_2 + b_2(u) + c_2\left(u^2\right)
\end{aligned}$$

that passes through three points equally spaced at the parameters values between zero and 1

$$\begin{aligned}
&P_1(x_1, y_1) \text{ at } t = 0 \longrightarrow x_1 = a_1 \\
&\qquad\qquad\qquad\quad \longrightarrow y_1 = a_2 \\
&P_2(x_2, y_2) \text{ at } t = \tfrac{1}{2} \longrightarrow x_2 = a_1 + \tfrac{1}{2} b_1 + \tfrac{1}{4} c_1 \\
&\qquad\qquad\qquad\quad \longrightarrow y_2 = a_2 + \tfrac{1}{2} b_2 + \tfrac{1}{4} c_2 \\
&P_3(x_3, y_3) \text{ at } t = 1 \longrightarrow x_3 = a_1 + b_1 + c_1 \\
&\qquad\qquad\qquad\quad \longrightarrow y_3 = a_2 + b_2 + c_2
\end{aligned}$$

By solving for the three coefficients, we obtain:

$$\begin{aligned}
a_1 &= x_1 & a_2 &= y_1 \\
b_1 &= -3x_1 + 4x_2 - x_3 & b_2 &= -3y_1 + 4y_2 - y_3 \\
c_1 &= 2x_1 - 4x_2 + 2x_3 & c_2 &= 2y_1 - 4y_2 + 2y_3
\end{aligned}$$

Lagrange interpolation produces acceptable results. However for larger values of *n*, the interpolated curve tends to wiggle due to the behaviour of higher degree polynomials. The term "wiggle" means that the curve oscillates between points, which gives slope oscillations. For this reason Lagrange interpolation is often not satisfactory for smooth and fair curves, especially for aesthetic purposes. Also as demonstrated in *Farin*'s book [44], Lagrange interpolation is said to be ill conditioned. This means that for small changes in the input data might result in serious changes of the result.

5.3.2 Hermite Interpolation

Lagrange interpolation only takes data points as input, another interpolation scheme known as Hermite Interpolation defines a curve in terms of its derivatives of order *n* at the point P_i. Therefore the data are derivative vector $P^r(u_i)$ of order $0 \le r \le n$ and the curve equation takes the following form *Forrest* [52], *Farin* [44], *Faux* [46]:

$$P(u) = \sum_{r=0}^{n} P^r(u_i) L_{r,i}(u)$$

where $L_{r,i}(u)$ are the blending functions:

$$L_{r,i}(u) = \frac{(u - u_i)^r}{r!}$$

In Hermite interpolation, both *nth* derivative vectors and data points are taken as input, thus *(n+1)* as many coefficients as in Lagrange interpolation. For example, cubic interpolation takes first order derivatives (tangents continuous) and quintic interpolation takes second order derivatives (curvature continuous).

A smooth curve segment is obtained by joining two end-points P_o and P_1 together with specified end-tangents t_0 and t_1, Figure 21. In order for a cubic *P(u) with* $0 \le u \le 1$, to meet these conditions, the following relations must stand:

$$P_0 = P(0); \quad t_0 = \dot{P}(0);$$
$$P_1 = P(1); \quad t_1 = \dot{P}(1)$$

Where $\dot{P}(u)$ is the first derivative of *P(u)* with respect to *u*:

$$P(u) = a + b(u) + c(u^2) + d(u^3)$$

$$\dot{P}(u) = \frac{dP(u)}{du} = b + 2uc + 3u^2 d$$

By feeding the above conditions in the cubic, the followings are obtained:

$$P_0 = P(0) = \mathrm{a}; \quad t_0 = \dot{P}(0) = \mathrm{b};$$
$$P_1 = P(1)\mathrm{a} + \mathrm{b} + \mathrm{c} + \mathrm{d}; \quad t_1 = \dot{P}(1) = b + 2c + 3d$$

Which are solved to evaluate the unknown coefficients

$$\begin{aligned} a &= P_0 \\ b &= t_0 \\ c &= -3P_0 + 3P_1 - 2t_0 - t_1 \\ d &= 2P_0 - 2P_1 + t_0 + t_1 \end{aligned}$$

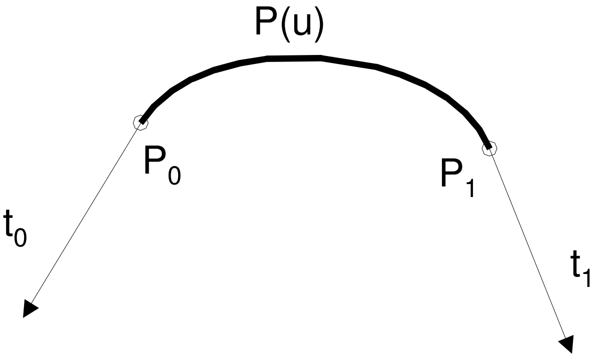


Figure 21: Construction of a curve segment with tangents

Hermite interpolation has one advantage over other methods (Bézier, B-spline): it stores interpolation points explicitly. In the B-spline from, they must be computed. In real life, curve tangents are difficult to measure form a physical model. For this reason Hermite interpolation is not frequently used, otherwise the tangents must be evaluated in order to satisfy continuity constraints between adjacent curve segments.

5.3.3 Approximation

In interpolation, a curve is constructed so that it satisfies a set of constraints precisely (position, tangent…). In some instances, especially with data from digitised models, there are too many data points to interpolate with a polynomial curve. An alternative to interpolation is *approximation*, where the curve does not pass through the points but near them (‘*near’* is bounded by a tolerance). Figure 22 shows an example of curve approximation with *n+1* points. The curve is constrained to pass through the first and last point P_0 and P_n. The problem becomes finding a curve $P(u)$ such that the distances $e_i = \|P_i - P(u_i)\|$ are as small as possible. A least squares approximation is typically used for this purpose. See *Farin*’s book [44] for further details.

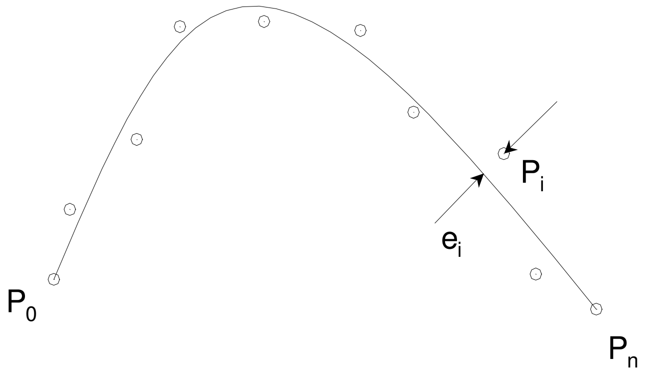


Figure 22: Curve approximation, *n+1* points. The curve is constrained to pass through P_0 and P_n

5.4 The Bernstein-Basis

The previous paragraph has shown that curves can be constructed by interpolating data points with Lagrange polynomials as well as derivative vectors with Hermite interpolation. Data fitting with interpolation is convenient for simple curve construction but control over shape is difficult because derivative vectors have to be completely specified.

As already mentioned, in the power-basis form, the coefficients [A, B, …] do not bear any apparent geometrical significance. The power-basis formulation is purely algebraic, which is not suited for design purposes. Also it is numerically unstable, inaccuracy in or computer-rounding errors lead to great shape variations in the resulting curve. Other techniques such as Bézier and B-spline provide better control over shape. Both methods are derived from the de Casteljau construction algorithm. Also the Bernstein polynomials, which form the basis functions of Bézier and B-splines is presented in this paragraph.

5.4.1 Bézier Curves

The curves that are known as Bézier curves were independently developed by *P. de Casteljau* in 1959 at Citroën and *P. Bézier* at Renault in 1962. Strangely enough the two engineers were not aware of each other's work despite the proximity of the two companies. Even though *P. de Casteljau* had first discovered the algorithm, only *P. Bézier* published his work, which is in a sense the generalisation of the algorithm. Hence why the curves bear his name today. The mathematical theory behind Bézier is the concept of Bernstein polynomials. But it was not before 1970 that *R. Forrest* from Cambridge University established the connection between the Bézier curves and the Bernstein polynomials.

5.4.2 The de Casteljau Algorithm

The de Casteljau Algorithm is the fundamental concept in free-form curve design with polygons. Bézier and B-Splines curves are direct application of this construction method.

A simple curve construction with the de Casteljau algorithm is given here. Let three points, P_0, P_1, P_2 in 3D space and *u* a parameter real. From the generic expression of the straight line, we can write the three equations of P_0^1 *and* P_1^1:

$$P_0^1(u) = (1-u)P_0 + uP_1$$
$$P_1^1(u) = (1-u)P_1 + uP_2$$

Then we create a straight-line segment between the moving points on the first two:

$$\mathrm{P}_0^2(u) = (1-u)P_0^1 + uP_1^1$$

and combine the first two equations into the third one, we get:

$$\mathrm{P}_0^2(u) = (1-u)^2 P_0 + 2u(1-u)P_1 + u^2 P_2$$

or in the power basis form it becomes:

$$\mathrm{P}_0^2(u) = P_1 + (P_0 + P_1)u + (P_0 - 2P_1 + P_2)u^2$$

This is a quadratic expression function of *u*, which in fact is a parabola. The de Casteljau construction consists of a repetition of linear interpolation for values of parameter *u* from zero to 1, as shown in Figure 23 below.

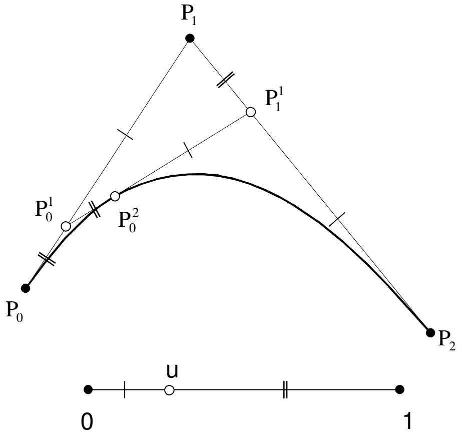


Figure 23: de Casteljau construction out of two fixed straight line segments and one varying segment.

The terms *u* and *(1-u)* correspond geometrically to equal ratios between each single- and double-ticked part of each straight-line segment:

$$ratio\left(P_0, P_0^1, P_1\right) = ratio\left(P_1, P_1^1, P_2\right) = ratio\left(P_0^1, P_0^2, P_1^1\right) = t/(1-t)$$

5.4.3 Bernstein form of Bézier Curves

Bézier curves can be defined by a recursive algorithm such as the de Casteljau's. However a more mathematical definition is often needed. The Bernstein form a Bézier curve, first established by *Forrest* [51]. It is not in the scope of this thesis to get in too many mathematical details, it is discussed elsewhere in *Bernstein* [11], and its application to Bézier curves in *Faux* [46] and *Farin* [44].

A *nth* degree Bézier curve is defined by:

$$P(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i$$

Where P_i are the position vectors of the *(n+1)* vertices and $B_{i,n}$ are the Bernstein polynomials that are obtained from the Weierstrass theorem and are defined by:

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

The cubic Bézier curve implemented in Renault's UNISURF CAD system, is a special case of this general curve, where *n=3* and *i=4*, Figure 24. In this case, the blending functions are:

$$B_{0,3} = \frac{6}{6} u^0 (1-u)^3 = 1 - 3u + 3u^2 - u^3$$

$$B_{1,3} = \frac{6}{2} u^1 (1-u)^2 = 3u - 6u^2 + 3u^3$$

$$B_{2,3} = \frac{6}{2} u^2 (1-u)^1 = 3u^2 - 3u^3$$

$$B_{3,3} = \frac{6}{6} u^3 (1-u)^0 = u^3$$

Where it is assumed that $\lim_{i \to 0} u^i = 1$.

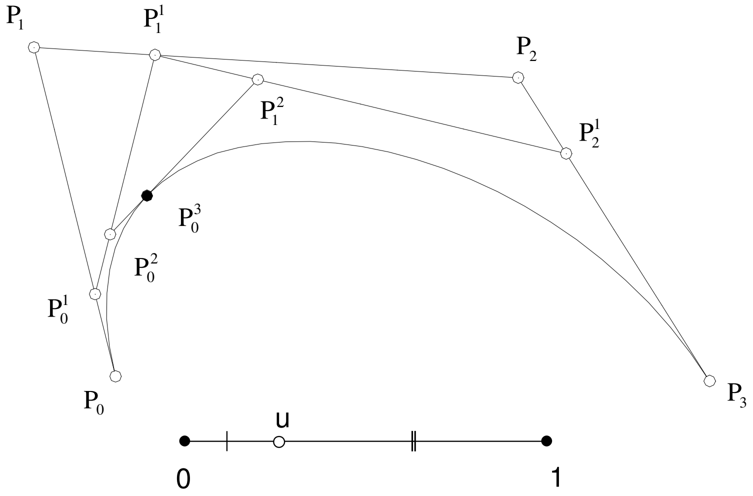


Figure 24: Cubic Bézier curve with four-vertex polygon

5.4.4 The Matrix Form

The Bernstein polynomials can be conveniently written in a matrix form:

$$P(u) = BP$$

where *P* is the points matrix, which for a cubic is

$$P = \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

and *B* the Bernstein polynomials

$$B = \lfloor B_{0,3}, \quad B_{1,3}, \quad B_{2,3}, \quad B_{3,3} \rfloor$$

The *B* matrix can be expressed as the product of two other matrices

$$B = UBm$$

where

$$U = \lfloor u^3 \quad u^2 \quad u \quad 1 \rfloor$$

and *Bm* is the coefficients of the Bernstein blending functions

$$Bm = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

5.4.5 The Recursive Form

An important property of the Bernstein polynomials is the recursive form, which will prove useful later for the construction of B-splines.

$$B_{i,n}(u) = (1-u)B_{(i,d-1)}(u) + uB_{(k-1,d-1)}(u)$$

5.4.6 Some Properties of the Bézier Curves

Any Bézier curve lies within its defining polygon and passes through the first point and last point. Modification of the polygon vertices will change the global shape of the curve. It can also be refined by elevating the order of the curve (adding more points in the polygon), but this will induce more computation, and is not an attractive option. Rather, composite Bézier curves are introduced. Curves may be composed of several Bézier curves in order to generate more complex shapes that are too complex for a single curve to handle. To achieve piecewise construction, continuity conditions are required. This leads to the generalisation of Bézier, called B-splines.

5.5 B-spline Curves

The motivation behind B-splines is that interpolated curves or curves consisting of one single segment are often inadequate, their shortcomings are:

> A high degree is required to satisfy a large number of constraints: an *n+1* degree polynomial is needed to pass through *n* data points.
>
> Interpolated curves are not well suited to interactively modify shape; Although Bézier polygons allow this, control is not sufficiently global.

B-splines address these problems by combining both the properties of piecewise data-point interpolation and Bézier-like control polygon in one fundamental definition.

> The degree of the curve is independent of number of control points
>
> Local shape control is possible because individual control points have only local influence.

The term *Spline* comes from early engineering applications such as shipbuilding and airplane. Draftsmen define free-form shapes using flexible stripes of metal or wood. These splines were distorted by means of weights applied at specific distances along them. The mechanical properties of the material used offered deformations with second order continuity, which is a property enclosed in modern mathematical splines. Although physical splines are rarely used, the underlying principle forms the basis of new algorithms.

In literature, B-splines also called fundamental splines are in some ways the natural generalisation of Bézier curves, *Back* [8]. *Woodwark's* statement in *Bowyer* [18] about B-splines summarises fairly well the situation:

"B-splines curves are just pieces of Bézier curve ingeniously knotted together; whatever the hype, don't forget this."

5.5.1 B-spline Blending Function

A B-spline is defined by its polygon vertices also called control points together with an array of knots where the pieces of curves join, A *nth* degree B-spline is defined by the following expression, *Farin* [44], *Piegl* [84], *Bowyer* [18]:

$$P(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i$$

Where P_i are the control points and $B_{i,n}$ are the recursive basis functions:

$$B_{i,n}(u) = \frac{u - u_i}{u_{i+n-1} - u_i} B_{i,n-1}(u) + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_{i+1}} B_{i+1,n-1}(u)$$

5.5.2 Rational Curves

The rational form of Bernstein Polynomials, often called *NURBS*, as Non-Uniform Rational B-spline, are the latest fashion in free-form curves and surfaces. Standard literature on NURBS is *Piegl* [84], *Farin* [45]. The main advantage over the non-rationals is their invariant property under projective transformation. This means that the projection of a rational curve is a rational curve. Also rational polynomials allow representation of shapes like where traditional non-rationals fail. This is the case of all conics, including circle.

In short, NURBS are defined as the ratio of two polynomials (a spline divided by a spline). They are represented with rational functions of the form

$$x(u) = \frac{X(u)}{W(u)} \qquad y(u) = \frac{Y(u)}{W(u)}$$

Where $X(u)$, $Y(u)$ and $W(u)$ are polynomials, that is, each of the coordinate functions have the same denominator.

Combining the above with the definition of B-splines, a *pth* degree rational B-spline is therefore

$$P(u) = \frac{\sum_{i=0}^{n} B_{i,p}(u) P_i w_i}{\sum_{i=0}^{n} B_{i,p}(u) w_i}$$

As before P_i are the polygon vertices, and $B_{i,p}$ are the B-spline basis functions. w_i are scalars called weights.

Like with Bézier and B-splines, shape control is achieved by moving the polygon vertices P_i, and in the case of rationals by changing w_i values. Figure 25 shows the effect on changing weight values of one polygon vertex on a cubic Bézier. Weights must be positive and nonzero.

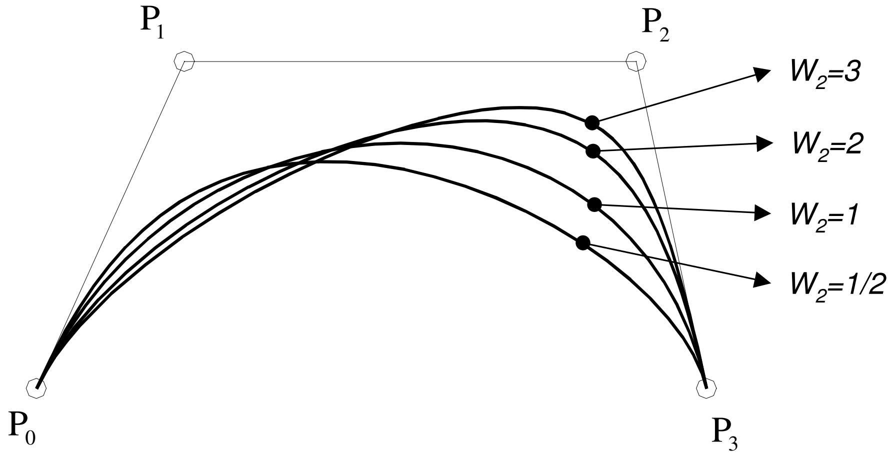


Figure 25: Rational cubic Bézier curve with W_2 varying

5.6 Summary

This chapter has introduced some techniques for modelling of free-form shapes. They can be classified into two categories: Those that are constructed by interpolating data points (Lagrange) and some degree of vectors associated with them (Hermite, Spline); and those that are constructed with the de Casteljau algorithm and modelled with Bernstein polynomials (Bézier, B-spline, NURBS).

All these curve and surface modelling schemes belong to the same family of curves. It is common to see geometric modellers representing internally all of these as NURBS, which is the most generic scheme, the others are seen as a special case of NURBS - only the modeller's interface (calling functions) differs. This approach is attractive with regards to representation homogeneity *Choi* [23] between modellers present weaknesses.

NURBS is weak for producing quality curves: polynomials of high degree tend to wiggle in between control points. NURBS representation shows immediate limitations because constraints are only applicable discretely at the control points. What goes on in between is function of these, but not always in a predictable manner. The most encouraging formulation is the Hermite interpolation which explicitly holds high order differential constraints which can be used as handles on the curve's shape. These considerations are treated in forthcoming chapters.

Chapter 6 Differential Geometry of Curves

Before exploring the depths of curve optimisation techniques, this chapter introduces some aspects of differential geometry necessary for their understanding. The notation details of the vector algebra used in this chapter is included in Appendix 4.

Early work on differential geometry applied to curves and surfaces dates from the XIXth century with French mathematicians like *Jean Frederic Frénet* 1816-1900, *Joseph Alfred Serret* 1819-1885, *Joseph Louis Bertrand* 1822-1900, Victor Mannheim 1831-1906, *Jean Gaston Darboux* 1842-1917. More modern literature includes *Aminov* [3], *Gibson* [53], *Nutbourne* [80], *Eisenhart* [41].

In describing the geometric character of a curve, six quantities are considered: the position, tangent, principal normal, bi-normal, curvature and torsion. The definition of curvature and torsion are central to the discussion of a curve's shape. The theorems of Existence and Uniqueness state that the torsion and curvature functions together totally define a curve up to a motion of Euclidean space. $k(u)$ and $\tau(u)$ are called the natural or intrinsic equations of a curve.

6.1 Frénet Frame

At this stage, it is necessary clarify the term "parameterisation". It can either mean "choice of parameter" (as here) or "conversion from implicit to parametric form" (the opposite of implicitisation). For the purpose of this discussion it is assumed that all example curves are described using a differentiable vector valued function of the form of Equation 1.

$$P(u) = \{x(u), y(u), z(u)\} \qquad \mathrm{u} \in [\mathrm{a}, \mathrm{b}]$$

Equation 1

where u is an arbitrary curve parameter, with the restriction that $\| P'(u) \| \neq 0$ for all u; this is called the regular parameterisation. An alternative parameterisation called arc length parameterisation given in Equation 2 is also considered. Under this parameterisation, $\| P'(u) \| = 1$ everywhere.

$$s(u) = \int_a^u \| P'(u) \| \, du$$

Equation 2

For the purpose of the discussions in this thesis, *P(u)* refers to an arbitrary regular parameterisation and *P(s)* refers to an arc length parameterisation.

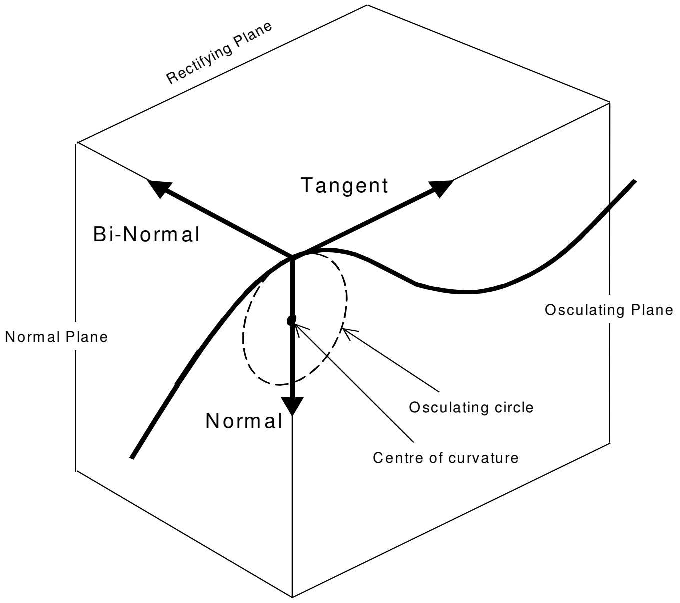


Figure 26: Frénet frame

The Frénet frame describes a sliding orthogonal coordinate system defined at each point on the curve. The three unit vectors of the Frénet frame are the tangent (*T*), normal (*N*) and bi-normal (*B*). These form three planes, which contain the three vectors above. [*T,N*] form the osculating plane, [*N,B*] form the normal plane and [*B,T*] form the rectifying plane (see Figure 26).

Because the three vectors making the frame are orthogonal and unit length, the following relations stand.

$$B = T \times N; \quad T = N \times B; \quad N = B \times T$$

Equation 3

6.1.1 Tangent

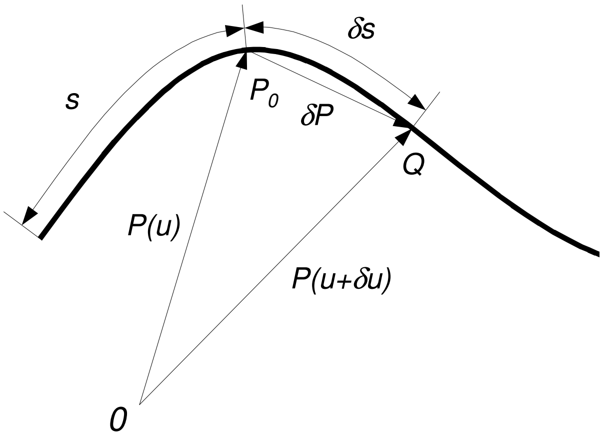


Figure 27: Tangent of a curve

The curve Figure 27 is a general 3D curve defined by its position vector $P = P(u)$. The vector $\delta P = P(u + \delta u) - P(u)$ represents the chord P_0Q joining the two points P_0 and Q with parameters *u* and δu. As $\delta u \longrightarrow 0$, the vector $\delta P / \delta u$ has a direction that approaches the direction of the tangent at P_0. If the arc length *s* is the parameter, then the chord length $|\delta P|$ and the arc length δs are equal in the limit, *Faux* [46], *Aminov* [3]. In Equation 4, *T* is a unit vector of direction the tangent of the curve at point P_0.

$$T(s) = \frac{dP(s)}{ds} = \lim_{\delta s \to 0} \frac{\delta P}{\delta s}$$

Equation 4

With respect to parameter *u*, for any point *P* of the smooth curve $P(u)$ there exist the tangent to $P(u)$, and the directing vector of the tangent is *T, Aminov* [3]. Assuming that the curve $P(u)$ has a derivative function *P'(u)*, the unit tangent can be written as follows:

$$T(u) = \frac{P'(u)}{|P(u)|} = \|P'(u)\|$$

Equation 5: Tangent function of parameter *u*

6.1.2 Principal Normal and Curvature

The normal is a unit vector that is perpendicular to the tangent. Evidently there is an infinity of normals to a curve at a point. Two of these are of particular interest: the normal which lies on the osculating plane, called the principal normal; and the normal which is perpendicular to this plane called the bi-normal.

If *P(s)* is an arc length parameterised curve, then *P'(s)* is a unit vector, and hence *P'(s) P'(s)=1.* Differentiating this relation, gives

$$P'(s)\,.P''(s) = 0$$

Equation 6

Which states that *P''* is orthogonal to the tangent vector, providing that it is not a null vector. This fact can also be interpreted from the definition of the second derivative *P''(s)*

$$P''(s) = \lim_{\Delta s -> 0} \frac{P'(s+\Delta s) - P'(s)}{\Delta s}$$

Equation 7

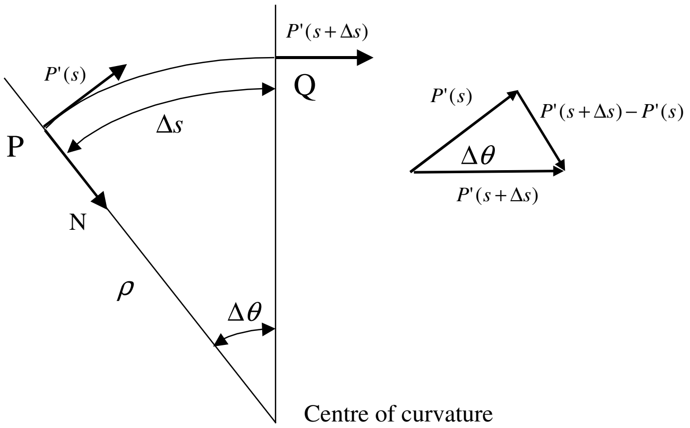


Figure 28: Principal normal and curvature

As shown in Figure 28, the direction of $P'(s+\Delta s) - P'(s)$ becomes perpendicular to the tangent vector as $\Delta s \to 0$. The unit vector which has the direction and sense of *P'(s)* is the principal normal vector to the curve at *s*.

$$N(s) = \frac{P''(s)}{|P''(s)|} = \frac{T'(s)}{|T'(s)|} = \|T'(s)\|$$

Equation 8: Normal function of arc *s*

Where $P'(s+\Delta s)$ moved from *Q* to *P,* then $P'(s), P'(s+\Delta s)$, $P'(s+\Delta s) - P'(s)$ form an isosceles triangle (see Figure 28), since $P'(s)$ and $P'(s+\Delta s)$ are unit tangent vectors. Thus $P'(s+\Delta s) - P'(s) = \Delta\theta.1 = |P''(s)\Delta s|$ as $\Delta s \to 0$ hence κ is the curvature, and its reciprocal ρ is the radius of curvature at *s*. It follows that

$$|P''(s)| = \lim_{\Delta s \to 0} \frac{\Delta\theta}{\Delta s} = \frac{1}{\rho} = \kappa$$

$$\kappa(s)N(s) = P''(s)$$

Equation 9

Equation 10: Curvature function of arc *s*

With respect to the parameter *u*, the curvature of a curve involves a vector product of the curves derivatives

$$\kappa(u)N(u) = \frac{(P'(u) \times P''(u)) \times P'(u)}{\|P'(u)\|^4} \quad \text{in } \Re^3$$

$$\kappa(u)N(u) = \frac{P'(u) \times P''(u)}{\|P'(u)\|^3} \quad \text{in } \Re^2$$

Equation 11: Curvature function of parameter *u*

6.1.3 Bi-normal and Torsion

The tangent vector *T* is a unit vector; the principal normal *N* are orthogonal to *T*. The vector product $T \times N$ is called the bi-normal of *P*, noted *B*. The bi-normal *B* is well defined when the curvature *k* is non-zero.

$$B(s) = \frac{P'(s) \times P''(s)}{|P'(s) \times P''(s)|} = \|P'(s) \times P''(s)\|$$

Equation 12: Bi-normal function of arc *s*

The bi-normal is perpendicular to the osculating plane and its rate of change is expressed by the vector

$$B'(s) = \frac{d}{ds}\big(T(s) \times N(s)\big) = \frac{dT(s)}{ds} \times N(s) + T(s) \times \frac{dN}{ds} = T \times N'$$

$$\text{Where} \quad \frac{dT(s)}{ds} = P''(s) = k(s)N(s)$$

Equation 13

Since *N* is a unit vector *N.N=1,* and *N.N'=0*. Therefore N'(s) is parallel to the rectifying plane *(B, T)* and N'(s) can be expressed as a linear combination of *B* and *T*:

$$N'(s) = \mu T + \tau B$$

Equation 14

Using Equation 13 and Equation 14,

$$B'(s) = T(s) \times \big(\mu T(s) + \tau B(s)\big) = \tau T(s) \times B(s) = -\tau N(s)$$

Equation 15

The torsion of an arc length parameterised curve is defined by the formula

$$\tau(s) = -\left\| P'(s) \times P'''(s) \right\|$$

Equation 16: Torsion function of arc *s*

With respect to the parameter *u*, the torsion is a function of the first, second and third derivatives.

$$\tau(u)=\frac{\det\left[P'(u),P''(u),P'''(u)\right]}{\left\|P'(u)\times P''(u)\right\|^{2}}$$

Equation 17: Torsion function of parameter *u*

6.1.4 Frénet-Serret Formulae

The well-known *Frénet-Serret* formulae are a direct result of the above definitions. They are defined by Equation 18: Frénet-Serret formulae for any curve that is thrice differentiable with non-vanishing second derivative.

$$\begin{aligned}
&\frac{dP(s)}{ds}=T(s)\\
&\frac{dT(s)}{ds}=k(s)N(s)\\
&\frac{dN(s)}{ds}=-k(s)T(s)+\tau(s)B(s)\\
&\frac{dB(s)}{ds}=-\tau(s)N(s)
\end{aligned}$$

Equation 18: Frénet-Serret formulae

Some implementation issues are worth mentioning. The above relations are function of the arc length *s,* hence are valid with arc length parameterised curves. However in reality curves are never arc length parameterised. To illustrate this, a plot of the first derivative magnitude $\|P'(u)\|$ against parameter *u* (Figure 31 paragraph 6.3.1) shows that parameter speed and curve speed are not uniform. It is therefore necessary to substitute the arc length variable *s* to a parameter variable *u* using Equation 19 obtained by derivation of Equation 2.

$$\frac{ds}{du}=\|P'(u)\|$$

Equation 19

The Frénet-Serret Formulae adapted to arbitrary parameterisation are written in a matrix from below

$$\begin{bmatrix} T'(u) \\ N'(u) \\ B'(u) \end{bmatrix} = \frac{ds}{du} \begin{bmatrix} 0 & \kappa(u) & 0 \\ -\kappa(u) & 0 & \tau(u) \\ 0 & -\tau(u) & 0 \end{bmatrix} \begin{bmatrix} T(u) \\ N(u) \\ B(u) \end{bmatrix}$$

Equation 20: Adapted Frénet-Serret Formulae

These adapted equations have been implemented with ACIS and give satisfactory results from a pure mathematical point of view. However the algorithm presents a significant drawback: It is a succession of derivation and normalisation, and because the normalisation is achieved with a division, it is inefficient to derive an expression that contains divisions. For this reason the so-called Frénet-Serret equations are not suitable for implementation purposes. Instead the relations involving the curve's derivatives without prior normalisation should be favoured. These are Equation 5, Equation 11 and Equation 17.

6.2 Differential Properties Interrogation

Interrogation techniques attempt to illuminate curve and surface characteristics that are not easily discernible using conventional rendering. The characteristics relevant to a curve's shape are closely related to indistinct geometric properties such as torsion and curvature.

6.2.1 Interrogation Strategy

Geometric modellers like ACIS [1], Parasolid [82] as well as the Djinn interface [38] exhibit functionality for differential geometry of curves but are incomplete. With ACIS and Parasolid, functions that return the tangent, normal and curvature of a curve at a given parameter value are present. As for the bi-normal and torsion these are non-existent. This perhaps is because curves are commonly modelled with cubic polynomials, which are curvature continuous but not necessarily torsion continuous; therefore it is assumed that the bi-normal and torsion are irrelevant. Djinn however is more complete, at page 313, a procedure called DJ Edge Curvature outputs all the differential properties defined by the Frénet-Serret equations of a given edge at a given point coordinate.

All these have limited use because they are discrete functions, which means that they return values at a given parameter value. However, many of the applications for curve optimisations developed later in this thesis need T, N, B, k and τ as continuous functions. It was found that optimisation algorithms in literature have adopted a devious strategy, *Jones* [65]. A curve is interrogated for some properties with dense sampling and the point cloud is used to interpolate a spline curve. The interpolated curve is used as graphic representation of the curve's property function, but does not carry mathematical meaning of any sort. *Munaux* and *Tiwari*, *Roy et. al.* [94], [99] have used this method extensively in curve optimisation algorithms using genetic algorithm. Their method gave satisfactory results for constructing curves with prescribed curvature profile but suffered from a lack of rigour and exactness. The problem with the dense sampling method is that it gives very little information of what happening between sampled points. The function might be discontinuous there and the analysis would miss it. This methodology has limited power but permits simple algorithms to work when no other options are available.

6.2.2 Continuous Interrogation

Recently ACIS has included the LAW class, which is a powerful facility for creating curve and surface geometry either directly in terms of parametric equation or indirectly by means of offsets, sweeps, and wraps, *Corney* [28]. A law is represented internally by a tree of C++ classes that know their dimensions, how to evaluate themselves, and how to take their exact (symbolic) derivatives with respect to any combination of variables. In addition, law utility functions numerically integrate, differentiate, and find roots. Many questions can be answered by knowing where some combination of them is maximal or minimal. The ACIS law class uses Function Representation for the definition of geometry, see paragraph 4.3, and enables accessing the true mathematical representation of curves: this is a continuous vector field function of a single parameter. Better still, these vector fields are differentiable functions and the result of a derivation is also a continuous differentiable function.

There are a great number of algorithms in the literature for doing curve optimisation, and all of them are specific to the NURBS representation and are bound to use discrete interrogation methods like the one described above when inquiring about differential properties. Clearly F-rep bears some immense potentials as a substitute to NURBS when a more fundamental description of a curve is needed. Immediate applications of F-rep to curve optimisation are the fundamental theorems of differential geometry that have been forgotten by the CAD community since the appearance of NURBS. This thesis reports on attempts to use fundamental differential geometry to serve the purpose of curve optimisation within CAD and documents the interface functions created for this purpose. Some of these have been implemented using the existing ACIS functionality while others, more wishful thinking, are defined and discussed but have not been implemented purely because of the limited functionality of the geometric modeller, ACIS.

6.3 Differential Properties Representation

The differential properties of a curve are obtained from the Frénet-Serret equations developed above. These properties must be represented and conveyed to the designer in some ways. The properties which are vector valued are best represented geometrically in the graphic interface. Other information such as discontinuities and topology could be represented elsewhere in text format. This paragraph presents some several techniques for graphic representation of differential properties. These are classified into two categories, indirect and direct, *Moreton* [73].

6.3.1 Indirect Methods

Indirect methods are characterised by plots of properties separately from the subject curve.

Intrinsic Equation Plots

A typical example of an indirect method is plotting a 2D curve showing curvature k versus parameter u of a curve $P(u)$; Figure 29 is actually a parametric vector equation of the form:

$$P(u) = \begin{pmatrix} u \\ k(u) \\ 0 \end{pmatrix}$$

Equation 21: Curvature plot

Since $\kappa(u) = \|T'(u)\|$, $\kappa(u)$ is always positive, Points at which the curvature is zero are called points of inflection. *Nutbourne* [80] mentions that if there is an S-bend in the curve, the vector *T'(u)* changes sign as the curve passes through a point of inflection. Thus the vector N(u) is flung from one side of the curve, this also implies that the bi-normal vector reverses direction. In presence of a planar curve this is not a desirable behaviour and he suggests an alternative definition of the normal vector to be either in the same direction as $T'(u)$ or in the reverse direction at some key points and thereafter maintain its presence, even at a point of inflection without any sudden reversals of direction. An advantage of this redefinition is that the curvature can now be signed and the graph of *k* as a function of *u* may cross the *u* axis smoothly. This redefinition of curvature is only valid for planar curves. An S-bend curve with torsion would not have null curvature at the inflection point, rather the bi-normal will swing around the curve.

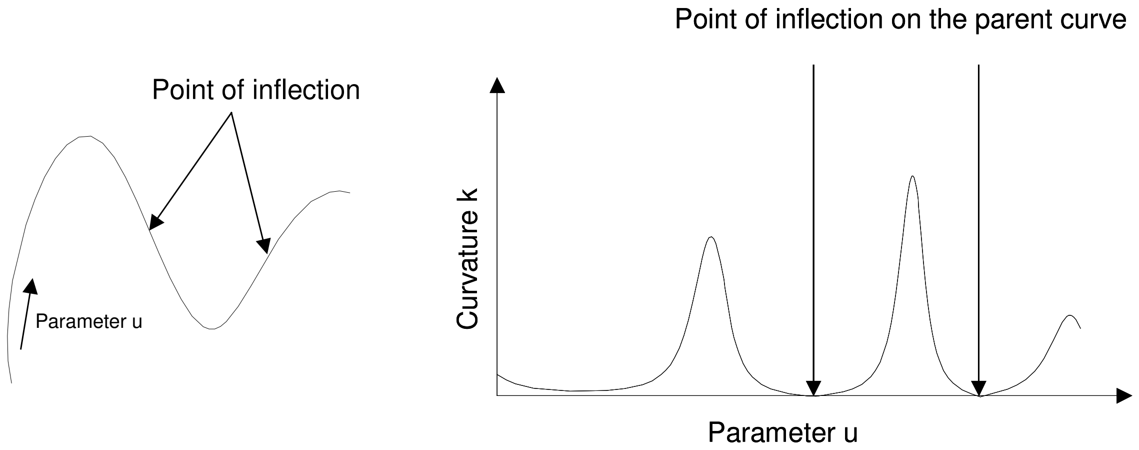


Figure 29: Curvature plot

An alternative to curvature is plotting the radii of curvature, which undoubtedly is more geometrically intuitive than the curvature metric. The main disadvantage with these kinds of plots is that the radii of curvature tend to vary over a large range. Also if the analysed curve exhibits flat sections, the radii will be infinite. Therefore it is necessary to limit the range of the plot.

Planar curves have null torsion however with space curves it is often necessary to look at the torsion plot. A torsion plot, presented in Figure 30 is obtained analogously to a curvature plot, its vector functions becomes:

$$P(u) = \begin{pmatrix} u \\ \tau(u) \\ 0 \end{pmatrix}$$

Equation 22: Torsion plot

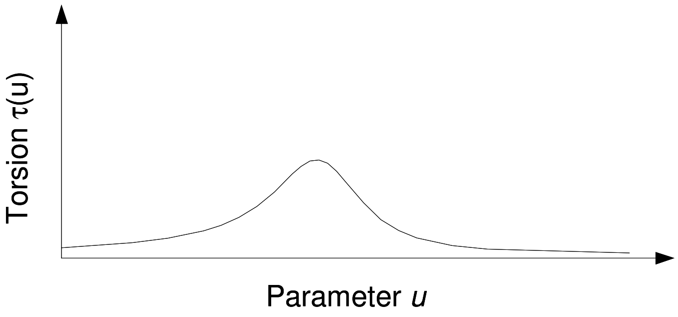


Figure 30: Torsion plot

Derivative Plot

In those applications, it is important to assess the uniformity of a curve’s parameterisation. For example, in CNC milling operations the tool path should be uniformly parameterised so that the cutting tool does not speed up or slow down while machining. Evaluation of a curve for cutter control is achieved by examining the plot of first derivative magnitude against parameter. The plot of first derivative magnitude, Figure 31, shows that the parameterisation is continuous but not uniform.

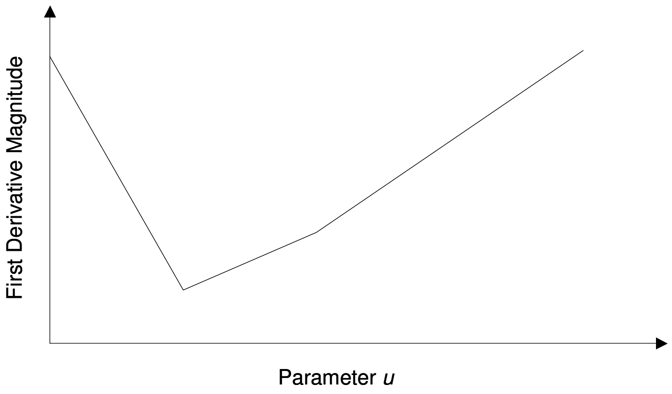


Figure 31: First derivative magnitude against parameter *u*

6.3.2 Direct Methods

Direct methods come into two major sub-categories, *Moreton* [73]: mapping techniques and compound rendering. Mapping techniques map a curve into a new curve. Polar mapping produces a curve with cusps that correspond to inflection points in the subject curve. Yet another measure for planar curves includes K-orthotomics curves, which are well known in the field of optics, *Hoscheck* [62]. Compound renderings result from displaying the geometric measures of a curve combined with the curve itself. It was already established that any curve could be represented as a continuous vector function. From there, it is easy to perform the basic vector algebra operations, introduced in Appendix 4. Several examples of compound rendering techniques are presented here.

The first example discussed is curvature fin. The normal vector of a curve at a given arbitrary parameter value u_0 is in the direction of the centre of the curvature circle. The product of curvature value $k(u_0)$ at this point with the normal $N(u_0)$ indicates the curvature distribution along the curve as well as the normal direction. Because the values of k are small, it is necessary to scale the product $k(u)N(u)$ by a constant s for visualisation purposes. Because of the necessary scale factor introduced, in some regions of the curve, the length of the vector $sk(u)N(u)$ might be greater than the radius of curvature. This causes the compound curve to make self-intersecting loop. In order to avoid this, the negate of $sk(u)N(u)$ is preferable. A compound curvature fin is obtained by adding the curve vector function with the above product function.

$$\text{curvature fin} = P(u) - sk(u)N(u)$$

Equation 23: Curvature fin compound

In the more general case of space curves, torsion also determines the curves shape. On the same basis as the curvature fin presented above, a torsion fin compound is obtained from the following equation:

$$\text{torsion fin} = P(u) + s\tau(u)B(u)$$

Equation 24: Torsion fin compound

It was established in previous chapters that torsion $\tau(u)$ is orientated along the bi-normal *B(u)* on the rectifying plane. An example of curvature and torsion fins is presented in Figure 32. The curve interrogation is not carried out by discrete sampling but rather by continuous differentiation. The segment lines joining the subject curve and the compound curves do not show points where the curve was sampled, they are purely for illustrative purposes.

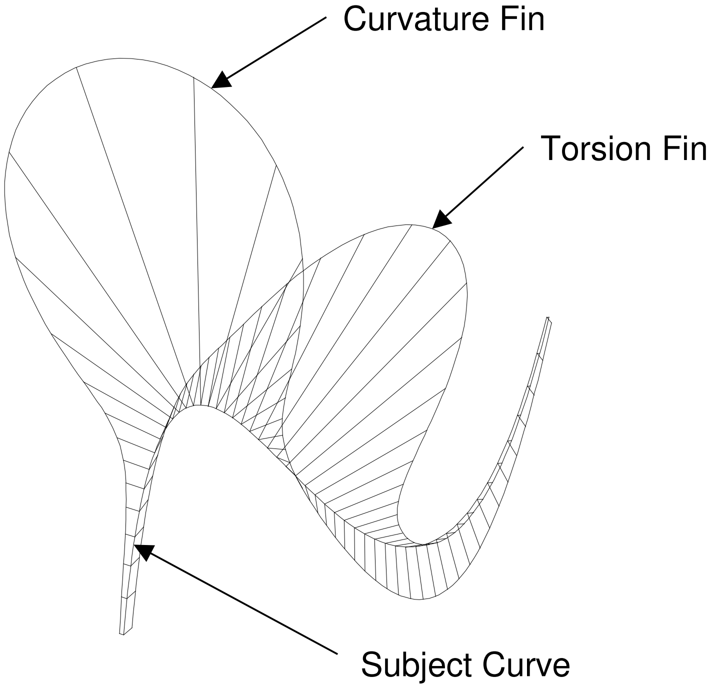


Figure 32: Curvature and torsion fins

It should be noted that such techniques are in effect creating a non-uniform offset curve to the original. Comparing these approaches with the indirect methods, attaching a fin whose width is proportional to the geometric measure has some advantages. Primarily, the curvature and torsion convey higher order quality information more clearly. It is possible to detect regions of undesirable curvature as well as to assess the overall curvature distribution. The main drawback is that it is necessary to appropriately scale the measures. Moreover it is difficult to detect inflections in areas of extremely low curvature. Considering only the normal and bi-normal directions, they produce a uniform offset that is at a constant distance. Since both vectors are unit (length of one unit) it is also necessary to scale those.

The normal fin is obtained from the equation:

$$\text{normal fin} = P(u) - sN(u)$$

Equation 25: Normal fin compound

Similarly, the bi-normal fin is obtained from the equation

$$\text{bi - normal fin } = P(u) - sB(u)$$

Equation 26: Bi-normal fin compound

This method does not convey any information about curvature distribution but shows inflections when using a sufficient offset distance, see Figure 33.

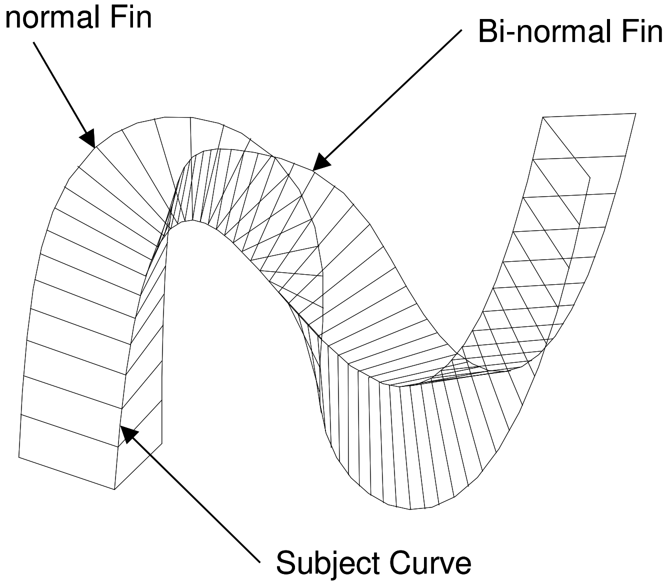


Figure 33: Normal and bi-normal fins

6.3.3 Discussions

The Frénet-Serret formulae give useful information on high order differential curve properties, but present limitations. In the presence of vanishing curvature the Frénet frame is no longer defined. The normal compound fin of a curve with a s-bend the is a null vector at the points of inflection and therefore the normalisation is not defined. This is a real problem because the compound fin shows a discontinuity where there is not.

More generally, where a curve has a flat section, its curvature vanishes and the Frénet frame is not defined. It becomes necessary to find a trihedral orthogonal frame whose orientation is arbitrary or governed by other laws than the Frénet-Serret equations and that is defined in regions of null curvature.

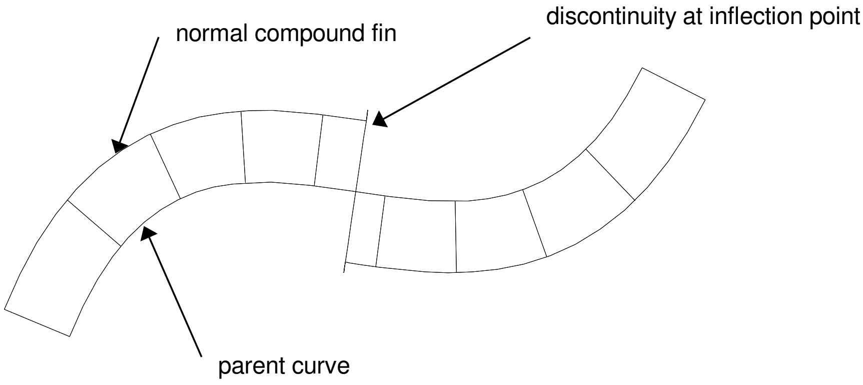


Figure 34: Normal compound fin

6.4 Parallel Transport Frame

An alternative approach to the Frénet curve framing is the parallel transport frame proposed by *Bishop* [14]. Typical applications of the parallel transport frame include the generation of ribbons and tubes from 3D space curves, and the generation of forward-facing camera orientations, *Hanson*.[58].

Parallel Vector Fields

Given a space curve *P(s)* parameterised by arc length *s*, a vector field *V(s)* is normal to the curve if it is everywhere perpendicular to the curve's tangent *T(s)*. A normal vector field is said to be parallel to the curve if it's derivative is tangential along the curve; that is *V'(s)*||*T(s)*. Such a vector field turns only as much as necessary for it to remain normal.

More generally an arbitrary vector field *V(s)* along a curve *P(s)* is parallel if its normal component is parallel and its tangential component is a constant multiple of the unit tangent field of *P(s)* The curve $Q(s) = V(s) + P(s)$ is parallel curve of *P(s)*.

Properties of Parallel Vector Fields

A curve *P(s)*, a parallel normal vector field *V(s)*, and the corresponding parallel curve *Q(s)* have the following properties:

- *V(s)* has constant length
- *V(s)* is perpendicular to both *P(s)* and *Q(s)*
- An initial normal vector V_0 at a point $P(s_0)$ generates a unique parallel field *V(s)* on *P(s)* such that $V(s_0) = V_0$
- If normal vectors V_0 and U_0 generate parallel fields *V(s)* and *U(s)* respectively, the angle between *V(s)* and *U(s)* is constant along the curve, that is $V(s).U(s) = V_0.U_0$ for all *s*.

Bishop Frame

The basic concept of the parallel transport frame, called Bishop frame here, is to observe that while $T(s)$ is unique, any convenient basis $(N1(s), N2(s))$ in the plane perpendicular to $T(s)$ can be chosen at each point. If the derivatives of $(N1(s), N2(s))$ depend only on $T(s)$ and not on each other, $N1(s)$ and $N2(s)$ can vary smoothly through the path regardless of the curvature. The following equations define the Bishop frame:

$$\begin{bmatrix} T'(s) \\ N_1^{'}(s) \\ N_2^{'}(s) \end{bmatrix} = \frac{ds}{du} \begin{bmatrix} 0 & k_1(s) & k_2(s) \\ -k_1 & 0 & 0 \\ -k_2 & 0 & 0 \end{bmatrix} \begin{bmatrix} T(s) \\ N_1(s) \\ N_2(s) \end{bmatrix}$$

$$\kappa(s) = \left((k_1)^2 + (k_2)^2\right)^{1/2}$$

$$\text{with} \quad \theta(s) = \arctan\left(\frac{k_1}{k_2}\right)$$

$$\tau(s) = -\frac{d\theta(s)}{dt}$$

Equation 27: Bishop Equations

6.5 Families of Curves

This paragraph shows a series of curves of different types, with a range of geometric features analysed with the new differential properties representation approaches. The aim of this material is to demonstrate how the new approaches can be used to analyse the geometry of curves. Also it serves the purpose of validating the algorithms implemented in HulaHoops, the CAD system developed in this research. Detailed information on HulaHoops is presented in Chapter 9 and Chapter 10.

Later in Chapter 8, the differential properties used to examine the geometry of curves are implemented in curve optimisation applications, which use these as constraints. The constraints prescribed by the user are used for recreating optimised curves.

6.5.1 Feature Curves

Feature curves are classified upon their dimension: planar or space and their curvature convex properties.

Convex Planar Curves

A planar convex curve is a curve embedded in a plane, thus has zero torsion. Its convex property state that it is monotonous, that is its signed curvature is the same sign every where. The curvature compound fin in Figure 35 is around the outside of the curve

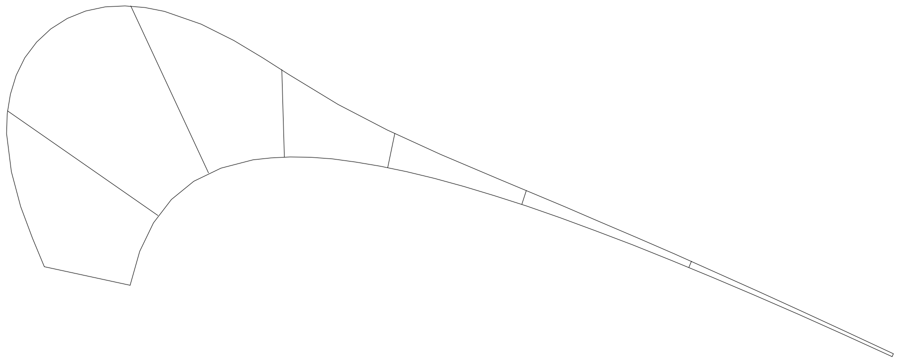


Figure 35: Convex curve

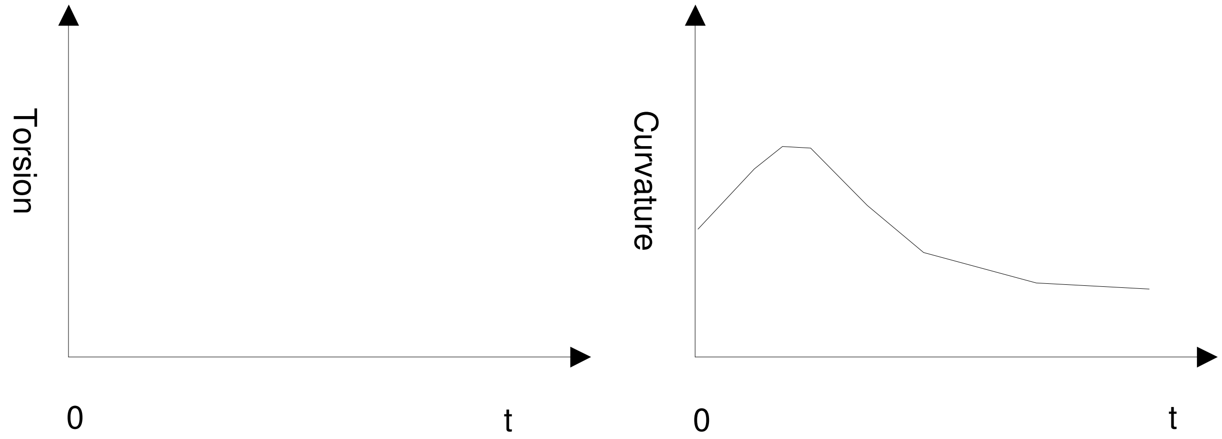


Figure 36:Convex curve curvature and torsion profiles

Convex Space Curves

Space curves have non-zero curvature and torsion profiles. The curvature profile, Figure 38, does not come to the zero value and the normal fin compound, Figure 37, does not swing direction, so: the curve is convex. Also the torsion profile is visibly showing a rise in curvature in the middle of the curve.

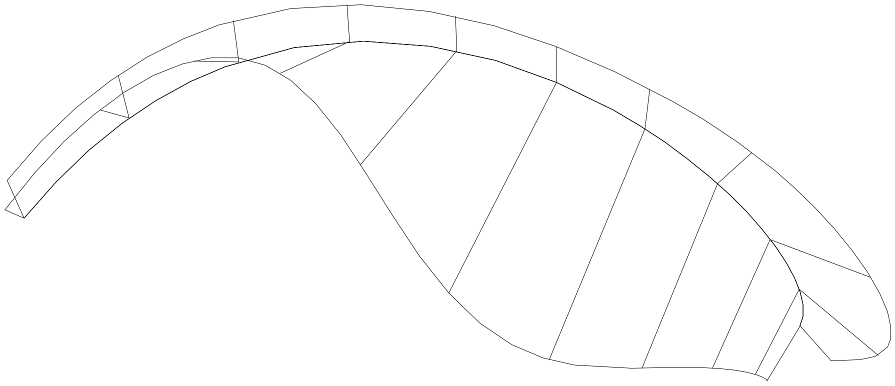


Figure 37: Convex space curve

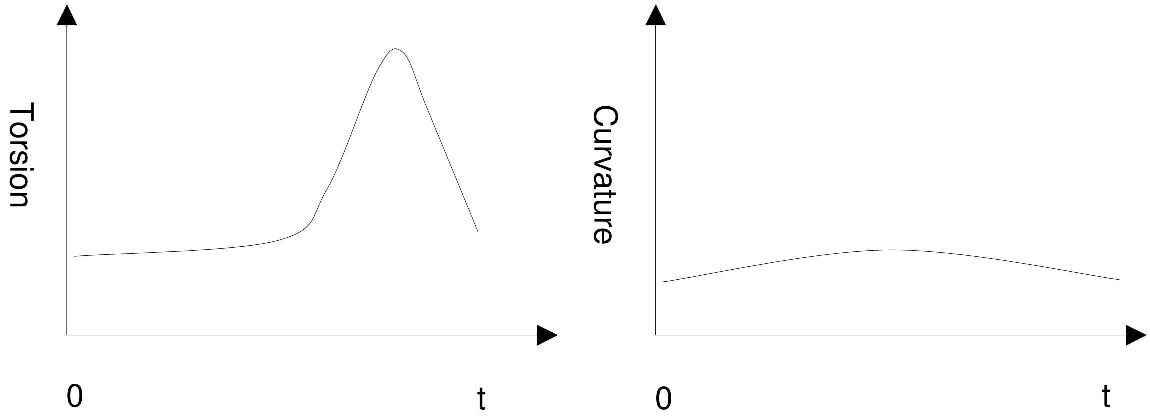


Figure 38: Convex space curve curvature and torsion profiles

S-bend Planar Curves

S-bend curves are curves which change normal direction, this is visible on the curvature compound fin Figure 39.

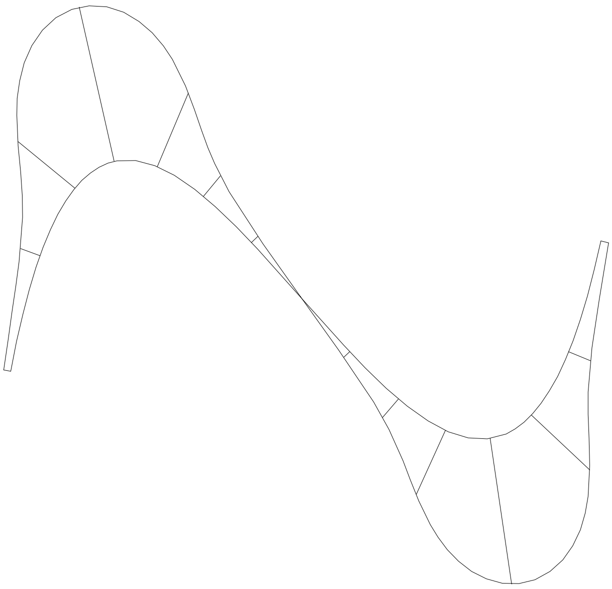


Figure 39: Planar S-bent curve curvature compound fin

The inflection point is the point where the compound fin crosses the curve. At this point the normal vector swings around the curve. As discussed in paragraph 6.3.3, this feature curve shows a deficiency of the Frénet formulation: The normal vector is not defined at the inflection point because it is the norm of the curvature vector, which is null at this point. The normal compound fin irregularities at the inflection point proves the presence of a discontinuity in the function. For this reason, the normal fin compound is not considered for engineering applications.

The curvature profile in Figure 40 has two picks of curvature on either side of the inflection point where curvature is zero.

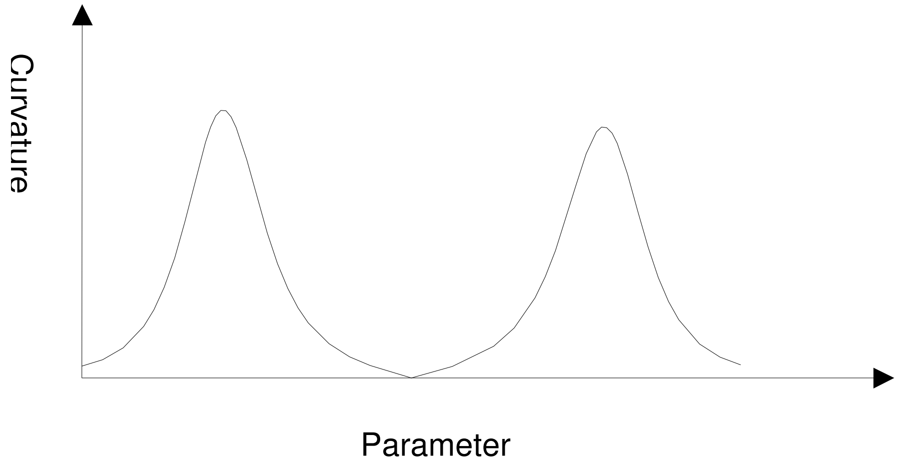


Figure 40: Planar S-bend curvature profile

S-bend Space Curves

Analysis of the S-bent curve with non-zero torsion reveals the behaviour of the torsion vector on either parts of the inflection point. As the bi-normal vector swings around the curve, the torsion increases in great proportions with its maximum at the inflection point.

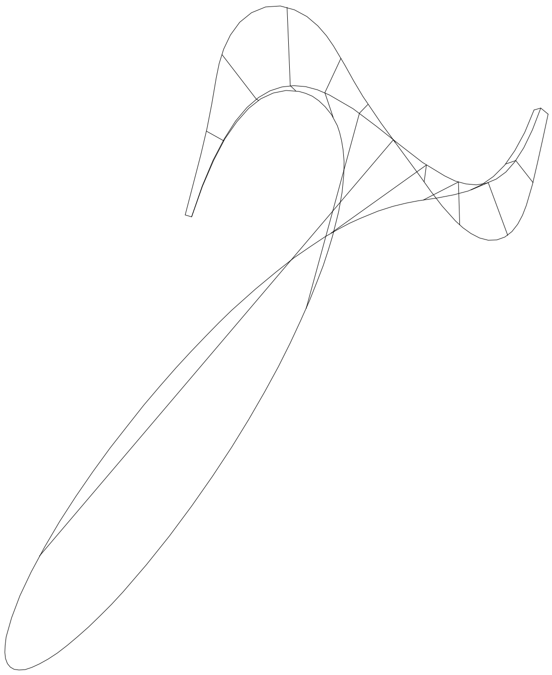


Figure 41: Space S-bend curvature and torsion compound fins

For the space S-bend curve, the curvature and torsion profiles are plotted on the same graph, Figure 42. The curvature profile is similar from the previous example, which was a planar S-bend curve, only this time the curvature profile does not touches the zero line where there was an inflection point before. This indicates that the curvature is never null. However the torsion picks where curvature is minimum, so the twist is greater where curvature is lesser.

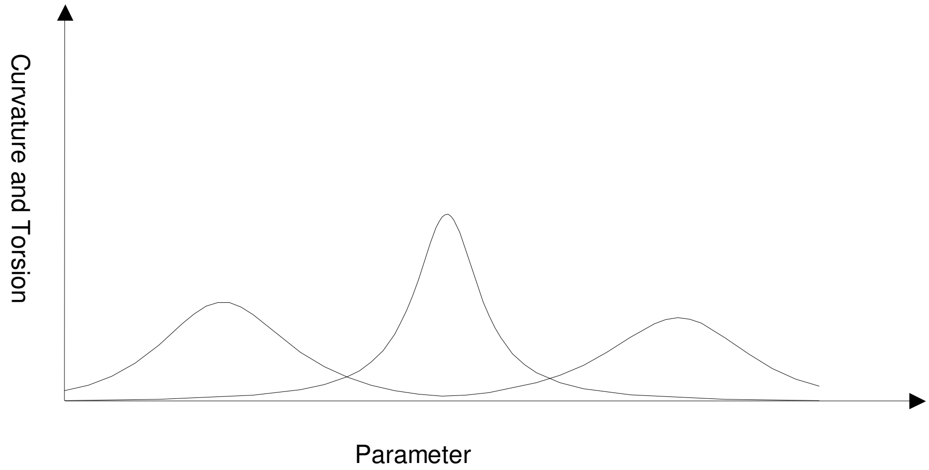


Figure 42: Curvature and torsion profiles

6.5.2 Helical Offsets

Helical offsets are obtained by controlling the offset distance and the twist along the parent entity. This offsets allow the construction of helices. Helical curves are special curves whereby the curvature and the torsion are proportional to each other. Helices give rise to a great variety of curves, some of their representatives are examined with the new approaches introduced earlier in this chapter.

Helix

The simplest case is the helix is a curve whose curvature and torsion profiles are non-zero constant. An example of Helix from Figure 43 shows the curve is drawn on to a cylinder whose cross-section is a circle. Also the curvature and torsion compound fins indicate that helix has constant curvature and torsion. This is confirmed by the curvature and torsion plots in Figure 43. Both plots are constant and torsion is greater than curvature, which means that the thread twist is more than the curvature radius.

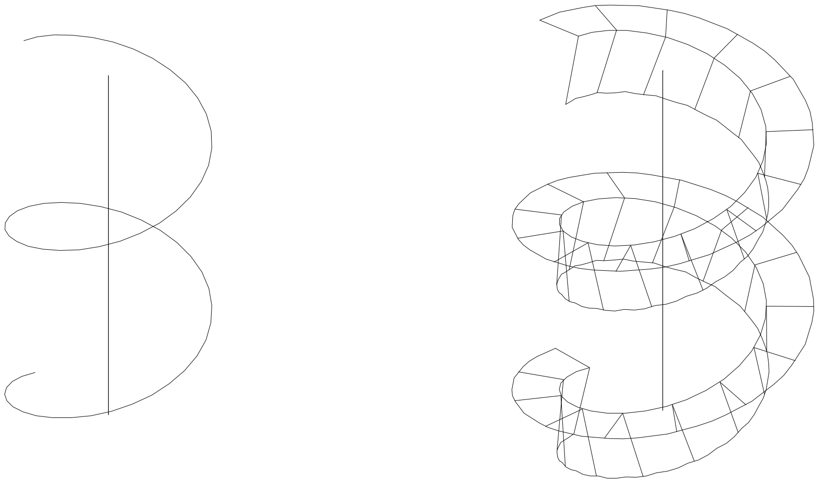


Figure 43: Helix curvature-and torsion compound fins

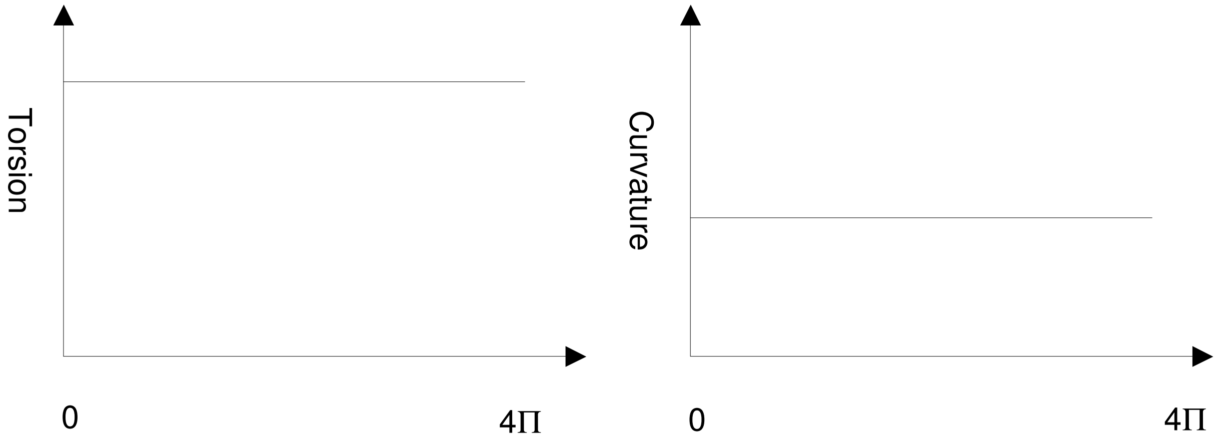


Figure 44: Helix curvature and torsion profiles

Expanding Helix

The expanding helix has an arbitrary curvature profile and a torsion profile that is proportional to the curvature profile. The compound fins, Figure 45, show that the curvature and the torsion are greater at the beginning of the curve than at the end, but are proportional.

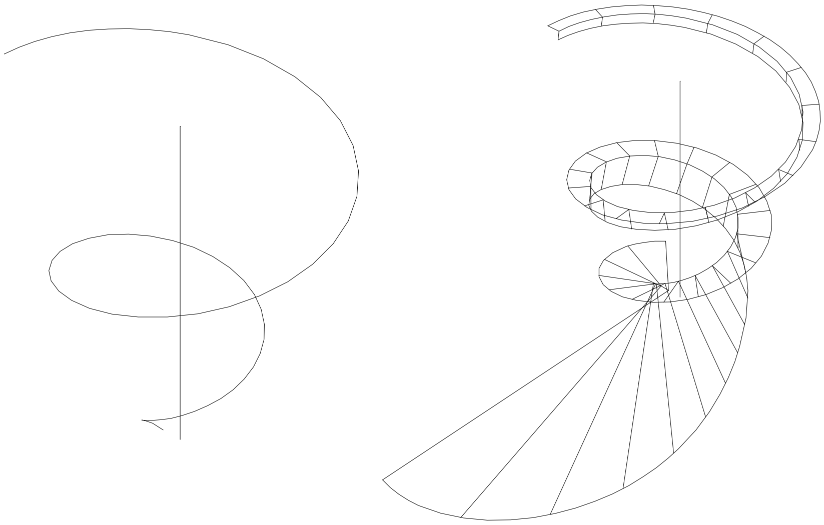


Figure 45: Expanding helix curvature and torsion compound fins

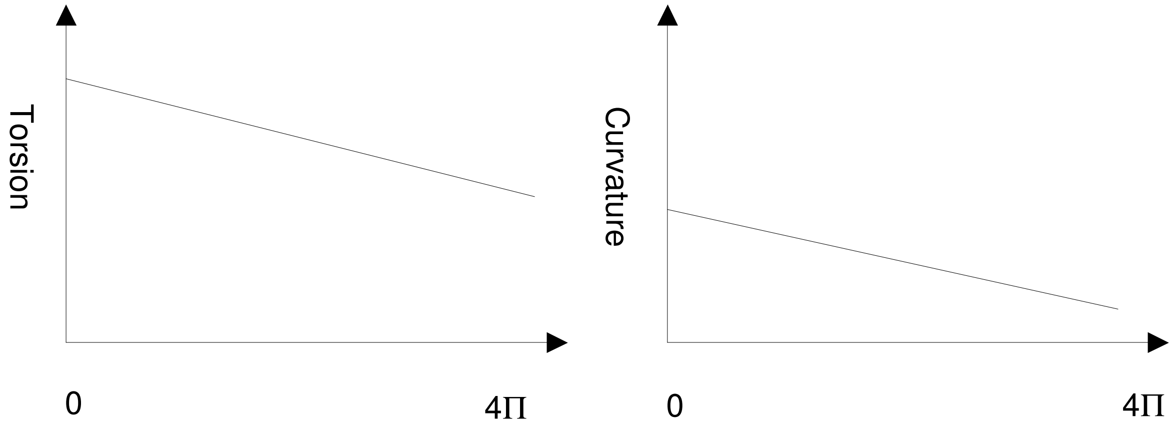


Figure 46: Expanding helix curvature and torsion profiles

Toroidal Round Helix

The toroidal round helix is constructed by offsetting a circle by a constant distance. The result is curve that curls around the parent circle. The curvature and torsion profile in Figure 48, indicate that both profiles are cyclic and are in opposition of phase.

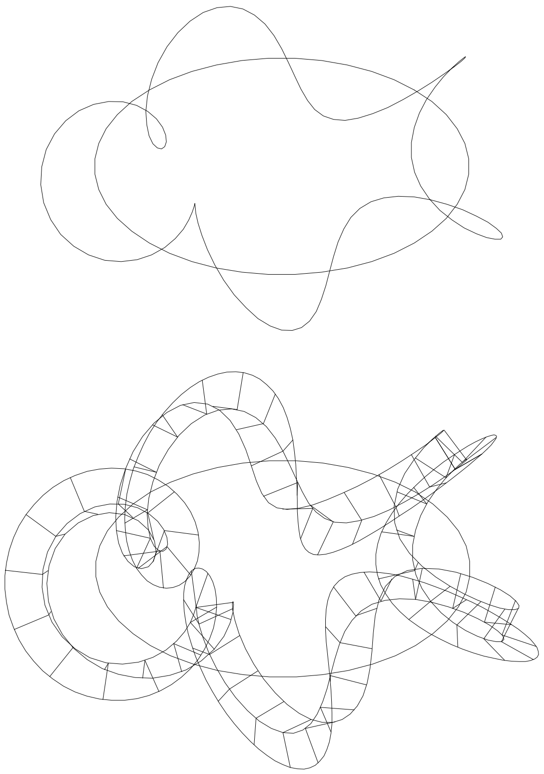


Figure 47: Toroidal round helix Frénet frame fins

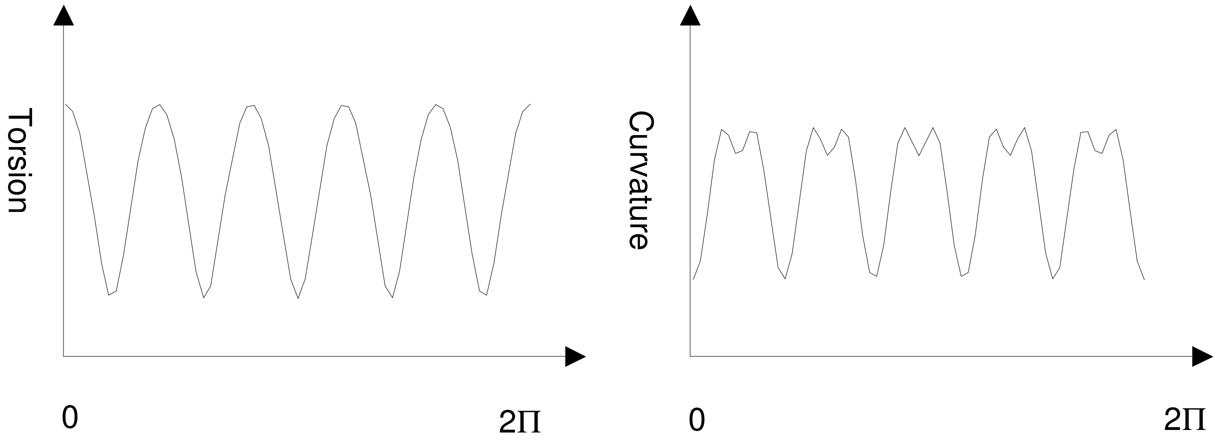


Figure 48: Toroidal helix curvature and torsion profiles

Chapter 7 Spline Curve Construction and Optimisation

The construction of piecewise curves implies providing additional data other that the interpolation points coordinates. These constraints give the opportunity to gain control on the curve's shape. This Chapter explores means of constructing spline curve piecewise using constraints defined at the joining knots. Further, ways of representing the constraints for optimisation is also proposed. For spline curve optimisation, position, differential and parameter constraints are considered.

7.1 Piecewise Construction

It is not feasible to make a long curve or complicated surface with one high degree polynomial. The problems come from difficult parameterisation and that high degree polynomials are computationally expensive (because of the number of terms). They are sensitive to inaccuracies and tend to give wiggly curves. To overcome these problems, parametric curves are best described as a piecewise phenomenon. This means that one curve is defined by several curves (spans) joined together in a continuous manner. The degree of continuity achievable between curve spans is one less than the degree of each spans. At best

- Quadratics join with C_1 continuity
- Cubic join with C_2 continuity
- Quintic join with C3 continuity

7.1.1 Control Points

The degree of the curve is one less than its order; i.e. the number of control points for each segment. A quadratic is defined by three points, a cubic four, and so on. In other words:

$$Ks = Ki - n$$

$$\begin{aligned} \text{where } \mathrm{K}s &= \text{number of segments} \\ \mathrm{K}i &= \text{number of control points} \\ n &= \text{degree of the curve} \end{aligned}$$

In B-spline construction, adjacent segments share control points. The number of common points is determined by the degree of the curve, thus the type of continuity in which segments blend.

Unlike Bézier curves which are totally defined by a set of control points, B-splines require both the degree of the curve and the parameter values at which the segments meet (knots) before the blending function can be evaluated. It should be noted that Bézier vertices and B-splines are different, except in certain cases when the whole B-spline is one single cubic Bézier span.

7.1.2 Knot Construction

Parameterisation

Explicit approximation is often used in applications, but since the geometric entities we are concerned with at present are of parametric form, parameterisation becomes unavoidable. In designing free form curves, positions, tangent vectors (and others) are known constraints. Still when parametric equations are used, a parameter value must be assigned to each data point. The resulting quality of the curve is heavily dependent on the parameterisation; hence an appropriate parameterisation method is required. The problem is straight forward in the case of Lagrange interpolation: The parameter can be evenly spaced along the curve for each data point, provided that data points are equally spaced, which in practice is never the case.

In general, the problem of parameterisation lies in finding the curve length between two data points and subsequently assigning parameter values to each of them. In the more general case of curve fitting through an ordered set of scattered data, an initial cord length parameterisation is commonly used for parameter estimate of the curve's arc length.

The cord length parameterisation method works well in most cases but presents weaknesses with sharp corners *Cohen* [26]. Alternatives solutions are centripetal and the *Foley* methods (see *Farin's Book* [44]). Also, in presence of dense data (obtained from digitisation for example), chord length parameterisation is difficult and other approximation techniques are favoured.

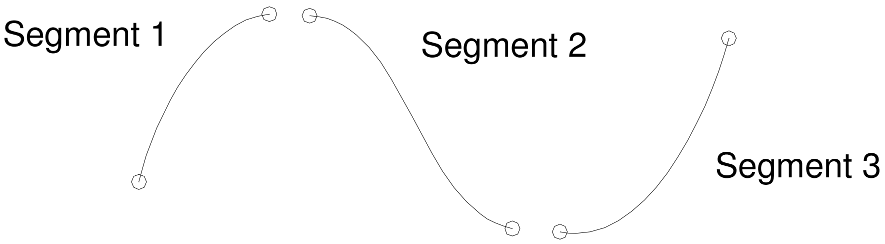


Figure 49: Knotted curve

The end point of one span is the start point of the next, these are known as *knots*, Figure 49. Each knot has a parameter value associated with it. In order to construct a curve piecewise construction, a knot sequence must be provided. The knot sequence, often called knot vector, depends on the number of control points and the degree of the curve:

$$Kp = Ki + n - 1$$

$$\begin{aligned}\text{where } Kp &= \text{number knots}\\ \mathrm{K}i &= \text{number of control points}\\ n &= \text{degree of the curve}\end{aligned}$$

Equation 28: Knot sequence calculation

The values in the knot sequence must never decrease but can be repeated. The number of repetitions is referred to as the knot multiplicity. Multiple knot values reduce the degree of continuity by one for each repeat. This should be avoided as it might create undesirable discontinuities in the curve. However this property is commonly used in B-spline construction to make the curve interpolate the start and finish control points. In the case of a cubic, the first and last three-knot values repeat; it would be the first and last two-knot values for a quadratic. The highest curve degree possible is one less than the number of control points. It was mentioned before that Bézier curves are a special case of B-splines. The cubic Bézier, of degree three, has four control points, one segment, and two knots. Using the general B-spline formulation, its knot vector is:

$$\text{Bezier\_knot\_Vector} = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1\}$$

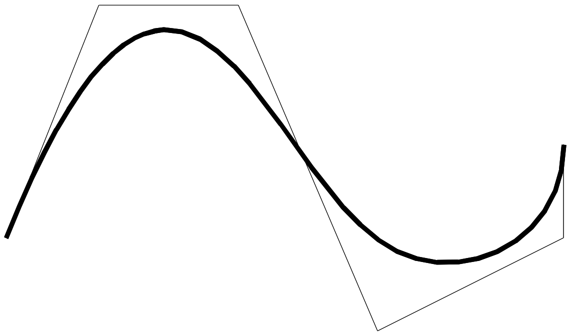
Illustrations

The following examples demonstrate the effect of parameterisation on a cubic curve created from the control point vertices below. The start and end parameter values are kept constant, the internal knot values are variable.

```
const position pt1  (0, 0, 0);
const position pt2  (10, 25, 0);
const position pt3  (25, 25, 0);
const position pt4  (40, -10,0);
const position pt5  (60, 0, 0);
const position pt6  (60, 10, 0);
```

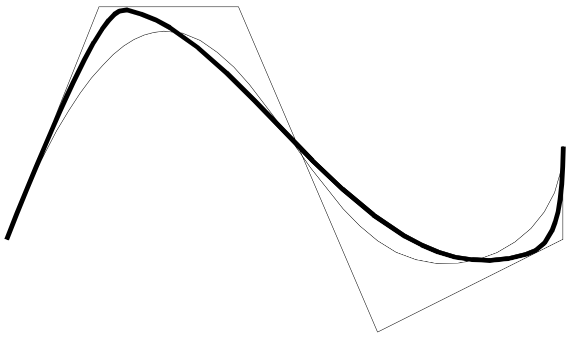
This knot vector creates three curve segments of equal parametric range and produce a smooth curve.

```
double const knots[nb_knots] = {0,0,0,0.33,0.66,1,1,1}
```



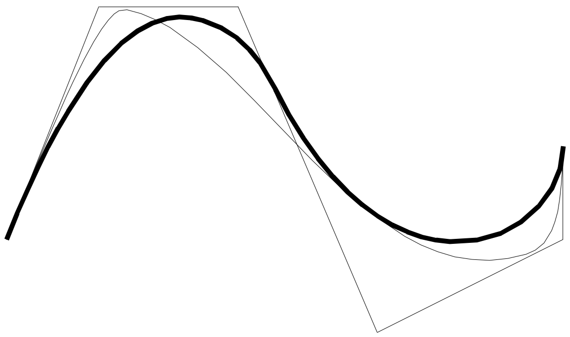
Here the first and last segments have a short parameter range. This produces a distorted curve that is concentrated near to the end points and spread in the middle.

```
double const knots[nb_knots] = {0,0,0,0.1,0.9,1,1,1}
```



The first two segments have a long parameter range followed by a long one. The difference from the previous curve is noticeable.

```
        double const knots[nb_knots] = {0,0,0,0.1,0.2,1,1,1};
```



Internal Knot Multiplicity

An example of internal knot multiplicity resulting in curve irregularity is presented here. A general cubic B-spline is generated from six control points as follows:

```
        int const Nb_CtrlPpts = 6;
        const position pt1  (0, 0, 0);
        const position pt2  (10, 25, 0);
        const position pt3  (25, 25, 0);
        const position pt4  (40, -10,0);
        const position pt5  (60, 0, 0);
        const position pt6  (60, 10, 0);
```

and the following knot vector: the internal knot has a multiplicity of two

```
        int const degree = 3;
        int const nb_knots = Nb_CtrlPpts + degree - 1 ;
        double const knots[nb_knots] = {0,0,0,0.33,0.33,1,1,1}
```

The curve is evaluated for first, second and third degree derivatives " just before", "on" and " just after" the knot where the curve is likely to be discontinuous. The derivative vectors obtained from the analysis are presented in Table 3. As predicted, the analysis shows that the second derivative vectors are discontinuous after the knot, however the curve remains C_1 continuous at this knot.

Axis	Just before	Parameter value = 0,3	Just after
	position	position	position
X	21.65	21.65	21.65
Y	18.35	18.35	18.35
	First Derivative Vector	First Derivative Vector	First Derivative Vector
X	15	15	15
Y	-15	-15	-15
	Second Derivative Vector	Second Derivative Vector	Second Derivative Vector
X	-460.055	-460.055	155.714
Y	-917.355	-917.355	-155.714
	Third Derivative Vector	Third Derivative Vector	Third Derivative Vector
X	-1394.11	-1394.11	-531.648
Y	-4449.45	-4449.45	930.633

Table 3: Derivative vectors showing curvature discontinuity

7.1.3 Curve Continuity

In a general sense, *continuity* describes how two items come together. Two types of continuity are generally discussed: C*n* and G*n*, where *n* refers to the *n*th derivative. C*n* continuity refers to continuity of the *n*th derivatives of the equations underlying the entities. This means that the magnitude and direction of the *n*th derivative must be continuous. G*n* continuity refers to continuity of geometric, or parameterisation-independent, properties, which means that only the direction of the *n*th derivative must be continuous. It should be noted that *parameter* continuity is not necessarily the same as *geometric* continuity. This is because of the arc length parameterisation problem discussed above. The difference between the two types of continuity is that G*n* allows the parameterisation to be changed to achieve desired continuity. By manipulating the parameterisation of the curve or surface, one can change the magnitude of vectors. The following list describes what is meant if two items (e.g., curves) meet with the specified continuity.

C_0 continuity

Two entities meet with C_0 continuity, when their zero derivatives are the same at their intersection, Figure 50. In the case of C_0 continuity, it may simply be said that the entities "are continuous": At all points along the intersection, the position of the entities are the same. G_0 continuity also means that the zero derivatives are the same at their intersection, but changing the parameterisation of one of the entities does not affect its position.

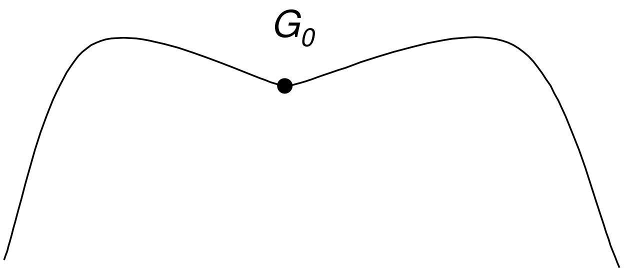


Figure 50: C_0 continuity

C_1 continuity

Means that the first derivatives, or tangents, are identical (in addition to C_0 continuity). The tangents of curves and surfaces are vectors, so both the magnitude and direction of the tangent vectors must be identical.

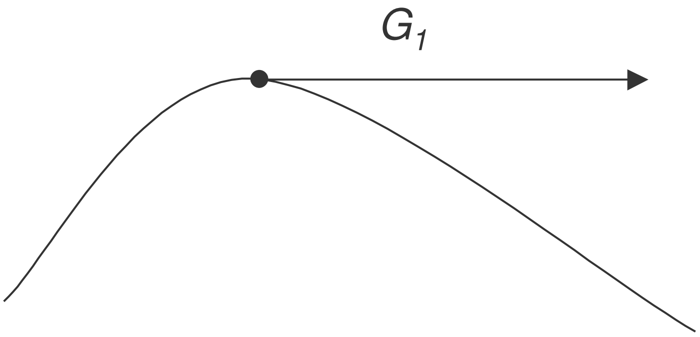


Figure 51: G_1 continuity

C_2 continuity

Means that the second derivatives agree (in addition to C1 continuity). Because curvature is a function of the first and second derivatives, one often says that the *curvature is continuous* if entities are C2.

C_2 continuity

Means that just the *direction* of the second derivatives are identical, Figure 52. The parameterisation of one of the entities can be changed to get the geometric curvatures (independent of parameterisation) to agree.

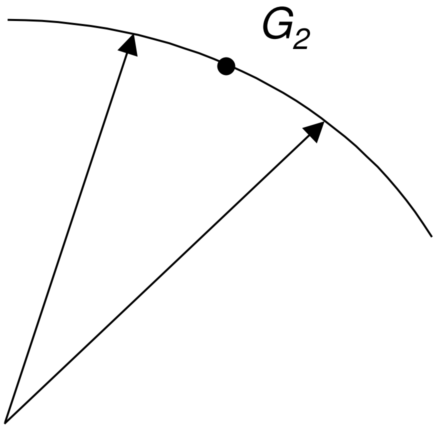


Figure 52: G_2 continuity

The derivative level, *n*, to which an object is continuous, refers to its *degree of continuity.* If a given object is continuous at the *n*th derivative, it is said to have *n*th degree of continuity (or degree of continuity *n*). To state that a given curve has a particular degree of continuity means that for *all* points on the curve's interior, the continuity is at least of that degree. The same holds for surfaces.

7.2 Knot Parameter Optimisation

In paragraph 7.1, it was established that curves can be made by joining knotted curve segments together. The main difficulty with this is finding an appropriate knot sequence to satisfy the desired shape of overall curve. Tests made in paragraph 7.1.2 showed the influence of parameterisation on a curve interpolated through fixed data points.

A simple application is created to prescribe a knot sequence by graphic representation. The knots position in the parameter space can be modified interactively to obtain a desired shape without moving the data points in the Cartesian space.

Given the data point coordinates, a first estimate is made by evaluating cord length parameterisation. This is evaluating the distance between each knot and getting the ratio of cumulated distances over the total sum of distances.

```
        const position pt1  (0, 0, 0);
        const position pt2  (40,25, 0);
        const position pt3  (80, -25, 0);
        const position pt4  (120, 0,0);
```

$$parameter_i^n = \frac{\sum_1^i cordlength_i}{\sum_1^n cordlength_i}, \quad 1 < i < n,$$

n is the number of knots

Equation 29: Cord length parameterisation

For the positions proposed above the chord lengths given by Equation 29 are as follows:

```
        Knot 1 = 0
        Knot 2 = 47.16
        Knot 3 = 111.20
        Knot 4 = 158.7
```

To gain insight on the significance of these results it is convenient to present them as a ratio of 1: The start parameter indexed *0* has value 0 and the end parameter indexed *n* has value 1. This means that the parameter knot vector is of the form [0, P1, P2,…Pi,…1]. Considering a unit length segment, the knot vector fractions it into *n-1* segments that can be represented graphically as shown in Figure 53.

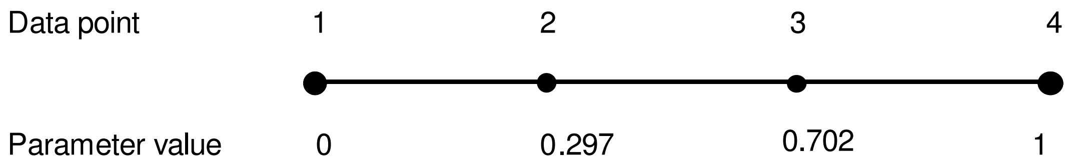


Figure 53: Cord length knot vector distribution

This technique allows designers to interactively assign parameter values at curve knots thus exposing control over the internal curve constraints in an attempt to minimise its energy. The examples presented above, show that parameterisation has some effect on the curve's shape. This curve construction method could also be used as a modelling technique implemented in a CAD environment. However it remains uncertain that designers would benefit from this application without some level of automation added to the optimisation process. The problem here is that no design goals are defined, which leaves the user with parameters to control without any indication on the achievements of his input. This is addressed in the following paragraph where the total curve energy is formulation is used as a measure of the design. The knot parameter distribution can be easily controlled manually for curves having up to two knots. But for a greater number of knots, the knot distribution is difficult to find manually, this is addressed in paragraph 7.4, where the knot parameter distribution is optimised by a GA.

7.3 Energy Formulation

The energy formulation is a measure of the curve's total energy. The formulation is the sum from start to end parameter of the curve's second derivative squared. In practice, the energy is obtained by numerical integration of curvature squared.

$$Energy = \int_{end}^{start} \left(\frac{d^2 P(s)}{ds^2} \right)^2 ds$$

s is the curve chord length $start < s < end$

Equation 30: Energy formulation

The energy formulation written in Equation 30 is tested on a range of curves. The aim of these experiments is to validate of the formulation and this is achieved by establishing the energy function map for up to two variables.

Example 1

The first example uses the curve shown in Figure 54, it is created from four data point coordinates, which are as follows:

```
        const position pt1  (0, 0, 0);
        const position pt2  (40,25, 0);
        const position pt3  (80, -25, 0);
        const position pt4  (120, 0,0);
```

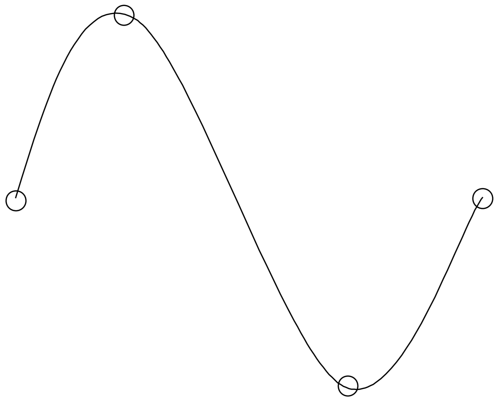


Figure 54: Test curve Example 1

Because it is created from four data points, this curve has two internal knots for which values must be assigned. They are varied incrementally in a loop algorithm and the energy is recorded in a 3D graph Figure 55. Because the knot values must be strictly increasing, knot 1 must be inferior to knot 2, hence the triangular shape base of the map. The map indicate that there is a curve which has a minimum energy for a set a knot values, and this means that the system is determined.

The overall shape of the map is quite flat in the region of minimum energy, which makes difficult for an optimisation algorithm to find the optimum. It appears clear on the graph that the function is symmetrical, due to the point coordinates arrangements. What appears to be a symmetry line running diagonally on the graph is also a region of interest because the surface lines are not smooth. A cross section of the map reveals a sudden fall of energy at some particular parameter values. Figure 56 is a cross section of the map at Knot1 = 0.2. The drop of energy is exactly at Knot2 = 0.8. Another cross section is made at Knot1 = 0.4, this time the drop of curvature is exactly at Knot2 = 0.6. For both cross sections, the drop of energy happens when two knot values add up to 1. In all cases the curves produced are regular, and as far as the author is aware, there is no clear cut answer to that phenomenon. Another test is carried out to find out if the same thing occurs again with a different curve, asymmetrical.

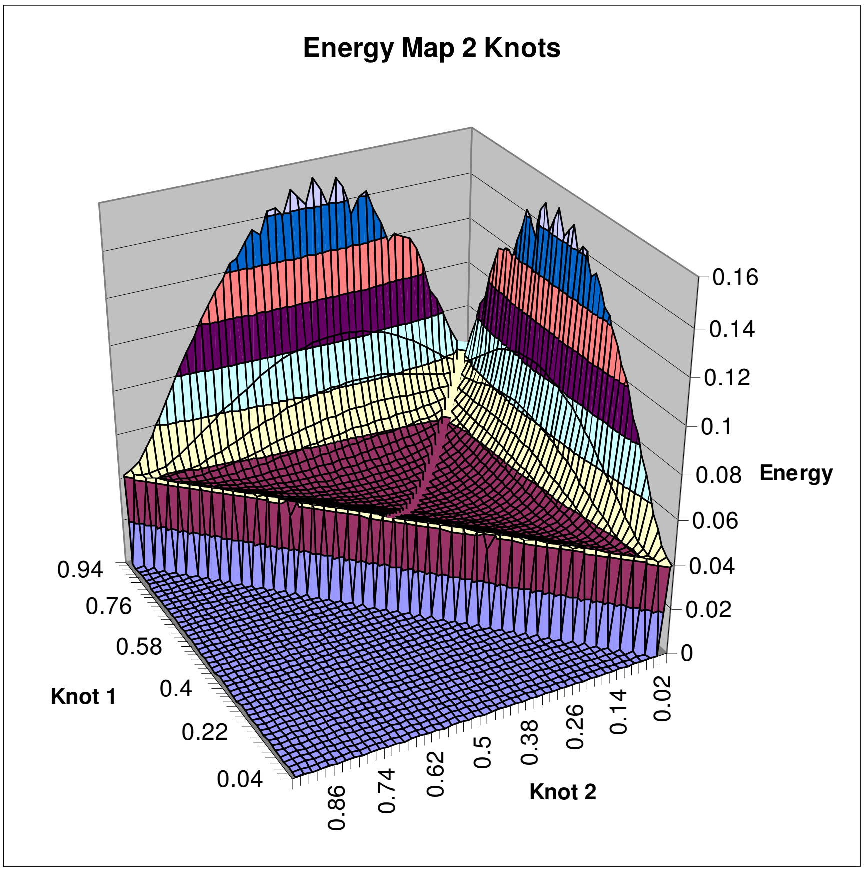


Figure 55: Energy function mapping Example 1

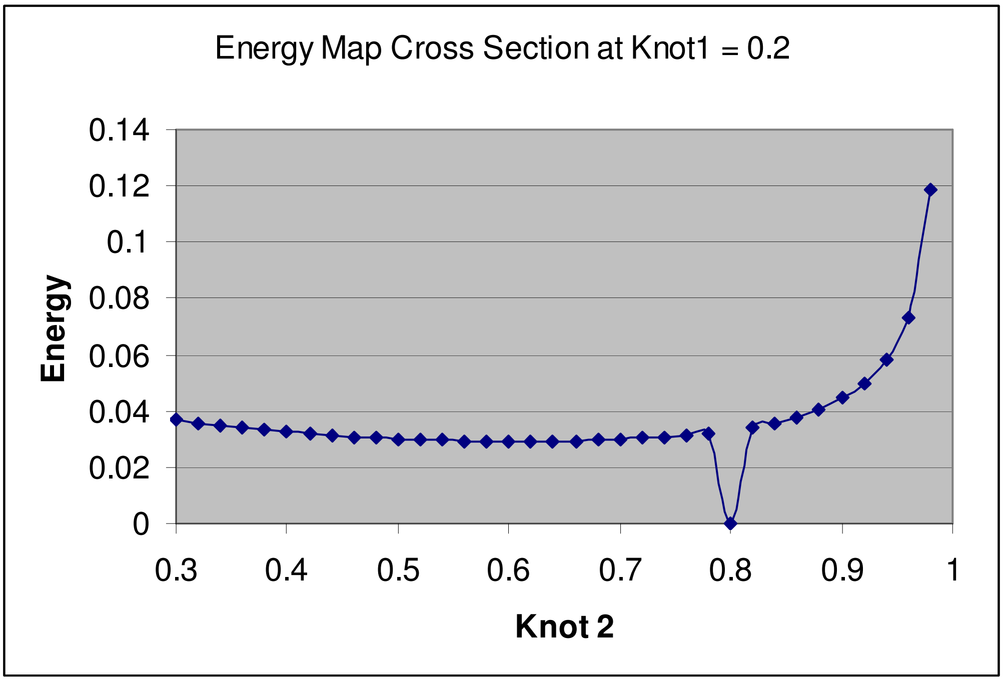


Figure 56: Energy Map Cross Section at Knot1 = 0.2 Example 1

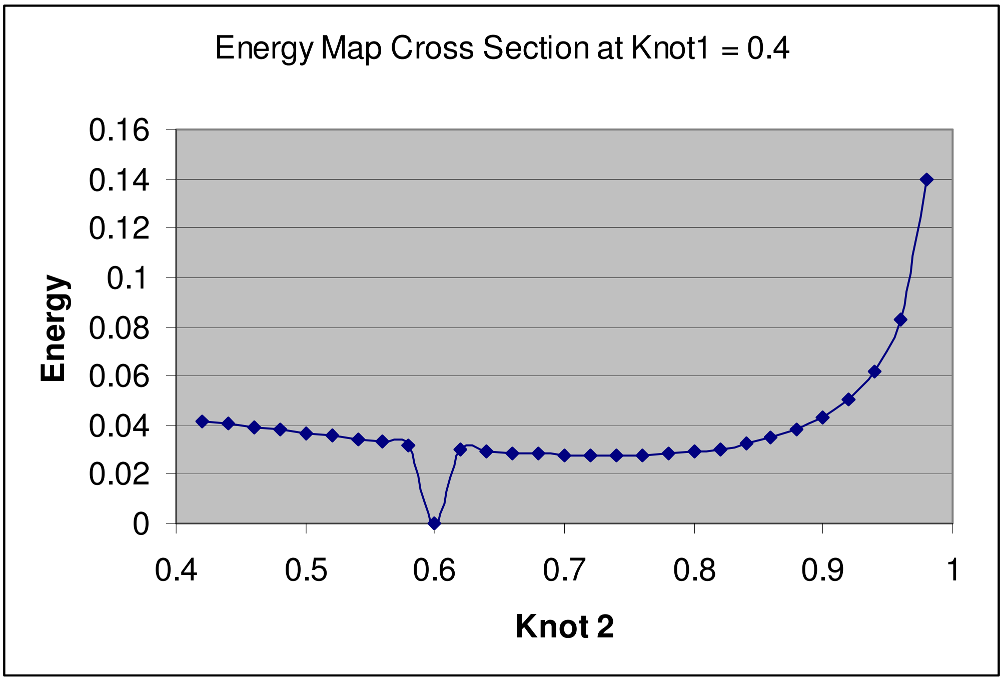


Figure 57: Energy Map Cross Section at Knot1 = 0.4 Example 1

Example 2

This test uses an arbitrary convex planar curve constructed out of four control points:

```
        const position pt1  (0, 0, 0);
        const position pt2  (40,25, 0);
        const position pt3  (80, -40, 0);
        const position pt4  (120, 0,0);
```

The third position is modified to make the curve asymmetrical. Figure 58 reveals that the energy map is smooth, there is no drop of energy at certain combinations of Knot values as observed in Example 1 above. There is no scope in this thesis for more investigations, but further work would map the energy function on greater variety of curves and document the occurrence of this problem. The curves for which the energy map exhibit any fall of energy should be closely examined to detect any irregularities in the curve definition. Most certainly, the problem comes from the ACIS function for interpolating data points with given knot values, used in this application.

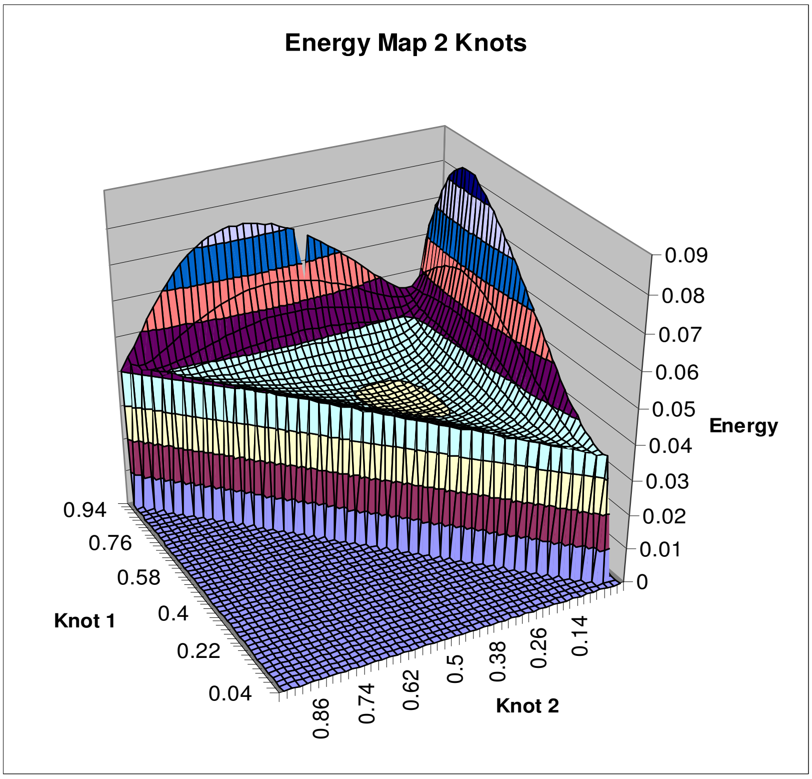


Figure 58: Energy function mapping Example 2

7.4 GA Energy Fairing

The manual parameter optimisation algorithm in the previous paragraph is applied to an automated optimisation process that finds the optimum knot vector given a set interpolated set of data points. The optimisation criterion used for evaluation is the curve's total energy. On the grounds that a smoother curve has less internal energy than a bumpy curve, the hopes are that an optimum set of parameter values, which produces a curve whose energy is minimum, can be found by an optimisation algorithm.

7.4.1 Variable bounds

The knot vector is the GA variables. Initial knot values are the cord length parameterisation, these are bounded in a manner so that the knot intervals never overlap:

```
        Knot[0]>knot[1]>knot[2]>…..>knots[I]>….>knot[n-1]> Knot[n]
```

The first and the last knot are kept constant throughout the process, therefore these are not bounded. The bound definition start at knot[1] and finish at knot[n-1]. The bounds are set as half the interval between the previous and the next bound minus the smallest value possible, which is the system resolution, to fulfil the strict inequality constraints.

```
        Arrayr lowerbound(n);
        lowerbound[0] = 0;
        lowerbound[1] = start_param+resabs;
        lowerbound[2] = param[1]+(param[2]-param[1])/2+ resabs;
        lowerbound[3] = param[2]+(param[3]-param[2])/2+ resabs;
        lowerbound[i] = param[i-1]+ (param[i]-param[i-1])/2)+ resabs;
        lowerbound[n] = param[n-1]+ (param[n]-param[n-1])/2+ resabs;
        Arrayr upperbound(numvar);
        upperbound[0] =0;
        upperbound[1] =param[2]-(param[2]-param[1])/2- resabs;
        upperbound[2] =param[3]-(param[3]-param[2])/2- resabs;
        upperbound[i] =param[i+1]-((param[i+1]-param[1])/2)- resabs;
        upperbound[n-1] =end_param-resabs;
        upperbound[n] =0;
```

Some precautions should be taken in the event one or several optimum knots values fall outside the bounds which are estimated from cord length. The bound should be re-initialised through the optimisation, based on the best solution found so far. This can be seen if a solution is found with knots close to the bounds.

7.4.2 Validation

A series of examples with features are taken to validate the parameter optimisation algorithm. Curves with high curvature gradient are undoubtedly the most challenging for Spline modelling. A representative example is of that feature is a curve going through a sharp 90° corner. The first model is set to interpolate a single knotted curve through three placed around a corner. The problem increases in complexity by adding interpolation points, they constrain the curve through more points thus restrain its freedom. The addition of geometric constraints result in making curves wiggle between points. In this experiment the number of variables taken by the GA, i.e. knot values, is pushed up to four.

Example 1

The first model is set to interpolate a single knotted curve through three points placed around a corner.

```
        const position pt1  (0, 0, 0);
        const position pt2  (100,0, 0);
        const position pt3  (100, 100, 0);
```

The curve is a cubic spline therefore the curve has one internal knot and therefore is a convex curve. From the point coordinates, the chord length parameterisation sets the knot parameter value at the middle of the parameter range.

```
        energy = 0.017471
        knot vector = {0,0.5,1}
```

The curvature compound in Figure 59 shows that the fin is around the outside of the curve and there is no change in the normal direction, which proves that the example curve is consistent with its definition.

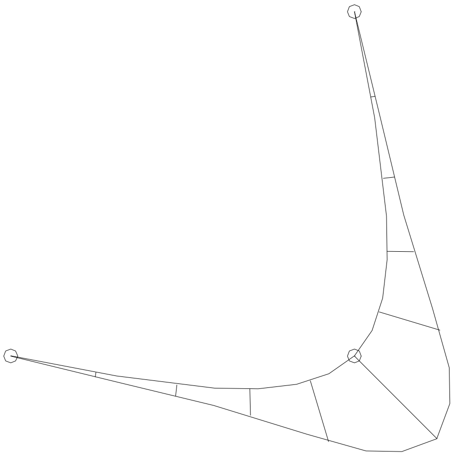


Figure 59: Original Example 1

The GA is set to its normal parameters and a test function is created to map the energy against the parameter value. The plot Figure 60 shows that the minimum energy is in the region of 0.5 on parameter value scale, which is what chord length parameterisation has indicated. Also the amplitude of energy between the best and the worst case is less than 0.01 with a ratio of 2. This indicates that the search space has low gradient and that could cause some problems for the GA.

Figure 61 shows the chronological events of the GA population. The energy recorded for individual does not seem to decrease. The solutions never are below 0.01 and the population appears to converge towards that limit which looks like a Pareto front. There are two possibilities, either the GA finds an optimum very rapidly and the rest of runs are redundant or the GA does not converge at all solutions are made randomly.

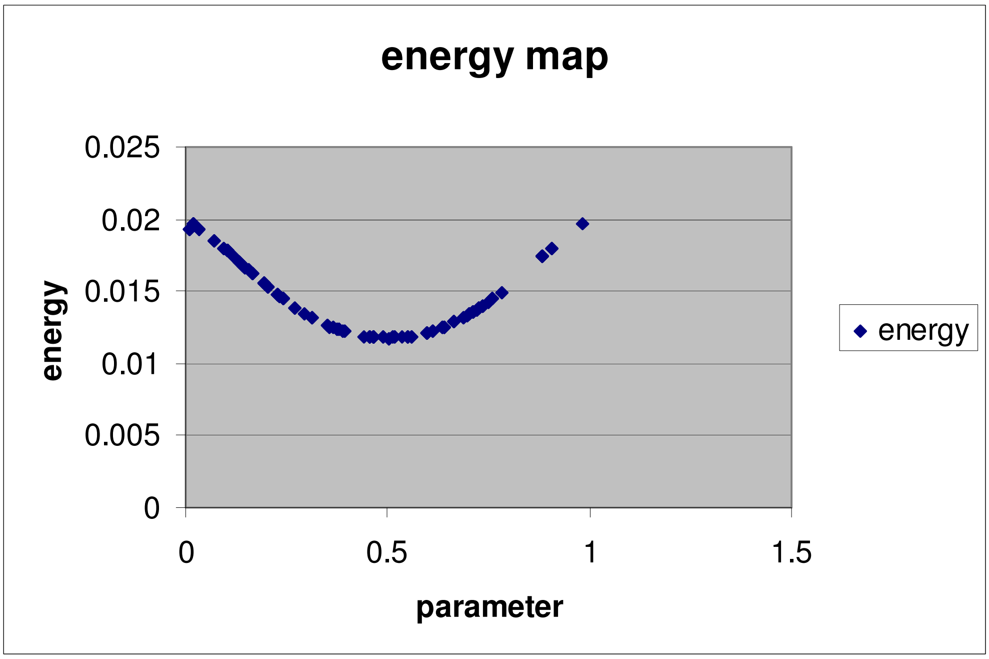


Figure 60: Evaluation function mapping Example 1

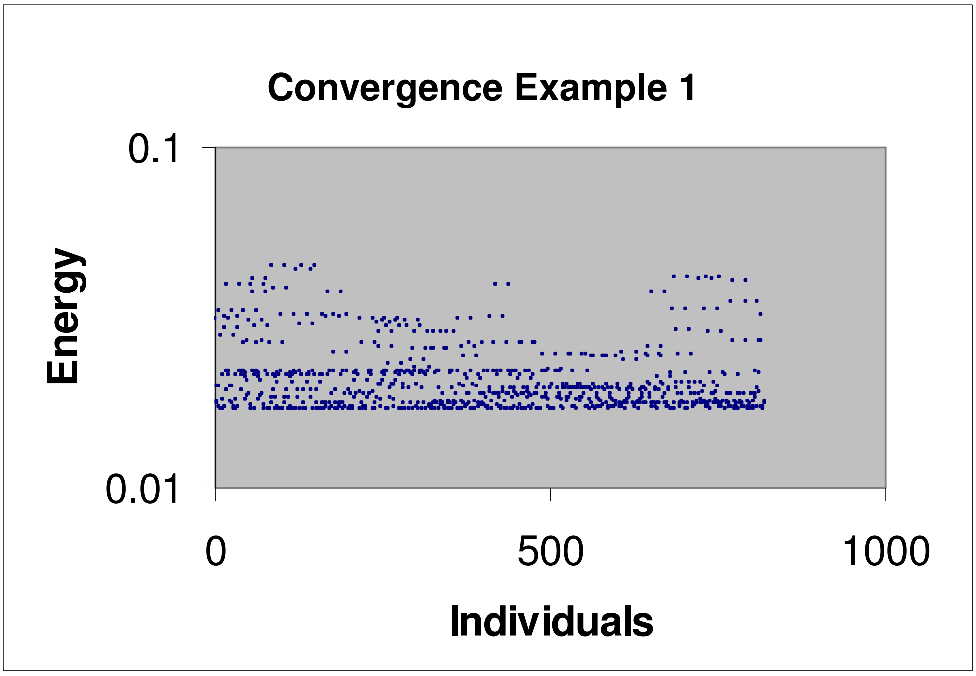


Figure 61: Energy GA converge Example 1

Example 2

This curve is obtained from five points placed in symmetrical square shape. The curvature compound fin in Figure 1 shows a concentration of curvature around the sharp corner with a swing in normal orientation on either side of the corner.

```
        const position pt1  (0, 0, 0);
        const position pt2  (50,0, 0);
        const position pt3  (100, 0, 0);
        const position pt4  (100, 50,0);
        const position pt5  (100,100 , 0);
```

The points are equally spaced along the x and y axis and the chord length parameterisation for this curve is as follows.

```
        knot vector = {0,0.25,0.5,0.75,1}
```

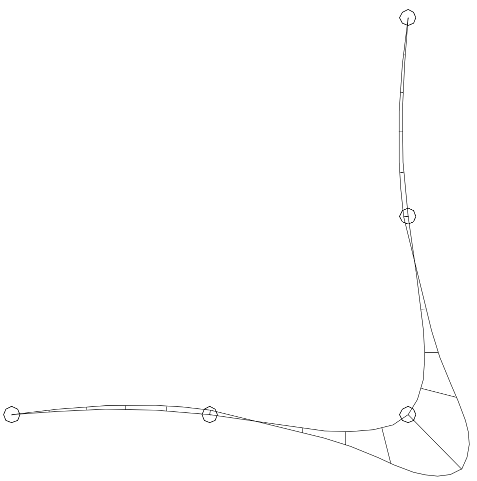


Figure 62: Original Example 2

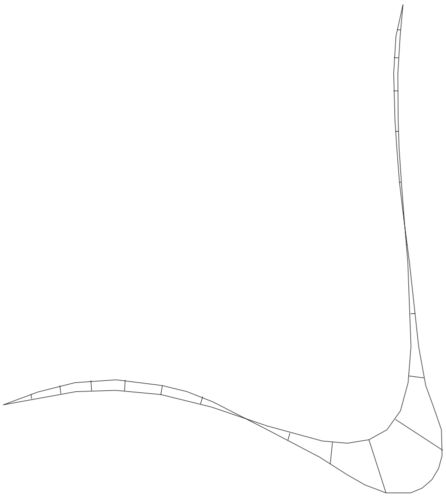


Figure 63: Optimised curve Example 2

Figure 63 shows the best solution found by application on the GA run. It would appear that the optimised curve is not any better than the original curve, it displays a more pronounced belly shape before the corner and a tighter line just after the corner.

```
        energy = 0.013016
        knot vector = 0,0.286111,0.50371,0.713889,1
```

The energy values in Figure 64 are grouped in a region of 0.9, and some individuals are unexpectedly better than the majority. Like in the previous example, the GA does no show any sign of converging.

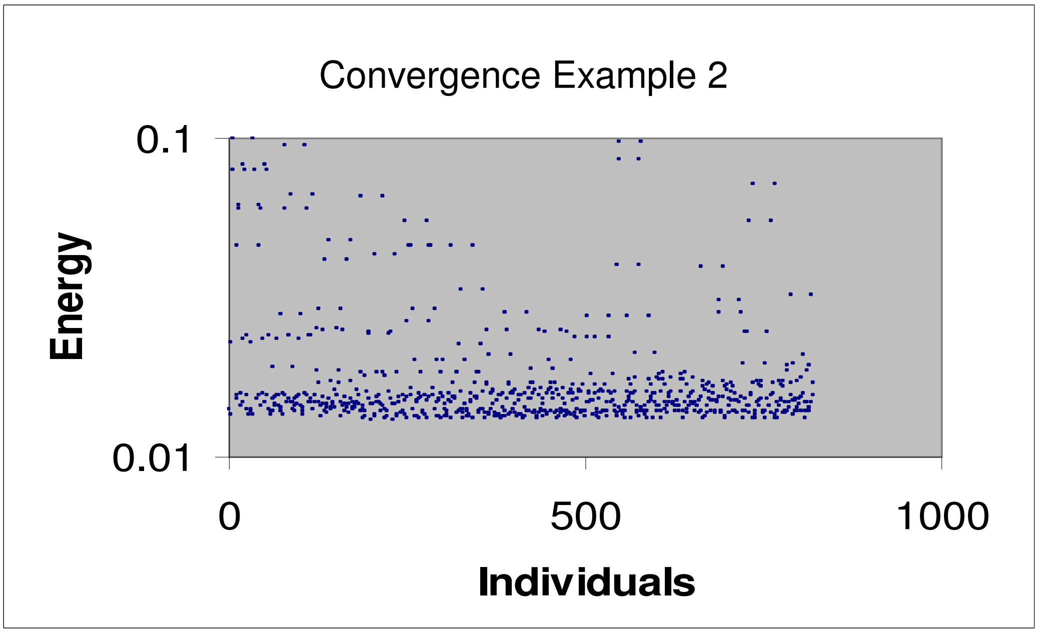


Figure 64: Energy plot Example 2

The best five solutions are these:

```
energy = 0.0141671
knot vector = 0,0.269206,0.555645,0.781508,1
energy = 0.0135851
knot vector = 0,0.246667,0.511129,0.702619,1
energy = 0.0135633
knot vector = 0,0.286111,0.474032,0.674444,1
energy = 0.013518
knot vector = 0,0.263571,0.466613,0.663175,1
energy = 0.0131302
knot vector = 0,0.252302,0.466613,0.696984,1
energy = 0.0130643
knot vector = 0,0.297381,0.50371,0.730794,1
```

It is interesting to notice that a significant change of the parameter values results in small variation of the curve's energy. And as we know from paragraph 7.2, parameter distribution has great effect a curve's overall shape. It could be that the system has too much freedom and is undetermined. Another experiment with added constraints is needed to validate the conclusions on this application.

Example 3

This curve is similar to the previous example but it is constrained to interpolate one more point around the corner. The extra constraint should reduce the solution search space, thus helping the GA to converge but increasing the number of variables.

```
const position pt1  (0, 0, 0);
const position pt2  (10,0, 0);
const position pt3  (29, 0, 0);
const position pt4  (31, 2,0);
const position pt5  (31, 31, 0);
const position pt6  (31, 41, 0);
```

The initial chord length parameterisation shown below produces the curve in Figure 65. The shape is wiggly symmetrically from the doubly constrained corner.

```
knot vector = {0,0.14,0.4,0.44,0.85,1}
```

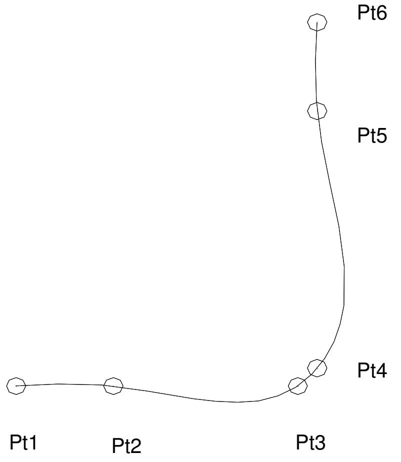


Figure 65: Original Example 3

The GA is set to run 50 individuals for 50 generations. Two of the top five solutions are studied. The best individual the GA has produced is the knot vector below.

```
energy = 0.0389927
knot vector = {0,0.125303,0.419407,0.492229,0.777902,1}
```

The curve Figure 66 is created from these values shows a reduction in curvature around both ends while making a sharp turn around the corner. The curve exhibits less of a wiggly shape than the original cord length parameterised.

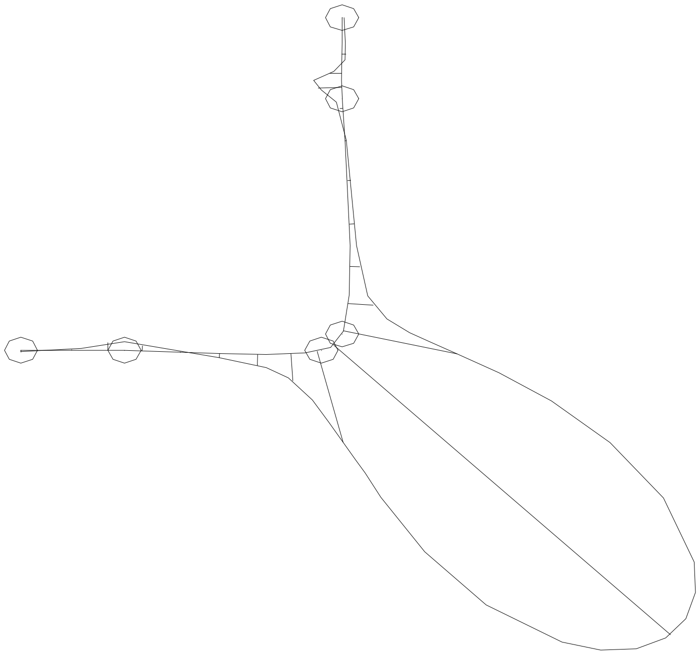


Figure 66: Optimised curve Example 3 run 1

Another GA run retuned a different knot vector for quite a similar energy values than the first run:

```
        energy = 0.0382332
        knot vector = {0,0.158246,0.392588,0.439407,0.855499,1}
```

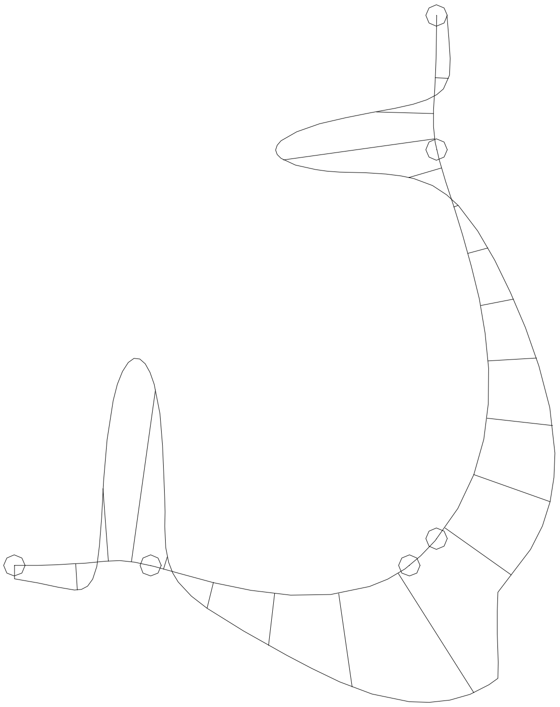


Figure 67: Optimised parameter curve GA run #2

The shape obtained from these values is more rounded than that from the first run and yet the energy is quite similar over the curve. Clearly, the optimisation shows unpredictable results that can come from the ineffective energy measure. Similar results were found by *Mussa* [76] who used an energy fairing algorithm powered by a GA, only in his algorithm, the curve's point coordinates were variables.

7.4.3 Discussion

it was observed in the curve construction process, that parameter distribution along the curve has a strong effect on the result of the construction. Chord length parameterisation is suitable to most applications, but for industrial purposes more options on the curve parameterisation must be available.

This application has made an attempt to optimise parameter distribution through a measure of the total curve energy. The results of experiments have shown that the algorithm fails to produce improved shape. It is questionable whether the GA cannot deal with the problem or the reasoning of this curve optimisation algorithm is flawed. This application is particular in the way that variables are dependent of each other: one variable change influences the balance of parameter distribution of the curve, and this is pushing the GA to its limit. Furthermore there is no guarantee that the optimisation process is sound: Example 3 has uncovered that individuals with energy close enough to be considered equal can produce shapes radically different. There is a curve with a minimum energy that satisfies a set of constraints but a parameter distribution algorithm driven by a genetic algorithm has not found it.

Still, as shown in paragraph 7.2, the interactive manual optimisation gives control on the shape modification and produces consistent results. It could be envisaged to convey the curve's total energy to the user as he modifies the knot values on the screen. This would give the user some information on the achievement of the optimisation. Because the total energy measure does not reflect the curvature distribution, it is necessary to look at the curvature compound fin as well.

7.5 Interpolated Splines

7.5.1 C_2 Cubic Spline

Because of the equations binding the number of knots and control points in a spline curve, its construction from a given set of knots and associated knot vector leads to an underdetermined system of linear equations, *Farin* [45]:

number of knots = number points + curve degree – 1

In the case of a cubic, the above equation results in two undetermined unknown. In order to add more equations to the system, end conditions are specified. Intuitively, it is apparent that something is missing: with cubics, the slope of the curve at each joint is determined from the continuity constraints between the two spans meeting at the knot. But what happens at the end points, where there is no continuity. The answer to that problem is to define vectors at the end points that will emulate continuity.

There are several possible ways of defining end conditions. The clamped end condition corresponds to the prescription of two derivatives that are taken from first two control points. Another end condition called natural specifies that the second order derivative is zero at the end points. Yet another technique called Bessel fits a quadratic through the first three data points and uses the first derivative of the quadratic as an end vector for the cubic.

The choice of end condition in cubic spline interpolation only affects the curve near to the end points. They do not matter so much in the interior.

Interpolated splines have the property that changing the position of one of the interpolated points will result in changing the overall shape of the curve, Figure 68, although the amplitude of this change reduces fairly quickly with distance from the change point. This can be a problem when trying to optimise a curve: Some improvements in one place might destroy it in others. We shall see later how a heuristic optimisation algorithm copes with that. Other curve construction methods especially those with Bézier control polygons allow local changes. These techniques are now reviewed in the following paragraphs

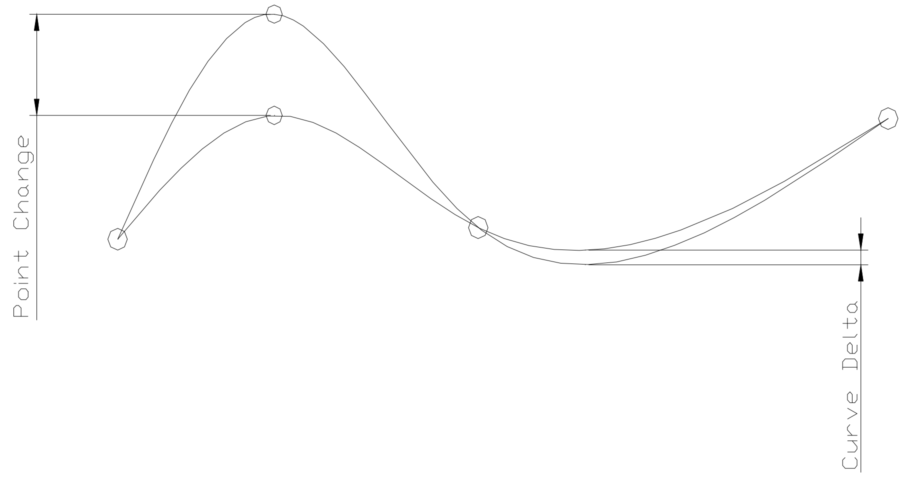


Figure 68: Global shape change with spline curves

7.5.2 Hermite Interpolation

In paragraph 5.3.2, it was established that Hermite polynomials hold differential constraints explicitly. It is therefore possible to construct piecewise curves with guaranteed continuity between curve segments, as shown in Figure 69.

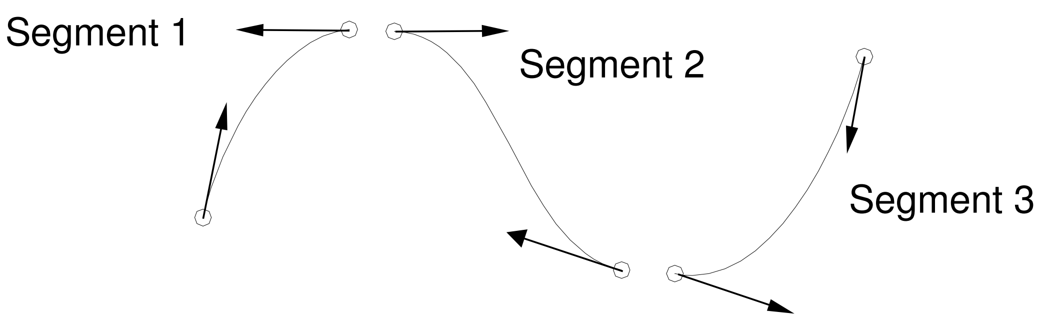


Figure 69: Piecewise Hermite interpolation with tangent continuity

Unlike the B-spline model, Hermite interpolation allows modifying the internal constraints; hence offers more control on the curve's shapes. An example of this is illustrated in Figure 70 and Figure 71, where the tangents at the knots are modified both in magnitude and direction.

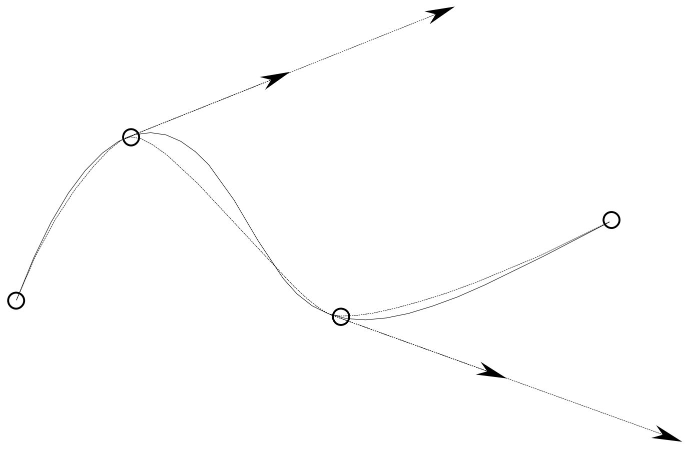


Figure 70: Hermite interpolant with varying tangent magnitudes

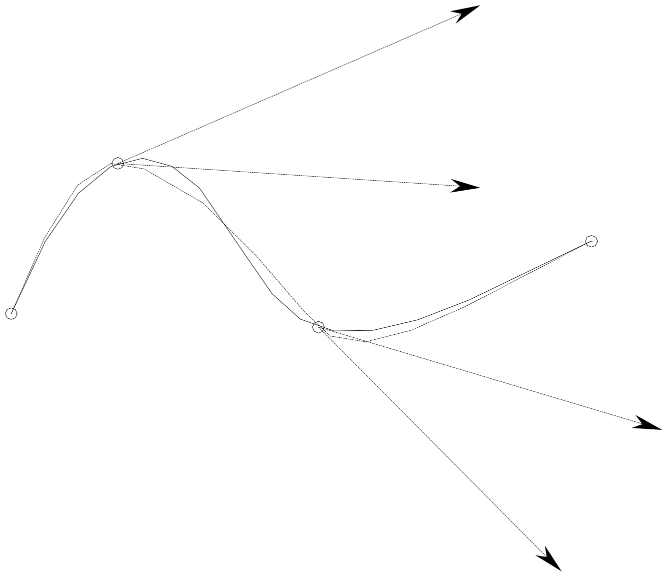


Figure 71: Hermite interpolant with varying tangent directions

In practice a cubic Hermite segment $[a,b]$requires the positions coordinates, the first derivative and the second derivative vectors at *a* and *b*. Because of the arbitrary parameterisation, the derivative vectors are not unit vectors and further are not representative of the tangent and normal respectively. Therefore some jugglery is needed to represent the constraints as tangents and normals graphically, so that they can be modified from the graphic interface.

- Input the position constraints
- Interpolate a cubic spline through these
- Get the first and second derivatives $P'(a), P'(b), P''(a), P''(b)$
- Evaluate the magnitude of the derivatives $\|P'(a)\|, \|P'(b)\|, \|P''(a)\|, \|P''(b)\|$
- Reparameterise and normalise the derivatives to get the tangents *T(a), T(b),* and the normals *N(a), N(b)*
- Scale the tangents and normals $T(a)\|P'(a)\|, T(b)\|P'(b)\|, N(a)\|P''(a)\|, N(b)\|P''(b)\|$
- Display as line segments the constraints which can be modified both in direction with a rotation and magnitude with a scalar

- Apply the rotation and scale transformations to the derivatives *P'(a), P'(b), P''(a), P''(b)* and construct a polynomial that satisfies the prescribed constraints.

7.6 Conclusions

This chapter has concentrated on the optimisation of construction constraints such as parameterisation and differential properties. This has led the research to propose algorithms that can give extra control on the curves shape.

The first idea put forward is providing handles on knot values through the graphic interface. It has the advantage of providing an easy straight forward handles on the parameterisation of curves. The major difficulty for the user it to have an idea of result before changing some values. The total energy measure is brought in the application to give an indication on the outcome of the optimisation: a smooth curve has less energy than a curve with sharp corners. An optimum parameter distribution can be easily found with curves that embrace one or two variables, but with more variables the problem can get too complex to be dealt with manually. Some problems more specific to the knot interpolation functionality have been uncovered. Irregularities in the energy map were found with symmetrical curves parameterised in the way that the sum of the knots internal knots equals the value of the last knot. Further investigations on the matter are needed to establish the root of this problem.

To answer the problem of complexity, an automated parameter optimisation driven by a GA is proposed uses the total energy formulation as evaluation criterion. This method however gave poor results and it has proven difficult to establish the exact cause of the problem. The examples have opened questions on the GA’s ability to deal with problem features such as interactive variables as well as on the validity of the total energy formulation.

Other internal constraints are put forward to the user through the graphic interface. These are the curves differential properties at the knots points, which are used as constraints in Hermite interpolation. There is nothing new here, similar methods are present in CAD systems like SURFACER [108].

The work undertaken in this chapter leads to the conclusion that the NURBS formulation is most useful for interpolating data point coordinates together with other internal constraints such as parameter distribution and derivative constraints. However tweaking the NURBS construction constraints has limited scope: The construction constraints identified as influential on curves' geometry are applicable at discrete points. The curve optimisation applications laid in this chapter have gambled on the fact that handles on curves' constraints would achieve better control on their shape. It was proven that more advance shape control could be achieved, but they are not necessarily helping the fact that NURBS are difficult to control. This is not all for curve optimisation, the alternative to NURBS representation is Function Representation (F-Rep), introduced in paragraph 4.3, supports global constraints. In the next Chapter, new curve optimisation applications are created using F-Rep for representing curves and constraints.

Chapter 8 Intrinsic Curve Optimisation

Despite efforts to broaden NURBS geometric handles in Chapter 7, NURBS is thought to be limited because constraints are applied locally and interpolated. The curve optimisation applications showed that the NURBS representation was not satisfactory to achieve optimisation and this chapter proposes alternative curve optimisation schemes using continuous vector functions for the representation of the differential properties. The basic idea carried out in this chapter is to use function representation of differential constraints in curve optimisation processes.

8.1 Curve Projection

Visualising three-dimensional geometry on a conventional 2D monitor screens can be difficult, and projecting three-dimensional objects onto two-dimensional planes can help understanding better the design.

Projection

Given a parametric space curve *P(u)={X(u), Y(u), Z(u)}*, Three 2D curves are created by setting in turn one of the components to null as follows.

$$X\_Y\ projection = \begin{pmatrix} X(u) \\ Y(u) \\ 0 \end{pmatrix}$$

$$Y\_Z\ projection = \begin{pmatrix} 0 \\ Y(u) \\ Z(u) \end{pmatrix}$$

$$Z\_X\ projection = \begin{pmatrix} X(u) \\ 0 \\ Z(u) \end{pmatrix}$$

Boxing

A bounding box parallel to the X,Y,Z axis that totally encloses the curve is created, This box is used to determine the faces on which the curve is projected.

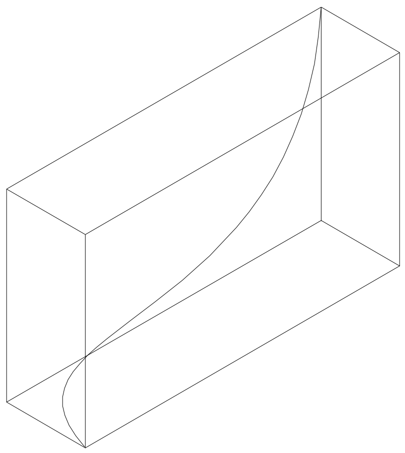


Figure 72: Curve Bounding Box

Implementation

Each component of the curve vector are extracted and copied in separate LAWs using the trem_law() ACIS LAW class member function. The arguments are the curve's LAW position_vector followed by a number that indicates which vector component (term) is returned; 1 is X, 2 is Y, 3 is Z.

```
        law* x_comp = new term_law(position_vector, 1);
        law* y_comp = new term_law(position_vector, 2);
        law* z_comp = new term_law(position_vector, 3);
```

New arrays of LAWs are created from two extracted vector components and a third one set to zero. These arrays are used to create new vectors that represent the projected curves.

```
X_Y_projection
law* x_y_array[3] = {x_comp, y_comp, zero};
law* x_y_proj = new vector_law (x_y_array,3);
Y_Z_projection
law* y_z_array[3] = {zero, y_comp, z_comp};
law* y_z_proj = new vector_law (y_z_array,3);
Z_X_projection
law* z_x_array[3] = {x_comp, zero, z_comp};
law* z_x_proj = new vector_law (z_x_array,3);
```

Figure 73 illustrates the results of the projection algorithm described above.

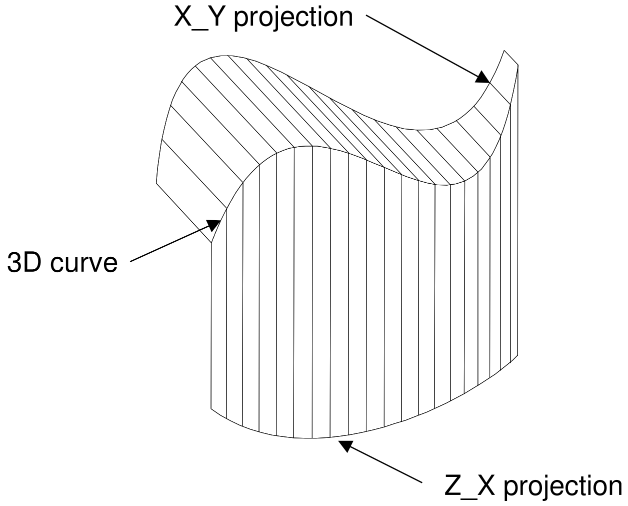


Figure 73: Projection in the X_Y and Z_X planes

Reconstruction

With the help of a geometric editor, one can modify a projection curve, i.e. two vector components, and subsequently reconstruct a curve from these and the third unchanged component. Only one projected curve should be modified at the time to ensure that one vector component is not modified in different projections. Also the projected curve should remain planar throughout modification, otherwise the third vector component would also be changed. This would cause a problem for reconstruction because the third component modification would not be taken into account. Figure 74 shows an example of direct modification of a curve's projection in the Z_X plane. Here the Y vector component is kept unchanged, while the X and Z components are modified.

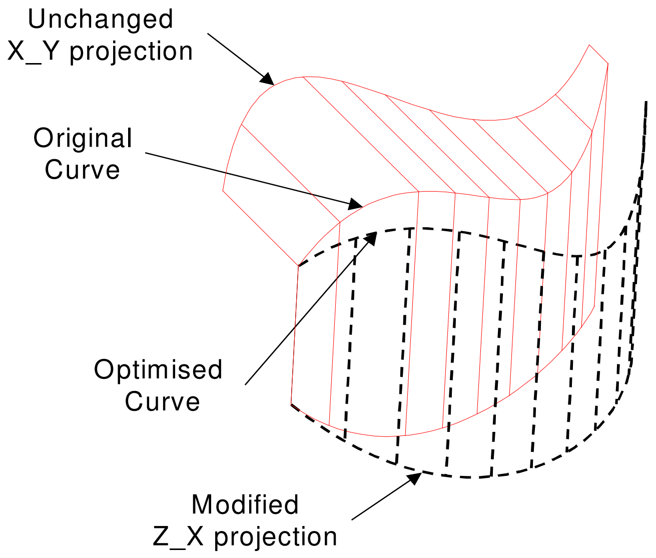


Figure 74: Direct modification of projections and reconstruction

Discussion

The projection technique is useful to visualise on three planes the path of a 3D curve. It enables to see how much curvature and how much torsion there is. Also it is possible to change one of the projections and recreate a 3D curve. However this technique has obvious limitations with helical curves, the projection of one of these would not be helpful

8.2 Prescribed Curvature Profile

It is common engineering practice to visualise the curvature plot to detect imperfections or judge on the quality of curves. Designers have to modify the curve manually and check the curve again, this loop is repeated until a satisfactory result is achieved. This paragraph aims to produce algorithms that can reconstruct a curve back from its intrinsic definition. The purpose of developing this modelling tool is to make reverse engineered curves from curvature and torsion profiles, hence cutting down the development time of curved models.

The theorem of existence and uniqueness introduced in Chapter 6 guarantee that given a non-vanishing curvature and torsion function, a solution curve exists and it is unique, up to a motion in space. Also null torsion curves are planar, and their shape is defined solely by the curvature profile, *Aminov* [3], *Eisenhart* [41], *Weatherburn* [120]. Two different methods are investigated and implemented in programs. The first application's strategy is to generate a curve whose curvature profile matches the objective profile. For this a great number of curves are generated and tested against the objective profile. The optimum solution search is powered by a genetic algorithm plugged on the data points coordinates. The second approach is making use of function representation for modelling the curve's intrinsic properties. New prescribed properties become constraints and algebraic algorithms can return to the curve definition.

8.2.1 Genetic Algorithm Driven

A GA driven optimisation algorithm for reconstructing curves from a prescribed curvature profile as constraint is proposed here. A complete listing of the programming details is included in Appendix 5 followed by tests performed on the optimisation process in Appendix 6. The objective of the optimisation is a target curvature profile prescribed by the user. A number of solutions named individuals are generated by the GA and are compared against the objective. The outcome of the algorithm is the best match to the objective.

Optimisation Process

Figure 75 describes in a flow chart the curve optimisation process with a GA. An initial curve is created by some means, typically by interpolating data points with a given tolerance. The curvature is evaluated at close parameter values and a general spline is interpolated through those points. A so-called curvature profile is then created and subsequently modified allowing capture of designer's intent. The modified curvature profile becomes the target. The remaining tasks are finding a curve that comes close to the target profile. This is achieved by modifying iteratively the data point coordinates. The iteration is loop is a heuristic of the GA type. The optimisation process reveals that the algorithm is an automated trial and error approach to solve the problem.

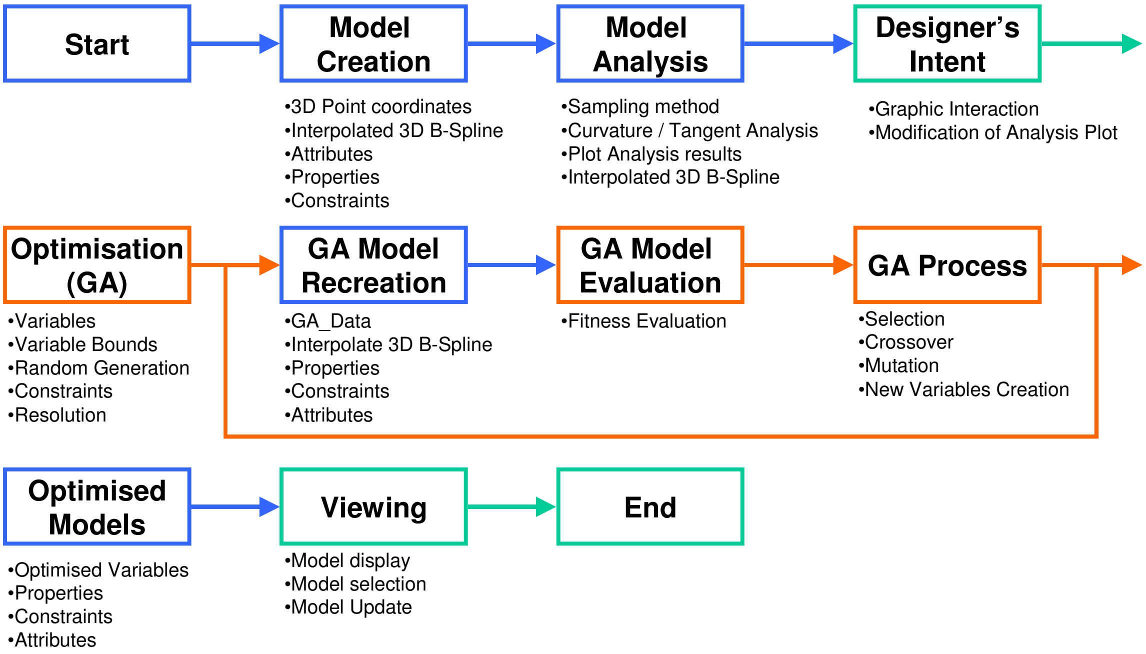


Figure 75: Curve Optimisation process with GA

Constraints

Because of the theorems of existence and uniqueness, we know that one curvature profile can have a multitude of curves, all the same but with a different orientation in space. A motion composed of a translation and a rotation can give the curve its original orientation *Eisenhart* [41], *Gibson* [53]. Because the optimisation process works on a parametric model, some constraints are introduced to reduce down its freedom of movement. The constraints are applied to the curve construction data in order to orientate the model in space.

- One fixed point stops 3 translations
- One vector stops 3 rotations

As a result of this, one point with the tangent direction at this point need to remain constant to ensure the orientation of the model. The starting point and its tangent are chosen to remain constant through out the process. Because tangent direction and curvature are related, keeping the start tangent constant the curvature profile start value must remain constant as well.

Test Curve

A test curve is used through out the development phase of the algorithm. It is an interpolated cubic planar curve obtained from the following data points. The end tangent directions are kept null (natural ends with zero curvature). This curve is s-bent, the curvature profile is goes to zero before making a sharp turn with high curvature. Also to simplifies maters, the curve is 2D, the components in the Z axis are kept constant null.

```
        position pt1  (0, 0, 0);
        position pt2  (16,20, 0);
        position pt3  (20, 30, 0);
        position pt4  (40, 0, 0);
```

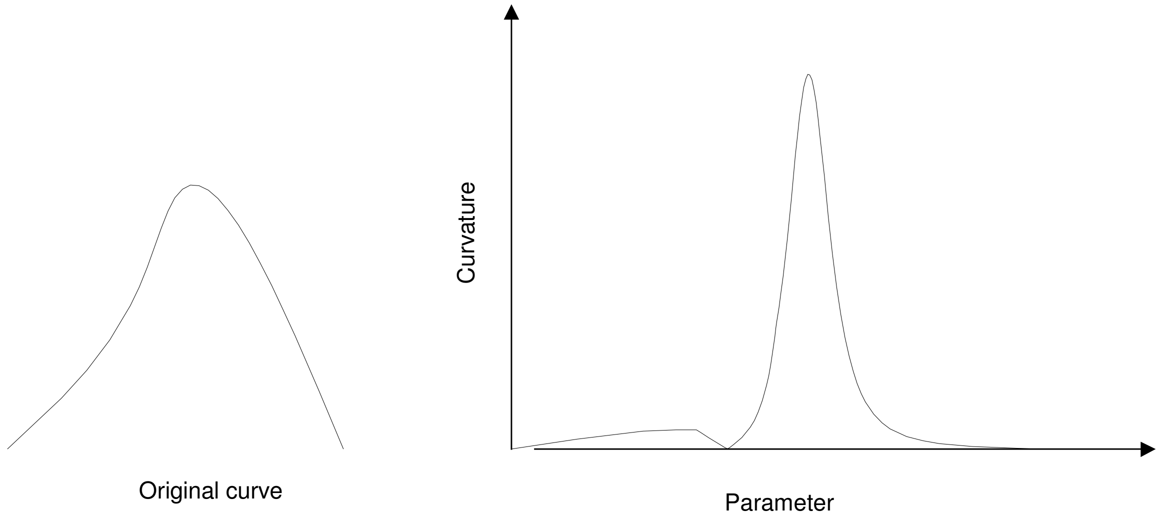


Figure 76: Test curve with curvature profile

Variable bounds

GA variables are bounded by an upper limit and a lower limit. This restricts the GA to generate points within these bounds. The GA variables minimum and maximum values are set in the GA process for each component of the points coordinates. A wider variable bound results in a bigger search map and more work for the GA to converge. A null variable bond constrains the variable to remain constant. The listing below is the bounding of four data points.

```
double upper_limit [ ] = { 0, 0, 0,             // pt1, fixed point
                           5, 5, 0,             // pt2, Z fixed
                           5, 5, 0,             // pt3, Z fixed
                           5, 5, 0};            // pt4, Z fixed

double lower_limit [ ] = { 0, 0, 0,             // pt1, fixed point
                           5, 5, 0,             // pt2, Z fixed
                           5, 5, 0,             // pt3, Z fixed
                           5, 5, 0};            // pt4, Z fixed
```

Given an optimisation objective, it is not guaranteed that the optimum solution lies within the variable bounds. The obvious answer to this problem is to set sufficiently large variable bounds and hope that it encloses the solution. Still there is little way of checking that the algorithm has found the best solution or it has been constrained by the variable bounds outside the optimum. In the optimisation process paragraph 7.4.1, it is suggested that the GA is run more than once each new GA run takes the best results of the previous run as initial variable values with the same variable bounds.

An experiment is set up to illustrate the problem of variables bounds. The algorithm is set to find a curve that is known in advance. First the GA is initialised with values and bounds that fall outside the solution. Second the GA is reinitialised with the values recorded from the first run.

The target solution taken from the test curve is composed of four construction points:

```
Target solution
position pt1  (0, 0, 0);
position pt2  (16,20, 0);
position pt3  (20, 30, 0);
position pt4  (40, 0, 0);
```

The initial GA values together with their lower and upper bounds are chosen to fall outside the target solution.

```
Initial values
position pt1_ga  (0, 0, 0);
position pt2_ga  (10,28, 0);
position pt3_ga  (23, 22, 0);
position pt4_ga  (40, 0, 0);
```

The intermediate solution, returned after the first run is shown bellow, clearly the variable bounds have prevented the GA from converging any closer to the solution. However when applying the variable bounds defined above, it is within range of the solution.

```
Intermediate solution
average_distance = 31.9274
pt1_ga    (0, 0, 0.)
pt2_ga    (14.8434, 23.1832, 0.
pt3_ga    (18.4668, 26.8695, 0)
pt4_ga    (40.3795, 0, 0)
```

Now GA is re-initialised with the intermediate solution and the best solution returned after the second run is 10% off the known solution and the fitness values, average distance is reduced ten times. The GA should be able, in theory, to find the solution because it lies within the variable bounds.

```
Final solution
average_distance = 3.17459
pt1_sol (0, 0, 0)                        // fixed poit
pt2_sol (14.7779, 18.3042, 0.1208)        (7.64%, 8.48%, fixed)
pt3_sol (19.9686, 30.461, 0.17876)        (0.16%, 1.54%, fixed)
pt4_sol (39.9934, 0.2498, 0)              (0.02%, -, fixed)
```

Reinitialising the GA with an intermediate solution has proven adequate to come out of the variable bound restrictions. Still it, in a real-life scenario where there is no prior knowledge of the solution, it is not possible to determine how many times the GA should be reinitialised so that its bounded variables enclose the solution. Also in reverse engineering applications, a tolerance of half a millimetre is usually required on data points obtained from clay model digitising, if the user inputs a curvature profile that creates a curve outside the variable bounds, the input is not valid. There should be ways to constrain the curvature to ensure the solution curve fulfils the engineering tolerances. Under the current definition of the problem , there is no way to establish a relationship between the objective and the solution. And if there was such relationship, it would be used to find the solution directly

Fitness and Evaluation Function

the evaluation function is the sum of the distances computed between the curvature profile of the solution curve $f(u)$ and the target curvature profile $F(u)$. The distance between the two curves is computed by dense sampling and averaging over the number of evaluation points, Equation 31.

$$Evaluation = \frac{\sum_{u=0}^{n} f(u) - F(u)}{n}$$

Equation 31: Evaluation function

The GA used in this application works to maximise the fitness: an individual with a high fitness value is close to the optimum. On the contrary, the objective of this optimisation algorithm is to minimise the average distance between two curves. For this reason, the fitness function is written as the invert of the evaluation function, Equation 32. Assuming that the average distance can be infinite, the fitness exists in the interval [0-1], 0 score is the worst and one score is the best.

$$Fitness = \frac{1}{1 + Evaluation}$$

Equation 32: Fitness function

In order to establish the design variable sensitivity, a test case was set up:

> The optimum solution is known in advance, and these initials values are the optimum.
>
> The variables are modified incrementally and the fitness score is recorded for each of these. Only two variables are varied at the time.

In Figure 77, the two variables in the bottom axes are data points and the fitness function is in the vertical axis. The fitness map shows the characteristics of a uni-modal optimisation problem. The test function is equal to one at it's nominal values. As the variables move away from these values, the fitness values decrease. However it was found that the fitness function map was not steep enough: Great variation in variable values lead to small change in fitness value. And the GA convergence graph Figure 78, is not showing signs of convergence.

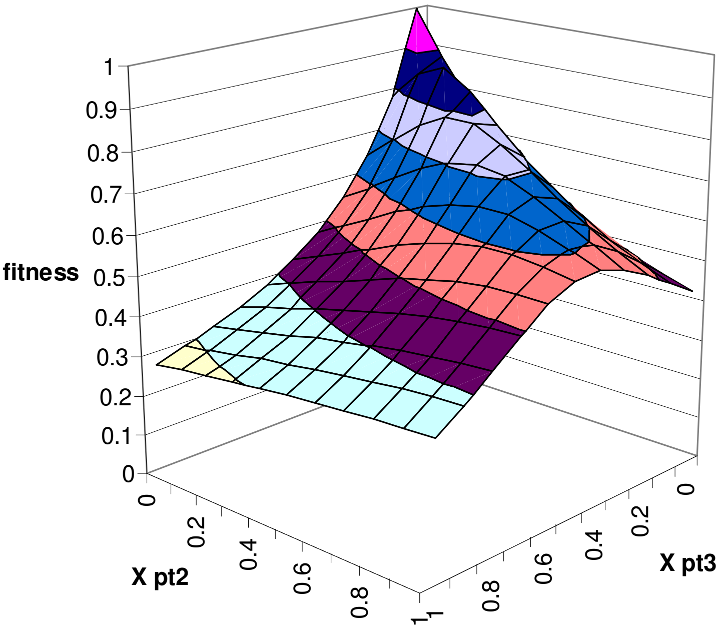


Figure 77: Fitness function test

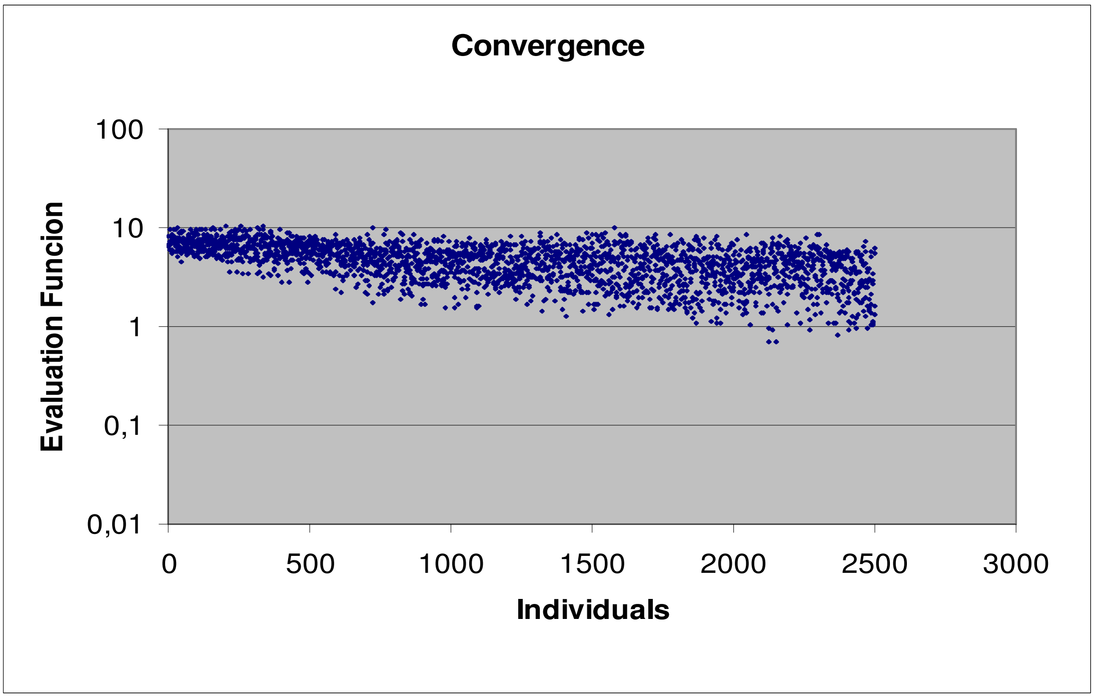


Figure 78: Convergence

Power Boost

Since the solution range varies from zero to one, and sing the results from 2.4, the fitness function map can be made steeper by applying a power function.

$$\text{Fitness} = \left(\frac{1}{1 + \text{average distance}} \right)^{n}$$

Equation 33: Fitness Boosting

A series of experiments, presented in Appendix 6.2, have studied the influence of fitness boosting on the GA performance. It was found that a the best results are produced for a power of three applied to the fitness.

Figure 77 shows the effects of such boosting with a power three. The map shows a much steeper slope around the optimum. This will help giving the best individuals a much greater chance of selection and reproduction.

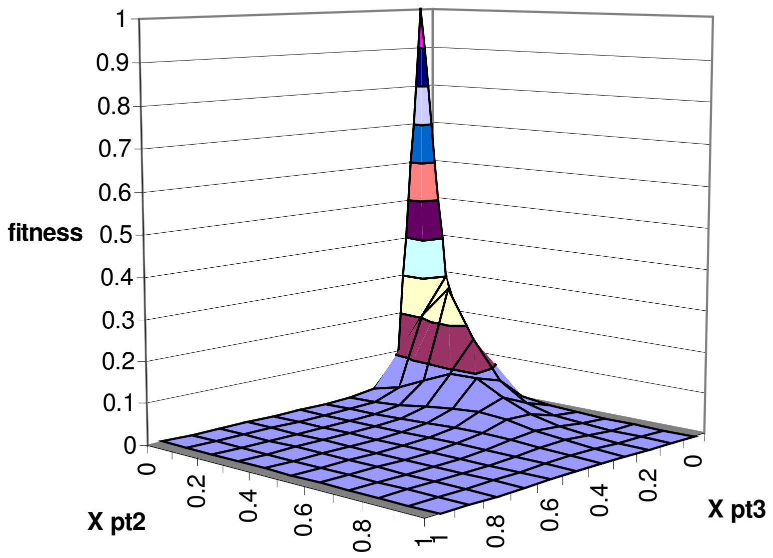


Figure 79: Fitness function boosting with power three

The effect of the power boost on the GA convergence, illustrated in Figure 80, is considerable. The Pareto front is more pronounced and drops rapidly. Here the best solutions have an average distance in the region of 0.1 compared with 1 previously. The power boost has managed to reduce by a factor of 10 the average distance between the two curvature profiles.

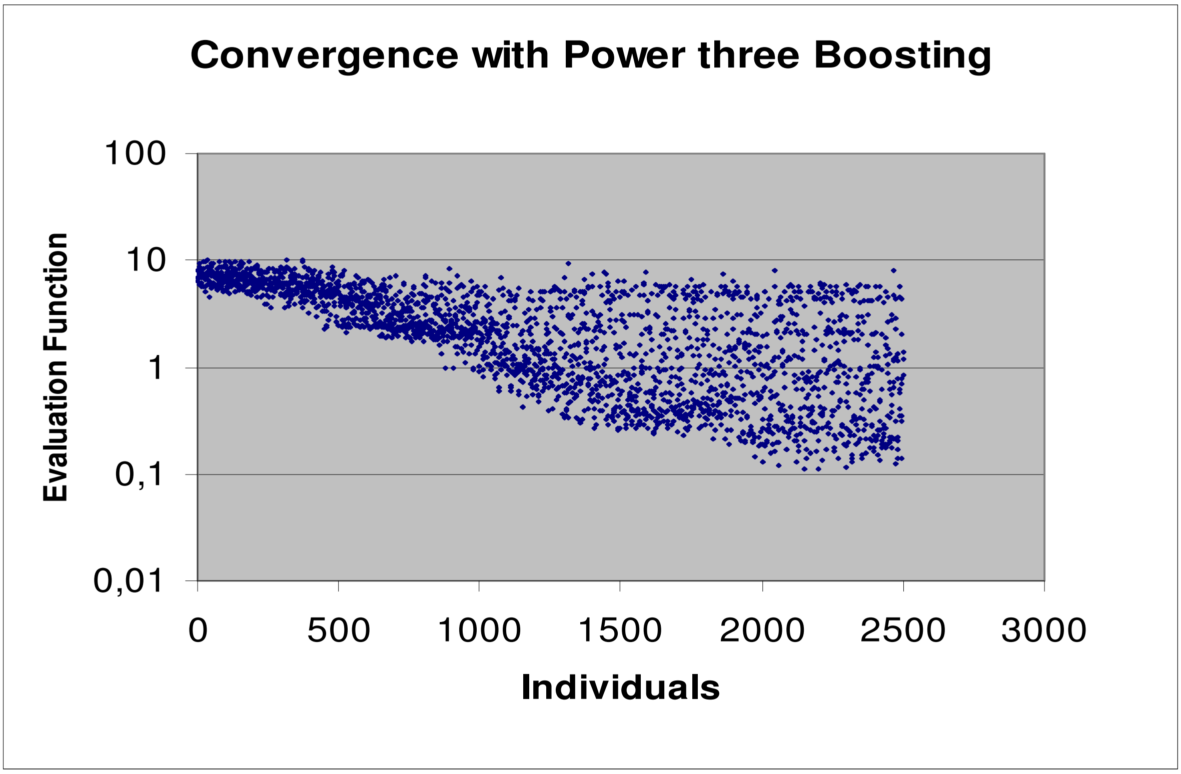


Figure 80: Convergence with boosting function

Validation

The aim of this section id to demonstrate how this algorithm is used to generate improved geometry. The feature curves introduced in paragraph 6.5.1 are used here to validate this GA driven curve optimisation algorithm. The following example curves exhibit features including planar and space curves, convex and S-bend. For each example, the GA convergence graphs are used to bench mark the GA performance. In addition, point coordinates are recoded in tables that show the modifications the algorithm has made to the curves.

Example 1

The first example is the simplest curve that can be made. It is constructed by interpolating a cubic spline through three data points located in a plane. The fitted curve is convex and has natural conditions on either ends, so the curvature is zero at these points.

```
        position pt1  (0, 0, 0);
        position pt2  (5,20, 0);
        position pt3  (40, 0, 0);
```

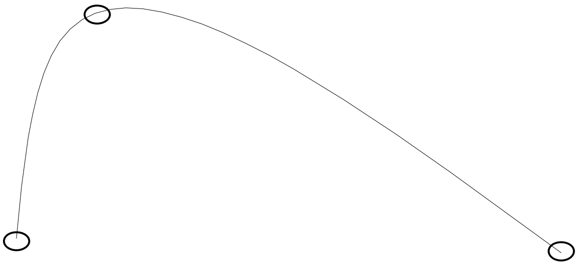


Figure 81: Original curve Example 1

The curvature profile of the original curve, Figure 82, shows a concentration of curvature in the first half of the curve. The profile is modified to shift curvature in the second half of the curve.

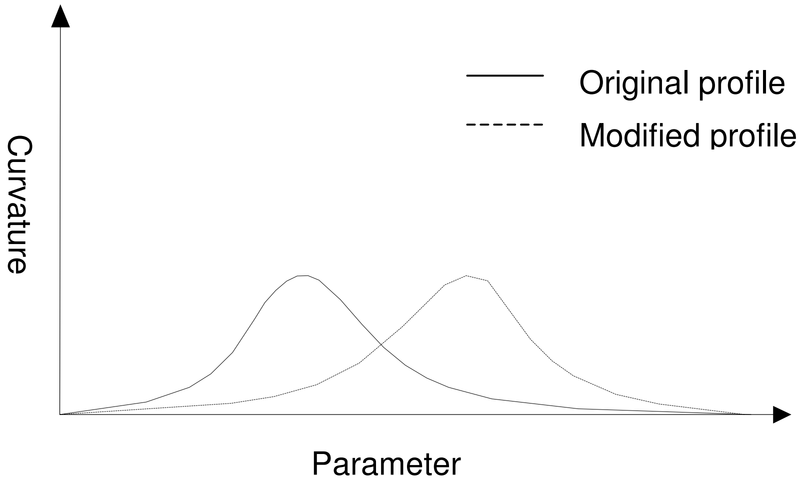


Figure 82: Modified curvature profile Example 1

The variable bounds are set to constrain the first and the last point as constant and all points 'z' coordinate axis as constant. The bounds on the internal variable point are set wide compared with the coordinate values, this is for ensuring the solution is within the variable definition range. This should not damage the GA performance since there are only four variables in total.

```
double upper_limit [9] = { 0, 0, 0,
                           50, 50, 0,
                           0, 0, 0};

double lower_limit [9] = { 0, 0, 0,
                           50, 50, 0,
                           0, 0, 0};
```

The convergence graph, Figure 83, shows a rapid conversion with a limit at one tenth of the unit.

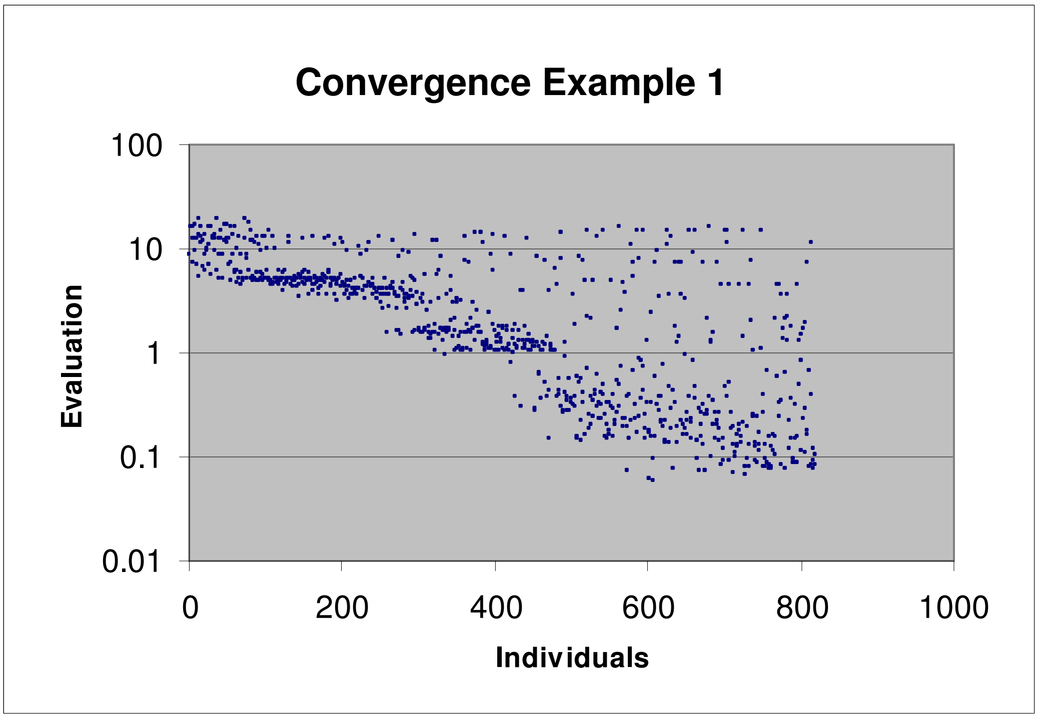


Figure 83: Convergence graph Example 1

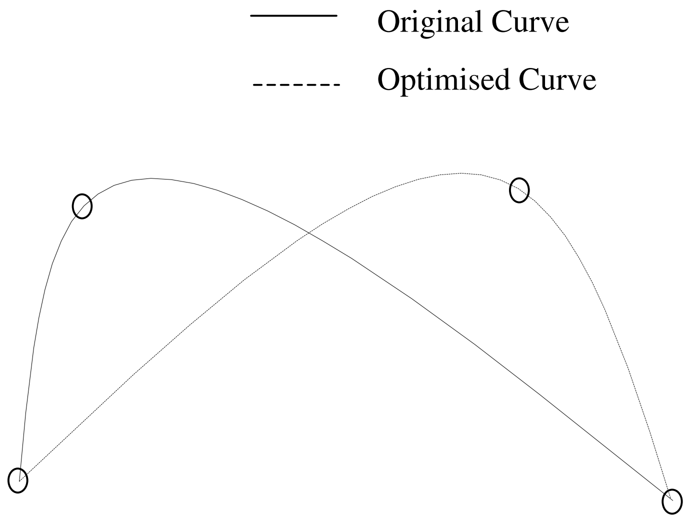


Figure 84: Optimised curve Example 1

The optimised curve drawn in Figure 84 has kept the same but it a mirror of the original. The curvature profile prescribed to shift the curvature at the end of the curve and the GA responded by moving the internal data point on the 'x' coordinate axis.

GA points

	pt1_ga	pt2_ga	pt3_ga
X	0.0000	30.4741	40.0000
Y	0.0000	19.1301	0.0000
Z	0.0000	0.0000	0.0000

Original points

	pt1	pt2	pt3
X	0.0000	5.0000	40.0000
Y	0.0000	20.0000	0.0000
Z	0.0000	0.0000	0.0000

Difference

	pt1	pt2	pt3
X	0.0000	-25.4741	0.0000
Y	0.0000	0.8699	0.0000
Z	0.0000	0.0000	0.0000

Table 4: Results Example 1

Example 2

This example is the same set up as the previous example: The start and end points are constant only this time the internal point coordinates are all variable. This means that the GA can try to have non-zero ‘z’ coordinate and thus introducing torsion in the curve. No torsion profile has been prescribed therefore the solution curve is planar and the ‘z’ component value should be zero.

```
double upper_limit [9] = { 0, 0, 0,
                           50, 50, 50,
                           0, 0, 0};

double lower_limit [9] = { 0, 0, 0,
                           50, 50, 0,
                           0, 0, 0};
```

The GA convergence is affected a great deal from this change of variables. Even though it has one more variable to deal with, the graph in shows that the GA converges to the optimum more rapidly than in the previous example

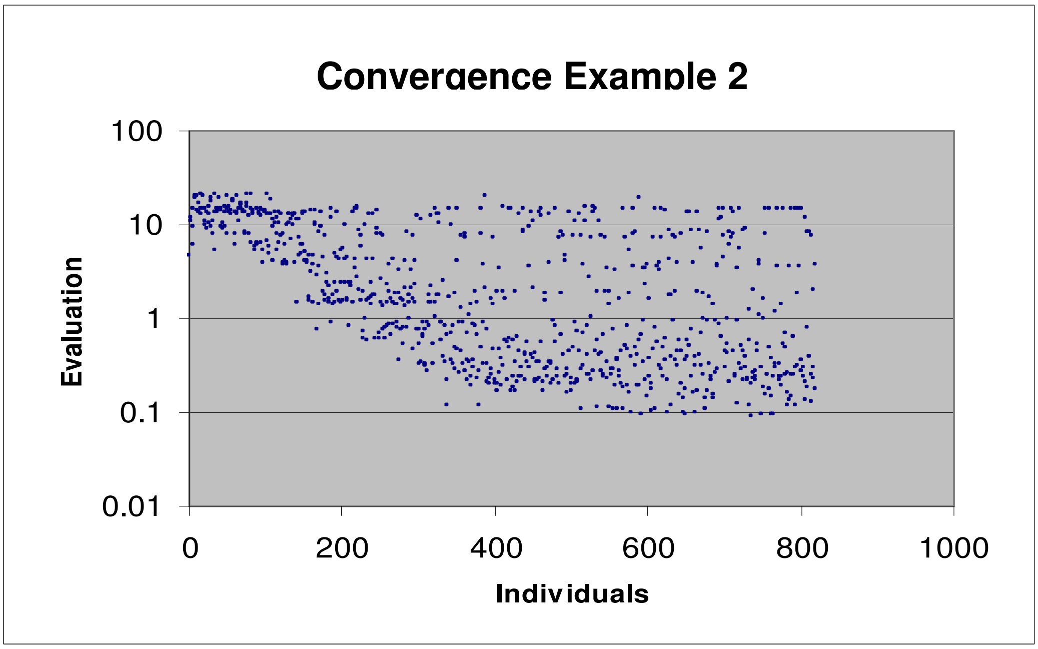
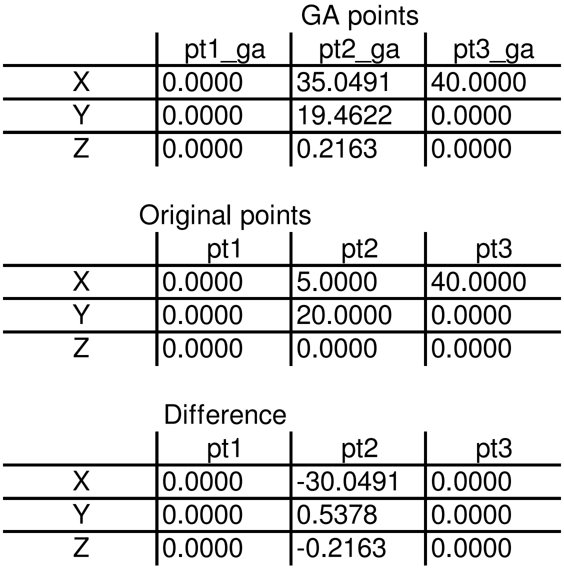


Table 5: Convergence graph Example 2



GA points

	pt1_ga	pt2_ga	pt3_ga
X	0.0000	35.0491	40.0000
Y	0.0000	19.4622	0.0000
Z	0.0000	0.2163	0.0000

Original points

	pt1	pt2	pt3
X	0.0000	5.0000	40.0000
Y	0.0000	20.0000	0.0000
Z	0.0000	0.0000	0.0000

Difference

	pt1	pt2	pt3
X	0.0000	-30.0491	0.0000
Y	0.0000	0.5378	0.0000
Z	0.0000	-0.2163	0.0000

Table 6: Results Example 2

The difference table shows that the second point ‘z’ component is nearly zero, the GA has found the correct answer with reasonable accuracy.

Example 3

This curve is a planar S-bend curve made from the four following interpolated data points, see Figure 85.

```
        position pt1  (0, 0, 0);
        position pt2  (16,20, 0);
        position pt3  (20, -30, 0);
        position pt4  (40, 0, 0);
```

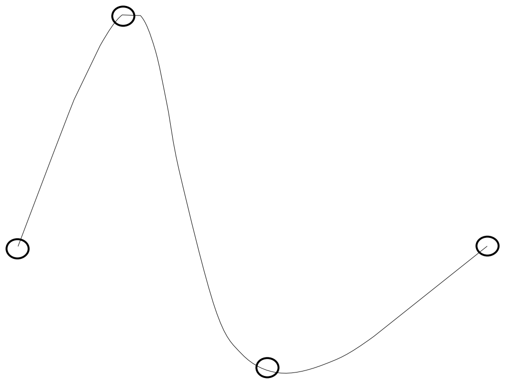


Figure 85: Original curve Example 3

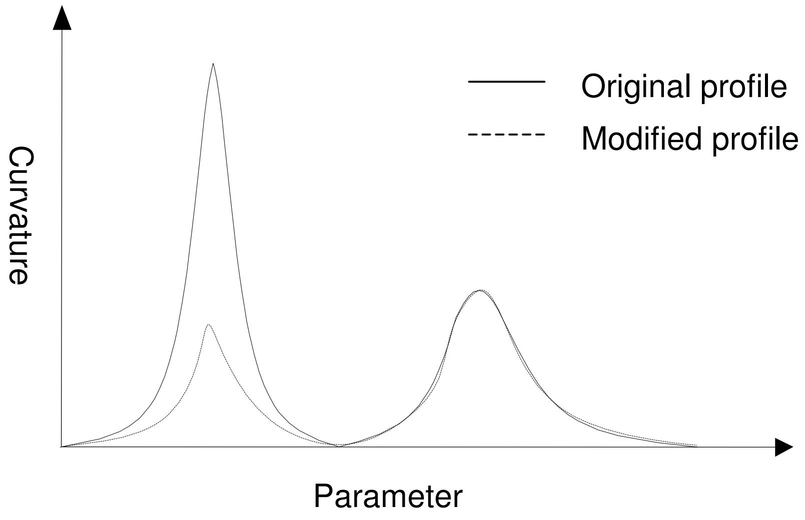


Figure 86: Modified curvature profile Example 3

The original curvature profile, in plain line Figure 86, shows two picks of curvature and one point of inflection. The profile is modified to reduce the first curvature pick, in doted line. The modifications are applied locally to the curvature profile, only the first half of the curvature profile is modified, second half is kept the same.

The variable bounds constrain the start point as a fixed point and all the points the z components to zero.

```
double upper_limit [ ] = { 0, 0, 0,
                           10, 10, 0,
                           10, 10, 0,
                           10, 10, 0};
double lower_limit [ ] = { 0, 0, 0,
                           10, 10, 0,
                           10, 10, 0,
                           10, 10, 0};
```

The convergence graph, Figure 87 indicate that the algorithm converges, with some solutions evaluated below the one line.

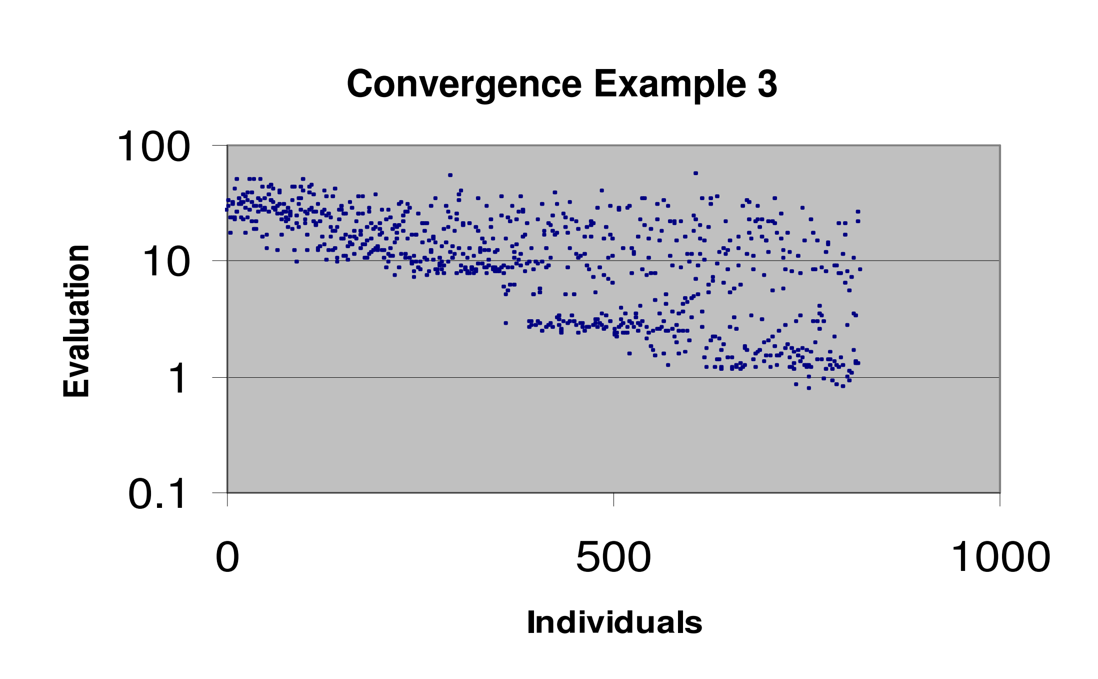


Figure 87: Convergence graph Example 3

The optimised curve, Figure 88, has a more rounded crest than the original curve, which was the desired outcome. Only the point in the affected area is modified, the optimised data points are recorded in Table 7, the difference with the original points show that the second point coordinates have been changed while the others are close of their original values.

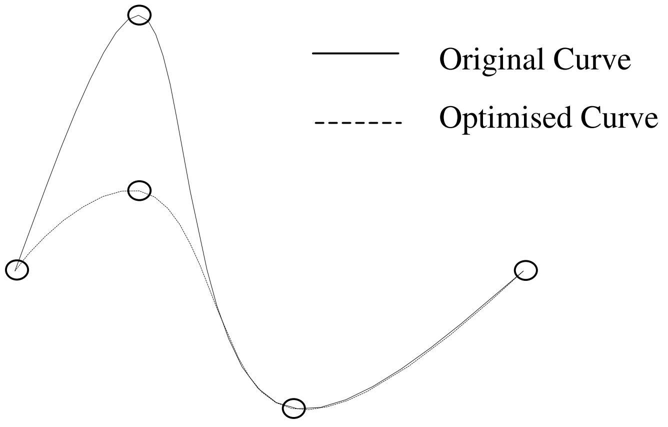


Figure 88: Optimised curve Example 3

GA points

	pt1_ga	pt2_ga	pt3_ga	pt4_ga
X	0.0000	12.7524	19.0729	38.6048
Y	0.0000	13.5537	-30.4664	-0.9365
Z	0.0000	0.0000	0.0000	0.0000

Original points

	pt1	pt2	pt3	pt4
X	0.0000	16.0000	20.0000	40.0000
Y	0.0000	20.0000	-30.0000	0.0000
Z	0.0000	0.0000	0.0000	0.0000

Difference

	pt1	pt2	pt3	pt4
X	0.0000	3.2476	0.9271	1.3952
Y	0.0000	6.4463	0.4664	0.9365
Z	0.0000	0.0000	0.0000	0.0000

Table 7: Results Example 3

Example 4

This curve is a space S-bend curve made from the four following points:

```
        position pt1  (0, 0, 0);
        position pt2  (10,10, 0);
        position pt3  (20, -10, 2);
        position pt4  (40, 0, 8);
```

The inflection point is located in the middle of the curve and the torsion is concentrated at the end of the curve.

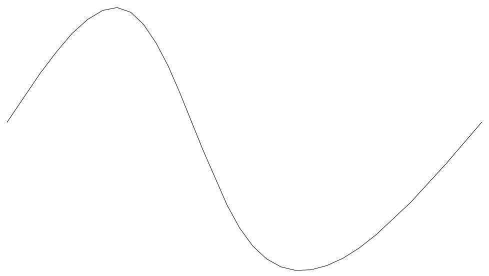


Figure 89: Original curve Example4

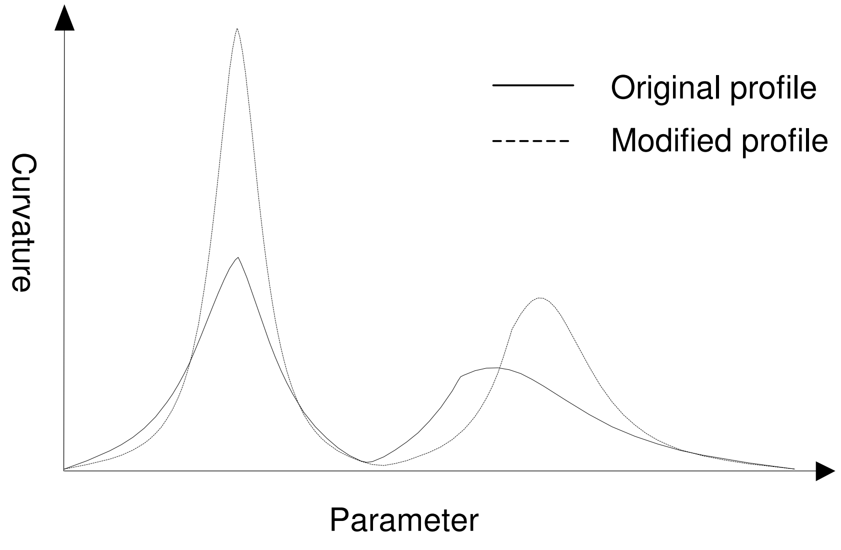


Figure 90: Modified curvature profile Example 4

The original curve’s curvature profile, Figure 90is not smooth and a more satisfactory profile is prescribed. The new profile, drawn in dotted line in Figure 90, accentuates the curvature picks and makes the profile more symmetric.

The variable bounds constrain the start point as a fixed point but all coordinate components are bounded variables.

```
        double upper_limit [12] = { 0.02, 0.02, 0.02,
                                    5, 5, 5,
                                    5, 5, 5,
                                    5, 5, 5};
                                          Parameter
          double lower_limit [12] = { 0.02, 0.02, 0.02,
                                    5, 5, 5,
                                    5, 5, 5,
                                    5, 5, 5};
```

The GA convergence Figure 91 is less evident than in the previous example. In this example torsion has been introduced in the curve and the z axis data point components have been set as variables.

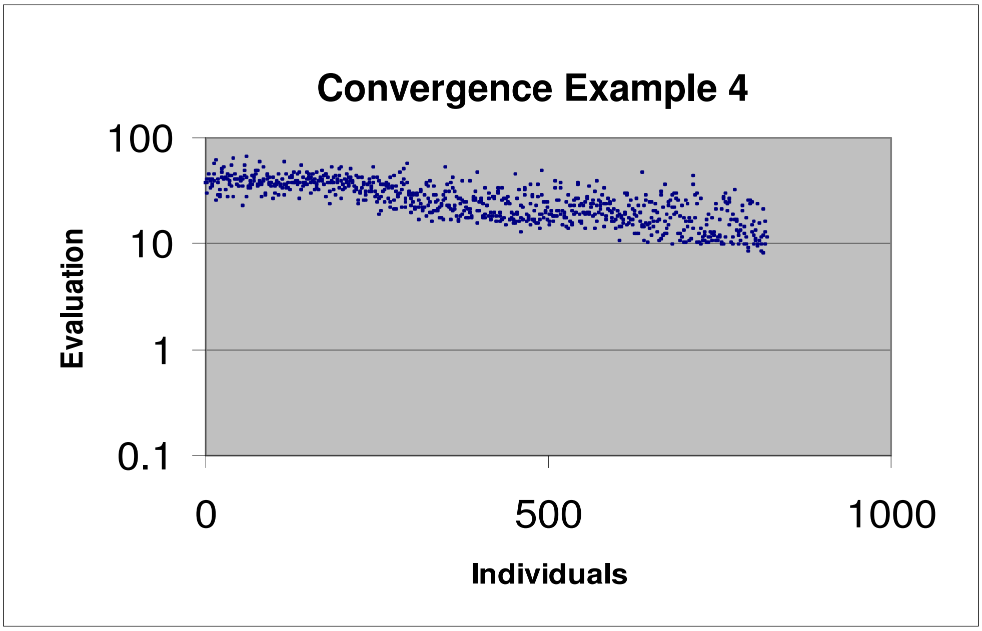


Figure 91: Convergence graph Example 4

GA points

	pt1_ga	pt2_ga	pt3_ga	pt4_ga
X	0.0000	10.0070	20.0177	38.6048
Y	0.0000	4.8101	-5.8936	-0.9365
Z	0.0000	0.0000	0.0000	0.0000

Original points

	pt1	pt2	pt3	pt4
X	0.0000	10.0000	20.0000	40.0000
Y	0.0000	10.0000	-10.0000	0.0000
Z	0.0000	0.0000	0.0000	0.0000

Difference

	pt1	pt2	pt3	pt4
X	0.0000	-0.0070	-0.0177	1.3952
Y	0.0000	5.1899	-4.1064	0.9365
Z	0.0000	0.0000	0.0000	0.0000

Table 8: Results Example 4

The changes of point coordinates summarised in Table 8 unveil that the ‘y’ axis component of the second and third points are modified. The other variable have a lesser variation.

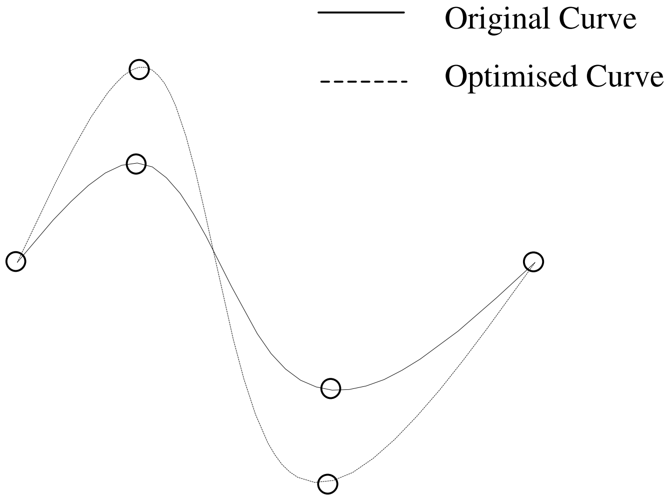


Figure 92: Optimised curve Example 4

The result curve is in accordance with the prescribed curvature profile despite that the algorithm did not converge as much as it should to guarantee an accurate result. Because the curve is allowed to twist, there the torsion profile should also be formulated in the evaluation function. As it stands there is no facility in the GA to include a second, independent evaluation function. A compromise can be set up where by the two evaluation functions are merged in one larger evaluation function. This could be achieved by adding or multiplying the two evaluation functions, but it is uncertain it would produce the right results. This strategy is off course far from the definition of multi-objective optimisation.

Discussion

This optimisation algorithm works well and produce almost acceptable results. However the run time is too long to envisage this algorithm to be incorporated in a design process. It is doubtful that industrial designers would accept waiting nearly one minute every time they modify a curvature profile. Also a lot of adjustments to the GA's parameters are required before it converges anywhere close to a solution. This point was mentioned earlier: it seems that another GA is needed to optimise the settings of the first GA. It was found, that if variables change decimal from one problem to another, the GA changes its behaviour noticeably and new settings are required. This is because of the way numbers are represented in a GA. The conversion from decimal to binary certainly causes problems. From the experiments conducted on this algorithm, it appears clear that the GA's performance drops as the problem complexity increases.

As it happens, during the development process of this algorithm, a little programming error was made in the evaluation function. In normal circumstances errors are easily spotted, a geometric modeller systematically returns error messages or returns a flag indicating the level of success achieved by a procedure, but in evolutionary computing, things are not as straight forward. In actual fact heuristics are able to do anything they are asked, even when the optimisation model is not mathematically sound, and it is incapable of telling if the operations it has made are valid or not. In this case, the optimisation algorithm was returning geometrically valid answers but that did not make much sense in the optimisation logic. For this reason countless tests on the evaluation function were made to try to understand the behaviour of the GA.

A range of curves with features is used to validate this curve optimisation application. Quite noticeably, the more variables the GA is dealing with the less it converges. This can be seen in Table 9, which summarises the GA performance with increased number of variables. A nine variable problem is performing one hundred times less than a four variable problem.

Engineering applications such as curve reconstruction from digitised data can have cross section curves made out of several thousand points. In the light of the present results, the GA will not cope with that many variables, only pieces of curves can be optimised at a time. This is not an advantage, because designers want to be able to work on the whole curve not on fractions of it.

Number of Variables	2	4	6	9
Best Solution	0.08	0.1	0.9	10

Table 9: GA performance with increased number of variables

The curves which are planar give better results than space curves for the same experiment configuration. While planar curves are solely defined by curvature functions, space curves are defined by the curvature and torsion functions. The application in its present form does not include torsion functions in its algorithm because the GA used here does not permit it. Including torsion function in the process would make the evaluation function double: one evaluation function for the curvature profile and another independent evaluation function from the torsion profile. And according to *Roy et al.* [99], we are in the presence of two evaluation functions, and the problem can be seen as multi-objective. On paper evolutionary computing is said to have wonderful capabilities: multi-objective, multi-modal, constrained (see Chapter 2). Researchers in the field *Roy et al.*[99] have shed light on real life optimisation promising a toolbox capable of handling a wide spectrum of problems. Unfortunately it has not been possible to find in *Tiwari's* work [99] procedures dealing with multi-objective optimisation problems needed for the example space curves optimisation. Also the techniques developed in the present research on the convergence of solutions is nowhere mentioned in his work.

8.2.2 Function Representation Driven

The disappointing conclusions on evolutionary computing have led the research to seek alternative means for doing curve optimisation. It was established in paragraph 6.2.2 that function representation was suited to the representation of differential properties, thus for optimising curves. In this paragraph, the problem of the determination of a curve given the curvature and torsion functions is discussed. The answer leads to finding a solution to the Frénet formulae. Approaches for both planar and space curves are considered.

Planar Curves

A procedure for finding the position vector $P(s) = X(s)I + Y(s)J$ from the curvature profile is proposed by *Nutbourne* [80] as follows:

$$X(s) = \int_0^s \cos\left[\int_0^\sigma \kappa(t)\right] d\sigma$$
$$Y(s) = \int_0^s \sin\left[\int_0^\sigma \kappa(t)\right] d\sigma$$

Equation 34

X(s) is measured along the initial tangent *T(0)* and *Y(s)* is measured along the initial normal *N(0).*

Implementing Equation 34 with ACIS poses some problems since it is not possible to perform symbolic integration of functions; the functionality offered by ACIS is limited to numerical integration. Equation 34 have been adapted to suit numerical integration: the first integral is replaced by a sum loop with an incremental parameter for *s*.

$$X(s) = \sum_{0}^{s} \cos\left[\int_{0}^{\sigma} \kappa(t)\right] d\sigma$$

$$Y(s) = \sum_{0}^{s} \sin\left[\int_{0}^{\sigma} \kappa(t)\right] d\sigma$$

Equation 35: Numerical integration of $\kappa(t)$

The problem with numerical integration is that approximation errors in *X(s)* and *Y(s)* are cumulated because of the sum as *s* increases, (see Figure 93). However it is clear that it would be possible to make an application that would reconstruct a curve from given curvature function with the procedure established by *Nutbourne* if symbolic integration was made available in a F-rep modeller.

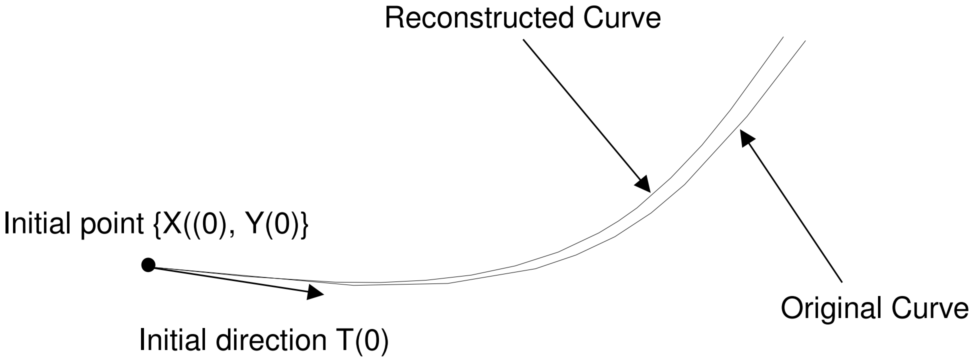


Figure 93: Curve reconstruction from curvature profile

Space curves

The case of space curves with non-null torsion function is more challenging, but several possible solutions have been found.

The first approach to finding solutions to the Frénet-Serret formulae, proposed by *Nutbourne* [80], is to set up a third differential equation for the normal N(s) by eliminating the tangent T(s) and the Bi-normal B(s) from the Frénet-Serret relations. When the solution to the equation is found, the position vector can also be found by integration of the tangent T(s).

This can also be achieved by writing three orthogonal complex vectors which components are the vectors of a moving frame of the Frénet type. The solution is found by solving an equation of the Ricatti form. Mathematical details are available from differential geometry volumes, *Aminov* [3], *Gibson* [53], *Eisenhart* [41].

The coordinates of a curve *P* can be expressed with Taylor expansion theorem, *Weatherburn* [120], *Farin* [44]. In practice it is not possible to implement this, the Taylor expansion can only serve as ground for theoretical reasoning.

Yet another NURBS approximation method is proposed by *Wolters* [122]. The technique attempts to approximate a curve by minimizing a metric defined in terms of curvature and torsion. This method is thought to be too complex to be implemented as it involved a lot of arithmetic due to the NURB representation.

Discussion

This attempt to recreate curves from prescribed curvature profiles was partially successful because the functionality exposed by the geometric modeller did not allow to accurately implement the theory in a functional application. Under the current functionality of ACIS, there is no possibility to perform symbolic integration. An substitute algorithm to symbolic integration was made up from incremental numerical integration, but could not perform well enough to produce acceptable results. However, given the right tools for the job, these algorithms have potential for success.

With the example of curves, a generic representation takes the natural equations (curvature and torsion), which give a unique solution independent of the coordinate system. Function representation avoids polynomial interpolation and provides a true definition of the geometry. From an optimisation point of view, this scheme is attractive as it implicitly holds properties that can be accessed directly without having to perform any property interrogation procedures. It was said that function representation is not efficient but the efficiency lost with the representation is largely regained when doing optimisation.

8.3 Prescribed Compound Fins

In paragraph 6.3.2 compound fins were introduced. The aim of this paragraph is to illustrate techniques prescribing compound fins and finding in reverse the corresponding curve.

Unfortunately given a 3D curve, the normal vector of its normal fin compound is not in the same direction as the normal vector of the curve. The calculus below shows the mathematical details of this. Given a general space curve *P(u)* and with the assumption that the representative function is continuous at every point on the curve, the normal fin compound is obtained from Equation 25 and is called normal fin here for simplicity.

$$\text{normal fin} = P(u) - sN(u)$$

The tangent of the normal fin is obtained by derivation of the normal fin.

$$\text{tangent normal fin} = \frac{\mathrm{d}\left(P(u) - sN(u)\right)}{\mathrm{du}}$$

By substitution with the help of the Frénet formulae,

$$\begin{aligned}\text{tangent normal fin} &= T(u) - s\left(\tau B(u) - kT(u)\right)\\ &= T(u)(1 + SK) - s\tau B(u)\end{aligned}$$

Equation 36: Derivation of 3D normal compound fin

Equation 36 shows that the tangent of the normal fin compound has a term in the direction of the bi-normal of the curve itself. This system of equations has one too many unknowns; consequently it cannot be solved. The same problem occurs with the bi-normal fin compound.

Curves for which the normals of the normal compound fin are in the same direction as the parent curve have been studied by *Bertrand* and *Mannheim*. These curves have curvature and torsion profile dependent, thus they are classifies as helices *Nutbourne* [80]. It is demonstrated in this paragraph that planar curves also have parallel normals with their normal compound fin. Therefore the projection technique introduced in paragraph 8.1 is used to reduce the dimensionality of space curves to planar curves. Finally, the transformation of *Combescure* [80] is considered for with general space curves with independent curvature and torsion profiles.

Projection Technique

The problem of modification of compound fins stated above is tackled in here using projection techniques. The first step is projecting the subject curve into the three planes forming the orthogonal reference x, y and z as shown in Figure 94. Using Equation 25 given in paragraph 6.3, a normal fin compound can be applied to each of the projections. Because the projections are planar, the compounds obtained are in the same plane as the projections hence the bi-normal is constant null.

At this stage the user can modify the profile of one compound at a time and it must remain in the same plane as the original compound and the curve. This later condition can be maintained by projecting the modified compound on the required plane. The compound modification allows the capture of the so-called designers' intent. This means that a design can be generated from a prescribed normal vector distribution. From the prescribed compound, a new projected curve is produced. This is achieved by manipulating the Frénet formulae.

$$\text{normal fin} = P(u) - sN(u)$$
$$\text{tangent normal fin} = \frac{\mathrm{d}\left(P(u) - sN(u)\right)}{\mathrm{du}}$$

By substitution with the help of the Frenet formula

$$\text{tangent normal fin} = T(u) - S(\tau B(u) - kT(u))$$
$$B(u) \text{ is null, therefore}$$
$$\text{tangent normal fin} = \left\| T(u)(1 + SK) \right\|$$
$$\text{normal normal fin} = \left\| \frac{d(\text{tangent normal fin})}{\mathrm{du}} \right\| = \left\| KN(u) \right\|$$

Equation 37: Derivation of 2D normal compound fin

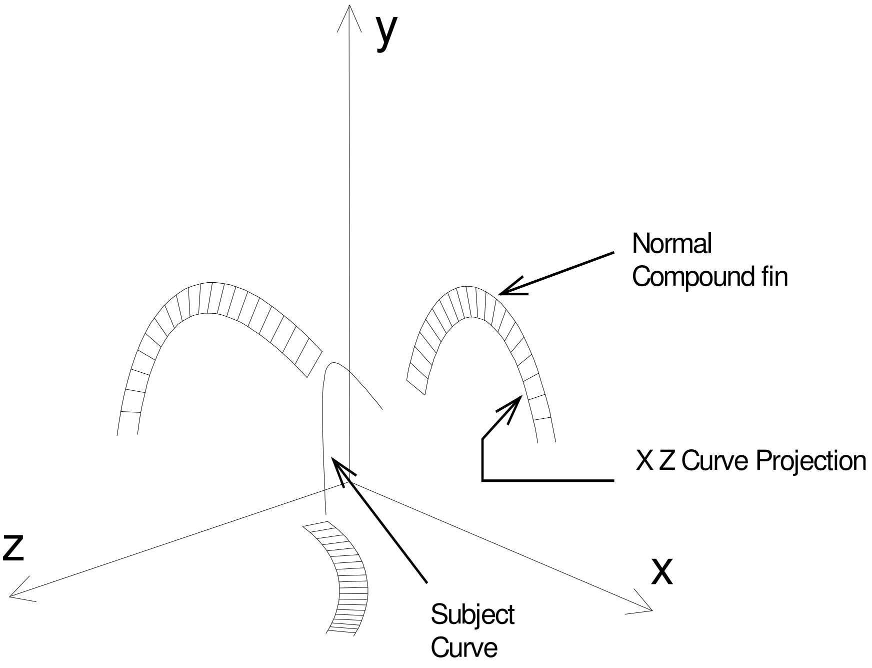


Figure 94 Normal fin compounds of curve projections

Equation 37 proves that with planar offsets, the normal vector field of a curve's normal fin compound is in the same direction as the normal vector field of the original curve's projection. The reciprocity explained above enables the reconstruction of a curve's projection from a prescribed planar compound fin. By scaling the normal vector by the same factor and subtracting this from the prescribed compound, a curve in the projection plane is reconstructed and has for normal compound fin the normal compound fin prescribed by the user. The remaining task of reconstructing a 3D curve is achieved by constructing a 3D vector from the two vector components prescribed and the third unchanged component. Figure 95 illustrates the optimisation process.

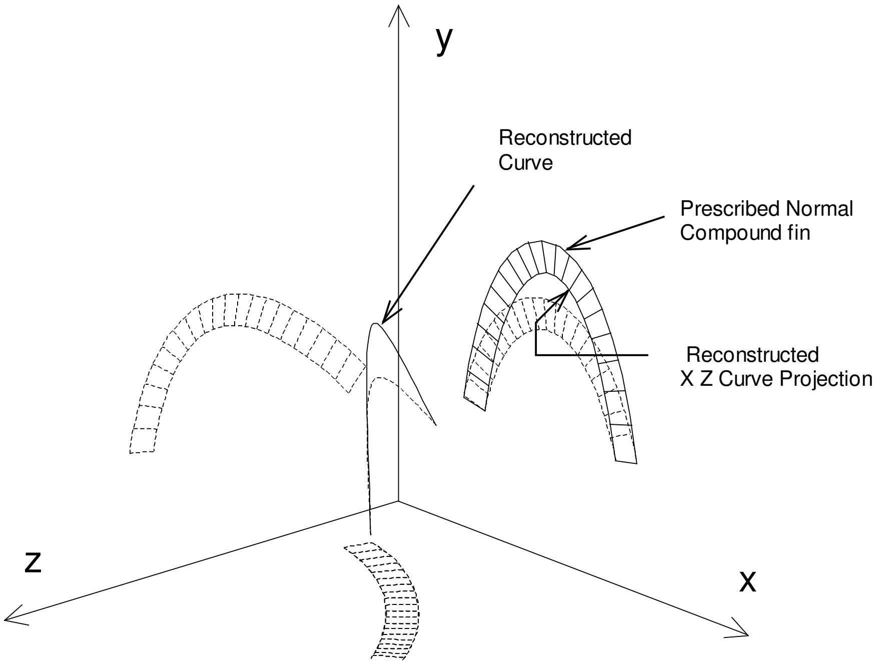

y
Reconstructed Curve
Prescribed Normal Compound fin
Reconstructed X Z Curve Projection
z
x

Figure 95: 3D curve reconstruction from prescribed normal fin compound

Combescure Transformation

The name of *Ed. Combescure* was found in the archives in “Comptes rendus des séances de l’académie des sciences”. In the algebra section, *Combescure* writes on a theorem by *M. Hermite* relating to the transformation of equations [27]; it is the only reference to *Combescure* that could be found. *Nutbourne* [80] also state that he could not trace the name to any mathematicians and there is no evidence that *M. Ed. Combescure* is the originator of the transformation theorem presented in this paragraph, but there is a possibility because other volumes dated around the publication date of [27] include articles by *Manheim, Bertrand, Chasles, Frénet* and others whom are know for their work on differential geometry of curves and surfaces. Further research in the archive of the “académie des sciences” in Paris could uncover the mystery author’s identity.

The Combescure transformation is applicable to deal with a wider range of curves. The basic idea is that the compound curve shares the same set of tangent vectors as the parent curve. From this property follows that the normal and bi-normal vectors are also parallel to that of the parent curve for points that are in one to one correspondence.

The Combescure compound fin vector *Q(s)* is described as a function of *s* and is obtained by adding the parent curve vector function with the Combescure transformation vector *R(s)*.

$$Q(s) = P(s) + R(s)$$

Equation 38

The Combescure transformation vector can be expressed in the curve's Frénet frame coordinate system.

$$R(s) = a(s)T(s) + b(s)N(s) + c(s)B(s)$$

Equation 39

Where *a(s), b(s), c(s)* are functions of the arc length of the parent curve. Differencing Equation 39 gives

$$R'(s) = [a'(s) - b(s)k(s)]T(s) + [b'(s) + a(s)k(s) - c(s)\tau(s)]N(s) + [c'(s) + b(s)\tau(s)]B(s)$$

Equation 40

The tangents have to be parallel so the following conditions must be met:

Equation 41 $b'(s) + a(s)k(s) - c(s)\tau(s) = 0$

Equation 42 $c'(s) + b(s)\tau(s) = 0$

Equation 43 $(ds/ds^*)[1 + a'(s) - b(s)k(s)] = 1$

The character * denotes the compound fin.

A particular example where b(s)=0 is of particular interest:

From Equation 39, $C(s) = a(s)T(s) + c(s)B(s)$, and because *b(s)=0*, from Equation 42 *c'(s)=0* so *c(s)=C* a constant. And therefore from Equation 41 $a(s) = C\tau(s)/\kappa(s)$. Substituting these results in Equation 39 gives

$$R(s) = C[\{\tau(s)/k(s)\}T(s) + B(s)]$$

Equation 44

Each point on the parent curve are moved in the rectifying plane along the vector *R(s).* Further more it is easy to see from the demonstration above that the inverse of the Combescure transformation applied to the compound fin is the parent curve. However the Combescure transformation does not work for planar curves or in sections of null curvature because of the fraction in Equation 44.

Discussion

The examples of reverse engineering of compound fins described above, all suffer from the same drawback, that is the reconstruction of curves from higher differential order curves may induce irregularities rather than removing them. For example if the compound curve has a dent, the resulting curve from the reverse process will inevitably swing direction at the points of inflection. For this reason the compound fins should only be limited to curve inspection. For the purpose of optimisation, the curvature and torsion profiles are more appropriate.

8.4 Conclusions

In this chapter, intrinsic curve optimisation algorithms have been established as Curve Projection in paragraph 8.1, Prescribed Curvature Profile in paragraph 8.2 and Prescribed Compound Fins in paragraph 8.3. These algorithms have demonstrated the implementation of interactive curve optimisation using the fundamental theorems of differential geometry introduced in Chapter 6. Compared with other curve optimisation methods available in literature *Farin* [44], *Sapidis* [106], *Ferguson* [48], *Anderson* [4], *Jones* [65] and the application called Prescribed Curvature Profile driven by Genetic Algorithm which uses NURBS, the proposed use F-Rep for exact vector representation of curves, hence avoiding dealing with control points and the problems of interpolation.

Compared with NURBS, continuous vector functions are more computationally expensive, but this drawback is largely overcome by the fact that the result of optimisation is guaranteed to be what the designer wants the first time. As far as the author is aware the proposed method is new and constitutes a small but significant advance in the field of curve optimisation. The F-Rep modelling functionality provided by ACIS did not fulfil the requirements for the applications implemented in this thesis. The limitations are justified by the fact that the F-Rep modelling capabilities in ACIS are designed for performing sweeps. The functionality needed for the curve optimisation applications have been created from existing functionality available within ACIS and the outcome is quite satisfactory. However for the purpose of curve recreation from prescribed intrinsic properties, symbolic integration is required, paragraph 8.2.2, and is not available from ACIS. In order to complete this work section, other math-specific tools combined with ACIS would provide the functionality needed to perform symbolic integration. Software like MATHLAB could be considered for this purpose. MATHLAB exhibits a C++ interface as well as a specific language interface. This reinforces the idea of flexible generic API put forward in this thesis

Some of the algorithms presented in this chapter have been implemented according to the three-circle diagram presented in Figure 1. The applications make use of foreign functionality borrowed from different satellite function libraries. The functionality needed by the application is made available through an application programming interface. Beyond the potential usefulness of the curve optimisation applications developed within the framework, they have demonstrated the need for appropriate flexible interfacing between applications and functionality suppliers.

.

Part III

Chapter 9 Application Programming Interface for Flexible Optimisation

The areas of design optimisation, geometric modelling, differential geometry have been identified early on as first-role actors of the framework. A first synthesis has been produced by innovative research in the optimisation of free form curves which required a combination of the three areas. These experiments have uncovered the need for flexibility in the framework architecture to integrate functionality from different sources into the framework. Studies in the interaction between the constituting fields has led to the determination, mapping and documentation of the requirements and solutions for the design of the framework. This part of the thesis contributes to further establish knowledge in the integration of flexible optimisation in the CAD environment through programming interfaces.

The remaining work of this research is going towards a computer implementation of the framework. It was found that optimisation could be represented by evolutionary computing algorithms; still questions remain. A better possibility for optimisation is given by representing geometry with arithmetic functions. The geometric representation is encapsulated in a geometric modeller such as ACIS. As for CAD, after investigation the solution to create a new user interface using the Microsoft Foundation Classes together with ACIS was retained. A running application called HulaHoops was created to demonstrate the research findings, a description of the user interface in the form of a user manual is included in Appendix 8.

The inter-disciplinary nature of this research has lead the research to study in ways to interface the constituting members of the framework. In the forthcoming discussions, these are called modules. The discussions that emerged from the work on design optimisation, geometric modelling and curve optimisation together with the solid modelling, interface found in Djinn, *Bowyer et. al.* [16], provide a design framework of the Oli application programming interface (API).

9.1 Framework Architecture

This paragraph presents a survey of selected CAD systems' architecture. Based on the requirements placed by the framework and the tasks set to perform, architectures models are proposed and discussed.

9.1.1 Conventional Architecture

At the initial stages, it was thought that a main stream turn-key CAD system would be used as a base framework for the research. Various CAD systems' architectures have been investigated for this purpose. The results drawn from this experience are reported here.

The UNIX platform was favoured over Microsoft, and the reasons put forward were processor speed and stability needed to run genetic algorithms. The down side of UNIX is the level of expertise required to set-up the environment. For example, setting-up the SDRC IDEAS open interface is a major task best left to professionals. The question of processor speed is relevant to conventional users that use a particular CAD system professionally for creating large 3D models. However for the purpose of research and development, flexibility is central. The Microsoft environment, far from being perfect, is flexible and intuitive, which are the qualities required for doing a research project. Another product from SDRC, IMAGEWARE SURFACER was considered as a replacement to IDEAS and this time the Microsoft environment was retained. Likewise the results were not satisfactory because of the architecture system's architecture that is not suitable for doing advanced application development.

The architecture of IDEAS as described in the documentation is presented in Figure 96. According to the diagram the open interface gives access to both the CAD model and the CAD system. In actual fact the so called open interface exposes the function headers of the user interface, but does not expose the geometric kernel. Some of the functionality necessary for doing optimisation was not available from the API. In particular interrogation functionality and differential geometry was missing. This is was also found in other commercial CAD systems like and AUTOCAD and generally with old systems whose geometric kernel was originally developed in C language and have not been re-written in an object-oriented language such as C++.

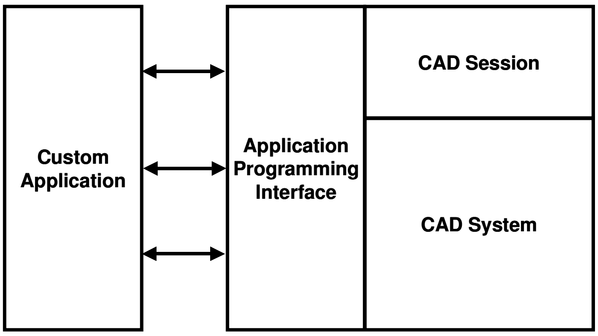


Figure 96: Conventional architecture

Moreover, the absence of real API is confirmed by the presence of obsolete data communication protocols “client server” like CORBA or COM. In the eventuality such systems had to be used for this research, Figure 97 shows what the framework architecture would look like. The CAD system (and its geometric kernel) is separated from the application (in this case the Flexo project) by a wrapper software, which is in actual fact a communication protocol wrapped in patch code to establish the link between the two parties. This scenario is far from satisfactory to achieve integration of flexible optimisation within the CAD environment.

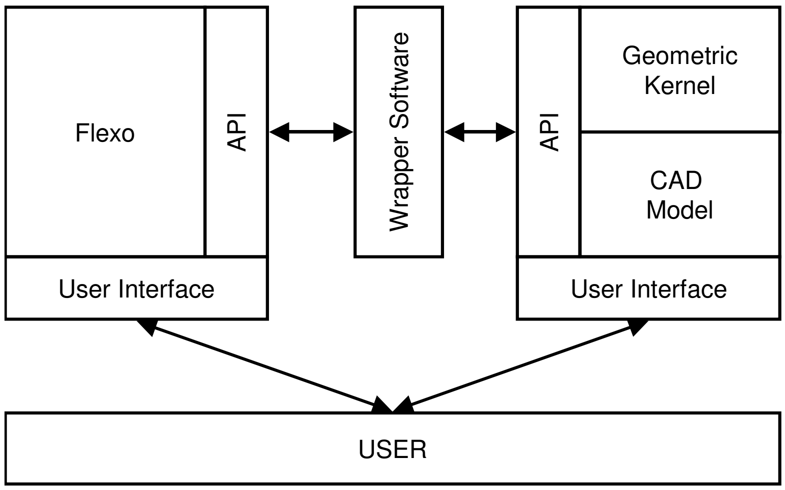


Figure 97: Conventional architecture strategy

Under those circumstances, It was proven impossible to develop a working application for design optimisation. This discussion joins what has already been said in paragraph 3.8 about cooperation between CAD vendors and researchers.

9.1.2 Modular Architecture

In order to achieve the objectives of the research, another approach is undertaken: "kit CAD", so to speak. This concept is a concept borrowed from the car industry, where a car is made of bits taken from different car models and assembled together to make a custom car. In the context of CAD, a kit CAD is made from various systems each of them serving a specific purpose. As already stated, ACIS is the underlying geometric modeller, an Open GL graphic interface supported by the Microsoft foundation classes (MFC), the Flexo optimisation toolbox and the Oli API for interfacing all these. In effect this strategy is exploding the traditional CAD environment into smaller systems independent from each other. The current trend in CAD is going towards this type of architecture; CAD vendors have understood that one CAD system could not enclose all the functionality required by the vast field of mechanical engineering. A new terminology called Computer Aided Engineering (CAE) has recently emerged out of the concept of the multidisciplinary requirements of engineering. Notably CATIA is structured in modules that the user can acquire on demand to perform some specific tasks. Nonetheless the API exposure remains limited making flexible interaction with outside applications difficult.

Process Oriented Architecture

In the Process oriented architecture, the applications processes are placed at the cross roads of all modules. Figure 98 shows that all modules have their own APIs, which exposes their functionality to applications. In this configuration, the applications are responsible for providing the modules with the right data. This involves making data type conversions every time the information is passed between modules. For example in the curve optimisation applications with GA in paragraph 8.2.1, the curves data points are represented as 3D coordinates in ACIS and as an array of variables in the GA. So the applications have to translate the data from the ACIS type to the GA type and vice versa.

The problem with the process oriented architecture is that modules live independently and have no awareness of each other. For example an object's geometric definition is stored in the geometric modeller but is also stored as an individual in the optimisation module and as a render in the graphic interface. The multiple existence of object representations without connection records causes some problems. If an object is deleted in one module, it won't necessarily be in the others. In the best case it will cause a repetition of function calls for a query that has been already made before. Further, in the worse case applications or modules will make a function call on an object that has been deleted elsewhere. Similarly objects can be left without knowledge of their existence by the applications in some modules, the memory requested for these objects cannot be recovered and technically speaking the application is creating a memory leak.

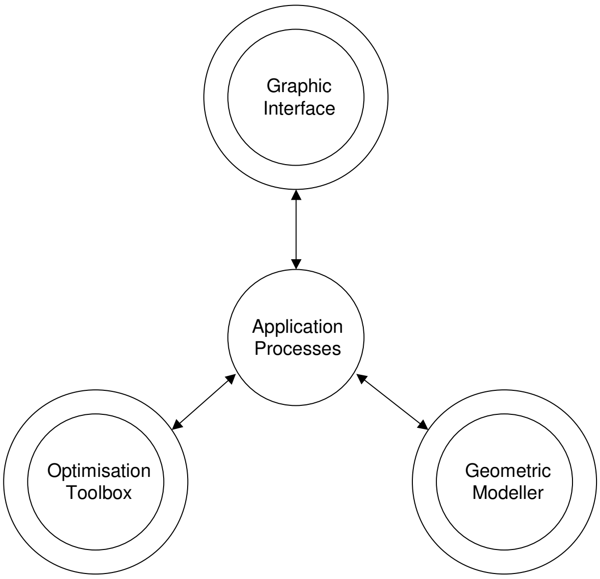


Figure 98: Task oriented architecture

Object Oriented Architecture

The object oriented architecture takes advantage of the principles of object orientation to design a centralised interface. Unlike with the task-oriented architecture, the applications are no longer central. The central core is occupied by a collection of generic objects common to all modules, see, Figure 99.

The common interface to all modules works like a switch board, establishing connections between modules on request. It is not necessary to store data in the interface, rather pointers to the data stored in the modules is recorded in the interface. The independent aspect of the interface is proven particularly useful to gain in flexibility. In some instances, curves are represented as NURBS or as functions for the purpose of optimisation. It is even possible that these two representations are implemented in different modellers. The discussions in Chapter 8 highlighted the fact that more than one modeller was needed and suggested using MATHLAB in conjunction with ACIS for differential calculus.

This discussion corroborates the need for an interface independent of a particular modeller. A proposition is made to define interface objects on the basis of their geometric nature. The objects are not defined in terms of a particular representation paradigm, rather the API supports associations with the objects representations across the framework. Access to the data is provided by attributes attached to the objects. As previously seen, the curve object can be represented geometrically as NURBS or as vector functions, which can be a differential representation of the curve (curvature, torsion). Non-geometric data are also supported, for example in the GA optimisation process, curves are evaluated upon some criteria and are given fitness values. This information is also included in the list of attributes.

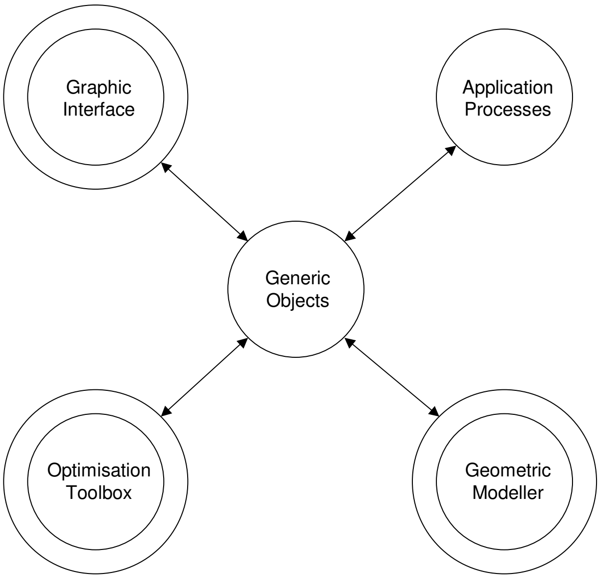


Figure 99: Object oriented architecture

Top and Bottom Interface

In the object oriented architecture model, the top interface is the first interface that is exposed to applications and modules. It is defined in terms of abstract geometric data and procedures. The implementation of these data is achieved by the functionality of the modules through their own APIs. This way the applications are not tied to any of the modules but to the generic interface. This guarantees that the applications still exist even if the modules are removed or replaced by others.

Discussion

At the moment, commercial turn-key systems are not adequate for developing applications because their architecture is not designed for this purpose. Their APIs expose high level functionality, closer to the user interface than the geometric modeller. The limited functionality together with inappropriate documentation make these systems unsuitable for research exercises. The alternative to turnkey CAD systems is the making a custom environment out of separate pieces. The elements are chosen upon their API functionality and documentations. The needs to have a generic interface independent of all representations (geometric, optimisation, graphic) has been established. The forthcoming paragraphs are about the definition and elaboration of the interface proposed above.

9.2 ACIS Interface

The ACIS geometric modeller provides the geometry foundation for applications. These may interface to ACIS through Application Procedural Interface (API), C++ classes and in some cases Direct Interface (DI) functions. AICS also provides an interface to Microsoft Foundation Classes (MFC) for Microsoft Windows platform applications.

API functions provide the main interface between applications and ACIS. An API is a function that an application calls to create, change, or retrieve data. An API function combines modelling functionality with application support features such as argument error checking and roll back . When an error occurs in an API routine, ACIS automatically rolls the model back to the state before that API routine was called. This ensures that the model is left in an uncorrupted state. ACIS guarantees API functions to remain consistent from release to release, regardless of modification to low-level ACIS data structures or functions.

The class interface is a set of C++ classes that are used to define the ACIS model geometry, topology and other characteristics. The classes may be used by applications to directly interact with ACIS through their public and protected data members and methods. The ENTITY class is a base from which many ACIS classes are derived. It implements common data and functionality that is mandatory in all classes that are permanent objects in the model, although ENTITY does not itself represent any specific object. The ATTRIB class which is derived from ENTITY, is used to derive specific attribute classes for the system and user attributes.

Direct Interface functions provide access to modeller functionality without the additional application support features of an API. Unlike APIs, these functions are not guaranteed to remain consistent from release to release. DI functions are not available to access all of the functionality in ACIS. They are generally used for performing operations that do not change the model, such as enquiries.

9.3 Flexo Interface

The documentation written by *Tiwari* presented in Appendix 2 references the GA programming structure, but does not reveal the existence of an interface for the optimisation toolbox. The GA experiments produced in this research have shed light on functionality needed for GAs. These functions which name begin with prefix Flexo also have for origin the experiments on GAs reported in this thesis.

9.4 Graphic and User Interface

Microsoft Foundation Classes (MFC) consist of hundreds of classes designed to make Windows programming easier and quicker. MFC offers the convenience of reusable code, because many of the tasks common to all Windows applications are provided.

The ACIS Microsoft Foundation Classes (AMFC) provide the code necessary to interface from ACIS to the Microsoft Foundation Classes. Most of the classes are derived from MFC classes with hooks into ACIS. In addition, several tool classes exist to facilitate operations that most applications require, such as camera movement, mouse movement, dragging operations, Boolean operations, and drawing lines, circles, and fillets. AMFC also contains wrapper functions to ACIS API functions.

AMFC provides powerful means of creating the barebones structure of an ACIS based application. AMFC has been used in this research to create an ACIS based application called

HulaHoops. This application aims to validate the research findings reported in this thesis. These comprise of curve optimisation algorithms and the programming interface for flexible optimisation presented in this chapter. This contradicts slightly with the framework architecture model presented in paragraph 9.1 in the sense that AMFC shortcuts the independent Oli interface which makes the application tools dependent of the geometric modeller. This concerns the mouse event handler, the graphic display, the file manager. However the concepts of a generic interface and modular architecture remain valid.

9.5 Oli Interface

9.5.1 Objectives

The Oli interface aims to define a set of abstract geometric data and procedures that different types of modellers can support in an uniform way. The primary purpose of the interface is to document the requirements placed by optimisation on it. The Oli API has focused on the functionality required for the optimisation of free-form curves. The underpinning implementation of the proposed API is proposed whenever the modules functionality allowed it. In the contrary, an abstract implementation is maintained but the routine takes no action.

9.5.2 Background

The Djinn interface [38] is a representation independent geometric interface for solid modelling. Djinn is has been designed and documented by consensus of application researchers and the UK Geometric Modelling Society.

Djinn is a representation independent API in the sense that it is not written for a specific underlying geometric modeller. The novel aspect of Djinn is the specification of operations on point-sets. It supports a version of ‘cellular modelling’ in which there is no boundary representation or CSG structure visible at the API. Thus the Djinn API is specified purely in terms of the required functionality and data abstraction needed to support it; it is left to the underlying implementation to define how the that requirement is met. On the same ground, Djinn is not meant to support a particular application, but rather provides functionality for a wide range of applications across the CAD/CAM environment. Djinn is a collection of documented functions that can support calls from applications to the modeller.

The Djinn interface is used in this research as a guide line for the definition of the Oli interface. Some of the issues addressed by Djinn, which are shown to be relevant to the design of the present API are re-formulated for the purpose of this research. The scope of the Djinn interface has more width than depth: at the moment no implementation has been proposed, which leaves opportunities to further the research, especially in the area of optimisation and free-form curves.

The Djinn specification identified that the diversity of representations of free-form geometry made the generic formulation of them a challenging task. Therefore it has chosen to provide support import of reference to external data through the top interface, and interrogation through the bottom interface. A set of procedures part of Djinn allow properties such as surface normals to be discovered. The discussions put forward in this paragraph 9.1 certainly go in that direction. Several representations of curves are needed by applications. NURBS is the most accepted format for curves and surfaces and is a convenient way to perform local operations on shape but is not suitable for a more formal approach to curve design and optimisation. Function representation (F-rep) is thought to hold more promises for systematic curve interrogation and reconstruction from prescribed properties. The opinion here is that F-rep is a far more generic representation paradigm than NURBS and it could very well serve the purpose of a generic interface for curve and surface modelling.

9.5.3 Generic Interface

The generic aspects of the Oli interface are formulated in the definitions, stated below, required for the functioning of the present framework regardless of the particular requirements of specific applications. However the discussion uses examples of research applications including the curve optimisation applications from the previous chapters.

Precision and Formality in the definition of the API

The Oli API is the specification of procedures that supports calls between a CAD modeller and some outside applications. The precision in the procedure definition is an issue that needs looking at. The Procedures can be defined in different languages and levels of formality. The functional requirements placed by the application on the API and vice versa are documented; the procedures are defined with their implementation in computer language C++.

Procedure Hiding

The interface takes advantage of object oriented programming for hiding procedure implementations at the API. The interface exposes a list of functions and classes that applications can invoke and pass parameters to it.

State

The API state is defined by a single variable. A state variable is recorded each time a call is made to the API. This state variable allows the API can go back to its last recorded state at any point in time. It can also be used to return the original API's state after a procedure was called and forced the API to an undesirable state. In some languages procedures need initialisation and finalisation, the state variable can provide state information to the procedures.

Side-effects

Procedures are said to have no 'side effects' when no data except arguments are accessed or altered. Arguments can either be input parameters or output parameters. Input parameters cannot be modified by a procedure, while data present in output parameters may be overwritten when a procedure is called.

This issue is addressed by the Oli API since the Flexo toolbox extracts parameters from the geometry, modify the parameters and subsequently return them with new values. It is essential that parameters integrity is maintained when travelling through the API.

Error Reporting

It is essential in API design to look at error reporting. Some feed back information must be provided to report on the level of success achieved by any of the procedures.

Labels

In the geometric modeller, pointers refer to each entity created and some memory space is allocated to store the entities. It is not good practice to give applications direct access to pointers; instead entities are named with labels referring to the objects. Interfacing object with labels is a safety measure to prevent applications doing illegal operations on the geometry and preserve the integrity of the model.

Applications perform operations that might change the structure of the model. For instance, what happens when a Boolean operation is applied on an entity and the result of the Boolean is two separate objects? The application needs to keep track of the changes otherwise it might show the label referring to an object that does not exist anymore, or has been split into two.

Looking back to Figure 99, which part of the model should take care of the labelling? One possible option is to assign a part of the API to labelling. Labels are made available to the applications to read only. Applications retrieve from the API (not the modeller) up-to-date information on the labels. This way, applications are kept independent of the modeller, easing implementation issues.

Persistence

It is expected that the API has some mechanism to remember its state as well as other data such as labels. The problem of persistence occurs when the API need to restoring its state when a new session is started. At another level, persistence is an issue since the API is modeller independent and different modellers might use different labelling conventions. A model imported from a different CAD system might reference entities otherwise than the implemented system. The API needs to be consistent in labelling and independent of geometric modellers.

To overcome the problem of non-existing functionality from the Kernel API, Oli has concentrated on combining existing procedures as an attempt to make-up some functionality. The aim is to reduce the number of procedure definitions and therefore minimise the memory usage and increase efficiency. Moreover, combining procedures also reduces development time.

9.5.4 Scope of the Oli API

Geometric Objects

Geometric entities such as curve, surface and solid are considered for the generic specification of the Oli API. At an implementation level, free-form curve geometry is considered, this serves the purpose of case study for validating the API.

Geometric Operations

The geometric operations supported by the Oli API consist of curve modification procedures. For NURBS the geometric operations mainly consist of modification of the control points coordinates, tangency conditions at control points and parameterisation. For F-rep curves geometric operations are offset transformations like the Frénet frame or the Combescure transformation.

Properties

Applications make calls to the modeller to interrogate object properties. Examples of properties for curves are curvature and torsion, Frénet frame and mass, moment of inertia for solids. Property interrogation is done through successive API function calls.

Constraints

Constraints are present everywhere in the framework. Constraints of geometric nature are the data points coordinates, tangency conditions, curvature, torsion. The geometric constraints are supported by the geometric modeller but their representation is defined at the API level.

Attributes

In the proposed framework, attributes are used for making associations between the abstract definition of objects in the interface and external data such as their geometric representations, optimisation definition and constraints.

Variables

Variables are modifiable values that define the model geometry. The nature of variables varies with the type of geometry the application is dealing with. The optimisation application works on variables extracted from the model. The job of the API is to make variables accessible to applications. Optimisation places requirements on the API in terms of the number and nature of variables that are passed across.

Chapter 10 Procedure Definition

This chapter summarises the definition of the principal procedures implemented in this research. They are classified in two sections: those regarding GA optimisation and those regarding geometric operations. These procedures are used by the application software HulaHoops proposed by the author for demonstrating the findings of the present research.

10.1 GA Procedures

This list is the input / output parameters of the GA module. Note that a method for translating from the type position (inherited from ACIS) of 3D points to the type arrayr (array of real values).

10.1.1 GA Process

Input

GA operators

```
integer umber of generations
integer Population sisze
double Cross over probability
integer Cross over level
double Mutation probability
Integer Mutation level
```

Variables

```
Integer Number of variables
double resolution
Translate (position vertices, array variables)
Translate (initial vertices, array initial population)
double upper limit
Arrayr lower limit
```

Fitness function

```
fitness function
double power boosting start
double power boosting end
```

Output

Individuals

```
Translate (array variables, position vertices)
double fitness value
```

Solutions

```
Translate (array variables, position vertices)
Error error message
File Debug files
```

GA process Implementation

This function takes in all the parameters to run the GA process and returns optimised variables. At the bottom the fitness function and fitness data methods are mentioned. These are invoqued by the algorithm to evaluate each individual. Inside these, the fitness function and evaluation function are inserted. As they are specific to an optimisation problem, the definition of the evaluation function is done by the application.

```
void flexo_GA_optimiser(
//input
//GA operators
  int nogen,                //number of generations
  int popsize,              //Population sisze
double crossp,              //cross over probability
  int prcrossp,             //Cross over level
  double mutp,              //Mutation probability
  int prmutp,               //Mutation level
                            //Variables
 int nvertices,                     //Number of variables vertices
  double res,                       //resolution (default resab
                                    //inherited from geom kernel)
  position initial_vertices[],//array of  initial positions
  double upper_limit[],             //array of limits upper
  double lower_limit[],             //array of limits upper
  double power_start,
  double power_finish,
                                    //Output
  position opt_vertices[],   //array of optimised vertices
  char const* final_curve,  //debug file
  char const* converge              //debug file
  ){
  ofstream outconverge (converge, ios::out );
  if ( !outconverge) {                  // overloaded ! operator
          cerr << "File could not be opened" << endl;
          exit ( 1 );                                    // prototype in stdlib.h
  }

  Individual *oldpop[ 100 ];
  Individual *intpop[ 100 ];
  Individual *newpop[ 100 ];

  double power = power_start;
  int lenstr = 0;
  int maxdecpt = 0;
  srand( 67 );         // generates true random numbers
  int numvar = nvertices * 3;

  input_parameters(popsize, nogen, crossp, prcrossp, mutp, prmutp, numvar);

Arrayr lowerbound(numvar);
Arrayr upperbound(numvar);
Array lenvar(numvar);
create_bounds_from_positions(    upperbound,
                                        lowerbound,
                                        numvar,
                                        initial_vertices,
                                        upper_limit,
                                        lower_limit);

matrices( lenvar, upperbound, lowerbound, numvar, lenstr, maxdecpt );
  initialise( popsize, oldpop, lenvar, upperbound, lowerbound, lenstr );
  initialise( popsize, intpop, lenvar, upperbound, lowerbound, lenstr );
  initialise( popsize, newpop, lenvar, upperbound, lowerbound, lenstr );
  for ( int i = 0; i < nogen; i++ ) {
  roulette( oldpop, intpop, popsize, maxdecpt );
reproduction( intpop, newpop, popsize, crossp, prcrossp, mutp, prmutp );
          for (int  j = 0; j < popsize; j++ ) {
          *oldpop[j] = *newpop[j];
          outconverge << newpop[j]->fitdata() <<endl;
          }
          power = power + (double (power_finish) - double (power_start)) / double (nogen);
```

```
 }
}

double Individual::fitdata() const{}
double Individual::fitfun() const{}
```

10.1.2 GA Translators

GA translators are used to transform data from position type (3D data points inherited by geometric modeller) the to Arrayr type defined in the GA data structure and vice versa.

The oli_create_positions_from_GA_variables() extracts positions from single element (index) of the GA population

```
void oli_create_positions_from_GA_variables(
  int nb_points,
  Arrayr &Varval,
  position new_positions[]
  ) {

  for (int i=0; i<nb_points; i++){
                new_positions[i].coordinate(0) = Varval[3*i];
                new_positions[i].coordinate(1) = Varval[3*i+1];
                new_positions[i].coordinate(2) = Varval[3*i+2];
  }
}
```

The oli_create_bounds_from_positions fiction takes initial point coordinates and returns bounded GA variables.

```
void oli_create_bounds_from_positions(
  Arrayr &upperbound,
  Arrayr &lowerbound,
  int numvar,
  position vertices_ga[],
  double upper_limit[],
  double lower_limit[]
  ) {

  for(int i = 0; i<numvar; i+=3){

  upperbound[i] = vertices_ga [i/3].coordinate (0) + upper_limit[i];
  upperbound[i+1]         = vertices_ga [i/3].coordinate (1) + upper_limit[i+1];
  upperbound[i+2]         = vertices_ga [i/3].coordinate (2) + upper_limit[i+2];
```

```
        lowerbound[i] =  vertices_ga [i/3].coordinate (0) - lower_limit[i];
        lowerbound[i+1] = vertices_ga [i/3].coordinate (1) - lower_limit[i+1];
        lowerbound[i+2] = vertices_ga [i/3].coordinate (2) - lower_limit[i+2];
        }
      }
```

10.1.3 Parameter Bounds

The Parameter Bounds function is used by the GA, it takes an array of knots and returns two arrays of bounded parameters.

```
      void flexo_parameter_bounds(
                                              int npts,
                                              double param[],
                                              Arrayr lowerbound[],
                                              Arrayr upperbound[]
                                              ){
      for(int i =0;i<npts;i++){
      lowerbound[i] = param[i-1]+ (param[i]-param[i-1])/2)+ resabs;
      upperbound[i] =param[i+1]-((param[i+1]-param[1])/2)- resabs;
        }
      }
```

10.2 Geometric Modeller Procedures

10.2.1 Creating Geometry

Create Curve

The Create Curve procedure take an array of data points, two unit vectors for start and end directions and a tolerance resolution. The procedure returns a curve of type bs3_curve inherited from ACIS that is reparameterised from 0 to 1.

```
      void oli_create_curve (    int nb_points,
                                             position positions[],
                                             unit_vector start_dir,
                                             unit_vector end_dir,
                                             double fitol,
                                             double &actual_tol,
                                             bs3_curve &bs3_label
```

```
                                                ) {


        bs3_label = bs3_curve_interp (
                                                                nb_points,
                                                                positions,
                                                                start_dir,
                                                                end_dir,
                                                                fitol,
                                                                actual_tol
                                                                );

        double start = 0;           // start parameter
        double end = 1;                     // end parameter

        bs3_curve_reparam (
                                                        start,
                                                        end,
                                                        bs3_label
                                                        );
}
```

Interpolate Knots

The procedure takes an array of points to interpolate with an array of corresponding knot values and tangent directions at the start and end curve as input. It returns an ACIS curve of type bs3_curve.

```
void oli_interpolate_knots( int nb_points,
                                                position positions[],
                                                double knots[],
                                                vector start_dir,
                                                vector end_dir,
                                                bs3_curve &bs3_label
                                                ) {

        bs3_label = bs3_curve_interp_knots (
                                        nb_points,
                                        positions,

                                        knots,
                                        start_dir,
                                        end_dir
                                        );

        double start = 0;            // start parameter
        double end = 1;                          // end parameter

        bs3_curve_reparam (
                                                         start,
                                                         end,
                                                         bs3_label
                                                         );
    }
```

10.2.2 Discrete Curve Interrogation

The discrete procedures for curve optimisation are the procedures that are used for spline representation curves. They are called discrete because they interrogate the geometry at a given parameter value.

Curvature Analysis

Given a number of points to sample, the curvature analysis procedure returns an array vectors on the curve that point in the direction of the normal of the curve and are curvature length.

void oli_curvature_analysis(

```
                                          int eval_pts,
                                          vector curvature[],
                                          position point[],
                                          bs3_curve &bs3_label
                                                         ){

        double param_value = 0;
        //evaluate curvature of 'myCurve' at a given parameter value
        for ( int param_index = 0; param_index < eval_pts; param_index ++ )
                {
                //returns position coordinates at a given parameter value
                point [param_index] = bs3_curve_position (
                                                param_value,
                                                bs3_label
                                                );

                // curvature evaluation at same parameter value
                curvature [param_index] = bs3_curve_curvature (
                                        param_value,
                                        bs3_label
                                        );

                //increment parameter value
                param_value = param_value + 1/double (eval_pts - 1) ;
    }
}
```

Curve Offset

Curve Offset procedure take an array of points an vectors an returns a array of points that are the transformation of the initial points by the vectors. This procedure is used for making the compound fin properties of a curve after interrogation. For better visuals, a scale factor multiplies the magnitude of the vectors.

```
void oli_curve_offset(
                int eval_pts,
                double scale_factor,
                position point [],
                vector analysis_vect [],
                position new_point []
                ){

    for (int param_index=0; param_index<eval_pts; param_index++){
                new_point [param_index] =
                point [param_index] += (analysis_vect [param_index]*scale_factor);
    }
}
```

Position Distance

The Position Distance procedure computes the distance between two curve at a given parameter value. This procedure is used in the optimisation process for evaluating a solution curve against a reference curve. The procedure returns an average of the sampled points distance as well as averages in x,y,z directions.

```
void oli_position_distance(
                                        int eval_pts,
                                        position point_ref [],
                                        position point_var [],
                                        double &average_distance,
                                        double &average_distance_X,
                                        double &average_distance_Y,
                                        double &average_distance_Z,
                                        double scale_factor
                                        ){


  for (int i=0; i<eval_pts; i++){
          average_distance =
          average_distance +
          fabs(((point_ref[i] - point_var[i]).len())*scale_factor);

          average_distance_X =
          average_distance_X +
          fabs((point_ref[i].x() - point_var[i].x())*scale_factor);

          average_distance_Y =
          average_distance_Y +
          fabs((point_ref[i].y() - point_var[i].y())*scale_factor);

          average_distance_Z =
          average_distance_Z +
          fabs((point_ref[i].z() - point_var[i].z())*scale_factor);
  }
  average_distance = average_distance / eval_pts;
  average_distance_X = average_distance_X / eval_pts;
  average_distance_Y = average_distance_Y / eval_pts;
  average_distance_Z = average_distance_Z / eval_pts;
}
```

10.2.3 Function Representation Procedures

Edge to Law

Assuming that the starting point of the optimisation is an EDGE enclosing a general space curve created by standard means of interpolation, it is possible to generate a function representation of the EDGE. First, the underlying curve (geometry definition) of an edge (topology definition) is obtained. The curve object is converted to a LAW object via some data wrapper. The implementation is enclosed in a function referenced ‘oli_edge_to_law().

```
void oli_edge_to_law(EDGE* edge_label,
                     curve_law*& my_curve_law){
double start = edge_label->start_param();
double end = edge_label->end_param();
// Get a copy of the curve
curve *my_curve = edge_label->geometry()->trans_curve();
// Create a curve_law_data wrapper.
curve_law_data *my_c_law_data = new curve_law_data(*my_curve, start, end);
// Create a curve law
my_curve_law = new curve_law(my_c_law_data);
};
```

Law to Edge

This function is the invert of oli_edge_to_law(), it takes a LAW function and returns the corresponding EDGE.

```
void oli_law_to_edge(
                                        law* my_curve_law,
                                        double start,
                                        double end,
                                        EDGE*& edge_label
                                        ){

int law_number =0; // for future use
law** other_laws =NULL;
CK_OK(api_edge_law (
                              my_curve_law,
                              start,
                              end,
                              edge_label,
                              law_number,
                              other_laws))
}
```

Natural Representation of Curves

The natural representation of curves introduced in paragraph 0 could not be implemented with ACIS because of the absence of integration capability on LAWs. However a virtual function is documented here. This function takes curvature and torsion LAWs, a starting point (for positioning in coordinate space) as input and returns a curve LAW as output.

```
Oli_curve_natural_rep(
                    law* curvature_law,
                    law* torsion_law,
                    position start_point,
                    law* curve_law)
```

10.2.4 Continuous LAW Interrogation

Frénet Frame

At this stage a function representation of the curve is available, many operations can be carried out with the help of the LAW class methods and functions. An implementation of The Frénet-Serret algorithm in paragraph 6.1.4 is presented below. The outcome of the function is four continuous functions including curve direction, curvature vector, curvature, torsion vector and torsion.

```
void oli_Frenet_frame_law(                   law * position_vector,
                                    law *&tangent,
                                    law *&curvature_vector,
                                    law *&curvature,
                                    law *&torsion_vector,
                                    law *&torsion){

// get the first derivative
int which =0;
law *first_deriv = position_vector-> derivative ( which);
//the tangent nomalised vector is:
tangent = new norm_law(first_deriv);

// ds/du = ||P'(u)||
law* point_five = new constant_law(0.5);
law* ds_du = new dot_law (first_deriv,first_deriv);
ds_du = new exponent_law(ds_du, point_five);

// dT / ds = kN
law* dT_ds = tangent->derivative (which);

//dT/du
law* dT_du = new division_law (dT_ds, ds_du);
curvature_vector = dT_du;
law* normal = new norm_law (dT_du);
oli_law_length(dT_du, curvature);

// dN/ds = tB - kT
law* dN_ds = normal->derivative(which);
law* dN_du = new division_law(dN_ds, ds_du);
law* kT = new times_law(curvature, tangent);
law* tB = new plus_law(dN_du, kT);
torsion_vector = tB;
law* bi_normal = new norm_law(tB);
oli_law_length(tB, torsion);

// dB/ds = -tN
law* dB_ds = bi_normal->derivative(which);
law* dB_du = new division_law(dB_ds, ds_du);
law* torsion_check = NULL;
oli_law_length(dB_du, torsion_check);
logical Error = torsion==(torsion_check);
torsion_check = new negate_law(torsion_check);
if (Error = false ){cout << "Error <Torsion not determined>"<<endl;}
}
```

Vector Length

A function that computes the length of a vector is also created.

```
void oli_law_length(law* in_law, law*& out_law){

law* x_term = new term_law(in_law,1);
law* y_term = new term_law(in_law,2);
law* z_term = new term_law(in_law,3);

law* two = new constant_law(2);
law* x_term_sq = new exponent_law(x_term, two);
law* y_term_sq = new exponent_law(y_term, two);
law* z_term_sq = new exponent_law(z_term, two);

law* plus_1 = new plus_law (x_term_sq, y_term_sq);
law* plus_2 = new plus_law (plus_1, z_term_sq);

law * point_five = new constant_law (0.5);
out_law = new exponent_law (plus_2, point_five);
}
```

Singularities

Also it is desirable to implement some mechanism that checks if the derivative of the function actually exists. This ACIS LAW class method is used to specify where in the given law there might be discontinuities. The array notes where the discontinuity occurred. The type indicates 0 if there is a discontinuity, 1 if the discontinuity in the 1st derivative, and any integer n if the discontinuity is in the n-th derivative. -1 means that it is not defined. In presence of a discontinuous function, an error message is retuned.

```
public: virtual int law::singularities (
double** where,                        // where discontinuity exist
int** type,                            // type of discontinuity
double start = -DBL_MAX,               // start
double end = DBL_MAX,                  // end
double** period = NULL                 // period
 ) const;
```

Projection

The projection procedure takes a multidimensional LAW function as input. The X,Y,Z terms are extracted from the LAW for constructing three projections in the X_Y, Y_Z and Z_X planes as outputs.

```
void oli_projection (        law* in_law,
                     law*& x_y_proj,
                     law*& y_z_proj,
                     law*& z_x_proj,
                     ){

law* x_comp = new term_law(in_law, 1);
law* y_comp = new term_law(in_law, 2);
law* z_comp = new term_law(in_law, 3);

law* zero = new constant_law (0);
//law* zero = NULL;

law* x_y_array[3] = {NULL};
x_y_array[0] = x_comp;
x_y_array[1] = y_comp;
x_y_array[2] = zero;

x_y_proj = new vector_law (x_y_array,3);
law* y_z_array[3] = {zero, y_comp, z_comp};
y_z_proj = new vector_law (y_z_array,3);

law* z_x_array[3] = {z_comp, zero,x_comp};
z_x_proj = new vector_law (z_x_array,3);
}
```

Energy Formulation

This function takes a pointer to an entity of type EDGE, gets its LAW representation, evaluate its curvature and numerically integrate the squared curvature from the parameter bounds.

```
double oli_curve_energy(
EDGE *edge_label,
double start, //start parameter for energy eval
            double &energy, //end parameter for energy eval
            double fitol
            ){

law *my_curve_law = NULL;
oli_edge_to_law(edge_label, my_curve_law);
law *my_curve_law = new curve_law(my_c_law_data);

//make curvature law
law *my_curvature_law = new curvature_law(my_c_law_data);

//make it squared
law *energy_law = new times_law(my_curvature_law,my_curvature_law);

int min_level=2;              // optional minimum
int* used_level =NULL;  // optional number of

api_integrate_law ( my_curvature_law,
                                  start,
                                  end,
                                  energy,
                                  fitol,
                                  min_level,
                                  used_level
                                  );
// clean up memory.
my_c_law_data->remove();
my_curve_law->remove();
my_curvature_law->remove();
energy_law->remove();

return energy;
};
```

10.2.5 Parameterisation

Cord Length Parameterisation

This function takes an array of points and returns an array of knot parameters for the cord length parameterisation.

```
void oli_cord_length_parameterisation(
                                    int *Nb_CtrlPpts,
                                    position vertices[], //array of Nb_CtrlPpts points
                                    cord_length_knot[], //array of Nb_CtrlPpts knots
                                    ){

double sum_distance[*&Nb_CtrlPpts-1];

for(int i=0; i<Nb_CtrlPpts-1;i++){
  sum_distance[i] = sum_distance[i-1]+(vertices[i+1]-vertices[i]).len();
  }

doule param[*&Nb_CtrlPpts-2];
for( i=0; i<Nb_CtrlPpts-2;i++){
  param[i] = sum_distance[i]/sum_distance[Nb_CtrlPpts-2];
  }
cord_length_knot[0] = 0;
cord_length_knot[Nb_CtrlPpts-1]=1;

for ( i = 0; i < Nb_CtrlPpts-2; i++ ) {
          cord_length_knot[i+1] = param[i];
  }
}
```

Conclusions and Further Work

This thesis has covered in depth material concerning the three constituting fields of the optimisation framework, which constitutes the foundation of the research. The exhaustive coverage has been necessary to the understanding of the disciplines involved in this research. In depth knowledge of these have permitted looking at ways of combining functionality originating from different sources; thus making more powerful assemblies capable of dealing with a wider range of engineering problems. Conclusions regarding the framework modules are regrouped under their respective headings below.

NURBS Curve Optimisation

NURBS Curve Optimisation is achieved by tweaking the curves' construction constraints to reach the level of quality required by the designers.

The modelling technique, developed in paragraph 7.2, has exposed complementary handles on a curve's shape to the user interface. The handles give interactive control over the curves' internal knot parameter values. The examples provided demonstrate the influence of parameterisation on curves' shape. This curve construction method could also be used as a modelling technique implemented in a CAD environment. However it remains uncertain that designers would benefit from this application without some level of automation added to the optimisation process. The low level of automation, is perhaps the main drawback of this particular application.

This is addressed by another application which drives uses the curve's total energy in an attempt to find an optimum set of knot values which will result in the minimum energy curve satisfying the same interpolation data. The results of experiments have shown that the algorithm fails to produce improved shape. It is questionable whether the GA cannot deal with the problem or the reasoning of this curve optimisation algorithm is flawed. This application is particular in the way that variables are dependent of each other: one variable change influences the balance of parameter distribution of the curve, and this is pushing the

GA to its limit. Furthermore there is no guarantee that the optimisation process is sound. Further work on this algorithm ought to investigate in the formulation of total energy to find a measure that is more representative of the curve's energy. The existing algorithm can also be checked with a more powerful GA that can handle interactive variables.

Other internal constraints are put forward to the user through the graphic interface. These are the curves differential properties at the knots points, which are used as constraints in Hermite interpolation. This modelling feature already exists in some CAD systems but is limited to tangent input. Higher degree deferential constraints input like normal and bi-normal should be considered in further developments.

The research made an attempt to find innovative ideas for graphic manipulation of NURBS, and has succeeded in implementing tools that give complementary handles on the curves' shape by representing internal constraints at the programming interface first and finally graphic interface. However NURBS based algorithms showed immediate limitations because constraints are only applicable discretely at the control points. What goes on in between is function of these, but not always in a predictable manner. NURBS is adequate for interpolating construction points but is not suitable for producing high quality curves.

Intrinsic Curve Optimisation

Another group of curve applications are using a different representation scheme other than NURBS. F-rep is a continuous geometric description of curves by vector functions.

In F-rep curves are defined by three independent functions for the X,Y,Z coordinates. Combinations of two these functions are used in an application, paragraph 8.1, for projecting curve onto planes. The projection technique is useful to visualise on three planes the path of a 3D curve. It enables to see how much curvature and how much torsion there is. Also it is possible to change one of the projections and recreate a 3D curve. However this technique has obvious limitations with helical curves. The projection of one of these would not be helpful.

F-rep functions are differentiable and this has permitted the development of new curve optimisation algorithms. These are based on fundamental differential geometry theorems discovered in the XIXth centuary. Six measures out defined in the Frénet-Serret relations, they define an orthogonal mobile frame that slides along the curve. The Frénet frame is composed of three unit vectors namely tangent, normal and bi-normal. They also define two measures: Curvature is the invert of the radius of curvature oriented by the normal and torsion is the amount of twist in the direction of the bi-normal.

Using the results of the Frénet-Serret relations, the research has developed a variety of tools designed for curve quality inspection. The quality is assessed by visualising the graphic representation of curves intrinsic properties. This is achieved by creating offsets curve for a direct representation of the Frénet frame or by plotting the curvature and torsion magnitude profiles for an indirect representation of the curve's intrinsic properties. These tools have been successfully implemented in the HulaHoops CAD framework and tested on a representative selection of curves feature curves. They proved to be adequate tools for analysing the geometric definition of curves as well as detecting irregularities.

Frénet's proposition for the description of curves' shape is closely related to its intrinsic equations. A proposition was made is this thesis to implement an algorithm which allows the construction of a curve given its curvature and torsion profile. This implies finding a solution to the Frénet-Serret relations. Many approaches are put forwards as feasible algorithms, but in all cases, at least one integration is necessary. The implementation was partially successful because the functionality exposed by the geometric modeller did not allow to accurately implement the theory in a functional application. Under the current functionality of ACIS, there is no possibility to perform symbolic integration. An substitute algorithm to symbolic integration was made up from incremental numerical integration, but could not perform well enough to produce acceptable results. However, given the right tools for the job, these algorithms have potential for success.

The Frénet frame together with curvature and torsion profiles have demonstrated that old pieces of mathematics could be brought back to life with the help of F-rep geometric algebra. The Function Representation paradigm give a more generic and more exact definition of curves than NURBS, thus it should be systematically incorporated in geometric modellers along side with NURBS.

Optimisation

Other conclusions regard optimisation. The primary aim of this research was to demonstrate the possible integration of optimisation with CAD. Sadly little effort was spent on developing an optimisation toolbox with suitable interaction mechanisms necessary for constituting a so-called optimisation framework. Perhaps the research in the field is still at its infancy and the focus is still on developing new algorithms rather than enhancing its interactivity.

The results drawn from optimisation could be examined by researchers in the field to orientate their activities on the representation of constraints. GAs must provide mechanisms to deal with constraints or the algorithm is at risk of producing non-sensical or corrupted geometry. A brief intrusion in the representation of constraints was made in creating functions for bounding variables coming from NURBS data point coordinates, [paragraph 8.2.1] and parameter values, [paragraph 0]. This functionality fulfilled the immediate needs encountered in this research, but more effort should go in this area of work.

Advance in the convergence of GAs was also proposed in this thesis. Paragraph 8.2.1 demonstrates the benefits of fitness function boosting to improve their speed of convergence. It was also found necessary to proceed to a fitness function test to validate the formulation of the evaluation function. This was achieved by creating loop algorithms isolating one variable at the time [paragraph 8.2.1]. Another observation is the absence in the Toolbox of indicator on the level of achievement of the algorithm. This could take the form of a measure of speed of convergence evaluated by the slope of the Paretto front.

The experiments reported in this thesis show that evolutionary computing is incapable of adapting to real life engineering problems. It is inaccurate and shameless to produce non-sense solutions, for example a curve that double backs on itself is a solution, but not a valid solution. This problem could be resolved by constraining the variables and as it stands, variable constraint is limited to the bounding of the upper and lower values. Something in that direction was proposed in paragraph where knot values are constrained so that they never overlap and cause a system failure. The inaccuracy side effect is caused by the binary coding of variables and is perhaps inherent to the algorithm. *Roy et. al.* [99], [102] have published on real coding in GAs, the results predict a reduction of coding error in an algorithm that takes its variables in decimal format; no implementation was proposed. This leads one to question whether the results given in the papers cited above have been adequately checked.

Geometric Modelling

The geometric modeller used for the research, namely ACIS, is well suited to application development. ACIS is structured in components regrouped under a wide common API. The documentation is satisfactory for the main API functions but is sometimes too brief on classes and methods.

Its structure in components enclose functionality purposely designed for specialised operations. The LAW component of ACIS which provides the functionality for curve function representation is independent from the NURBS component, but both have the ENTITY class as common parent. It is possible from an ENTITY that is a curve to get to F-rep representations via NURBS: a long route inside the ACIS interface. Also there are question marks over the conversion from F-rep to NURBS. The variety of shapes that can be obtained with F-rep is much greater than with NURBS. This shows while converting the model from the former to the latter that it could take tens of control points to describe a basic shape. It was also noticed that the control points are more dense in the regions of high curvature. It takes more precision and more constraints to define a shape that has high accelerations than a flat section. The conversion of a NURBS curve to F-rep and back to NURBS could improve the distribution of the control points along the curve. This is valuable gain for model digitisation, which is made at regular intervals, regardless of the curvature [paragraph 3.2]. Maybe it is the case, but it has not been possible to establish this as a fact.

The LAW component in ACIS was intended for sweeps with twists, nowhere is mentioned functionality designed for the specific purpose of curve optimisation. In the light of the results achieved by this research, it could be envisaged that the functionality requirements documented in this thesis is met by the LAW component in the future. In most cases it has been possible to create the functionality out of what was available, but in one instance the experimentations came to halt because it was not possible to proceed to symbolic integration of functions. The solutions to this could be placing a request at Spatial's desk to provide this functionality or finding a third party module that could do this; MATHLAB is under consideration. Some of the functionality needed is present in SVLIS [17] but that has the drawback of being CSG, so there is still no solution available off the shelf.

The philosophy of ACIS is compatible with the idea of a generic interface for the CAD environment. Many of the features making the Oli generic interface, deployed in paragraph 9.5.3 are available from the ACIS API. This has facilitated the design of the Oli interface which could use the functionality already existing in ACIS. For example, generic attributes are implemented in ACIS for making associations with external data. There also is the possibility of making API functions supporting API features like state records, roll-backs and error checks. The need for a generic interface still remains strong. Since recent acquisition of Spatial by Dassault Systemes, they propose a connection interface between CATIA and ACIS. The internal details of this product deserve investigating to establish the nature of its API and the perspectives it offers.

Oli Interface

The concepts of flexible optimisation, together with generic interfacing are slowly emerging from the recent restructuring of CAD vendors. This research has demonstrated that with appropriate interfacing of packages that are designed with open doors to the outside world, more systems could integrate frameworks, thus providing engineers with the possibility of using a panel of engineering design tools. This thesis aimed at documenting the requirements placed by optimisation on an API. Some fresh ideas are thrown in the pot, but much work remains untouched. New projects in this area need to come up and push forward some of the concepts elaborated in this thesis.

Framework

It has been necessary to create a new “kit CAD system from scratch in order to integrate flexible optimisation in it. In this instance both parties share the responsibility for this inconvenience. On one hand CAD systems are closed to research applications and on the other hand optimisation has not come yet to this level of research sufficient to support an API.

The new environment is composed of modules that provide functionality to the framework. The boundary line between interfaces modules and applications is a blurred one. All modules play an important part in the picture, but the delegation of responsibilities across the framework is subjective. The division of labour should follow the principles of interfacing. For example, some of the functionality absent from the Flexo toolbox was replaced by functionality implemented in Oli between its top and the bottom interface. In an ideal situation the generic API should remain abstract, that is a barebones data structure, but should not be the host of function implementations, they must be supported by the modules. The aim of interfacing is to create functionality at its simplest form. Data is gradually reduced through a succession of interfaces which provide clients with the least data as possible. Thus, the objective of interfacing is to delegate as much labour as possible to the interface below it. Requirements at the API level as well as at implementation level have been documented and in some instances, an implementation if these requirements have been proposed. From this point it is down to the geometric modelling community and researchers in optimisation to answer to the calls this thesis has made.

Further Work

The optimisation of NURBS construction parameters was carried out using a GA, the outcome of the application developed have shown that the GA used in this research is struggling in some instances: Interactive variable problems, paragraph 7.4 and high number of variables, paragraph 8.2.1. The aim of further work on GAs is to broaden the problem spectrum it can handle. A lot more work is necessary before optimisation can claim to be integrated within any sort of framework. The implementation in the Flexo project has simply pushed back many of its responsibilities on to the Oli interface and applications rather than designing a real optimisation framework. These observations and propositions deserve further study by optimisation people, the results drawn from this thesis can give some indications for the direction of further developments.

It was found that curve optimisation could be performed through reconstruction from prescribed differential properties. These applications were partly successful because of the current limitations of ACIS functionality regarding symbolic integration. This functionality should take ACIS LAW functions as input and return the primitive LAW functions of the input. This would enable application researchers to implement algorithms for the recreation of curves back from their differential properties without the help of GAs.

This thesis has developed examples for the purpose of curve optimisation and most particularly their aesthetic aspects. Application researchers could make direct usage of the algorithms proposed in this thesis for other purposes. For instance curve framing could be useful to the study of fluid mechanics. More generally, F-rep together with differential geometry could find applications in other fields. *Corney* [28] gives a study example of a mechanical cam design. Recently the author has started on the study of a variable geometry car suspension, where the wheel camber varies with the steering angle to overcome the tyre deformation under cornering.

References

[1] ACIS Version 6.3 online ducumentation. Web page available from http://www.spatial.com

[2] Altair Engineering, "OPTISTRUCT" [WWW page]. Altair Engineering Inc., 1999 [cited 22 June 2000]. Available from < http://www.altair.com/>

[3] Aminov, Y., Differential Geometry and Tpology of Curves. Gordon and Breach Science Publishers, 2000.

[4] Anderson, E., Anderson, R., Boman, M., Elmroth, T., Dahlberg, B. & Johansson, B. (1988). Automatic construction of surface with prescribed shape, In: *Computer Aided Design*, Vol. 20, No. 6, 317-324.

[5] Archer, L.B., Systematic Method for designers. In *Development in Design* Methodology (Ed. N. Cross), 1984, pp. 57-82 (John Wiley, London).

[6] Bäck T., et al., 1997, *Handbook of Evolutionary Computation,* Oxford University Press, New York

[7] Bäck, T., Hammel, U. and Schwefel, H.P. (1997). 'Evolutionary computation: comments on the history and current state'. I.E.E.E. trans. on evolutionary computation, vol. 1, no. 1, 3-17.

[8] Barry, P.J. and Goldman, R.N., 1989. What is the natural generalisation of Bézier curves? In Mathematical Methods in Computer Aided Geometric Design. Tom Lyche and Larry L. Shumaker (eds), pp. 71-85.

[9] Bentley, P.J., Wakefield, J.P., Generic Evolutionary Design.

[10] Berchtold, J., Bowyer, A., Bézier surfaces in set theoretic geometric modelling, www.bath.ac.uk/~ensab/.

[11] Bernstein, S., 1912. Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités. Harkov Soobs. Matem ob-va, 13:1-2.

[12] Bézier, P., 1966. Définition numérique des courbes et surfaces I. Automatisme XI, pp. 625-632.

[13] Bézier, P., 1967. Définition numérique des courbes et surfaces II. Automatisme XII, pp. 17-21

[14] Bishop, R., L., There is more than one way to frame a curve. Amer. Math. Monthly 82, 3 (March 1975), 246-251.

[15] Bowyer, A. Davenport, J.H., Milne, P., Padget, J., and Wallis, A.F., 1989. A geometric algebra system. In Woodwork, J. (Ed) Geometric Reasoning Clarendon Press, UK.. pp. 1-28.

[16] Bowyer, A., Cameron, S. A., Jared, G. E. M., Martin, R. R., Middleditch, A. E., Sabin, M. A. & Woodwark, J. R. (1997). Ten questions that arose in designing the Djinn API for solid modelling, In: Proc. 1997 International Conference on Shape Modelling and Applications, Aizu-Wakamatsu, Japan.

[17] Bowyer, A., SvLis, Introduction and User Manual. Information Geometers, www.bath.ac.uk/~ensab/.

[18] Bowyer, A., Woodwark, J., 1993. An Introduction to Computing with Geometry. Information Geometers Ltd.

[19] Bremicker, M., Chirehdast, M., Kikuchi, N., Papalampos, P.Y., !991. Integrated Topology and Shape Optimisation in Structural Design. Mechanics of Structures and Machines, Vol. 19, No. 4, pp. 551-587.

[20] Buchanan, S.A., De Pennington, A., 1993. Constraint Definition System: a computer-algebra based approach to solving geometric-constraint problems. Computer Aided Design, 25(12), pp. 741-750.

[21] Cagan, J., Agogino, A.M., 1991. Dimensional, variable expansion – A formal approach to innovative design. Res. in Engng Des.

[22] Caldecote, V. The design Team in Relation to the individual Designer. The Practice of and Education for Engineering Design. *Proc. Instn. Mech. Engrs,* 1963-4, 178(B), 16,19.

[23] Choi, B.K., Surface Modelling for CAD/CAM, Advances in Industrial Engineering. Elsevier, 1991.

[24] Clausing, D.P., 1994. Total quality development – The development of competitive products. American society of mechanical engineers Press. New York.

[25] Coello, C.A.C. (1999). An updated survey of evolutionary multiobjective optimization techniques: state of the art and future trends. In: 1999 congress on evolutionary computation. I.E.E.E., Washington, D.C. (U.S.A.).

[26] Cohen, E.and O'Dell, C.L., 1989. A data Independent Parameterisation for Spline Approximation. In Mathematical Methods in Computer Aided Geometric Design. Tom Lyche and Larry L. Shumaker (eds), pp. 155-166.

[27] Combescure, E., Comptes Rendus des Séances de l'Académie des Sciences, Tome soixante-quatrieme, janvier-juin 1867.

[28] Corney. J. Lim. T., 2001. 3D Modelling with ACIS. Saxe-coburg publications.

[29] Culley, S.J., Wallace, A.P., 1994. Optimum Design of Assemblies with Standard Components. Proc. Of Adaptive Computing in Engineering Design and Control '94. Plymouth, UK.

[30] Dankwort, C.W., Podehl, G., 1998. A new aesthetic design workflow – Results from the European project FIORES. Dagstuhl Seminar 98461: CAD Tools and Algorithms for Product Design. International Conference and Research Centre for Computer Science, Schloss Dagsthul, Warden, Nov. 1998.

[31] David, B. T., 1987, “Multi-Expert Systems for CAD,” Intelligent CAD Systems I: Theoreticaland Methodological Aspects, Springer –Verlag, Noordwijkerhout, the Netherlands, pp. 57-67.

[32] de Boor, C., 1972. On calculating with B-splines. Journal Approx. Theory 6, pp.19-42.

[33] De Casteljau, P., 1959. Outillages méthodes Calacul. Technical report, A. Citroën, Paris.

[34] Deb, K. (1999). Evolutionary algorithms for multi-criterion optimization in engineering design. In: Miettinen, K., Makela, M.M., Neittaanmaki, P. and Periaux, J. (eds.). Evolutionary algorithms in engineering and computer science. John Wiley & Sons Ltd., U.K.

[35] Deb, K. (1999A). Evolutionary algorithms for multi-criterion optimization in engineering design. In: Miettinen, K., Makela, M.M., Neittaanmaki, P. and Periaux, J. (eds.). *Evolutionary algorithms in engineering and computer science.* John Wiley & Sons Ltd., U.K.

[36] Deb, K. (1999B). ‘Multi-objective genetic algorithms: problem difficulties and construction of test problems’. *Evolutionary computation,* vol. 7, no. 3, 205-230.

[37] Deb, K., Optimisation for Engineering Design, Algorithms and Examples. Prentice Hall of India, New Delhi, 1995.

[38] Djinn: A geometric interface for solid modelling. Specification and Report. The Geometric Modelling Society. Information Geometers Ltd, 2000.

[39] Edelsbrunner,H, Algorithms in Combinatorial Geometry. Springer-Verlag,1987.

[40] Einar, H., Skappel, E., “FORMELA, 1973. A general design and production data system for sculpted products,” *in ComputerAided Design,* Vol. 5, No. 2, pp. 68-76

[41] Eisenhart, L. P., A Treatise on the Differential Geometry of Curves and Surfaces. Ginn and Co.

[42] Evbuomwan, N. F. O., Sivaloganathan, S., Jebb, A. A survey of design philosaphies, models, methods and systems. *Pro. Of the institution of mechanical engineers.* 1996, pp. 301-320 B4.

[43] Farin, G., 2002. A history of curves surfaces in CAD. In Farin, G., Kim, M.S. and Hoscheck, J. (eds), Handbook of 3D modelling and graphics, pp. 1-22. Elsevier.

[44] Farin, G., Curves and Surfaces for CAGD, a practical guide. Fifth Edition. Morgan Kaufmanm Publishers, 2002.

[45] Farin, G., NURB Curves and Surfaces, from Projective Geometry to Practical Use. A K Peters, 1995.

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

[47] Feilden, G. B. R., et al. Engineering Design (The Feilden’s Report’), 1963 (Her Majesty’s Stationary Office London)

[48] Ferguson, D., Frank, P., Jones, K., 1988. “Surface shape control using constrained optimisation on the B-spline representation,” *in Computer Aided Geometric Design,* Vol. 5, No. 2, pp. 87-103,

[49] Ferguson, J.C., 1964. Multivariable curve interpolation. Journal ACM II/2, pp.221-228.

[50] Finkelstein, L. and Finkelstein, A. C. W., Review of Design Methodology. IEE Proc., June 1983, 130 (Part A, No 4).

[51] Forrest, A.R., 1972. Interactive Interpolation and Approximation of Bézier polynomials. Computer Journal, Vol.15 No.1, pp. 71-79.

[52] Forrest, A.R., 1972. Mathematical Principles for Curve and Surface Representation. In Curved Surfaces in Engineering, computer methods for design and manufacture, 15th-17th March 1972 Churchill College, Cambridge.

[53] Gibson, C.G., Elementary Geometry of Differentiable Curves, an undergraduate introduction. Cambridge University Press 2001.

[54] Goldberg, D.E. (1989). *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley Publishing Company, Massachusetts (U.S.A.).

[55] Goldberg, D.E., 1989, *Genetic Algorithms in Search, Optimisation, and Machine learning,* Addison-Wesley Publishing company, inc., Reading, MA.

[56] Goodman, T.N.T, 1989. Shape Preserving Representations. In Mathematical Methods in Computer Aided Geometric Design. Tom Lyche and Larry L. Shumaker (eds), pp. 333-351.

[57] Gordon, W. and Riesenfeld, R.E., 1974. B-spline curves and surfaces. In Barnhill et al. ’74, pp.95,126.

[58] Hanson, A., J., and Ma, H., Parallel Transport Approach to Curve Framing. Department of Computer Science, Indiana University, 1995.

[59] Higashi, M., Kohji, K., Mosaka, M. “Generation of high quality curve and surface with smoothly varying curvature,” *in EUROGRAPHICS’88 Conference, pp. 79-92*

[60] Holland, J.H., 1975. Adaptation in Natural and Artificial Systems, MIT Press.

[61] Hoscheck, J., 1984. “Detecting regions with undesirable curvature,” *in Computer Aided Geometric Design,* Vol. 1, No2, pp. 183-193.

[62] Hoscheck, J., 1985. Smoothing of curves and surfaces. Computer Aided Geometric Design 2, pp. 97-105, North Holland.

[63] Jared, G. Geometric Modelling Lecture Notes. Cranfield University.

[64] Jared, G., Roy, R., Grau, J. and Buchannan, T. (1998). ‘Flexible optimisation within the CAD/CAM environment’. *Proceedings of the C.I.R.P. international seminar on intelligent computation in manufacturing engineering*. Capri (Italy), 1-3 July, 503-508.

[65] Jones, A. K., 1994. Curvature Integration through Constrained Optimisation. In Sapidis, N., (eds), Designing Fair Curves and Surfaces. Shape Quality in Geometric Modelling and Computer Aided Design. SIAM conference, 1994.

[66] Juster, N..P., 1985. The design process and methodologies. Technical report, University of Leeds, UK.

[67] Kaufman, E., Klass R., 1988. "Smoothing surfacesusing reflection lines for families of splines," *in Computer Aided Design,* Vol. 20, No. 2, pp. 73-78.

[68] Laakko, T., Mäntylä, M., 1993. Feature modelling by incremental feature Recognition. Computer Aided Design, 25(8), pp. 479,492.

[69] Linden, G., Westberg, S., 1993. "Fairing of Surfaces with Optimisation Techniques using FANGA curves as Quality Criterion," *in Computer Aided Design,* Vol. 25, No. 7, pp. 411-420,

[70] Liu, X., Bregg, D.W. and Fishwick, R.J. (1998). Genetic approach to optimal topology/controller design of adaptive structures. *International Journal for Numerical Methods in Engineering 41,* 815-830.

[71] Luckman, J., An Aproach to the Management of Design. In *Developments in Design Methodology* (Ed. N. Cross), 1984, pp. 83-97 (John Wiley, London).

[72] Medland, A.J., 1986. The computer based design process.Kogan page London.

[73] Moreton, H.P., 1995. Simplified curve and surface interrogation via mathematical packages and graphic libraries and hardware. In Computer Aided Design, Elsevier Science Ltd. vol. 27, No. 7, pp. 523-543.

[74] Mostow, J., 1985. Towards better models of the design process. AI Mag., Spring 1985, pp. 44-57

[75] Mussa, R. Roy, R. and Jared, G. "Surface Optimisation within a CAD/CAM environment using genetic algorithms", *in WSC3, Cyberspace,* 21-30 June 1998

[76] Mussa, R., "Integration of Genetic Algorithm into CATIA for Surface Optimisation", M.SC. Thesis, Cranfield University, 1998.

[77] Mussa, R., Roy, R. and Jared, G. (1998). Surface optimisation within the CAD/CAM environment using genetic algorithms. In: Roy, R., Furuhashi, T. and Chawdhry, P. K. (eds.). *Advances in soft computing.* Springer, London (U.K.).

[78] Neilson, G.M., Foley, T.A., 1989. A survey of Applications of an Affine Invariant Norm. In Mathematical Methods in Computer Aided Geometric Design. Tom Lyche and Larry L. Shumaker (eds), pp. 445-467.

[79] Nissan Technical Centre Europe (NTCE), Cranfield, UK, 1999

[80] Nutbourne, A., Martin, R., Differential Geometry Applied to Curve and Surface Design, Volume 1: Foundations. Ellis Horwood Limited

[81] Pahl, G., Beitz, W., 1971. Engineering Design. Original German Edition, 1st edition 1971; English edition (Ed K. Wallace), 1984 (The Design Council London).

[82] Parasolid version 13.2.152 online ducumentation. Web page available from http://www.parasolid.com

[83] Parmee, I.C., Denham, M.J., 1994. The Integration of Adaptive Search Techniques with Current Engineering Design pratice. Proc. Of Adaptive Computing in Engineering Design and Control '94, Plymouth, Uk.

[84] Piegl, L., Tiller, W., The NURBS Book, Monograph in Visual Communication. Second edition. Springer, 1997.

[85] Porta, C.G., Fröwis, R., and Kleindienst, U. An Introduction to Sculpted Metrology (Holometry). in *Reverse Engineering.* J. Hoschek and W. Dankwort, Verlag Ullstein GMGH, 1996, pp. 109-124

[86] Pottmann, H., 1889. "Visualising curvature discontinuities of free-form surfaces," *in EUROGRAPHICS'89 Conference,* pp. 529-536.

[87] Preparata, F.P. and Shamos, M.I, Computational Geometry, an introduction. Springer-Verlag, 1985.

[88] Raasch, "Structural optimisation and it's influence on the design process," in *Reverse Engineering.* J. Hoschek and W. Dankwort, Verlag Ullstein GMGH, 1996, pp. 109-124

[89] Rogero, J.M., Tiwari, A., Munaux, O., Rubini, P.A., Roy, R. and Jared, G. (2000). Applications of evolutionary algorithms for solving real-life design optimisation problems. In: *6th international conference on parallel problem solving from nature (P.P.S.N. VI) workshop,* Paris (France), 16-20 September.

[90] Roy, R. (1997). *Adaptive search and the preliminary design of gas turbine blade cooling system.* PhD. Dissertation, Engineering Design Centre, University of Plymouth, Plymouth (U.K.).

[91] Roy, R. and Parmee, I.C. (1997). An overview of evolutionary computing for multimodal function optimisation. In: Chawdhry, P.K., Roy, R. and Pant, R.K. (eds.). Soft computing in engineering design and manufacturing. Springer, U.K.

[92] Roy, R., Engineering Design Optimisation. Lecture notes 1, Cranfield University, 2000.

[93] Roy, R., Jared, G., Grau, J. and Buchannan, T. (1998). Adaptive design decision support within CAD environment. In: Sivalogonathan, S. and Shahin, T.M.M. (eds.). *Design reuse.* Professional engineering publishing, 255-262.

[94] Roy, R., Jared, G., Tiwari, A. and Munaux, O. (2000) An Overview of Evolutionary-based Multi-objective, Multi-modal and Constrained Optimisation Techniques. Cranfield, UK: Technical Report Cranfield University, Flexo Report – 2.

[95] Roy, R., Jared, G., Tiwari, A. and Munaux, O. (2000). Design Optimisation: A Survey of Industries. Technical Report Cranfield University, Cranfield, U.K., Flexo Report - 5.

[96] Roy, R., Jared, G., Tiwari, A. and Munaux, O. (2000). Families of design. Cranfield, UK: Technical Report Cranfield University, Flexo Report – 3.

[97] Roy, R., Jared, G., Tiwari, A. and Munaux, O. (2000). Flexo, project scope definition, Cranfield, UK: Technical Report Cranfield University, Flexo Report – 1.

[98] Roy, R., Jared, G., Tiwari, A. and Munaux, O. (2000). Specification of Flexible Optimisation Framework. Technical Report Cranfield University, Cranfield, U.K., Flexo Report - 6.

[99] Roy, R., Jared, G., Tiwari, A., and Munaux, O. (2000). Development of 'Tool Box' for Real-life Engineering Design Optimisatiom. Cranfield, UK: Technical Report Cranfield University, Flexo Report – 8.

[100] Roy, R., Jared, G., Tiwari, A., and Munaux, O. (2000). Industrial Requirements for Flexible Optimisation. Cranfield, UK: Technical Report Cranfield University, Flexo Report – 4.

[101] Roy, R., Parmee, I., Purchase, G. 1996. *"Sensitivity Analysis of Engineering Design Using TAGUCHI's Methodology,"* In: Proceedings of the 1996 ASME Design Engineering Technical Conferences, August 18-22, 1996. Irvine, California.

[102] Roy, R., Tiwari, A., Munaux, O. and Jared. G. (2000). Real-life engineering design optimisation: features and techniques. In: *5th online world conference on soft computing in industrial applications (W.S.C.5).* Cyberspace, 4-18 September.

[103] Roy, R., Tiwari, A., Munaux, O. and Jared. G. (2000C). Real-life engineering design optimisation: features and techniques. In: Martikainen, J. and Tanskanen, J. (eds.). *CDROM Proceedings of the 5th online world conference on soft computing in industrial applications (W.S.C.5) – ISBN 951-22-5205-8*, IEEE, Finland.

[104] Rzevski, G., On the design of a design methodology. In *Design: Science: Method, Proceedings of the 1980 Design Research Society Conference* (Ed. R. Jacques and J. A. Powell), 1981, pp. 6-17 (Westbury House, Guildford Surrey).

[105] Sabin, M., 1989. Open Questions in the Application of Multivariate B-splines. In Mathematical Methods in Computer Aided Geometric Design. Tom Lyche and Larry L. Shumaker (eds), pp. 529-537.

[106] Sapidis, N., (eds), Designing Fair Curves and Surfaces. Shape Quality in Geometric Modelling and Computer Aided Design. SIAM conference, 1994.

[107] Schwefel, H.P. (1995). *Evolution and optimum seeking.* John Wiley & Sons, Inc., New York (U.S.A.).

[108] SDRC Imageware SURFACER, User Manual Software version 10.0, 2000

[109] SDRC Ideas, User Manual Software version 8.x, 2000

[110] Sederberg, T., Parry, S., 1986. "Free-Form deformation of solid geometric models," *in SIGGRAPH'86,* Vol. 20, No. 10, pp. 151-160.

[111] Serrano, D., 1984. Constraints Management in Conceptual Design. Ph.D. Thesis, Massachusetts Intitute of technology, Cambridge USA.

[112] Sinha, S., Seneviratne, P. Part to art. in *Reverse Engineering.* J. Hoschek and W. Dankwort, Verlag Ullstein GMGH, 1996, pp. 8-17.

[113] Sriram, D., et al., 1989. Knowledge based system applications in engineering design: research at MIT. AI Mag., 10(3), 79-96.

[114] Stanley, T.J. and Mudge, T. (1995). A parallel genetic algorithm for multiobjective microprocessor design. *Proceedings of the Sixth International Conference on Genetic Algorithms,* 597-604.

[115] Steuer, R.E. (1986). Multiple criteria optimization: theory, computation, and application. Wiley, New York (U.S.A.).

[116] Szilvasi-Nagy, M., 1997. "Detecting regions with undesirable curvature," *in Computer Aided Geometric Design,* Vol. 14, No. 8, pp. 699-706.

[117] The concise Oxford dictionary. Ninth Edition. Clarendon Press, Oxford.

[118] Thornton, C. *"CADET: Software Support Tool for Design Constraints,"*Cambridge University, Massachusetts, USA.

[119] Watabe, H., Okino, N., 1993. "A study on genetic shape design," in S. Forrest, ed., Proceedings of the fifth International Conference on Genetic Algorithms, Morgan Kaufmann, pp. 445-450.

[120] Weatherburn, C., E., Differential Geometry of Three Dimentions, Vol 1, Cambridge University Press, 1939.

[121] Weinert, K., Mehnen, J., and Prestifilippo, "Optimal surface reconstruction from digitized point data using CI methods," Technical report No. CI-5/97, Dortmund University, Germany

[122] Wolters, H.J., Farin, G., "Geometric Curve Approximation," Computer Aided Design 14 pp. 499-513, 1997.

Appendix 1 Optimisation in Industry

The aim of this chapter is to discuss the current status of design optimisation in industry in order to provide a framework for the present research. The details provided in this document are based on the information collected from designers in industry. This document addresses the following issues related to design optimisation.

> To make a study of design processes in manufacturing industry: This includes the study of the current status of design optimisation in industry, the drivers, methodology and tools for design improvement are also outlined.
>
> To identify the tools for design improvement: This constitutes the identification of the main optimisation algorithms and their industrial use. The factors inhibiting the industrial use of these algorithms are also discussed.
>
> To identify the need for flexible optimisation: Based on industrial requirements, this highlights the functionality sought in the flexible optimisation framework.

1.1 Introduction

The introduction of lean production techniques in manufacturing industry has greatly enhanced the efficiency of its operations. However, the full potential of lean manufacturing could only be attained if attempts are made to obtain best possible results with the given resources. This warrants the introduction of optimisation techniques in the product design cycle. This industry survey, which is detailed in *Roy et. al.* [100], forms the basis of this chapter.

1.2 Objectives

This section addresses the following issues related to design improvement.

To make a study of design processes in the manufacturing industry: This includes the study of current status of design optimisation in industry, the drivers, methodologies and tools for design improvement are also outlined.

To identify the tools for design improvement: This constitutes the identification of main optimisation algorithms and their industrial use. The factors inhibiting the industrial use of these algorithms are also discussed.

To identify the need for flexible optimisation: Based on industrial requirements, this highlights the functionality sought in the flexible optimisation framework.

The document begins with a brief explanation of the methodology adopted for the investigation whose results have been summarised in this document.

1.3 Design improvement in Industry

This section explains the design process in industry in terms of the drivers, methodology, tools and current status of design improvement.

1.3.1 Design Process

Design requirements are obtained on the basis of the information from customers and competitors collected by the marketing department. These requirements are converted into conceptual design followed by detailed design, which in most cases is represented on CAD/CAM systems. A prototype is built for experimental testing on a full-scale model before the design is sent for manufacturing. The feedback from design analysis, prototype testing and manufacturing is manually incorporated in the design requirements, conceptual and detailed designs. In a typical product life cycle, the design goes around these feedback loops several times for optimisation before being manufactured and dispatched. This iterative process contributes significantly to lengthening the design cycle, and depends critically on the skills of the designers.

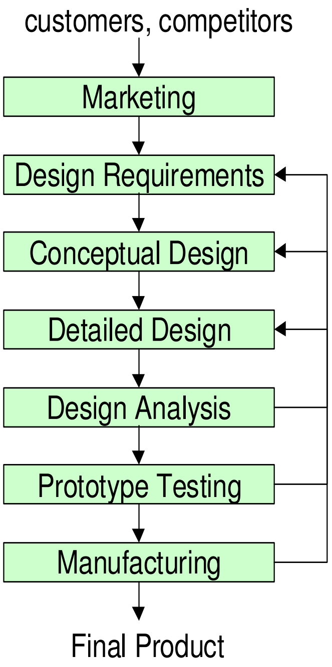


Figure 100: Typical design cycle

1.3.2 Methodologies and Tools

The methodologies adopted for design improvement in many industry sectors are guided by the design tools that are being used. Each of these design tools looks at a particular aspect of design, for example manufacturability, functionality, etc. It was observed that the following design tools are the most popular in industry.

Design for assembly: This is mainly used by companies whose products are composed of several components, e.g., automotive components like Anti-lock Breaking Systems (ABS) and Vehicle Stability Controls (VSC).

Design for manufacture: Mostly the companies that are involved in mass production use design for manufacture. This is particularly true for small to medium sized components, which require automation for production, e.g., anti-vibration and electronic components.

Design for performance: This is observed in production of high performance, high value components, e.g., racing cars and digital photocopiers.

Design for quality: In modern industry, it is observed that most of the companies use this design tool.

Structural optimisation: This is particularly seen in components subjected to stress, e.g., engine mounting brackets, crashworthiness designs, multi- body systems and metal forming.

The above-mentioned design tools, with the exception of structural optimisation, can be effectively applied without complex mathematical analysis. These tools are usually applied externally to the CAD/CAM environment, after the completion of the design definition phase. On the other hand, the aim of structural optimisation, which enjoys a very important place in the automotive industry, is to improve the external shape and structure of components within the CAD/CAM environment simultaneously with the product definition phase. Structural optimisation is a generic term to describe optimisation of geometric entities that make the engineering products.

1.3.3 Current Status

It was observed that design optimisation in industry is an iterative process of creating a model, using a CAD system, carrying out some analyses which give indications of how to improve the design and then modifying the model and repeating the process. This manual process contributes significantly to lengthening the design cycle and depends critically for its success on the skill of the designer. Trial-and-error finds widespread use in industry for improving the design. On the other hand, parametric modelling and optimisation algorithms are rarely used. The industry visits coupled with an Internet-based study enabled the identification of design drivers used in different industry sectors. Table 10 summarises the findings of this study in a 2D matrix depicting some product families against design drivers. Each product family uses one or several design drivers that are identified from specification requirements. The presence of a design driver in a given product family is depicted by ‘1’ in the relevant box of the table.

	Product Families											
Design Drivers	Automotive Wheel	Tyres	Production Car Body Shell	Motor Racing Body Shell	Fan Blade	Air Foil	Ship Hull	Sailing Boats	Machine Tool	Cutting Tool	Hi Fi Speakers	Rubber Components
Sress	1	1			1	1	1	1	1	1		1
Manufacturability	1	1	1									
Weight	1				1	1		1				1
Vibration	1				1	1			1	1	1	1
Aesthetics	1		1									
Fluid Dynamics		1			1		1	1				
Acoustics		1			1				1		1	
Thermodynamics					1				1	1		
Aerodynamics			1	1		1						

Table 10: Optimisation in industry

1.4 Tools for Design Improvement

This section outlines the features of design improvement in industry. It discusses the potential of optimisation algorithms and their industrial use. Finally, it explains the factors that inhibit the use of optimisation algorithms in industry.

1.4.1 Industrial Use of Optimisation Algorithms

Despite the immense potential of optimisation algorithms in handling industrial design optimisation problems, it was observed that they are not used by any of the surveyed companies. Some commercial optimisation packages like OPTISTRUCT from Altair Engineering [2], or DOT from Vanderplaats Research and Development, Inc are proven adequate in many engineering applications but can only be operated outside the CAD/CAM Environment. These capabilities are rarely used in industry since they provide only one type of optimisation algorithms to tackle all types of problems. Although most of these packages are provided externally to the CAD/CAM environment, some of them like the surfacing tools in CATIA, and the FEA and CFD tools in I-DEAS are integrated with the CAD/CAM systems.

The study of literature in the field of real-life optimisation shows that optimisation algorithms have the potential of handling a number of real-life design problems. On the other hand, as revealed in this section, optimisation algorithms are rarely used in industry. This contradiction in the use of optimisation algorithms in industry and research indicates the presence of factors inhibiting the industrial applications of optimisation algorithms. These inhibitors are discussed in detail in the next section.

1.4.2 Inhibitors

Extensive internet-based search and numerous company visits have been used to investigate the use of optimisation algorithm across engineering industries *Roy et. al.* [100]. The survey shows that optimisation in general in not widely used. However some large technology leading make use of optimisation techniques with a limited scope. This is the case of British Aerospace which is involved in research programs with the aim of optimising gas-turbine blade for jet engines, *Rogero* [89]. The research shows that both classical evolutionary optimisation techniques are not popular in industry, the use of these techniques is yet very limited as the complexity of real-life optimisation problems has prevented the industry from exploiting the potential of optimisation algorithms *Roy et. al.* [100]. The industry has, therefore, relied on either trial-and-error or over-simplification for dealing with its optimisation problems independently of the design tools. There are a number of factors inhibiting the industrial use of optimisation algorithms. Some of these factors are outlined below:

The first and the foremost inhibitor to the use of optimisation algorithms in industry is the important role of skills and experience in the design improvement process. In design activities, there are often a large number of parameters influencing the design performance. Most of these parameters are difficult to control, so the designers prefer using their skills and experience with a trial-and-error strategy to guide the search towards optimum solutions. This makes the design process difficult to be encoded in an algorithmic form.

The features of real-life optimisation problems, especially the presence of qualitative issues and lack of prior problem knowledge, also inhibit the successful application of optimisation algorithms to industrial problems.

The lack of robust optimisers that can deal with a variety of engineering problems also prevents the use of optimisation algorithms. This is particularly true for industries that deal with a wide range of designs.

All the currently available optimisation packages are not integrated within CAD/CAM systems making their use cumbersome. The designers need to extract the design parameters from CAD models, input these into the optimisation packages and bring the optimised parameters back to the CAD system. There are a number of difficulties associated with this off-line optimisation, which prevents the designers from using the optimisation algorithms. The extraction and transfer of data between the CAD/CAM system and the optimisation package is manual or semi-automatic. The data transfer often leads to loss of quality and information, which makes the optimisation process inaccurate. This off-line scenario of optimisation also makes designers lose control over the design process. Finally, the inflexible nature of the scenario makes the process iterative and time consuming.

1.5 Need for Flexible Optimisation

In order to attain a holistic view of real-life optimisation, it is essential to develop a flexible optimisation framework that allows the selection of appropriate techniques and parameters for a design optimisation problem. In the recent past, some work carried out in the field of flexible optimisation. *Roy et. al.* [93], *Jared* [64] and *Mussa* [75] specifically addressed the issue of enhancing the optimisation capabilities of existing CAD/CAM systems. However, the above-mentioned work has limited scope in terms of the types of optimisation techniques employed, integration with CAD/CAM systems, and incorporation of designers' intent. Therefore, in contrast to this research, the previous work has adopted a tactical rather than a strategic view of the concept of flexible optimisation.

The framework, in this research, is developed based on the industrial requirements for flexible optimisation. The flexible optimisation wheel, shown in Figure 101 depicts the different combinations possible within the flexible optimisation framework. This framework provides a platform for dealing with various settings of design tools and techniques, geometry representation schemes and optimisation algorithms.

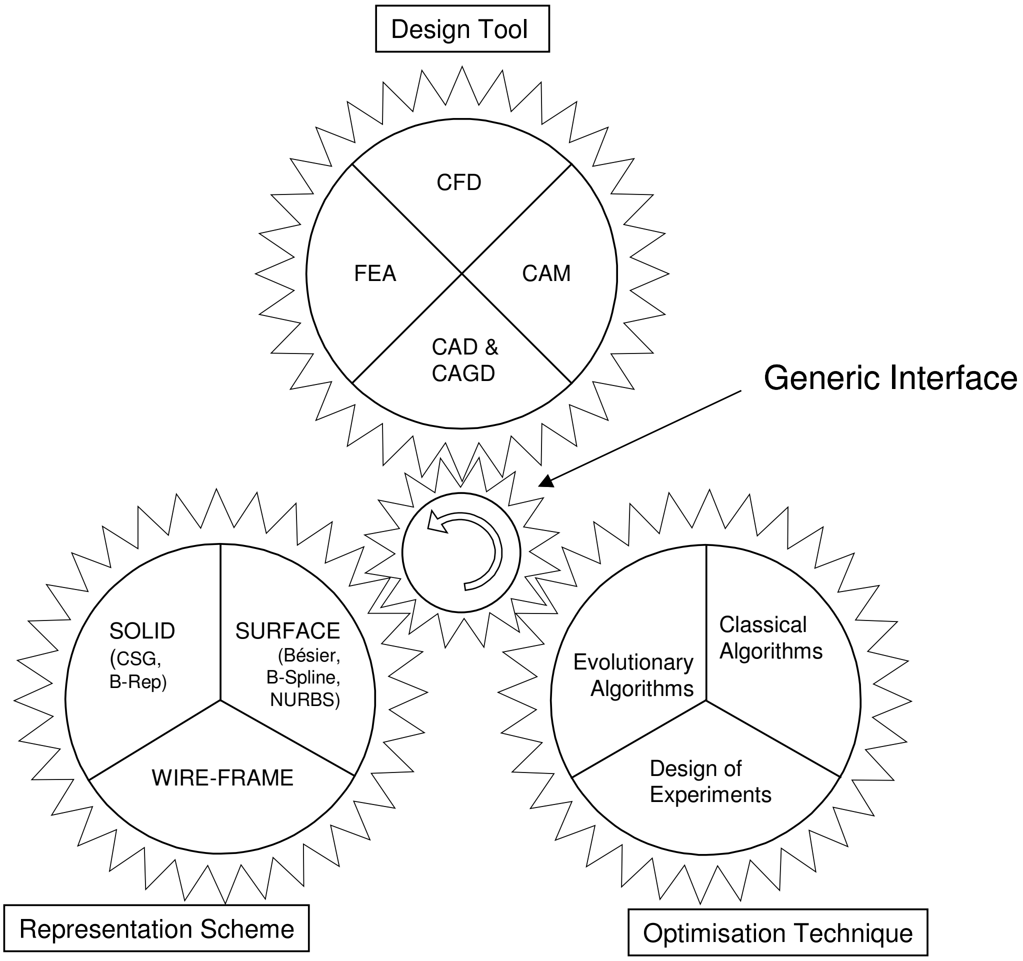


Figure 101: Flexible optimisation wheel

The research involves the development of a toolbox containing engineered multi-objective optimisation algorithms, *Deb* [34], capable of solving a variety of real-life design problems. In order to provide the optimisation capability online, it is essential for the flexible optimisation toolbox to be integrated within the CAD/CAM environment.

Appendix 2 Documentation of Flexible Optimisation 'Toolbox'

This Appendix presents the first version of the documentation for the optimisation 'toolbox'. All the classes and functions used in the 'toolbox' are documented here. These classes and functions have so far been used for the C++ coding of Non-dominated Sorting Genetic Algorithm (NSGA) II and Generalised Regression Genetic Algorithm (GRGA). Both these techniques form part of the optimisation 'toolbox'.

The main features of this code can be summarised as follows:

Binary and real coding.

Tournament and roulette wheel selection for binary coding.

Tournament selection for real coding.

Single point and uniform crossover for binary coding.

Simulated binary crossover for real coding.

Interactive setting of control parameters.

Self-reliant classes having appropriate data members and member functions:

- Array class for integers.
- Arrayr class for reals.
- Individual class for binary solutions.
- Individualr class for real solutions.

Generalised functions in main program for:

- Ranking.
- Fitness assignment.

- Fitness sharing.
- Selection.
- Elitism.

Extensive re-use of classes and functions is possible in future.

The classes and functions used in this code are listed below.

2.1 Array

This class is used to create, manipulate and destroy the binary GA chromosomes. This includes input, output, assignment and comparison of chromosomes. It also gives information regarding the existing chromosomes. The various member functions that form part of this class are as follows:

`Array( int, bool )`
Default constructor.
Input 1: Size of the array.
Input 2: 'True' for an array having only 0's and 'False' for an array having randomly placed 0's and 1's.

`Array( const Array & )`
Copy constructor.
Input: Array that has to be copied to form a new array.

`~Array()`
Destructor.

`static int getArrayCount()`
Returns count of arrays instantiated.
Output: Number of arrays instantiated.

`int getSize() const`
Returns size.
Output: Size of an array.

`const Array &operator=( const Array & )`

Overloaded assignment operator.
Input: Array that has to be copied to the given array.
Output: Modified given array.

```
bool operator==( const Array & ) const
```

Determines if two arrays are equal.
Input: Array that has to be compared with the given array.
Output: 'True' if arrays are equal and 'False' if they are unequal.

```
bool operator!=( const Array & ) const
```

Determines if two arrays are unequal.
Input: Array that has to be compared with the given array.
Output: 'True' if arrays are unequal, otherwise 'False'.

```
int &operator[]( int )
```

Overloaded subscript operator for non-constant arrays.
Input: Subscript value.
Output: Value stored at given array location.

```
const int &operator[]( int ) const
```

Overloaded subscript operator for constant arrays.
Input: Subscript value.
Output: Value stored at given array location.

```
friend ostream &operator<<( ostream &, const Array & )
```

Overloaded output operator.
Input 1: Reference to 'ostream' object, like 'cout'
Input 2: Source array.
Output: Reference to 'ostream' object, like 'cout'.

```
friend istream &operator>>( istream &, Array & )
```

Overloaded input operator.
Input 1: Reference to 'istream' object, like 'cin'.
Input 2: Target array.
Output: Reference to 'istream' object, like 'cin'.
In addition to the above functions that can be accessed by the user, there are also some data members that are hidden from the users. These data members are listed below:

```
int *ptr
```

Pointer to the first element of the array.

```
static int arrayCount
```

Number of Arrays instantiated.

`int size`
Size of the array.

2.2 Arrayr

This class is used to create, manipulate and destroy the real GA chromosomes and other real arrays. This includes input, output, assignment and comparison of chromosomes. It also gives information regarding the existing chromosomes. The various member functions that form part of this class are as follows:

`Arrayr( int, int, bool )`
Default constructor for arrays that do not have bounds on their entries.
Input 1: Size of the array.
Input 2: Accuracy of array entries in terms of number of significant places after the decimal point.
Input 3: 'True' for an array having only 0's and 'False' for an array having random real numbers.

`Arrayr( const Arrayr &, const Arrayr &, int, int )`
Default constructor for arrays that have bounds on their entries.
Input 1: Array storing upper-bound value for each entry of the array that needs to be constructed.
Input 2: Array storing lower-bound value for each entry of the array that needs to be constructed.
Input 3: Size of the array.
Input 4: Accuracy of array entries in terms of number of significant places after the decimal point.

`Arrayr( const Arrayr & )`
Copy constructor.
Input: Array that has to be copied to form a new array.

`~Arrayr()`
Destructor.

`static int getArrayCount()`
Returns count of arrays instantiated.
Output: Number of arrays instantiated.

int getSize() const

Returns size.
Output: Size of the array.

const Arrayr &operator=(const Arrayr &)

Overloaded assignment operator.
Input: Array that has to be copied to the given array.
Output: Modified given array.

bool operator==(const Arrayr &) const

Determines if two arrays are equal.
Input: Array that has to be compared with the given array.
Output: 'True' if arrays are equal and 'False' if they are unequal.

bool operator!=(const Arrayr &) const

Determines if two arrays are unequal.
Input: Array that has to be compared with the given array.
Output: 'True' if arrays are unequal, otherwise 'False'.

double &operator[](int)

Overloaded subscript operator for non-constant arrays.
Input: Subscript value.
Output: Value stored at given array location.

const double &operator[](int) const

Overloaded subscript operator for constant arrays.
Input: Subscript value.
Output: Value stored at given array location.

friend ostream &operator<<(ostream &, const Arrayr &)

Overloaded output operator.
Input 1: Reference to 'ostream' object, like 'cout'
Input 2: Source array.
Output: Reference to 'ostream' object, like 'cout'.

friend istream &operator>>(istream &, Array &)

Overloaded input operator.
Input 1: Reference to 'istream' object, like 'cin'.
Input 2: Target array.
Output: Reference to 'istream' object, like 'cin'.
In addition to the above functions that can be accessed by the user, there are also some data members that are hidden from the users. These data members are listed below:

`double *ptr`
Pointer to the first element of the array.

`static int arrayCount`
Number of Arrays instantiated.

`int size`
Size of the array.

2.3 Individual

This class defines a solution in terms of the binary chromosome, parents, decoding information and objective function values. It can decode the chromosome into its decision variables, calculate number and values of constraints, and evaluate the degree of violation of constraints by a solution. It can also perform single point crossover, uniform crossover, mutation and assignment operations involving problem solutions. This class can also be used to extract information regarding a solution in terms of the chromosome, parents, variable values and objective function values. On the same grounds, it can also be used to modify a solution with respect to the chromosome, parents and variables.

`Individual( int, Array &, Arrayr &, Arrayr & )`
Default constructor for creating random individuals.
Input 1: Length of the chromosome.
Input 2: Array storing the number of chromosome bits corresponding to each problem variable.
Input 3: Array storing upper-bound value for each variable of the problem.
Input 4: Array storing lower-bound value for each variable of the problem.

`Individual( Array, Array, Array )`
Default constructor for creating individuals when the chromosome and parents are specified.
Input 1: Array representing the chromosome.
Input 2: Array representing the first parent.
Input 3: Array representing the second parent.

`Individual( const Individual & )`
Copy constructor.
Input: Individual that has to be copied to create a new individual.

`Arrayr confun() const`

Returns the values of constraints.
Output: Array containing the constraint values.

`void crossover( Individual &, Individual &, Individual &, double, int ) const`

Performs single point crossover operation between the given individual and the individual specified in the parameter list.
Input 1: First individual created by the crossover operation.
Input 2: Second individual created by the crossover operation.
Input 3: Crossover partner of the given individual.
Input 4: Probability of performing crossover operation.
Input 5: Accuracy of crossover probability.

`double error() const`

Evaluates the degree of violation of constraints corresponding to the given individual.
Output: Degree of violation of constraints.

`const Array &getChrom() const`

Returns the chromosome corresponding to an individual.
Output: Array representing the chromosome of the given individual.

`int getChromlen() const`

Returns the chromosome length of the given individual.
Output: Length of the given individual's chromosome.

`static Arrayr getLowerbound()`

Gets the lower-bound value for each problem variable.
Output: Array storing lower-bound value for each variable.

`int getNumcon() const`

Returns the number of constraints.
Output: Number of constraints.

`int getNumfun() const`

Returns the number of objective functions.
Output: Number of objective functions.

`int getNumvar() const`

Returns the number of variables in the problem.
Output: Number of variables.

`const Array &getParent1() const`

Returns the chromosome corresponding to the first parent of the given individual.
Output: Array representing the chromosome of the first parent.

`const Array &getParent2() const`

Returns the chromosome corresponding to the second parent of the given individual.
Output: Array representing the chromosome of the second parent.

`static Arrayr getUpperbound()`

Gets the upper-bound value for each problem variable.
Output: Array storing upper-bound value for each variable.

`static Array getVarlen()`

Returns the number of bits in an individual's chromosome, corresponding to each problem variable.
Output: Array storing the number of chromosome bits corresponding to each variable.

`Arrayr getVarval() const`

Returns the variable values corresponding to the given individual.
Output: Array storing the values of problem variables.

`Individual &mutation( double, int )`

Performs mutation operation on the given individual.
Input 1: Probability of performing mutation operation.
Input 2: Accuracy of mutation probability.
Output: Individual after mutation.

`Arrayr obja() const`

Returns the calculated values of objective functions.
Output: Array containing the objective function values.

`Arrayr objfun() const`

Returns the stored values of objective functions.
Output: Array containing the objective function values.

`const Individual &operator=( const Individual & )`

Overloaded assignment operator.
Input: Individual that has to be copied to the given individual.
Output: Individual after assignment.

`void setChrom( Array & )`

Allows the modification of an individual's chromosome.

Input: Chromosome array that replaces the existing one to form a new chromosome.

`static void setLowerbound( Arrayr & )`
Allows the modification of lower-bounds of variables.
Input: Array storing the updated lower-bound value for each variable of the problem.

`void setParents( Array &, Array & )`
Allows the modification of an individual's parents.
Input 1: Array representing the new first parent.
Input 2: Array representing the new second parent.

`static void setUpperbound( Arrayr & )`
Allows the modification of upper-bounds of variables.
Input: Array storing the updated upper-bound value for each variable of the problem.

`static void setVarlen( Array & )`
Allows the modification of the distribution of chromosome bits allotted for the representation of each variable.
Input: Updated array storing the number of chromosome bits corresponding to each variable.

`void unicross( Individual &, Individual &, Individual &, double, int) const`
Performs uniform crossover operation between the given individual and the individual specified in the parameter list.
Input 1: First individual created by the crossover operation.
Input 2: Second individual created by the crossover operation.
Input 3: Crossover partner of the given individual.
Input 4: Probability of performing crossover operation.
Input 5: Accuracy of crossover probability.
In addition to the above functions that can be assessed by the user, there are also some data members that are hidden from the users. These data members are listed below:

`Array chrom`
Chromosome of the individual.

`static Arrayr lowerbound`
Lower-bounds of problem variables.

`Arrayr obj`
Values of objective functions.

`Array parent1`

First parent chromosome of the individual.

`Array parent2`
Second parent chromosome of the individual.

`static Arrayr upperbound`
Upper-bounds of problem variables.

`static Array varlen`
Number of chromosome bits corresponding to each variable.

2.4 Individualr

This class defines a solution in terms of the real chromosome, parents, precision information, bounds and objective function values. It can calculate the number and values of constraints, and evaluate the degree of violation of constraints by a solution. It can also perform simulated binary crossover, mutation and assignment operations involving problem solutions. This class can also be used to extract information regarding a solution in terms of the chromosome, parents, variable values and objective function values. On the same grounds, it can also be used to modify a solution with respect to the chromosome, parents and variables.

`Individualr( int, int )`
Default constructor for creating random individuals when the bounds on variables are not specified.
Input 1: Length of the chromosome.
Input 2: Accuracy of variable values in terms of number of significant places after the decimal point.

`Individualr( Arrayr &, Arrayr &, int, int )`
Default constructor for creating random individuals when the bounds on variables are specified.
Input 1: Array storing upper-bound value for each variable of the problem.
Input 2: Array storing lower-bound value for each variable of the problem.
Input 3: Length of the chromosome.
Input 4: Accuracy of variable values in terms of number of significant places after the decimal point.

`Individualr( Arrayr &, Arrayr &, Arrayr & )`
Default constructor for creating individuals when the chromosome and parents are specified.
Input 1: Array representing the chromosome.
Input 2: Array representing the first parent.
Input 3: Array representing the second parent.

`Individualr( const Individualr & )`
Copy constructor.
Input: Individual that has to be copied to create a new individual.

`Arrayr confun() const`
Returns the values of constraints.
Output: Array containing the constraint values.

`void crossover( Individualr &, Individualr &, Individualr &, double, int, double ) const`
Performs simulated binary crossover operation between the given individual and the individual specified in the parameter list.
Input 1: First individual created by the crossover operation.
Input 2: Second individual created by the crossover operation.
Input 3: Crossover partner of the given individual.
Input 4: Probability of performing crossover operation.
Input 5: Accuracy of crossover probability.
Input 6: Distribution index for crossover.

`double error() const`
Evaluates the degree of violation of constraints corresponding to the given individual.
Output: Degree of violation of constraints.

`const Arrayr &getChrom() const`
Returns the chromosome corresponding to an individual.
Output: Array representing the chromosome of the given individual.

`int getChromlen() const`
Returns the chromosome length of the given individual.
Output: Length of the given individual's chromosome.

`static Arrayr getLowerbound()`
Gets the lower-bound value for each problem variable.
Output: Array storing lower-bound value for each variable.

`int getNumcon() const`

Returns the number of constraints.
Output: Number of constraints.

`int getNumfun() const`
Returns the number of objective functions.
Output: Number of objective functions.

`int getNumvar() const`
Returns the number of variables in the problem.
Output: Number of variables.

`const Arrayr &getParent1() const`
Returns the chromosome corresponding to the first parent of the given individual.
Output: Array representing the chromosome of the first parent.

`const Arrayr &getParent2() const`
Returns the chromosome corresponding to the second parent of the given individual.
Output: Array representing the chromosome of the second parent.

`static Arrayr getUpperbound()`
Gets the upper-bound value for each problem variable.
Output: Array storing upper-bound value for each variable.
`Arrayr getVarval() const`
Returns the variable values corresponding to the given individual.
Output: Array storing the values of problem variables.

`Individualr &mutation( double, int, double )`
Performs mutation operation on the given individual.
Input 1: Probability of performing mutation operation.
Input 2: Accuracy of mutation probability.
Input 3: Distribution index for mutation.
Output: Individual after mutation.

`Arrayr obja() const`
Returns the calculated values of objective functions.
Output: Array containing the objective function values.

`Arrayr objfun() const`
Returns the stored values of objective functions.
Output: Array containing the objective function values.

`const Individualr &operator=( const Individualr & )`

Overloaded assignment operator.
Input: Individual that has to be copied to the given individual.
Output: Individual after assignment.

`void setChrom( Arrayr & )`
Allows the modification of an individual's chromosome.
Input: Chromosome array that replaces the existing one to form a new chromosome.

`static void setLowerbound( Arrayr & )`
Allows the modification of lower-bounds of variables.
Input: Array storing the updated lower-bound value for each variable of the problem.

`void setParents( Arrayr &, Arrayr & )`
Allows the modification of an individual's parents.
Input 1: Array representing the new first parent.
Input 2: Array representing the new second parent.

`static void setUpperbound( Arrayr & )`
Allows the modification of upper-bounds of variables.
Input: Array storing the updated upper-bound value for each variable of the problem.
In addition to the above functions that can be assessed by the user, there are also some data members that are hidden from the users. These data members are listed below:

`Arrayr chrom`
Chromosome of the individual.

`static Arrayr lowerbound`
Lower-bounds of problem variables.

`Static int maxdecpt`
Maximum accuracy of variable values in terms of number of significant places after the decimal point.
`Arrayr obj`
Values of objective functions.

`Arrayr parent1`
First parent chromosome of the individual.

`Arrayr parent2`
Second parent chromosome of the individual..

Appendix 3 Geometric Modelling

3.1 Boundary Representation complements

3.1.1 Topological Hierarchy

The boundary representation (B-rep) of a model is a hierarchical decomposition of the model's topology:

- Body: The highest level of model object, and is composed of lumps.
- Lump: A 1D, 2D, or 3D set of points in space that is disjoint with all other lumps. It is bounded by shells.
- Shell: A set of connected faces and wires, and can bound the outside of a solid or an internal void (hollow).
- Face: A connected portion of a surface bounded by a set of loops.
- Loop: A connected series of coedges. Generally, loops are closed, having no actual start or end point.
- Wire: A connected series of coedges that are not attached to a face.
- Coedge: Represents the use of an edge by a face.
- Edge: A curve bounded by vertices.

Bodies

Bodies are the highest-level entities in ACIS solid models. Typically, a body is a single solid or sheet component, such as a washer, a stripped-down engine block, a zero thickness plate, or a cross section. A body can also be made of several disjoint bodies treated as one. Figure 102 shows an example of a body with more than one lump. When a square block (solid lines) is cleared by a cylinder (dashed lines) whose diameter is slightly larger than the length of a side of the block, the block is separated into four lumps. Although the four lumps are not physically joined, they are still treated as a single body in ACIS.

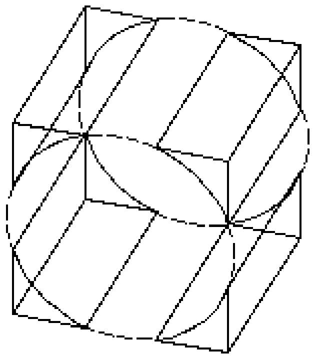


Figure 102: Body with four lumps

Lumps

A lump represents a bounded, connected region in space. A lump is an entire connected set of points, whether the set is 3D, 2D, 1D, or a combination of dimensions. Thus, a solid block with a dangling outside face is one lump, as is a solid block with an internal cavity. Two disconnected sheets are represented as two lumps. A body contains zero or more lumps, each of which represents a set of points that are disjoint from those represented by all other lumps in the body. Figure 103 illustrates a body with two solid lumps. The large block represents one solid lump that completely encloses a void (dotted line). The second lump (small block with solid lines) is completely enclosed in the void.

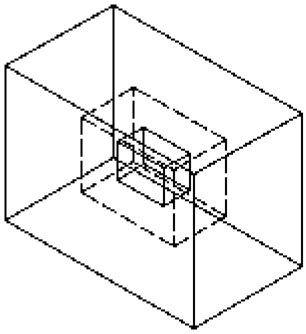


Figure 103: Lumps

Shells

A shell, Figure 104, is an entire connected set of faces and/or wires, including connections through a nonmanifold vertex. Faces are connected together along common edges or at common vertices; wires may be connected to faces at end vertices. A solid block with a dangling sheet is one shell, but a block with a cavity is two shells. A solid block with many embedded faces that are all connected through some path to the exterior faces is one shell, but a solid block with a disconnected "floating" embedded face is two shells (but one lump). The most common type of shell is made up only of complete, finite single-sided faces, two of which meet at every edge, with compatible insides and outsides.

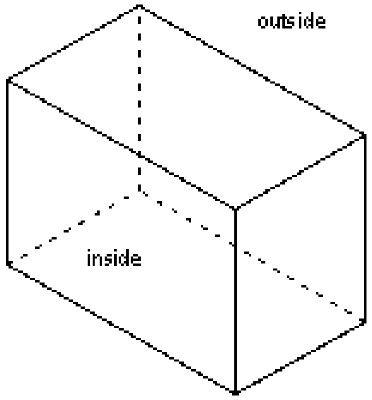


Figure 104: Shells

Faces

A face is a portion of a single geometric surface in space, the two-dimensional analogue of the body. Zero or more loops of edges constitute the boundary of a face. Figure 105 shows a loop of the top face (there are six faces) of a rectangular block.

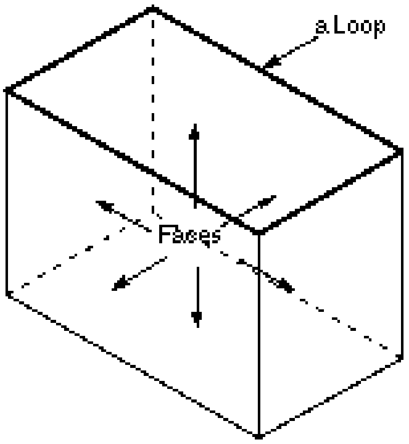


Figure 105: Faces

Loops

A loop represents a connected portion of the boundary of a face. It consists of a set of coedges linked in a doubly linked chain, which may be circular or open-ended. If either end of an open-ended loop is at a finite point, then the face containing the loop is necessarily incomplete. If either end is at infinity, then the face is infinite. The illustration in Figure 106 contains three closed loops. Each loop is the boundary of a complete, finite face. In the actual physical structure, the adjacent parallel lines are coincident.

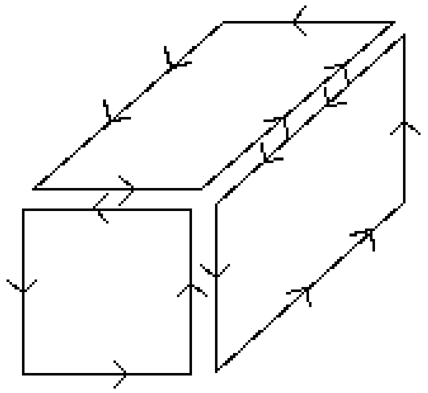


Figure 106: Loops of edges

Edges

An edge is the topology associated with a curve. The direction of an edge can be either the same direction as its underlying curve, or it can be the opposite direction. If it is the same as the curve direction, the edge's sense relative to the underlying curve is forward; otherwise, its sense is reversed. Each edge contains a record of its sense relative to its underlying curve. An edge is bounded by one or more vertices, referring to one vertex at each end.

Coedges

A coedge records the occurrence of an edge in a loop of a face. The introduction of coedges permits edges to occur in one, two or more faces, and so makes possible the modelling of sheets and solids (manifold or not). In a manifold solid body shell, each edge is adjacent to exactly two faces; therefore, the edge has two coedges, each associated with a loop in one of the faces (the two faces can be the same, and even the loops can be the same). In this case, the two coedges always go in opposite directions along the edge. In a nonmanifold body shell, there may be more than two coedges associated with an edge.

Wires

A wire is a connected collection of edges that are not attached to faces and do not enclose any volume. Wires may represent abstract items like profiles, construction lines and centerlines, or idealizations of rod or beam-like objects or internal passages. They are also commonly used to form wire frames to be surfaced to form solid-bounding shells.

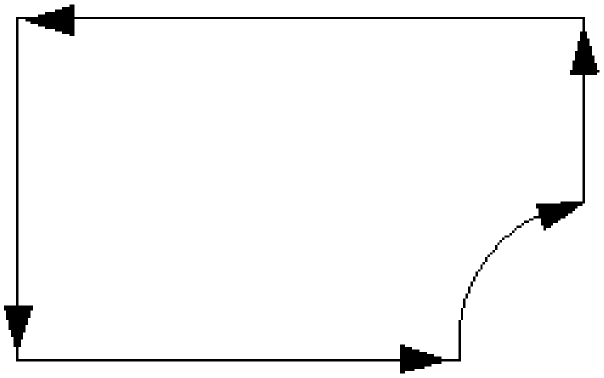


Figure 107: Wire body

A shell may contain a single wire or faces with multiple wires attached to them at vertices. A shell with just a wire is called a wire shell, a lump with only a wire shell is a wire lump, and a body with only wire lumps is a wire body.

Each wire is classified as being exterior representing an infinitesimally thin piece of material, or interior representing an infinitesimally thin passageway within bulk material.

Vertices

Vertex (Figure 108) refers to a point in object space and to the edges that it bounds.

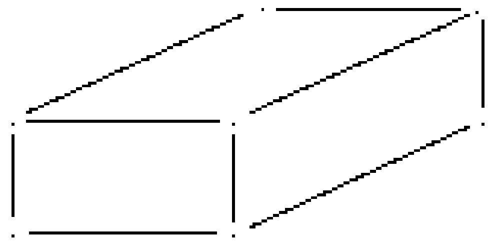


Figure 108: Vertices

3.1.2 Euler-Poincaré formula

More generally, B-Rep complies with the Euler-Poincaré formula. Along with his other endeavours, Leonard Euler worked on body topology relationships. There are many versions in existence of his equations for managing topology. For instance, one of Euler's most famous equations for what he considered to be a valid manifold object was that the number of faces in an object minus the number of edges in that object plus the number of vertices in that object must always equal the number two (2). A more complete formulation of the equation, given below, includes hole loops and genus.

$$V - E + F - H = 2 (M - G)$$

V - Vertices
E - Edges
F - Faces
H - Hole Loops
M - Multiplicity
G - Genus

The Following properties of the Euler-Poincaré formula should be noted.

- Integer values of V,E,F etc
- V,E,F,H,M,G all ≥0
- If V=E=F=H=M=0, then G=M=0
- If M>0 then V≥ and F≥M
- E-P formula defines '6D LATTICE' Euler operators

3.1.3 Graph Theory

A graph is a mathematical abstraction of relationships. Many real-world situations can conveniently be described through a diagram or graph consisting of a set of points (nodes or vertices) together with lines (edges) joining various pairs of these points. This graphical representation helps us understand connectivity relationships and is the basis for graph theory.

Graph diagrams tell whether or not two given points are joined by a line and the manner in which they are joined. There is no unique way of drawing a graph.

One use of graph theory in geometric modelling is to abstract a given model's cells into a graph. Each cell of the geometric model becomes a point of the graph. Points of the graph are connected with lines (or edges) only if the cells of the geometric model are adjacent with faces. Last sentence is not clear at all

Another use of graph theory is to abstract a given model's faces into a graph. Each face of the geometric model becomes a point of the graph. Points of the graph are connected with lines (or edges) only if the faces of the geometric model are adjacent.

In classical B-Rep modelling, adjacency relationships are defined by the well known winged-edge data structure, Figure 111, in which the following data structure are stored for each edge:

- Two vertices defining the edge,
- Two faces F1 and F2 that meet at the edge, and
- Four of its adjacent edges: the "next edge" in a counter-clockwise or clockwise traversal about either F1 or F2.

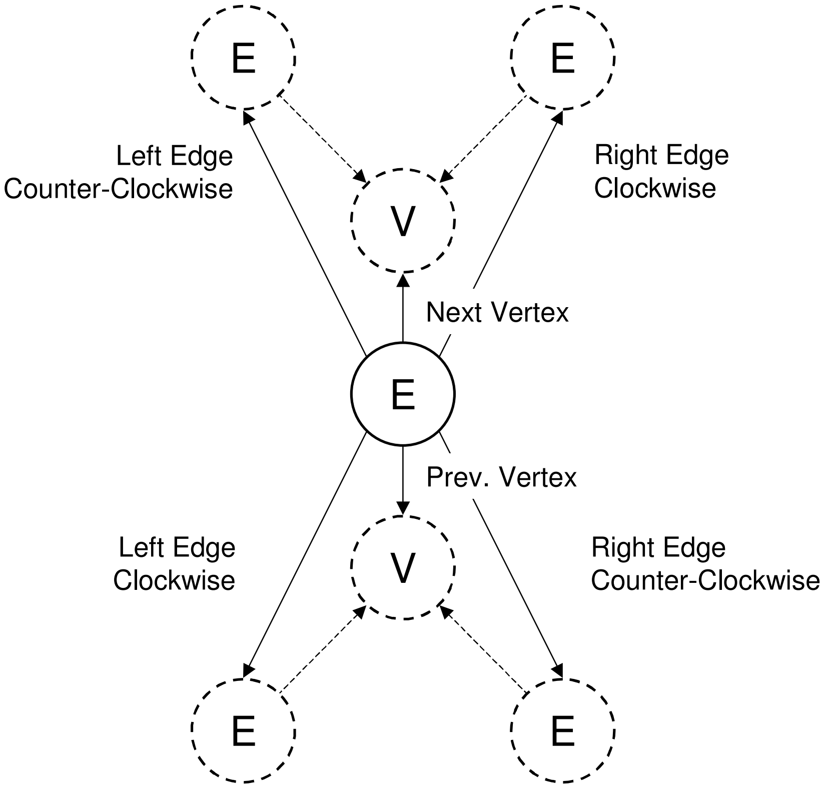


Figure 109: Winged edge data structure

3.2 Boolean Operations

The four regular Boolean operations, union, intersection, difference and symmetric difference, were previously introduced in set-theoretic modellingFigure 18. In this paragraph, some special cases and non-regular Boolean operations are presented:

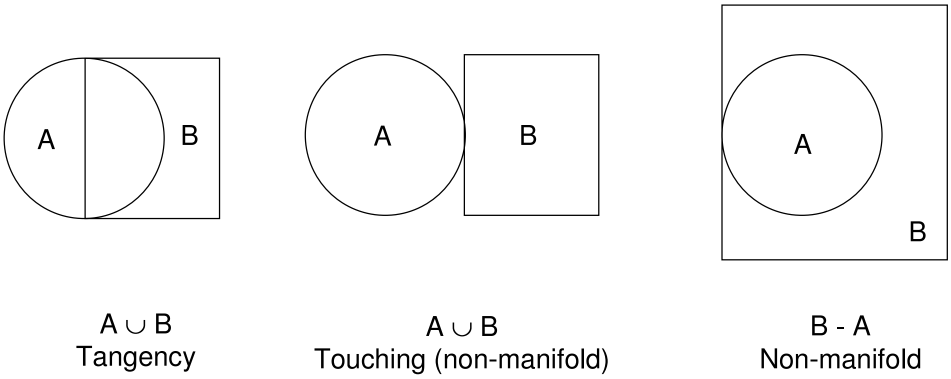


Figure 110: Special cases of Boolean operations

3.3 Sweep and Swings

The Euler-Poincaré formula given earlier in this chapter is also useful in creating solids from profile surfaces. These are sweeps (translational) and swings (rotational).

Figure 111 is an example of Euler operations in sweeping. A solid is obtained from a face with vertices by successively changing the values of M, F, V, E.

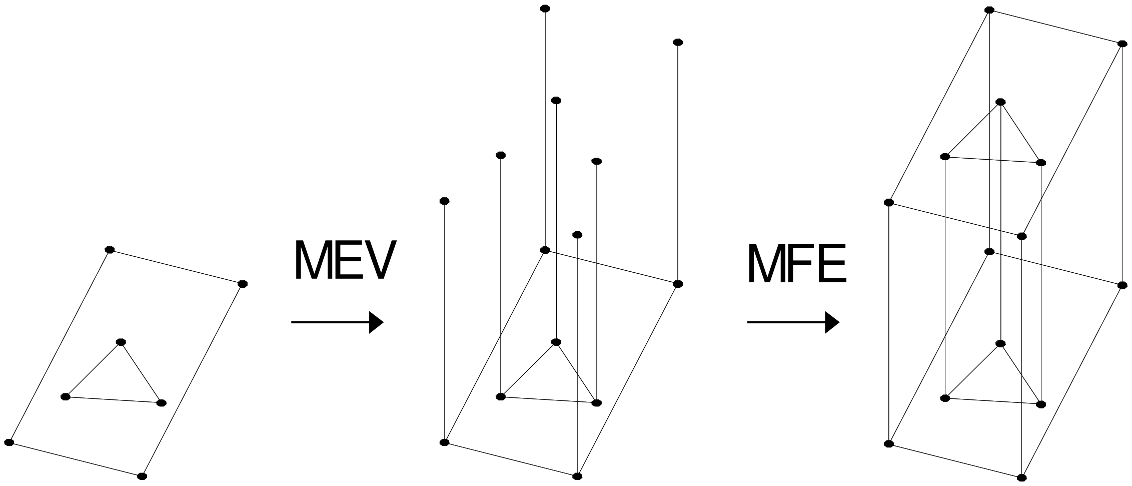


Figure 111: Euler operations in Sweeping

3.4 Tessellated Models

The *Oxford* dictionary [117] gives the following definition to tessellation:

"An arrangement of polygons without gaps or overlapping especially in a repeated pattern"

This paragraph introduces and discuses some tessellation techniques used in geometric modelling. These are Spatial Occupancy (S.O.E.), Octree and Triangulation.

3.4.1 Spatial Occupancy Enumeration (S.O.E)

In spatial occupancy enumeration, the 3D space is divided into identical cubical cells and at a particular resolution, as shown in Figure 112. To avoid dividing the entire space into cubes, the model is first bounded in a box. Each cube coordinates are recorded with a logical one if the cube is filled or zero if it isn't. Its application domain is large but has a number of drawbacks:

- Not an exact representation
- Needs a large amount of storage for reasonable resolution

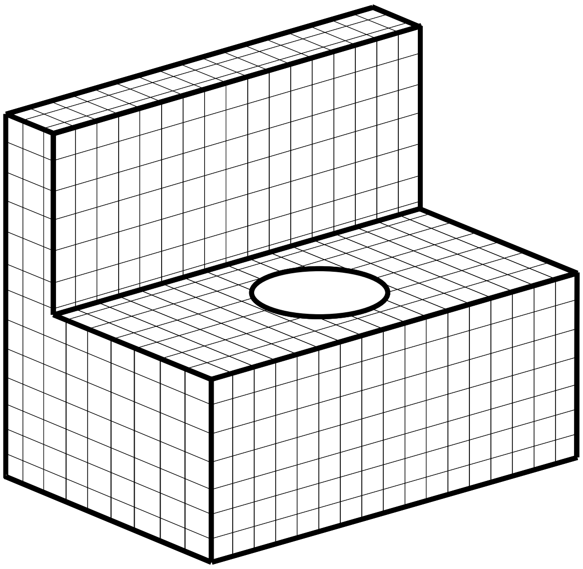


Figure 112: Spatial occupancy model

3.4.2 Octree Decomposition

Similarly to S.O.E, octree models decompose solids in a list of cells, but the cells are not necessarily identical. The octree decomposition method is said to be adaptive because cells are divided unevenly until each cell contains solid or void only. In theory, this recursive division loop can go on forever, unless some resolution limit is set. This method is more efficient than S.O.E and needs less storage space.

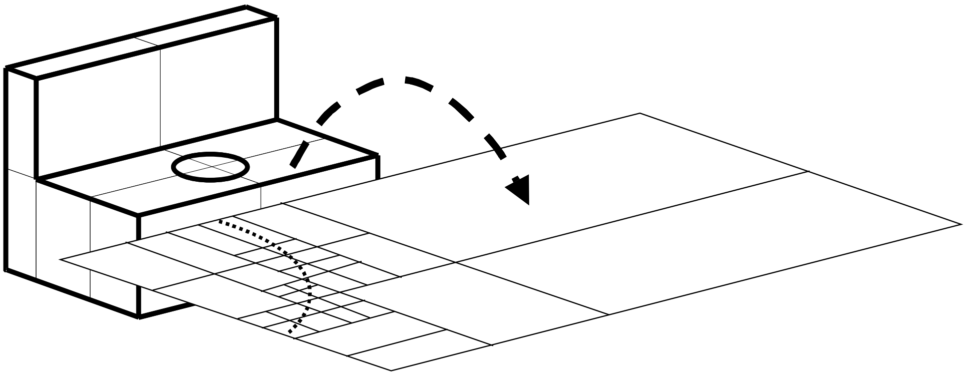


Figure 113: Octree decomposition model

3.4.3 Polyhedral Models

Unlike S.O.E, which is imposed upon the data and octree, which adapts to it, Polyhedral tessellation is derived from the data. The two best-known methods are Dirichlet tessellation (also known as Voronoi Diagrams) and Delaunay triangulations.

Suppose a finite set of scattered data points unevenly distributed (site), as shown in Figure 114. Dirichlet tessellation defines, for each data point, a plane region that is closest to the data point than any other point (bold lines). In fact it expresses the proximity information of a set of objects in space. This tessellation is obtained from intersecting point-pairs equidistant lines (dashed lines). The fine lines joining data points to each other are the Delaunay triangulation. More detailed information on tessellated models is available in [18], [87] and [39].

Also triangulated tessellations are particularly suited to surface reconstruction in reverse engineering and finite element analysis meshing: [78], [105].

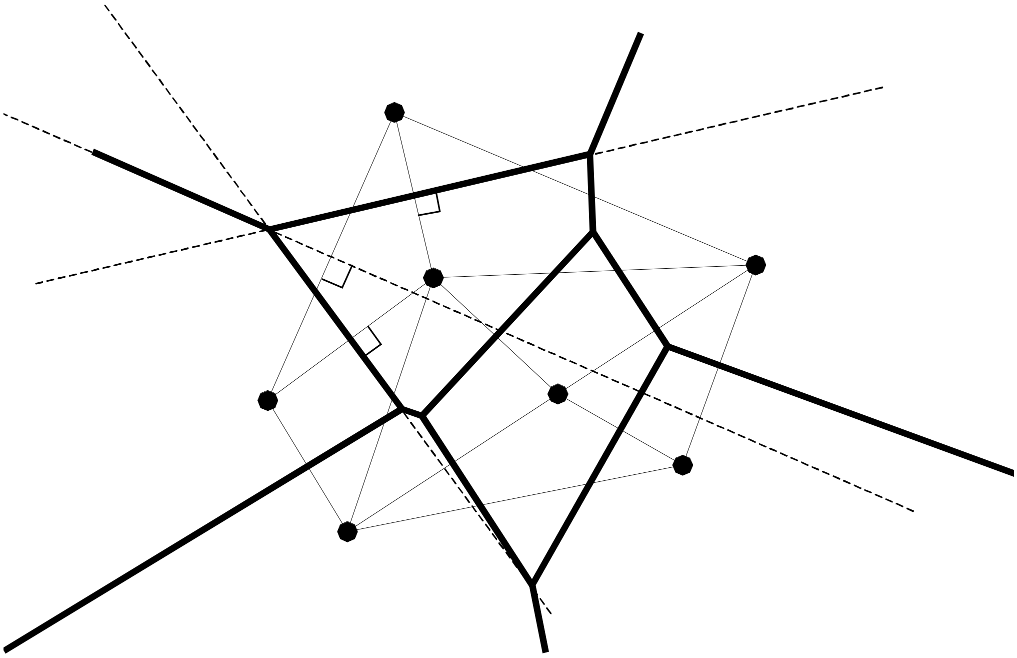


Figure 114: Dirichlet tessellation and Delaunay triangulation

Appendix 4 Vector Algebra

4.1 Vector Algebra.

$A \bullet B = x_a x_b + y_a y_b + z_a z_b$

$A \times B = y_a z_b - z_a y_b,\ x_a z_b - z_a x_b,\ x_a y_b - y_a x_b$

$\|C\| = \sqrt{(C \bullet C)}$

$normC = C / \|C\|$

4.2 Differentiation of a Vector

If a vector *a* is function of a parameter *u*, then its components are also function of *u*:

$$a(u) = a_1(u)x + a_2(u)y + a_3(u)z$$

And if the derivatives of $a_1(u), a_2(u), a_3(u)$ exist, the derivative of *a* is as follows:

$$\frac{d^i a}{du^i} = \frac{d^i a_1}{du^i} x + \frac{d^i a_2}{du^i} y + \frac{d^i a_3}{du^i} z$$

Consequently, with any vectors $a(t), b(t)$ and any scalar function $k(t)$ the following properties are deduced, which will come useful at later stage for the purpose of curve analysis and optimisation:

$$\frac{d}{du}(a+b)=\frac{da}{du}+\frac{db}{du}$$
$$\frac{d}{du}(ka)=\frac{dk}{du}a+k\frac{da}{du}$$
$$\frac{d}{du}(a.b)=\frac{da}{du}.b+a.\frac{db}{du}$$
$$\frac{d}{du}(a\times b)=\frac{da}{du}\times b+a\times\frac{db}{du}$$

N.B: $(a\times b)$ denotes the cross product of *a* with *b*

In addition, if *a(t)* is a unit vector, then *a.a=1*. Thus

$$\frac{d(a.a)}{du}=2a.\frac{da}{du}=0$$

and for a unit vector *a*,

$$a.\frac{da}{du}=0$$

This means that *a* and $\frac{da}{du}$ are perpendicular.

Appendix 5 Implementation of Curve Optimisation Application Using GA

The listing below is the C++ code implementation of the application for curve reconstruction from prescribed curvature using genetic algorithm presented in paragraph 8.2.1.

```
// GA_main.cpp
// created by Olivier Munaux and Ashutosh Tiwari, 06/09/01 Cranfield University
// main driver file for Flexo

#include "wot.h"                    // include Flexo interface header
#include "Translators.h"            // include Translators function library header
#include <math.h>

//****************    Data Initialisation    ******************************

double average_dist = 0;
double average_dist_old = 10000;
double average_dist_X = 0;
double average_dist_Y = 0;
double average_dist_Z = 0;
Individual *oldpop[ ];
Individual *intpop[ ];
Individual *newpop[ ];
position new_points [ ];
int I = 0;
int popsize = 0;
int nogen = 0;
double crossp = 0;
int prcrossp = 0;
double mutp = 0;
int prmutp = 0;
int power_start = 3;
int power_finish = 5;
double power = power_start;
int rank_derivative = 0;
int match_accuracy =0;
position curve_point [ ];
vector vect_deriv [ ];
```

```
position transform_curve_point [ ];
position curve_point_ga [ ];
vector vect_deriv_ga [ ];
position transform_curve_point_ga [ ];
double scale_factor =0;
double fitness = 0;
bs3_curve bs3_ga = NULL;
bs3_curve bs3_ga_old = NULL;
EDGE *ed_ga_old = NULL;
int npts_ga;
double fitol = RESAB;
double actual_tol = 0;
//creates unit vectors for start direction
unit_vector start_dir(0, 0, 0);
unit_vector end_dir (0, 0, 0);

int main()
{
  oli_ACIS_library_initialise();            //Initialise library

//****************   user input    *************************

cout << "**** Spline Definition ****" <<endl;
  cout <<endl;
  cout << "Enter number of data point" <<endl;
  int npts = 0;
  cin >> npts;
  npts_ga = npts;
  double X,Y,Z;
  position vertices [100];
  position vertices_ga [100];
  int k;
  for (k=0; k<npts; k++){
  cout << "Enter point " << k+1 << " coordinates" <<endl;
  cout << "X: ";
  cin >> X;
  cout << "Y: ";
  cin >> Y;
  cout << "Z: ";
  cin >> Z;
  position pos (X,Y,Z);
  position pos_ga (X,Y,Z);
  vertices [k] = pos;
  vertices_ga [k] = pos;
  }
  cout <<endl;
```

```
    cout << "Enter start tangent" <<endl;
    cout << "X: ";
    cin >> X;
    start_dir.component(0) = X;
    cout << "Y: ";
    cin >> Y;
    start_dir.component(0) = Y;
    cout << "Z: ";
    cin >> Z;
    start_dir.component(0) = Z;
    cout <<endl;
    cout << "Enter end tangent" <<endl;
    cout << "X: ";
    cin >> X;
    end_dir.component(0) = X;
    cout << "Y: ";
    cin >> Y;
    end_dir.component(0) = Y;
    cout << "Z: ";
    cin >> Z;
    end_dir.component(0) = Z;

  //*************   create original curve   ****************************
    bs3_curve bs3_original = NULL;
    EDGE *ed_original = NULL;
    oli_create_curve (
                                        npts,
                                        vertices,               // positions[]
                                        start_dir,              // start_dir
                                        end_dir,                // end_dir
                                        fitol,                  // fitol
                                        actual_tol,             // actual_tol
                                        bs3_original,   // bs3_label
                                        ed_original
                                        );
  char *sat_name = "original.sat";      // save into SAT file
  oli_save_entity(sat_name, ed_original);

  //********************* analysis definition    ******************
    cout << "**** analysis definition ****" <<endl;
    cout <<endl;
    cout << "Enter derivative rank" <<endl;
    cin >> rank_derivative;
  cout << "Enter Scale Factor" <<endl;
    cin >> scale_factor;
    oli_input_sample_pitch (match_accuracy);
  oli_curve_nderiv(
                                        match_accuracy,
                                        bs3_original,
                                        curve_point,
                                        vect_deriv,
                                        rank_derivative
                                        );
oli_vector_normal_position_transform(
                                         match_accuracy,
                                         scale_factor,
                                         curve_point,
                                         vect_deriv,
                                         transform_curve_point);
  bs3_curve bs3_curvature = NULL;
  EDGE *ed_curvature = NULL;

  oli_create_curve (       match_accuracy,
                           transform_curve_point,  // positions[]
                           start_dir,                              // start_dir
                           end_dir,                                // end_dir
                           fitol,                                  // fitol
                           actual_tol,                             // actual_tol
                           bs3_curvature,                  // bs3_label
                           ed_curvature);

oli_save_entity(sat_name, ed_curvature);

//***************   optimisation definition      **********************
cout << "**** optimisation definition ****" <<endl;

  double upper_limit [12] = { 0.02, 0.02, 0.02,
                                                  10, 10, 10,
                                                  10, 10, 10,
                                                  0.02, 0.02, 0.02};
  double lower_limit [12] = { 0.02, 0.02, 0.02,
                                                  10, 10, 10,
                                                  10, 10, 10,
                                                  0.02, 0.02, 0.02};

  int numvar = 0;
  int lenstr = 0;
  int maxdecpt = 0;
  srand( 67 );         // generates true random numbers
  numvar = npts_ga * 3;
  input_parameters(popsize, nogen, crossp, prcrossp, mutp, prmutp, numvar);
  Arrayr lowerbound(numvar);
Arrayr upperbound(numvar);
Arrar lenvar(numvar);
  create_bounds_from_positions(
                                  upperbound,
                          lowerbound,
                                  numvar,
                                  vertices_ga,
                                  upper_limit,
                                  lower_limit);

  outtestFile << upperbound << endl;
  outtestFile << lowerbound << endl;
matrices( lenvar, upperbound, lowerbound, numvar, lenstr, maxdecpt );
  initialise( popsize, oldpop, lenvar, upperbound, lowerbound, lenstr );
  initialise( popsize, intpop, lenvar, upperbound, lowerbound, lenstr );
  initialise( popsize, newpop, lenvar, upperbound, lowerbound, lenstr );

  for ( int i = 0; i < nogen; i++ ) {
  roulette( oldpop, intpop, popsize, maxdecpt );
  reproduction( intpop, newpop, popsize, crossp, prcrossp, mutp, prmutp );
          for (int  j = 0; j < popsize; j++ ) {
          *oldpop[j] = *newpop[j];
          outDistanceFile << newpop[j]->fitdata() <<endl;
          }
          power = power + (double (power_finish) - double (power_start)) / double (nogen);
  }

//*******************    statistics    *************************
  for (int j=0; j<popsize; j++){
  outtestFile <<endl;
  outtestFile <<"Individual "<< j <<endl;
  outtestFile <<"fitness function value ="<< newpop[j]->fitfun() <<endl;
  outtestFile <<"distance value ="<< newpop[j]->fitdata() <<endl;
  for (int i=0; i<npts_ga; i++){
  outtestFile<< "X" << i << " = "<< newpop[j]->getVarval()[3*i] <<endl;
  outtestFile<< "Y" << i << " = "<< newpop[j]->getVarval()[3*i+1] <<endl;
  outtestFile<< "Z" << i << " = "<< newpop[j]->getVarval()[3*i+2] <<endl;
}
          }
char const* leader = "";
FILE *fp;
  if (!(fp = fopen("final_curve.txt", "w")))
  {fprintf(stderr, "fopen() failed \n");}
bs3_curve_debug (         bs3_ga_old,
                                  leader,
                                  fp);
```

```
int dim ;
int deg ;
logical rat;
int num_ctrlpts;
position *ctrlpts [ ];
double* weights [ ];
int num_knots;
double* knots [ ];

bs3_curve_to_array (
            bs3_ga_old,             // given curve
            dim,                    // returned dimension
            deg,                    // returned degree
            rat,                    // returned rational
  num_ctrlpts,              // returned number of  control points
            ctrlpts,                // returned control points
weights,            // returned weights
num_knots,                  // returned number of  knots
            knots                   // knot);

outFinalCurve << "returned dimension " << *& dim <<endl;
outFinalCurve << "returned degree " << *& deg <<endl;
outFinalCurve << "returned rational " << *& rat <<endl;
outFinalCurve << "returned number of control points " << *& num_ctrlpts <<endl;

for (i=0; i<num_ctrlpts; i++){
  outFinalCurve << "returned control points " << ctrlpts[i] <<endl;
  outFinalCurve << "returned weights " << weights[i] << endl;
}

outFinalCurve << "returned number of knots " << *& num_knots <<endl;

for (i=0; i<num_ctrlpts; i++){
  outFinalCurve << "returned knots " << *& knots[i] <<endl;
}

outFinalCurve << endl;

oli_ACIS_curve (bs3_ga_old,  // bs3_label
                          ed_ga_old       // edge_label*
                          );

sat_name = "ga_curve.sat";
oli_save_entity(sat_name, ed_ga_old);
```

```
oli_curve_nderiv(
                                        match_accuracy,
                                        bs3_ga_old ,
                                        curve_point,
                                        vect_deriv,
                                        rank_derivative
                                        );
oli_vector_normal_position_transform(
                                         match_accuracy,
                                         scale_factor,
                                         curve_point,
                                         vect_deriv,
                                         transform_curve_point
                                         );

  bs3_curve bs3_curvature_final = NULL;
  EDGE *ed_curvature_final = NULL;

  oli_create_curve (      match_accuracy,
                                        transform_curve_point,
start_dir,
                                        end_dir,
fitol,
                                        actual_tol,
                                bs3_curvature_final,
  ed_curvature_final);
sat_name = "curvature_final.sat";
oli_save_entity(sat_name, ed_curvature_final);
return 0;
}

//***************   fitness value   **************************
double Individual::fitdata() const{
  Arrayr Varval(getNumvar());
for (int i = 0; i < getNumvar(); i++ ) {
          Varval[i] = getVarval()[i];
  }
  create_positions_from_GA_variables(
                                        npts_ga,
                                        Varval,
                                        new_points);

  bs3_ga = bs3_curve_interp (
                                                npts_ga,
                                                new_points,
                                                start_dir,
                                                  end_dir,
                                                  fitol,
                                                  actual_tol
                                                  );
double start = 0;            // start parameter
double end = 1;                   // end parameter

bs3_curve_reparam (
                     start,
                     end,
                     bs3_ga);

oli_curve_nderiv(
                          match_accuracy,
                          bs3_ga,
                          curve_point_ga,
                          vect_deriv_ga,
                          rank_derivative);

oli_vector_normal_position_transform(
                          match_accuracy,
                          scale_factor,
                          curve_point_ga,
                          vect_deriv_ga,
                          transform_curve_point_ga);

oli_position_distance(
                match_accuracy,
                transform_curve_point,
                transform_curve_point_ga,
                average_dist,
                average_dist_X,
                average_dist_Y,
                average_dist_Z,
                scale_factor);

if (average_dist >= average_dist_old){

bs3_curve_delete ( bs3_ga );
else{
average_dist_old = average_dist;
bs3_curve_delete (bs3_ga_old);
bs3_ga_old = bs3_curve_copy ( bs3_ga );
bs3_curve_delete ( bs3_ga );}
cout<< average_dist <<endl;
return average_dist;}
```

```
//***************************************************
//*************    Fitness Function     ********************

double Individual::fitfun() const{
   fitness = pow((1/(1 + fitdata())), power);
   return fitness;
}
```

Appendix 6 GA tests

6.1 Fitness Function Mapping

The first series of tests performed are the mapping of the fitness function. These tests are a record of the fitness values returned by the fitness function plotted in a graph showing the scores against data point coordinates moving away incrementally and positively from their nominal values. The nominal values are in effectively a solution known in advance for which the fitness value is maximum.

The mapping of the fitness function is one way of visualising the solution space. This also helps to determine the nature of the optimisation problem that guides the choice of the algorithm for the problem. The curve used here is a spline interpolated through 4 data points and the optimisation is carried on the curvature property as objective function. Since the visual space is limited to three axes, one or two point coordinates maximum are variable, the others are maintained as constant as shown in Table 11.

Point	Pt 2			Pt 3		
Coordinate	X	Y	Z	X	Y	Z
6.1.1						
6.1.2						
6.1.3						
6.1.4						
The cells ticked are the point coordinates set as variables						

Table 11: Experiment table

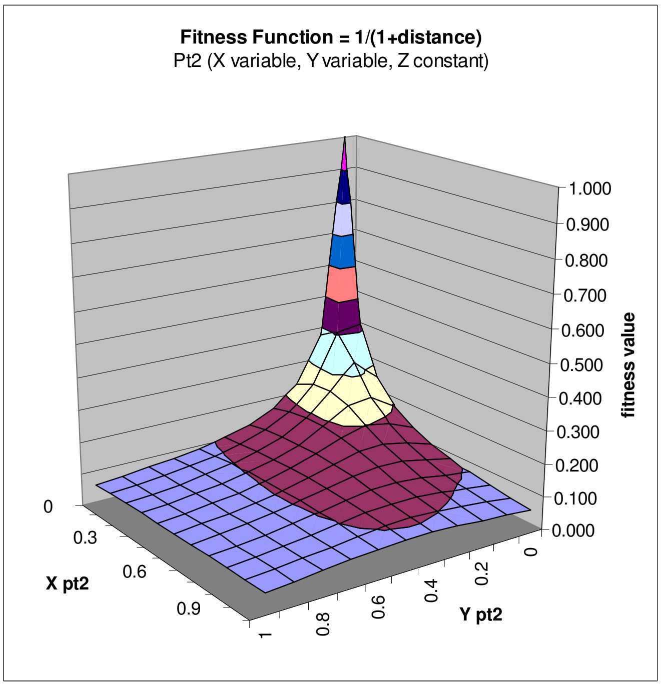
In this experiment, the coordinates vary one unit of their nominal value with an increment of 0.1 of the unit. The implementation of each test is presented below with its table value and graph analysis in subsequent paragraphs.

These tests show that for all variables, the fitness function exhibits a high response within 0.2 of the unit out from the nominal value. This means that the optimisation algorithm will converge quickly and accurately towards the optimum solution. Also in all the maps it is observed that there is a unique peak, which means that the function is uni-modal.

6.1.1 Pt2 (X variable, Y variable, Z constant)

```
//curvature analysis (2nd derivative)
PROJECT: curvature_fitness_fun_1
double fitness = sum_dist;

for (double k=0; k<1; k+=0.1) {
            for (double l=0; l<1; l+=0.1) {
            int npts_ga = 4;
            position pt1_ga (0, 0, 0);
            position pt2_ga (k+10, l+5, 5);
            position pt3_ga (20, 30, 10);
            position pt4_ga (40, 0, 7);
            }
}
```

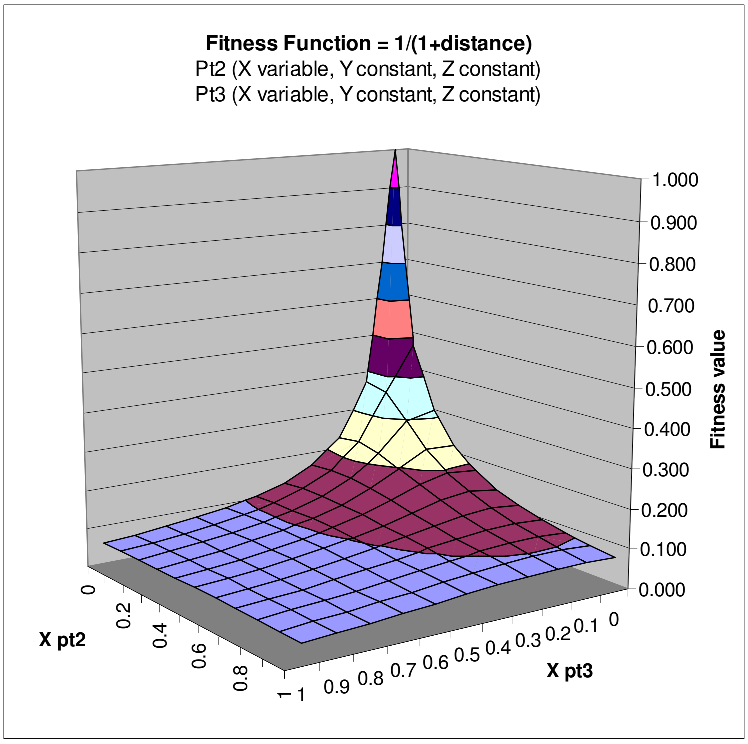
Y X	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	1.000	0.374	0.230	0.166	0.130	0.106	0.090	0.078	0.069	0.062	0.056
0.1	0.418	0.408	0.253	0.180	0.139	0.113	0.095	0.082	0.072	0.064	0.058
0.2	0.265	0.333	0.257	0.187	0.144	0.117	0.098	0.085	0.074	0.066	0.059
0.3	0.195	0.243	0.237	0.187	0.148	0.120	0.101	0.087	0.076	0.068	0.061
0.4	0.154	0.186	0.202	0.180	0.148	0.122	0.103	0.089	0.078	0.069	0.062
0.5	0.128	0.150	0.168	0.164	0.144	0.122	0.104	0.090	0.079	0.070	0.063
0.6	0.109	0.126	0.141	0.146	0.136	0.120	0.104	0.090	0.080	0.071	0.064
0.7	0.095	0.108	0.121	0.128	0.125	0.115	0.102	0.090	0.080	0.071	0.064
0.8	0.085	0.095	0.105	0.113	0.115	0.109	0.099	0.089	0.080	0.072	0.065
0.9	0.077	0.085	0.093	0.101	0.104	0.102	0.096	0.087	0.079	0.072	0.065
1	0.070	0.076	0.084	0.090	0.095	0.095	0.091	0.085	0.078	0.071	0.065



6.1.2 Pt2 (X variable, Y constant, Z constant); Pt3 (X variable, Y constant, Z constant)

```
//curvature analysis (2nd derivative)
PROJECT: curvature_fitness_fun_1
double fitness = sum_dist;

for (double k=0; k<1; k+=0.1) {
                for (double l=0; l<1; l+=0.1) {
                int npts_ga = 4;
                position pt1_ga (0, 0, 0);
                position pt2_ga (k+10, 5, 5);
                position pt3_ga (l+20, 30, 10);
                position pt4_ga (40, 0, 7);
                }
}
```

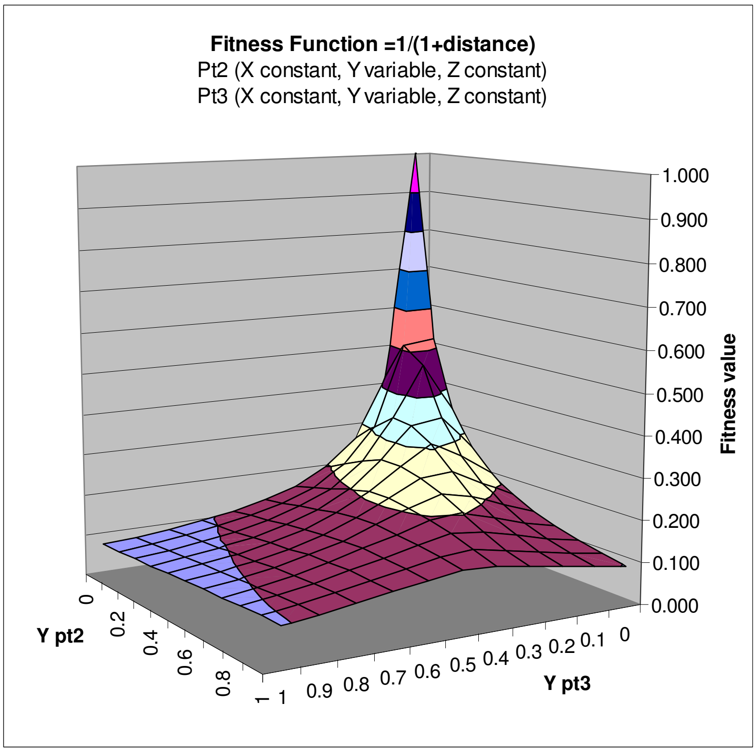
Y \ X	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	1.000	0.371	0.228	0.164	0.128	0.105	0.089	0.078	0.068	0.061	0.055
0.1	0.479	0.362	0.236	0.171	0.133	0.109	0.092	0.080	0.070	0.063	0.057
0.2	0.315	0.297	0.221	0.168	0.133	0.110	0.093	0.081	0.071	0.064	0.058
0.3	0.235	0.239	0.198	0.159	0.130	0.109	0.093	0.081	0.072	0.064	0.058
0.4	0.187	0.195	0.175	0.148	0.124	0.106	0.092	0.081	0.072	0.064	0.058
0.5	0.156	0.164	0.154	0.136	0.118	0.102	0.090	0.079	0.071	0.064	0.058
0.6	0.133	0.140	0.136	0.124	0.110	0.098	0.087	0.077	0.070	0.063	0.058
0.7	0.117	0.122	0.121	0.113	0.103	0.093	0.083	0.075	0.068	0.062	0.057
0.8	0.104	0.108	0.108	0.104	0.096	0.088	0.080	0.073	0.066	0.061	0.056
0.9	0.093	0.097	0.098	0.095	0.089	0.083	0.076	0.070	0.064	0.059	0.055
1	0.085	0.088	0.089	0.087	0.083	0.078	0.073	0.067	0.062	0.058	0.054



6.1.3 Pt2 (X constant, Y variable, Z constant); Pt3 (X constant, Y variable, Z constant)

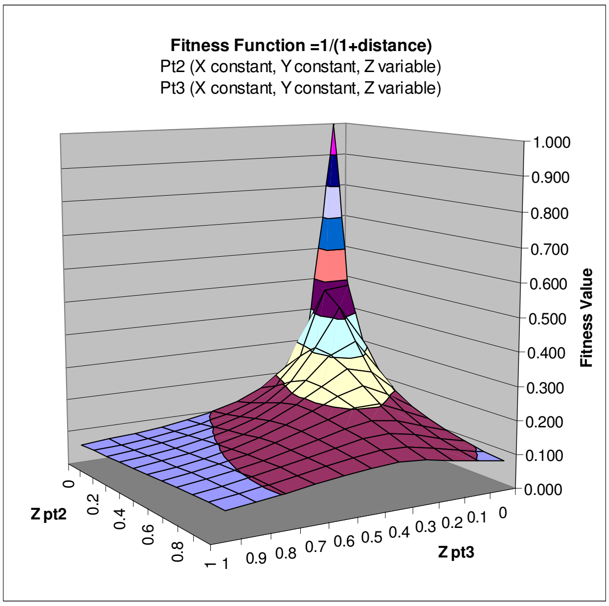
```
//curvature analysis (2nd derivative)
PROJECT: curvature_fitness_fun_1
double fitness = sum_dist;

for (double k=0; k<1; k+=0.1) {
            for (double l=0; l<1; l+=0.1) {
            int npts_ga = 4;
            position pt1_ga (0, 0, 0);
            position pt2_ga (10, k+5, 5);
            position pt3_ga (20, l+30, 10);
            position pt4_ga (40, 0, 7);
            }
}
```

Y X	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	1.000	0.417	0.264	0.193	0.153	0.127	0.108	0.095	0.084	0.076	0.069
0.1	0.522	0.514	0.308	0.217	0.168	0.137	0.116	0.100	0.089	0.079	0.072
0.2	0.353	0.472	0.346	0.241	0.183	0.147	0.123	0.106	0.093	0.083	0.075
0.3	0.267	0.351	0.341	0.261	0.198	0.158	0.131	0.112	0.098	0.087	0.078
0.4	0.214	0.269	0.309	0.262	0.210	0.168	0.139	0.118	0.102	0.090	0.081
0.5	0.179	0.217	0.257	0.249	0.212	0.175	0.146	0.124	0.107	0.094	0.084
0.6	0.154	0.182	0.213	0.230	0.205	0.177	0.151	0.129	0.111	0.098	0.087
0.7	0.135	0.156	0.180	0.202	0.196	0.174	0.152	0.132	0.115	0.101	0.090
0.8	0.120	0.137	0.156	0.175	0.183	0.168	0.151	0.134	0.118	0.104	0.093
0.9	0.108	0.122	0.137	0.153	0.166	0.161	0.147	0.133	0.119	0.107	0.095
1	0.098	0.109	0.122	0.135	0.148	0.152	0.142	0.130	0.119	0.107	0.097



6.1.4 Pt2 (X constant, Y constant, Z variable); Pt3 (X constant, Y constant, Z variable)

```
//curvature analysis (2nd derivative)
PROJECT: curvature_fitness_fun_1
double fitness = sum_dist;

for (double k=0; k<1; k+=0.1) {
                for (double l=0; l<1; l+=0.1) {
                int npts_ga = 4;
                position pt1_ga (0, 0, 0);
                position pt2_ga (10, 5, k+5);
                position pt3_ga (20, 30, l+10);
                position pt4_ga (40, 0, 7);
                }
}
```



Y X	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	1.000	0.358	0.218	0.157	0.123	0.101	0.085	0.074	0.065	0.059	0.053
0.1	0.483	0.476	0.270	0.183	0.138	0.111	0.093	0.079	0.070	0.062	0.056
0.2	0.319	0.433	0.312	0.212	0.156	0.122	0.101	0.086	0.074	0.066	0.059
0.3	0.238	0.338	0.304	0.232	0.174	0.135	0.110	0.092	0.079	0.070	0.062
0.4	0.190	0.257	0.276	0.231	0.185	0.146	0.119	0.099	0.085	0.074	0.065
0.5	0.158	0.204	0.240	0.218	0.185	0.153	0.126	0.106	0.090	0.078	0.069
0.6	0.135	0.169	0.203	0.202	0.179	0.154	0.131	0.111	0.095	0.082	0.072
0.7	0.118	0.143	0.172	0.184	0.170	0.151	0.132	0.114	0.099	0.086	0.076
0.8	0.105	0.125	0.148	0.164	0.160	0.145	0.130	0.115	0.101	0.089	0.079
0.9	0.094	0.110	0.129	0.145	0.148	0.139	0.127	0.114	0.102	0.091	0.081
1	0.086	0.099	0.114	0.129	0.136	0.132	0.122	0.112	0.102	0.092	0.083

6.2 Fitness Boosting

The general concept of the fitness boosting technique is introduced in paragraph 0, this appendix documents the experiments carried out during the development.

This experiment uses the same test function as in appendix 6.1 above. Here a power function is applied to the fitness function for boosting. The power is incremented from a start value to an end value at every generation as shown in the implementation below. It is desirable to proceed with an increment on the power value to prevent penalising average solutions in order to leave enough genetic diversity within the population. A series of test are presented below, the setting values are given for each experiment in a frame above each graph.

Implementation

```
int power_start = 1;                          //start power value
int power_finish = 8;                         //end power value
double power = power_start;                   //initialise power at start value
int nogen = 50;                       //number of generations
int popsize = 50;                     //population size

for ( int i = 0; i < nogen; i++ ) {
  roulette( oldpop, intpop, popsize, maxdecpt );
  reproduction( intpop, newpop, popsize, crossp, prcrossp, mutp, prmutp );
                  for (int  j = 0; j < popsize; j++ ) {
         *oldpop[j] = *newpop[j];

         outDistanceFile << newpop[j]->fitdata() <<endl;
         outpower<<power<<endl;
         }

  power = power + (double (power_finish) - double (power_start)) / double (nogen);}
```

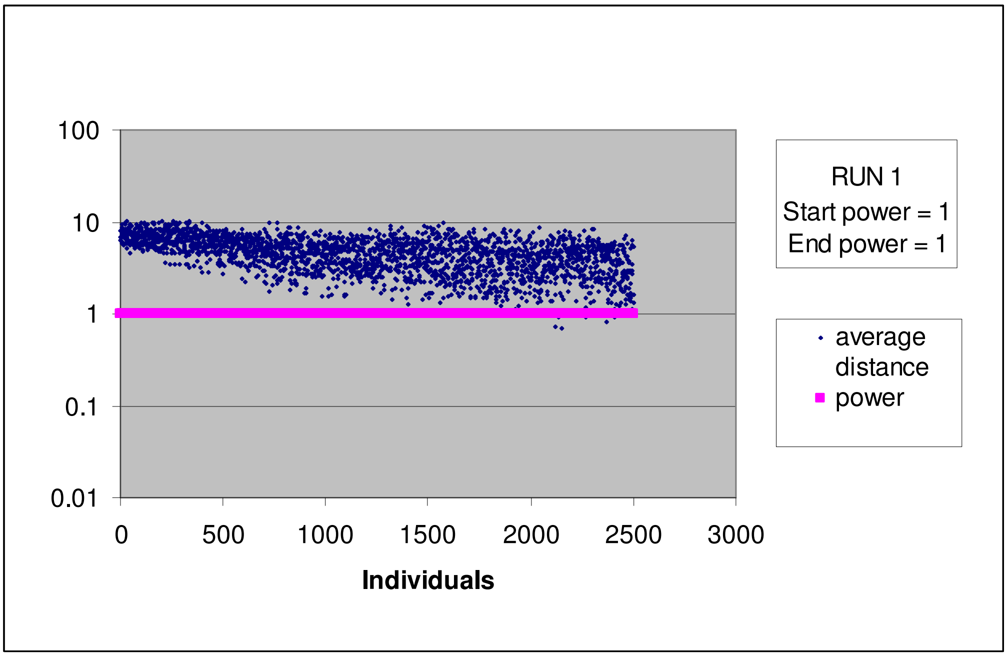
Settings

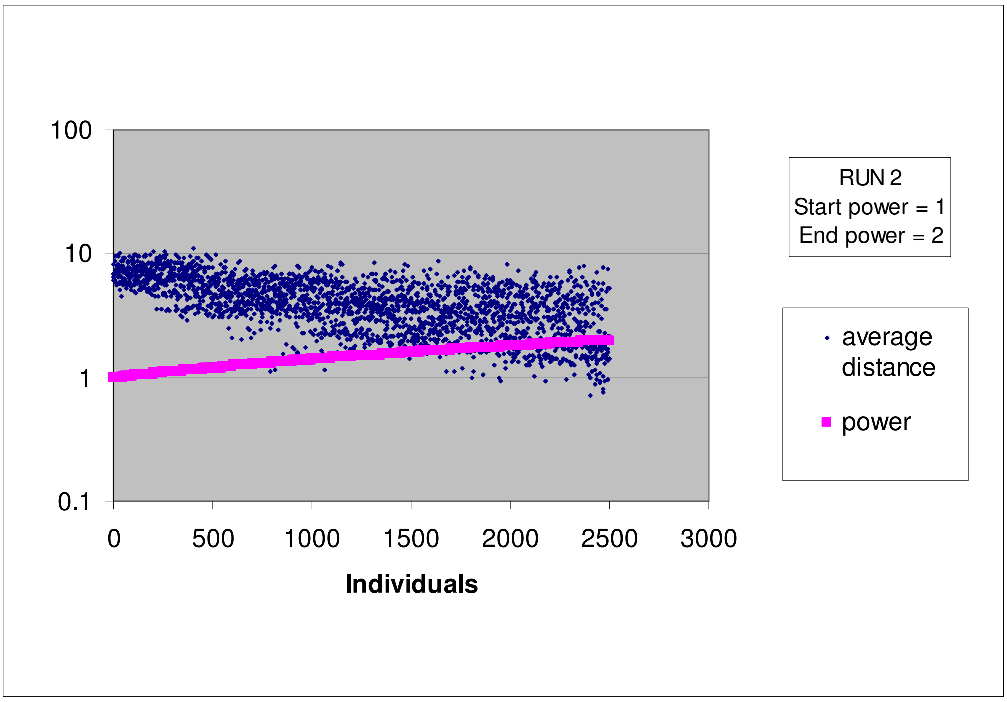
position pt1 (0, 0, 0);
position pt2 (10, 5, 5);
position pt1 (0, 0, 0);
position pt2 (10, 5, 5);
position pt3 (20, 30, 10);
position pt4 (40, 0, 0);

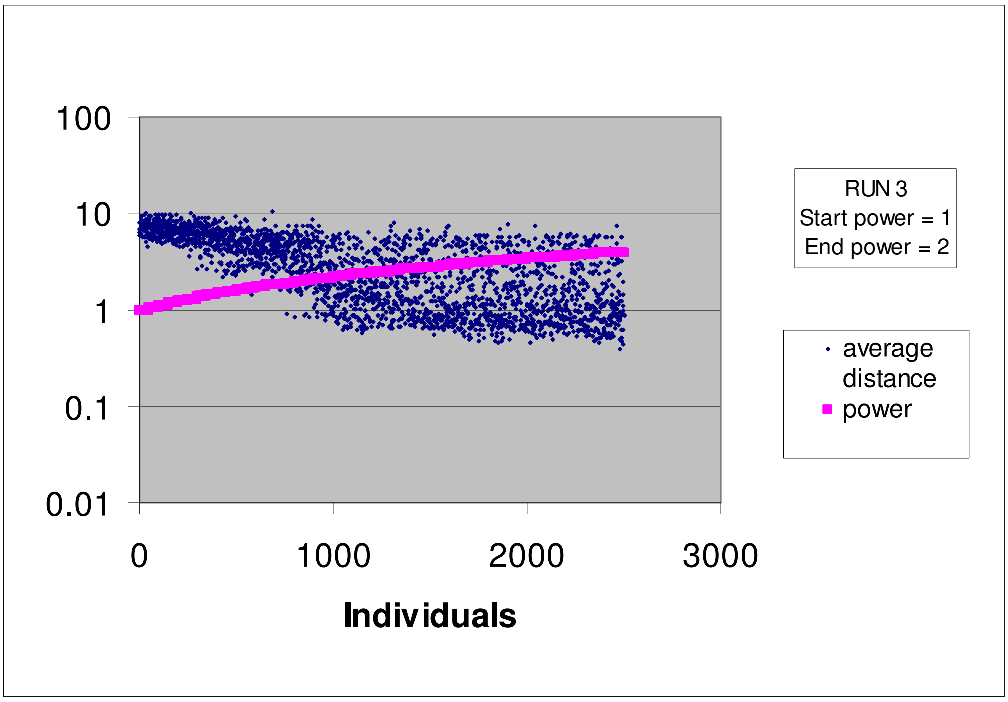
point	50
gen	50
pop	50
xover	0.8
xover level	2
mutation	0.02
mtation level	2

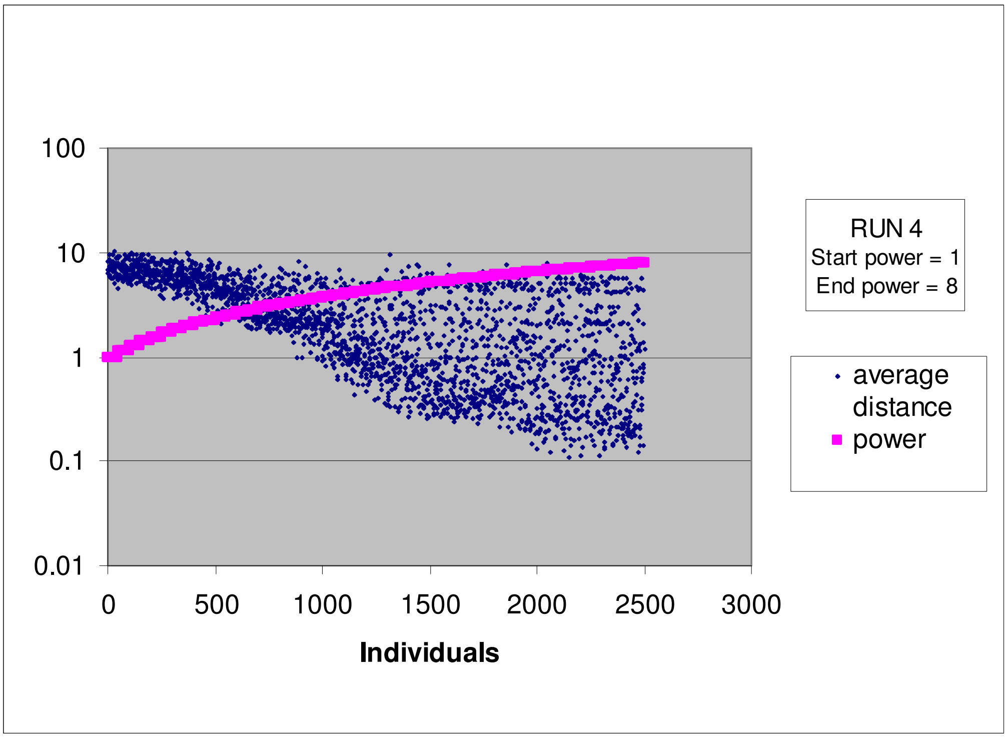
```
double upper_limit [12] = { 0.02, 0.02, 0.02,
                            10, 10, 10,
                            10, 10, 10,
                            0.02, 0.02, 0.02};

double lower_limit [12] = { 0.02, 0.02, 0.02,
                            10, 10, 10,
                            10, 10, 10,
                            0.02, 0.02, 0.02};
```

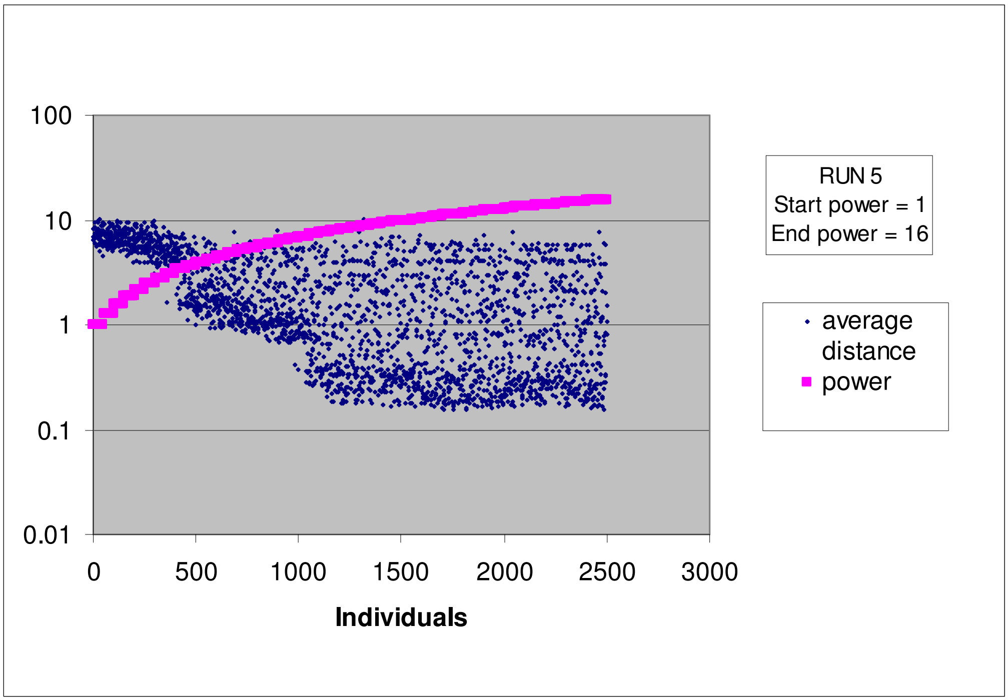
Experiments




100
10
1
0.1
0
500
1000
1500
2000
2500
3000
Individuals
RUN 2
Start power = 1
End power = 2
average distance
power


100
10
1
0.1
0.01
0
1000
2000
3000
Individuals
RUN 3
Start power = 1
End power = 2
average distance
power


100
10
1
0.1
0.01
0
500
1000
1500
2000
2500
3000
Individuals
RUN 4
Start power = 1
End power = 8
average distance
power


100
10
1
0.1
0.01
0
500
1000
1500
2000
2500
3000
Individuals
RUN 5
Start power = 1
End power = 16
average distance
power

Observations

The main observation that can be drawn from this experiment is that power applied to the fitness function results in better convergence of the GA. This is true up to certain limit. With a power range of 1 to 16 the solution band becomes too narrow, and population diversity is too small to allow the GA to converge towards an optimum. The best results are obtained with a power range of 1 to 8, which are the settings retained for the application of curve optimisation.

6.3 GA Operators

For the tests on GA operators, the best power settings found previously are selected here and kept constant all the way through the experiments.

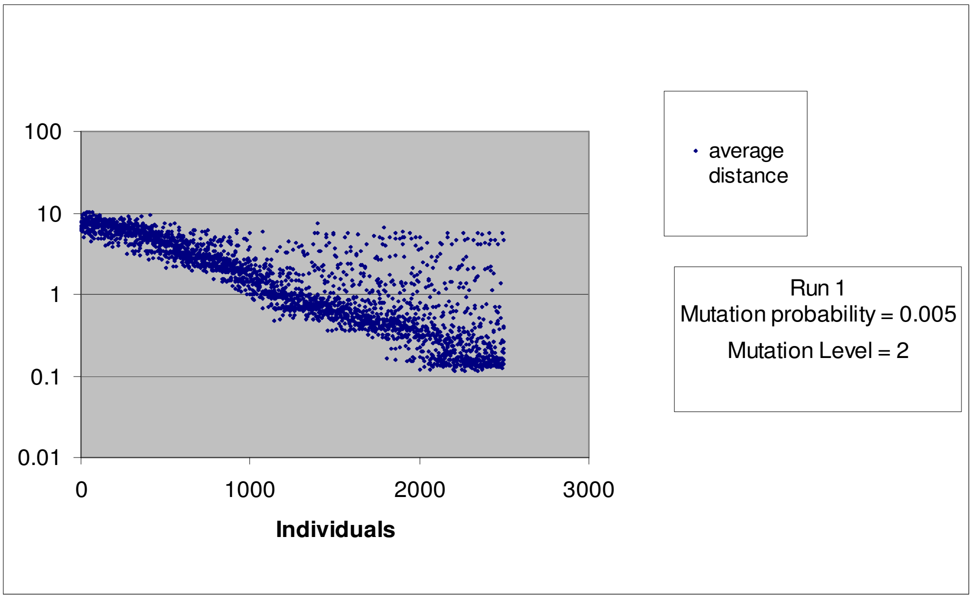
6.3.1 Mutation

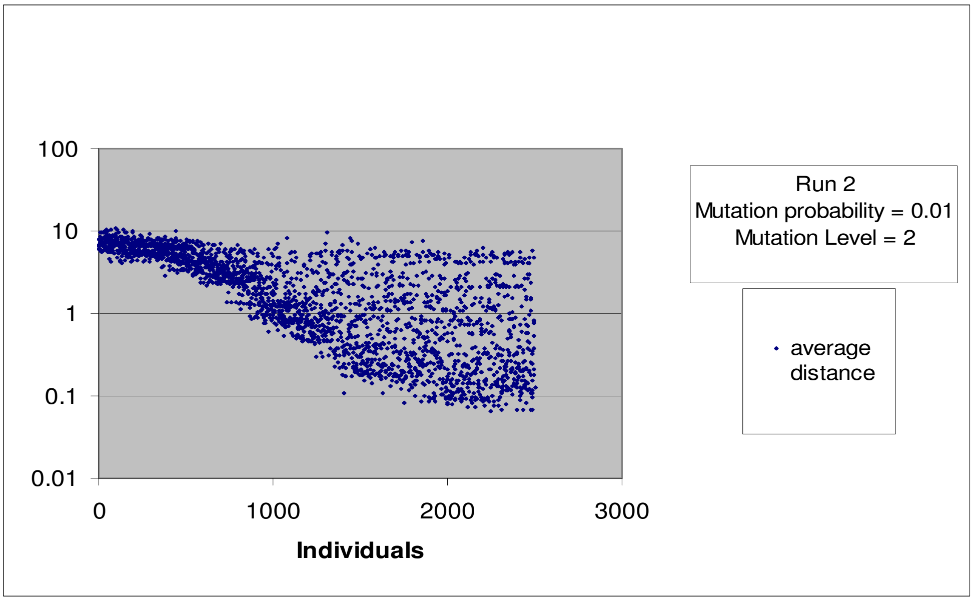
The mutation operator has two settings: the probability mutation occurs on an individual and the chromosome level mutation is applied. Both settings are explored in this experiment starting with the mutation probability.

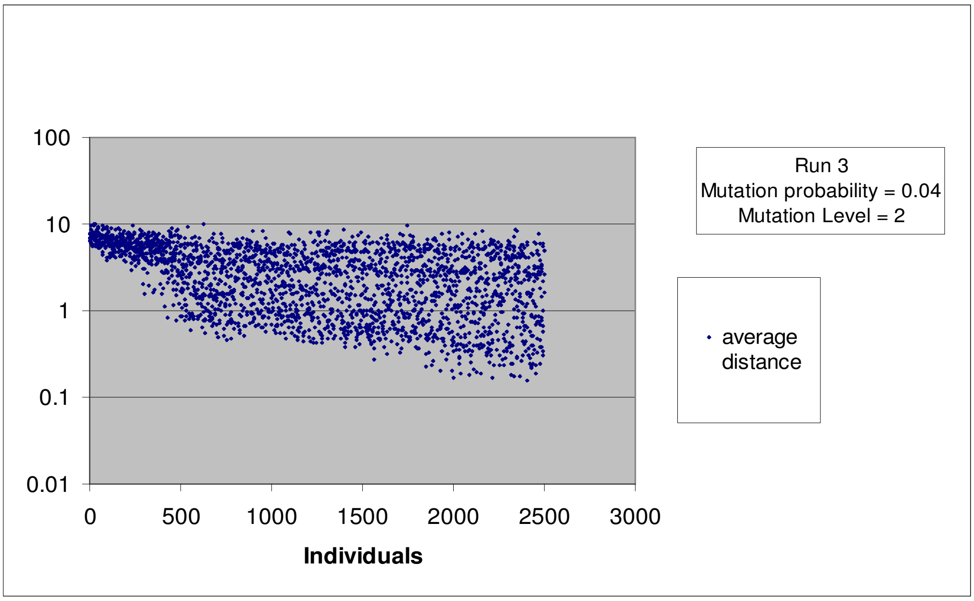
Settings

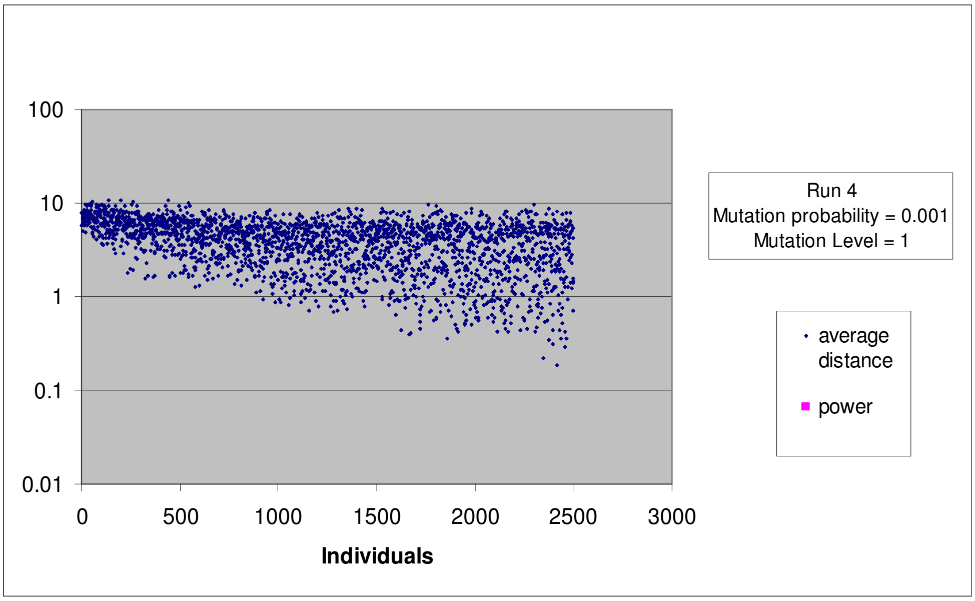
```
position pt1 (0, 0, 0);
position pt2 (10, 5, 5);
position pt1 (0, 0, 0);              double upper_limit [12] = { 0.02, 0.02, 0.02,
position pt2 (10, 5, 5);                                          10, 10, 10,
position pt3 (20, 30, 10);                                        10, 10, 10,
position pt4 (40, 0, 0);                                     0.02, 0.02, 0.02};

point              50                double lower_limit [12] = { 0.02, 0.02, 0.02,
gen                50                                             10, 10, 10,
pop                50                                             10, 10, 10,
xover             0.8                                        0.02, 0.02, 0.02};
xover level         2
power start         1
power finish        8
```

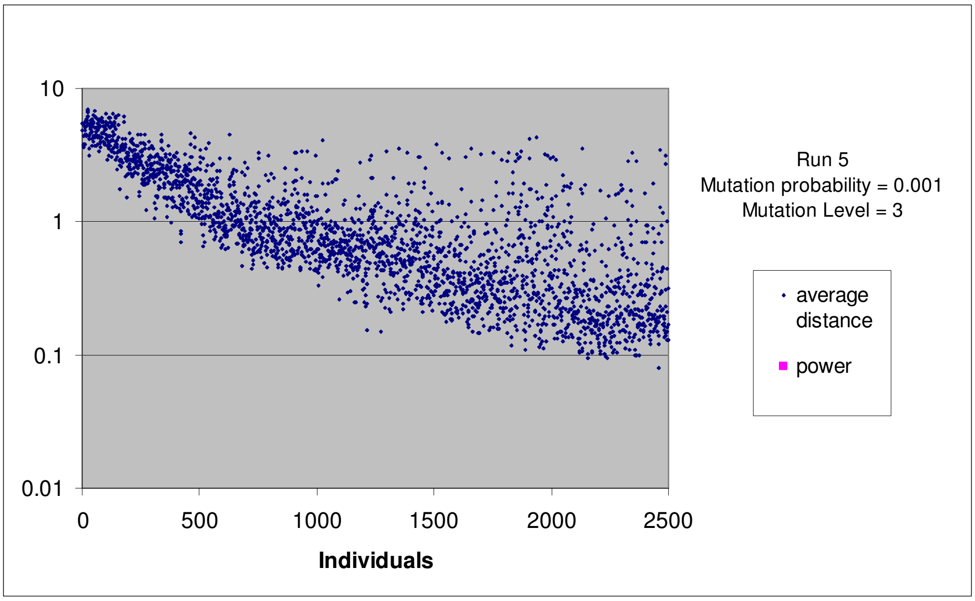
Graphs


100
10
1
0.1
0.01
0
1000
2000
3000
Individuals
average distance
Run 1
Mutation probability = 0.005
Mutation Level = 2


100
10
1
0.1
0.01
0
1000
2000
3000
Individuals
Run 2
Mutation probability = 0.01
Mutation Level = 2
average distance


100
10
1
0.1
0.01
0
500
1000
1500
2000
2500
3000
Individuals
Run 3
Mutation probability = 0.04
Mutation Level = 2
average distance


100
10
1
0.1
0.01
0
500
1000
1500
2000
2500
3000
Individuals
Run 4
Mutation probability = 0.001
Mutation Level = 1
average distance
power



Observations

As the probability increases, the solution band gets wider with a Paretto front less pronounced, the solutions are more spread out. The GA is able to explore the search space and find solutions towards the optimum quickly. However with a probability of 0.04, it is observed that too much mutation is introduced to the population results of a weak Paretto front: not enough solutions are close to the front and the GA has difficulty to converge.

Like for the probability, the mutation level, it has the effect of concentrating solutions towards the Paretto front as it increases. This is an interesting property because it is preferable to have a solution band following the trend of the Paretto front. The settings retained after this experiment are a probability of 0.01 and a level of 2.

6.3.2 Cross-over

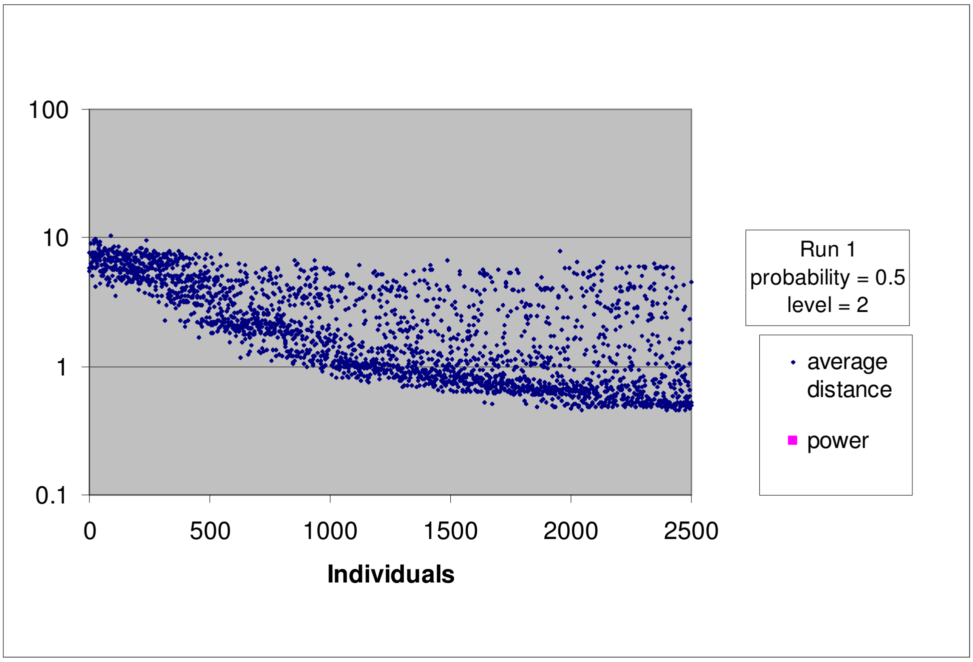
Settings

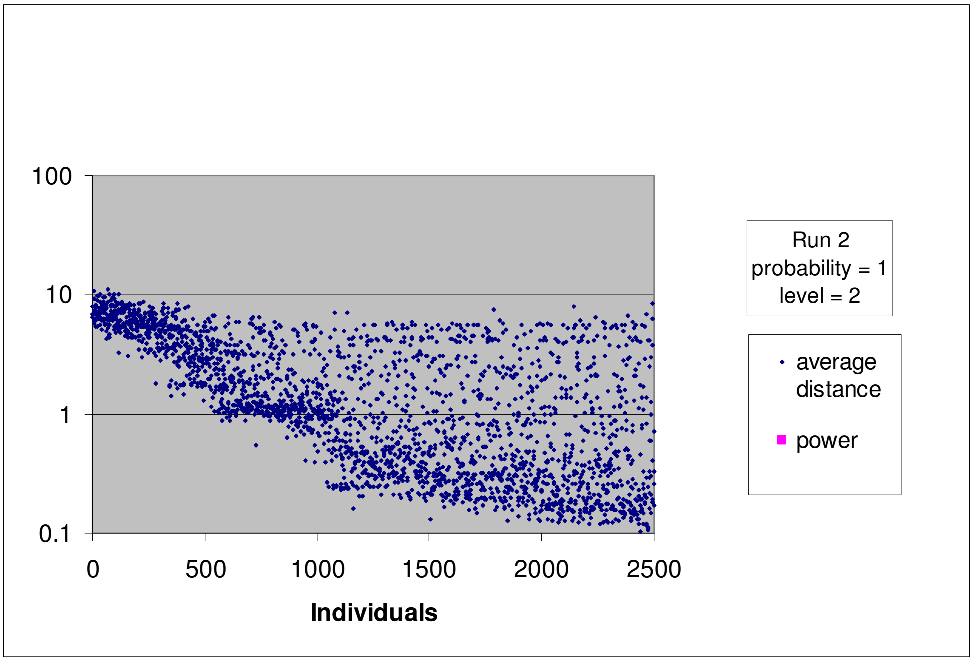
```
position pt1 (0, 0, 0);
position pt2 (10, 5, 5);
position pt1 (0, 0, 0);
position pt2 (10, 5, 5);
position pt3 (20, 30, 10);
position pt4 (40, 0, 0);
```

point	50
gen	50
pop	50
mutation	0.01
mtation level	2
power start	1
power finish	8

```
double upper_limit [12] = { 0.02, 0.02, 0.02,
                            10, 10, 10,
                            10, 10, 10,
                            0.02, 0.02, 0.02};

double lower_limit [12] = { 0.02, 0.02, 0.02,
                            10, 10, 10,
                            10, 10, 10,
                            0.02, 0.02, 0.02};
```

Graphs





Observations

It is observed that with an increasing cross-over probability the GA converges better, in actual fact the best result is obtained with a probability of 1. As for the cross over level it is no effect on the convergence.

Appendix 7 Implementation of Curve Optimisation Application Using F-Rep

The code presented below traces out the curvature fin, Equation 23, the torsion fin, Equation 24, the curvature plot, Equation 11, and the torsion plot, Equation 17 of a space curve.

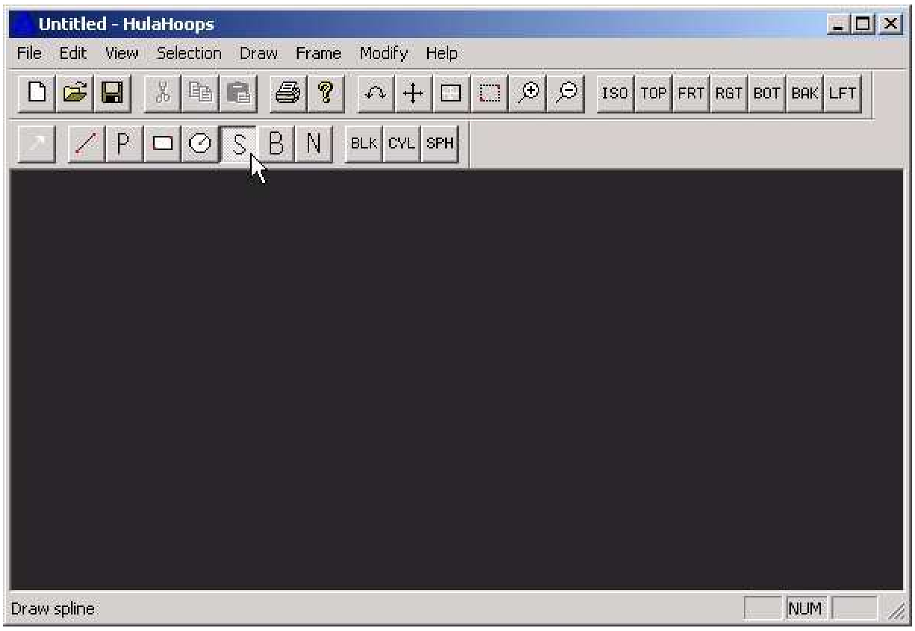
```
#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include "wot.h"

int main()
{
ENTITY_LIST *curve_bs3 = new ENTITY_LIST;
ENTITY_LIST *u_c_list = new ENTITY_LIST;
ENTITY_LIST *u_t_list = new ENTITY_LIST;
//Initialise modeler and LAW class library
oli_ACIS_library_initialise();
initialize_law ();

// Construct a curve from control points
int const Nb_CtrlPpts = 4;
position pt1  (40, 40, 40);
position pt2  (45, 120, 120);
position pt3  (85,10, 80);
position pt4  (100, 70, 30);
position  vertices[*&Nb_CtrlPpts] = {pt1, pt2, pt3, pt4};//,pt5};//,pt6};
bs3_curve bs3_label = NULL;
int const degree = 3;
int const nb_knots = Nb_CtrlPpts + degree - 1 ;
double const knots[nb_knots] = {0,0,0,1,1,1}; //that makes a cubic Bezier
double weights[nb_knots]= {(double)NULL}; //it's not a rational
double fitol = resabs;
logical rational = FALSE;
logical closed = FALSE;
logical periodic = FALSE;
int dimension = 3;

bs3_label = bs3_curve_from_ctrlpts   (
                                        degree,
                                        rational,
                                        closed,
                                        periodic,
                                        Nb_CtrlPpts,
                                        vertices,
                                        weights,
                                        fitol,
                                     nb_knots,
                                     knots,
                                     fitol,
                                     dimension
                                     );

//reparameterise the bs3_curve from  0 to 1...
double start =0;
double end =1;
bs3_curve_reparam ( start,end,bs3_label);
// saves the curve into a SAT file
EDGE* edge_label = NULL;
oli_ACIS_curve (bs3_label, edge_label);
oli_add_entity_List(edge_label, curve_bs3);

curve_law* position_vector = NULL;
law *tangent = NULL;
law *curvature_vector = NULL;
law *curvature = NULL;
law *torsion_vector = NULL;
law *torsion = NULL;
// gets the LAW representation of the curve
oli_edge_to_law(edge_label, position_vector);
// gets the Frenet frame
oli_Frenet_frame_law (
                                     position_vector,
                                     tangent,
                                     curvature_vector,
                                     curvature,
                                     torsion_vector,
                                     torsion
                                     );

//Trace the Curvature fin
law *scale_factor_curvature = new constant_law(10);
law *curvature_scaled = new times_law(normal, scale_factor_curvature);
law *curvature_fin = new minus_law(position_vector, curvature_scaled);

//Trace the Torsion fin
law *scale_factor_torsion = new constant_law(10);
law *torsion_scaled = new times_law(bi_normal, scale_factor_torsion);
law *torsion_fin = new plus_law(position_vector, torsion_scaled);

// Curvature Vs parameter plot
int in_which = 0;
char *in_name ="u";
```

```
law* u = new identity_law (          In_which, *in_name);
law* zero = new constant_law(0);
law *u_c_array[3] = {u, curvature,zero};
law *u_c = new vector_law(u_c_array, 3);
law *u_c_plot = u_c;

//Torsion Vs parameter
law *u_t_array[3] = {u, torsion,zero};
law *u_t = new vector_law(u_t_array, 3);
law *u_t_plot = u_t;

// MAKE EGDES AND SAVE INTO SAT FILES
EDGE* curvature_fin_edge = NULL;
oli_law_to_edge(
                                        curvature_fin,
                                        start,
                                        end,
                                        curvature_fin_edge
                                        );
oli_add_entity_List(curvature_fin_edge, frenet_frame_list);
EDGE* torsion_fin_edge = NULL;
oli_law_to_edge(
                                        torsion_fin,
                                        start,
                                        end,
                                        torsion_fin_edge );

oli_add_entity_List(torsion_fin_edge, frenet_frame_list);
EDGE* u_c_plot_edge = NULL;
oli_law_to_edge(
                                        u_c_plot,
                                        start,
                                        end,
                                        u_c_plot_edge );

oli_add_entity_List(u_c_plot_edge, u_c_list);
EDGE* u_t_plot_edge = NULL;
oli_law_to_edge(
                                        u_t_plot,
                                        start,
                                        end,
                                        u_t_plot_edge );
oli_add_entity_List(u_t_plot_edge, u_t_list);
return 0;
_t_list);
```

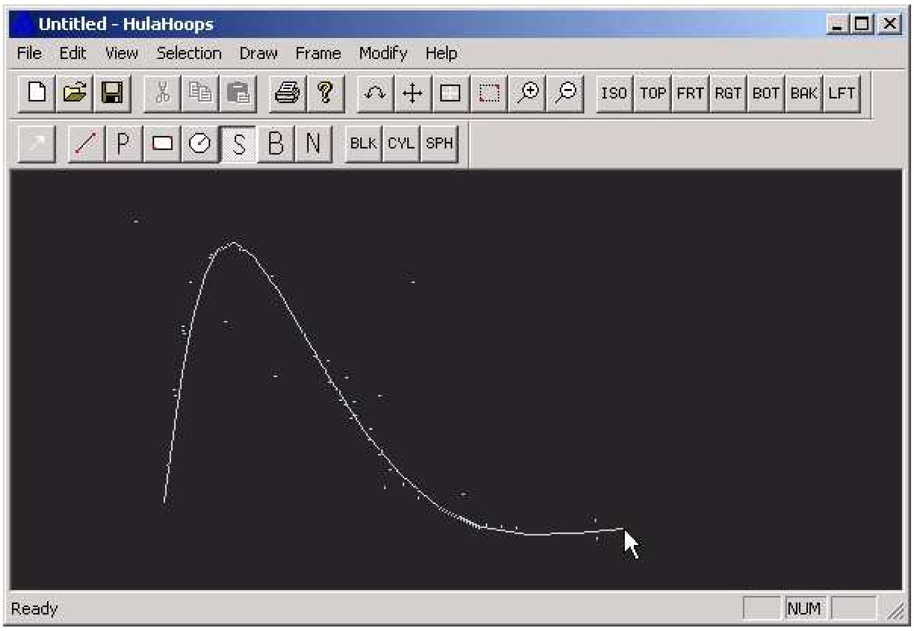
Appendix 8 HulaHoops User Interface

8.1 Draw an Interpolated Spline

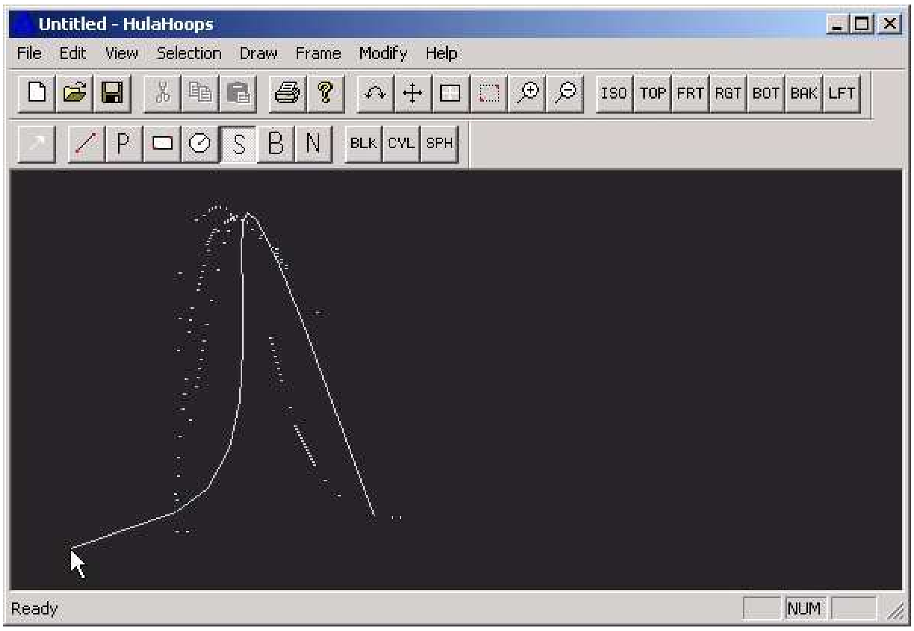
- Click on ‘S’ icon or


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Draw spline
NUM

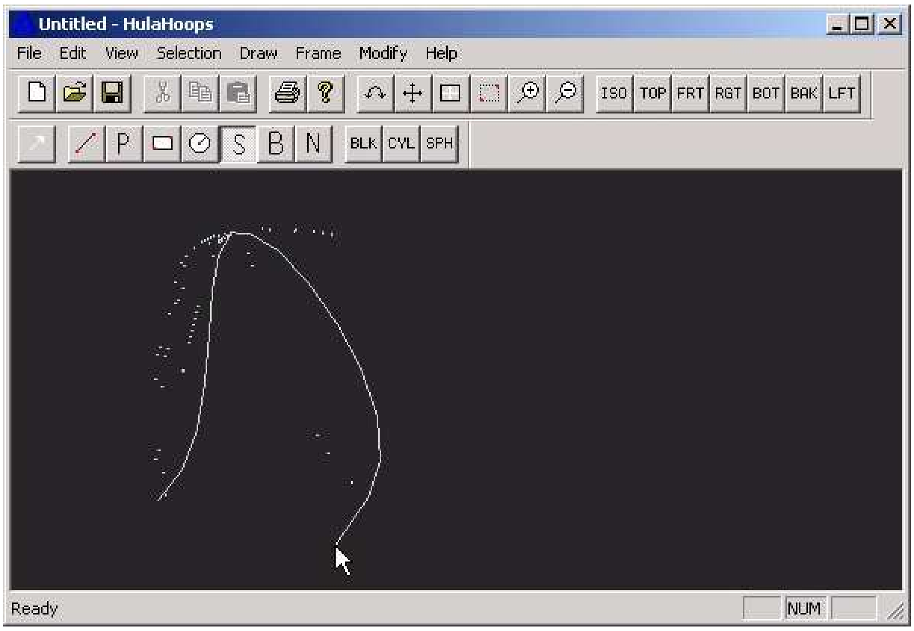
- Menu ‘Draw’ ‘Curve’ ‘Spline’
- • Left click to select data points
- • Right click to finish


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

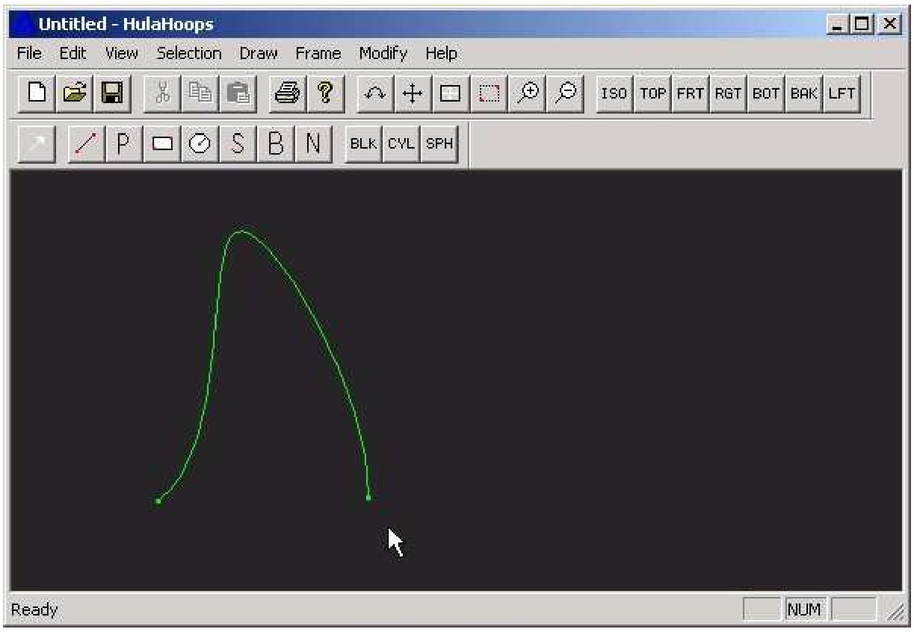
- •Left click to enter start tangent
- • Right click for NULL (natural tangent with zero curvature)


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

- •Left click to enter end tangent
- • Right click for NULL (natural tangent with zero curvature)

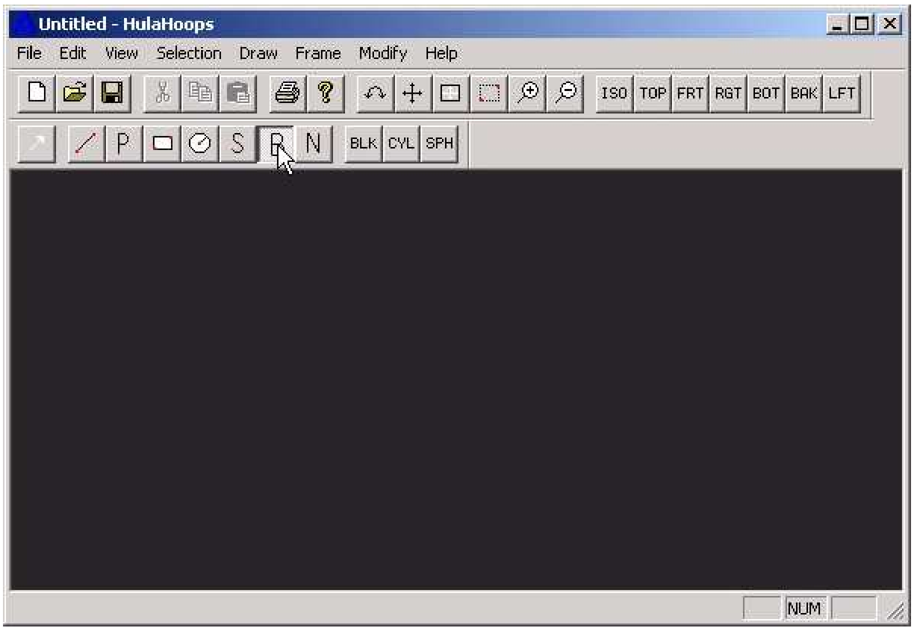

Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

- •Here is the result
- • Note that this Spline is planar in the X-Y plane because the default view is TOP

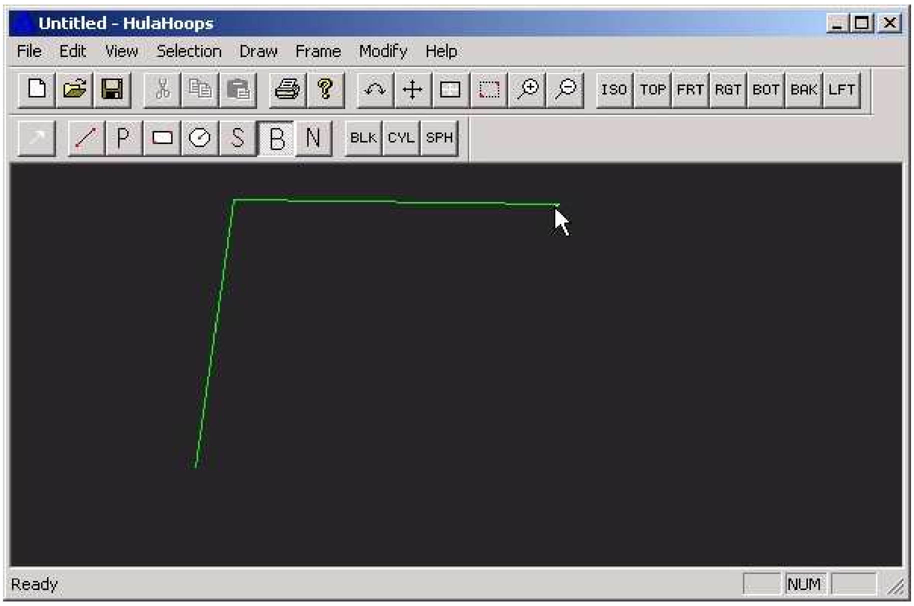

Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N
BLK CYL SPH
Ready
NUM

8.2 Draw a 3D Bézier curve

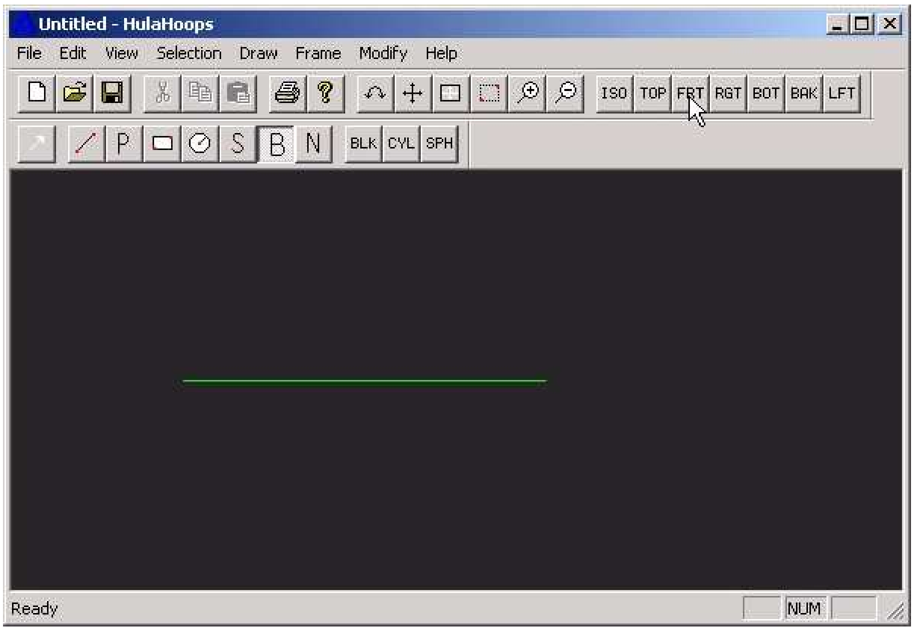
- •Click on 'B' icon or
- • Menu 'Draw' 'Curve' 'Bezier'


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
NUM

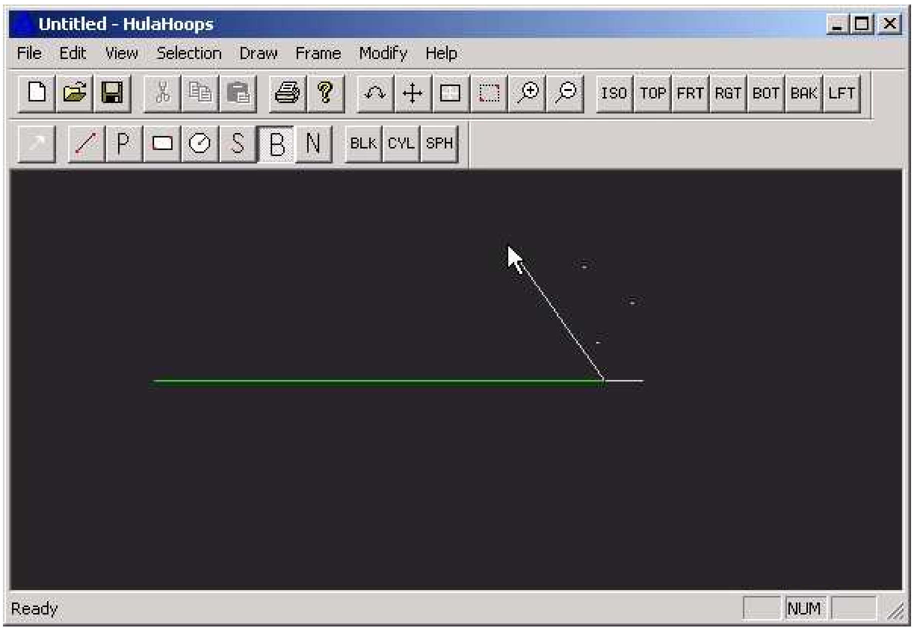
- ·Left clicks to input 3 of 4 polygon points


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

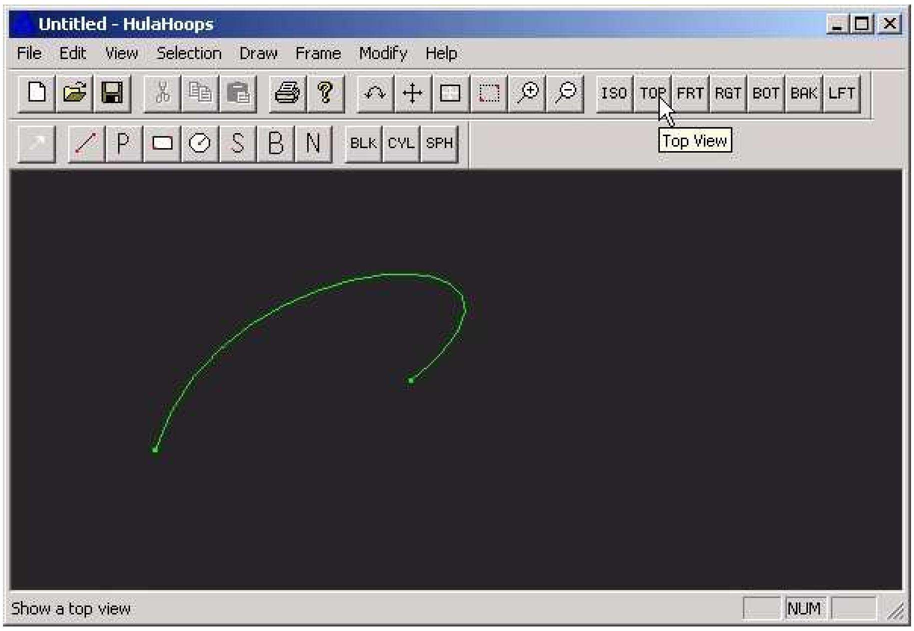
- •Change to FRONT view: click on FRT icon


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

- •Input 4th polygon point in the Z-X plane

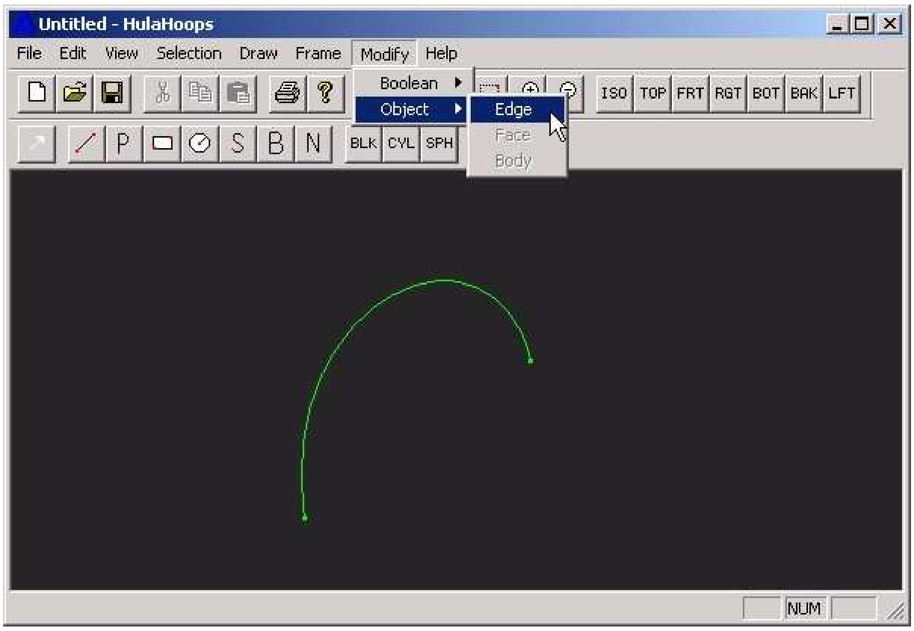

Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

- •That's done it
- • Go back to TOP view

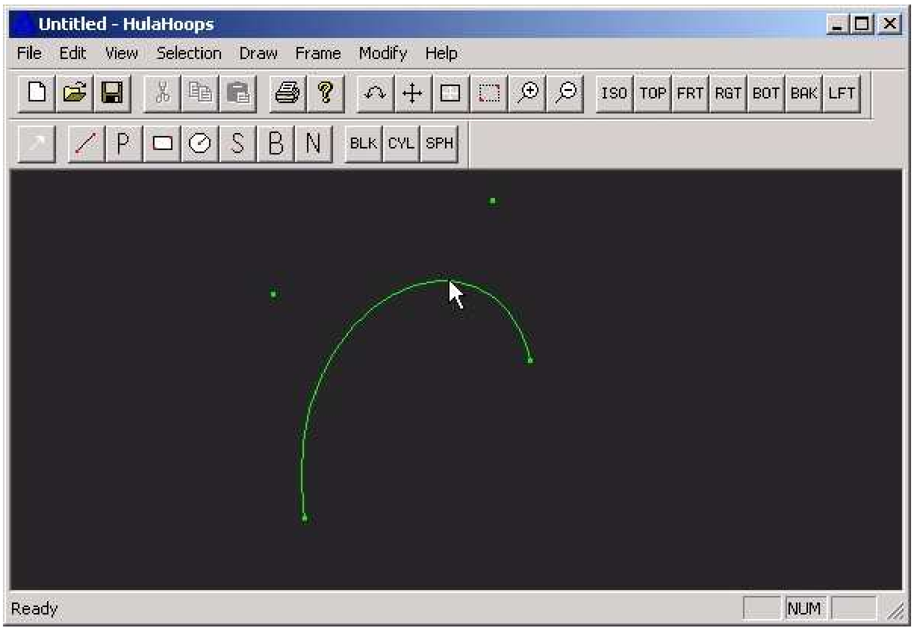

Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Top View
Show a top view
NUM

8.3 Shape Modification

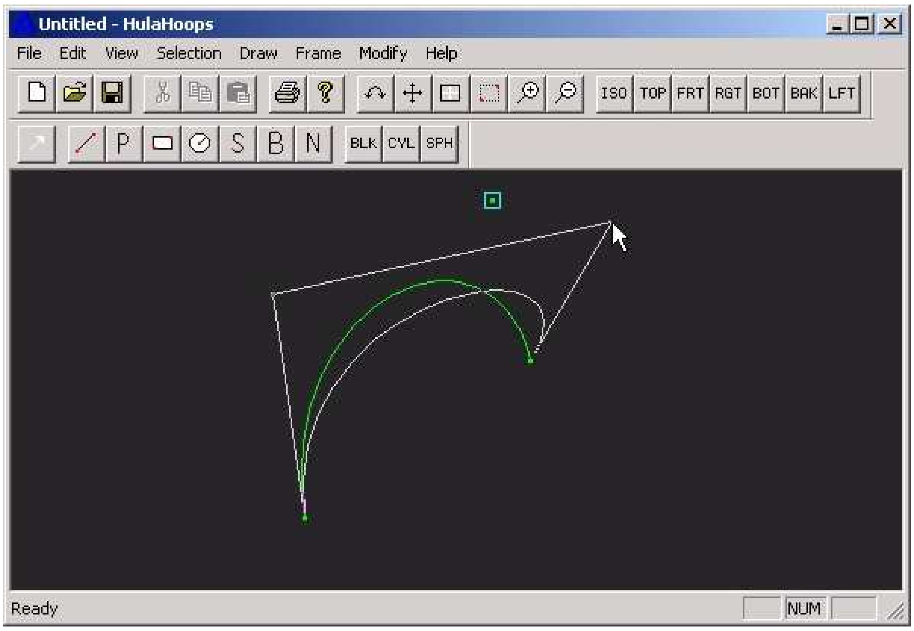
- Menu ‘Modify’ ‘Edge’

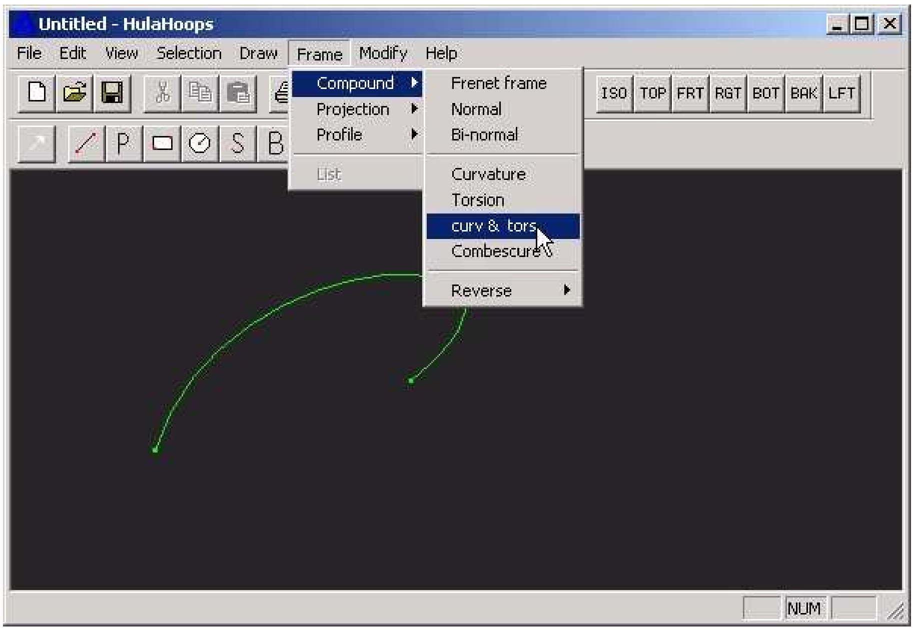

Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
Boolean
Object
Edge
Face
Body
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
NUM

- •Left click to select a curve
- • The control points now appear on screen


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

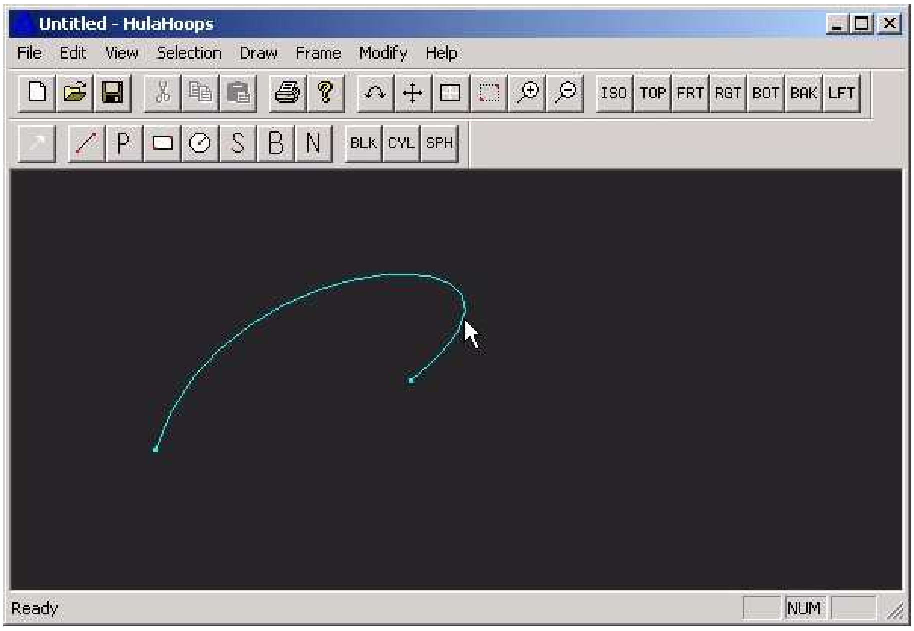
- •Left button down to drag a point
- •The Bézier polygon shows colour white
- •Right click to exit the modification tool


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N
BLK CYL SPH
Ready
NUM

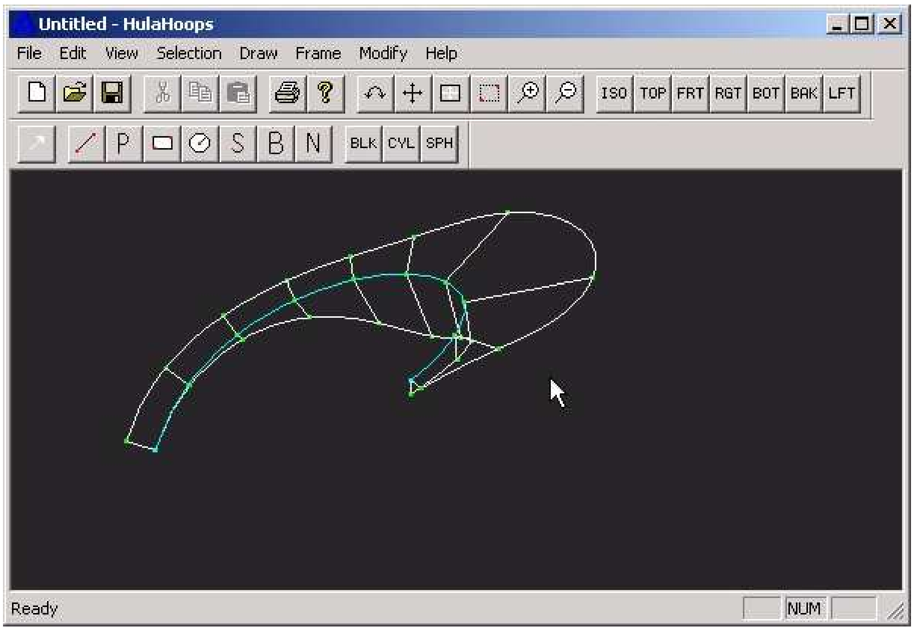


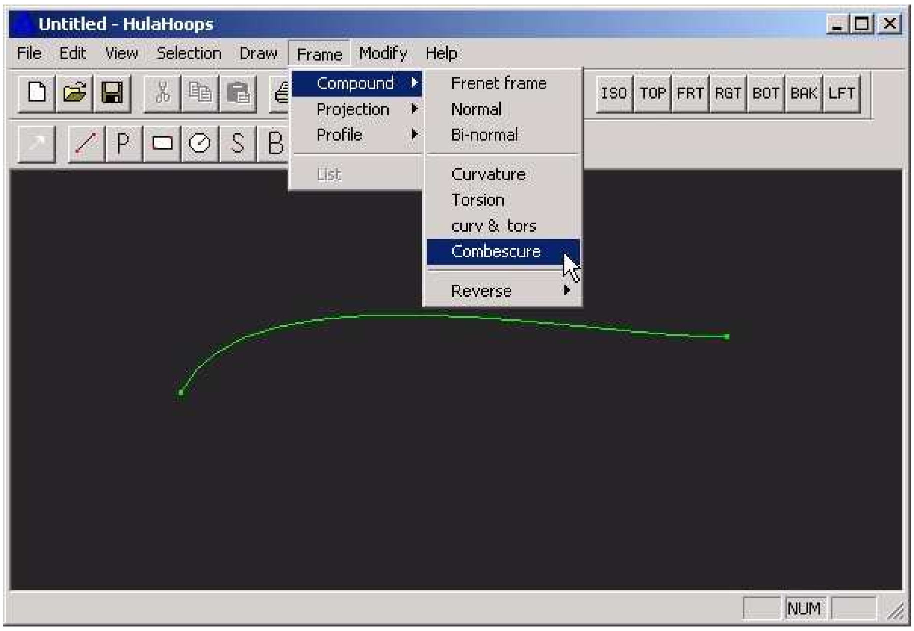
8.4 Curvature and Torsion Differential PropertiesMenu ‘Frame’ ‘ Compound’ ‘curv & torsion’

- •Menu ‘Frame’ ‘ Compound’ ‘curv & torsion’
- • Left click to select a curve, it should go blue



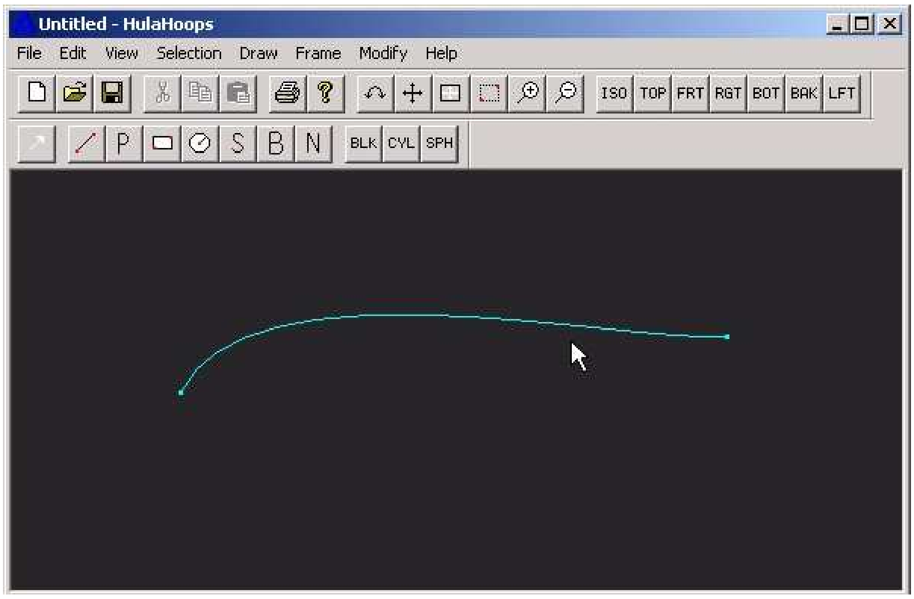
- Right click to validate
- The curvature and torsion compound fins are now on screen
- Note that if the inquired curve was planar, only the curvature would be returned


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

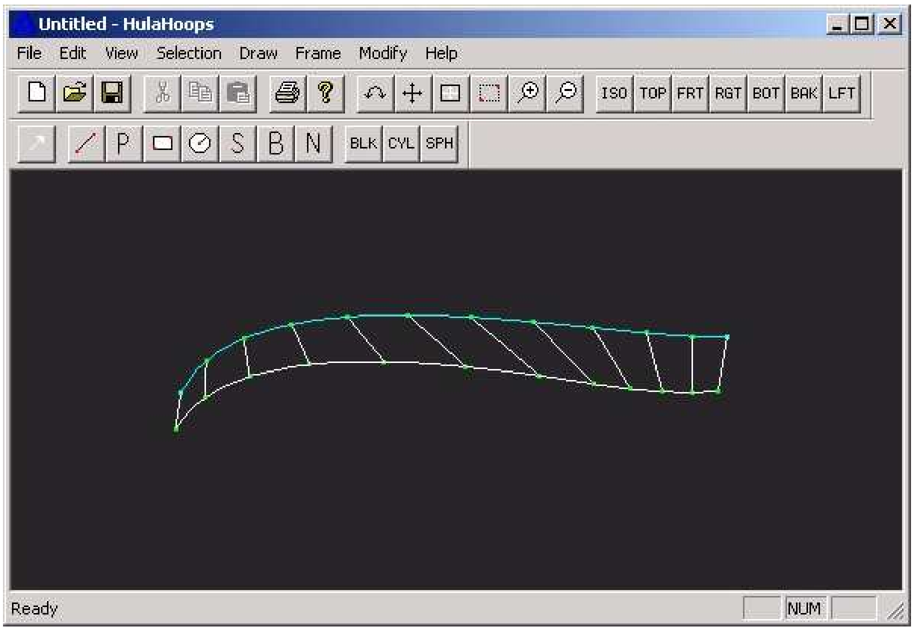


8.5 Combescure Differential Properties Menu ‘Frame’ Compound’ ‘Combescure’

- Select a curve, it should go blue

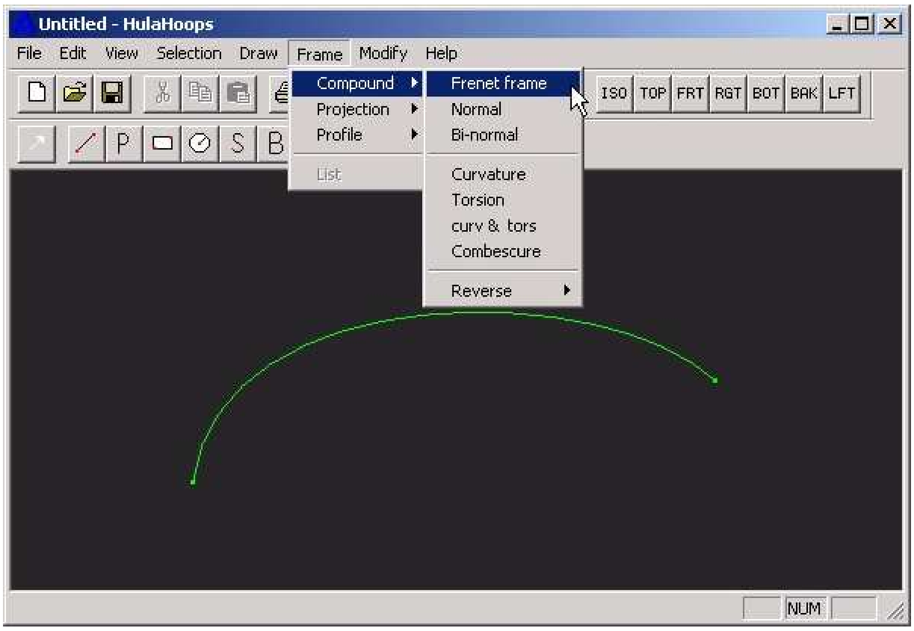

Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH

- Right click to confirm

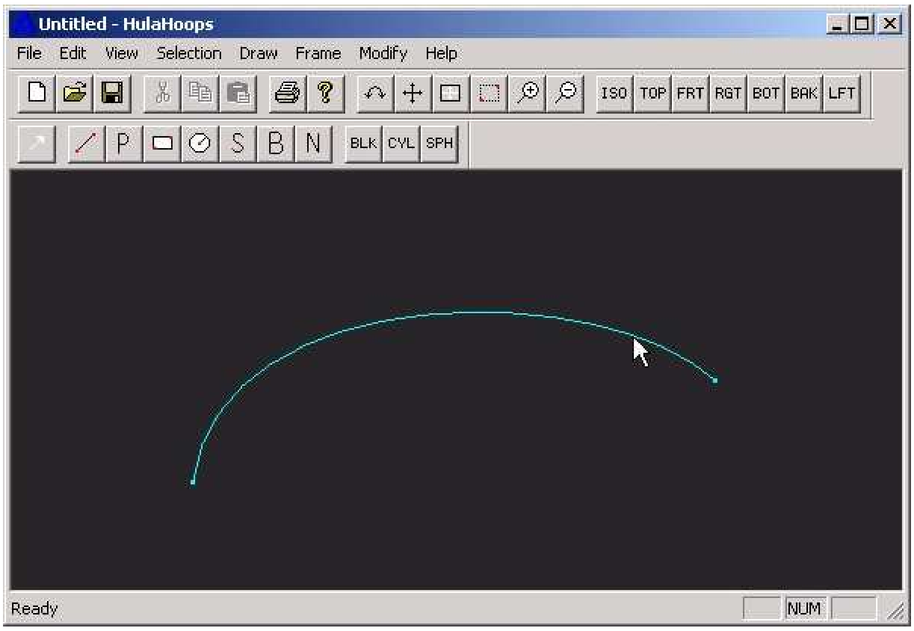

Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

8.6 Frénet Frame Differential Properties

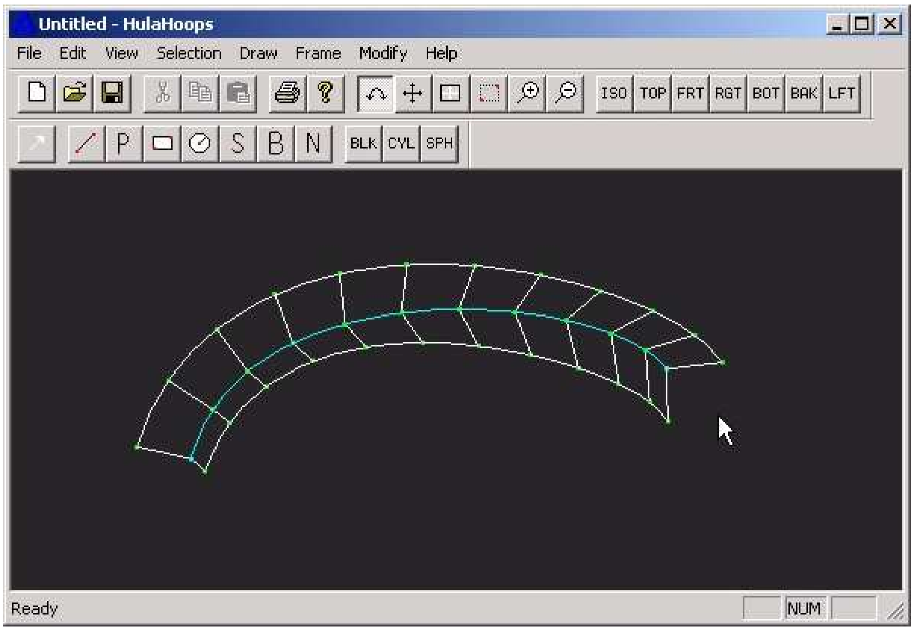
- Menu ‘Frame ’ Compound’ ‘Frenet frame’
- •Left click to select a curve, it should go blue



- • Right click to validate

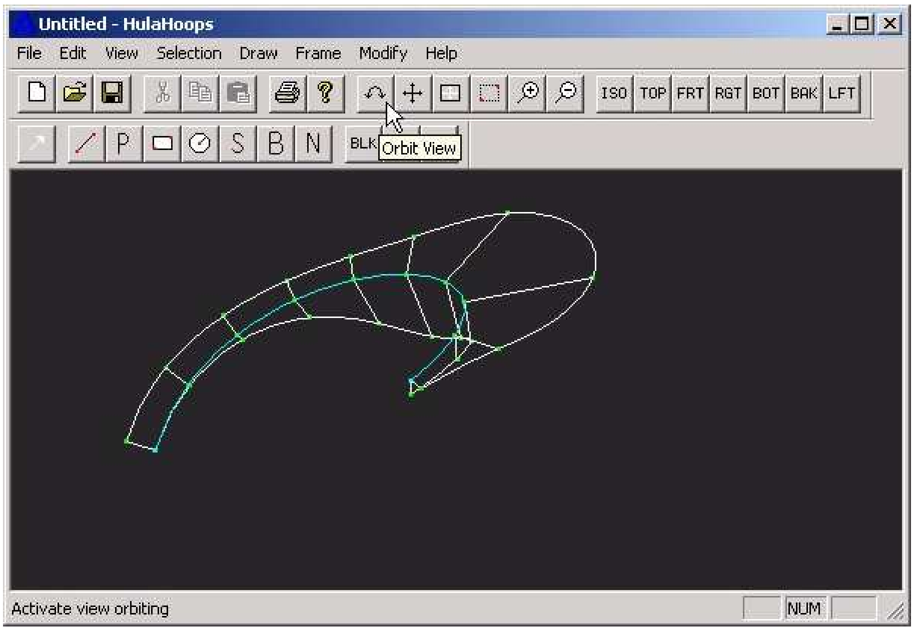

Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

- The Frenet frame normal and bi-normal compound fins show on screen
- If the curve is planar, only the normal fin is given
- The normal and bi-normal fins can be obtained separately from Menu 'Frame' 'Compound' 'Normal' or 'Bi-normal'
- The Frenet frame is valid for any curves with non vanishing curvature
- Right click to exit the tool, the fins and hair are destroyed


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CYL SPH
Ready
NUM

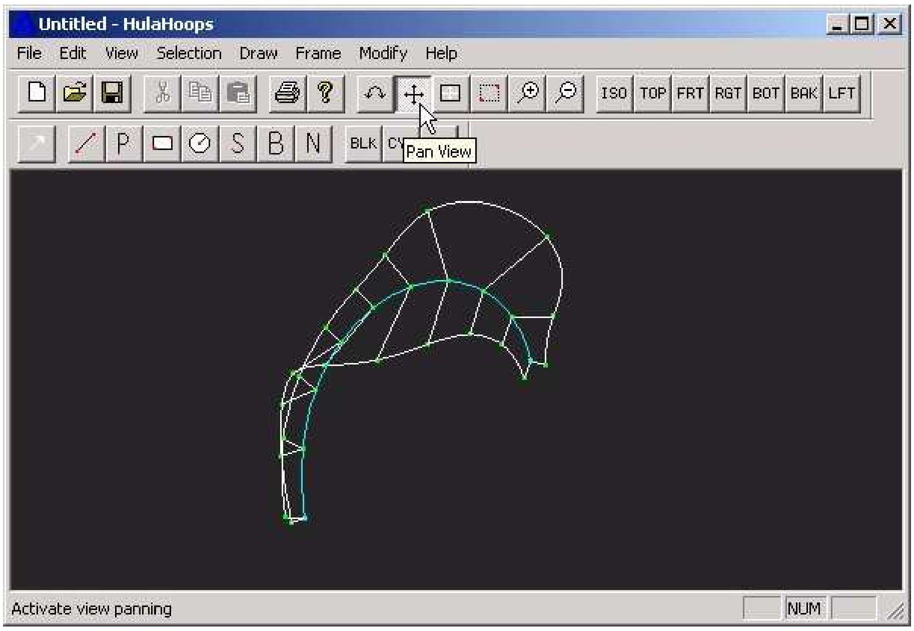
8.7 Viewing

- Menu ‘View’ ‘Orbit’ or
- Bent arrow icon
- Left button down to rotate the model
- Right click to exit the tool


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK
Orbit View
NUM

Activate view orbiting

- •Menu ‘View’ ‘Pan’ or cross arrow icon
- • Left button down to move the model in the view plane
- • Right click to exit the tool


Untitled - HulaHoops
File Edit View Selection Draw Frame Modify Help
ISO TOP FRT RGT BOT BAK LFT
P S B N BLK CV
Pan View

Activate view panning