Concepts and Solutions for Efficient Handling of the Digital Ink

Khaireel A. Mohamed

Dissertation zur Erlangung des Doktorgrades (Dr.-Ing.)
Technische Fakultät, Albert-Ludwigs-Universität Freiburg

Dekan: Prof. Dr. Hans Zappe

Referenten: Prof. Dr. Thomas Ottmann (1. Gutachter)
Prof. Dr. Heinrich Müller (2. Gutachter)

Datum der Disputation: 25. März 2009

To the loving memory of my late father.

Ahmad Bambong
(1945 – 2007)

To my most special friend and confidant,
who stuck steadfastly by me through thick and thin,
whose strong words of encouragement ring honest and true.

An inspiration to all things positive.

A most respectable and reliable man.
A brother who I love and fully trust.

Tobias Langner

"The proper office of a friend is to side with you when you are in the wrong.
Nearly anybody will side with you when you are in the right."
Mark Twain, 1898.

Zusammenfassung

Wir präsentieren in dieser Arbeit einen neuartigen Ansatz zum Glätten digitaler Handschrift und zum Rendern der daraus resultierenden mathematischen Kurven. Dieser Prozess kann bereits durchgeführt werden, bevor der komplette Digitalisierungsvorgang auf dem Transducer (signalgebendes Eingabegerät) abgeschlossen ist. Die Vorteile dieses Verfahrens ergeben sich in zweierlei Hinsicht: Einerseits wird die visuelle Rückmeldung während des Schreibens aufrechterhalten, ohne dass charakteristische Merkmale der Schrift wie der Stil und das Flair aufgrund der Natur der Digitalisierung in Mitleidenschaft gezogen werden. Andererseits – und dies ist viel entscheidender – ermöglicht die Verwendung der von uns konzipierten sehr effizienten Lösungen, die verfügbaren Rechnerressourcen für andere rechenaufwändige Aspekte einer multimodalen Anwendung zu verwenden.

Im ersten Teil dieser Arbeit illustrieren wir verschiedene Konzepte, welche verwendet werden können, um eine gegebene Menge von Punkten in der Ebene mittels sorgfältig platzierter Kurven zu verbinden. Bei der Nutzung klassischer Interpolationsmethoden nimmt man an, dass die Punkte einer wohldefinierten, zweifach differenzierbaren Funktion mit bestimmtem Grad entstammen. Im Verlaufe dieser Arbeit werden wir schließlich diese Annahme fallen lassen und unsere eigenen Ansätze miteinbeziehen, die es ermöglichen, durch eine Kombination von Interpolation und Regression eine Menge von unregelmäßig verteilten Punkten mit einer *wünschenswerten* Näherung der ursprünglichen Kurve zu verbinden (die hier genannte wünschenswerte Näherung genügt dabei dem Approximierungssatz von Weierstrass). Eine solche unregelmäßige Verteilung der Punkte ist schließlich eine der inhärenten Eigenschaften einer handgeschriebenen Spur, wie sie von einem Transducer geliefert wird. Um die Gültigkeit unseres Ansatzes und die Korrektheit der Behauptungen nachzuweisen, zeigen wir, dass die geglätteten Versionen der Spuren ausschließlich durch Abschnitte konischer Funktionen, genauer gesagt Ellipsen, nachgebildet werden können, die an den Endpunkten jeweils harmonisch ineinander übergehen. Diese charakteristischen Eigenschaften, die eine sehr aufwändige Berechnung mit sich bringen, veranlassten uns dazu, verschiedene signifikante Aspekte quadratischer Splines zu untersuchen. Dabei entdeckten wir, dass anstatt einer linearen Anzahl von Berechnungen pro Punkt (mit mindestens vier Quadratwurzeloperationen pro Berechnung) eine konstante Anzahl genügt. Diese Ergebnisse begründen schließlich das Herzstück unseres *Active-Smoothing*-Algorithmus'.

Im zweiten Teil erläutern wir die Struktur von handgeschriebenen Spuren im Hinblick auf deren wesentliche Eigenschaften, die wiederum die Grundla-

gen der Erkennung von *Gesten-Befehlen* darstellen. Ein Gesten-Befehl ist das Ergebnis einer gestenartigen Eingabe mit dem Stift, die dem Zeichnen einer speziellen Form ähnelt, und dem Ausführen der mit dieser Geste verbundenen vordefinierten Befehle.

Bezieht man die zeitlichen Komponente einer Spur mit ein, sind wir mit unseren Techniken in der Lage, aus den räumlichen und zeitlichen Eigenschaften der Eingabe auf einer einzigen Schreibfläche zwischen Handschrift (und Zeichnungen) und Gesten zu unterscheiden. Die räumlichen Eigenschaften von Gesten weisen im Bezug auf den Schwerpunkt auffällige Eigenschaften auf, die eine – von uns als *Disorientierung* bezeichnete – räumliche Zuordnung der Gesten ermöglichen. Das bedeutet nun, dass man bei der Verwendung eines Tabletops mittels eines linearen Diskriminators feststellen kann, von welcher Seite des Gerätes eine bestimmte Geste ausgeführt wurde – ohne die Verwendung zusätzlicher externer Tracking-Mechanismen.

Die Anwendung der Ergebnisse beider Teile ermöglicht es uns somit nicht nur mit digitaler Tinte effizient umzugehen, sondern auch darüberhinaus in intuitiver Weise mit der Arbeitsumgebung zu interagieren.

Abstract

We present a novel approach that smooths digital handwritings and renders the resultant high quality, symbolically represented curves, while the primitive-resolution sampling process from the transducer device is still ongoing. The repercussion from this is two-fold; firstly, the visual impact while *writing*, is sustained, without compromising on the integrity of the 'style' and 'flair', and any of the other features within, that may be lost to the rigid and perfunctory sampling routine between the hardware and the software. And, more importantly, the second repercussion, through the conceived efficiencies of our combined and simplified underlying methods together, alleviates on what limited resources that are available and enables other routines handling other computationally demanding aspects of a multi-modal application access to more processor time.

The first part of the thesis canvases the various concepts of passing a set of well-placed curves through a set of given points on the 2D plane. The classical interpolation methods assume that the curves are twice differentiable at every single point, and that the points are part of an unknown but well-defined mathematical function of a certain degree. As we progress through the thesis, we eventually lift this assumption and incorporate our techniques, which includes a combination of interpolation and regression methodologies, to *desirably* estimate a set of indeterministically scattered points – which is an apparent and inherent characteristic of a typical handwritten trace that comes out of the transducer device. A desirable estimate is one that adheres to the Weierstrass approximation theorem. To prove the validity of our approach and the correctness of our propositions, we show that the smoothed versions of these traces can entirely be made up of conic sections, particularly the ellipses, joined together at their endpoints in confluent harmony. These formative and computationally expensive grounds, when viewed collectively, led us to discover certain imperative aspects of the quadratic spline curves that reduce linear time calculations per input point, involving at least four square-root operations per calculation, down to a constant. Essentially, these propositions are what constitute our *active-smoothing* algorithm. The symbolically represented curves and the delayed rendering procedure ensure that we maintain linear space storage complexity with respect to the number of original points in the trace. Our results show that this form of representation has no adverse effects on random access navigation, which is a process of "active visible scrolling".

In the second part, we expound the composition of handwritten traces in observance of their intrinsic features that are the fundamentals of *gesture command* recognition. A gesture command is the result of invoking gesture-like movements with the pen, that resemble special shapes drawn in a certain distinctive way, recognised as certain predefined instructions that are to be carried out. We show that our temporal techniques applied on the traces and involving their spatial features can distinguish writings (and drawings) apart from gesturings, on a common, non-segregated ink environment. The spatial features, we found, when concentrated about their centre of gravities, exhibited conspicuous properties that allow, what we term as, the '*dis*-orientation' of the gestures; that is, when exploiting the tabletop environment, the linear discriminator is able to determine whether a gesture came from a person who is seated at the northern, southern, eastern, or western edge of the table, without the help of external tracking devices.

With all the ramifications of the results from both these parts combined, we not only get the solutions necessary to efficiently handle the digital ink, but also to consistently interact with it in its appropriate environment.

Keywords: Digital ink, active-smoothing, spline curves, pen gestures
CR Categories: A.2, I.7.2

Acknowledgements

When I began putting my thoughts to words, and my words into programs, graphs, figures, and tables for this thesis, I was pleasantly surprised to find so many people who were willing to help me out in whatever way they can – to ensure that I finish this piece of work properly and completely. To all those people, I thank you for your effort, your concern, and your kind help. However, there are several among you who I must exclusively mention by name.

First off, my two best mates, Tobias Langner and Marius (*Militarius Victorius Tobascus*) Heinzmann, without whom, my German experience will never be complete. Tobes, for your dedicated support and tonnes of suggestions on how to make this document – its ideas, phrases, and figures – better, from start to finish. Marius, for plowing through every single word in this thesis and for offering numerous hints for improvements – know that your witty sense of humour and wise words kept me going strong (and laughing)!

To my colleagues at the Chair of Algorithms and Data Structures – Christoph Hermann, Christine Maindorfer, Martina Welte, Tobias Lauer, and Frank Dal-Ri – thanks for all your inputs. Also not forgetting Marco Schulze for taking time to patiently read through this strenuously long piece of dissertation – thank you for all your "encrypted" comments. To the fair lady Karen Tso-Sutter, never once had I had an uninteresting conversation with you about everything and anything.

Most important of all, I want to express my most sincere and deepest gratitude to my *Doktorvater* Prof. Dr. Thomas Ottmann for giving me this opportunity to improve my mental capabilities, to exploit my talents in research, and to have the privilege of working alongside with you. So much I have learnt, from so wise a person.

And last but not least, to Prof. Amitava Datta, for having faith in my abilities since my Honours year at The University of Western Australia.

Contents

Part I

Preface

CHAPTER 1

Introduction

The term *digital ink* refers to the technology that digitally represents handwriting in its natural form. Simply put, the traces of handwritten marks left on the screen are emulated by the system with a series of connected geometric dots and line-segments to represent the imprints; none of which are parsed as characters or other recognisable symbols. In a typical digital ink system, a transducer device such as digital tablets and whiteboard screens are equipped with electromagnetic fields that can capture the movement of a special-purpose pen, or stylus, and record the movement on the LCD screen. The recorded handwriting can then be saved as a handwritten document or converted to text (or to whatever appropriate shape the ink may represent) using several popular state-of-the-art recognition technologies [53, 42, 21].

What follows immediately are the techniques to manipulate the digital ink data, and seamlessly integrate or relate them to other digital documents and applications. This is collectively referred to as an *annotation* process, a combination of routines that include the automatic and manual grouping of digital ink strokes within a document, classifying the annotations according to their types, and anchoring the annotations to appropriate regions or positions in a document [47, 28]. The process may further include 'reflowing' the annotations in a new document layout so that the annotations conform and adapt to the new layout while preserving the original intentions and meanings of the annotations [49].

Specifically, this thesis examines the manipulation of ink data – the rudimentary raw sampled points from the transducer device – and proposes several methods to improve and maintain the quality of the rendered traces on annotated documents. The methods constitute our smoothing technique, one that is similar to many curve fitting and approximating procedures, but differ in the main approach and in the resultant efficiency. The latter matters most when we turn the offline algorithm into an online *active-smoothing* solution, that continuously smooths a digital ink annotation while the document is still being annotated.

1.1 The Digital Ink Metaphor

The visible trails left behind by handmade expressions of markings, strokes, and lines are the basic composites of what we describe as *traces*. And when these expressions are applied directly on a receptive surface of a digital transducer device by using a digital pen (and sometimes by merely touching the device with fingers), or indirectly by dragging the mouse to simulate controlled movements of a pen on a screen-canvas, we have a 'pen-base interface' that captures digital ink traces.

Fundamentally, digital ink traces are a finite set of sampled points that the hardware device can afford to provide its device-driver. The number of points supplemented is directly proportional to the resolution of the sampling frequency of the hardware, and are obtained from a pair of successive *pen-down* and *pen-up* events. These xy-coordinate points are then relayed immediately to domain-specific application softwares as a fully formed 'Trace' – we denote the term Trace with a capital 'T' to refer to the actual geometric data structure that contains a set of (x, y) points. A sequence of these Traces when accumulated and presented properly can form meaningful graphics of what we (humans) perceive as characters, words, drawings, or commands. The softwares may then decide to process and archive them as ink documents, notes, and messages for later retrieval and exchanges through telecommunications means, or interpret them as appropriate computer actions.

The obvious property of a Trace, when observed on a 2D plane, is its spatial relationship between the Trace and the other components on the plane (including the plane itself), as well as between the Trace and other Traces. The less obvious property is the Trace's temporal relationship to these same components. From our experiences with various hardware devices, we found that it is advantageous to include timestamp information in addition to each sampled xy-coordinate point. The importance of which are further explained in later chapters when we discuss the temporal issues concerning the digital ink Traces, which are used to solve grouping problems, feature identification, automatic domain segregation, and random-access search and replay.

The Consortium of the World Wide Web (W3C) has been (and is still) working on a standard to universally mark-up the digital ink traces for both 'archival' and 'streaming' modes that are compatible for all applications relying on web-based (XML schema) data sets. Based on the consortium's latest working draft [108] with a Request for Comments (RFC) option that is still active to date, the proposed W3C standard, known as InkML, keeps the notion of a 'time-interval' in the overhead metadata with the assumption that the transducer device used to capture each xy-coordinate point is sampling at a constant and fixed rate. This assumption makes it redundant to store the timestamp for each xy-coordinate point and saves about a third of the overall file size. This is acceptable if the resulting marked up InkML is final, where no further changes to the data set needs to be done.

However, in the case of our study, we are using the raw InkML representation to satisfy various objectives that may involve further post-processing actions. These include computing geometrical solutions to reduce and compress the digital ink Traces, or to add superfluous points to smooth the Traces at finer granularities for representation at higher resolutions. Either of which, when applied to the original ink Traces and then passed through a digital-ink-player facility, will make the reanimated ink expressions appear awkward and unnatural when replayed in real time. Furthermore, if we are to allow cross-platform and cross-device functionalities where we can create Traces using the digital pen (on the digital screen at a fixed sampling rate), or by using the mouse (where the sampling rate varies), then we must incorporate the additional timestamps to ensure that the integrity of the post-processed trace data is maintained.

1.2 Object-Oriented Derivations

In its simplest form, a Trace T is a set of $\{(x_i, y_i, t_i)\}$ tuples, deduced directly from each complete pair of pen-down and pen-up events made up of the xy-coordinate points with attached timestamps t, for $1 \leq i \leq n$, where n is the total number of points sampled. All Traces are considered unique and are distinctively identifiable by their identities. A Trace's ancestor is the (Java) `Path2D` object, which implements the `Shape` interface. Together, they represent a form of geometric shape, mainly incorporating two elements; the bounding box of the geometry, and a `PathIterator` object that allows the traversal of the tuples within the Trace, which describes the trajectory *path* of the `Shape`'s outline. It follows that a Trace has $n - 1$ line-segments.

1.3 Spatial Indexing

Traces coming in from the `InkEnvironment` are usually stored in the temporal sequence that they arrive in, in the most basic of circumstances. However, due to the inherently spatial nature of the appearance of all Traces on a 2D canvas, we found more advantages when they are rearranged with respect to the horizontal and vertical axes of the canvas on the screen-dimension. For example, an array of Traces on the 2D plane is first stored in a common R-Tree structure [40] and is later indexed into further sub-groups based on the Trace-group constraints if and when the number of Traces becomes too large. The R-Tree structure ensures efficient spatial representation (e.g. see Figure 1.1). It also provides better search times of $O(\log_m n + k)$ for a given rectangular range, where n is the total number of Traces on the plane, k is the number of Traces to report that are inside the range, and m is the maximum branching factor of the tree, compared to the temporal-list order, especially when interacting with users on the spatial domain.

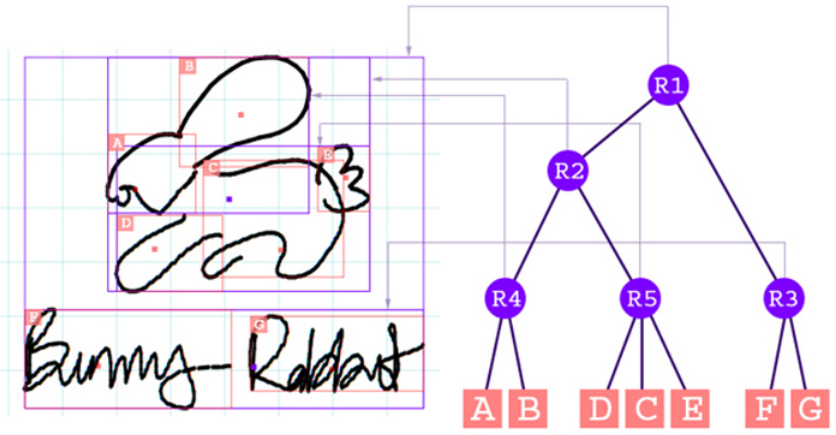


Figure 1.1: Spatial representation of Traces using the R-Tree structure.

We obtain the bounding rectangle of a single Trace entity from its Trace class representation T_i, which returns the top-left and bottom-right coordinates, $(\min_x\{T_i\}, \min_y\{T_i\})$ and $(\max_x\{T_i\}, \max_y\{T_i\})$, respectively. These information are then used as parameters to guide with the building of the R-Tree representation for all Traces $T_1, T_2, \ldots, T_n$ on the `InkEnvironment`. Each node in the R-Tree corresponds to the smallest bounding rectangle that encloses its child nodes. If a Trace is spatially contained in several nodes, it is only stored in one node. The tree parameters ensure that the following properties are met at all times:

- The depth of the overall R-Tree is balanced (similar to a B-Tree structure [18]) and is as shallow as possible;
- Indexing of Traces is completely dynamic, that is, insertions and deletions can be intermixed with searches; and
- No periodic reorganisation is required.

1.4 Temporal Indexing and InkML

Temporal indexing is carried out when we no longer need to interact with the Traces on the `InkEnvironment`, that is, when we are satisfied with what is in the (final) percept of the `InkEnvironment`; or whenever we require further sub-groupings of the current Traces to distinguish between freehand 'drawings' (TraceGroup type-I) and freehand 'writings' (TraceGroup type-II). Deploying the temporal indexing allows us to proceed with the archival of the digital ink data (to file) and/or stream it over a network as dictated and specified by the InkML standard.

This part strengthens the non-(user)interactive digital ink manipulation as compared to the R-Tree structure, which includes saving, retrieving, replaying, and streaming. Figure 1.2 gives a graphical overview of the same Trace components in Figure 1.1 when aligned to the time axis. We mention here that the Trace components on the time domain do not overlap each other. This is because we have only catered for 'single-user-input' facility that distinctly allows for only one pen (or mouse) input at one time instant on the `InkEnvironment`. However, suppose that we accord a 'multi-users-input' facility, and that the transducer device is able to support this facility, then we can expect that the Trace components to overlap at certain time instances for arbitrary periods. In this case, we will need additional augmentation on the primary data structure to support simultaneous and concurrent Trace entries. In this study however, we will not pursue the idea for 'multi-users-input' facility.

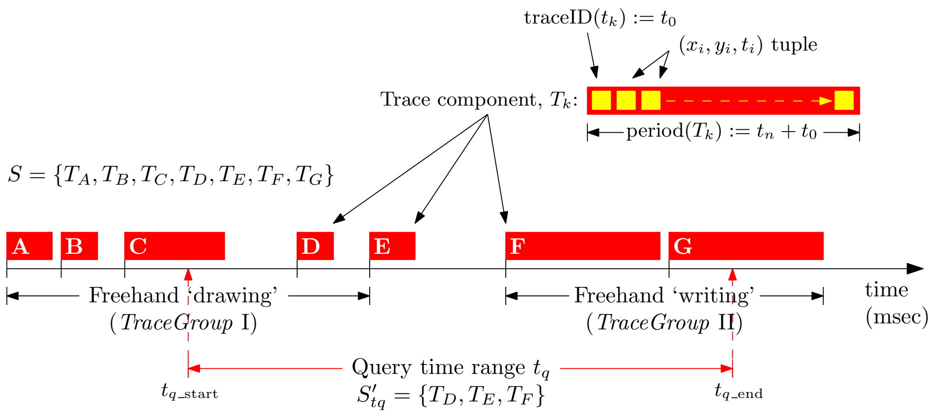

traceID(t_k) := t_0
(x_i, y_i, t_i) tuple
Trace component, T_k:
period(T_k) := $t_n + t_0$
$S = \{T_A, T_B, T_C, T_D, T_E, T_F, T_G\}$
A
B
C
D
E
F
G
Freehand 'drawing' (*TraceGroup* I)
Freehand 'writing' (*TraceGroup* II)
time (msec)
Query time range t_q
$S'_{tq} = \{T_D, T_E, T_F\}$
t_{q_start}
t_{q_end}

Figure 1.2: Temporal representation of the same Trace components in Figure 1.1 on the timeline.

Given a query time-range $t_q = [t_{q\_start}, t_{q\_end}]$, we retrieve all Traces from the R-Tree whose *traceID*s fall within the given range, and whose period after its *traceID* does not exceed t_{q_end}. This resulting S'_{t_q} contains Trace components T_k such that $t_{q\_start} \leq traceID(T_k) \leq tq\_end$ and $traceID(T_k)+ \ period(T_k) \leq tq\_end$. The T_k in S are arranged in ascending order of their *traceID*s, and are immediately ready to be used for the activities described above.

Granted that when the number of Trace components becomes too large, the search time may take too long on the main R-Tree and degenerates into linear time complexity of $O(n)$. One possibility to reduce this search time is to keep another separate array to store copies of the original Trace components in sorted time-order, and apply binary search (of $O(\log n)$ complexity) on it for the given query time-range. This doubles the memory space needed to maintain an ink application to support the `InkEnvironment`.

1.5 Digital Ink Domains

Generally, the way in which people interact with any kind of Information and Communication Technology (ICT) is influenced by two aspects; input devices and software design. Firstly, the hardware of input devices specify what information is exchanged between man and machine and how this exchange takes place. For example, by moving a mouse, a user sends 'events' to a computer which encodes information such as relative coordinates representing mouse movements. The second aspect is the software design of the user interface and the way in which the electrical signals from the hardware are interpreted. Incoming signals from mouse movements (representing discrete relative coordinates) are generally mapped to absolute pointer movements on the screen and as such, visualized to the user.

Over the years we have come to accept the fact that graphical user interfaces are not only visually helpful, but optimal in almost every way we can imagine for users of the desktop computers. The designs of which are tailored specifically for user-oriented goals such as completing that word document with style, enhancing the artwork for that poster, searching and getting information from the web, or even controlling that regional train tracks to follow schedule. Such diversity in the range of applications used for the desktops triggered sharp interests in both the Intelligent User Interface (IUI) and Human-Computer Interaction (HCI) communities to continue enhancing and providing guidelines for future interfacing trends. Where it seems that the former community favours *adaptivity* concepts and the latter *adaptability*, we have also seen in the recent years the synthesis of the two [11]. We shall further dwell into these two concepts, as well as one other, known simply as *perceptual*, in chapter 12 when we discuss the distinction between writing, drawing, and gesturing.

1.5.1 Touch Screens and Tablets

People first began using touch screens and graphics tablets in the mid 1970s, after the successful filing of two successive US patents by Hurst and Parks [45, 46] on their electrographic inventions. Essentially, they include the discoveries of the 'graphic data transcription system', the 'graphical input system', and the 'graphical input table' [67, 66, 9]. The improved and combined technologies as we know of today allow for the seamless communication between humans and computers by the direct touching of the screen, or alternatively, through the use of digital pens (sometimes also known as *styluses*).

While these natural input modalities provide some 'improvements' in several situations where they are deployed, particularly in classrooms and collaborative workplace scenarios, they still lacked in terms of usability if coordination with other modalities such as the keyboard and mouse are not taken into consideration. There are surveys compiled by McKay that show that touch screens are hard to write on when the monitor is standing upright [76]. Ideally, the screen should be mounted horizontally. However, touch screens do not only react to pen input but to any kind of physical contacts, thus making it impossible for users to rest their hands on the surface while writing.

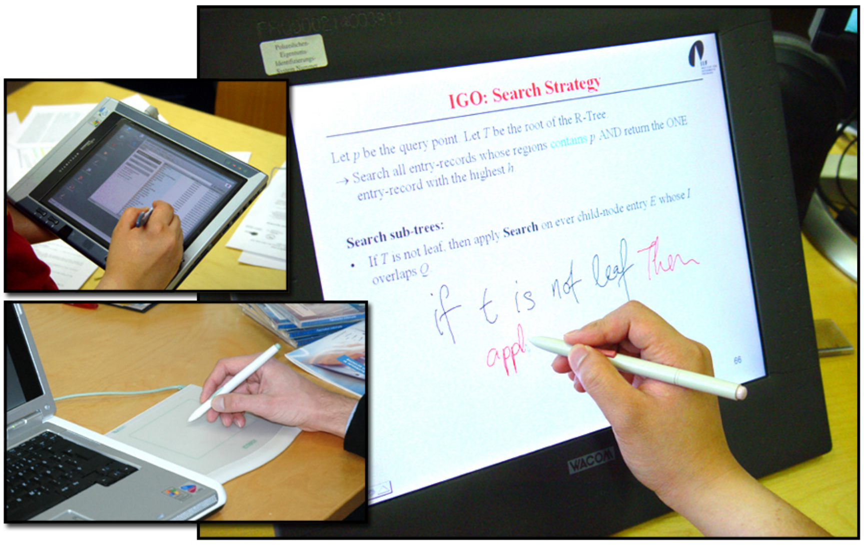


Figure 1.3: Examples of pen-input devices. From top-left, anti-clockwise: Tablet PC, A WACOM interactive LCD panel, and a Penabled$^{\mathrm{TM}}$pad and pen devices.

Originally targeted toward special interest groups and professions the first LCD displays with integrated tablet, and was later on known as interactive LCD panels, became available for the average consumer by the end of the 90s. These devices operate like regular graphics tablets described before with the main difference being that the tablet is transparent and mounted on top of an LCD panel. Hence, compared to common touch screens, the surface does not react to any kind of contact but only to the input of a special pen, thus enabling presenters to naturally write on a horizontally mounted screen. Using the pen, people can directly interact with the applications by making annotations on the presented slides, navigating through subsequent slides, etc. These early versions often proved to be too small, had low resolutions, and limited processing powers which resulted in a noticeable time delay during freehand writing. Nowadays, we observe significant advancements in display size, screen resolution, as well as processing power, paired with lower prices. Hence, when mounted horizontally such devices provide the same ease of use as traditional overhead projectors while at the same time enabling users to access and use the full functionality offered by the underlying ICT installation.

1.5.2 Wall-Mounted Digital Boards

The abundant writing surface allowed by huge wall-mounted digital boards (see Figure 1.4) is harder to perceive for user-machine interactions as compared to the comfortably sized tablets and office whiteboards. Freehand writings are an entity on their own, containing the flair and style of expressions of the writer, which do not necessarily follow an ordered sequence or format when ample space is provided. To date we see a number of digital ink-based studies that concentrate on managing contents and editing 'informally', which came about as a result of this entity. Whether it is grouping a set of freehand words into a continuous flow of sentence, or combining a set of drawings to form the physics of represented objects, the programs rely on a lot of command buttons in toolboxes and pull-down menus to affect their causes. And since all of these applications are tailored for the desktop setup (including the tablets), the authors need not mention any proximity problems. Try putting these same applications on the huge wall-mounted digital boards and we can expect users to complain about its lack of suitable interface.

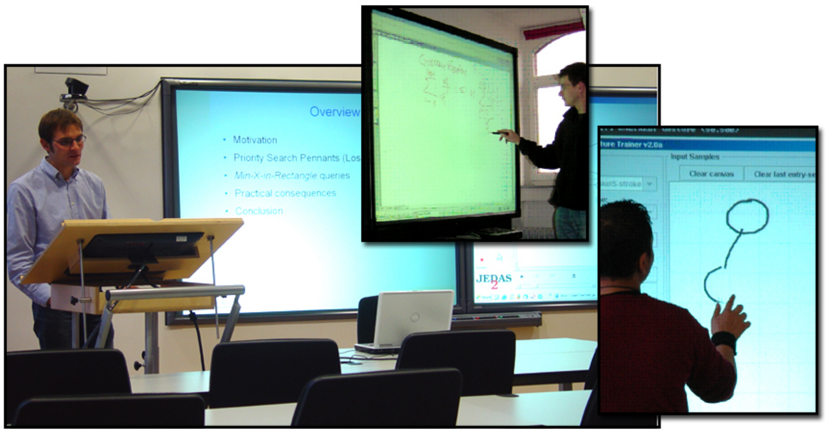


Figure 1.4: Examples of digital boards.

CHAPTER 2

Order and Overview of Chapters

2.1 Objectives and Problem Specification

This work originated from an extension of the pilot project ‘Authoring on the Fly’ (AOF) [48, 50], where an open document type was introduced, covering a range of media classes that were made to synchronize through a time-based replay mechanism. The capture of freehand writings during a recording of a ‘live’ event, simultaneously with several other input modals, including audio and video signals, plays an important role in conveying a complete package of multimedia information. The same freehand writings are later replayed and animated in their formation in exactly the same way as they were recorded, and made to run in-sync with the other modals. The replay of this open document is also made controllable by a random-access timeline browser.

However, there were two shortcomings related to the freehand model that were not addressed until the feedbacks from the commercial spin-off “Lecturnity” [52] were returned for evaluation. The first was the quality of the recorded freehand writings, or rather, the lack of it. The second was that the replay mechanism did not convincingly reproduce the handwritings in the same manner that they were recorded.

Both these problems are somewhat related: What Lecturnity was programmed to do was to represent the freehand writings simply as a set of time-ordered points, stored directly from the generated output of the transducer device. This inadvertently left the quality of the written Traces at the mercy of the efficiency of the system’s processor on which Lecturnity was running. That is, a slower system would produce undesirable and pixelated freehand writings when compared to a much faster system. It made the Traces artificially unintelligible, even after post-processing. This is mainly due to the program dropping several sampled coordinates in favour of serving the other higher priority and computationally expensive modals; which in turn, resulted in inaccurate time-stamps in the sampled coordinates, that led to the second problem during replay.

Thus, our main objective is to overcome the two consanguineous problems.

We seek a viable solution whereupon the quality of the handwritten Traces are desirably improved, in accordance to the Weierstrass approximation theorem (explained later on in Theorem 4.1), during the recording phase as well as in the midst of replay. This shall be achieved using proper curves instead of the rudimentary raw sampled points, through symbolic representation of the curves' data structures. All these are to be done taking as little of the processor's time as possible. To state it simply, we want to apply our concept of *active-smoothing* on the freehand writings, and symbolically represent them for use in random access navigation.

Also, as a follow up to the main approach, we want to exploit the digital ink environment deeper so that it can accommodate both passive writing and interactive *gestures*, all in one non-segregated platform. We are keen to explore the possibilities of ink-based interactions that extend the properties imbued within the freehand Traces, particularly, the types of practical applications that they can be made to support.

2.2 Structure

The thesis is spanned over two main parts to tackle our objective statements above and to present our solutions to the problem specifications. We end off Part I here through the next chapter where we introduce the geometric components and their properties that make up a Trace and the other structures required to explain our concepts.

Part II is where we take the algebraic discussions of continuous curves and discrete Trace-segments from chapter 3, and examine the effects of simplifying a Trace before bombarding it with superfluous points to resemble high resolution smooth curves. The Trace is simplified by removing 'unnecessary' points and made to retain only the crucial ones that still define the overall shape of the Trace. From this discussion of the simplification of polylines, we move on to show how a set of indeterministic points can be made to fall in place so that, when properly ordered, they can be composed of only second-order curves. We deliberate this fact through our discussions by approximating these indeterministic points entirely with elliptic arcs, and later on with only rational quadratic Bézier curves. At the end of Part II is where we assemble the pieces together and present our *active-smoothing* solution. The preliminary ideas of which we have communicated in an ACM article published in the proceedings on Educational Multimedia and Multimedia Education [89].

Then in Part III, we review several techniques that enable gesticulations for the Traces, based on the intrinsic features that relate directly to a linear discriminator. These very stable techniques take us further in-depth into the kinds of collaborative applications that can be conceived; we give two examples of this with our work on the huge wall-mounted interactive whiteboards, and a game of Monopoly that we conceived for the primitive multi-user tabletop environment.

Finally, we sum up and conclude all of our findings in Part IV, and highlight several of the more prominent points uncovered while in the course of preparing this thesis.

CHAPTER 3

Geometric Entities of the Digital Ink

We lay the preliminary grounds in this Chapter and introduce the primary components in their geometric forms in 2D space that make up the various compositions of digital freehand writings. In particular, we put forward the concepts on how these components are stored and in what order, and how they can be manipulated efficiently to effect the solutions we present in the next two Parts of this dissertation.

The digital ink data structures are inherently discrete and are meant to support the geometrical elements in operations such as archiving, indexing, retrieving, and rendering. In later chapters, we will scrutinize the effects *rendering* has on digital ink applications. It is in this final operation that the balance of the technology being embraced and appreciated, or discarded and neglected, hangs upon. Its efficient handling (or lack thereof) is expected to mandate the closeness of the digital system in resembling the *feel* of writing on paper with liquid ink. Not only that, we should also expect an emphasis on a desirably high visual quality of the output renditions of the digitised handwritings on LCD screens, and further reflect this on print. This final operation draws its sustainability and efficiency from the former three operations mentioned above. Thus it becomes important that we study the primary components that make up the digital ink to comprehend the environment we are working with, which will help to realise the significance of our proposed methods and solutions in the rest of the thesis.

3.1 Discrete Ink Entities

Every component of the digital ink is made up of primary 2D points (or vertices). Each point $p_a \equiv (x_a, y_a, t_a)$ represents a location on the 2D plane and is identified by a unique (system) time-stamp t_a. That is, we make no assumptions that a set of points in the plane are in general position, and for those points that are

Component: Point p ♠

equal in both their x and y coordinates, they have different values of t without loss of generality.

Component: Edge E ♠

Definition 3.1 *An Edge $E_a = \overline{p_a p_b}$ is a directed line-segment from a start point $p_a \equiv (x_a, y_a, t_a)$ to an end point $p_b \equiv (x_b, y_b, t_b)$, where $t_a < t_b$, and is identified by the time-stamp t_a.*

Notice that we have begun adopting the convention of starting a well-defined term with a capital letter. This distinguishes the technical meaning of the term in its intended capacious definition, from the general meaning it may otherwise suggest.

The Edge is analogous to the LineSegment – both are referred to often in the field of computational geometry, and they carry a similar meaning to the Chord in mathematical geometry. These three terms are all part of the straight line, and are not to be confused with the full line concept, particularly when we later discuss the intersection of tangent lines. Furthermore, an Edge is unique in the digital ink environment, in that there can only be one incoming Edge and one other outgoing Edge per corresponding point.

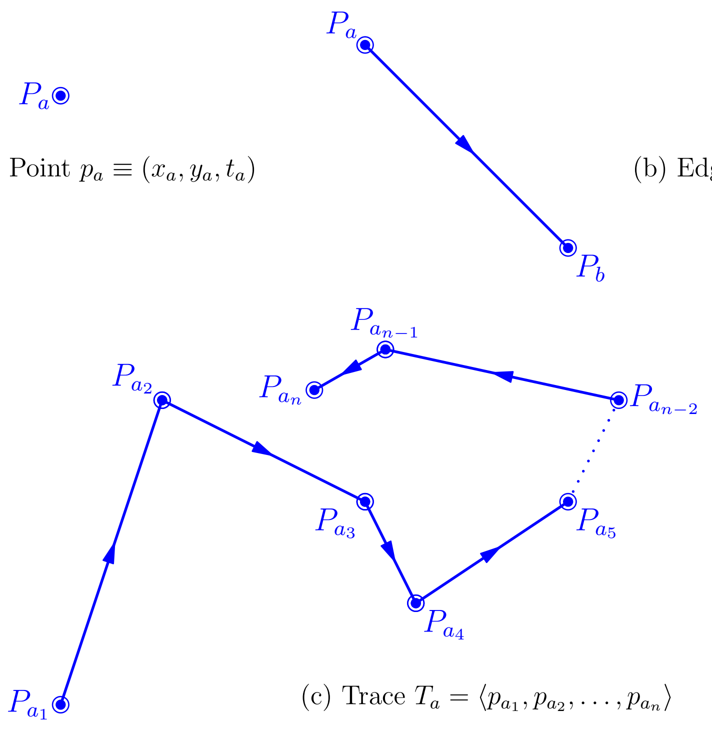
Definition 3.2 [TRACE]. *A TRACE $T_a = \langle p_{a_1}, p_{a_2}, \ldots, p_{a_n} \rangle$ is a collection of n time-ordered points, where two adjacent points p_{a_i}, $p_{a_{i+1}} \in T_a$ are joined by a single Edge $E_{a_i} = \overline{p_{a_i} p_{a_{i+1}}}$, for $1 \leq i \leq n-1$. The unique time-stamp of the starting point at p_{a_1} identifies T_a.*

We may also consider a TRACE T_a as $\langle E_{a_1}, E_{a_2} \ldots, E_{a_{n-1}} \rangle$; a collection of time-ordered Edges. This, however, is not as tight a definition as compared to Definition 3.2, since every intermediate point in T_a is repeated twice, as the end-point of E_{a_i} is also the start-point of $E_{a_{i+1}}$, for $i \in [a_1, a_{n-1})$. If such is the case, then we note that a TRACE must then be composed of at least one edge or two time-ordered points; as opposed to the above definition where it is possible to have only a single point inside a TRACE.

In Figure 3.1(b) and (c), we indicated arrowheads on each directed Edge to suggest that the discrete components presented here are not scalars. That is, they are direction-dependent and are to be interpreted based on the order of which the individual points appear inside each component.

Definition 3.3 [TRACESEGMENT]. *A TRACESEGMENT TS_α refers to a single contiguous portion of the collection of points from a well-defined TRACE T_a; i.e. $TS_\alpha \subseteq T_a$, for $\alpha \in [a_1, a_n)$, and that the trace-lengths are $\|TS_\alpha\| \leq \|T_a\|$.*

Essentially, by Definition 3.3, a TRACESEGMENT is also a TRACE. The difference between these two entities becomes stark when we use them inside applications. A TRACE will usually refer to a single raw entity, sampled from



(a) Point $p_a \equiv (x_a, y_a, t_a)$

(b) Edge $E_a = \overline{p_a, p_b}$

(c) Trace $T_a = \langle p_{a_1}, p_{a_2}, \ldots, p_{a_n} \rangle$

Figure 3.1: Time-ordered primary components: (a) Point p_a, (b) Edge $\overline{p_a p_b}$, and (c) TRACE T_a.

a transducer device, starting at a *pen-down* event and ending with a *pen-up* event. These two *pen*-events coincide with p_{a_1} and p_{a_n} respectively. A TRACE-SEGMENT, on the other hand, is merely a reference to a sub-component of the original TRACE, which we use to manipulate the entity in our deliberations and solutions, with additional special properties at its endpoints.

These definitions are almost similar to de Berg *et al.*'s [24] description of the *doubly-connected edge list* (DCEL) data structure, which stores detailed records for each face, edge, and vertex of a subdivision in the plane. While "directions" in the DCEL components are implicit, such that an edge is made up of two half-edges going in opposite directions in order to explicate the inner from the outer faces, it does not indicate the explicit "direction" of a single Edge as we stated in Definition 3.1. For example, walking around on the path of a set of connected half-edges in counter-clockwise direction is meant to discover a bounded face to the left of each half-edge. At this point, we are more interested in the full edges (rather than the half-edges) that join to make up a path to describe our TRACES. We are not yet concerned with the faces these edges bound. However, we will revisit this notion in Chapter 9 when we discuss *contour-wraps* related to font-processing in further details.

TRACEs in digital freehand writings do not require the 'multi-edge per vertex' property which is prevalent in the DCEL. That is, as we have shown previously, there can only be at most one incoming Edge and at most one outgoing Edge per vertex in a TRACE.

3.2 Continuous Ink Entities

In Part II of the thesis, we shall dwell deeper in the discussions on the continuous and symbolic representation of the discrete entities introduced in section 3.1. The continuous entities are the "smoothed" manifestations of the manipulated discrete components, particularly the TRACE. While we may argue that the continuous elements are still (somewhat) discrete when represented on a digital canvas, we shall set it up here to define that these components remain in their highest quality (while maintaining their *styles* and *flair*) in whatever resolution they are placed in. That is, we want to guarantee that they neither degenerate nor pixelate when viewed in deep levels of zoom.

3.2.1 The Notion of 'Curves' vs. CURVEs

A curve in our deliberations refers to a system of points in $\Re^2$ whose ordered set of coordinates (x, y) satisfy a given function. For each value of x there is a corresponding value of y evaluated by a function f, so that the graph of the function f is the set of all ordered pairs $(x, f(x))$, for all x in the domain X. Here, we take the two prominent terms to mean as follows; (i) that a well-defined "function" $f : X \to Y$ on $\Re^2$ is simply an expression which maps a value of $x \in X$ on to a unique value of $y \in Y$, where $X, Y \subset \Re$, and (ii) that a true "curve" spanning the $\Re^2$ plane is the viewable graph of an underlying function f whose ordered pair $p \equiv (x, y) \equiv (x, f(x)) \in f$ refers to the Cartesian coordinates of the curve.

Let us assume, for the sake of making our discussions easier, that all the functions we are dealing with are x-monotone. Of course, one can simply lift this assumption by breaking up the curve into x-monotone pieces and handle each one progressively, or by applying a certain 2D translation (matrix) on the curve, on the necessary interval(s), to handle the general case.

The definition of our CURVE (denoted by small-capital cases), on the other hand, refers to a subset of points of an algebraic function mapped from a closed interval, and whose graph depicts a segment of the corresponding curve related to the function. We state this formally as follows.

Definition 3.4 [Curve]. *A* Curve *K on the interval $[x_a, x_b] \subset X$ is a segment of a well-defined function $f : X \to Y$ on the same interval $[x_a, x_b]$, so that the endpoints p_A and p_B of K are $(x_a, f(x_a))$ and $(x_b, f(x_b))$, respectively.*

In other words, in terms of the set of points in K and $f(x)$, Definition 3.4 establishes that $K \subset f(x)$ on the interval $[x_a, x_b]$. Also, we shall assume that $f(x)$ is twice differentiable on the interval $[x_a, x_b]$, and consequently, we can determine the first and second derivatives for all the points $p_i \in K$.

Suppose we have two functions $f(x)$ and $g(x)$, from which we derive two Curves $K_f \subset f(x)$ and $K_g \subset g(x)$ on the intervals $[x_a, x_b]$ and $[x_b, x_c]$, respectively. Furthermore, let us also suppose that $f(x)$ intersects $g(x)$ at x_b, so that $f(x_b) = g(x_b)$. Then clearly, the end-point of K_f is exactly the start-point of K_g. Thus on the combined interval $[x_a, x_c]$, we have a CompositeCurve KC (denoted also by small-capital cases) made up of K_f and K_g connected at the common point $(x_b, f(x_b))$.

Definition 3.5 [CompositeCurve]. *A* CompositeCurve *$KC = \{K_1, K_2, \ldots, K_m\}$ on the interval $[x_0, x_m] \subset X$ is a continuous piecewise function on the same interval $[x_0, x_m]$, composed of m contiguously connected* Curve*s on the subintervals $[x_0, x_1], [x_1, x_2], \ldots, and [x_{m-1}, x_m]$, corresponding respectively to $K_1 \subset f_1(x), K_2 \subset f_2(x), \ldots,$ and $K_m \subset f_m(x)$, and where $f_1(x_1) = f_2(x_1), f_2(x_2) = f_3(x_2), \ldots,$ and $f_{m-1}(x_{m-1}) = f_m(x_{m-1})$.*

It follows from Definition 3.5 that the endpoints p_0 and p_m of KC are $(x_0, f_1(x_0))$ and $(x_m, f_m(x_m))$, respectively.

3.2.2 Crucial Points

The point connecting any two adjacent Curves in a CompositeCurve KC has a special property in our set up; it is one of the crucial points of KC.

Definition 3.6 [Crucial points - Part I]. *The crucial points in a* Composite-Curve *KC are (i) the two endpoints of KC, and (ii) the set of all points from which the adjacent* Curve*s K_i and K_{i+1} intersect, for all $K_i, K_{i+1} \in KC$.*

From the setup given in Definition 3.5, the crucial points of the CompositeCurve $KC = \{K_1, K_2, \ldots, K_m\}$ containing m Curves is the set of $m + 1$ points $\{(x_0, f_1(x_0)), (x_m, f_m(x_m))\} \cup \{(x_1, f_2(x_1)), \ldots, (x_i, f_{i+1}(x_i)), \ldots, (x_{m-1}, f_m(x_{m-1}))\} \subset KC$. These crucial points essentially hold the CompositeCurve together, and in a way, they symbolically describe the shape or outline of KC. We shall revisit this notion again later when we discuss Part II of the crucial points definition when dealing with discrete curves reconstruction in Section 3.3.1. Thereafter we shall point out that retaining the intermediate points between two consecutive crucial points in KC is not necessary since we can then "symbolically"represent KC with just its crucial points.

3.2.3 Algebraic and Geometric Viewpoints

The Curve in Definition 3.4 is partly based on the notion proposed by Yap [137], who discussed two main viewpoints for observing and manipulating continuous curves on discrete planes; namely, the algebraic viewpoint and the geometric viewpoint. Algebraically, we treat a Curve as a system of algebraic solutions to be solved, where if we apply symbolic or algebraic techniques on the system of equations, we will arrive at a set of solutions that are exact and complete. The geometric viewpoint, on the other hand, contrasts this. The same Curve is described via geometrical means, which solutions are obtained through numerical techniques. While the geometric solution may pose a problem of being incomplete due to carry-on errors per computation step, its approach is highly appropriate when it comes to dealing with curves in unknown and non-deterministic environments.

Yap's further comments coincide with our chain of thoughts; that while the algebraic viewpoint is more general than the geometric viewpoint, it alone cannot completely address our domain of specifications. In this study, we deal directly with the raw data of digital "curves" sampled from transducer devices. The values returned by the interface components are always discrete - that is, they are strict integer values that are either 'floor'-ed or 'ceiling'-ed. In other words, the trace-signals we receive from a transducer device are complete whole numbers, and that each coordinate pair (x, y) comes from the discrete positive integer domain, corresponding to the pixel location on the device. Clearly, as a result of this, we have already lost precision of the (intended) actual floating-point values x and y, since real numbers are not sustainable in the electronics. Thus, it becomes apparent that we are restricted only to making estimations based on the raw integer values in our computations to approximate the smooth Curves we are after. In consequence, we can only expect a likable reconstructed image of the trace-signals which ultimately depends on our treatment and management of the given discrete points.

The fact remains that when the original traces are observed even more closely, say at a deeper zoom level, we are more likely than not to see pixelated lines joining up the individual points scattered on arc-like trajectories. Up close, these points appear somewhat random, irregular, and non-deterministic. So here, the algebraic viewpoint completely breaks down on the Trace T, if we apply it directly. It becomes tedious to generate a proper curve to go through all these uneven points in T. Suppose there are n such points in T, then the technique to interpolate all n points in T returns an $(n-1)$-degree polynomial function $f(x)$, and assuming that we have chosen the classical polynomial interpolation method, runs in $O(n^3)$ time. (There is a more complex method which extends the polynomial interpolation technique that improves the runtime to $O(n^2)$ [68].) This, though, is the main problem. The overall resultant curve we get that passes through all these unevenly positioned points, is not necessarily the curve we are looking for, and may in fact be the wrong solution. Runge's interpolation phenomenon has shown that for large values of n, the interpolation polynomial $f(x)$ may oscillate wildly between data points [35]. Thus, due mainly to these reasons, this approach by itself is not at all suitable to cleanly and efficiently solve our problem.

Fortunately, the geometric viewpoint offers us several ways around the above problems, and at an achievable level of improved efficiency. The method allows us the ability to control our approach so that we return only a connected set of well-defined, low-order algebraic curves that – when properly pieced together – *acceptably* estimates the original TRACE.

3.3 Handling Digital Ink Entities

Building on the arguments highlighted in the previous section, we shall, for the rest of the thesis, deliberate all entities related to the digital ink to be based on the geometric viewpoint, and afterwards combine it with the algebraic viewpoint. On top of this, we will also take into considerable account of the classical concepts where methods exist to pass spline curves through a set of given points in the plane.

The main difference of our approach here, compared to the classical methods, is that we do not "fit" a single algebraic curve $f(x)$ faithfully through every point p_i in the TRACE T, but rather, we "estimate" (or "regress") the n points in T with a series of m contiguously connected CURVEs. Each CURVE K_j represents a subset of points in T, so that the entire TRACE T is collectively represented by the COMPOSITECURVE $KC = \{K_1, K_2, \ldots, K_m\}$.

We noted earlier in Section 3.2.2 and by our Definition 3.6, that the point connecting any two adjacent CURVEs K_j and K_{j+1} has to be a "crucial" point of KC. So if we want to make a clean transition, to transform a discrete T into a continuous KC, then it follows that KC must contain the crucial points in T. In other words, when two adjacent CURVEs in KC connect, they connect at a crucial point of T. This will ensure that we preserve the characteristics of the appearance of T accurately with KC. We shall later observe that once these crucial points have been identified, the rest of the non-crucial points in T can be expected to fall into place, and they serve as the necessary trajectory guide in determining a proper function $f_j(x)$ – spline or otherwise – from which we can extract the relevant CURVE $K_j \in KC$. Furthermore, once K_j has been truly determined, it becomes unnecessary to keep the non-crucial points anymore.

3.3.1 Road Map: From Discrete Traces to Smoothed Curves

What concerns us the most in Part II of the thesis is to discover how we can properly represent T with KC. The solution must (i) be efficient in runtime, (ii) be within an acceptable error bound, (iii) 'smooth' out the uneven pixelated lines, (iv) maintain precise resolution in deep zoom levels, and above all, (v) preserve the extraordinary features of the raw TRACE that are the defining style and flair contained in the original handwriting.

Let us state the problem plainly. Suppose we are given a set T of n ordered points $\{p_1, p_2, \ldots, p_n\}$, where $p_i = (x_i, y_i)$ for all $p_i \in T$, and from which same set there exists a subset of crucial points that breaks T up into m segments $TS_1, TS2, \ldots, TS_m$. Suppose also that the smooth representation of the TRACE T is the COMPOSITECURVE KC. Then the task is to find the finite set of proper functions $f_j(x)$ from which to describe each elemental CURVE K_j in KC, based on the corresponding points in the segment TS_j in T, and then symbolically render the whole of KC smoothly at any desired resolution.

Clearly, the resultant $KC = \{K_1, K_2, \ldots, K_m\}$ has to be made up of a finite set of m elemental CURVEs, joined contiguously to represent m related segments of T. By Definition 3.6 and from our discussions above, we know that any two adjacent CURVEs in KC must join at one of the crucial points of T. This then implies that there should be $m + 1$ crucial points in T. Subsequently, it follows that the approach to the problem is first to identify the $m + 1$ crucial points of T, and then use them to break down T into m appropriate segments $TS_1, TS2, \ldots, TS_m$.

In our case, we only consider a segment TS_j of T when it contains the subset of ordered points between, and including, two consecutive crucial points of T. The CURVE K_j shall then be derived directly from TS_j.

Definition 3.7 [Crucial points - Part II]. *The crucial points of the* TRACE T *are the subset of points in* T *that are the* inflection points *and the* sharp-edge vertices *of* T *with respect to all other points in* T*, and that they define the* distinctive shape *of* T.

This means that the TRACE T is made up of two mutually exclusive subsets of points: (i) the set of *crucial* points $S \subset T$, and (ii) the set of *non-crucial* points $S' \subset T$, where $S \cup S' = T$ and $S \cap S' = \emptyset$. Furthermore, if we are to remove all the non-crucial points of T, then by Definition 3.7, the remaining crucial points will demarcate the basic skeleton shape of T.

Thus far, we have established that KC is a piecewise amalgamation of m functions (by Definition 3.5) - which means that KC is continuous and is *not* disjoint. Viewed from this perspective, it becomes obvious that the first few crucial points to immediately stand out, apart from the two endpoints $p_1, p_n \in T$, are those "sharp-edged" vertices where CURVEs of two distinctly opposing functions have been joined. The other not-so-obvious crucial points are those points on KC where "inflections" occur. We shall discuss these special types of crucial points in more details in Section 3.3.4 through Definitions 3.14 and 3.15.

Now consider the following lemmas.

Lemma 3.8 *Let $f(x)$ be an n-degree polynomial curve on $\Re^2$. Then there are at most $n-2$ inflection points on $f(x)$.*

Lemma 3.9 *Let p_j and p_{j+1} be two consecutive inflection points on $f(x)$. Then the jth segment of the curve $f(x)$ between p_j and p_{j+1} is* convex.

What these observations lead us to, is the conclusion that the non-crucial, ordered points in a segment $TS_j \subseteq T$, are arranged in a rather clean and almost arc-like, convex trajectory. Since non-crucial points are neither sharp-edges nor inflections, they consequently become the basis for guiding the construction of a "convex" CURVE. Furthermore, this convex property allows for a low-order polynomial function $f_j(x) \supseteq K_j$ to be suitably used to estimate this "convex" segment TS_j. In fact, we shall show conclusively later on in Part II that a second-order polynomial function $f_j(x)$ is sufficient to achieve this. Our proofs include the sole use of elliptic arcs (which are inherently second-order curves) as the elemental components in KC, and this met all the five criteria we highlighted above in representing T suitably with KC.

The following propositions sum up our discussions thus far, in converting a digital TRACE T into a smooth COMPOSITECURVE KC. They shall provide the conceptual outlines from which we will use to expand (and exploit) in the next few chapters.

Proposition 3.10 *Let the* TRACE *T be a set of n ordered points, and let T be made up of m contiguously connected segments $TS_1, TS_2, \ldots,$ and TS_m. If each segment TS_j of T is the subset of ordered points of T between, and including two, consecutive crucial points of T, then TS_j is a* convex *segment.*

Proposition 3.11 *Let $TS_j \subseteq T$ be a convex segment, and let the* CURVE *$K_j \in KC$ be the smooth representation of TS_j. If $f_j(x)$ is the function where K_j is derived from, then $f_j(x)$ need not be higher than a second-order polynomial function, and that subsequently, $K_j \subset f_j(x)$ is a* convex CURVE.

Proposition 3.12 *Let the* COMPOSITECURVE *KC, made up of m contiguously connected* CURVE*s, be the smooth representation of the* TRACE *T. Then by Propositions 3.10 and 3.11, KC contains only second-order polynomial convex* CURVE*s, contiguously joined at the crucial points of T.*

3.3.2 Continuity and Joints

So far we have assumed that the elemental CURVEs in KC are contiguously joined in such a way that the harmony of the overall shape of KC is maintained. In this section, we point out that any two adjacent CURVEs in KC join with either G^0, G^1, or G^2 continuity at a crucial point of T. The underlying raw points of T, neighbouring the crucial point, contain the information we need to determine which of the three continuity-joints we have to apply, so that we sustain the local characteristics of T in KC at that particular joint.

We shall make explicit use of this section to verify a theorem by Barsky and de Rose [8], which states:

Theorem 3.13 *(Barsky and de Rose, 1989) Two parameterizations meet with G^1 continuity if and only if they have a common unit tangent vector; they meet with G^2 continuity if and only if they have common unit tangent and curvature vectors.*

The parameterizations in Theorem 3.13 refer to, in our case, the set of CURVEs in $KC = \{K_1, \ldots, K_n\}$ that we use to estimate a single TRACE T. Establishing continuity on (the representation of) an unbroken TRACE T means that we also allow for "piecewise" joints within T whenever necessary. The simplest piecewise joint is the lowest level G^0 (and C^0) continuity at a point $p_j \in T$, in which two CURVEs K_i and K_{i+1} are connected at their respective endpoints. G stands for geometric continuity, while C stands for parametric continuity and C is the more rigorous of the two. For reasons of simplicity, however, we shall use both terms interchangeably.

If we further take into consideration the tangency and possibly the curvature about the point p_j, we would then be observing for G_1 and G_2 continuities respectively. We summarize their descriptions as follows:

- G^1 **continuity (Tangency):** Two CURVEsjoined at a point where a common tangent exists. The tangency continuity means that the tangent at the end of the CURVE K_i is parallel to the tangent at the beginning of the CURVE K_{i+1}, at the common point p_j. This is on top of the fact that both CURVEs must already possess G^0 continuity at p_j.

- G^2 **continuity (Curvature):** Two CURVEsjoined at a point where a common curvature exists. The curvature continuity means that, with all previous G^0 and G^1 continuity conditions fulfilled, the point p_j connecting K_i and K_{i+1} is the point with second-order geometric continuity in the two combined segments.

The approach, then, to get a good and smooth estimate for T in the form of a set of connected CURVEs KC, with all the original handwritten style and flair intact, lies in identifying all the necessary crucial points $p_j \in T$, and then determining which of the three continuity levels each of those points should be. We do this in three stages:

First, we measure the derivatives and curvature at every point in T. This will tell us the relations a point has with respect to its neighbours, and with respect to the estimated trajectory it is currently running on. Second, from the curvature measurements, we deduce whether a point is an inflection point or a sharp-edge vertex. If so, we mark it as a crucial G^2 or G^0 continuity point, respectively. This effectively partitions the Trace into more manageable segments, where each segment is part of a common arc. Finally, we complete the "estimate" by taking the points in each partitioned segment and regress proper curves through them, while in the process, further identifying points in T that are the crucial G^1 continuity tangent joints, whenever necessary.

3.3.3 Deriving First and Second Derivatives

At this juncture, let us assume that the set points in the TRACE T on a closed interval $[x_a, x_b]$ is the subset of points of an abstract function $f : [x_a, x_b] \rightarrow \Re^2$ that is twice differentiable on the same interval (unless proven otherwise based on some specific (error) measurements). Since we cannot yet algebraically define what $f : [x_a, x_b]$ is, based only on the raw points in T, we are construed to using geometrical methods to compute the numerical estimates of the derivatives at all n points in T. On top of this constraint, we also note that the classical techniques such as Newton's, Horne's, Atkin's, or Müller's methods [12], all assume that the function is monotonous on one axis. This allows us to describe (portions of) T as a continuous function $y = f(x)$, from which we can interpret every point $p_i \in T$ as $p_i \equiv (x_i, y_i) \equiv (x_i, f(x_i)) \in f$.

Discretely determining the first derivative $f'(x)$ at a sampled point p_i in the midst of the TRACE T, then becomes a matter of adhering to the following differential approximation:

$$f'(x) \approx \frac{\Delta f(x)}{\Delta x} = \frac{f(x+h) - f(x-h)}{2h}.$$

For reasons of symmetry, the approximation above is based on the two immediate neighbours of the point $p \equiv (x, f(x))$. However in our case, we cannot guarantee that the distance h between the neighbouring points are always the same. Hence, we give a more appropriate approximation of the first derivative of the point p_i with respect to the actual distance on the horizontal axis, and denote it simply as dp_i.

$$\begin{aligned} dp_i = f'(x_i) &\approx \frac{f(x_{i+1}) - f(x_{i-1})}{h_{i+1} + h_i}, \text{ where } h_i = x_i - x_{i-1}, \\ &= \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}, \text{ for } 1 < i < n. \end{aligned} \tag{3.1}$$

In the case of computing the derivatives at the endpoints p_1 and p_n of T, where there is only one other neighbouring point, we apply the more direct Lagrange's forward and backward differences accordingly for dp_1 and dp_n.

$$\begin{aligned} dp_1 = f'(x_1) &\approx \frac{f(x_2) - f(x_1)}{h_2} = \frac{y_2 - y_1}{x_2 - x_1}, \text{ and} \\ dp_n = f'(x_n) &\approx \frac{f(x_n) - f(x_{n-1})}{h_n} = \frac{y_n - y_{n-1}}{x_n - x_{n-1}}. \end{aligned}$$

We note here that this measure of first derivative at p_i should not be used *directly* to identify crucial points, or for any other identification process, no matter how accurate the value may be, since we should always assume that all the points in T are not necessarily positioned. Rather, all computations involving the first derivative must additionally incorporate other characteristic aspects of the point, in order to provide the safe bounds we need in *even*ing out approximations to be used in further computations.

A coincidental example here is where we take the resultant values of the first derivative to compute the second derivative. By measuring the rate of change of the first derivative through applying the same approximation method, we even out the estimates in the resultant measures of the second derivative. Recall the second-order (symmetrical) differential approximation of a point $p \equiv (x, f(x))$,

$$f''(x) \approx \frac{f'(x+h) - f'(x-h)}{2h}.$$

Since we have made the assumption, that the $f(x)$ representation of T is twice differentiable, then using five consecutive neighbouring points, we can similarly approximate the second derivative of the point p_i and denote it simply as d^2p_i.

$$\begin{aligned} d^2p_i = f''(x_i) &\approx \frac{f'(x_{i+1}) - f'(x_{i-1})}{h_{i+1} + h_i} \\ &= \frac{1}{x_{i+1} - x_{i-1}} \left(\frac{y_{i+2} - y_i}{x_{i+2} - x_i} - \frac{y_i - y_{i-2}}{x_i - x_{i-2}} \right), \text{ for } 2 < i < n-1. \end{aligned} \tag{3.2}$$

3.3.4 Deriving Curvature, Inflection Points, and Sharp-Edge Vertices

In the general sense, *curvature* refers to the amount by which a geometric object deviates from being flat. In our case, the object above refers to a point in Euclidean space relating to the radius of the curvature of the circle that touches it. The curvature k that we measure at $p_i \in T$ is the extrinsic curvature, which describes a space "curve" entirely of its "curvature", torsion, and the initial starting point and direction [69, 4].

For a parametric point $p_i(t) \equiv (x_i(t), y_i(t)) \in T$, the curvature k_i is defined as

$$k_i = \frac{d\phi_i}{ds_i} = \frac{d\phi_i/dt}{ds_i/dt}, \tag{3.3}$$

where ϕ_i is the tangential angle and s_i the arc length given by

$$\frac{ds_i}{dt} = \sqrt{\frac{dx_i}{dt}^2 + \frac{dy_i}{dt}^2} = \sqrt{x_i'^2 + y_i'^2}. \tag{I}$$

We need the tangent identity to compute the $d\phi/dt$ derivative;

$$\tan\phi_i = \frac{dy_i/dt}{dx_i/dt} = \frac{y_i'}{x_i'},$$

so that we have the following expressions:

$$\begin{aligned}
\frac{d}{dt}(\tan\phi_i) = \sec^2\phi_i \frac{d\phi_i}{dt} &= \frac{x_i'y_i'' - y_i'x_i''}{x_i'^2} \\
(1 + \tan^2\phi_i)\frac{d\phi_i}{dt} &= \frac{x_i'y_i'' - y_i'x_i''}{x_i'^2} \\
\frac{d\phi_i}{dt} &= \frac{x_i'y_i'' - y_i'x_i''}{(1 + \frac{y_i'^2}{x_i'^2})x_i'^2} \\
\frac{d\phi_i}{dt} &= \frac{x_i'y_i'' - y_i'x_i''}{x_i'^2 + y_i'^2}
\end{aligned} \tag{II}$$

Putting (I) and (II) back into Equation 3.3, we get

$$k_i = \frac{x_i'y_i'' - y_i'x_i''}{(x_i'^2 + y_i'^2)^{3/2}} \text{ , for } 2 < i < n - 1. \tag{3.4}$$

The expression above is a direct implication that is applicable to proper continuous curves. In our approach with the indeterministic points in T, we add an additional step to further even out the curvature reading at p_i with its neighbour p_{i-1}, with

$$\widehat{k_i} = \frac{1}{2}(k_i + k_{i-1}). \tag{3.5}$$

By monitoring the curvature values computed at every point p_i, we can determine the characteristics of those points with respect to the underlying curve of the TRACE T. For example, if the curvature values measured at five subsequent points in T are of the same sign and are somewhat constant, or constantly increasing (or decreasing), then this means that the underlying curve passing through these five points is convex. Furthermore, the curve for this range of points has a gentle gradient, if the curvatures are constant, or it has an increasingly steep gradient if the curvatures are constantly increasing (or decreasing). On the other hand, if the measured curvatures at these five points are all zero (or hovering around zero), then the underlying curve is simply a straight line.

Definition 3.14 [Inflection point]. *An inflection point on a well-defined curve $f(x)$ is a point $p_i \equiv (x_i, f(x_i)) \in f(x)$ at which the curvature $k(p_i)$ changes sign from $k(p_{i-1})$.*

There are other ways to define an inflection point, and the literature in differential calculus describes three others. The inflection point on $f(x)$ is a point p_i at which

- the tangent at p_i crosses itself; or
- the first derivative dp_i is at an extremum with respect to $f(x)$; or
- the second derivative d^2p_i changes sign from d^2p_{i-1}.

For the purpose of our discussions, we shall restrict ourselves to Definition 3.14 whenever we mention inflection points, as it gives us the strongest basis to deal with non-deterministic points. Thus, as we traverse from one point to the next in T, we can deduce the following about the underlying projectile of the "curve" from the measure of the curvatures that we picked up:

- Circular arc – when the measure is at a somewhat constant value.
- Straight line – when the measure is close around zero.

However, if we encounter a huge jump in values (known as a "spike") when tracking from $k(p_{i-1})$ to $k(p_i)$, then the curve we are travelling on has just made an abrupt sharp turn. That is, we have just come across a sharp-edge vertex at the point p_i.

Definition 3.15 [Sharp-edge vertex]. *A sharp-edged vertex is detected at $p_i \in T$ when the absolute difference between two consecutive curvature measurements $k(p_{i-1})$ and $k(p_i)$ is unusually high, with respect to the readings of the average curvature differences of all other neighbouring points in T.*

The measure of curvatures after $k(p_i)$ will undoubtedly come back down to its pre-spiked constant values. And if these values changed sign after the spike, then not only is the point p_i a sharp-edge vertex, but it is also a cusp – a special case of the sharp-edge vertex which is the dual of the inflection point [99]. Also, for completeness, we shall consider the two endpoints $p_1, p_n \in T$ to be the special cases of the sharp-edge vertex stated in Definition 3.15, because they sharply signify the start and the end of the CompositeCurve itself on the Re^2 plane.

3.4 Support Data Structures

Specifically, there are two things we need to consider when we represent Traces on the 2D plane for search and retrieval; its temporal order, and its spatial arrangement. The former is a natural and straight-forward order – every single point, and thus every single Edge and Trace is time-ordered. The latter, on the other hand, would require additional processing to distinguish the boundaries of the "shapes" of the Traces' paths with respect to one another.

Furthermore, once we have the Traces converted to and represented as proper Curves, we want to make sure that the underlying data structures are subtle enough to handle both types of entities. The common object that the Trace and the Curve share when it comes to rendering them on the plane, is the "path" of (ordered) points. The path provides instructions in the form of a sequence of orders for the on-screen drawing mechanism to know which two dots to connect, and at what resolution.

When we construe our observations and deliberations to the 2D environment, then it is clear that there are already several well-defined and efficient data structures that we can use to store and order the main elements of the Trace. So instead of picking one particular structure, we provide an object-oriented description of the design of the data structures that is generic yet explicit enough for the reader to choose (and extend) a known structure and implement them in whichever language is appropriate.

3.4.1 Active Point

From the discussions in section 3.3, we find it to be more efficient, and perhaps, convenient, to pack into a primitive point which holds only the (x, y) coordinate information, with additional *active* measurement variables. These variables include, among others, the first and second derivatives, the curvature, and the cumulative arc-length.

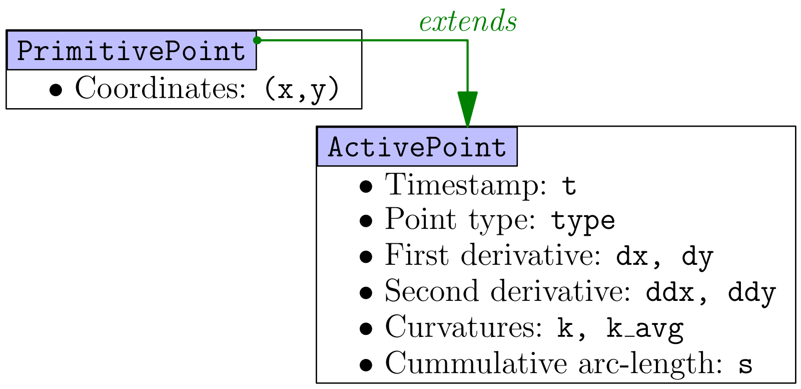


Figure 3.2: `ActivePoint` extends `PrimitivePoint`, where the individual derivatives are $dx = x', dy = y', ddx = x'', ddy = y''$, and where $k\_avg = \widehat{k}$. A 'point type' refers to the category in which the `ActivePoint` falls in – either `Normal`, `Inflection`, or `SharpEdge`.

Let us call this 'packed' primitive point `ActivePoint`. Then it follows that an `ActivePoint` extends the definitions of a `PrimitivePoint` with the active contents listed in Figure 3.2. In section 3.5, we will discuss the "Active Sampling" algorithm that computes all necessary measurements that can immediately deduce the point type as either `Normal`, `Inflection`, or `SharpEdge`.

3.4.2 Temporal Traces

A structure similar to the doubly-connected `LinkedList` serves well as a container to store the `ActivePoints` in their temporal order. This structure is essentially, our `Trace`. Now, the same `LinkedList` container also serves well to store a set of `Traces` in their temporal order. We depict this straight-forward relations in Figure 3.3.

3.4.3 Spatial Traces

More often than not, we need to answer queries requested directly from the 2D plane, when interfacing with the graphics becomes inevitable. For example, given a query point q, return all the `Traces` stabbed by q. Or, given a query rectangle r, return all the `Traces` overlapping the range of r. These kinds of queries are efficiently handled if we have an underlying spatial structure that indexes the 2D boundaries of each `Trace` in accordance to their positions on the plane.

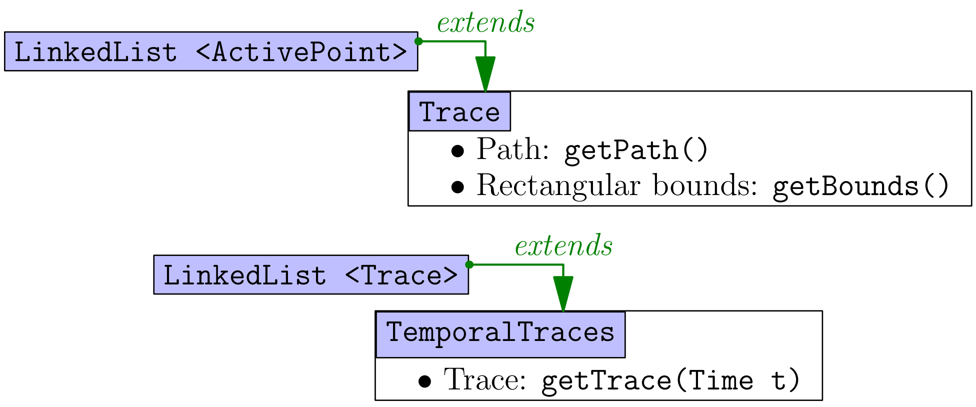


Figure 3.3: Temporally ordering `ActivePoints` and `Traces`.

There are many geometric structures that can store 2D data and offer efficient space and runtime to build and answer both stabbing and range queries. They are mostly cascaded 1D structures like the Range-tree, Interval-tree, Segment-tree, and the Priority-search-tree [25] constructed to handle two dimensional data. These though, can be costly when we need to perform online updates. We want to consider the non-static insertions and deletions of `Traces` on the canvas environment, and seek for a structure that actively and reliably balances its contents in good acceptable runtime.

For the purpose of our discussions, let us concentrate on placing the spatial `Traces` in an R-tree structure [40], or any of its extended variants [114, 33, 3]. This, we shall discuss further in-depth when we deliberate on the effects of spatial interfaces. For now though, we shall state generically how the structure for `SpatialTraces` is conceived through Figure 3.4.

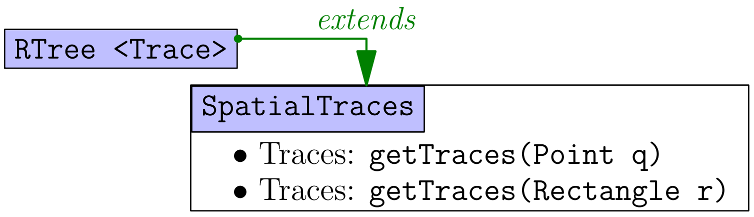

RTree <Trace>
extends
SpatialTraces
• Traces: getTraces(Point q)
• Traces: getTraces(Rectangle r)

Figure 3.4: Structure to order `Traces` according to their spatial locations.

3.5 Active Sampling

The technique we present here goes beyond the notion of simply storing (x, y) coordinates into a `PrimitivePoint`. That is, whenever the recently sampled points become available for the computation of the measure-values highlighted in section 3.3, we carry them out immediately. And from the way the values are primed, we shall see that the only latency introduced in this process is a time(-stamp) difference of three consecutive points. For example, suppose p_n is the newest sampled point from the TRACE environment at time t_n. Then it is only at this instant that we can compute the second derivative at p_{n-2}, since d^2p_{n-2} requires two future and past values as stated in Equation 3.2.

The purpose of sampling "actively" is to be able to perform all the necessary computations needed to classify the point types within a `Trace`, even before the very final point of the `Trace` is sampled. We can go on further to add that the domain-specific application may execute any additional routines to the identified crucial points, even before the pen-up event is detected. We state the active sampling routine in the `ActiveSampling.addPoint` pseudo-codes in Algorithm 1, showing that we can perform the entire identification process in just one sweep of the points in the `Trace`.

An `ActivePoint` is instantiated as a `Normal` point type, and we we mark it as crucial by changing the point type to either type `Inflection` or type `SharpEdge`, depending on the reading of the curvature.

Finally, let us now formally denote a TRACE T which has undergone the active sampling process shown above, as an ACTIVETRACE T^{M}.

Definition 3.16 [ACTIVETRACE]. *An* ACTIVETRACE T^{M} *has all its points* $p_i \in T^{\mathrm{M}}$ *marked as either* `SharpEdge`*,* `Inflection`*, or* `Normal`*.*

Both T and T^{M} contain the same set of points; the latter is the product of Algorithm 1 based on the former, and that, one can be plainly expressed as a function over the other as

$$T^{\mathrm{M}} \longleftarrow \text{ActiveSampling}(T = \{p_1, \ldots, p_n\}). \tag{3.6}$$

Algorithm 1: `ActiveSampling.addPoint` (`ActivePoint` p_n)

Output: Computes the 'active' attributes in p_n and adds it into the current Trace T.

```
1  Set Type($p_n$) ← SharpEdge;
2  if $T$ is empty then Add $T \leftarrow p_n$;
3  else
4      Set dy/dx($p_n$) ← $(p_n.y - p_{n-1}.y)/(p_n.x - p_{n-1}.x)$;
5      if $p_{n-2}$ DO NOT exist then Set dy/dx($p_{n-1}$) ← dy/dx($p_n$);
6      else
7          Set Type($p_{n-1}$) ← Normal;
8          Set dy/dx($p_{n-1}$) ← $(p_n.y - p_{n-2}.y)/(p_n.x - p_{n-2}.x)$;
9          if $p_{n-3}$ exists then
10             Set $p_k \leftarrow p_{n-2}$;
11             Set ddy/ddx($p_k$) ← $(p_{k+1}.y - p_{k-1}.y)/(p_{k+1}.x - p_{k-1}.x)$;
12             Set $p_k.\hat{k}$ ← Compute average curvature;  /* Equation 3.5 */
13             if $|p_k.\hat{k}| >$ SHARP_TOL then Set Type($p_k$) ← SharpEdge;
14             else if sign($p_k.\hat{k}$) $\neq$ sign($p_{k-1}.\hat{k}$) then
15                 Set Type($p_k$) ← Inflection;
```

Part II

Active-Smoothing & Symbolic Representation

CHAPTER 4

Confluent Lines Over Ordered Point Sets

Smoothing out handwritten Traces by blending in a steady stream of undulating Curves through a set of indeterministically positioned points on the 2D plane, by finding the path through which provides the most *desirable* approximation. This is a problem that overlaps two classical and well-known mathematical conundrums; *interpolation* and *regression*. In interpolation, an exact fit is required of the resultant set of Curves over all the points in the Trace, while regression allows for some degree of freedom for an estimated fit by minimizing the error measures between the original Trace and its corresponding generated Curves. An approximation of a Trace T by a CompositeCurve KC is deemed desirable, if for every original point $p \in T$, its distance to the corresponding 'smoothed' point $p_\theta \in KC$ is confined to an allowable error bound value ϵ.

While there are authors who go on to *extrapolate* their final solutions [118, 10], by constructing a set of new points for their curves that are outside the jurisdiction of the given discrete set in the original Traces, many consider their results less meaningful and are subject to greater uncertainty [80, 125, 83]. We concur with this view and further find that extrapolated curves over handwritten Traces tend to exaggerate several non-existential features of the original Traces, and, more disturbingly, contribute to the suppression of the personal 'style' and 'flair' that are embedded inside the rudimentary handwritings. We show a few examples of what we mean by this in the figures throughout the chapter.

This danger of artificially instantiating curves that over-estimate the original Traces is also eminent in the regression process – if we allow too much freedom to be dictated in the constraints. Interpolation, then it seems, is the logical answer if everything intrinsic about a Trace is to be preserved. But this is true only if we assume that the points lie exactly where they were supposed to be and were not interfered by noisy interruptions, however minimal, through the sampling and digitizing processes. These indeterministically scattered points, though, leave important clues on their trails that we can pick up from observing their overtures and trajectory displacements. Thus, it makes sense to retain several of the more dominant original points that are *crucial* in maintaining the overall shape of the Trace and ensure that our generated Curves pass through them. The less dominant ones, on the other hand, should not be dismissed completely but acted upon as guides in controlling the paths of the Curves as they traverse from one crucial point to the next, to within their spheres of trajectory.

Our methodologies that we shall present in chapter 9 go further than to just desirably approximate handwritten Traces with a set of well-placed second-order Curves. Over the next few chapters, in the run-up to the final approach, we shall be assembling the pieces necessary to realise our *active-smoothing* solution. Simply put, active-smoothing turns the 'passive' routine of smoothing Traces 'active'. That is, in the classical cases, the smoothing problem is tackled as an 'offline' problem, where complete data of the Traces are at our disposal. Contrary to this, the 'online' problem warrants that we work with incomplete data *actively.* Which means we perform all the necessary computations on the current set of available points, while rendering the smoothed arcs on to the screen, *and* while data is still being received from the transducer device.

We will show how to carry this out efficiently through the insurmountable reduction of the expensive and inherently linear computations per input point, down to a constant. But first, let us prepare ourselves with the obligatory literature and state-of-the-art related to this unique problem domain. Spline curves, described by algebraic polynomials, interest us. This chapter discusses them thoroughly and we arranged it to objectively lay our grounds in support of our goal to achieve *active-smoothing.*

4.1 Undulating Spline Curves

The Curve we described in Definition 3.4 takes two forms of representations; either by the direct algebraic expression $f(x, y)$, or by parametrically combining the individual axis functions $P(t)|_r = (x(t), y(t))$, for a predefined resolution r. In the digital world, this translates the Curve to be composed of a finite number of short line-segments.

Thereafter, the curves that we see on screen are in fact 'approximations'. And on close inspection, they have a tendency to visually breakdown into pixelated reditions at very deep zoom levels. Naturally, to get a better approximation, we increase the resolution r to generate more line-segments per unit length. Thus, if the Curve is to be treated simply as a container to store line-segments (or points), then increasing r would inevitably increase the amount of data required to represent the Curve. In consequence, this large data size is liable for the slowdown factor added to the process of drawing out the Curve, as well as performing any future computations on the Curve.

When too few points are not satisfactorily pleasing on the output renditions, and when too many points make it overwhelmingly tedious to maintain precise operational efficiency, then clearly, we need a better way to represent the Curves than just with a set of elementary points. The Curves should only describe the formation of its mathematical entity and not be burdened with generating the line-segments to be drawn.

A curve that is made to pass through a set of predetermined points is often referred to as spline curves. The term originated in manual design, where a 'spline' is a word used to describe a long, narrow, and relatively thin strip of wood (or metal) used by shipbuilders and draftsmen. This strip was held in place by weights to create a curve which could then be traced.

An unconstrained natural spline curve of degree n comes under a class of functions that maps a set of real numbers into itself, known as the 'algebraic polynomials'. It takes the form

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x + a_1, \tag{4.1}$$

where n is a non-negative integer and $a_1, \ldots, a_n$ are real constants. These polynomials uniformly approximate continuous functions on a closed and bounded interval $[a, b]$, and which resultant curves adhere to the Weierstrass approximation in Theorem 4.1.

Theorem 4.1 (Weierstrass Approximation [12]) *Suppose that f is defined and continuous on $[a, b]$. For each $\epsilon > 0$, there exists a polynomial $P(x)$, with the property that*

$$|f(x) - P(x)| < \epsilon, \textit{ for all } x \textit{ in } [a, b].$$

The extensive proof of this theorem is given by Burden and Faires [12].

4.2 Lagrange Interpolation

An important reason for considering the algebraic polynomials class is that the derivative and the indefinite integral of a polynomial are easy to determine, and they are also in the form of polynomials. The Lagrange polynomials are favoured over the Taylor polynomials when approximating continuous functions because they do not have the limiting property of agreeing with a given function at only a specific point like Taylor's do [12].

The Lagrange polynomial is determined by specifying the points on the plane through which it must pass. Each point contributes to a coefficient in Equation 4.1, and that the total number of points on the plane that the curve goes through is the degree of the polynomial.

Let $T = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be the set of points to be interpolated using the Lagrange polynomial. Suppose that the points in T are sampled from a continuous x-monotone function, then $x_1 < x_2 < \dots < x_n$ and that $(x_i, y_i) = (x_i, f(x_i))$. We can rewrite Equation 4.1, so that the coefficients are expressed as a function of the n input points as follows:

$$P(x) = L_1(x)f(x_1) + L_2(x)f(x_2) + \dots + L_n(x)f(x_n)$$
$$P(x) = L_1(x)y_1 + L_2(x)y_2 + \dots + L_n(x)y_n = \sum_{k=1}^{n} L_k(x)\ y_k.$$

The coefficients $L_k(x)$ of the Lagrange polynomial is computed by the equation

$$L_k(x) = \frac{(x-x_1)(x-x_2)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_n)}{(x_k-x_1)(x_k-x_2)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_n)}$$
$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x-x_i}{x_k-x_i}.$$

Example 4.2 *Given three points in T, so that $T = \{(235, 27), (300, 50), (365, 37)\}$, evaluate the Lagrange polynomial at $x = 288$.*

Solution. We first compute the three coefficients for the Lagrange polynomial $P(x)$. In nested form, they are

$$L_1(x) = \frac{(x-300)(x-365)}{(235-300)(235-365)} = \frac{(x-665)x+109500}{8450};$$
$$L_2(x) = \frac{(x-235)(x-365)}{(300-235)(300-365)} = \frac{(x-600)x+85775}{-4225};\ \text{and}$$
$$L_3(x) = \frac{(x-235)(x-300)}{(365-235)(365-300)} = \frac{(x-535)x+70500}{8450}.$$

This gives the Lagrange polynomial

$$P(x) = 27\frac{(x-665)x+109500}{8450} + 50\frac{(x-600)x+85775}{-4225} + 37\frac{(x-535)x+70500}{8450}.$$

Then, an approximation at $x = 288$, we get $P(x = 288) = 48.4634$. ■

Coincidentally, the three points in T are sampled from Figure 4.1, forming part of the top profile of a stretching cheetah. We rendered the approximated

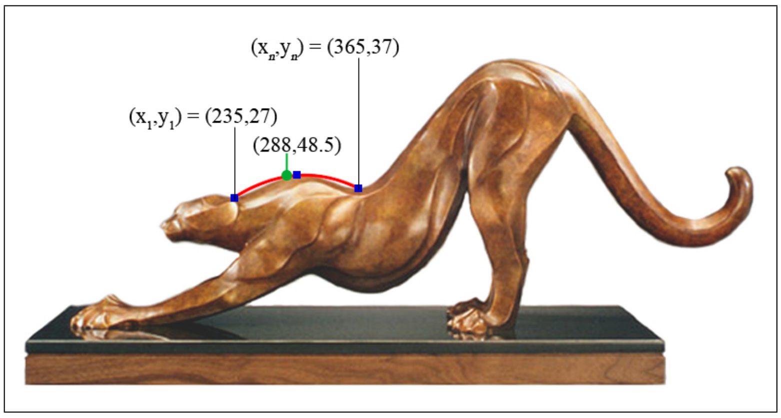


Figure 4.1: Interpolating three points with the Lagrange polynomial.

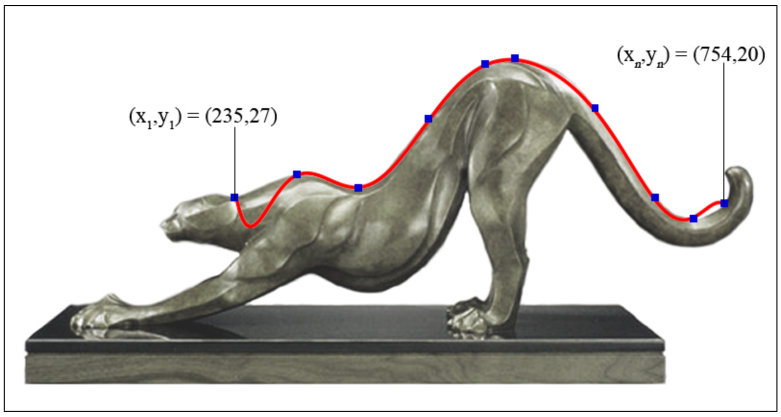


Figure 4.2: Interpolating ten points with the Lagrange polynomial.

curve generated by the Lagrange polynomial $P(x)$ from Example 4.2 on the closed range $x \in [235, 365]$ in red.

Suppose we now add seven more points to T, choosing specific points from the top profile of the cheetah until we reach its tail. The generated 10th-order Lagrange polynomial returns the curve on the closed interval $x \in [235, 754]$, as depicted in Figure 4.2. The generated curve steadily approximates the profile between the second point and the point before the last. We observe two small fluctuations at both endpoints, which resulted from the curved being forced to pass through all ten points in T.

The fluctuations become even more uncontrollable when we further add four more points at the head of the cheetah. This is evident in Figure 4.3. The 14th-order curve passes through all 14 points in T, but clearly, the resultant interpolation is undesirable.

In general, we learn two important lessons from this example. Firstly, forcing

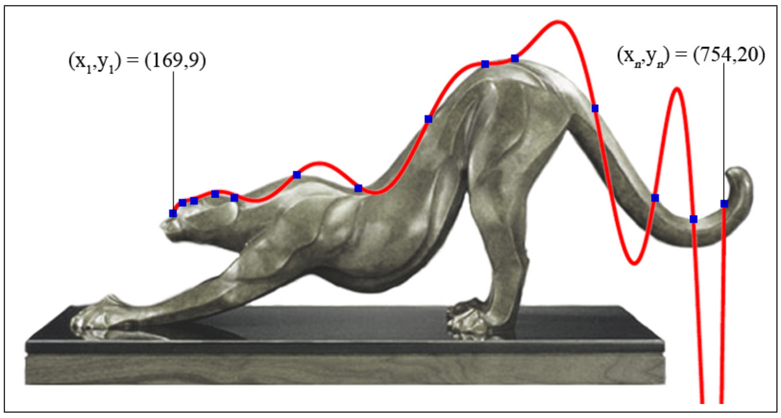


Figure 4.3: Interpolating 14 points with the Lagrange polynomial.

a single algebraic polynomial curve to pass through n points, if n is large, may lead to unpredictable fluctuations. The oscillatory nature of high-degree polynomials and the property that a fluctuation over a small portion of the interval can induce large fluctuations over the entire range restricts their use – this problem is known widely as the Runge phenomenon [2, 35]. This is in addition to the expensive precomputations to determine all n coefficients of $P(x)$. And secondly, a smaller n tend to produce acceptable and stable arcs, so that we can join several successive arcs to approximate a longer curve. These arcs can be made flexible and easy to control as we shall discuss in the next two sections.

It is because of these reasons that the state-of-the-art of approximating a set of points applies the techniques of procuring only second- and third-order spline curves. The points in the given set are grouped according to segments, and on each segment, a spline curve is described for it. The variety of spline curves dictates the behaviour of the overall approximation, in terms of the efficiency in instantiating them as well as the efficiency of controlling them to fit a set of criteria.

Interchangeably, we shall use the word ‘polynomial’ to also mean ‘curve’, from what we have already gathered in this section. We shall now take a look at two very popular controllable curves in the upcoming sections; the cubic splines and Bézier curves. Together they provide us with the fundamentals of the ‘piecewise-polynomial approximation’ method.

4.3 Cubic Splines

A set of piecewise-third-order polynomials constitutes the cubic spline, and it passes through a set of n rudimentary points. A general cubic polynomial involves four constants, and hence, this provides sufficient flexibility in the cubic spline procedure. It also ensures that the interpolant is continuously differentiable. The second derivative of each polynomial is commonly set to zero at the endpoints, since this provides a boundary condition that completes the system of $n-2$ equations. This forces the spline to be straight lines outside the endpoints, while not disrupting its smoothness, and leads to a simple tridiagonal system which can be solved easily to give the coefficients of the polynomials.

Definition 4.3 *Let $T = \{(x_1, y_1), \dots, (x_n, y_n))\}$ be an x-monotone Trace containing n points on the 2D plane, so that $x_1 < x_2 < \dots < x_n$. Then $S(x)$ is the* cubic spline interpolant *if there exists n cubic polynomials $S_k(x)$ with the coefficients a_k, b_k, c_k, and d_k, so that*

$$S_k(x) = S(x) = a_k(x - x_k)^3 + b_k(x - x_k)^2 + c_k(x - x_k) + d_k, \tag{4.2}$$

for $x \in [x_k, x_{k+1}]$, where $k = 1, \dots, n-1$, and further satisfying the four following properties:

(i) That the spline passes though all n points; i.e. $S(x_k) = y_k$ for $k = 1, \dots, n$;

(ii) That the spline forms a continuous curve over the entire interval $[x_1, x_n]$; i.e. $S_k(x_{k+1}) = S_{k+1}(x_{k+1})$ for $k = 1, \dots, n-2$;

(iii) That the spline forms a smooth function; i.e. $S'_k(x_{k+1}) = S'_{k+1}(x_{k+1})$ for $k = 1, \dots, n-2$; and

(iv) That the second derivative is continous; i.e. $S''_k(x_{k+1}) = S''_{k+1}(x_{k+1})$ for $k = 1, \dots, n-2$;

The first and second derivatives are fundamental to this cubic spline procedure, and with reference to Equation 4.2, they are

$$S'_k(x) = 3a_k(x - x_k)^2 + 2b_k(x - x_k) + c_k; \text{ and} \tag{4.3}$$
$$S''_k(x) = 6a_k(x - x_k) + 2b_k. \tag{4.4}$$

Now let $x = x_k$, then from Equation 4.2, we get

$$\begin{aligned} S_k(x_k) &= d_k; \text{ and} \\ S_{k-1}(x_k) &= a_{k-1}(x_k - x_{k-1})^3 + b_{k-1}(x_k - x_{k-1})^2 + c_{k-1}(x_k - x_{k-1}) + d_{k-1}. \end{aligned}$$

From property *(ii)* in Definition 4.3, we know that $S_{k-1}(x_k) = S_k(x_k)$. Then, $h = x_k - x_{k-1}$, the following is true:

$$\begin{aligned} d_k &= a_{k-1}h^3 + b_{k-1}h^2 + c_{k-1}h + d_{k-1} \\ \Leftrightarrow d_{k+1} &= a_k h^3 + b_k h^2 + c_k h + d_k. \end{aligned} \tag{4.5}$$

Similarly, from the first derivative of $S_k(x)$ in Equation 4.3,

$$\begin{aligned} S'_k(x_k) &= c_k; \text{ and} \\ S'_{k-1}(x_k) &= 3a_{k-1}(x_k - x_{k-1})^2 + 2b_{k-1}(x_k - x_{k-1}) + c_{k-1}. \end{aligned}$$

By property *(iii)* in Definition 4.3, $S'_{k-1}(x_k) = S'_k(x_k)$, so that

$$\begin{aligned} c_k &= 3a_{k-1}h^2 + 2b_{k-1}h + c_{k-1} \\ \Leftrightarrow c_{k+1} &= 3a_kh^2 + 2b_kh + c_k. \end{aligned} \tag{4.6}$$

Finally, from the second derivative of $S_k(x)$ in Equation 4.4,

$$\begin{aligned} S''_k(x_k) &= 2b_k; \text{ and} \\ S''_{k-1}(x_k) &= 6a_{k-1}(x_k - x_{k-1}) + 2b_{k-1}; \end{aligned}$$

and by property *(iv)* in Definition 4.3, $S''_{k-1}(x_k) = S''_k(x_k)$, we get

$$\begin{aligned} 2b_k &= 6a_{k-1}h + 2b_{k-1} \\ \Leftrightarrow 2b_{k+1} &= 6a_kh + 2b_k. \end{aligned} \tag{4.7}$$

Putting this all together, and let $M_k = S''_k(x_k) = 2b_k$, we solve for the coefficients a_k, b_k, c_k, and d_k of $S_k(x)$. By property *(i)* of Definition 4.3, $d_k = y_k$ when x is set to x_k. From Equation 4.7,

$$\begin{aligned} 6a_kh &= 2b_{k+1} - 2b_k \\ a_k &= \frac{M_{k+1} - M_k}{6h}. \end{aligned}$$

Rewriting Equation 4.5, we get

$$\begin{aligned} c_kh &= d_{k+1} - a_kh^3 - b_kh^2 - d_k \\ c_kh &= y_{k+1} - y_k - h^2(a_kh + b_k) \\ c_k &= \frac{y_{k+1} - y_k}{h} - \frac{h(M_{k+1} + 2M_k)}{6}. \end{aligned}$$

We now have the information to determine all the weights for our $n-1$ equations that describe all the cubic polynomials for the interpolated cubic spline on T. We relate the system of equations in matrix form as follows. From Equation 4.6, and substituting the derived expressions for a_k and b_k, in terms of M_k, we get

$$\begin{aligned} c_{k+1} &= 3a_kh^2 + 2b_kh + c_k \\ \frac{y_{k+2} - y_{k+1}}{h} - \frac{h(M_{k+2} + 2M_{k+1})}{6} &= 3\frac{M_{k+1} - M_k}{6h}h^2 + M_kh + \frac{y_{k+1} - y_k}{h} - \frac{h(M_{k+1} + 2M_k)}{6} \\ \frac{h}{6}(M_k + 4M_{k+1} + M_{k+2}) &= \frac{1}{h}(y_k - 2y_{k+1} + y_{k+2}) \\ M_k + 4M_{k+1} + M_{k+2} &= \frac{6}{h^2}(y_k - 2y_{k+1} + y_{k+2}), \end{aligned}$$

for $k = 1, \dots, n-2$. This gives the matrix equation

$$\begin{pmatrix} 1 & 4 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 4 & 1 \end{pmatrix} \begin{pmatrix} M_1 \\ M_2 \\ M_3 \\ \vdots \\ M_{n-1} \\ M_n \end{pmatrix} = \frac{6}{h^2} \begin{pmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-2} - 2y_{n-1} + y_n \end{pmatrix}. \qquad (4.8)$$

The system in Equation 4.8 is under-determined, as the main matrix is made up of $n-2$ rows and n columns. For us to generate a unique cubic spline, we need to institute two other conditions upon the system, and in doing so, we get several different types of cubic splines. We highlight three of them – the natural cubic spline, the parabolic runout spline, and the cubic runout spline – in the following sub-sections.

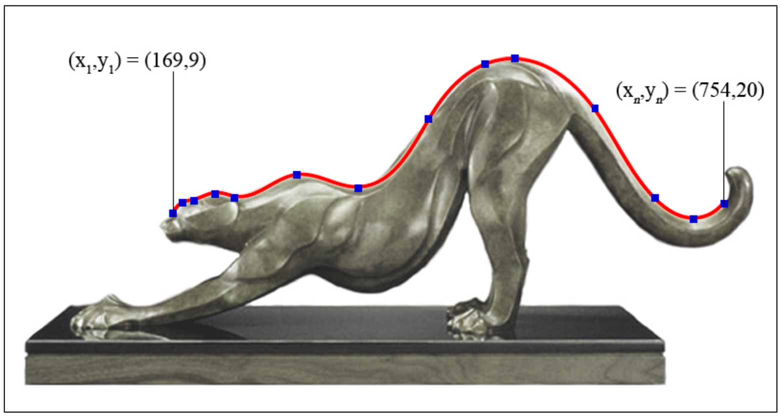


Figure 4.4: Interpolating 14 points with the Natural Cubic Spline.

4.3.1 Natural Cubic Spline

The natural cubic spline, as what we have been discussing thus far, stipulates that the spline extends itself as a simple line outside the endpoints. This means that the second derivatives at the endpoints are equal to zero, so that

$$M_1 = M_n = 0.$$

As such, we can remove the first and last columns of the matrix in Equation 4.8 since they correspond to M_1 and M_n, which are both zero. The resultant $(n-2)$ by $(n-2)$ matrix is unique and will determine the remaining solutions for M_2

to M_{n-1}.

$$\begin{pmatrix} 4 & 1 & 0 & \dots & 0 & 0 \\ 1 & 4 & 1 & \dots & 0 & 0 \\ 0 & 1 & 4 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 4 \end{pmatrix} \begin{pmatrix} M_2 \\ M_3 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{pmatrix} = \frac{6}{h^2} \begin{pmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-2} - 2y_{n-1} + y_n \end{pmatrix}. \tag{4.9}$$

4.3.2 Parabolic Runout Spline

McKinley and Levine [77] prescribed the parabolic runout spline to obey the conditions that the second derivatives at the endpoints M_1 and M_n are equal to M_2 and M_{n-1}, respectively, so that

$$M_1 = M_2, \text{ and } \; M_n = M_{n-1}.$$

This forces the spline to extend itself as a parabolic curve at its endpoints. McKinley and Levine claim that this type of cubic spline is useful for approximating periodic and exponential data. We state the matrix expression for for the parabolic runout spline in Equation 4.10.

$$\begin{pmatrix} 5 & 1 & 0 & \dots & 0 & 0 \\ 1 & 4 & 1 & \dots & 0 & 0 \\ 0 & 1 & 4 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 5 \end{pmatrix} \begin{pmatrix} M_2 \\ M_3 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{pmatrix} = \frac{6}{h^2} \begin{pmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-2} - 2y_{n-1} + y_n \end{pmatrix}. \tag{4.10}$$

4.3.3 Cubic Runout Spline

Finally, we can force the spline to degrade to a single cubic curve over the last two intervals at both its endpoints [77]. This results in a more pronounced curvature at the endpoints when compared to the natural and parabolic runout splines. By assigning

$$M_1 = 2M_2 - M_3, \text{ and } \; M_n = 2M_{n-1} - M_{n-2},$$

we get the matrix expression in Equation 4.11.

$$\begin{pmatrix} 6 & 1 & 0 & \dots & 0 & 0 \\ 1 & 4 & 1 & \dots & 0 & 0 \\ 0 & 1 & 4 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 6 \end{pmatrix} \begin{pmatrix} M_2 \\ M_3 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{pmatrix} = \frac{6}{h^2} \begin{pmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-2} - 2y_{n-1} + y_n \end{pmatrix}. \tag{4.11}$$

4.3.4 Other Types of Cubic Splines

The vast literature contains many other types of cubic splines, both popular and unpopular, but we shall not go into details discussing every one of them here. The setting up of all cubic splines, in general, are similar to the ideas that we summed up and presented in this chapter. However, to end our current deliberations before moving on to the Bézier curves, let us briefly mention four other cubic splines that came across us as interesting; the Clamped Spline, the Cardinal Spline, the Catmull-Rom Spline, and the B-Spline.

A Clamped Spline satisfies the boundary conditions that $S'(x_1)$ is equal to the derivative at $p_1 \equiv (x_1, y_1) \in T$, and similarly $S'(x_n)$ equals the derivative at $p_n \equiv (x_n, y_n) \in T$ [12]. The matrix in Equation 4.8 can be easily reformulated to reflect these constraints.

The Cardinal Spline takes the positions of the current point and its two immediate neighbours, and averages out the positions using a tension value between 0 and 1 [117]. This smooths the line and makes a path that is gently curved through the points rather than zig-zagging through them.

Catmull-Rom Splines are formulated such that the tangent at each point $p_i \in S(x)$ is calculated using the previous and next points on the spline. They are continuous and differentiable throughout the closed interval, have local control and interpolations, but they *do not* lie within the convex hull of their control points [14, 27].

Finally, the B-Splines (short for *basis*-Splines) of Schoenberg share many important properties with Bézier curves, because the former is a generalization of the latter [97, 16]. They refer to spline curves parameterised by spline functions that are expressed as linear combinations of the B-Splines. They are simply a generalization of the Bézier curves, which we shall discuss in the next section, and they can avoid the Runge phenomenon without increasing the degree of the B-Splines.

However, B-Splines are still polynomial curves and polynomial curves cannot represent many useful simple curves such as circles and ellipses. Thus, a generalization of B-spline, NURBS (Non-Uniform Rational B-Spline), is required, and we shall discuss its development through the Bézier curves.

4.3.5 Related Works Involving Cubic Splines

In the field of geology, the main complication in geological shape analysis is that many shapes are difficult to orient and cannot be represented as single-valued functions in either Cartesian or polar coordinates. Evans *et al.* [31] showed that the parametric cubic spline is a versatile solution to describe such shapes, especially when studying the outlines of microfossils. Their spline interpolation technique is applied to the Archimedean Spiral and to the Rock Fold classification. The former is a smooth curve containing great variations in curvature, and the latter is a classification process for shapes of rock folds based on the orthogonal thickness curve or by the amount of convergence in "dip isogons". Their results show that the spline curves were able to retain the convergence properties of the curves they want to approximate, but at a cost of including more data points.

Plass and Stone [98] presented their work on tackling the problems that arise when trying to use 2D curved shapes in an interactive design environment, where their design methodology was to represent all shapes analytically using piecewise-cubic polynomials. Their developed algorithm takes a set of sampled points and iteratively derives a single parametric cubic polynomial that lies close to the data points as defined by an error metric based on least-squares. The combination of this algorithm with dynamic programming to determine the crucial points from the sampled data set returned good results over a range of shapes and applications. Many other works on such a similar interactive design exist that involve cubic splines and, among the more interesting ones, are explained by Banks and Cohen [5] on real-time spline curves, by Toraichi *et al.* [122] on coding system for evaluating meat quality, and by Flory and Hofer [34] on fitting cubic curves on manifolds.

Metafonts and true-type fonts (TTFs) are wholely described by a set of instructional curves, and parts of the contours of these fonts involve cubic splines. Knuth's [58] Metafont manual highlights a system for the design of alphabets suited to raster-based devices that print or display text. More often than not, we also see several works involving the reverse engineering of this; that is, given a font's contour, the authors seek to capture the outline of the font [63, 136]. Sarfraz and Raza [63] combined spline techniques with genetic algorithms to search for candidate crucial points based on the Akaike's Information Criterion (AIC), which on the average, takes 50 epochs for the mutation algorithm to arrive at a solution.

4.4 Bézier Curves

Spline curves, we saw, were conceived with the intention that they interpolate the entire given set of n points in the plane. And as such, those points must be rigid in their own positions and rigid with respect to the positions of their immediate neighbours. In other words, if a single point is misplaced, then the impact and accuracy of the resultant spline curve may be affected. This is where regression comes in. A mixture of regression and interpolation techniques may prove useful when approximating a set of points whose positions we cannot completely trust, and the Bézier curves offer such a methodology.

Given a set of $m+1$ control-points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_m$, the parametric Bézier CURVE $K(t)$ of degree m is described using the Bernstein polynomial $B_{i,m}(t)$ as

$$K(t) = \sum_{i=0}^{m} B_{i,m}(t)\mathbf{P}_i = \sum_{i=0}^{m} \binom{m}{i}(1-t)^{m-i}t^i\mathbf{P}_i, \text{ for } t \in [0,1]. \tag{4.12}$$

For the same set of control-points, a *rational* Bézier CURVE $K(t)$ adds adjustable weights to provide closer approximations to arbitrary shapes. Its numerator is a weighted Bernstein-form Bézier CURVE and the denominator is a weighted sum of Bernstein polynomials.

$$K(t) = \frac{\sum_{i=0}^{m} B_{i,m}(t)\mathbf{P}_i\ w_i}{\sum_{i=0}^{m} B_{i,m}(t)w_i} = \frac{\sum_{i=0}^{m}\binom{m}{i}(1-t)^{m-i}t^i\mathbf{P}_i\ w_i}{\sum_{i=0}^{m}\binom{m}{i}(1-t)^{m-i}t^i w_i}, \text{ for } t \in [0,1]. \tag{4.13}$$

The Bézier CURVE $K(t)$ is derived from the recursive midpoint rule of de Casteljau [43]. Since this formation from the recursive midpoint rule links directly to our methodologies later, we shall then present de Casteljau's method in a more appropriate manner in section 8.1. The Bézier CURVE $K(t)$ has the following properties:

- It is closed under perspective transformation, and can represent conic sections exactly;
- It always passes through the first and last control-points, and lies within the convex hull of all the control-points;
- It is tangent to $\overline{\mathbf{P}_0\mathbf{P}_1}$ and $\overline{\mathbf{P}_{m-1}\mathbf{P}_m}$ at its endpoints; and
- It can be translated and rotated by performing these operations directly on the control-points.

4.4.1 Parametric Continuity

Normally a single Bézier CURVE is not enough to sufficiently and desirably define or approximate a complex shape. A piecewise construction technique is often unavoidable [27, 12]. In doing so, there are two types of continuity that we need to consider when joining two consecutive Bézier CURVEs; the parametric continuity, and the geometric continuity. We have already explained the latter in section 3.3, in terms of G^0, G^1, and G^2 (geometric) continuity.

Parametric continuity of the i^{th} degree is denoted by C^i. This means that two adjacent CURVEs have identical i^{th} degree parametric derivatives, as well as all lower derivatives. For example, let us reflect the first three degrees of parametric continuity as we did with the geometric continuity as follows:

- C^0 continuity: Two adjacent Curves share a common endpoint.
- C^1 continuity: Two adjacent Curves have the same tangent vector, both in magnitude and direction, at their common shared point.
- C^2 continuity: Two adjacent Curves have the same second-order parametric derivatives, both in magnitude and direction, at the common shared point, on top of having C^1 continuity.

In general, two Curves which are parametric-continuous to a certain degree is also geometric-continuous to that same degree. But the reverse is not so.

4.4.2 Controlling Cubic Bézier Curves

A cubic Bézier Curve $K(t)$, from Equation 4.12, is defined by four control-points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$ in the expression

$$K(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3. \tag{4.14}$$

Let $TS = \{p_1, \ldots, p_n\}$ be a TraceSegment such that the points in TS are aligned in a trajectory where the connected Edges form a path between p_1 and p_n with at most two extremums. Then the task is to find a suitable cubic Bézier Curve $K(t)$ whose endpoints are *anchored* at p_1 and p_n, and where the path between the Curve $K(t)$ and the TraceSegment TS satisfies the Weierstrass approximation stated in Theorem 4.1, for a predefined value of ϵ.

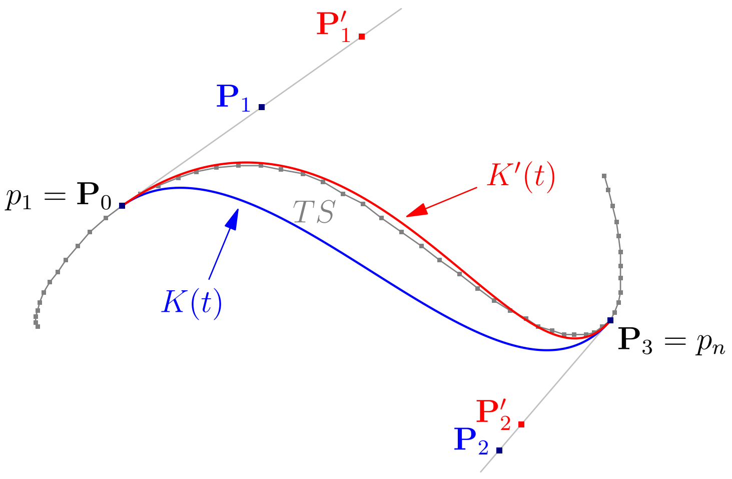


Figure 4.5: Estimating with a cubic Bézier Curve and the effects of repositioning the control-points $\mathbf{P}_1$ and $\mathbf{P}_2$ to $\mathbf{P}'_1$ and $\mathbf{P}'_2$, respectively, to get the corresponding Curves $K(t)$ and $K'(t)$.

At a glance, one can see that the intermediate points in TS play the important role of serving as guides in determining the shape and accuracy of $K(t)$. They may or may not lie on $K(t)$, and hence, are synonymous to the correlation coefficients in a regression process (in this case, the regression of points for a cubic Curve). In contrast to the spline techniques, the information drawn out from these points in TS are not explicitly used to directly generate $K(t)$. Instead, they are referred to implicitly, and it suffices to ensure that the parametric and geometric continuities at the anchor-points are maintained.

This implies that we fix the tangent lines through the anchor-points p_1 and p_n (which corresponds to the control-points $\mathbf{P}_0$ and $\mathbf{P}_3$ of $K(t)$) to ensure that smoothness is maintained between the neighbouring Curves. As a consequence, the positions of the coinciding control-points $\mathbf{P}_1$ and $\mathbf{P}_2$ are limited, and can be moved only up and down the two fixed tangent lines to arrive at a desirable approximation for TS.

4.4.3 Related Works Involving Cubic Bézier Curves

Several authors have showcased their works in obtaining the optimal positions for the control-points $\mathbf{P}_1$ and $\mathbf{P}_2$. Yang *et al.* [136] used a curve-fitness evaluation cost to guide the recursive adjustment routine of the positions for the control-points. We found their methods to be rather heuristic, and perhaps, suspicious to a point that desirable approximations can be achieved. Let us explain.

For any point $p_i \in TS$, there exists a corresponding parametric point $K(t_i)$ on the Bézier Curve that is the intersection of the line perpendicular to the baseline $\overline{\mathbf{P}_0\mathbf{P}_3}$. The curve-fitness cost function is the sum of the distance-squared between p_i and $K(t_i)$ expressed as follows:

$$D(K, TS) = \sum_{i=2}^{n-1} \|K(t_i) - p_i\|^2. \tag{4.15}$$

The task defined by Yang *et al.* was then to find a Bézier Curve $K(t)$ that will minimize $D(K, TS)$.

Let L_{T0} and L_{T3} be the tangent lines through the anchor-points $\mathbf{P}_0$ and $\mathbf{P}_3$, measured from the data set in TS. Let v_1 and v_2 respectively mark the $\frac{1}{3}$ and $\frac{2}{3}$ relative positions of the baseline distance from $\mathbf{P}_0$. Let L_1 and L_2 be the lines through v_1 and v_2 that are perpendicular to the baseline $\overline{\mathbf{P}_0\mathbf{P}_3}$. Then, initially, Yang *et al.* set the control-point $\mathbf{P}_1$ as the intersection point of L_{T0} and L_1. Similarly, the control-point $\mathbf{P}_2$ is the intersection of L_{T3} and L_2. We give an example of the notations in Figure 4.6.

Next, we summarize their iteration step as follows:

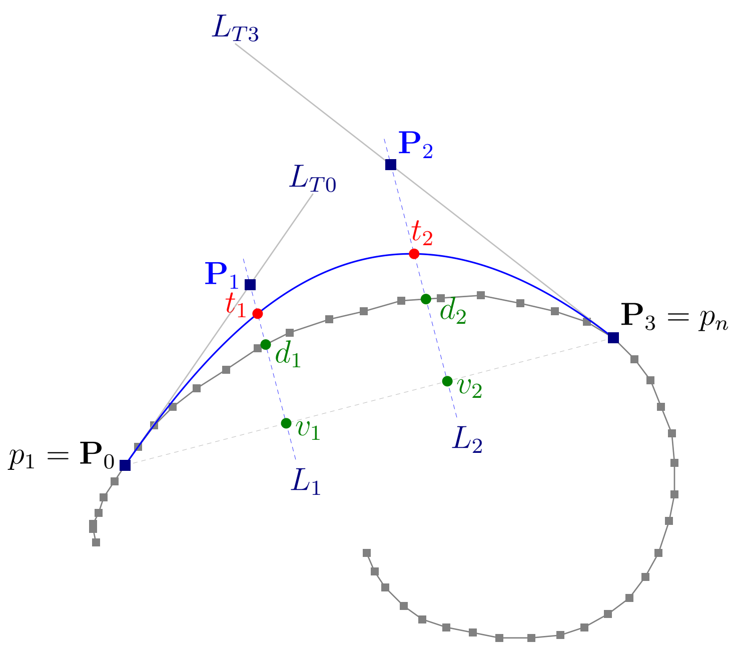


Figure 4.6: Determining $\mathbf{P}_1$ and $\mathbf{P}_2$ by Yang *et al.*

- Let t_1 and d_1 be the points where L_1 cuts the instantiated Curve $K(t)$ and the TraceSegment TS, respectively. Similarly, let t_2 and d_2 be the points cut by L_2. Let us denote them simply as t_i and d_i, for $i = 1, 2$.
- If $\|t_i - d_i\| > 0$ then $\mathbf{P}_i$ is moved downwards along L_i for a distance of $\frac{1}{10}\|\mathbf{P}_i v_i\|$, otherwise, it is moved upwards for the same distance ratio.
- If $\|t_i - d_i\| = 0$ but $D(K, TS)$ is large, then $\mathbf{P}_1$ is moved to the left, parallel to the baseline, at half the distance between $\|\mathbf{P}_0 v_1\|$. The other control-point $\mathbf{P}_2$ is moved to the right, parallel to the baseline, at half the distance between $\|v_2 \mathbf{P}_3\|$.

At every new positions of $\mathbf{P}_1$ and $\mathbf{P}_2$ at the end of one iteration step, a new Curve $K(t)$ is computed, and so are the points t_i, d_i, and v_i corresponding to the new line L_i. The recursion continues until $D(K, TS)$ reaches a suitably small value.

In their results, Yang *et al.* showed that it takes on average only 10 iterations before 100% fit is achieved. We find their results rather debatable mainly because repositioning the control-points $\mathbf{P}_1$ and $\mathbf{P}_2$ as they did break the C^1 (and C^2) continuities at the endpoints of $K(t)$. And so, while the approximations were deemed desirable from the curve-fitness cost function, we suspect that the Curves generated may not be visually correct.

Sarfraz *et al.* [112] offered a more mathematical initial approach in their recursive technique to determine the starting positions for $\mathbf{P}_1$ and $\mathbf{P}_2$. They made a claim that from the cubic expression in Equation 4.14, the blending coefficients $3t(1-t)^2$ and $3t^2(1-t)$ corresponding to the control-points $\mathbf{P}_1$ and $\mathbf{P}_2$, "maximizes" at $t = \frac{1}{3}$ and $t = \frac{2}{3}$, respectively. These occur at the original points $p_{(n-1)/3} \in TS$ and $p_{2(n-1)/3} \in TS$, where n is the total number of points in the original TraceSegment. The two unique conditions, $K(t = \frac{1}{3}) = p_{(n-1)/3}$ and $K(t = \frac{2}{3}) = p_{2(n-1)/3}$, are then used to solve the simultaneous equations for $\mathbf{P}_1$ and $\mathbf{P}_2$;

$$p_{(n-1)/3} = (1-\frac{1}{3})^2\mathbf{P}_0 + 3\frac{1}{3}(1-\frac{1}{3})^2\mathbf{P}_1 + 3(\frac{1}{3})^2(1-\frac{1}{3})\mathbf{P}_2 + (\frac{1}{3})^3\mathbf{P}_3, \text{ and}$$
$$p_{2(n-1)/3} = (1-\frac{2}{3})^2\mathbf{P}_0 + 3\frac{2}{3}(1-\frac{2}{3})^2\mathbf{P}_1 + 3(\frac{2}{3})^2(1-\frac{2}{3})\mathbf{P}_2 + (\frac{2}{3})^3\mathbf{P}_3;$$

where $\mathbf{P}_0 = p_1$ and $\mathbf{P}_3 = p_n$ are the known endpoint coordinates of TS.

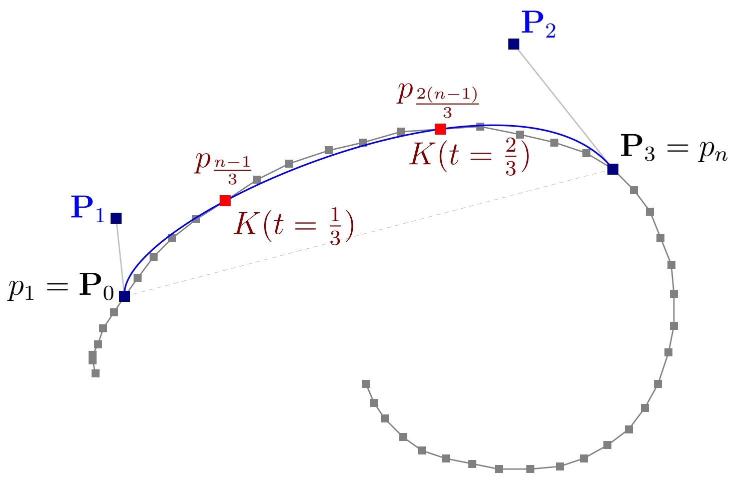


Figure 4.7: Determining initial $\mathbf{P}_1$ and $\mathbf{P}_2$ by Sarfraz. This ensures that the CURVE $K(t)$ passes through the unique conditions at $K(t = \frac{1}{3}) = p_{(n-1)/3}$ and $K(t = \frac{2}{3}) = p_{2(n-1)/3}$. However, as evident from this figure, the geometric and parametric continuities at the anchor-points $\mathbf{P}_0$ and $\mathbf{P}_3$ are broken.

The same curve-fitness cost function from Equation 4.15 is used to measure the error of the generated $K(t)$ against the original points in TS. However, unlike the recursive steps in Yang *et al.*'s method, Sarfaz *et al.* repositioned the control-points $\mathbf{P}_1$ and $\mathbf{P}_2$ strictly to lie along the tangent lines L_{T0} and L_{T3}, respectively.

In another study, Pal *et al.* [94] proposed a similar method to Sarfaz *et al.*'s, except that they used $t = \frac{1}{4}$ and $t = \frac{3}{4}$ instead. Their speciality field on face-recognition is the grounds behind their observation for the two values of t.

The remainder formula composed by Sederberg and Farouki [113] is a technique for estimating known and well-defined mathematical functions with the cubic Bézier CURVES. It evolves around the Lagrange polynomials we discussed earlier in section 4.2. However, we note that their method is not suitable for approximating unknown and arbitrary shapes.

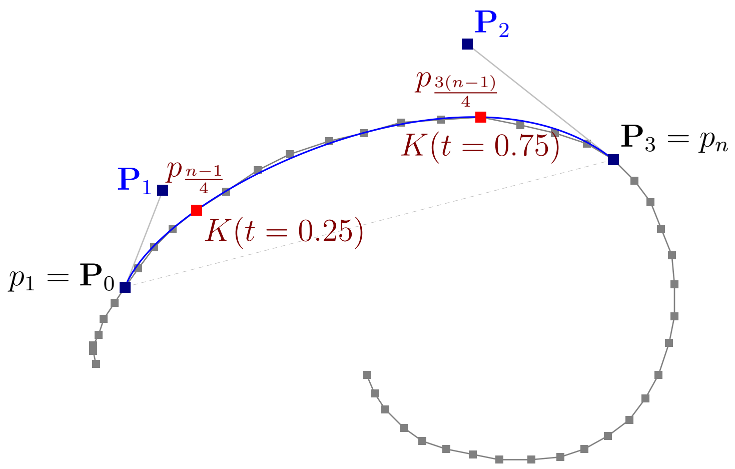


Figure 4.8: Determining initial $\mathbf{P}_1$ and $\mathbf{P}_2$ by Pal *et al.*. This ensures that the CURVE $K(t)$ passes through the unique conditions at $K(t = 0.25) = p_{(n-1)/4}$ and $K(t = 0.75) = p_{3(n-1)/4}$. Also in this particular example, the geometric and parametric continuities at the anchor-points $\mathbf{P}_0$ and $\mathbf{P}_3$ are broken.

We mentioned earlier that non-rational CURVES cannot desirably estimate and cannot always satisfy the Weierstrass approximation theorem. There are very few authors who opted to use the rational version of the cubic Bézier CURVES, as these techniques are often very complex as we need to determine the weights w_0, w_1, w_2, and w_3, in addition to finding the optimal positions for the control-points $\mathbf{P}_1$ and $\mathbf{P}_2$. Sarfraz demonstrated this with the rational cubic interpolant based on the cubic Hermite function [111].

4.5 Are Quadratic Curves Sufficient?

After reviewing the methodologies behind the cubic curves, we ask ourselves if this approximation with third-order curves is really necessary. The main reason, we think, that cubic Bézier curves were used was because it originated from the manual method of *hinting* – a user actually has to be present in front of the application program and manually set the control-points $\mathbf{P}_1$ and $\mathbf{P}_2$ until he/she deems the arc approximating that particular portion of the whole curve is visually satisfactory. Thus, the next logical question is to ask if it is worth to spend processor resources for the hinting subroutines?

This thesis recommends that second-order curves are sufficient. This is rather evident from the works of various authors that we highlighted in this chapter.

That is, a 'cubic' curve is used to estimate an underlying set of points that are inherently 'quadratic'.

Already, if we apply Proposition 3.12, then it is sufficient that TS is made up of only a set of second-order CURVEs. Granted that the correct placements and accuracies of the combined CURVEs depend heavily on selecting the most appropriate crucial points among all the p_i in TS, but we shall see in the next chapter that this can be done in a rather efficient manner. This sub-problem of determining the optimal crucial points in TS is analogous to the polyline simplification problem, and the results of which we use to build our own smoothing algorithm in the later chapters.

The rest of the thesis reviews and proposes two other approximation techniques for T, both of which involve only second-order CURVEs. Earlier on, we stated our observations that polynomial curves cannot represent circles and ellipses, but in chapter 7 we prove that the opposite is true – that the scattered and indeterministic points in handwritten TRACEs can be desirably approximated based solely on elliptic arcs. The second technique in chapter 8 returns our deliberations back to the Bézier CURVEs, where we put together a technique involving rational quadratic Bézier CURVEs.

4.6 Symbolic Representation

At the start of section 4.1, we pointed and reasoned out why a CURVE should not be treated simply as a container to hold a set of 'smoothed' points. What is essential is that it holds just enough information, as a set of agreed instructions for the curve-rendering application to act upon. In the abstract sense, this set of limited instructions in a CURVE is a symbolic representation of the actual curve that is to be drawn on screen, or prepared for print. Ideally, a symbolic CURVE must be:

- Reproducible - the representation should give the same CURVE every time;
- Computationally quick and accurate;
- Easy to manipulate (under Euclidean transformations);
- Flexible; and
- Easy to combine with other segments of the CURVE.

These are rather similar to the definitions stated in Guru and Nagendraswamy [39] of their representation of 2D shapes by symbolic features. The symbolic features preserve both contour and region information, and are translation, rotation, reflection, and scale invariant.

Often it makes sense to have the represented information accorded firmly to a point (or points) on the 2D plane at zoom level 1 (original 1-on-1 display resolution), and then augment additional facts to the point (or points). For example, in the case of the cubic spline curves, such symbolic information would be the actual point p_k of which the Curve will pass through, as well as appending all four dully-computed cubic coefficients a_k, b_k, c_k, and d_k that guides the formation of the Curve through p_k. In the case of the Bézier curves, simply storing the control-points (and the rational weights, if a rational curve is used) are sufficient, since each curve can be independently distinguished from its neighbours.

Thus, a set of symbolic Curves representing a single Trace containing n points will maintain an $O(n)$ storage space. They also aid in detailing instructions to the rendering program, which can then decide at how fine a resolution is needed to draw the curves in order to provide the best visual impact under any zoom level.

CHAPTER 5

Polyline Simplification

We laid the grounds in chapter 3 to identify the crucial points in a TRACE T. Suppose now we decide to keep only those crucial points in T and remove all other non-crucial points. Then this is congruous to the solutions of the polyline simplification problem.

The main goal of the polyline simplification problem is to reduce the line data substantially, while still preserving its appearance with respect to the original. In other words, we keep only the critical points in the input polygonal path that are vital to the overall shape of the polyline. The rest of the points are removed (or not) depending on the desired level of simplification. The process of simplifying lines is widely studied in the field of cartography [55, 79, 80], where cartographers use different levels of details on different levels of scales for displaying the same geographic data. From among the numerous techniques, we shall focus our attention to three of the most prominent ones in this chapter, that can be closely related to our problem of approximating curves over the digital handwritings. They are namely, the Douglas-Peucker, the Reumann-Witkam, and the Opheim [30, 102, 93] algorithms.

While the methods differ in many ways from the active-sampling routine we proposed earlier, we are interested to know if the simplified polylines contain exactly those points that we would deem crucial, particularly the inflection points and the sharp-edge vertices. If such is the case, then we can draw a conclusion in this chapter and continue to build on our earlier grounds and support McMaster's conjectures [80] to integrate the simplification and smoothing algorithms in line generalization.

5.1 Categorization of Algorithms

Classical polyline simplification is the selective removal of ‘unwanted’ points, based on a predefined tolerance value, that is made to serve two main objectives; to reduce data volume, and to maintain the quality of polylines when scale is reduced. Many algorithms have been put forward for this task of finding a subset of the original set of points from a given polyline that is small enough, and yet, in its simplified form, *desirably* approximates the original.

These algorithms were classified by McMaster [80] into five categories, in terms of their handling of which of the n points in the original polyline to keep:

- Independent Point Routine
- Localized Processing Method
- Constrained Extended Local Processing Routine
- Unconstrained Extended Local Processing Routine
- Global Routine

Methods in the first four categories are sequential and process points in order of their appearance from start to end. The global routines in the last category process the polyline in its entirety.

The simplest of the five categories is the Independent Point Routine, where all mathematical relationships between points are ignored, and the choice of which points to retain is based on an independent criterion (or criteria). Two notable examples where the algorithms under this category are computationally efficient but give poor results are: the ‘n-th point routine’, where every n-th coordinate point along the traversal is kept; and the ‘random point selection routine’, where $1/n$ of the original points are randomly retained. Although the resultant simplified polylines are unacceptable if high quality results are desired, McMaster still regarded them as useful preprocessing steps especially for thinning out very dense coordinate strings.

The Localized Processing Method takes into account the immediate neighbours of a point. For example, the brute-force Vertex Reduction algorithm [121] weeds out successive points that are clustered too closely to a single point p_i that comes to within a predefined radial distance ϵ centred at p_i. There are many more methods falling into this category, using more or less similar approaches to the Vertex Reduction algorithm, including an algorithm that thresholds the distance between a point and the line connecting its immediate neighbours, and another that works with the angles between line-segments.

Both the Constrained and Unconstrained Extended Local Processing Routines consider points that are beyond the immediate neighbourhood and process entire segments of the polyline. The Reumann-Witkam [102] algorithm in section 5.4 is one such routine. It determines the *trend* of the original line at a particular point and eliminates all subsequent points that do not follow this trend. Other examples include the Opheim algorithm [93] as well as the Reumann-Witkam-Douglas-Peucker algorithm [104], which we shall explain in further details in section 5.5.

The one popular and widely used member of the fifth category is Douglas-Peucker algorithm [30]. This Global Routine explained in section 5.3 uses a divide-and-conquer approach, based on a predefined tolerance value ϵ. It has the uncanny ability to accurately select critical points, but which comes at a cost of quadratic worst-case runtime.

We rate the quality of a simplified polyline higher, the closer it resembles the original polyline, taking into account the natural trade-off between the reduction rate of the simplification and its accuracy. Because of this, all simplification algorithms induce positional errors in the data set. Like a filter, though, the simplification process may turn out to be a precursor advantage to our cause when viewed from the applicative domain with the handwritten TRACES. We will see later on that a simplified polyline is analogous to the result of cleansing a Trace from noisy data.

5.2 Measuring the Quality of Simplified Polylines

Quantifying the goodness of simplified polylines is not a straight-forward preconception, if the polylines have characteristics of handwritten TRACES. Still, we have to put together a set of measurable criteria so that we can compare the different line simplification algorithms.

To begin with, one can say that a good approximation of a polyline is the simplified polyline that never deviates much from the original. However, we do not want to end up with an approximation that is the exact copy of the original – in which case, there is *no deviation* at all between the approximation and the original. Thus, putting this in another way, one can better say that a good approximation of a polyline is the simplified polyline where only the important points in the original are kept, and the rest of the points forming unimportant details of the polyline are eliminated. We consider a point important or *critical* if it has a visible influence on the overall shape of the polyline.

From an abstract viewpoint, a critical point is one where the *direction* of the polyline changes notably. If we relate this notion to the previous chapter, then a 'critical' point in a polyline is equivalent to a 'crucial' point in a TRACE. However, there is a subtle difference between the two terms in the perspective of the polyline versus the perspective of the TRACE. In the latter, the crucial points are the proponents we need to convert the TRACE into a smooth CURVE entity, and it may or may not necessarily be the case that a set of crucial points in a TRACE contains all the critical points if the TRACE is treated as a polyline.

There were several psychological studies related to the science of cartography where participants were asked to determine the critical points of a given set of polylines extracted from maps, based solely on their visual impressions. A critical point in this application domain is a point fitting our abstract viewpoint above, as well as having incorporated in it information about the geometric structure of geographic phenomena [71]. In one analysis, Marino [70] collated the results from the participants and used them to establish new line simplification algorithms. The highly heuristical algorithms produced various degrees of outcomes and, unfortunately, did not become popular among cartographers. In another analysis, White [134] used the results as benchmarks to examine how good different algorithms were at selecting those marked critical points. Though the main setback was that different groups of people gave different sets of critical points.

In lieu of this, we shall then stick to the standard mathematical measures instead, to determine the quality of simplified lines. Particularly, we shall pay close attention to McMaster's six numerical measures, reduced from his original 30, used to evaluate the differences between an original polyline and its simplified version [78].

McMaster categorised his measures into two groups; (I) the Linear Attribute Measurements, and (II) the Linear Displacement Measurements. In the first group, the original and simplified polylines are first treated independently, by computing their line attributes, such as angularity and change in the number of coordinates, in their entirety, before comparing the numbers. Measures in the second group, on the other hand, compare the differences such as vector and areal displacements *directly*, point to point. We give a summary overview of the two measures in Tables 5.1 and 5.2.

So, we can safely postulate that the smaller the measured values between the original and simplified polylines are, the better the quality of the simplification, and thus more importantly, the more accurate the positions of the identified critical points from the original polyline are. This suggests that we are measuring some sort of an error signal. But McMaster's various measurements do not really tell much if we use them independently. When combined, though, we can expect to get a measure that is more reliable.

In our combined and slightly modified version of the error measurements, we look to the eliminated points of the original polyline and calculate their euclidean distances with respect to the line-segments in the simplified polyline that replaced those points. We add up the squared distances, divide the sum by the number of points in the original polyline, and then take its square root.

No.	**Measure**	**Description**
1.	Percentage change in the number of coordinates	Divides the number of points in the simplified polyline by the number of points in the original. *Simple measure for the effectiveness of a simplification.*
2.	Percentage change in the standard deviation of the number of coordinates, per inch	Indicates whether a simplified polyline has uniform density compared to the original. *Useful if the baseline was accidentally digitized with varying density.*
3.	Percentage change in angularity	Evaluates how much variation in the original polyline was removed.

Table 5.1: McMaster's Linear Attribute Measurements.

No.	**Measure**	**Description**
1.	Total vector displacement, per inch	Calculates the geometric shift of the simplified polyline against the original. *An approximation is deemed better, the less displacement it produces.*
2.	Total areal displacement, per inch	As above.
3.	Percentage change in the number of curvilinear segments	Curvilinear segments are portions of a line where all changes of direction have the same positive or negative sign, capturing the general trend of the line. *In many cases, this measure is similar to the angularity measure.*

Table 5.2: McMaster's Linear Displacement Measurements.

Let $T = \{p_1, \ldots, p_n\}$ be a polyline containing n points. Also, for any eliminated point $p_i \equiv (x_i, y_i) \in T$, let $p'_i \equiv (x'_i, y'_i)$ be a corresponding point on the line-segment $\overline{p_r p_s}$ connecting two *kept* points $p_r, p_s \in T$, where $r < i < s$. Then, the square of the error measure err_T of the simplified polyline, with respect to the original T, is computed as

$$\mathrm{err}_T^2 = \frac{1}{n} \sum_{i=0}^{n} (x_i - x'_i)^2 + (y_i - y'_i)^2. \tag{5.1}$$

This way, we obtain an error measurement that is independent of the length of the polyline T, and is analogous to the standard deviation in probability theory. Furthermore, this has two advantages over McMaster's areal displacement measure, in that firstly, it prevents a simplification from deviating too much from

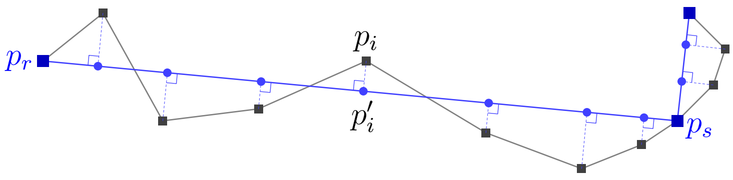


Figure 5.1: Measuring error of a simplified polyline.

the original line in any point (thus strengthening the choice of critical points), and secondly, it punishes strong deviations in few points more severely than small deviations in many points. Both of these advantages are intuitively desired properties in deciding a simplified polyline's quality; that is, the smaller this error measurement is, the better the approximation would seem to the human eye.

The example figures in the next few sections, as we discuss the various line simplification algorithms, all contain values of the error measurements of the various simplified polylines against one original example. We shall use these to illustrate visually our last statement as we progress towards the optimal algorithm in section 5.6.

5.3 The Douglas-Peucker (DP) Algorithm

The Douglas-Peucker (DP) algorithm is the most popular among the many classical simplification routines in the literature [30]. In a related study, McMaster showed that the DP algorithm produces the best simplification results when compared against seven other methods [80]. Jenks even claimed that the resultant polyline yielded by the DP algorithm is "perceptually superior" to that produced by the Distance algorithm and the Angular-Change algorithm [55]. The conception of several new variants of the DP algorithm is further proof of this [44, 126, 101], mainly with techniques to speed up the DP simplification procedure.

McMaster's dubbing of the DP algorithm being a Global Routine algorithm [79] draws its similarity to it being an *offline* algorithm in the geometric sense. That is, the DP algorithm requires the full knowledge of all the points in a given polyline in order for it to deduce the near-optimal critical points' positions. This is as opposed to the *local* routine algorithms that we shall discuss in Reumann-Witkam's [102] and Opheim's [93] algorithms.

Using a 'divide-*refine*-and-conquer' approach, the DP algorithm begins by building a very rough approximation of the original line, connecting the endpoints. Let us, in general, term these two endpoints as p_A and p_B. Subsequently, all the intermediate points lying in the trajectory between p_A and p_B are tested until a single point p_i is found and marked as critical, if and only if it satisfies

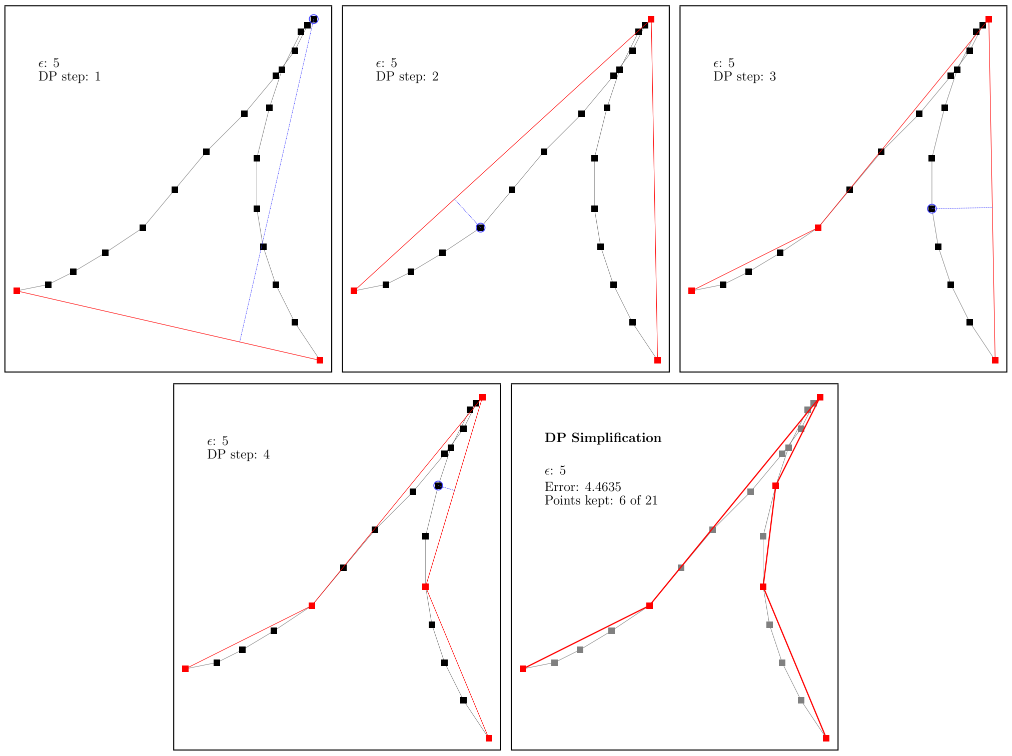


Figure 5.2: The Douglas-Peucker (DP) algorithm.

the following two conditions:

(i) That p_i is the farthest point away from the line $\overline{p_A p_B}$; and

(ii) That the orthogonal distance between p_i and $\overline{p_A p_B} > \epsilon$, where ϵ is a predetermined corridor distance.

If such a p_i exists, then the DP algorithm repeats the above step with two divided segments with the new endpoints p_A and p_i, and p_i and p_B. Otherwise, it terminates. We give an example of the DP algorithm in Figure 5.2.

The worst-case runtime of the naïve DP algorithm is $O(n^2)$, while on average it takes $O(n \log n)$ time. Hershberger and Snoeyink [44] developed a more sophisticated version of the DP algorithm that reduces the worst-case runtime. The speed up factor is achieved by noting that the farthest intermediate point p_i must be on the convex hull of the polyline segment from p_A to p_B. The hull is constructed in $O(n)$ time using Melkman's algorithm [81], and it takes $O(\log n)$ time to perform a binary search on the hull's vertices for p_i. Hershberger and Snoeyink showed a delicate way of how to reuse the hull information in the recursive steps, so that the final analyses of their algorithm gives a worst-case complexity of $O(n \log n)$ time. Unfortunately, the overhead of their algorithm slows it down for small average sized problems, and as a further setback, it returns the wrong result if the original line is self-intersecting.

5.4 The Reumann-Witkam (RW) Algorithm

A Local Processing Routine technique, the intuitive Reumann-Witkam (RW) algorithm [102] needs only a single pass to identify all points in a given polyline that it deems critical. Starting at the critical endpoint p_1, the RW algorithm runs by recursively building a 'tolerance' corridor using the input width ϵ.

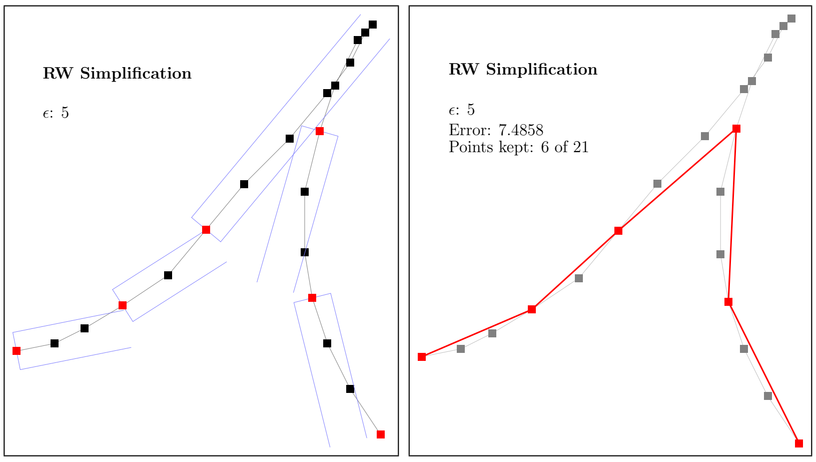


Figure 5.3: The Reumann-Witkam (RW) algorithm.

Let p_i be the latest critical point identified by the RW algorithm. Then the tolerance corridor is built on top of p_i, based on the *trend* of the line at p_i.

Typically, the *trend* at p_i refers to the gradient (or the approximation of the gradient) of the line at p_i. The corridor itself is parallel to this trend and it is pointed in the direction of p_{i+1}. All points after p_i lying inside the corridor are eliminated, until the traversal reaches the first point p_j where it no longer follows the trend and subsequently lies outside the corridor. The point p_j is then identified as the next critical point, and the RW algorithm repeats again from there. This process continues until all n points in the polyline are processed.

As evidenced from the example in Figure 5.3, there is a special case when the RW algorithm returns an incorrect simplification. Here, we see that an obvious critical point at the top of the figure has been left out – compare this to the resultant simplification of the DP algorithm in Figure 5.2. This error only happens whenever there exists a *sharp* turning point (or points) within the boundaries of the tolerance corridor. And because those sharp-edge points did not violate the corridor condition stated above, they are eliminated by the RW algorithm and deemed as non-critical.

5.5 Improving the Solutions of Reumann-Witkam's

The limitation of the RW simplification can be easily overcome, since we know that the main cause of the error is due to the algorithm missing all sharp-edge points in the polyline. This problem was not previously identified because the RW algorithm was deployed chiefly in the field of cartography to simplify contour lines. Hence, we assume here that (perhaps) contour lines had never taken the form of the example we gave in Figure 5.3.

However, after scouring the literature, we found an approach by Opheim [93] that extended the RW algorithm by imposing an additional constraint. In a way, this tries to reduce the stark error made by the RW simplification, but still, it may not produce a satisfactory result. On the other hand, we will show how, by combining the RW and the DP algorithms, we can achieve a better result in rectifying the RW simplification error.

5.5.1 The Opheim Algorithm

Opheim used exactly the same routine as the RW algorithm, except that his algorithm is constrained by an extra maximum distance check s_{max} [93]. After constructing the tolerance corridor at the latest critical point p_i, all points p_k after p_i are eliminated if and only if the following two conditions hold:

(i) The point p_k lies inside the tolerance corridor; and

(ii) The length of the chord $\overline{p_i p_k} \leq s_{max}$.

A critical point p_j is found if it violates the two conditions, and the Opheim algorithm repeats again from there. This process continues until all n points in the polyline are processed.

5.5.2 The RW-DP Algorithm

The combined RW-DP, like the Opheim algorithm, also uses the original RW routine to first *locally* identify the critical points. However, unlike Opheim's approach with the additional s_{max} constraint, we apply the *global* DP subroutine during the traversal on to every segment of the polyline identified between the previous critical point p_i and the newly identified critical point p_j.

In essence, we only need to run once more through all m points in the segment between p_i and p_j, before locating the elusive sharp-edge point, if one does exist. Thus, we maintain the overall runtime of the RW-DP algorithm to stay within $O(n)$ bounds, while improving the quality of simplification if such erroneous

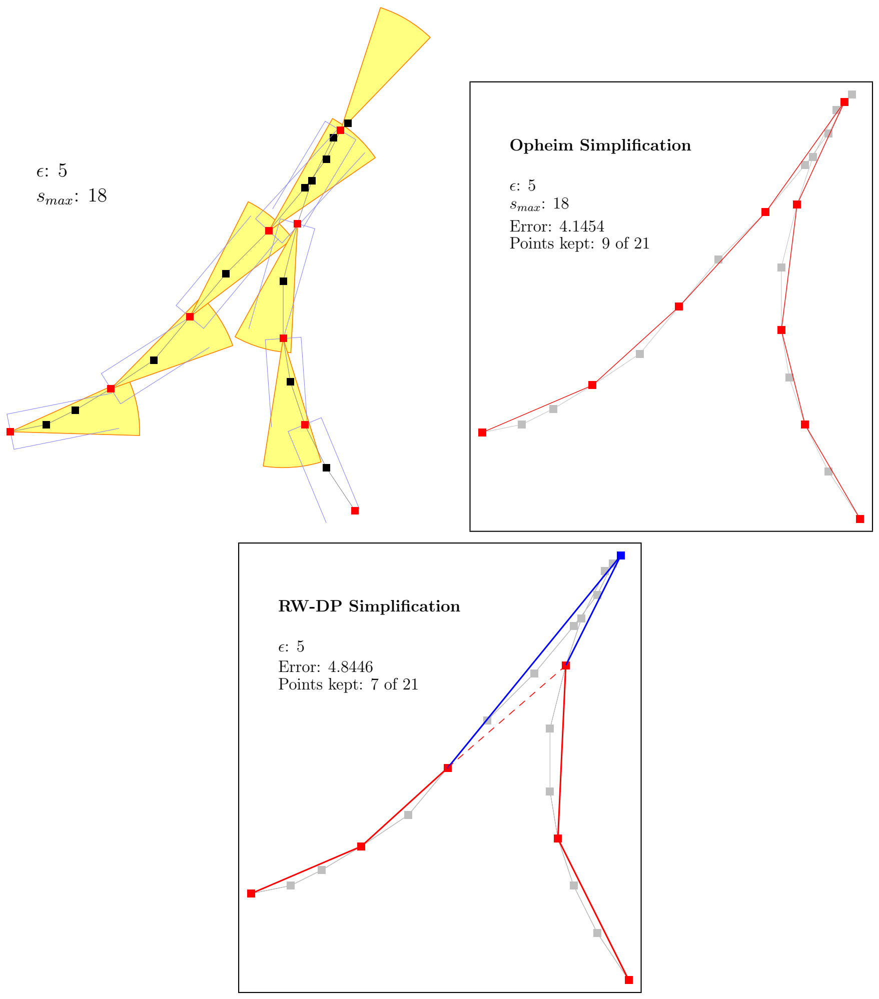


Figure 5.4: The Opheim and RW-DP algorithms. From top-left, clockwise: Tolerance corridors (blue rectangles) and additional s_{max} constraints (yellow cones); resultant Opheim simplification; resultant RW-DP simplification.

situations arise, as seen in the resultant Figure 5.4 compared to the original RW simplification in Figure 5.3.

5.6 The Optimal Polyline Simplification

Let $T = \{p_1, \dots, p_n\}$ be a polyline containing n points. Then, from the quality measure we proposed in section 5.2, it is possible to compute an optimal simplification of T for a given reduction rate. Suppose we want to have a simplified T keeping only m points, where $m < n$. Then the naïve approach to get the optimal simplification of T is to try out all possible point subsets of size m, determine the error for each such approximation, and then select one that minimizes this error. However, we have shown in our work with Richter [104] that we can do this in a more elegant manner.

The problem of finding an approximation with minimal error exhibits the characteristic of the "optimal substructure", one where an optimal solution comprises the optimal solutions to subproblems.

Lemma 5.1 *Let $E[i,j,k]$ be the minimal error of a simplification that approximates T between p_i and p_j, using exactly k intermediate points. Then for $k \geq 1$ and $1 < i < j < n$,*

$$E[i,j,k] = \min_{i<l<j-k} \{ \ E[i,l,0] + E[l,j,k-1] \ \}.$$

Proof. Let p_l be the first point after p_i that we decide to keep. Then the total error made by this approximation is the error made between p_i and p_l, plus the error made between p_l and p_j. An optimal approximation must minimize this sum, given that the two subsegments are also optimal. Thus, for all possible choices of p_l, we know the minimal errors $E[i,l,0]$ and $E[l,j,k-1]$, and we can determine the point p_l contributing to the minimal error and compute $E[i,j,k]$ using the equation above. ■

Let T' be the optimal simplified version of the original polyline T. If T' is to retain only m points from T, then the optimal approximation of T will have an error of $E[1,n,m-2]$ with T'. A simplification routine always retains the two original endpoints p_1 and p_n, and so we are only left to find $m-2$ intermediate points in T to keep. By Lemma 5.1, we need to compute $E[1,l,0]$ and $E[l,n,m-3]$ for all p_l between p_1 and p_{n-m-2}. The first of the two terms is a straight-forward computation, since there is only one way to approximate the line between p_1 and p_l using zero intermediate points. The second term, on the other hand, requires recursive computations, where in each recursion, the third parameter of the second term is reduced by one – until it reaches zero, in which case the final term can then be directly computed as above.

Clearly, from the recursion steps, one can see that the general singular terms $E[i,j,0]$ need to be calculated $O(n)$ times each. We can speed up this computation by avoiding to calculate the subproblems more than once using the scheme of Dynamic Programming [19], where we only need to store calculated values once in a table and look them up if we need them again. That is, we start by calculating $E[i,j,0]$ for all possible values of i and j directly, and store them in the top row of the table. From these, we compute the next row using the equation in Lemma 5.1; i.e. computing $E[i,j,1]$ for all possible values of i and j. We continue until all k rows of the table are filled. We can then extract from the completed table an approximation that yields the optimal error by tracing back which terms that were summed to construct it.

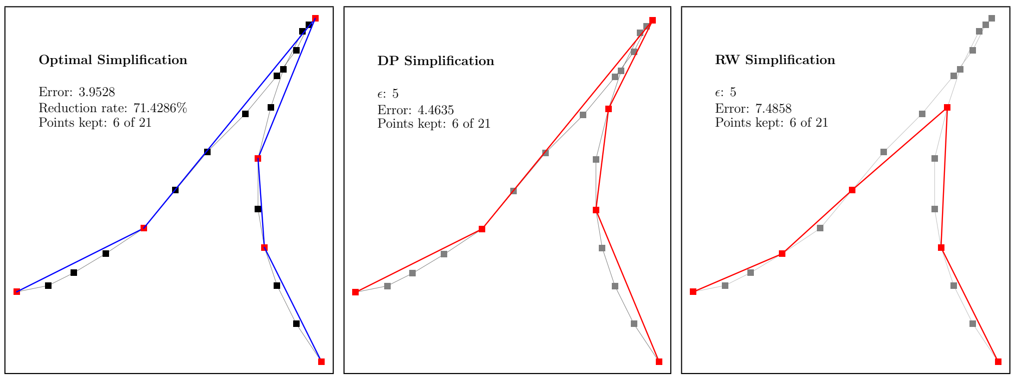


Figure 5.5: Optimal simplification for $m = 6$, compared to DP and RW simplifications for $\epsilon = 5$.

The space requirement for this dynamic calculation is cubic in the number of points n, because of the size of the table. The runtime is $O(n^4)$, since to compute one table entry, we first have to minimize all entries in the row above. While this is far better than the naïve runtime of $\frac{n!}{m!(n-m)!} = O(n!)$, this algorithm is still unacceptable for regular use. It does, however, become a very useful benchmark for evaluating the quality of non-optimal algorithms.

5.7 Simplified Polylines for Smoothing Routines

At the beginning of the chapter, we asked ourselves if we can use the simplified version of a polyline as a precursor to our smoothing routine. And up until this juncture, we have maintained a strict distinction between the *critical* and *crucial* terminologies, for a good reason. Putting this now in a clearer perspective, we want to know if the critical points identified by the simplification algorithm are those of the crucial points we described in chapter 3.

To better realise the significance of this, let us now fast forward a little bit into what we can expect from the future chapters, in terms of fulfilling our goals of smoothing a handwritten TRACE. Such an example is of the 'smoothed simplification' of the same TRACE that we have been observing throughout this chapter, as depicted in Figure 5.6, alongside the results of two other polyline simplications that we have discussed. We have purposely chosen the examples where there are exactly six points kept from the original 21 for the purpose of our comparative arguments.

The *crucial* points marked red in the smoothed simplification are the sharp-edge vertices characterized by Definition 3.15, and they coincide with the *critical* sharp-edge vertices identified by the Optimal and DP algorithms. The other three crucial blue points are either G^1 (tangent joints) or G^2 (inflections) continuity points, two of which match the critical points from the optimal simplification.

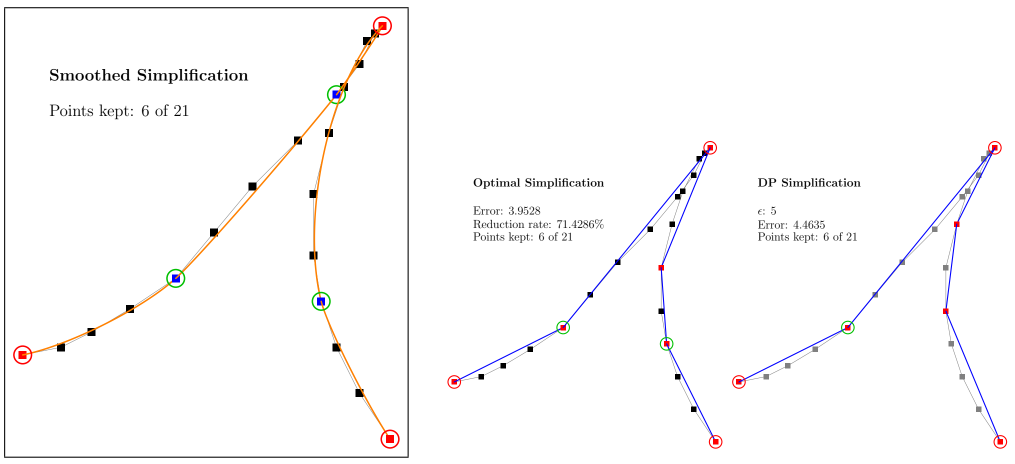


Figure 5.6: Smoothed simplification versus polyline simplifications.

These crucial points are the necessary proponents that approximate and define the (orange) curve segments accurately. As a direct consequence of this, the smoothed simplification correctly reveals a tiny loop at the top of the original Trace, which was an intended preconception of our example – an important feature which is not evident in the two resultant simplified polylines.

After reviewing the five polyline simplification algorithms, our conclusion is not to consider the simplification routines by RW or Opheim for our forthcoming deliberations, since they are both erroneous in their solutions, especially in identifying the critical sharp-edge vertices. We shall also dismiss the Optimal algorithm due to its impractical runtime. This leaves us in favour of the original DP and the extended RW-DP algorithms, whose simplified solutions are the closest to that of the Optimal algorithm's.

The DP algorithm was shown by White to exhibit a useful characteristic [134]; one which we can translate into Proposition 5.2 below to work within our domain specification.

Proposition 5.2 *Let T be a* Trace *as stated in Definition 3.2. Then, given a suitable predetermined corridor distance ϵ, the Douglas-Peucker algorithm retains all sharp-edge vertices in the simplified version of T.*

Proof. Consider the main principle of the DP algorithm: It finds the one intermediate point p_i between a set of ordered points p_A and p_B in T, that is farthest away from the baseline $\overline{p_A p_B}$. This principle, when applied recursively, returns all the critical points outside the tolerance corridor ϵ in T.

Now, a sharp-edge vertex is a sharp turning point on a curve's running trajectory, and by Definition 3.15, is also an extremum point. This makes the sharp-edge vertex p_j the farthest point away from an enclosing baseline $\overline{p_A p_B}$, and as long as p_j is outside the range of ϵ, p_j is retained as a critical point by the principle of the DP algorithm. ■

Lemma 5.3 *The Douglas-Peucker simplification omits inflection points.*

Proof. This follows directly from the proof in Proposition 5.2. An inflection point is a saddle point on the curve, and thus, is not an extremum point that fulfils the main principle of the DP algorithm. ■

CHAPTER 6

Rendering Curves with Elliptic Arcs

Representing TRACEs by second and third order parametric splines, as we have discussed in the previous chapter, comes under one class of curves; which we saw is *naturally iterable* by varying the parameter t from 0.0 to 1.0, to describe any portion of the curve's entity between two selected points. We shall now mention one other distinct class of curves, whose behaviour requires more effort to control, so that we can compare and contrast their streamlined performances. This class comprises the *conic* splines, formed by arcs of conics.

A conic is a classical mathematical curve derived by manipulating conical structures [12], and is the abstract terminology used to generally describe parabolas, hyperbolas, ellipses, and circles. Our deliberations shall concentrate plainly on ellipses and circles, the latter being a special case of the former. We will spend time in this chapter and the next to prove that a set of guided elliptic arcs can be ordered to conform to our earlier CompositeCurve definitions to symbolize a set of discretionary handwritten TRACEs. However, in the course of our unravelling this, we will also show that this technique of involving elliptic arcs is not without complications and intensive computations – even after simplifications. The result of which is the reason why we would favour the methods in the later chapters when compared to this one, whenever we desire *active-smoothing*.

6.1 The Ellipse – an Abstract Conic

Ellipses are essentially *conics* with very special properties, and we shall use the two terms interchangeably whenever we draw common generalities between them. We begin by confirming that the ellipse is indeed a form of a conic represented by the well-known implicit equation of degree two, containing six coefficients.

Implicit conic equation. ♠

$$f(x, y) = ax^2 + 2hxy + by^2 + 2ex + 2gy + c = 0 \tag{6.1}$$

Many authors have made it a common practice to arbitrarily choose one of the six coefficients, a, h, b, e, g, or c, and then using it as a divisor for $f(x, y)$ to eliminate one term, leaving only five degrees of freedom to deal with. Furthermore, we shall closely follow the naming conventions for these coefficients in Equation 6.1 as was introduced by Pavlidis [95], specifically for their meaningful cognition when we further manipulate them in our deeper ruminations.

Ellipse by focus and directrix. ♠

Definition 6.1 *Let $F \equiv (x_F, y_F)$ be a focus point of an ellipse, and let $D = Ax + By + C = 0$ be its directrix line. Then for any point P on the ellipse, the eccentricity is given by $\eta = \|\overline{PF}\| / \|\overline{PD}\| < 1$; and that the equation of the ellipse is*

$$\eta(Ax + By + C) = \pm\sqrt{(x - x_F)^2 + (y - y_F)^2}. \tag{6.2}$$

We interpret this formation of the ellipse as the locus of all points P whose distance from the focus point F is proportional to the parallel distance of the line $\overline{PD}$ with respect to the major-axis, from the directrix line D. We see in Figure 6.1 that D is always orthogonal to the major-axis of the resulting ellipse and cuts it at P_d. The same figure describes an ellipse whose eccentricity η is 0.75993, where given as examples, $\eta = \|\overline{FP_0}\| / \|\overline{P_0D_0}\| = \ldots = \|\overline{FP_3}\| / \|\overline{P_3D_3}\|$. Collectively, we call both these major and minor orthogonal axes, the *semi-major axes.* They meet at the centre of the ellipse at E, and on which the ellipse is mounted.

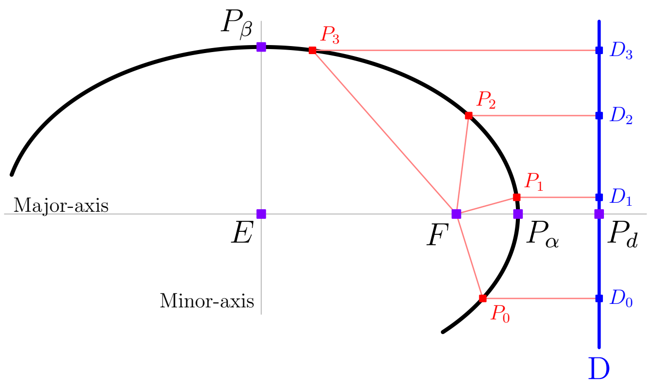


Figure 6.1: An elliptic arc with $\eta = 0.75993$ cutting the major- and minor-axes at points P_α and P_β, respectively. The directrix line D cuts the major-axis at P_d, and the semi-major axes meet at the centre of the ellipse at E.

Expanding and rearranging Equation 6.2, we get the same expression and thus affirming the ellipse with the general conic Equation 6.1.

$$\eta^2(Ax+By+C)^2-(x-x_F)^2+(y-y_F)^2=0$$

$$\begin{aligned}(\eta^2A^2-1)x^2+2(\eta^2AB)xy+(\eta^2B^2-1)y^2+2(x_F+\eta^2AC)x\\+2(y_F+\eta^2BC)y+(\eta^2C-x_F^2-y_F^2)=0\\f(x,y)=0\end{aligned}$$

Here, the six coefficients in $f(x,y)$ are expressed in terms of the ellipse's eccentricity, focus point, and the directrix line; where $a=\eta^2A^2-1$, $h=\eta^2AB$, $b=\eta^2B^2-1$, $e=x_F+\eta^2AC$, $g=y_F+\eta^2BC$, and $c=\eta^2C-x_F^2-y_F^2$. These coefficients categorise the type of conic that $f(x,y)$ represents. That is, the conic is

- an *ellipse* if $ab-h^2>0$;
- a *circle* if $a=b$ and $h^2=0$;
- a *parabola* if $ab=h^2$; or
- a *hyperbola* if $ab-h^2<0$.

The conic degenerates into a pair of straight lines if these coefficients we end up with reduce $f(x,y)$ into a product of two first-degree polynomials. In circumspect, for all the cases that $ab-h^2>0$, the ellipse is a non-degenerate conic and it is a planar compact non-singular algebraic curve of the second degree.

The general implicit conic Equation 6.1 well describes any given ellipse floating in the 2D plane. That is, the composite function $f(x,y)$ is the 'collective' description of three major properties – which we shall explain in more details in the upcoming sections. They are; (i) the angle at which an ellipse is tilted at with respect to the horizontal axis; (ii) the distance of the centre of the ellipse from the origin $(0,0)$; and (iii) the scaling factor induced on a "primitive" ellipse whose centre sits on the origin and whose semi-major axes are iso-oriented[1]. Such a primitive ellipse is well-known by the simplified Cartesian equation

Composite properties of $f(x,y)$. ♠

Primitive ellipse equation. ♠

$$\frac{x^2}{\alpha^2}+\frac{y^2}{\beta^2}=1 \tag{6.3}$$

where α and β are distances between the centre of the ellipse E and P_α, and between E and P_β, respectively, as shown in Figure 6.1.

[1]Iso-oriented lines are lines that are either parallel or perpendicular to the horizontal line on the 2D plane.

6.2 Problems with Rendering Composite Functions

Let us reiterate the problem here once more to understand why it is not expedient to sketch elliptic arcs directly from the composite function $f(x, y)$ when estimating arbitrary handwritten curves: Presented with a set of time-ordered points T, we first need to find the set of free-floating ellipses on the 2D plane, then extract only the necessary portions of their curvatures that best approximate the points in T, and finally reconstruct them into a seamless flow of well-aligned elliptic arcs resembling T. We have previously discussed that by Proposition 3.12, this problem is indeed decomposable; that is, the single CompositeCurvecomprising the original points in T, is made up of a set of finite second-order Curves. So the problem boils down to finding the one true ellipse whose appropriate portion of the arc well fits one of the second-order Curves. The broken-down problem of fitting ellipses to Curves is similar to several conic-fitting problems discussed by many authors [95, 97]. However, the main difference here is that we are more interested in looking for the quickest and most efficient way, and one without the need for error computations, if possible, to string together the second order elliptic arcs to form a symbolic representation of T.

Although generally descriptive, the implicit expression $f(x, y)$ in Equation 6.1 is not convenient for many applications to render the ellipse it describes. This becomes even more challenging if we want to reconstruct only a portion (or an arc) of the ellipse, while provided with very limited scattered points T as input criteria. Issues that come to mind include questions such as: What position does a point p in T lie on the ellipse with respect to the others? Do all the points in T relate to each other to give the same six coefficients in $f(x, y)$? What range of values for x and y are 'legal' for rendering the portion of the arc that we want from $f(x, y)$?

Rendering such objects is most efficient when there exists a way to re-express $f(x, y)$ into a singular function, where there is a direct one-to-one and non-composite relationship between the x and y variables. One way to do this is to keep one of the variables, say, x, constant and rearrange the expression explicitly to be in terms of y.

$$\begin{gathered} ax^2 + 2hxy + by^2 + 2ex + 2gy + c = 0 \\ by^2 + 2(hx + g)y + (ax^2 + 2ex + c) = 0 \end{gathered}$$

Solving this for y gives us an expression that we can relate to the constant x, so that y becomes a function of x.

$$\begin{aligned} y &= \frac{-2(hx + g) \pm \sqrt{4(hx + g)^2 - 4b(ax^2 + 2ex + c)}}{2b} \\ \Leftrightarrow f(x) &= \frac{1}{b}[-hx - g \pm \sqrt{(hx + g)^2 - b(ax^2 + 2ex + c)}] \end{aligned} \tag{6.4}$$

Clearly, the function $f(x)$ has real solutions if and only if its square-root term is also real. In other words, the square-root term dictates the 'legal' range of values of x for y to have solutions that we can render on the 2D plane. Thus,

$$\begin{aligned}(hx+g)^2 - b(ax^2+2ex+c) &\geq 0\\ (h^2-ab)x^2 + 2(gh-be)x + (g^2-bc) &\geq 0\\ (x+x_1)(x+x_2) &\geq 0\end{aligned}$$

where

$$x_{1,2} = \frac{1}{h^2-ab}[be-gh \pm \sqrt{(gh-be)^2-(h^2-ab)(g^2-bc)}]. \tag{6.5}$$

Therefore, assuming that x_1 is strictly less than x_2, the same ellipse $f(x,y)$ – now redefined as $f(x)$ – lies on the x-range of $x_1 \leq x \leq x_2$. We note here that for every value of x that does not reduce the square-root term in $f(x)$ to zero, there exist two real y values; such that both pairs of (x,y) refer to points on the upper and lower arcs of the ellipse. This also holds if we instead reverse the analogy to keep y constant and re-express $f(x,y)$ in terms of $f(y)$.

The technique of breaking up the ellipse equation and solving parts of it via each axis while keeping the other constant is analogous to Bresenham's ellipse drawing algorithm [12]. Although not as complex, compared to the Bresenham's algorithm, the simple mathematical technique explained above serves it purpose without having to split $f(x,y)$ into octants and without having to go through a series of *if-then-else* conditions to render the ellipse. These methods of rendering an ellipse does not, however, provide a useful insight to our main problem, given the limited input options.

Alternatively, rendering such complex representation of the objects can also be made efficient if we are able to reformulate the composite function in terms of a single parametric variable. For example, we can split the composite Cartesian Equation 6.3 of the 'primitive' ellipse into functions uniquely defined for each axis dimension, and unite them with a common parametric variable θ.

$$\left.\begin{aligned}x(\theta) &= \alpha\ \cos\theta\\ y(\theta) &= \beta\ \sin\theta\end{aligned}\right\} \qquad \text{for } 0 \leq \theta < 2\pi \tag{6.6}$$

In contrast, we cannot achieve such a one-to-one parametric relationship between the axis dimensions with the general conic Equation 6.1. However, by no straight-forward means, Pavlidis showed that it is possible to turn this general implicit form into *a set of iterative* parametric equations for each axis that, together, approximates the curve at predetermined points [95]. However, there are limitations in Pavlidis's elaborate and almost heuristical method, which requires the normalizations of the coefficients in $f(x,y)$, and which do not give reliable results if several special cases are not attended to; for example, when the value of $ab - h^2$ is near zero. Thus, we agree with Pavlidis that this implicit form is not convenient for proving properties about conics – let alone, about ellipses.

Unless of course, we can tone it down to a more manageable form.

So in the course of the next few sections, we will show how we can 'water-down' the general conic Equation 6.1 that describes any free-floating ellipse in 2D. We shall explain how to extract only the essential information for which we can use to effectively render the correct portion and the most suitable arc of the ellipse that best conform to a given segment of a CompositeCurve we want to symbolically represent.

6.3 Extracting Essential Ellipse Information

Our vantage approach is to effectively render any portion of a free-floating ellipse anywhere on the 2D plane given a known function $f(x, y)$, with of its all coefficients defined. In retrospect, we shall then consider information such as the centre, the distances with respect to the centre, the angles related to the centre, and the positioning of the semi-major axes of the ellipse as *essential.* We first show how we extract these essential information from the implicit Equation 6.1, while in the process, reduce $f(x, y)$ into a more accessible form involving matrices, and then later on discuss how to coordinate these pieces of information to achieve our goal.

Definition 6.2 *Let* $\mathbf{Q} = \left(\begin{smallmatrix} a & h \\ h & b \end{smallmatrix}\right)$, *where* a, b, *and* h *are the* x^2, y^2, *and* xy *coefficients of the implicit Equation 6.1, respectively. Then the matrix* $\mathbf{Q}$ *is called the* control matrix.

Let $\mathbf{x} = \left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right)$ and $\mathbf{g} = \left(\begin{smallmatrix} 2e \\ 2g \end{smallmatrix}\right)$ be two column vectors. Then, we can express the implicit Equation 6.1 in terms of the control matrix $\mathbf{Q}$.

$$\begin{aligned} f(x, y) &= ax^2 + hxy + hxy + by^2 + 2ex + 2gy + c = 0 \\ f(x, y) &= \begin{pmatrix} ax + hy & hx + by \end{pmatrix} \left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right) + \left(\begin{smallmatrix} 2e \\ 2g \end{smallmatrix}\right) \left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right) + c = 0 \\ f(x, y) &= \begin{pmatrix} x & y \end{pmatrix} \left(\begin{smallmatrix} a & h \\ h & b \end{smallmatrix}\right) \left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right) + \left(\begin{smallmatrix} 2e \\ 2g \end{smallmatrix}\right) \left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right) + c = 0 \end{aligned}$$

$$f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}} \mathbf{Q} \mathbf{x} + \mathbf{g}^{\mathrm{T}} \mathbf{x} + c = 0 \tag{6.7}$$

where $\mathbf{x}^{\mathrm{T}}$ and $\mathbf{g}^{\mathrm{T}}$ refer to the transpose of the vectors $\mathbf{x}$ and $\mathbf{g}$, respectively.

The general conic equation $f(x, y)$ gives us the partial derivatives

$$\begin{aligned} \nabla f(x, y) &= \left(\frac{\delta f(x, y)}{\delta x}, \frac{\delta f(x, y)}{\delta y}\right) \\ &= (2ax + 2hy + 2e,\ 2hx + 2by + 2g), \end{aligned} \tag{6.8}$$

from which the slope of the tangent at any point on the conic is dictated by

$$\frac{dy}{dx} = \frac{-\frac{\delta f(x,y)}{\delta x}}{\frac{\delta f(x,y)}{\delta y}} = \frac{-(ax + hy + e)}{hx + by + g}. \tag{6.9}$$

Any ellipse described by $f(x, y)$ has $\frac{dy}{dx} = 0$ at its top-most and bottom-most points P_i and P_j, since the geometrical tangents at these two specific points are parallel to the horizontal axis of the plane. Hence, we observe from Equation 6.9 that

$$ax + hy + e = 0$$

is the equation of the diameter of the ellipse joining P_i and P_j where it bisects the chords parallel to the axis of x. Subsequently, the geometrical tangents at the left-most and right-most points P_k and P_l on the curve are parallel to the vertical axis of the plane, such that $\frac{dx}{dy} = 0$. The same observation with Equation 6.9 gives us the other diameter of the ellipse

$$hx + by + g = 0$$

connecting P_k and P_l, so that it bisects the chord parallel to the axis of y. Thus, where these two diameter lines meet, we get the centre $E \equiv (x_E, y_E)$ of the free-floating ellipse $f(x, y)$, where

$$x_E = \frac{gh - eb}{ab - h^2} \quad \text{and} \quad y_E = \frac{eh - ag}{ab - h^2}. \tag{6.10}$$

Now, the inverse of the control matrix $\mathbf{Q}$ is not defined for parabolas, since $\det(\mathbf{Q}) = ab - h^2 = 0$. But for ellipses and hyperbolas, we can write the vector for the centre coordinate $\mathbf{x}_E = \binom{x_E}{y_E}$ in terms of $\mathbf{Q}$.

$$\begin{aligned}
\mathbf{x}_E = \begin{pmatrix} x_E \\ y_E \end{pmatrix} &= \frac{-1}{ab - h^2} \begin{pmatrix} eb - gh \\ -eh + ag \end{pmatrix} \\
\mathbf{x}_E &= \frac{-1}{2} \frac{1}{\det(\mathbf{Q})} \begin{pmatrix} b & -h \\ -h & a \end{pmatrix} \begin{pmatrix} 2e \\ 2g \end{pmatrix} \\
\mathbf{x}_E &= \frac{-1}{2} \mathbf{Q}^{-1} \mathbf{g}
\end{aligned} \tag{6.11}$$

Lemma 6.3 *The two semi-major axes of a free-floating ellipse $f(\mathbf{x})$ defined in Equation 6.7 are the two orthogonal eigenvectors $\mathbf{v}_\alpha$ and $\mathbf{v}_\beta$ of the control matrix $\mathbf{Q}$ passing through the centre of the ellipse at $E \equiv (x_E, y_E)$.*

Proof. As long as $f(\mathbf{x})$ does not describe a circle, we get two unique eigenvalues λ_α and λ_β of $\mathbf{Q}$ from its characteristic polynomial.

$$\begin{aligned} \det(\lambda\mathbf{I} - \mathbf{Q}) &= 0 \\ \det\begin{pmatrix} \lambda - a & -h \\ -h & \lambda - b \end{pmatrix} &= 0 \\ \lambda^2 - (a+b)\lambda + ab - h^2 &= 0 \end{aligned}$$

$$\lambda_{\alpha,\beta} = \frac{1}{2}\left(a + b \pm \sqrt{(a-b)^2 + 4h^2}\right) \qquad (6.12)$$

Let $\mathbf{D} = \left(\begin{smallmatrix} \lambda_\alpha & 0 \\ 0 & \lambda_\beta \end{smallmatrix}\right)$ be a diagonal matrix whose diagonal entries are the eigenvalues of $\mathbf{Q}$. Since the 2×2 control matrix $\mathbf{Q}$ is symmetric, that is $\mathbf{Q} = \mathbf{Q}^{\mathrm{T}}$, then Schur's theorem [12] states that there exists an orthogonal matrix $\mathbf{V}$ such that $\mathbf{V}^{-1}\mathbf{Q}\mathbf{V} = \mathbf{V}^{\mathrm{T}}\mathbf{Q}\mathbf{V} = \mathbf{D}$, where $\mathbf{V}^{\mathrm{T}} = \begin{pmatrix}\mathbf{v}_\alpha & \mathbf{v}_\beta\end{pmatrix}$ contains the two eigenvectors of $\mathbf{Q}$ that form an orthonormal set. Furthermore, from the corollary of Schur's theorem, we know that the corresponding eigenvalues λ_α and λ_β of $\mathbf{Q}$ are real numbers, satisfying both the unique conditions of $\mathbf{Q}\mathbf{v}_\alpha = \lambda_\alpha\mathbf{v}_\alpha$ and $\mathbf{Q}\mathbf{v}_\beta = \lambda_\beta\mathbf{v}_\beta$.

Later on in Lemma 6.7, we show that the angle ϕ at which the ellipse $f(\mathbf{x})$ is tilted at is governed by the elements in the same control matrix $\mathbf{Q}$. In effect, $\mathbf{Q}$ directly influence the orientations of the eigenvectors $\mathbf{v}_\alpha$ and $\mathbf{v}_\beta$ with respect to the orientation of the ellipse's semi-major axes. Thus, if we pass both eigenvectors $\mathbf{v}_\alpha$ and $\mathbf{v}_\beta$ through the centre of the ellipse at E, they will consequently lie exactly on the semi-major axes of the ellipse. ■

Extracting the eigenvectors of any matrix is a computationally expensive task. That is, working out the eigenvectors by hand (although more tedious) is simpler than computing them – since figuring out eigenvectors require some cognitive formulation of variables that are not directly translatable to manipulating variables in codes. Still, there exists many numerical techniques in the literature to deal with this task, that are inherently iterative, such as the Power Method (Vector Iteration), the Deflation Method, and the Householder's Method [12]. All of these return the approximation of only one eigenvector $\mathbf{v}$ of $\mathbf{Q}$ corresponding to one particular eigenvalue λ.

The iterative procedures begin with an initial seed value for the vector $\mathbf{v}$ and its corresponding λ, stopping only when certain tolerance criteria are met. Subsequent computations for other eigenvectors require the use of *another* method to achieve the desired approximates. For example, one may start with the Power Method to obtain the *largest* eigenvalue and its corresponding eigenvector. We note here that this method fails if there is no dominant eigenvalue for $\mathbf{Q}$ and its rate of convergence for closely spaced eigenvalues can be slow. Afterwhich, to find the next eigenvector corresponding to the *second largest* eigenvalue of $\mathbf{Q}$, one needs to apply the result from the Power Method to the Deflation method, and repeat the iterative procedure.

In our case, though, we can avoid estimating the eigenvectors though this arduous task of iterative computations and still obtain the exact values of $\mathbf{v}_\alpha$ and $\mathbf{v}_\beta$. We do this by directly making use of the *ratio-relationship* that λ_α and λ_β have with the control matrix $\mathbf{Q}$ in the solution space of $(\mathbf{Q} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}$.

Proposition 6.4 *Let λ_α and λ_β be the two unique eigenvalues of $\mathbf{Q}$, where $\mathbf{Q}$ is the control matrix describing an ellipse $f(\mathbf{x})$ defined in Equation 6.7. Then the two corresponding orthogonal eigenvectors $\mathbf{v}_\alpha$ and $\mathbf{v}_\beta$ of $\mathbf{Q}$ are directly related to the elements in $\mathbf{Q}$ with respect to λ_α and λ_β in a* ratio-relationship *as follows:*

$$\mathbf{v}_\alpha = \begin{pmatrix} a - \lambda_\beta \\ h \end{pmatrix}, \quad \mathbf{v}_\beta = \begin{pmatrix} h \\ b - \lambda_\alpha \end{pmatrix} \tag{6.13}$$

We know from Lemma 6.3 that non-trivial solutions exists for $(\mathbf{Q} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}$, for any control matrix $\mathbf{Q}$ in $f(\mathbf{x})$ describing ellipses. Let $\mathbf{v} = \binom{v_0}{v_1}$ be the general eigenvector of $\mathbf{Q}$ corresponding to an eigenvalue λ. Then from the eigenspace of $\mathbf{Q}$,

$$(\mathbf{Q} - \lambda\mathbf{I})\mathbf{v} = \begin{pmatrix} a - \lambda & h \\ h & b - \lambda \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

we get the expressions $(a-\lambda)v_0 + hv_1 = 0$ and $hv_0 + (b-\lambda)v_1 = 0$. The coefficients of v_0 and v_1 are equivalent when we substitute either of the eigenvalues λ_α or λ_β in place of λ. In other words, both expressions plainly turn into the form of $k_0v_0 + k_1v_1 = 0$, so that the ratio of the constants $k_0 : k_1$ is exactly the same as $(a - \lambda) : h$ in the first expression, and $h : (b - \lambda)$ in the second. And since a vector is simply the *ratio-relationship* between each axis dimension, we can appropriately express the eigenvectors $\mathbf{v}_\alpha$ and $\mathbf{v}_\beta$ as a relation of the elements in $\mathbf{Q}$ with respect to λ_α or λ_β as stated in Equation 6.13. This relation also holds true if we reverse the associations and take the other ratio-relationship, like so.

$$\mathbf{v}_\alpha = \begin{pmatrix} h \\ b - \lambda_\beta \end{pmatrix}, \quad \mathbf{v}_\beta = \begin{pmatrix} a - \lambda_\alpha \\ h \end{pmatrix}$$

Example 6.5 *Find the eigenvalues and the corresponding eigenvectors of an ellipse floating in 2D space defined by $f(\mathbf{x}) = 25x^2 + 10xy + 4y^2 - 250x - 16y + 541 = 0$.*

Solution. The control matrix in $f(\mathbf{x})$ is $\mathbf{Q} = \left(\begin{smallmatrix} 25 & 5 \\ 5 & 4 \end{smallmatrix}\right)$. We compute the eigenvalues of $\mathbf{Q}$ from Equation 6.12 to get $\lambda_\alpha = 26.129703$ and $\lambda_\beta = 2.870296$. Then using the Power Method and, subsequently, the Deflation method, we get the corresponding eigenvectors (denoting them as $\mathbf{v}'_\alpha$ and $\mathbf{v}'_\beta$, respectively), after 28 iterations, to be

$$\mathbf{v}'_\alpha = \begin{pmatrix} 0.975412 \\ 0.220385 \end{pmatrix} \quad \text{and} \quad \mathbf{v}'_\beta = \begin{pmatrix} 0.220385 \\ -0.975412 \end{pmatrix}.$$

Now we compare the results above to the eigenvectors we obtain by applying Equation 6.13 in Proposition 6.4 in the next two steps:

$$\mathbf{v}_\alpha = \begin{pmatrix} a - \lambda_\beta \\ h \end{pmatrix} = \begin{pmatrix} 22.129704 \\ 5 \end{pmatrix} \quad \text{and} \quad \mathbf{v}_\beta = \begin{pmatrix} h \\ b - \lambda_\alpha \end{pmatrix} = \begin{pmatrix} 5 \\ -22.129703 \end{pmatrix}.$$

Normalizing the eigenvectors $\mathbf{v}_\alpha$ and $\mathbf{v}_\beta$, we get

$$\widehat{\mathbf{v}}_\alpha = \frac{\mathbf{v}_\alpha}{\|\mathbf{v}_\alpha\|} = \begin{pmatrix} 0.975413 \\ 0.220385 \end{pmatrix} \quad \text{and} \quad \widehat{\mathbf{v}}_\beta = \frac{\mathbf{v}_\beta}{\|\mathbf{v}_\beta\|} = \begin{pmatrix} 0.220385 \\ -0.975413 \end{pmatrix}.$$

Thus, we get similar results as the approximations of $\mathbf{v}'_\alpha$ and $\mathbf{v}'_\beta$ from the iterative procedures. ■

The outcomes in Example 6.5 indicate that the vectors $\mathbf{v}_\alpha$ and $\mathbf{v}'_\alpha$ are scalar multiples of each other, and so are the vectors $\mathbf{v}_\beta$ and $\mathbf{v}'_\beta$. One ramification here is that the method in Proposition 6.4 offers a more efficient way of computing the eigenvectors of the control matrix $\mathbf{Q}$ in the ellipse $f(\mathbf{x})$.

Let us suppose that λ_α is strictly greater than λ_β. Then, from Lemma 6.3, it follows that the eigenvector $\mathbf{v}_\alpha$ through (x_c, y_c) is the major-axis of the ellipse $f(\mathbf{x})$. Also, from $\mathbf{v}_\alpha$, we know that the angle ϕ at which the ellipse is tilted at, with respect to the major axis, is

$$\phi = \arctan(\frac{h}{a - \lambda_\beta}) \tag{6.14}$$

While on the one hand, this requires the pre-computation of λ_β to obtain the angle ϕ, there exists, on the other hand, a more straight-forward solution, involving only the control matrix $\mathbf{Q}$ through Lemma 6.7, which we shall explain shortly.

At the moment, let us turn our attentions to reducing the implicit expression $f(\mathbf{x})$ in Equation 6.7, into a more manageable and familar form.

Lemma 6.6 *Let $E \equiv (x_E, y_E)$ be the centre of a free-floating ellipse $f(\mathbf{x})$ defined by Equation 6.7. Then translating the ellipse to the origin at $(0,0)$ with respect to E gives the expression*

$$f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} - \frac{1}{4}\mathbf{g}^{\mathrm{T}}\mathbf{Q}^{-1}\mathbf{g} + c = 0 \tag{6.15}$$

Proof. Let $(x, y) = (x + x_E, y + y_E)$ be the translated coordinates placing the centre of the ellipse at the origin. Then plugging $(x, y) = \mathbf{x}$ into the implicit Equation 6.7,

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + c = 0 \\ &= a(x + x_E)^2 + 2h(x + x_E)(y + y_E) + b(y + y_E)^2 \\ &\quad + 2e(x + x_E) + 2g(y + y_E) + c = 0. \end{aligned}$$

Expanding and rearranging the terms,

$$\begin{aligned} f(\mathbf{x}) =& ax^2 + 2hxy + by^2 \\ &+ (2ax_E + 2hy_E + 2e)x + (2hx_E + 2by_E + 2g)y \qquad (6.16) \\ &+ ax_E^2 + 2hx_Ey_E + by_E^2 + 2ex_E + 2gy_E + c = 0. \end{aligned}$$

The coefficient of the x-term cancels out to zero.

$$\begin{aligned} 2x\ [ax_E + hy_E + e] &= 2x\ [\frac{1}{ab - h^2}(a(gh - eb) + h(eh - ag)) + e] \\ &= 2x\ [\frac{1}{ab - h^2}(-eab + eh^2) + e] \\ &= 2x\ [\frac{-e(ab - h^2)}{ab - h^2} + e] = 0 \qquad \text{(I)} \end{aligned}$$

Similarly, the coefficient of the y-term also cancels out to zero.

$$\begin{aligned} 2y\ [hx_E + by_E + g] &= 2y\ [\frac{1}{ab - h^2}(h(gh - eb) + b(eh - ag)) + g] \\ &= 2y\ [\frac{-g(ab - h^2)}{ab - h^2} + g] = 0 \qquad \text{(II)} \end{aligned}$$

Simplifying the constant term, we get

$$\begin{aligned} &ax_E^2 + 2hx_Ey_E + by_E^2 + 2ex_E + 2gy_E + c \\ &= \frac{1}{(ab - h^2)^2}\ [a(gh - eb)^2 + 2h(gh - eb)(eh - ag) + b(eh - ag)^2 \\ &\quad + 2e(gh - eb)(ab - h^2) + 2g(eh - ag)(ab - h^2)] + c \\ &= \frac{1}{(ab - h^2)^2}\ [ag^2h^2 + be^2h^2 - 2egh^3 + ae^2b^2 + 2abegh - ba^2g^2] + c \\ &= \frac{1}{(ab - h^2)^2}\ [e(gh - eb)(ab - h^2) + g(eh - ag)(ab - h^2)] + c \\ &= \frac{1}{ab - h^2}\ [e(gh - eb) + g(eh - ag)] + c \qquad \text{(III)} \\ &= ex_E + gy_E + c \end{aligned}$$

Putting (I), (II), and (III) back together into (6.16), we obtain an independent expression without the terms x_E and y_E.

$$
\begin{aligned}
f(\mathbf{x}) &= ax^2 + 2hxy + by^2 + \frac{e(gh-eb)+g(eh-ag)}{ab-h^2} + c = 0 \\
f(\mathbf{x}) &= \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} - \frac{1}{4(ab-h^2)}\begin{pmatrix} 2eb-2gh & -2eh+2ag \end{pmatrix}\begin{pmatrix} 2e \\ 2g \end{pmatrix} + c = 0 \\
f(\mathbf{x}) &= \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} - \frac{1}{4}\begin{pmatrix} 2e & 2g \end{pmatrix}\frac{1}{ab-h^2}\begin{pmatrix} b & -h \\ -h & a \end{pmatrix}\begin{pmatrix} 2e \\ 2g \end{pmatrix} + c = 0 \\
f(\mathbf{x}) &= \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} - \frac{1}{4}\mathbf{g}^T\mathbf{Q}^{-1}\mathbf{g} + c = 0
\end{aligned}
$$

■

We see here that once we have the ellipse sitting exactly with its centre at the origin, both the single x and y terms in $f(\mathbf{x})$ disappear, and that the constant term simply involves $\mathbf{Q}^{-1}$. We can further reduce this expression exclusively contain the x^2 and y^2 terms – if we know how much to tilt the ellipse so that its major axis sits exactly on the horizontal axis on the 2D plane.

Lemma 6.7 *The ellipse $f(\mathbf{x})$ is tilted at an angle $\phi = \frac{1}{2}\arctan(\frac{2h}{a-b})$.*

Proof. We determine the angle at which the semi-major axis of the ellipse is tilted, with respect to the horizontal axis, by solving the coefficient of the xy-term of $f(\mathbf{x})$ either in Equation 6.7, or in the simpler expression in Equation 6.15.

Let $\mathbf{R}_\phi = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$ be the rotation matrix that transforms $f(\mathbf{x})$ by the angle ϕ so that the major axis of the ellipse becomes parallel to the horizontal axis of the plane. Also, let $\mathbf{x} = \binom{x}{y} = \mathbf{R}_\phi\mathbf{x}$, so that $x = x\cos\phi - y\sin\phi$ and $y = x\sin\phi + y\cos\phi$. Then placing $\mathbf{x}$ into Equation 6.15, we obtain

$$
\begin{aligned}
f(\mathbf{x}) = {} & a(x\cos\phi - y\sin\phi)^2 + 2h(x\cos\phi - y\sin\phi)(x\sin\phi + y\cos\phi) \\
& + b(x\sin\phi + y\cos\phi)^2 - c_1 = 0
\end{aligned}
$$

where $c_1 = 1/4\ \mathbf{g}^{\mathrm{T}}\mathbf{Q}^{-1}\mathbf{g} - c$. Rearranging the terms, we get the expression

$$
\begin{aligned}
f(\mathbf{x}) = {} & (a\cos^2\phi + 2h\cos\phi\sin\phi + b\sin^2\phi)\ x^2 \\
& + (-2a\cos\phi\sin\phi + 2h(\cos^2\phi - \sin^2\phi) + 2b\cos\phi\sin\phi)\ xy \\
& + (a\sin^2\phi - 2h\cos\phi\sin\phi + b\cos^2\phi)\ y^2 + c_1 = 0
\end{aligned}
\tag{6.17}
$$

Solving for the xy coefficient, we get

$$\begin{aligned}(b-a)2\cos\phi\sin\phi + 2h(\cos^2\phi - \sin^2\phi) &= 0\\ -(a-b)\sin 2\phi + 2h\cos 2\phi &= 0\\ \frac{\sin 2\phi}{\cos 2\phi} &= \frac{2h}{a-b}\\ 2\phi &= \arctan(\frac{2h}{a-b}).\end{aligned}$$

Hence, the ellipse is tilted at $\phi = \frac{1}{2}\arctan(\frac{2h}{a-b})$. ■

The angle ϕ here is equivalent to the one given in Equation 6.14, computed from the eigenvector $\mathbf{v}_\alpha$ of $\mathbf{Q}$.

Thus, after extracting ϕ and $\mathbf{x}_E$ from $f(\mathbf{x})$, we get, when we transform $f(\mathbf{x})$ to rotate by an angle of $-\phi$ and then displacing it by $-\mathbf{x}_c$, a primitive ellipse that cuts the horizontal axis at $(-\alpha, 0)$ and $(\alpha, 0)$, and the vertical axis at $(0, -\beta)$ and $(0, \beta)$. Consequently, the xy term in Equation 6.16 disappears at $\phi = \frac{1}{2}\arctan(\frac{2h}{a-b})$, and that the equation simply becomes $f(\mathbf{x}) = \frac{x^2}{\alpha^2} + \frac{y^2}{\beta^2} - 1 = 0$.

At this stage, we now get a chance to extract the semi-major lengths $\|\alpha\|$ and $\|\beta\|$ – both of which are directly related to the original free-floating ellipse.

Theorem 6.8 *Let $\|\alpha\|$ and $\|\beta\|$ be the lengths, from the centre of a free-floating ellipse in a 2D plane defined by $f(\mathbf{x})$ in Equation 6.7, to the points where the ellipse cuts its major and minor axes, respectively. Let ϕ be the angle at which the major-axis of the ellipse is tilted with respect to the horizontal-axis of the plane, and let $k = \frac{1}{4}\mathbf{g}^{\mathrm{T}}\mathbf{Q}^{-1}\mathbf{g} - c$ be the constant factor from Equation 6.15.Then, for $\mathbf{Q} = \begin{pmatrix} a & h \\ h & b \end{pmatrix}$, $\mathbf{x}_\phi = \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix}$, and $\mathbf{y}_\phi = \begin{pmatrix} \cos\phi \\ -\sin\phi \end{pmatrix}$,*

$$\|\alpha\| = \sqrt{\frac{k}{\mathbf{x}_\phi^{\mathrm{T}}\mathbf{Q}\mathbf{x}_\phi}}, \qquad \|\beta\| = \sqrt{\frac{k}{\mathbf{y}_\phi^{\mathrm{T}}\mathbf{Q}\mathbf{y}_\phi}}. \tag{6.18}$$

Proof. Following Lemma 6.6 and Lemma 6.7, the free-floating ellipse described by $f(\mathbf{x})$ would now have its centre at the origin and tilted so that its semi-major axes lie exactly on the horizontal and vertical axes of the 2D plane. As such, we can write the simplified equation of the ellipse as

$$f(\mathbf{x}) = a_\phi x^2 + b_\phi y^2 - k = 0 \qquad \text{by (6.17), from proof in Lemma 6.7}$$

where

$$\begin{aligned}a_\phi &= a\cos^2\phi + 2h\cos\phi\sin\phi + b\sin^2\phi = \mathbf{x}_\phi^{\mathrm{T}}\mathbf{Q}\mathbf{x}_\phi, \text{ and}\\ b_\phi &= a\sin^2\phi - 2h\cos\phi\sin\phi + b\cos^2\phi = \mathbf{y}_\phi^{\mathrm{T}}\mathbf{Q}\mathbf{y}_\phi.\end{aligned}$$

Simplifying the expression of $f(\mathbf{x})$, we get

$$\begin{aligned} f(\mathbf{x}) = \frac{a_\phi}{k}x^2 + \frac{b_\phi}{k}y^2 - 1 = 0 \\ \frac{x^2}{k/a_\phi} + \frac{y^2}{k/b_\phi} = 1. \end{aligned} \tag{IV}$$

Comparing (IV) with the primitive ellipse equation, we obtain the lengths

$$\begin{aligned} \alpha^2 &= k/a_\phi & \beta^2 &= k/b_\phi \\ \|\alpha\| &= \sqrt{\frac{k}{\mathbf{x}_\phi^T \mathbf{Q}\mathbf{x}_\phi}} & \|\beta\| &= \sqrt{\frac{k}{\mathbf{y}_\phi^T \mathbf{Q}\mathbf{y}_\phi}}. \end{aligned}$$

■

Corollary 6.9 *Let λ_α and λ_β be the eigenvalues of $\mathbf{Q}$, such that $\lambda_\alpha > \lambda_\beta$. Then*

$$\lambda_\alpha = \mathbf{y}_\phi^{\mathrm{T}}\mathbf{Q}\mathbf{y}_\phi \quad \textit{and} \quad \lambda_\beta = \mathbf{x}_\phi^{\mathrm{T}}\mathbf{Q}\mathbf{x}_\phi. \tag{6.19}$$

Proof. Based on the argument we gave in the proof of Lemma 6.3, an ellipse centred at the origin is simply a unit circle under a linear map associated with the symmetric control matrix $\mathbf{Q}$, such that $\mathbf{Q} = \mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}}$, where $\mathbf{D} = \left(\begin{smallmatrix} \lambda_\alpha & 0 \\ 0 & \lambda_\beta \end{smallmatrix}\right)$ is a diagonal matrix whose diagonal entries are the eigenvalues of $\mathbf{Q}$ and $\mathbf{V}$ is a real unitary matrix having as columns the corresponding eigenvectors of $\mathbf{Q}$. Then we known that the semi-major axes of the ellipse lie along the eigenvectors of $\mathbf{Q}$, and for $\lambda_\alpha > \lambda_\beta$, the norms of the major and minor axes are, respectively,

$$\|\alpha\| = \sqrt{\frac{k}{\lambda_\beta}} \quad \text{and} \quad \|\beta\| = \sqrt{\frac{k}{\lambda_\alpha}}.$$

Comparing the above expressions with Equation 6.18, we get $\lambda_\alpha = \mathbf{y}_\phi^{\mathrm{T}}\mathbf{Q}\mathbf{y}_\phi$ and $\lambda_\beta = \mathbf{x}_\phi^{\mathrm{T}}\mathbf{Q}\mathbf{x}_\phi$. ■

If the general conic equation $f(\mathbf{x})$ represents an ellipse, then we can perform an extraction procedure by applying the methods explained in this section to retrieve the four essential information about the ellipse: the angle of tilt ϕ, the centre coordinate E, and the two semi-major lengths $\|\alpha\|$ and $\|\beta\|$. These are enough to reliably render the perfect ellipse given just the implicit form of $f(\mathbf{x})$, and with a few more manipulations, the perfect arc between designated points on the curve.

6.4 Reconstructing a Perfect Elliptic Arc

In section 6.1, we gave the example of splitting the composite Cartesian equation $f(\mathbf{x}) = \frac{x^2}{\alpha^2} + \frac{y^2}{\beta^2} = 1$ into its two equivalent parametric forms on each axis dimension. They are both expressed as a function of a single variable θ; that is, we describe a point P_θ lying on the ellipse, based on its relative angular parametric position with respect to the centre of the ellipse, as

$$P_\theta \equiv (x(\theta),\ y(\theta)) = (\|\alpha\| \cos\theta,\ \|\beta\| \sin\theta),$$

for $0 \leq \theta < 2\pi$. Note that the centre of this ellipse is at the origin $(0, 0)$. From here on, we shall refer to the point the P_θ in its vector notation $\mathbf{P}_\theta$, for convenience to mean the same thing, and interpret $\mathbf{P}_\theta$ as a point lying on $f(\mathbf{x})$.

We shall apply the same principles as above to describe any point lying on an ellipse floating freely on the 2D plane given by $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + c = 0$; such that for $\theta = 0$ and $\theta = \pi$, we get two points on the ellipse lying exactly on its major axis that are $\|\alpha\|$ distances away from the centre at $E \equiv (x_E, y_E)$ on opposite sides. Also, for $\theta = \frac{\pi}{2}$ and $\theta = \frac{3\pi}{2}$, where two other points lie exactly on the ellipse's minor axis that are $\|\beta\|$ distances away from the centre on opposite sides. All other points lying on the curve $f(\mathbf{x})$ are similarly related to θ, where θ gives the relative angular parametric position with respect to the centre E.

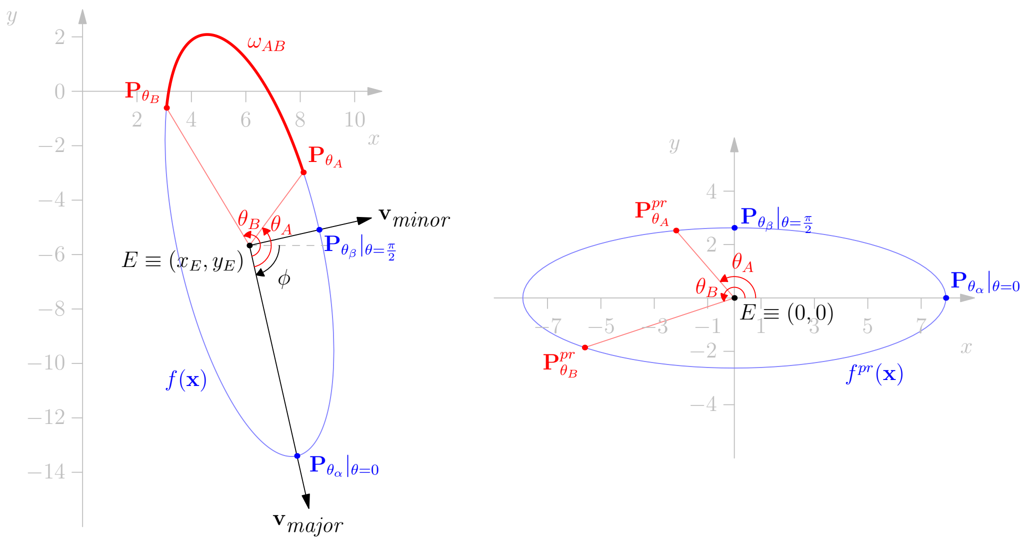
Definition 6.10 *The vector* $\mathbf{v}_{\text{major}}$ *is the eigenvector* $\mathbf{v}_\alpha$ *of* $\mathbf{Q}$ *whose origin is the centre* E *of a free-floating ellipse* $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + c = 0$*, intersecting* $f(\mathbf{x})$ *at* $\|\alpha\|$ *distance away from* E *at* $\mathbf{P}_{\theta_\alpha}$*, forming the* major-axis*.*

Definition 6.11 *The vector* $\mathbf{v}_{\text{minor}}$ *is the eigenvector* $\mathbf{v}_\beta$ *of* $\mathbf{Q}$ *whose origin is the centre* E *of a free-floating ellipse* $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + c = 0$*, intersecting* $f(\mathbf{x})$ *at* $\|\beta\|$ *distance away from* E *at* $\mathbf{P}_{\theta_\beta}$*, forming the* minor-axis*.*

As a consequence of being derived from the eigenvectors of $\mathbf{Q}$, both $\mathbf{v}_{\text{major}}$ and $\mathbf{v}_{\text{minor}}$ are orthogonal to each other and they intersect at the centre E of the ellipse $f(\mathbf{x})$ while forming its semi-major axes. We say that the free-floating ellipse $f(\mathbf{x})$ is 'mounted' on $\mathbf{v}_{\text{major}}$ and $\mathbf{v}_{\text{minor}}$.

Definition 6.12 *A point* $\mathbf{P}_\theta$ *on a free-floating ellipse* $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + c = 0$*, centred at* E *and mounted on* $\mathbf{v}_{\text{major}}$ *and* $\mathbf{v}_{\text{minor}}$*, is a parametric point related to the parametric variable* θ*, where* θ *is the angle made between the line* $\overline{\mathbf{P}_\theta E}$ *and* $\mathbf{v}_{\text{major}}$*.*

The above definitions suggest that the points where the ellipse intersects $\mathbf{v}_{\text{major}}$ and $\mathbf{v}_{\text{minor}}$ are at $P_{\theta=0}$ and $P_{\theta=\frac{\pi}{2}}$, respectively. This is graphically shown in Figure 6.2.



(a) Original plane (b) Primal plane

Figure 6.2: A free-floating ellipse $f(\mathbf{x})$, tilted at an angle ϕ with respect to the horizontal line and mounted on $\mathbf{v}_{\text{major}}$ and $\mathbf{v}_{\text{minor}}$ on the original plane (a), and its primitive dual $f^{\text{pr}}(\mathbf{x})$ on the primal plane (b).

Proposition 6.13 *A free-floating ellipse* $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + c = 0$ *has a* primitive dual $f^{\text{pr}}(\mathbf{x}) = \frac{x^2}{\alpha^2} + \frac{y^2}{\beta^2} - 1 = 0$*. For any point* $\mathbf{P}_\theta \in f(\mathbf{x})$*, there exists its* primitive dual point $\mathbf{P}^{\text{pr}}_\theta \in f^{\text{pr}}(\mathbf{x})$ *related by the transformation*

$$T_1(\mathbf{P}_\theta) = \mathbf{R}^{\mathrm{T}}_\phi \mathbf{P}_\theta - \mathbf{x}_c = \mathbf{P}^{\text{pr}}_\theta \tag{6.20}$$

for $\phi = \frac{1}{2}\arctan(\frac{2h}{a-b})$*, and where* $\mathbf{x}_E = \binom{x_E}{y_E}$ *is the vector representing the centre of the original ellipse* $f(\mathbf{x})$*, and where* $\mathbf{R}^{\mathrm{T}}_\phi$ *is the transpose of the rotation matrix introduced in the proof of Lemma 6.7.*

Proposition 6.14 *Following Proposition 6.13, let* $\mathbf{P}^{\text{pr}}_\theta = \binom{x^{\text{pr}}}{y^{\text{pr}}} = \binom{\|\alpha\|\cos\theta}{\|\beta\|\sin\theta}$*. Then, the angle* θ *related to* $\mathbf{P}^{\text{pr}}_\theta$ *is*

$$\theta = \arctan(\frac{\|\alpha\| y^{\text{pr}}}{\|\beta\| x^{\text{pr}}}). \tag{6.21}$$

Thus, any point $\mathbf{P}_\theta$ on the original ellipse $f(\mathbf{x})$ lying on the original plane has a one-to-one mapping to a unique point $\mathbf{P}^{\text{pr}}_\theta$ on a well-defined primitive ellipse $f^{\text{pr}}(\mathbf{x})$ lying on the *primal* plane. We write the reverse transformation from the primal plane back to the original plane as follows:

$$T_2(\mathbf{P}^{\text{pr}}_\theta) = \mathbf{R}_\phi \mathbf{P}^{\text{pr}}_\theta + \mathbf{x}_c = \mathbf{P}_\theta \tag{6.22}$$

Figure 6.2 illustrates this relationship between points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ on the original plane and the points $\mathbf{P}^{\mathrm{pr}}_{\theta_A}$ and $\mathbf{P}^{\mathrm{pr}}_{\theta_B}$ on the primal plane, respectively. Consequentially, we can further define an arc of the same ellipse in a similar way.

Definition 6.15 *An arc ω_{AB} on $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + c = 0$ is the set of points between $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$, for all $\mathbf{P}_\theta \in \omega_{\mathrm{AB}} \subseteq f(\mathbf{x})$ constrained by $\theta_A \leq \theta \leq \theta_B$.*

Through Proposition 6.13 and Proposition 6.14, we avoid Pavlidis's complicated and expensive method of incremental reconstruction of the elliptic arc [95] – without the need for any normalizations of the coefficients, especially when values of $ab - h^2$ is near zero; and without the need of having to treat the ellipse as its loose equivalent of the parabola. Since θ is the exact parametric position of the point $\mathbf{P}_\theta$ on $f(\mathbf{x})$, so does its primitive dual $\mathbf{P}^{\mathrm{pr}}_\theta$ on $f^{\mathrm{pr}}(\mathbf{x})$. Computing the arc ω_{AB} is the same as computing all the points on $f^{\mathrm{pr}}(\mathbf{x})$ between θ_A and θ_B, and then transforming them back onto the original plane using the reverse transformation in Equation 6.22. We state these straight-forward steps in the pseudo-codes in Algorithm 2.

Algorithm 2: GetEllipticArc ($f(\mathbf{x})$, $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$, *res*)

```
    Output: The arc $\omega_{A,B}$ of the ellipse $f(\mathbf{x})$ from $\mathbf{P}_{\theta_A}$ to $\mathbf{P}_{\theta_B}$ in steps res.
 1  Set $\|\alpha\|$, $\|\beta\| \leftarrow$ from $f(\mathbf{x})$ ;                 /* Theorem 6.8 */
 2  Set $\mathbf{P}^{\mathrm{pr}}_{\theta_A} \leftarrow T_1(\mathbf{P}_{\theta_A})$ ;                 /* Equation 6.20 */
 3  Set $\mathbf{P}^{\mathrm{pr}}_{\theta_B} \leftarrow T_1(\mathbf{P}_{\theta_B})$ ;                 /* Equation 6.20 */
 4  Set $\theta_A$, $\theta_B \leftarrow$ from $\mathbf{P}^{\mathrm{pr}}_{\theta_A}$, $\mathbf{P}^{\mathrm{pr}}_{\theta_B}$ ;          /* Equation 6.21 */
 5  Set $\phi = \frac{1}{2}\arctan(\frac{2h}{a-b})$ ;                   /* Lemma 6.7 */
 6  Set $\mathbf{x}_c = -1/2\ \mathbf{Q}^{-1}\mathbf{g}$ ;              /* from Equation 6.11 */
 7  Initialize $\omega_{A,B} \leftarrow$ empty;
 8  for $\theta \leftarrow \theta_A$ to $\theta_B$ step res do
 9      $\mathbf{P}^{\mathrm{pr}}_\theta \leftarrow \begin{pmatrix} \|\alpha\| & \cos\theta \\ \|\beta\| & \sin\theta \end{pmatrix}$;
10      $\mathbf{P}_\theta \leftarrow T_2(\mathbf{P}^{\mathrm{pr}}_\theta)$ ;                        /* Equation 6.22 */
11      Add $\omega_{A,B} \leftarrow \mathbf{P}_\theta$;
12  return $\omega_{A,B}$;
```

CHAPTER 7

Estimating with Elliptic Arcs

Let us now return to our problem at hand described in section 6.2, and continue with our given set of ordered points $TS \subseteq T$, decomposed and identified by Algorithm 1 through Proposition 3.12. This implies that the trajectory of the placement of the points in TS already resemble a second-order curve based on the constraints imposed in the identification process. So our task here is to find a set of well-placed ellipses, where each $f(\mathbf{x})$ has an arc ω_{AB} that best estimates the CompositeCurve KC representing TS.

We already know how to construct and render an elliptic arc efficiently between any two arbitrary points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ – both of which are related to the parametric variable θ. If we know the ellipse $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + c = 0$ on which $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ lie, we will also know the derivatives of $f(\mathbf{x})$ at $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$. Furthermore, if we fully know $f(\mathbf{x})$ by all its six coefficients, we can easily extract the four essential information about it – namely, the centre $E \equiv (x_E, y_E)$, the angle ϕ at which the major axis of $f(\mathbf{x})$ is tilted, and the two semi-major distances $\|\alpha\|$ and $\|\beta\|$ – to guide us with the drawing of ω_{AB}. Thus the missing pieces of the puzzle here are the six coefficients to $f(\mathbf{x})$, and we must derive them from our only source of input in TS.

Four essential information of $f(\mathbf{x})$. ♠

Contrary to the norm where authors divide $f(\mathbf{x})$ by its constant term c to eliminate one coefficient and concentrate on the five that are left, on the assumption that $c \neq 0$, we begin this chapter by arguing that we can always find six unique conditions for $f(\mathbf{x})$ based only on three different points lying on the ellipse. The first four conditions are fixed while the last two form the grounds we need to ensure a certain degree of freedom for the curve we are after. Once it becomes clear that the linear system of equations are solvable for all six conditions, we shall carry on to highlight the types of ellipses we can expect to get with our choices made from the last two conditions, based on triangle geometry. Following which, we give a thorough analysis from the results of these choices and then show that we can minimize on error checking between the computed

Six conditions to build $f(\mathbf{x})$. ♠

arc ω_{AB} and the set of original points in TS.

7.1 Deriving Six Conditions from TS

Let us construe ourselves to the TraceSegment TS, whose trajectory of internal points contains only a single extremum, so that it is possible to approximate TS evenly with a single second-order Curve K, as stated in Proposition 3.11. Appropriately in this chapter and the next, we shall refer to K as the elliptic arc ω_{AB}. Also, let us continue to denote a point $\mathbf{P}$ with the subscript θ as $\mathbf{P}_\theta$ to indicate that it lies on the ellipse $f(\mathbf{x})$. The elliptic arc ω_{AB} must lie within the bounds of the constraint-triangle ΔABC whose vertices A, B, and C are, respectively, the anchor-points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$, and $\mathbf{P}_C$, where the tangent lines at $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ meet. The first and the last of the ordered series of points in TS correspond to $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ – and appropriately, they must lie on the ellipse. The rest of the points in TS may or may not lie on ω_{AB}, as their main purpose is to serve as a trajectory guide for producing the arc we want.

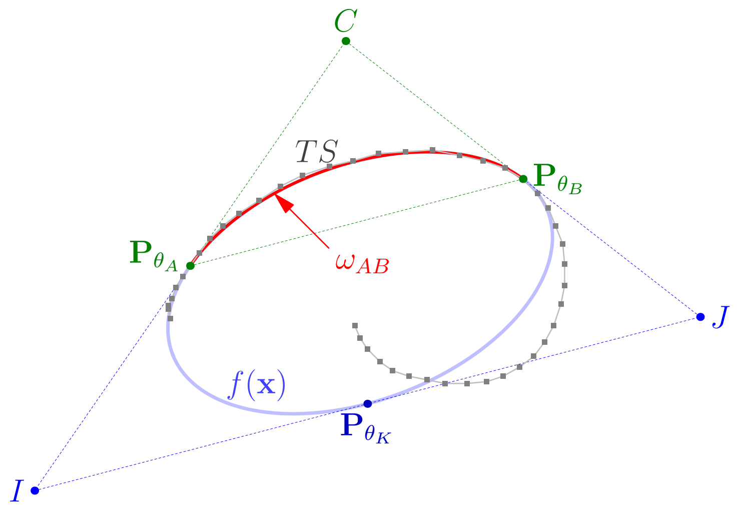


Figure 7.1: An inellipse $f(\mathbf{x})$ with respect to ΔABC and the extended ΔCIJ.

Based on just these points in TS, we derive six unique conditions needed to describe an ellipse, and then use them to solve a linear system of six unknowns to get all the coefficients for $f(\mathbf{x})$. The first four conditions determine the constraint-triangle ΔABC, based on the locations and first derivatives of the anchor-points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$.

Let $\mathbf{P}_{\theta_A} = \binom{x_A}{y_A}$ and $\mathbf{P}_{\theta_B} = \binom{x_B}{y_B}$, and let the directional derivatives at $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ be $\frac{dy_A}{dx_A}$ and $\frac{dy_B}{dx_B}$, respectively[1]. Then from the implicit Equation 6.1, we get the first two conditions.

[1]When discretization is concerned, we derive the derivatives of a point $p_i \in S'$ by means of

Condition 7.1 *The ellipse $f(\mathbf{x})$ passes through $\mathbf{P}_{\theta_A}$.*

$$f(\mathbf{P}_{\theta_A}) = x_A^2.a + 2x_Ay_A.h + y_A^2.b + 2x_A.e + 2y_A.g + c = 0$$

Condition 7.2 *The ellipse $f(\mathbf{x})$ passes through $\mathbf{P}_{\theta_B}$.*

$$f(\mathbf{P}_{\theta_B}) = x_B^2.a + 2x_By_B.h + y_B^2.b + 2x_B.e + 2y_B.g + c = 0$$

The third and fourth conditions are from the tangent Equation 6.9. With respect to $\mathbf{P}_C$, the tangents at $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ correspond to the slopes of the directed lines $\overline{\mathbf{P}_{\theta_A}\mathbf{P}_C}$ and $\overline{\mathbf{P}_C\mathbf{P}_{\theta_B}}$.

Condition 7.3 *The ellipse $f(\mathbf{x})$ is tangent to $\overline{\mathbf{P}_{\theta_A}\mathbf{P}_C}$ at $\mathbf{P}_{\theta_A}$.*

$$\frac{dy_A}{dx_A} = \frac{-x_A.a - y_A.h - e}{x_A.h + y_A.b + g}$$
$$\Leftrightarrow x_Adx_A.a + (x_Ady_A + y_Adx_A).h + y_Ady_A.b + dx_A.e + dy_A.g = 0$$

Condition 7.4 *The ellipse $f(\mathbf{x})$ is tangent to $\overline{\mathbf{P}_C\mathbf{P}_{\theta_B}}$ at $\mathbf{P}_{\theta_B}$.*

$$\frac{dy_B}{dx_B} = \frac{-x_B.a - y_B.h - e}{x_B.h + y_B.b + g}$$
$$\Leftrightarrow x_Bdx_B.a + (x_Bdy_B + y_Bdx_B).h + y_Bdy_B.b + dx_B.e + dy_B.g = 0$$

Two final conditions seal the uniqueness of the ellipse that we seek, and they are the crucial variants to the combined six conditions. Together, they give us an ellipse $f(\mathbf{x})$ whose arc ω_{AB} will either match or miss completely the set of criteria embodied in TS. We highlight in section 7.2 several of these varieties that we deem most suitably befitting our solutions to the main problem. There are several possibilities on how to do this from the vast literature, but mainly involving geometry where an ellipse is inscribed in a 'contact' triangle.

Definition 7.5 *The* contact-*triangle ΔCIJ contains an inscribed ellipse $f(\mathbf{x})$ that is tangent at three points on each of the three edges $\overline{CI}$, $\overline{CJ}$, and $\overline{IJ}$, derived from the contraint-triangle ΔABC in the following rationale:*

- *the common vertex C of ΔABC and ΔCIJ is the point $\mathbf{P}_C = \binom{x_C}{y_C}$;*
- *the vertex A of ΔABC is the point $\mathbf{P}_{\theta_A}$ lying on the edge $\overline{CI}$ of ΔCIJ, and is a tangent point of the ellipse;*
- *the vertex B of ΔABC is the point $\mathbf{P}_{\theta_B}$ lying on the edge $\overline{CJ}$ of ΔCIJ, and is a tangent point of the ellipse;*

numerical estimation, as was previously discussed in chapter 3

- *the points* $\mathbf{P}_I = \binom{x_I}{y_I}$ *and* $\mathbf{P}_J = \binom{x_J}{y_J}$ *are the vertices* I *and* J *of* ΔCIJ*;*
- *there exists a point* $\mathbf{P}_{\theta_K} = \binom{x_K}{y_K}$ *lying on the edge* $\overline{IJ}$ *of* ΔCIJ*, and is a tangent point of the ellipse;*
- *the chords* $\overline{AJ}$*,* $\overline{BI}$*, and* $\overline{C\mathbf{P}_{\theta_K}}$ *are concurrent at a point* G*, where* G *is known as the Brianchon point [128]; and*
- *the nested triangles* ΔABC *and* ΔCIJ *are not necessarily similar.*

Lemma 7.6 *Let* ΔCIJ *be a contact-triangle adhering to Definition 7.5. Then there exists an* inellipse $f(\mathbf{x})$ *unique only to* ΔCIJ *that is tangent at the three contact points* $\mathbf{P}_{\theta_A}$*,* $\mathbf{P}_{\theta_B}$*, and* $\mathbf{P}_{\theta_K}$*.*

Proof. Let us assume that there is always a unique inellipse $f(\mathbf{x})$ even if ΔCIJ is not a proper contact-triangle. In other words, the points $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$, and $\mathbf{P}_{\theta_K}$ are not properly positioned on the edges $\overline{CI}$, $\overline{CJ}$, and $\overline{IJ}$, so that the chords $\overline{AJ}$, $\overline{BI}$, and $\overline{C\mathbf{P}_{\theta_K}}$ are not concurrent at any single point. While it may still be possible for us to define an $f(\mathbf{x})$ that will pass through all three of the contact points, we can also sum up the following two deductions:

- that there is more than one $f(\mathbf{x})$ we can define that all three contact points will lie on; and
- that none of the $f(\mathbf{x})$'s above is an inellipse of ΔCIJ, since the Brianchon point G of ΔCIJ does not exist.

The first deduction suggests non-uniqueness, while the second suggests an impossibility. These contradict our earlier assumption, and thus it follows that, if and only if ΔCIJ is a proper contact-triangle, then there exists a unique inellipse to it. ■

So it follows from Lemma 7.6 that we only need to determine the proper contact-triangle ΔCIJ to be assured of a 'legal' ellipse. By legal, we mean that it is possible to construct an ellipse under the given conditions. Also, once we determined the proper ΔCIJ, we would have correctly identified the three contact points $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$, and $\mathbf{P}_{\theta_K}$. Since we already know the first two, building ΔCIJ from ΔABC will give us the third. And it is through this essential point $\mathbf{P}_{\theta_K}$ and the derivative at it that we draw the last two of the six conditions to complete our system of equations, and solve it for the six coefficients of $f(\mathbf{x})$.

Let $\frac{dy_K}{dx_K}$ be the tangent at $\mathbf{P}_{\theta_K}$. Since this tangent is the edge $\overline{IJ}$ of ΔCIJ, we can express it as $\frac{dy_K}{dx_K} = \frac{y_I - y_J}{x_I - x_J}$.

Condition 7.7 *The ellipse* $f(\mathbf{x})$ *passes through* $\mathbf{P}_{\theta_K}$*.*

$$f(\mathbf{P}_{\theta_K}) = x_K^2.a + 2x_K y_K.h + y_K^2.b + 2x_K.e + 2y_K.g + c = 0$$

Condition 7.8 *The ellipse $f(\mathbf{x})$ is tangent to $\overline{\mathbf{P}_I\mathbf{P}_J}$ at $\mathbf{P}_{\theta_K}$.*

$$\frac{dy_K}{dx_K} = \frac{-x_K.a - y_K.h - e}{x_K.h + y_K.b + g}$$
$$\Leftrightarrow x_K dx_K.a + (x_K dy_K + y_K dx_K).h + y_K dy_K.b + dx_K.e + dy_K.g = 0$$

The next section explains the various methods we can apply to get the points $\mathbf{P}_I$ and $\mathbf{P}_J$, based only on ΔABC, from which two points we use to compute $\mathbf{P}_{\theta_K}$ and its corresponding derivative.

7.2 Inconics and Inellipses

An *inconic* is a conic inscribed in a triangle, and it is tangent to all three edges of the triangle. Its dual is known as the *circumconic*, where there exists another conic that passes through all three vertices of the same triangle. Our approach to the current problem favours the inconics (as opposed to the circumconics), in which we want to run a curve inside the constraint-triangle ΔABC.

Any trilinear equation of the form

$$x^2\alpha^2 - 2xy\alpha\beta + y^2\beta^2 - 2xz\gamma\alpha - 2yz\gamma\beta + z^2\gamma^2 = 0, \tag{7.1}$$

where x, y, and z are functions of the side lengths i, j, and c, is an inconic, and that Weisstein claims that every inconic has such an equation [131]. That is, elaborating this specifically to fit our context, for any ΔCIJ, the side lengths are $i = \|CJ\|$, $j = \|CI\|$, and $c = \|IJ\|$, and it is common in the literature to describe the inconic in terms of its trilinear coordinates $\alpha : \beta : \gamma$.

Furthermore, the lines connecting the vertices of ΔCIJ to the corresponding 'contact' points of the inconic are concurrent in the Brianchon point of the triangle [128]. Revisiting our example from the previous section, the Brianchon point G of ΔCIJ is the intersection of the chords $\overline{AJ}$, $\overline{BI}$, and $\overline{C\mathbf{P}_{\theta_K}}$. In practice, the inconic parameters are stated simply in terms of the trilinear coordinates of the Brianchon point as

$$x : y : z = \frac{1}{\alpha} : \frac{1}{\beta} : \frac{1}{\gamma}. \tag{7.2}$$

Also, the centre of an inconic with parameters $x : y : z$ is the Kimberling point

$$cy + bz : az + cx : bx + ay. \tag{7.3}$$

Naturally, we shall only concern ourselves with a specific type of inconics – the *inellipses*. And since we do not have the 'given' ΔCIJ, we will need to seek enough properties to build it from the constraint-triangle ΔABC, with the help of *known* inellipses to guide us.

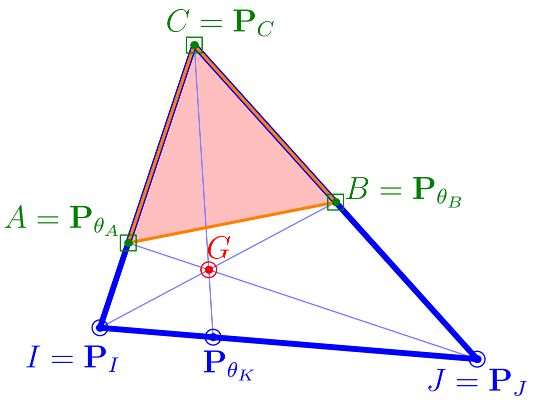


Figure 7.2: Extending the constraint-triangle ΔABC to get the contact-triangle ΔCIJ, so that the inellipse $f(\mathbf{x})$ is tangent at $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$, and $\mathbf{P}_{\theta_K}$.

There are several well-known classical inellipses (or inconics, depending on the values of $\alpha : \beta : \gamma$) in which many authors study their behavioural patterns and the consequences of their constructions. Out of which we picked four to study in this thesis. All these inellipses are conceived from the unique attributes of the contact-triangle ΔCIJ, in one way or another. This in turn leads to our observation that there are two main categories in which we can construct ΔCIJ based primarily on ΔABC: (i) strictly by the measurement of lengths, and (ii) strictly by the measurement of angles.

The first category includes the Steiner and the Mandart inellipses, while the second comprises the Orthic inconic and the Brocard inellipse; all of which are thoroughly described in Weisstein's report [130]. These two categories result in a fixed ΔCIJ that is oblivious to any other external conditions that may shape the arc ω_{AB} that we want from the inellipse $f(\mathbf{x})$, except for the fact that $f(\mathbf{x})$ will contain the points $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$, and $\mathbf{P}_{\theta_K}$, and is tangent at all three of them. Later on in section 7.3, as an alternative to countermand the fixed-ness of ΔCIJ, we shall introduce a third category. It measures neither the lengths nor the angles related to ΔABC, but rather takes into account an external factor based on the information we can gather about TS. This allows us one degree of freedom when constructing ΔCIJ from ΔABC.

For now though, we shall take a closer look at the classical inellipses to sustain our quest of finding the proper ΔCIJ given only ΔABC.

7.2.1 Steiner Inellipse

The Steiner inellipse of ΔCIJ is an inellipse whose trilinear equation is

$$i^2\alpha^2 - 2ij\alpha\beta + j^2\beta^2 - 2ic\gamma\alpha - 2jc\gamma\beta + c^2\gamma^2 = 0, \tag{7.4}$$

with inconic parameters

$$x : y : z = i : j : c. \tag{7.5}$$

Chakerian [15] calls this, appropriately, as the midpoint ellipse, since the inellipse $f(\mathbf{x})$ is tangent to the contact-triangle ΔCIJ at the midpoints of all three edges $\|CI\|$, $\|CJ\|$, and $\|IJ\|$.

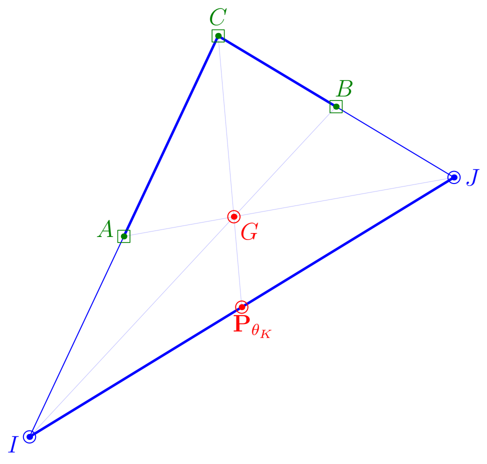


Figure 7.3: The Steiner triangle ΔCIJ from ΔABC.

Definition 7.9 *Following Definition 7.5, the vertices I and J of the Steiner ΔCIJ with respect to ΔABC are, respectively,*

- *the point $\mathbf{P}_I$ that is equidistant to $\|\mathbf{P}_{\theta_A}\mathbf{P}_C\|$ from $\mathbf{P}_{\theta_A}$ on the opposite side of $\mathbf{P}_C$, such that*

$$\mathbf{P}_I = \begin{pmatrix} x_I \\ y_I \end{pmatrix} = \begin{pmatrix} x_A - dx_A \\ y_A - dy_A \end{pmatrix}; \textit{ and}$$

- *the point $\mathbf{P}_J$ that is equidistant to $\|\mathbf{P}_C\mathbf{P}_{\theta_B}\|$ from $\mathbf{P}_{\theta_B}$ on the opposite side of $\mathbf{P}_C$, such that*

$$\mathbf{P}_J = \begin{pmatrix} x_J \\ y_J \end{pmatrix} = \begin{pmatrix} x_B + dx_B \\ y_B + dy_B \end{pmatrix}.$$

The point $\mathbf{P}_{\theta_K}$ is the midpoint of $\overline{IJ}$, such that

$$\mathbf{P}_{\theta_K} = \begin{pmatrix} x_K \\ y_K \end{pmatrix} = \begin{pmatrix} (x_I + x_J)/2 \\ (y_I + y_J)/2 \end{pmatrix}.$$

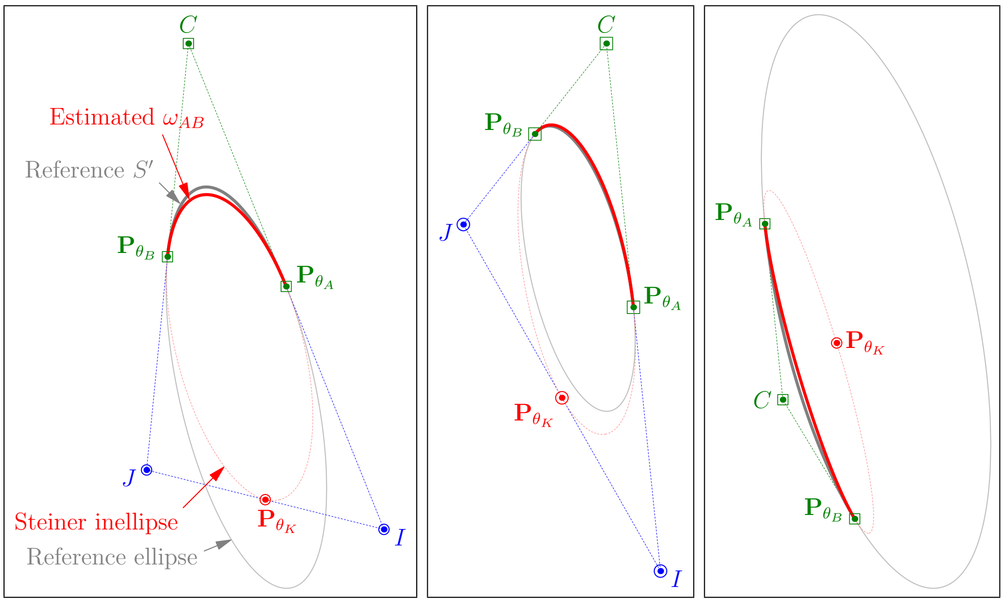


Figure 7.4: Estimating ω_{AB} with the Steiner inellipse.

The centre of the Steiner inellipse is the Brianchon point G, and it has been argued[2] that the inellipse $f(\mathbf{x})$ has a maximum area of any inellipse, such that it is exactly $\frac{\pi}{3\sqrt{3}}$ times the area of the Steiner triangle ΔCIJ.

We illustrate three resultant arcs ω_{AB} from the computed Steiner inellipse in Figure 7.4, based on selected points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ from a known reference ellipse that we used in our earlier examples. We refer to the original arc of the reference ellipse in between $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ as TS

The Steiner inellipse is one of the most stable inellipses that can be constructed given only the constraint-triangle ΔABC as starting parameters. This is because we can always find, and rather easily too (in comparison to the next three methods), the Steiner triangle. The arc ω_{AB} it produces is firm, and this is the main reason why we shall use the Steiner inellipse as a basis of our adaptive method, where we will later show how we can tweak the methodologies within to adapt it to the external conditions in TS.

7.2.2 Orthic Inconic

The Orthic inconic of ΔCIJ is an inconic whose trilinear equation is

$$i^2S_I^2\alpha^2 - 2ijS_IS_J\alpha\beta + j^2S_J^2\beta^2 - 2icS_IS_C\gamma\alpha - 2jcS_JS_C\gamma\beta + c^2S_C^2\gamma^2 = 0, \quad (7.6)$$

[2]There has been a recent claim that a Simmons inellipse has a bigger area than the Steiner inellipse [130].

where S_I, S_J, and S_C are the Conway triangle notations [129], such that

$$\begin{aligned} S_I &= \frac{1}{2}(-i^2 + j^2 + c^2) = ij\cos I, \\ S_J &= \frac{1}{2}(i^2 - j^2 + c^2) = ic\cos J, \text{ and} \\ S_C &= \frac{1}{2}(i^2 + j^2 - c^2) = jc\cos C. \end{aligned}$$

The inconic is an inellipse for acute triangles, and a hyperbola for obtuse triangles. Here, we shall only consider the cases where $\angle ICJ < \frac{\pi}{2}$ radians. When the Orthic inconic is an inellipse, its Brianchon point G is the orthocentre of ΔCIJ

$$x : y : z = \sec I : \sec J : \sec C, \tag{7.7}$$

and the inellipse has an area of

$$A = \frac{2\pi\sqrt{2}ijc\sqrt{\cos I \cos J \cos C}}{(i^2 + j^2 + c^2)^{\frac{3}{2}}}\Delta, \tag{7.8}$$

where Δ is the area of the Orthic ΔCIJ. A line connecting a vertex of the Orthic ΔCIJ to its corresponding contact point is always perpendicular to the edge that it intersects.

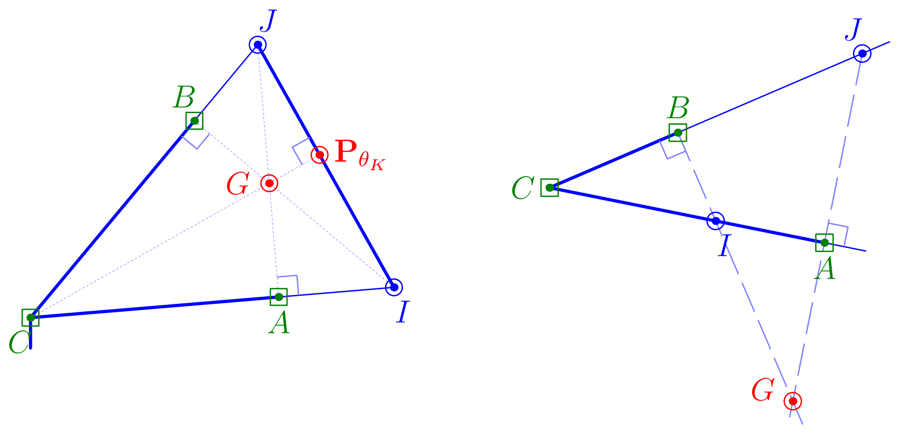
Definition 7.10 *Following Definition 7.5, the vertices I and J of the Orthic ΔCIJ with respect to ΔABC are, respectively,*

- *the point $\mathbf{P}_I$ that is the intersection of the tangent line through $\mathbf{P}_{\theta_A}$ and the co-tangent line through $\mathbf{P}_{\theta_B}$, in such a way that the point $\mathbf{P}_{\theta_A}$ is between $\mathbf{P}_C$ and $\mathbf{P}_I$ on the edge $\overline{CI}$; and*
- *the point $\mathbf{P}_J$ that is the intersection of the tangent line through $\mathbf{P}_{\theta_B}$ and the co-tangent line through $\mathbf{P}_{\theta_A}$, in such a way that the point $\mathbf{P}_{\theta_B}$ is between $\mathbf{P}_C$ and $\mathbf{P}_J$ on the edge $\overline{CJ}$.*

The point $\mathbf{P}_{\theta_K}$ is the intersection of the edge $\overline{IJ}$ of ΔCIJ and the line through the point $\mathbf{P}_C$ and the Brianchon point G.

We compute the Cartesian coordinates for $\mathbf{P}_I = \binom{x_I}{y_I}$ based on Definition 7.10, where the tangent and co-tangent lines through $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$, respectively, are

$$y_I - y_A = \frac{dy_A}{dx_A}(x_I - x_A) \quad \text{and} \quad y_I - y_B = \frac{-dx_B}{dy_B}(x_I - x_B).$$



(a) Orthic triangle ΔCIJ (b) Improper Orthic triangle

Figure 7.5: Proper vs. improper Orthic triangles

Solving simultaneously for x_I and y_I, we get

$$x_I = \frac{dx_A[dy_B(y_B - y_A) + dx_B x_B] + dy_A dy_B x_A}{dx_A dx_B + dy_A dy_B} \quad \text{and} \tag{7.9}$$

$$y_I = \begin{cases} y_A + \frac{dy_A}{dx_A}(x_I - x_A) & \text{if } dx_A \neq 0, \\ y_B - \frac{dx_B}{dy_B}(x_I - x_B) & \text{otherwise.} \end{cases} \tag{7.10}$$

Similarly, we do the same for $\mathbf{P}_J = \binom{x_J}{y_J}$, where the tangent and co-tangent lines through $\mathbf{P}_{\theta_B}$ and $\mathbf{P}_{\theta_A}$, respectively, are

$$y_J - y_B = \frac{dy_B}{dx_B}(x_J - x_B) \quad \text{and} \quad y_J - y_A = \frac{-dx_A}{dy_A}(x_J - x_A).$$

Since we know that $\angle ICJ$ is acute, then there are always solutions for $\mathbf{P}_I$ and $\mathbf{P}_J$. However, the Brianchon point G will not lie inside ΔCIJ if either $\mathbf{P}_I$ or $\mathbf{P}_J$ lies between $\mathbf{P}_C$ and $\mathbf{P}_{\theta_A}$, or $\mathbf{P}_C$ and $\mathbf{P}_{\theta_B}$, respectively. If such is the case, then the point $\mathbf{P}_{\theta_K}$ will lie outside ΔCIJ, as illustrated by the improper Orthic triangle in Figure 7.5(b). Subsequently, there will be no solution for the inellipse $f(\mathbf{x})$ that satisfies the six conditions listed in section 7.1. On the other hand, if the points $\mathbf{P}_I$ and $\mathbf{P}_J$ suggest a proper Orthic triangle, then we need only two more steps to obtain $\mathbf{P}_{\theta_K}$. The first is to determine the Brianchon point $G \equiv (x_G, y_G)$, which is the intersection of the chords $\overline{AJ}$ and $\overline{BI}$. Then $\mathbf{P}_{\theta_K}$ is the intersection of the edge $\overline{IJ}$ with its co-tangent line through G, which we get by simultaneous solving the equations

$$y_K - y_I = \frac{dy_K}{dx_K}(x_K - x_I) \quad \text{and} \quad y_K - y_G = \frac{-dx_K}{dy_K}(x_K - x_G).$$

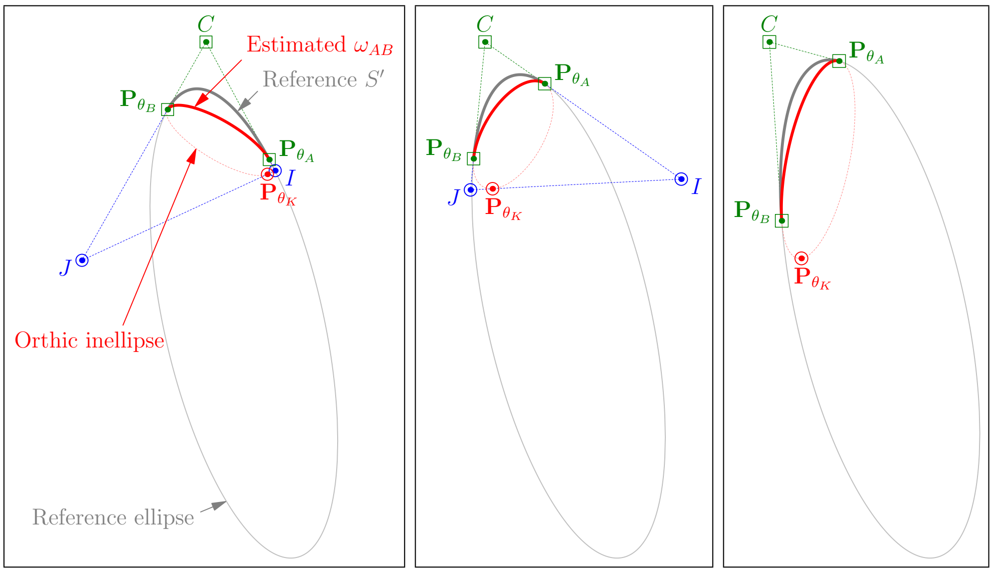


Figure 7.6: Estimating ω_{AB} with the Orthic inellipse.

Figure 7.6 depicts three examples of the resultant arcs ω_{AB} from the computed Orthic inellipses. Again, we based the selected points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ from the same reference ellipse from the previous examples. Here, we purposely chose the points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ that led to the construction of proper Orthic triangles ΔCIJ. We also note that as restrictive as the formation of the Orthic inellipse is, the estimated arc it offers is not as accurate as we may want it to be. This later becomes more obvious when we apply the Orthic inellipse to the disorderly set of points in TS that are sampled from a raw handwritten trace.

7.2.3 Brocard Inellipse

The Brocard inellipse of ΔCIJ is an inellipse with the trilinear equation

$$\frac{1}{i^2}\alpha^2 - \frac{2}{ij}\alpha\beta + \frac{1}{j^2}\beta^2 - \frac{2\gamma}{ic}\alpha - \frac{2\gamma}{jc}\beta + \frac{\gamma^2}{c^2} = 0, \tag{7.11}$$

and whose inconic parameters are

$$x : y : z = \frac{1}{i} : \frac{1}{j} : \frac{1}{c}. \tag{7.12}$$

The Brocard triangle ΔCIJ is dictated by a unique angle φ, and is one that is determined by an inherently geometrical means; see Figure 7.7 for an example. Where the chords $\overline{JA}$, $\overline{IB}$, and $\overline{C\mathbf{P}_{\theta_K}}$ concurrently meet at G, they meet at φ angle away from the edges $\overline{JC}$, $\overline{IJ}$, and $\overline{CI}$ respectively. There exists only

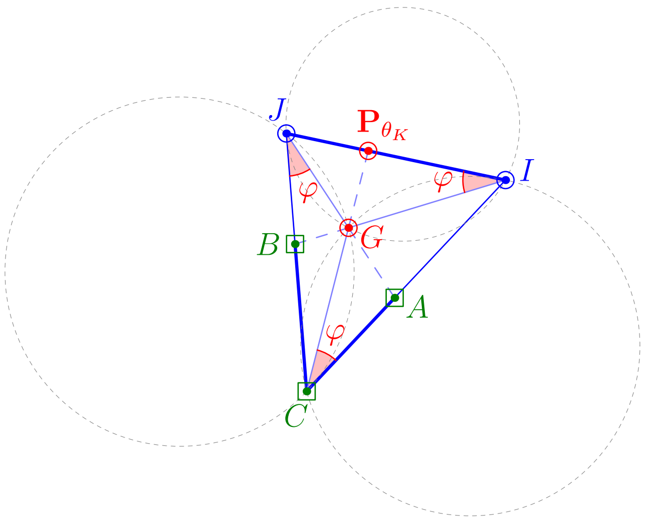


Figure 7.7: The Brocard triangle ΔCIJ.

one such angle φ for any ΔCIJ, and when this happens the Brianchon point G is called the Brocard point. In other words, the Brocard point satisfies the condition that

$$\angle GCI = \angle GIJ = \angle GJC = \varphi. \tag{7.13}$$

The angle φ is the Brocard angle of ΔCIJ and is known by the relationship

$$\cot\varphi = \cot C + \cot I + \cot J, \tag{7.14}$$

proven through the analysis of the geometric construction of the Brocard point G, when given a fixed ΔCIJ. We sum up its construction in the following statements using Figure 7.7 as reference:

- Construct a circle through points C and I, and tangent to the edge $\overline{IJ}$.
- Construct a second circle through the points I and J, and tangent to the edge $\overline{CJ}$.
- Construct a third circle through the points J and C, and tangent to the edge $\overline{CI}$.
- The Brocard point G is the unique intersection of the above three circles.

The centre of the first circle is the point where the perpendicular bisector of $\overline{CI}$ intersects the line through the point I that is perpendicular to $\overline{IJ}$. This is symmetrically reflected in the second and third circles. This resultant point G

is the *first* Brocard point of ΔCIJ. There exists a second Brocard point for the same Brocard angle φ satisfying the other condition

$$\angle GCJ = \angle GJI = \angle GIC = \varphi.$$

Just as intricate is the building of the Brocard ΔCIJ from the constraint-triangle ΔABC, since, to begin with, we know neither the positions of the vertices I and J nor the Brocard angle φ. In fact, we cannot find a straight-forward mathematical formula or a set of solvable equations to build ΔCIJ from ΔABC. We need to rely on a primarily geometrical means, as well as using a recursive technique to get to G, through φ. However, let us first lay the definition necessary for the Brocard triangle.

Definition 7.11 *Following Definition 7.5, the vertices J and I of the Brocard ΔCIJ with respect to ΔABC and the (first) Brocard point G are, respectively,*

- *the point $\mathbf{P}_J$ that is the intersection of the edges $\overline{CB}$ and $\overline{AG}$, so that $\angle CJA = \varphi$; and*
- *the point $\mathbf{P}_I$ that is the intersection of the edges $\overline{CA}$ and $\overline{BG}$, so that $\angle BIJ = \angle CJA = \varphi$.*

The point $\mathbf{P}_{\theta_K}$ is then the intersection of the edge $\overline{IJ}$ and the extended chord $\overline{CG}$, so that $\angle IC\mathbf{P}_{\theta_K} = \angle BIJ = \angle CJA = \varphi$.

Definition 7.11 suggests a sequential means of building ΔCIJ, by first finding the point $\mathbf{P}_J$, then $\mathbf{P}_I$, and finally $\mathbf{P}_{\theta_K}$, while simultaneously ensuring that the Brocard angle φ is met through the correct positioning of G. So to get $\mathbf{P}_J$, we begin with the point $\mathbf{P}_{\theta_A}$. We must then place G in such a way so that the extended chord $\overline{AG}$ will cross the extended edge $\overline{CB}$ at $\mathbf{P}_J$. More specifically, we can place G so that it partially fulfils the Brocard conditions at $\angle GJC$ and $\angle GCI$. Geometrically, referring to our example in Figure 7.8,

$$\angle GJC = \angle CGI, \text{ if and only if } \angle CGA = \angle BCA = \mu.$$

Since we already know the angle $\mu = (\angle BCA)$, then by Thales' theorem [26] we can construct a circle Θ where the edge $\overline{CA}$ sits snugly inside it, and where any third point, say G_i, on the circumference of Θ makes the angle μ with the points $\mathbf{P}_C$ and $\mathbf{P}_{\theta_A}$; that is, we get $\angle CG_iA = \mu$.

We are only interested in the point G_i where it sits on the arc of Θ bounded by the extended edges $\overline{CB}$ and $\overline{CA}$. In actual fact, this range of possible positions for G_i is even smaller, since we can further eliminate part of this arc bounded by the extended edge $\overline{CB}$ and a line l_B through $\mathbf{P}_{\theta_B}$ that is parallel to $\overline{CA}$. That is, we know that $\mathbf{P}_I$ will not lie on the extended edge $\overline{CA}$ if we select G_i to lie

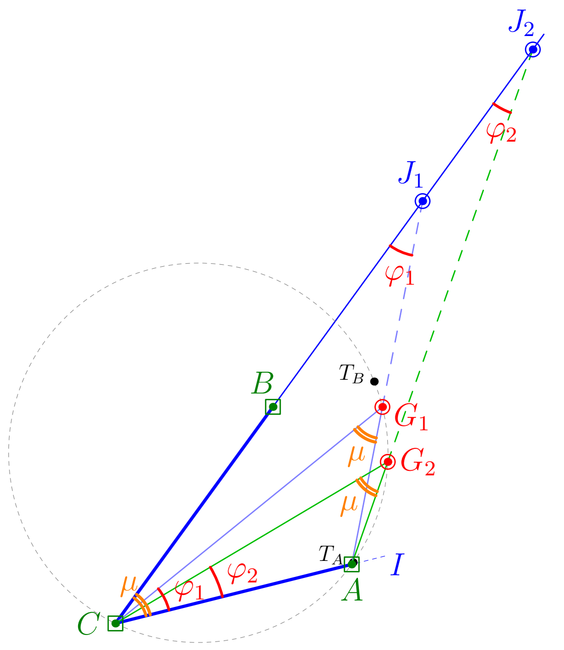


Figure 7.8: Determing the Brocard point G based on angle μ.

within this out-of-range region, and thus we should not consider those positions in our computations. Similarly, we eliminate the other part of the arc bounded by the extended edge $\overline{CA}$ and a line l_A through $\mathbf{P}_{\theta_A}$ that is parallel to $\overline{CB}$.

Lemma 7.12 *Let T_A and T_B be the points on the circle Θ intersected by the lines l_A and l_B, respectively, within the bounds of the extended edges $\overline{CA}$ and $\overline{CB}$. Then, for the Brocard ΔCIJ built from ΔABC, the Brocard point G lies on the arc of Θ between T_A and T_B.*

Proof. From our discussions above, it is clear that the Brocard point G must lie within the arc of Θ between T_A and T_B, in order for the points $\mathbf{P}_I$ and $\mathbf{P}_J$ to lie on the extended edges $\overline{CA}$ and $\overline{CB}$. ■

Corollary 7.13 *The search for the Brocard point G of ΔCIJ on the arc of Θ between T_A and T_B takes logarithmic steps.*

Proof. Let us first select G to lie on the halfway point between T_A and T_B on Θ. Following which we obtain the points $\mathbf{P}_J$ and $\mathbf{P}_I$ as per stated in Definition 7.11. By determining the point $\mathbf{P}_J$ in this way, we fulfil part of the Brocard angle condition at $\angle GJC$ and $\angle GCI$. Then with the point $\mathbf{P}_I$, we obtain the angle $\angle GIJ$, and as long as the three angles $\angle GJC$, $\angle GCI$, and $\angle GIJ$ are not equal, we have not found the Brocard angle φ. This also means we have not found the

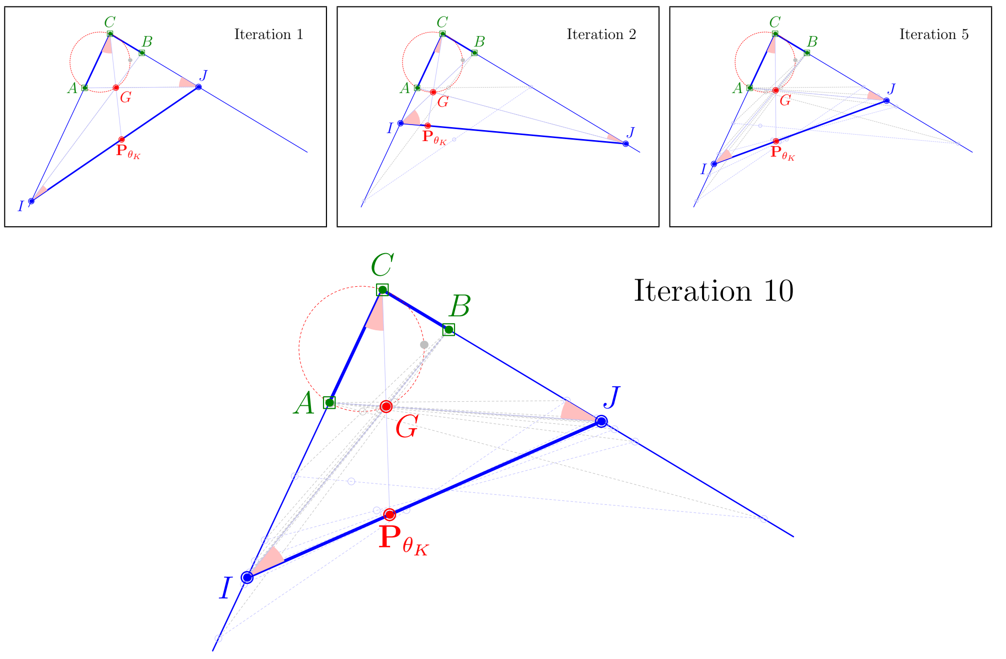


Figure 7.9: Building the Brocard triangle ΔCIJ from ΔABC.

correct Brocard point G for ΔCIJ. We rectify this by adjusting G to another position within T_A and T_B.

- If $\angle GIJ < \angle GCA$ then pick a new G that is on the halfway point between T_A and the current G.
- If $\angle GIJ > \angle GCA$ then pick a new G that is on the halfway point between T_B and the current G.

We repeat the steps above to compute the related $\mathbf{P}_J$ and $\mathbf{P}_I$, with respect to the new G, each time only examining exponentially decreasing portions of Θ between T_A and T_B until the Brocard angle condition in Equation 7.13 is satisfied. ■

The pseudo-codes in Algorithm 3 contains the iterative method described above to obtain the Brocard ΔCIJ. It takes the constraint-triangle ΔABC as input, as well as the value ϵ that will stop the iteration process once all three Brocard angles $\angle GJC$, $\angle GCI$, and $\angle GIJ$ are within $\angle\epsilon$ of each other.

We illustrate the building of ΔCIJ in Figure 7.9, where it took 10 iterations before the algorithm terminates, with a setting of $\epsilon = 0.05$ degrees.

Lemma 7.14 *The iteration method in Algorithm 3 to obtain the Brocard ΔCIJ*

```
Algorithm 3: ComputeBrocardTriangle ($\Delta ABC$, $\epsilon$)
   Output: The Brocard $\Delta CIJ$ where the Brocard angles $\varphi$ differ by $\epsilon$.
 1 Set $\mu \leftarrow \angle BCA$;
 2 Set $\Theta \leftarrow$ ConstructCircle($\mathbf{P}_C$, $\mathbf{P}_{\theta_A}$, $\mu$);
 3 Set $l_A \leftarrow$ Line through $\mathbf{P}_{\theta_A}$, parallel to $\overline{CB}$;
 4 Set $l_B \leftarrow$ Line through $\mathbf{P}_{\theta_B}$, parallel to $\overline{CA}$;
 5 Set $T_A \leftarrow$ Intersection($\Theta$, $l_A$) within the bounds of $\Delta ABC$;
 6 Set $T_B \leftarrow$ Intersection($\Theta$, $l_B$) within the bounds of $\Delta ABC$;
 7 Initialize $\delta \leftarrow \mu$;
 8 while $|\delta| > \epsilon$ do
 9     $G \leftarrow$ MidPointArc($\Theta$, $T_A$, $T_B$);
10     $\mathbf{P}_I \leftarrow$ Intersection(ExtendedEdge($\overline{AG}$), ExtendedEdge($\overline{CB}$));
11     $\mathbf{P}_I \leftarrow$ Intersection(ExtendedEdge($\overline{BG}$), ExtendedEdge($\overline{CA}$));
12     $\delta \leftarrow \angle GIJ - \angle GCA$;
13     if $\delta > 0$ then $T_A \leftarrow G$;
14     else $T_B \leftarrow G$;
15 return $\Delta CIJ$;
```

from the constraint-triangle ΔABC terminates and converges to a single solution.

Proof. The directed iterative steps in Lines 13 and 14 ensures that the iteration picks up the correct finer portion of the arc on Θ where the Brocard point G lies, and in doing so, converges towards the correct value of φ within the tolerance of ϵ. ■

Thus, there is only one location on the circumference of the circle Θ for G to be the (first) Brocard point of ΔCIJ, based solely on the three fixed points A, B, and C. Once we have determined G, we can then determine $\mathbf{P}_{\theta_K}$, and subsequently obtain the Brocard inellipse $f(\mathbf{x})$.

Figure 7.10 depicts the three resultant arcs ω_{AB} from the computed Brocard inellipse – again based on the same underlying example ellipse that we used in the earlier sections.

7.2.4 Mandart Inellipse

The Mandart inellipse of ΔCIJ is tangent at the vertices of the *extouch* triangle $\Delta AB\mathbf{P}_{\theta_K}$ (of ΔCIJ). That is, from the example in Figure 7.11, the points A, B, and $\mathbf{P}_{\theta_K}$ are the points of tangency of ΔCIJ with its three *excircles* centred

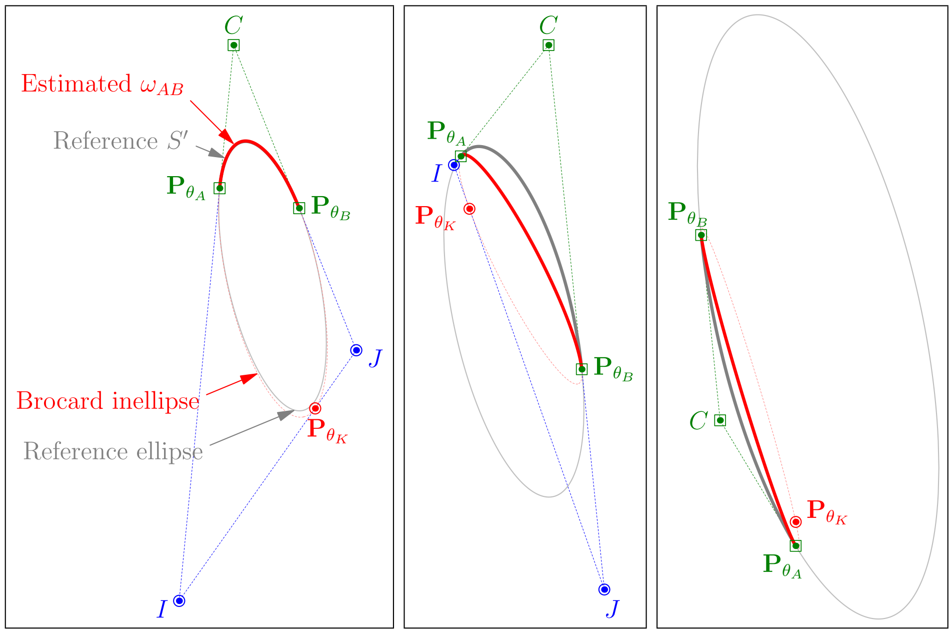


Figure 7.10: Estimating ω_{AB} with the Brocard inellipse.

at R, S, and T. The Mandart inellipse has the inconic parameters

$$x : y : z = \frac{i}{j+c-i} : \frac{j}{i+c-j} : \frac{c}{i+j-c} \tag{7.15}$$

where i, j, and c are the sidelenghts of ΔCIJ. Each of the three excircles are tangent at two other contact points on the extended edges of ΔCIJ, which incidentally bind the circles. For instance, the circle centred at R is tangent at R_1 on the extended edge $\overline{JI}$ and at R_2 on the extended edge $\overline{JC}$. Similarly, the circle centred at S is tangent at S_1 and S_2 on the extended edges $\overline{IC}$ and $\overline{IJ}$, and the circle centred at T is tangent at T_1 and T_2 on the extended edges $\overline{CJ}$ and $\overline{CI}$. Because of the positioning of these excircles, the Brianchon point G is known as the Nagel point of ΔCIJ.

Furthermore, the nine contact points of the three excircles are related to the semi-perimeter of ΔCIJ. Let σ be the semi-perimeter of ΔCIJ, where

$$\sigma = \frac{1}{2}(i+j+c).$$

Then

- $\|CT_1\| = \|CT_2\| = \sigma$;
- $\|JR_1\| = \|JR_2\| = \sigma$; and

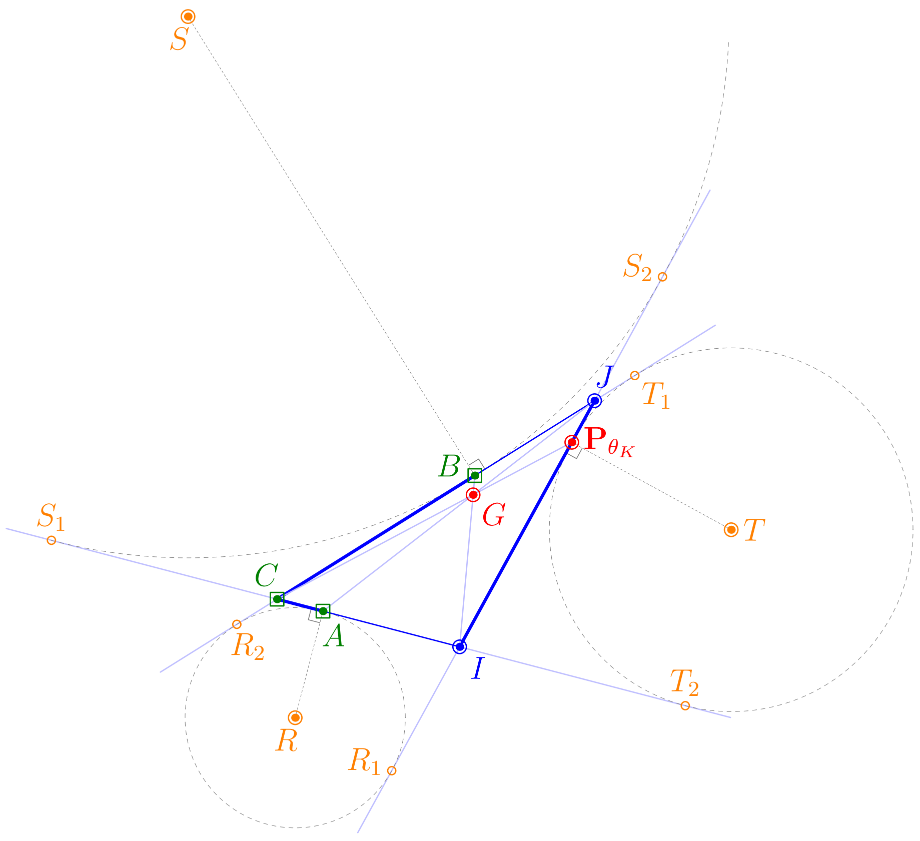


Figure 7.11: The Mandart triangle ΔCIJ with excircles R, S, and T.

- $\|IS_1\| = \|IS_2\| = \sigma$.

Also, the following three expressions hold:

$$\|CA\| = \|CR_2\| = \|J\mathbf{P}_{\theta_K}\| = \|JT_1\| = \sigma - i = d_1 \tag{7.16}$$
$$\|CB\| = \|CS_1\| = \|I\mathbf{P}_{\theta_K}\| = \|IT_2\| = \sigma - j = d_2 \tag{7.17}$$
$$\|IA\| = \|IR_1\| = \|JB\| = \|JS_2\| = \sigma - c = d_3 \tag{7.18}$$

We can multiply the above lengths in such a way that the following equality holds true:

$$\|CB\| \cdot \|J\mathbf{P}_{\theta_K}\| \cdot \|IA\| = \|\mathbf{P}_{\theta_K}I\| \cdot \|AC\| \cdot \|BJ\|.$$

This is equivalent to

$$\frac{\|CB\|}{\|BJ\|} \cdot \frac{\|J\mathbf{P}_{\theta_K}\|}{\|\mathbf{P}_{\theta_K}I\|} \cdot \frac{\|IA\|}{\|AC\|} = 1.$$

Then, by Ceva's theorem [119], the lines $\overline{C\mathbf{P}_{\theta_K}}$, $\overline{JA}$, and $\overline{IB}$ are concurrent at the Nagel point G. This realization of the Cevian length's relation to the semi-perimeter enables us to generalize the Mandart ΔCIJ from the constraint-triangle ΔABC.

Definition 7.15 *Following Definition 7.5 and Equations 7.16 to 7.18, the vertices I and J of the Mandart ΔCIJ with respect to ΔABC are, respectively,*

- *the point $\mathbf{P}_I$ lying on the extended edge $\overline{CA}$ that is d_3 distance away from $\mathbf{P}_{\theta_A}$; and*
- *the point $\mathbf{P}_J$ lying on the extended edge $\overline{CB}$ that is d_3 distance away from $\mathbf{P}_{\theta_B}$.*

The length of the edge $\overline{IJ}$ is exactly $d_1 + d_2$, and that the point $\mathbf{P}_{\theta_K}$ lies on the edge $\overline{IJ}$, and is d_1 distance away from $\mathbf{P}_J$ as well as d_2 distance away from $\mathbf{P}_I$.

The key to building the proper Mandart ΔCIJ from ΔABC is to find the unknown length of d_3 that will fit $\overline{IJ}$ exactly onto the bounds of the extended edges $\overline{CA}$ and $\overline{CB}$. As was with the case of finding the Brocard triangle, we now face a similar situation where there are more unknown variables than there are unique conditions for us to solve the vertices of the Mandart ΔCIJ via direct mathematical means. Thus, we propose another iterative approach, making use of the properties of the Mandart triangle that we have met so far.

In particular, we shall use the excircle centred at T together with its two tangent points T_1 and T_2, as well as the pivotal vertex $\mathbf{P}_C$ to explain our concept. We begin by stating the following three observations, using Figure 7.12 as reference:

Observation I. The points T_1 and T_2 are on a circle Θ of radius σ centred at $\mathbf{P}_C$, where σ is the semi-perimeter of ΔCIJ.

Observation II. The point $\mathbf{P}_I$ lies between $\mathbf{P}_{\theta_A}$ and T_2, and is d_2 distance away from T_2.

Observation III. The point $\mathbf{P}_J$ lies between $\mathbf{P}_{\theta_B}$ and T_1, and is d_1 distance away from T_1.

Lemma 7.16 *The point T_1 lies on the extended edge $\overline{CB}$ between $T_{1_{min}}$ and $T_{1_{max}}$, if $d_1 > d_2$. The points $T_{1_{min}}$ and $T_{1_{max}}$ are related in their lengths in the following manner:*

$$\|CT_{1_{min}}\| = \|AT_{1_{max}}\| = d_1 + d_2.$$

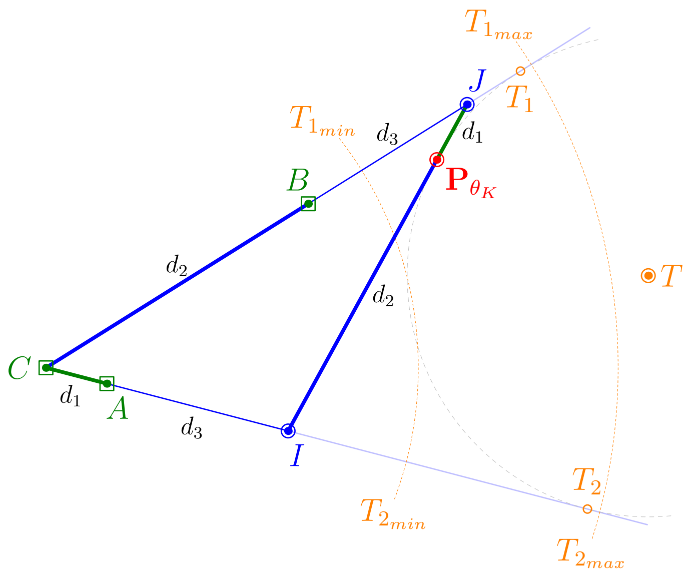


Figure 7.12: Building the Mandart ΔCIJ based on the semi-perimeter information.

If, on the other hand, $d_2 > d_1$, then the point T_2 lies on the extended edge $\overline{CA}$ between $T_{2_{min}}$ and $T_{2_{max}}$. The points $T_{2_{min}}$ and $T_{2_{max}}$ are related in their lengths in the following manner:

$$\|CT_{2_{min}}\| = \|BT_{2_{max}}\| = d_1 + d_2.$$

Proof. Suppose that $d_3 = 0$. Then it follows that $T_{1_{min}}$ and $T_{2_{min}}$ are the points that are $d_1 + d_2$ distance away from the centre of the circle Θ centred at $\mathbf{P}_C$. On the other hand, if $d_3 > 0$, then the maximum distance that $T_{1_{max}}$ and $T_{2_{max}}$ can have is bounded by the length of $\overline{IJ}$, where $\|IJ\| = d_1 + d_2$, from the tip of either $\mathbf{P}_{\theta_A}$ or $\mathbf{P}_{\theta_B}$, whichever is farther away from $\mathbf{P}_C$, to the extended edge on the opposite side. Take for example in Figure 7.12, where $d_2 > d_1$. Then $T_{2_{max}}$ is the point on the extended edge $\overline{CA}$ that is $d_1 + d_2$ distance away from $\mathbf{P}_{\theta_B}$. As a consequence of **Observation I**, $T_{1_{max}}$ is exactly the same distance away from $\mathbf{P}_C$ as $T_{2_{max}}$ is, since they both lie on the circumference of the same circle Θ centred at $\mathbf{P}_C$. ■

The two minimum and maximum values set up our search range for T_1 and T_2 that will lead us to obtain the positions of the vertices I and J of ΔCIJ.

Corollary 7.17 *The recursive search method for the Nagel point G of the Mandart ΔCIJ given the constraint-triangle ΔABC terminates after a finite number of logarithmic steps.*

Proof. Clearly, if we correctly determine the length d_3, then we can easily obtain the Nagel point G in constant time, since it is also the Brianchon point of ΔCIJ.

Now, to get the correct length of d_3, we first set up the search range for T_1 and T_2 based on Lemma 7.16.

Beginning with the first estimate, we select T_1 to be the midway point between $T_{1_{min}}$ and $T_{1_{max}}$. Again, as a consequence of **Observation I**, T_2 is automatically selected as the midpoint between $T_{2_{min}}$ and $T_{2_{max}}$. We then determine the points $\mathbf{P}_I$ and $\mathbf{P}_J$ based on T_1 and T_2, as per explained in **Observations II** and **III**. This immediately ensures that the first two properties in Definition 7.15 are met. The third property, and thus the Mandart ΔCIJ, is fulfilled if and only if the length we measure from the resultant edge $\overline{IJ}$ is equal to $d_1 + d_2$. That is, if $\|IJ\| \neq d_1 + d_2$, then we need to adjust T_1 and T_2 for the next iteration.

- If $\|IJ\| < d_1 + d_2$ then pick a new T_1 that is on the halfway point between $T_{1_{max}}$ and the current T_1.
- If $\|IJ\| > d_1 + d_2$ then pick a new T_1 that is on the halfway point between $T_{1_{min}}$ and the current T_1.

The new T_2 will automatically follow the new T_1 based on **Observation I**. We repeat the steps above to compute the related $\mathbf{P}_I$ and $\mathbf{P}_J$. And by using $\|IJ\|$ as reference at each iteration, we fine-tune the search, each time examining only smaller portions of the search range, until we find the Mandart ΔCIJ. ■

The pseudo-codes in Algorithm 4 contains the iterative method described above to obtain the Mandart ΔCIJ, and we can state it simply without any reference to either the circle Θ or the reference circle centred at T. It takes the constraint-triangle ΔABC as input, as well as the value ϵ that will stop the iteration process once the length of the edge $\overline{IJ}$ is satisfactorily determined.

We give an illustration of the building of ΔCIJ in Figure 7.13 from the given ΔABC, where it took 12 iterations before the algorithm terminates, with a setting of $\epsilon = 0.01$.

We complete this section with Figure 7.14 where we show the three resultant arcs ω_{AB} from the computed Mandart inellipse.

7.3 Adaptive Midpoint Inellipse

Let us now consider a slight variation to the classical inellipses that we have met so far, where we shall not impose a fixed contact-triangle ΔCIJ based on ΔABC to determine its inellipse $f(\mathbf{x})$. That is, we want to allow the generation of ω_{AB} in accordance to the trajectory of the ordered points in TS, without compromising the integrity of the curve within against the amount of computation required to obtain it. Out of the four inellipses we discussed, the Steiner inellipse and the Orthic inconic offer “direct” computations to get to the contact-triangle. However, only the Steiner inellipse is stable enough to allow us to further tweak

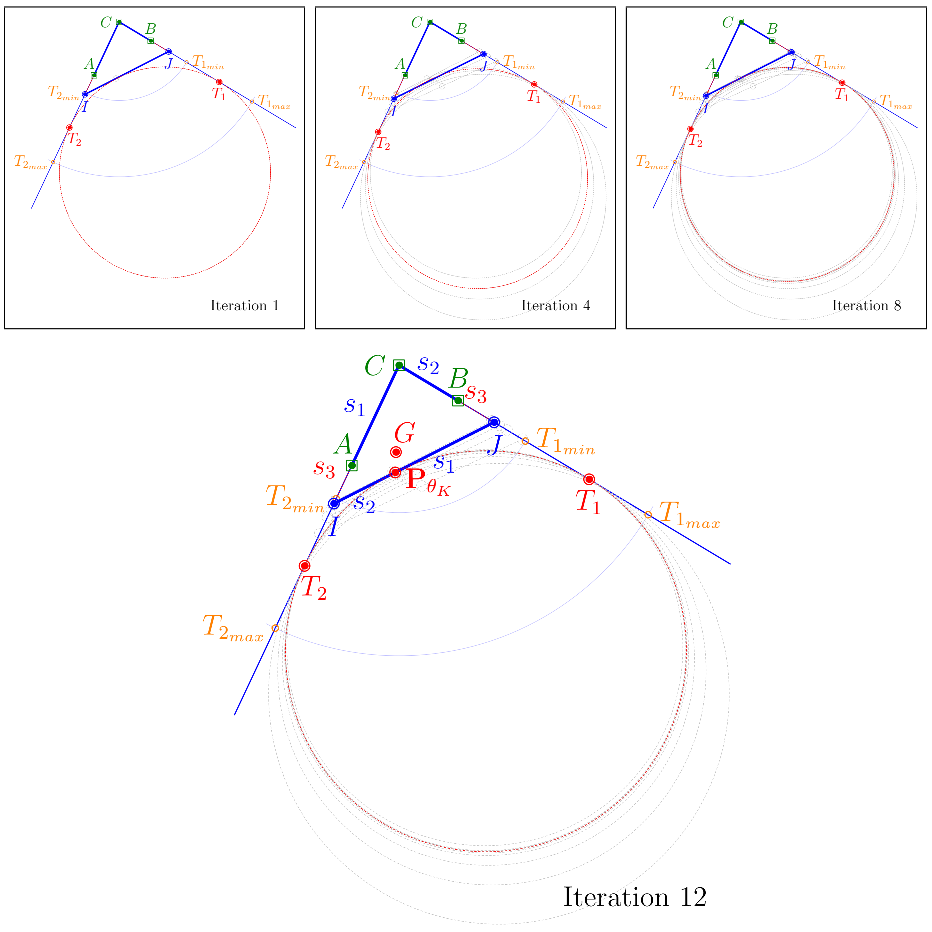


Figure 7.13: Building the Mandart triangle ΔCIJ from ΔABC.

its internal methods to make it more susceptible to the varying conditions in TS. Thus, in addition to its stability and speedy computations, the Midpoint Lemma 7.18 below completes our adaptive approach to generate an arc ω_{AB} that will fit desirably onto TS.

Lemma 7.18 *Let M be the midpoint of the vertices A and B of the constraint-triangle ΔABC, and let the point $\mathbf{P}_{\theta_L}$ be the intersection of the* median *line $\overline{CM}$ and the inellipse $f(\mathbf{x})$ of ΔCIJ. Then, the derivative at $\mathbf{P}_{\theta_L}$ is the slope of the baseline $\overline{AB}$.*

Proof. We shall stick to the earlier assumption we made that the curve between the vertices $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ of ΔABC is quadratic. That is, the point on the second-order curve where the curvature changes its sign, when traversing from A to B (or vice-versa), is the point $\mathbf{P}_{\theta_L}$ that is farthest away from the baseline $\overline{AB}$, and whose tangent is parallel to the slope of the line $\overline{AB}$. This tangent line

Algorithm 4: ComputeMandartTriangle (ΔABC, ϵ)

```
Output: The Mandart $\Delta CIJ$ where $\|IJ\| < \epsilon$.
1  Set $d_1 \leftarrow \|CA\|$, $d_2 \leftarrow \|CB\|$;
2  if $d_1 > d_2$ then
3      $P \leftarrow$ Intersection($\overline{CB}$, Circle($rad = d_1 + d_2$, $ctr = \mathbf{P}_{\theta_B}$))
4  else
5      $P \leftarrow$ Intersection($\overline{CA}$, Circle($rad = d_1 + d_2$, $ctr = \mathbf{P}_{\theta_A}$))
6  Initialize $d_{min} \leftarrow d_1 + d_2$;
7  Initialize $d_{max} \leftarrow \|CP\|$;
8  Initialize $\delta \leftarrow d_{max}$;
9  while $|\delta| > \epsilon$ do
10     $d_{mid} \leftarrow (d_{min} + d_{max})/2$;
11     $\mathbf{P}_I \leftarrow$ Intersection($\overline{CA}$, Circle($rad = d_{mid} - d_2$, $ctr = \mathbf{P}_C$));
12     $\mathbf{P}_J \leftarrow$ Intersection($\overline{CB}$, Circle($rad = d_{mid} - d_1$, $ctr = \mathbf{P}_C$));
13     $\delta \leftarrow \|IJ\| - d_1 - d_2$;
14     if $\delta < 0$ then $d_{min} \leftarrow d_{mid}$;
15     else $d_{max} \leftarrow d_{mid}$;
16 return $\Delta CIJ$;
```

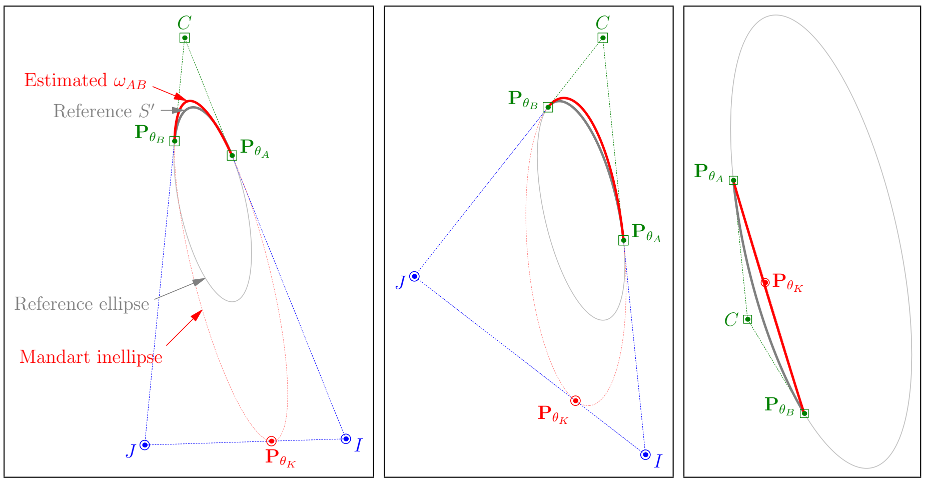


Figure 7.14: Estimating ω_{AB} with the Mandart inellipse.

cuts the edges $\overline{CA}$ and $\overline{CB}$ at the points q and r, respectively, and this conforms to de Casteljau's midpoint theorem, which we shall explain in further details in section 8.1, so that $\mathbf{P}_{\theta_L}$ is the midpoint of q and r – see Figure 7.15 for an example. The resultant ΔqrC is similar to ΔABC. ■

In this case, we do not need to find the contact-triangle ΔCIJ, since the above lemma ensures that we fulfil Condition 7.7 and Condition 7.8 with the point $\mathbf{P}_{\theta_L}$ and its derivative. However, it is not difficult to see that the point $\mathbf{P}_{\theta_K}$ lies on the exact opposite of $\mathbf{P}_{\theta_L}$ on the ellipse $f(\mathbf{x})$, on the same median line $\overline{CM}$. Similarly, the tangent at $\mathbf{P}_{\theta_K}$ is also the slope of the line $\overline{AB}$, and that the tangent line cuts the extended edges $\overline{CA}$ and $\overline{CB}$ at the points I and J, respectively. Consequently, the point $\mathbf{P}_{\theta_K}$ is the midpoint of I and J, and it also lies on the extended median line $\overline{CM}$. The resultant contact-triangle ΔCIJ is similar to ΔABC, but it is not necessarily the Steiner triangle.

Corollary 7.19 *The centre E of the inellipse $f(\mathbf{x})$ of ΔCIJ lies on the extended median line $\overline{CM}$.*

Proof. The line through the points C and M is one of the three median lines of the contact-triangle ΔCIJ, where it connects the vertex C to the midpoint of the edge $\overline{IJ}$. The incentre of any triangle is where the three median lines meet. Subsequently, the centre of the inellipse (or incircle) is also the incentre of the triangle, which in our case is called the contact-triangle ΔCIJ. Hence, E lies on the line through C and M. ■

So, in comparison to the method deriving the Steiner inellipse, the adaptive midpoint we have been referring to in our case is the point $\mathbf{P}_{\theta_L}$, lying on the inellipse $f(\mathbf{x})$ in the trajectory of $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$. The "adaptation" process of $\mathbf{P}_{\theta_L}$ then refers to locating the most appropriate point inside the constraint-triangle ΔABC from the set of discrete points $p_i \in TS$. We present three approaches to do this.

Adaptation I. Pick the point $p_i = \mathbf{P}_{\theta_L}$ that lies on $\overline{CM}$. If no such point exists, then pick the point $\mathbf{P}_{\theta_L}$ that is the intersection of the line-segment $\overline{p_i p_{i+1}}$ and the line $\overline{CM}$.

Adaptation II. Pick the first point $p_i = \mathbf{P}_{\theta_L}$ that has the farthest perpendicular distance from the baseline $\overline{AB}$.

Adaptation III. Pick the point $p_i = \mathbf{P}_{\theta_L}$ whose derivative is the closest to the slope of $\overline{AB}$. If several of such points exists, then break the tie by selecting the first point with the shortest perpendicular distance to $\overline{CM}$.

Adapting to option **I** implies that we force the derivative at $\mathbf{P}_{\theta_L}$ to be the slope of the line $\overline{AB}$, while by options **II** and **III**, we simply adapt the measured derivative at the selected point.

In keeping to the earlier standards, we shall again use the same underlying ellipse to guide our examples in this section. However, in addition to the arbitrarily selected anchor-points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$, we also chose a random number of discrete points lying in the trajectory of the anchor-points to form the elements in TS. We illustrate the outcomes of the three adaptations in the Figure ?.

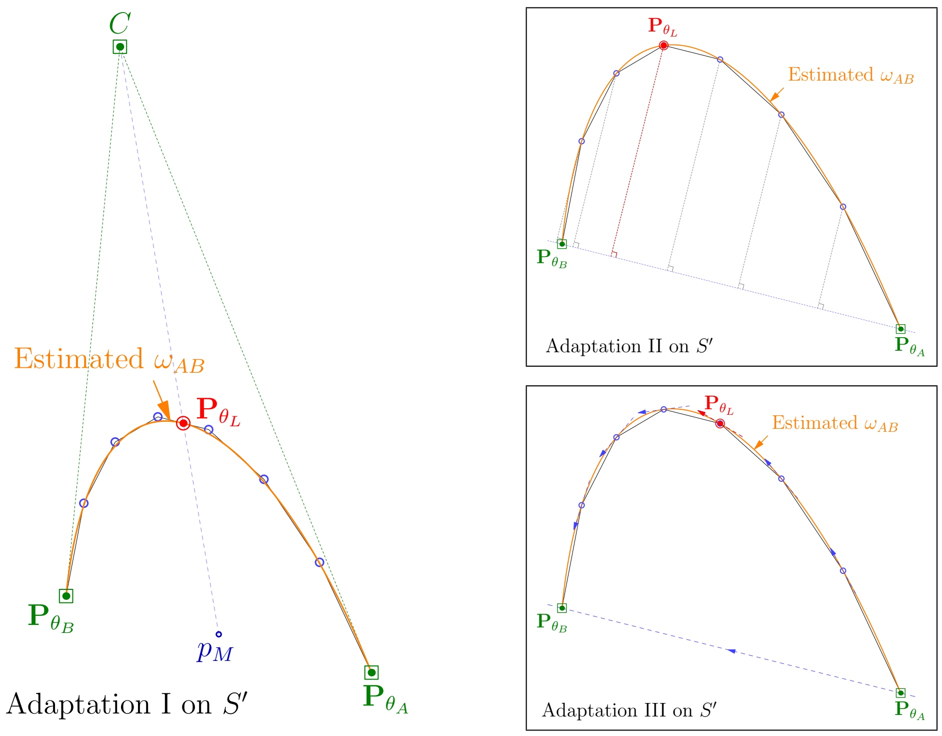


Figure 7.15: Adaptation methods I, II, and III to estimate ω_{AB} onto TS.

7.4 Measuring the Quality of Approximated Curves

Having now understood the various ways in which we can instantiate and control an inellipse $f(\mathbf{x})$ with incomplete information, we should also have a way to measure how the resultant arcs of these inellipses fair when we use them to estimate any given TRACE T. Earlier on, we showed how well an approximated simplified polyline measures up to its original, in terms of the line-segments connecting the points *kept* with respect to the points *eliminated* (see Equation 5.1). The smaller the measured error value err_T is, the better the quality of the approximation.

Similarly, we shall use this same but slightly modified approach to measure the quality of the approximated CURVES– elliptic, or otherwise – with respect to the original T, for the same reasons we gave in section 5.2. We will be discussing one more family of curves in chapter 8, and we will see that our discussions here on the quality of approximations should generally be applicable for both types. For now though, let us concentrate on measuring the quality of curves with regards to the elliptic arcs.

Let ω_{AB} be the elliptic arc estimating a section of T between $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$. The arc ω_{AB} is part of the ellipse described by $f(\mathbf{x})$, whose centre is at $E \equiv (x_E, y_E)$. Let $TS = \{p_1, p_2, \ldots, p_m\} \subseteq T$ be the section of T that is approximated by ω_{AB}, where the endpoints p_1 and p_m correspond to $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$, respectively. Also, for all $p_i \in TS$, let L_i be a straight line through E and p_i. Then, for any $p_i \equiv (x_i, y_i)$, there exists a corresponding point on the arc $p_{\theta i} \equiv (x_{\theta i}, y_{\theta i}) \in \omega_{AB} \subset f(\mathbf{x})$, which is the intersection of the ellipse $f(\mathbf{x})$ and the line L_i, closest to p_i. Now, if we use the same analogy we did with measuring the quality of simplified lines from chapter 5, then it is the same case here when we consider the endpoints of TS *kept* and all others in between them *eliminated*. In further contrast, the arc ω_{AB} is used in place of the line-segment $\overline{\mathbf{P}_{\theta_A}\mathbf{P}_{\theta_B}}$ to find $p_{\theta i}$ using the centre E of $f(\mathbf{x})$ as reference, instead of p'_i. It follows then, the square of the error measure $\mathrm{err}_{T_{AB}}$ of ω_{AB}, with respect to the original T_{AB}, is computed as

$$\mathrm{err}^2_{T_{AB}} = \frac{1}{m}\sum_{i=1}^{m-1} \|\overline{p_i p_{\theta i}}\|^2, \tag{7.19}$$

where

$$\|\overline{p_i p_{\theta i}}\|^2 = (x_i - x_{\theta i})^2 + (y_i - y_{\theta i})^2.$$

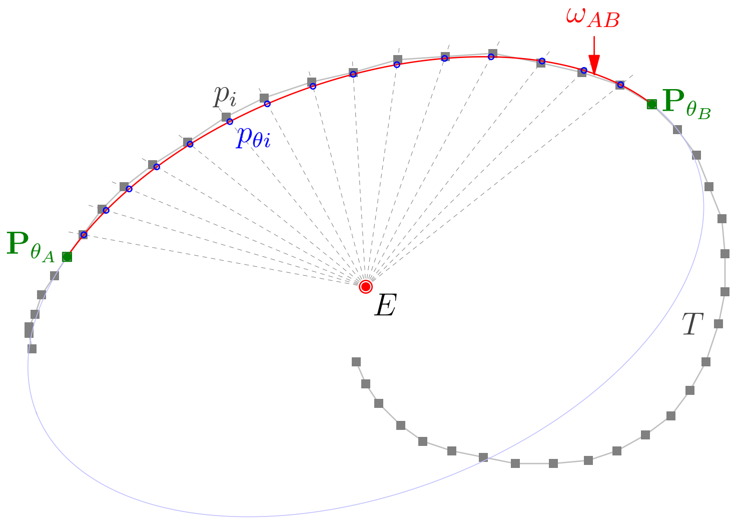


Figure 7.16: Measuring error of an elliptic arc.

Thereupon, the full error measure for the entire TRACE T is the sum of all error measures $\mathrm{err}_{T_{AB}}$ in all k segments of T.

$$\mathrm{err}_T = \sum_{j=1}^{k} \mathrm{err}_{T_{AB\_j}} \tag{7.20}$$

7.5 Elliptic Segmentation

Up until this point, we have established the various ways in which we can obtain an elliptic arc to approximate a portion of a full ellipse without access to complete information. We built this up until the last section where saw that the same can be achieved when instituting the arc ω_{AB} over one segment of a Trace $TS \subseteq T$, with or without complete information of the underlying Curve's characteristics derived from TS. The choices for TS in our examples so far have been arbitrary; mainly to illustrate that the smoothed version of the original Trace T can be composed of a set of well-placed elliptic arcs ω_{AB}, and that the keys for the confluent joints of these arcs lie in making the correct choices for the corresponding TraceSegments TS. In other words, only when the decomposition of T into its segments are properly primed to receive elliptic arc estimates, can there be a desirable solution to the problem of smoothing T by its *elliptic segmentation*.

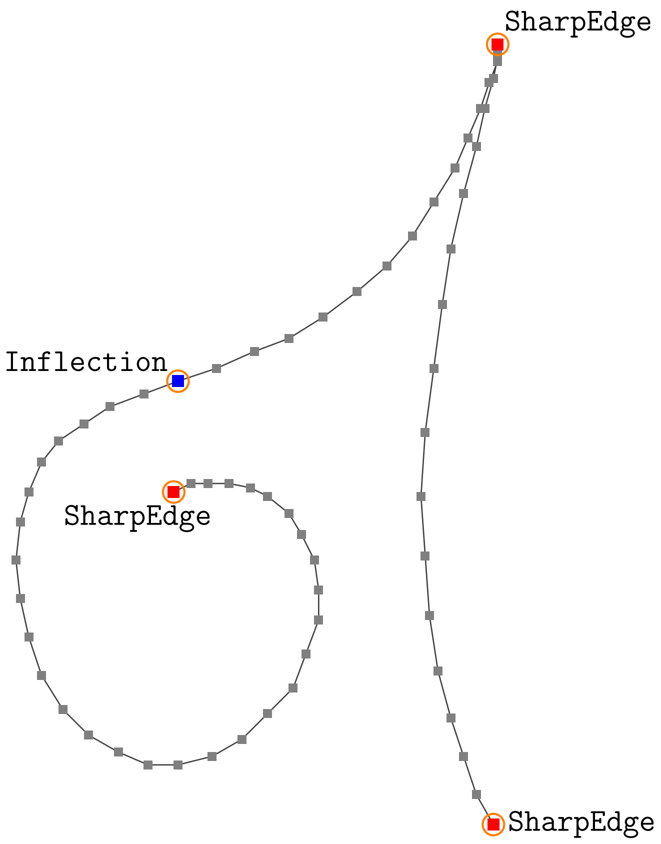


Figure 7.17: An ActiveTrace T^{M} obtained by putting the original Trace T through the active-sampling algorithm from section 3.5, showing the marked `SharpEdge` and `Inflection` crucial points. The rest of the grey points are `Normal`.

Our observations in the earlier chapters pertaining to the handwritten Traces showed that any Trace T can have within it piecewise connections and sharp edges. And considering that all the elliptic arcs we used for our approximations are inherently second-order Curves, then care must be taken when handling the inflection points in T. Recall that we termed these two types of points 'crucial', and stated in Definition 3.14 and Definition 3.15 on how we can actively detect all inflection points and sharp-edge vertices in T, through the active-sampling algorithm. This then becomes the first of two steps of the elliptic segmentation of T, and that the product of the active-sampling algorithm (Algorithm 1) on T is its equivalent ActiveTrace T^{M}; where all the internal points $p_i \in T^{\mathrm{M}}$ are marked as either `SharpEdge` (G^0 continuity), `Inflection` (G^2 continuity), or `Normal`. We give an example of this in Figure 7.17.

Suppose that there are $k+1$ such `SharpEdge` and `Inflection` points in T^{M}. Then, using these crucial points as segment markers of T^{M}, we get k contiguously ordered segments of T^{M}, so that $T^{\mathrm{M}} = \langle TS_1^{\mathrm{M}}, \ldots, TS_k^{\mathrm{M}} \rangle$, where the endpoints of each segment are the identified crucial points of T^{M}.

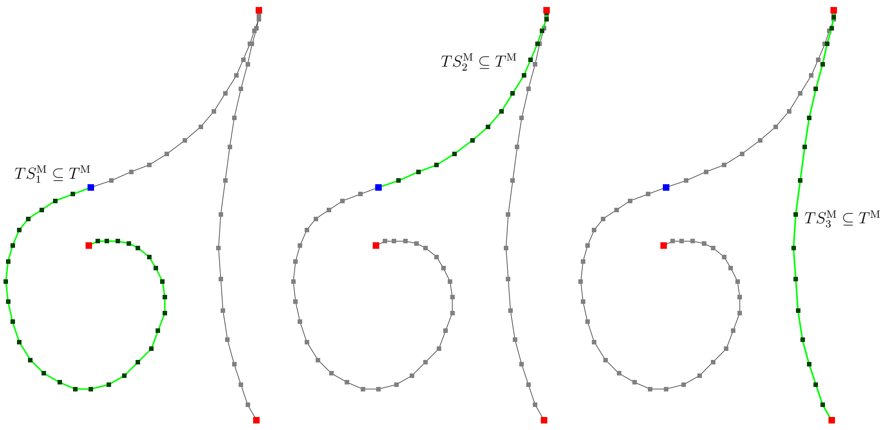

$TS_1^{\mathrm{M}} \subseteq T^{\mathrm{M}}$
$TS_2^{\mathrm{M}} \subseteq T^{\mathrm{M}}$
$TS_3^{\mathrm{M}} \subseteq T^{\mathrm{M}}$

Figure 7.18: Step 1 of the elliptic segmentation of the Trace T.

Let $TS_j^{\mathrm{M}} = \{p_{j1}, \ldots, p_{jm}\}$ be j^{th} segment of T^{M}. Then the endpoints of the intermediate segment TS_j^{M} are also the endpoints of its neighbouring segments TS_{j-1}^{M} and TS_{j+1}^{M}; that is, $p_{(j-1)m} = p_{j1}$ and $p_{jm} = p_{(j+1)1}$, for $1 < j < k$. Consequently, by our Proposition 3.12, every segment $TS_j^{\mathrm{M}} \subseteq T^{\mathrm{M}}$ is composed of a set of second-order curves, and particularly in this Chapter, we shall interpret the second-order curves to be the elliptic arcs ω_{AB}. This leads us to the second step of the elliptic segmentation – the breakdown step of the main problem – to identify the rest of the crucial G^1 points in each segment TS^{M} of T^{M}.

7.5.1 The Naïve Approach

Without loss of generality, let us refer to the ActiveTrace segment $TS^{\mathrm{M}} \subseteq T^{\mathrm{M}}$ whose endpoints are marked as either `SharpEdge` or `Inflection`, simply as TS, where $TS = \{p_1, \ldots, p_m\}$. Also, we consider every point $p_i \in TS$ as an `ActivePoint`; that is, as we established in section 3.4, an `ActivePoint` p_i is embedded within it, additional information with regards to its positional standing with respect to other neighbouring points in the Trace. Such *active* information include the first and second derivatives, as well as the measure of curvature at p_i.

The naïve approach makes direct use of these active information as it iterates from p_1 to p_m. The objective is to first find a valid elliptic arc for an initial set of points, and then stretch that arc as far as possible, stopping only at a point, say p_{g+1}, down the segment TS when the arc can no longer be instantiated. The last point p_j where the arc was valid becomes the crucial G^1 continuity point, contributing to the elliptic segmentation of T. The process is repeated from p_j until all m points in TS are considered. We give the pseudo-codes of the naïve approach in Algorithm 5.

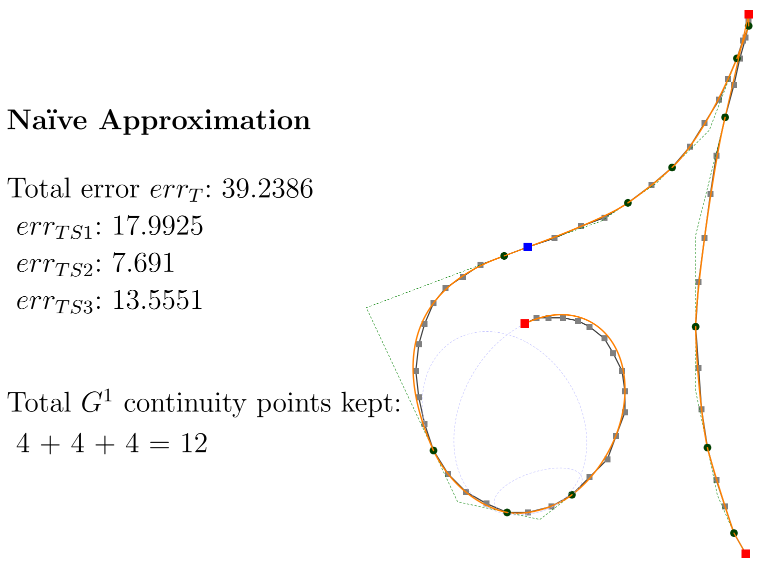


Figure 7.19: The naïve approximation of the Trace T. The green circular dots indicate the G^1 continuity points p_j identified by the approach. The dotted green lines are the tangent lines through p_j, demonstrating the unbroken and confluent joints between segments.

One can use any of the elliptic arcs which are the products of the inellipses or inconics discussed in section 7.2 and section 7.3. However, in this case of approximating TS, we should favour any one of the three adaptive-midpoint inellipses over the others, since we have access to the expected underlying curve's characteristics in TS. This access to the characteristics comes when there are at least three points in TS made available for our approximation methods. The resultant adaptive-midpoint elliptic arcs produced are judgementally better estimates than the arcs from the other methods.

The other notion from this and future approaches that we need to address is the concept of the *invalid* elliptic arcs – how and why they happen. That is, by our discussions of Proposition 3.12, second-order CURVEs can make up the segment TS, and yet we find situations where we cannot get valid arcs to approximate over a set of points arranged in the second-order trajectory of TS.

There are two explanations for this: One is that the set of points are aligned, or almost aligned, to form a straight line; and this is evident if we look at the measures of curvature in each of the `ActivePoint` – they are hovering at about zero. A straight line, by definition, is a degenerate second-order curve. The second reason is that the tangents at the endpoints of TS are parallel, or almost parallel, to each other. And when they are completely parallel, the tangent lines through $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ will not intersect at $\mathbf{P}_C$, and hence, we cannot instantiate the constraint-triangle ΔABC. This second reason is also a consequence of the first, since the gradients at both endpoints of a straight line-segment are equal. Thus, when either of these two situations arises, the six conditions necessary to instantiate the ellipse $f(\mathbf{x})$ extracted from the current set of points returns no solutions from the linear system of six equations that we highlighted in section 7.1. In other words, the six conditions derived from the set of points we want to approximate cannot yield the six coefficients required to describe $f(\mathbf{x})$.

We saw from the steps above that whenever we encounter invalid arcs while the algorithm is still running in the midst of segments, we simply dismiss the point p_j that contributed to the problem, and either select another point for consideration, or stop and revert back to the previous point whose corresponding arc is valid. This, however, becomes a substantial problem if we cannot find a valid arc for the set of points in the final segment of TS. Thus, confronted with such a case, we offer the following solutions.

Solving the first situation is straight-forward. The points are already poised to receive a straight line, so we simply pass a straight line-segment connecting the endpoints $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$. As a matter of observation, this situation almost always happens in the beginning and/or ending parts of the original segment TS_j, whose endpoints are either marked as `SharpEdge` or `Inflection`. Because if we extend the segment TS_j, say into TS_{j+1}, and find the continuation of the straight line at the end of TS_j, then it is most likely that the point p_{jm} is an inflection point.

Dealing with the second situation, however, requires some work. Let $TS' = \{p_1, \ldots, p_m\}$ be the final segment of TS, that yielded an invalid arc because of parallel (or near parallel) gradients at its endpoints. We need to backtrack to a previous point before p_m, preferably to one that lies in the middle of this last segment, say p_k, where $k = m/2$, so that we get valid vertices $\mathbf{P}_C$ for the corresponding constraint-triangles ΔABC. Then, we can attempt to instantiate a valid arc for each of the sub-segment $\{p_1, \ldots, p_k\}$ and $\{p_k, \ldots, p_m\}$. If any of the two sub-segments still results in an invalid arc, or if there were only two points in the last segment of TS' to begin with, then the solution is to 'relax' the last two of the six conditions (Condition 7.7 and Condition 7.8) that are to be fed into the system of linear equations, and this involves guessing the choice for $\mathbf{P}_{\theta_K}$. Coincidentally, any of the other inellipses discussed in section 7.2 will provide the alternative choice in place of the adaptive-midpoint inellipse, since they deal with incomplete information. In retrospect, the resultant elliptic arcs by these methods will maintain the G^1 continuity at the endpoints of TS'.

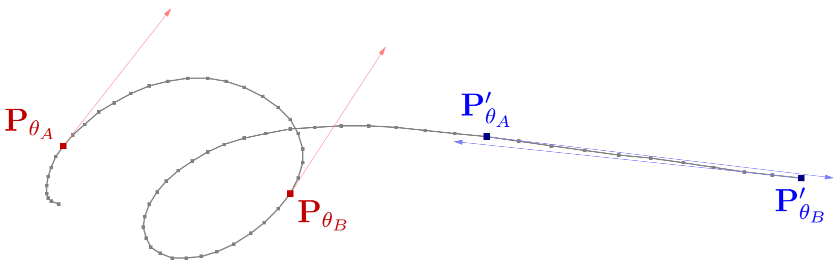


Figure 7.20: Two types of (almost) parallel tangents. The blue segment marked by the tangent lines through $\mathbf{P}'_{\theta_A}$ and $\mathbf{P}'_{\theta_B}$ is an example of the first situation, where the segment is almost already a straight line. The red segmented marked by the tangent lines through $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ describes the second situation.

The approximation of the TRACE T in Figure 7.19 contains an example of both situations in the third segment TS_3. The top end of the segment connecting the sharp-edge vertex and the first G^1 continuity point p_j is a straight line-segment. The bottom end deals exclusively with the second situation as we described above, where we utilized the Steiner inellipse to complete the arc approximation. We list the recursive method to handle both the situations in Algorithm 6.

7.5.2 An Improved Approach

Notice that we did not pay attention to the quality of approximated arcs over TS in the previous section. While the connected arcs adhered to the G^1 continuity constraint at their endpoints, the overall smoothed TS may not necessarily be desirable. Thus here, we propose an improvement to the naïve approach, that takes into consideration the error measure err_{TS}.

Let us recap that $TS = \{p_1, \ldots, p_m\}$ is a segment of the original TRACE T, whose endpoints p_1 and p_m are marked as either `SharpEdge` or `Inflection`.

Algorithm 5: `NaïveSegmentation` (ActiveTrace $TS_j^{\mathrm{M}} \subseteq T^{\mathrm{M}}$)

```
Output: A set of elliptic arcs $K$ that approximates $TS_j^{\mathrm{M}} = \{p_1, \ldots, p_m\}$,
        and marks the $G^1$ continuity points identified in $TS_j^{\mathrm{M}}$.
1  Initialize $K \leftarrow \{\}$;
2  Set $\mathbf{P}_{\theta_A} \leftarrow p_1$, Set $flg \leftarrow 0$;
   // Iterate through all $m$ points in $TS_j^{\mathrm{M}}$.
3  for $i = 3$ to $m$ do
4      Set $\mathbf{P}_{\theta_B} \leftarrow p_i$;
5      Set $TS' \leftarrow \{\mathbf{P}_{\theta_A}, \ldots, \mathbf{P}_{\theta_B}\}$;
6      Set isArcValid $\leftarrow$ False;
7      $f(\mathbf{x}) \leftarrow$ Instantiate inellipse Adaptive-Midpoint($TS'$);
8      Let $ln_{\mathrm{AB}}$ be the baseline connecting $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$;
9      if $f(\mathbf{x})$ exists AND both $\mathbf{P}_C$ and $\mathbf{P}_{\theta_L}$ are on the same side of $ln_{\mathrm{AB}}$
       then Set isArcValid $\leftarrow$ True;
       // Attempt to stretch the arc.
10     if $flg = 0$ AND isArcValid = True then Set $flg \leftarrow 1$;
11     else if $flg = 1$ AND isArcValid = False then
12         Reset $\mathbf{P}_{\theta_B} \leftarrow p_{i-1}$;
13         Reset $TS' \leftarrow \{\mathbf{P}_{\theta_A}, \ldots, \mathbf{P}_{\theta_B}\}$;
14         $f(\mathbf{x}) \leftarrow$ Instantiate inellipse Adaptive-Midpoint($TS'$);
15         Add $K \leftarrow$ GetEllipticArc($f(\mathbf{x})$, $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$);  /* Algorithm 2 */
16         Mark $p_{i-1}$ As crucial $G^1$ continuity point;
17         Reset $\mathbf{P}_{\theta_A} \leftarrow p_{i-1}$, Reset $flg \leftarrow 0$;
   // Tidy up the last segment.  See Algorithm 6.
18 if $\mathbf{P}_{\theta_A} \neq p_m$ then Add $K \leftarrow$ HandleLooseSegment($TS'$);
19 return $K$;
```

The improved approach extends the main objective of the naïve approach after having found the point p_{j+1}. Where instead of immediately keeping p_j as the crucial G^1 continuity point, we review all the valid elliptic arcs found so far, and pick the one arc ω_{AB} whose error measure over the points it approximates is the smallest. Let p_k be the corresponding anchor-point $\mathbf{P}_{\theta_B}$ of ω_{AB} with the smallest error measure, where $k \leq j$. Then p_k is the crucial G^1 continuity point contributing to the elliptic segmentation of T. The process repeats again from p_k until all m points in TS are considered.

Terminating this iterative process at the last segment of TS is also slightly different from the naïve approach. If after having considered the elliptic arc at the final point p_m and found that the error measure is not the smallest among all the

Algorithm 6: HandleLooseSegment (ActiveTrace T^{M})

```
Output: A set of elliptic arcs that approximates $T^{\mathrm{M}} = \{p_1, \ldots, p_m\}$, and
        marks the $G^1$ continuity points identified internally in $T^{\mathrm{M}}$.
   // Case 1: Approximating entire $T^{\mathrm{M}}$ with a single arc.
1  $f(\mathbf{x}) \leftarrow$ Instantiate inellipse Adaptive-Midpoint($T^{\mathrm{M}}$);
2  if $f(\mathbf{x})$ exists then return GetEllipticArc($f(\mathbf{x})$, $p_1$, $p_m$);
   // Case 2: Attempt the Steiner inellipse.
3  $f(\mathbf{x}) \leftarrow$ Instantiate inellipse Steiner($T^{\mathrm{M}}$);
4  if $f(\mathbf{x})$ exists then return GetEllipticArc($f(\mathbf{x})$, $p_1$, $p_m$);
   // Case 3: Points aligned on a straight line.
5  if All $p_i \in T^{\mathrm{M}}$ has curvature hovering about zero then return
   LineSegment($p_1$, $p_m$);
   // Case 4: General.
6  Initialize $K \leftarrow \{\}$;
7  Set $p_k \leftarrow p_{m/2}$;
8  Mark $p_k$ As crucial $G^1$ continuity point;
9  Add $K \leftarrow$ HandleLooseSegment($\{p_1, \ldots, p_k\}$);
10 Add $K \leftarrow$ HandleLooseSegment($\{p_k, \ldots, p_m\}$);
11 return $K$;
```

valid arcs so far, then we still need to select the arc whose corresponding point p_k gave the smallest error measure, where $k < m$. Following which, we need to observe the points from p_k to p_m again, until we terminate the procedure precisely at p_m, or when the situation arises when no more valid arcs can be instantiated. In the latter case, we revert to the solutions we pointed out in the previous section.

We list the pseudo-codes of this improved approach in Algorithm 7.

From the resultant approximation in Figure 7.21, we get an overall improvement of 37.17% in the error measurement compared to the earlier approximation of the same example in Figure 7.19.

7.5.3 Speeding Up with a Guided Trajectory Approach

The two approaches that we have just seen are indeed computationally expensive methods, in terms of their execution times. Notice that while the runtimes are both proportionally linear in the number of points in TS, visiting each point actually instantiates an inellipse $f(\mathbf{x})$ (see Line 7 in Algorithm 5, and Line 8 in Algorithm 7). Solving the system of linear equations via the LU factorization method on a $k \times k$ matrix takes $O(2k^2)$ computations [12]. One can argue that

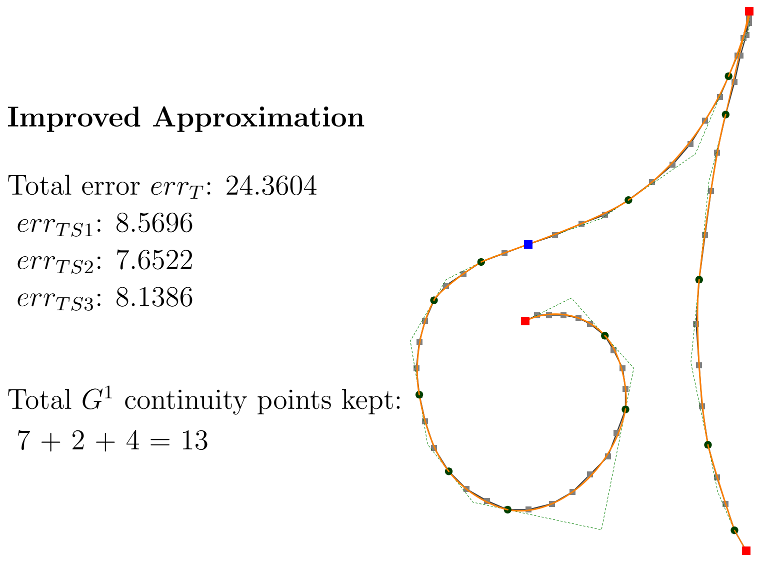


Figure 7.21: The improved approximation of the Trace T.

this time factor is negligible, since in our case, $k = 6$ is a small constant number.

However, we can still avoid performing many ‘unnecessary’ computations by exploiting the exigent information made available from the consequence of Proposition 3.12, and also from what we already know from the methods in the polyline simplification algorithms we discussed in chapter 5, particularly, the Douglas-Peucker (DP) algorithm [30]. In fact, we have already prepared the grounds for this *guided trajectory approach* in the `HandleLooseSegment()` routine in Algorithm 6, when attempting to approximate a set of elliptic arcs for the final segment of TS. We now show how we can cultivate this method and harvest it as a general *elliptic segmentation* algorithm.

Let $TS = \{p_1, \ldots, p_m\} \subseteq T$ be the ActiveTrace segment whose endpoints are marked as either `SharpEdge` or `Inflection`.

The objective here is to approximate the entire segment TS with a single elliptic arc, and then recursively refine it by breaking down the arc into several segments, if a certain error measure criteria is not met. We can split this arc because by Proposition 3.12, TS is solely composed of second-order Curves. And that in turn, can be decomposed into several, but shorter, well-connected second-order Curves. Furthermore, the solution after the refinement process is correct because the endpoints of each broken down Curve maintains its G^1 continuity with its neighbours, thus ensuring a confluent connection of arcs within the harmony of the whole Curve.

One can see that this approach is very similar to the DP algorithm, and it *is* mainly the `HandleLooseSegment()` routine from Algorithm 6. However, there are two additional cases that we need to consider: (i) the measure of error

Algorithm 7: `ImprovedSegmentation` (ActiveTrace $TS_j^{\mathrm{M}} \subseteq T^{\mathrm{M}}$)

```
Output: A set of elliptic arcs $K$ that approximates $TS_j^{\mathrm{M}} = \{p_1, \ldots, p_m\}$,
        and marks the $G^1$ continuity points identified in $TS_j^{\mathrm{M}}$.
1  Initialize $K \leftarrow \{\}$;
2  Initialize $p_s$, $\mathrm{err}_s$;
3  Set $\mathbf{P}_{\theta_A} \leftarrow p_1$, Set $flg \leftarrow 0$;
   // Iterate through all $m$ points in $TS_j^{\mathrm{M}}$.
4  for $i = 3$ to $m$ do
5      Set $\mathbf{P}_{\theta_B} \leftarrow p_i$;
6      Set $TS' \leftarrow \{\mathbf{P}_{\theta_A}, \ldots, \mathbf{P}_{\theta_B}\}$;
7      Set isArcVAlid $\leftarrow$ False;
8      $f(\mathbf{x}) \leftarrow$ Instantiate inellipse Adaptive-Midpoint($TS'$);
9      Let $ln_{\mathrm{AB}}$ be the baseline connecting $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$;
10     if $f(\mathbf{x})$ exists AND both $\mathbf{P}_C$ and $\mathbf{P}_{\theta_L}$ are on the same side of $ln_{\mathrm{AB}}$
       then
11         Set isArcVAlid $\leftarrow$ True;
12         err $\leftarrow$ measure error between $TS'$ and $f(\mathbf{x})$;
13         if err $<$ $\mathrm{err}_s$ then
14             $\mathrm{err}_s \leftarrow$ err;
15             $p_s \leftarrow p_i$;
       // Attempt to stretch the arc.
16     if $flg = 0$ AND isArcVAlid = True then Set $flg \leftarrow 1$;
17     else if $flg = 1$ AND isArcVAlid = False then
18         Reset $\mathbf{P}_{\theta_B} \leftarrow p_s$;
19         Reset $TS' \leftarrow \{\mathbf{P}_{\theta_A}, \ldots, \mathbf{P}_{\theta_B}\}$;
20         $f(\mathbf{x}) \leftarrow$ Instantiate inellipse Adaptive-Midpoint($TS'$);
21         Add $K \leftarrow$ GetEllipticArc($f(\mathbf{x})$, $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$); /* Algorithm 2 */
22         Mark $p_s$ As crucial $G^1$ continuity point;
23         Reset $\mathbf{P}_{\theta_A} \leftarrow p_s$;
24         Reset $flg \leftarrow 0$;
25         Reset $i \leftarrow s + 2$
   // Tidy up the last segment. See Algorithm 6.
26 if $\mathbf{P}_{\theta_A} \neq p_m$ then Add $K \leftarrow$ HandleLooseSegment($TS'$);
27 return $K$;
```

for refinement and the refinement process itself, and (ii) the handling of cyclic segments.

In the former case, it is imperative that we find the most opportunistic point to properly split TS that will work in our favour. That is, where previously we chose the middle point in TS to recursively call the refinement routine, we should now instead select any point $p_k \in TS$ that lies exactly on the current approximated elliptic arc ω_{AB}. In general, p_k is the point with the smallest error distance with respect to its corresponding point $p_{\theta k} \in \omega_{AB}$, that we explained in section 7.4. If several p_ks exist, then we pick one that lies relatively in the middle of ω_{AB}. This ensures that we keep all our approximated arcs as close to the original points in TS as possible.

The current `HandleLooseSegment()` routine assumes that TS is a non-cyclic segment; that is, the trajectory of points in TS does not complete a full $360°$-turn from the start to the end of the segment. Thus, in order to drop this assumption and address the latter case above, we need to make one more amendment to the method in Algorithm 6, to ensure that the trajectory of points in the segments we deal with are appropriate for this kind of elliptic arc segmentation.

Definition 7.20 *An* ActiveTrace *segment TS is* appropriate *for an elliptic arc approximation if all the points in TS fall into its own constraint triangle ΔABC.*

For example, the segment TS_1 of the Trace T in Figure 7.22 is a cyclic TraceSegment. If we join the tangent lines at the endpoints $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ of TS_1 to meet at $\mathbf{P}_C$, then the resulting constraint triangle ΔABC contains none of the intermediate points in TS_1. Hence, by Definition 7.20, TS_1 is not an *appropriate* segment for an elliptic arc segmentation.

This evidently becomes the pre-condition on the original TS before invoking the main procedure described above. As it already is, the breakdown step in the DP algorithm presents a smart solution for our situation here in making a TraceSegment appropriate. That is, we take the point $p_j \in TS$ that has the furthest perpendicular distance from the baseline $\overline{p_1 p_m}$, and use it to split TS into two sub-segments $\{p_1, \ldots, p_j\}$ and $\{p_j, \ldots, p_m\}$. We repeat this DP-step over the new sub-segments for as long as they are still not appropriate for an elliptic arc segmentation. As a consequence of this, all of the identified points p_j becomes the crucial G^1 continuity points contributing to the elliptic segmentation of T. The pseudo-codes in Algorithm 8 describes this step of making an ActiveTrace segment appropriate.

As an added advantage, we now have a chance to correct the stray gradients at those 'indeterministic' points in an *appropriate* ActiveTrace segment TS. The endpoints of TS are now sturdy enough for this correctional computation. However, we shall not abuse it for all the points in TS; suffice only to apply the correction to the identified crucial point p_k each time before invoking the recursive refinement step.

That is, for any intermediate point $p_k \in TS$, where TS is an *appropriate* Ac-

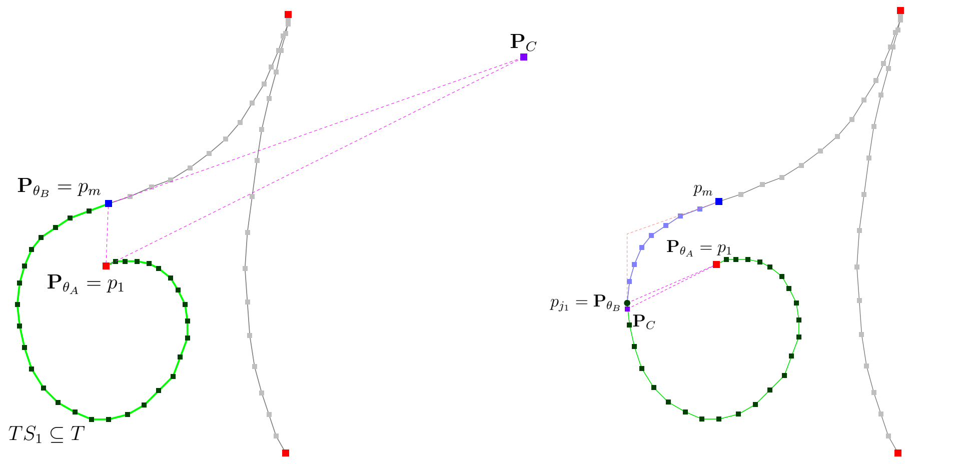

$\mathbf{P}_C$
$\mathbf{P}_{\theta_B} = p_m$
$\mathbf{P}_{\theta_A} = p_1$
$TS_1 \subseteq T$
p_m
$\mathbf{P}_{\theta_A} = p_1$
$p_{j_1} = \mathbf{P}_{\theta_B}$
$\mathbf{P}_C$

Figure 7.22: Making TS_1 *appropriate* for elliptic segmentation. **Step 1.** After constructing ΔABC, we find that none of the intermediate points in TS_1 lie inside ΔABC. After which, $t_{j1} \in TS_1$ is identified as the farthest point away from $\overline{p_1 p_m}$. The point p_{j1} splits TS_1 into two sub-segments $\{p_1, \ldots, p_{j1}\}$ and $\{p_{j1}, \ldots, p_m\}$. **Step 2.** The sub-segment $\{p_{j1}, \ldots, p_m\}$ is now appropriate, but the sub-segment $\{p_1, \ldots, p_{j1}\}$ is not, and the refinement process is repeated until the whole TS_1 is made up of appropriate sub-segments.

Algorithm 8: MakeSegmentAppropriate (ActiveTrace T^{M})

```
Output: A set of appropriate TraceSegments in $TSS$ from $T^{\mathrm{M}}$,
        following the outline stated in Definition 7.20.
1 Set $\Delta ABC \leftarrow$ compute ConstraintTriangle($T^{\mathrm{M}}$);
2 if All $p_i \in T^{\mathrm{M}}$ are contained in $\Delta ABC$ then return $\{T^{\mathrm{M}}\}$;
3 Initialize $TSS \leftarrow \{\}$;
4 Set $p_j \leftarrow$ point with the furthest perpendicular distance from $\overline{p_1 p_m}$;
5 Mark $p_j$ As crucial $G^1$ continuity point;
6 Add $TSS \leftarrow$ MakeSegmentAppropriate($\{p_1, \ldots, p_j\}$);
7 Add $TSS \leftarrow$ MakeSegmentAppropriate($\{p_j, \ldots, p_m\}$);
8 return $TSS$;
```

tiveTrace segment, we can determine a better estimate of the gradient of the elliptic arc at p_k, based on its position with respect to the two endpoints p_1 and p_m in TS.

Proposition 7.21 *Let* $\tan(\theta_A)$ *be the gradient of the line* ln_A *through the points* p_1 *and* p_k*, and let* $\tan(\theta_B)$ *be the gradient of the line* ln_B *through the points* p_k *and* p_m*. Then the angle of the gradient line of the curve passing through the three points* p_1*,* p_k*, and* p_m*, at* p_k*, is the halfway sum of the angles* θ_A *and* θ_B*.*

In other words, let $\tan(\theta_k)$ *be the gradient of the elliptic arc approximating the appropriate* ActiveTrace *segment* TS *at* p_k*, then*

$$\theta_k = \frac{1}{2}(\theta_A + \theta_B). \tag{7.21}$$

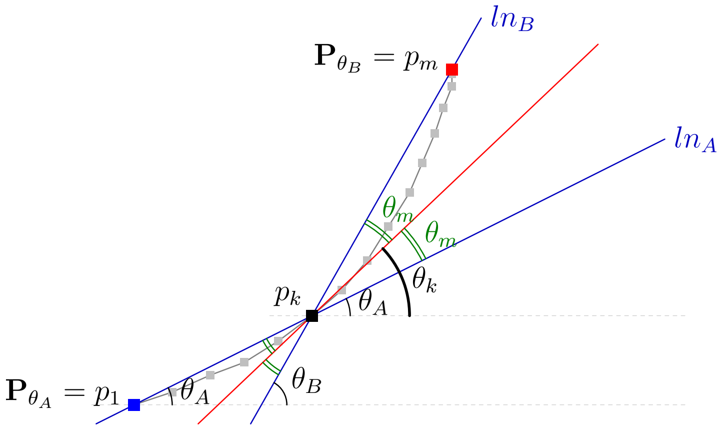


Figure 7.23: Getting a better estimate of the gradient of the curve at p_k.

Let us refer to Figure 7.23, which is based on the second segment $TS_2 \subseteq TS$ from our current examples, to explain this. Where the lines ln_A and ln_B cross at p_k, the opposite external angles affected by the lines are essentially the angular guides for the tangent of the curve at p_k. That is, the tangent line splits each of the external angles into two equal portions, as indicated by θ_m in the figure. From the reference point p_k, we get $2\theta_m = \theta_B - \theta_A$. It then follows that

$$\theta_k = \theta_A + \theta_m = \frac{1}{2}(\theta_A + \theta_B).$$

We see this as another way of viewing de Casteljau's geometric observations of the parametric curves [43], restricted to within the domain of second-order curves. We will see more proof of this when we discuss the Bézier curves in the next chapter.

Lemma 7.22 *Let the derivatives of the line* ln_A *and* ln_B *from Proposition 7.21 be expressed in terms of the points* $p_1 \equiv (x_1, y_1)$*,* $p_k \equiv (x_k, y_k)$*, and* $p_m \equiv (x_m, y_m)$*, so that*

$$\tan(\theta_A) = \frac{dy_A}{dx_A} = \frac{y_k - y_1}{x_k - x_1} \quad \textit{and} \quad \tan(\theta_B) = \frac{dy_B}{dx_B} = \frac{y_m - y_k}{x_m - x_k}.$$

Also, let $\alpha = dx_B dy_A + dx_A dy_B$ *and* $\beta = dx_A dx_B - dy_A dy_B$*. Then the tangent of the curve at* p_k *is*

$$\tan(\theta_k) = \frac{dy_k}{dx_k} = \frac{-\beta \pm \sqrt{\beta^2 + \alpha^2}}{\alpha}. \tag{7.22}$$

Proof. From Equation 7.21, we take the tangent on both sides of the equation to get the following equivalent expressions.

$$\begin{aligned} \tan(2\theta_k) &= \tan(\theta_A + \theta_B) \\ \frac{2\tan(\theta_k)}{1-\tan^2(\theta_k)} &= \frac{\tan(\theta_A)+\tan(\theta_B)}{1-\tan(\theta_A)\tan(\theta_B)} \\ \frac{2\tan(\theta_k)}{1-\tan^2(\theta_k)} &= \frac{dy_A/dx_A + dy_B/dx_B}{1-(dy_A/dx_A)(dy_B/dx_B)} \\ \frac{2\tan(\theta_k)}{1-\tan^2(\theta_k)} &= \frac{dx_B dy_A + dx_A dy_B}{dx_A dx_B - dy_A dy_B} = \frac{\alpha}{\beta} \end{aligned}$$

Rearranging this gives us the quadratic equation

$$\alpha \tan^2(\theta_k) + 2\beta \tan(\theta_k) - \alpha = 0.$$

And solving it gives us the estimated derivative of the expected curve passing through the three points p_1, p_k, and p_m, at p_k, where $\frac{dy_k}{dx_k} = \tan(\theta_k) = \frac{-\beta \pm \sqrt{\beta^2+\alpha^2}}{\alpha}$. ■

The two solutions returned from Equation 7.22 refer to (i) the tangent line described in Proposition 7.21, which does not intersect the line-segment $\overline{p_1 p_m}$, and (ii) another line that is perpendicular to the tangent line in (i). Naturally, we take the former as our solution for $\tan(\theta_k)$.

We state the full routine for this guided trajectory approach in Algorithm 9.

Algorithm 9: `GuidedTrajSegmentation` (ActiveTrace $TS_j^{\mathrm{M}} \subseteq T^{\mathrm{M}}$, ϵ)

Output: A set of elliptic arcs K that approximates $TS_j^{\mathrm{M}} = \{p_1, \ldots, p_m\}$, and marks the G^1 continuity points identified in TS_j^{M}.

```
1 Initialize $K \leftarrow \{\}$;
2 Set $TSS \leftarrow$ MakeSegmentAppropriate($TS_j^{\mathrm{M}}$);          /* Algorithm 8 */
3 foreach $TS_i$ in $TSS$ do
4     Add $K \leftarrow$ RefineFitSegmentation($TS_i$, $\epsilon$);      /* Algorithm 10 */
5 return $K$;
```

Clearly, after putting these all together, we can see that we do not need to compute all the ellipses for all the points in TS. In fact, we only need to compute at most two times the number of crucial points retained in TS. This makes the approach more favourable to us, in terms of its runtime, over the previous two methods we introduced earlier, since we do not need to compute the ellipses at

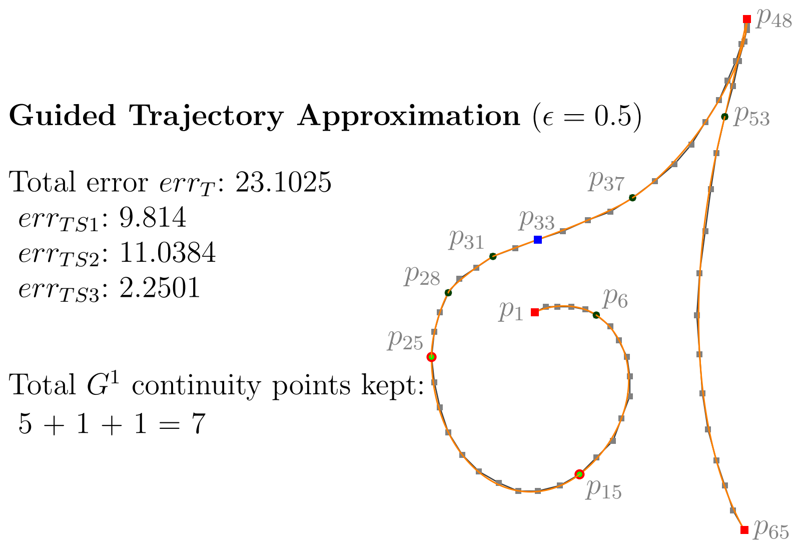


Figure 7.24: Approximating the TRACE T with the guided trajectory approach.

every single point. On the average, as was discussed in section 5.3, this divide-conquer-and-refine approach takes $O(n \log n)$ time to complete, where n is the number of points in TS.

Furthermore, the resultant quality of the approximation is fairly similar to the improved approach. The example we have been using to illustrate our approach, judging from the results in Figure 7.24, demonstrates this with a quality improvement of 41.12% over the naïve approach; from the previous quality improvement of 37.17% by the improved approach.

7.6 Results and Discussions

We present two more examples of our elliptic segmentation method to smooth the TRACES in Figure 7.25 and Figure 7.26, based on the guided trajectory approach. The former employs the Adaptive-Midpoint II in constructing the underlying elliptic arcs, while the latter is made up of Steiner inellipses. Both approximations were set at $\epsilon = 2.5$ pixels, per point.

We also ran another experiment to observe the percentage of the points kept for a given set of 2500 freehand written TRACES, each with a varying number of internal points. The data set were made to go through the elliptic segmentation process based on the Adaptive-Midpoint II, the Steiner, the Mandart, and the Brocard inellipses. We omitted the Orthic inconic as we pointed out earlier that there are cases where the method fails to return valid approximations. We also set the values of ϵ to range from 1.1 to 2.1 pixels per point, and we noted the number of points kept as each TRACE was put through the elliptic segmentation

Algorithm 10: `RefineFitSegmentation` (ActiveTrace T^{M}, ϵ)

```
   Output: A set of elliptic arcs $K$ that approximates $T^{\mathrm{M}} = \{p_1, \ldots, p_m\}$,
           and marks the $G^1$ continuity points identified in $T^{\mathrm{M}}$.
   // Straight line check.
 1 if All $p_i \in T^{\mathrm{M}}$ has curvature hovering about zero then return
   LineSegment ($p_1$, $p_m$);
   // Default approximation with the Adaptive-Midpoint
      inellipse.
 2 $f(\mathbf{x}) \leftarrow$ Instantiate inellipse Adaptive-Midpoint($T^{\mathrm{M}}$);
   // If default fails, revert to the Steiner inellipse.
 3 if $f(\mathbf{x})$ DOES NOT exist then $f(\mathbf{x}) \leftarrow$ Instantiate inellipse Steiner($T^{\mathrm{M}}$);
   // Now refine, if necessary.
 4 Initialize $K \leftarrow \{\}$;
 5 Initialize $p_k$, $err$;
 6 Set $\{p_k, err\} \leftarrow$ ComputeErr_SplitPt($T^{\mathrm{M}}$, $f(\mathbf{x})$);
 7 if $err \leq \epsilon$ then Add $K \leftarrow$ GetEllipticArc($f(\mathbf{x})$, $p_1$, $p_m$);
 8 else
 9     Mark $p_k$ As crucial $G^1$ continuity point;
10     Reset $\frac{dy_k}{dx_k} \leftarrow$ compute $\tan(\theta_k)$;          /* Proposition 7.21 */
11     Add $K \leftarrow$ RefineFitSegmentation($\{p_1, \ldots, p_k\}$, $\epsilon$);
12     Add $K \leftarrow$ RefineFitSegmentation($\{p_k, \ldots, p_m\}$, $\epsilon$);
13 return $K$;
```

Algorithm 11: `ComputeErr_SplitPt` (Trace T, Ellipse $f(\mathbf{x})$)

```
   Output: (i) The split point $p_k \in T$, and (ii) the total error measure $\mathrm{err}_T^2$.
 1 Initialize $p_k$, $\mathrm{err}_T^2 \leftarrow 0.0$;
 2 foreach $p_i$ in $T$ do
 3     Set $err_i \leftarrow$ compute error measure at $p_i$ using $f(\mathbf{x})$ as reference;
 4     Note $p_k \leftarrow$ largest per point error so far;
 5     $\mathrm{err}_T^2 += err_i$;
 6 return $\{p_k, \mathrm{err}_T^2\}$;
```

process, based on the guided trajectory approach. We present our findings in Table 7.1.

From the results in Table 7.1, the Adaptive-Midpoint II has a clear edge in keeping the number of points low, compared to the other three methods. We know it uses the advantage of the local segment information to guide the arcs,

Method	$\epsilon = 1.1$	$\epsilon = 1.6$	$\epsilon = 2.1$
Adaptive-Midpoint II	17.16%	11.92%	8.11%
Steiner Inellipse	25.92%	18.78%	17.02%
Mandart Inellipse	43.48%	35.87%	32.48%
Brocard Inellipse	45.02%	37.89%	37.02%

Table 7.1: Average percentage of points kept per Trace.

as opposed to having to guess the final two conditions in order to instantiate the Steiner, Mandart, and Brocard inellipses.

So while the three classical inellipses still produced desirable approximations with respect to the predefined values of ϵ, it required them to have more crucial points to successfully carry out the elliptic segmentation procedure. This inevitably leads to the methods requiring more time to produce the similar approximations as the Adaptive-Midpoint II.

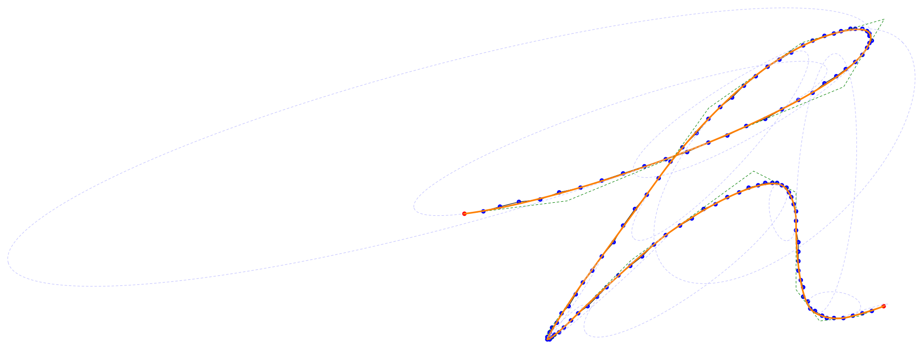


Figure 7.25: An example of an elliptic segmentation, highlighting the ellipses in the background that make up the elliptic arcs arranged in confluent positions.

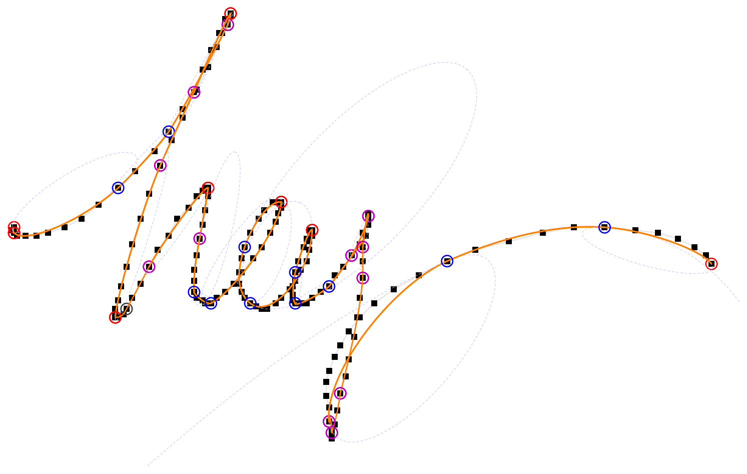


Figure 7.26: Another example of the elliptic segmentation.

CHAPTER 8

The Bézier Curve Approach

The last two chapters proved that we can indeed approximate a collection of handwritten TRACEs solely with a set of well-defined and well-positioned elliptic arcs, to produce desirably high quality curves in place of the pixelated TRACEs. We saw that the transitional connection made between the simplification of polylines and the passing through of proper smooth curves over the subjugated trajectory of indeterministic points, lies in correctly identifying only the *crucial* points in a TRACE. What was lacking, though, was the 'speed' factor. Computing ellipses are expensive. And while we made it a point to compute them only when it becomes necessary, we want to expedite this process of generating and finding desirable curves for approximating TRACEs even further.

In lieu of this, let us now dwell in another family of curves – the Bézier curves. Particularly, the second-order types, both rational and non-rational. In due course of the next two chapters, we will see that the behaviour of these curves can be made to assimilate the properties of the elliptic arcs that we uncovered, perform very minimal error checking, and still produce desirably high quality approximations for a collection of handwritten TRACEs.

Named after the automobile engineer Pierre Bézier, the curves were designed so that people in Pierre's profession would find it most understandable when using them in their everyday routines at the Renault car company in the early 1960s. To the engineers, a Bézier curve was considered in terms of its centre of mass with respect to a given set of point masses. This intuitively provided the means that make these descriptive curves applicable for chronicling free-form shapes, and the vast literature supports this notion with numerous examples, notably illustrating this property with cubic Bézier curves.

Here is where we draw the line to segregate the differences from the classical methodologies above, against our proposed techniques that deal exclusively with quadratic Bézier curves. The development of which paves the way towards the denouement of our accelerated *active-smoothing* concept in our discussions in the upcoming sections. Recall that we began our arguments in the earlier chapters by building our grounds based on second-order curves to efficiently solve the problem of smoothing handwritten TRACEs. We shall continue that trend here, and show that *rational* quadratic Bézier curves are as good as, if not better than, the cubic curves when applied in the domain of active smoothing. These appraisals are in terms of the overall speed of establishing the desirable curves, the quality of approximations that second-order Bézier curves produce, and, most importantly, the simplicity of the combined techniques.

8.1 De Casteljau's Recursive Midpoint Rule

A Bézier CURVE $K(t)$ is a parametric curve described over a closed interval $t \in [0, 1]$. That is, from a start-point $\mathbf{P}_{\theta_A}$ at $t = 0$, the CURVE takes its shape as it traverses over the permitted range of values of t, and then terminates completely at an end-point $\mathbf{P}_{\theta_B}$ at $t = 1$. The endpoints of the CURVE, $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$, are also known as the *anchor*-points. The number of points in $K(t)$ is determined by the resolution of the traversal-step over the values of t, and that the CURVE $K(t)$ is the CURVE given by our Definition 3.4.

Before we embark on our discussions on the Bézier CURVEs, it is imperative to note that this family of curves was first introduced by Paul de Casteljau [43]. It was through his fundamental observation of the 'subdivision' property that Pierre Bézier was able to parametrically describe the curve as above, which henceforth became famously known as the Bézier curve. This property refers to the 'recursive midpoint rule' that subdivides a single curve into two new ones at the half-way parametric mark. That is, we can divide $K(t)$ into two equal parametric domains of $[0, 0.5]$ and $[0.5, 1]$. Subsequently, each new curve can be further put through the recursive midpoint rule, thus subdividing it (and the original curve) into very fine granulations. In retrospect, the two new Bézier curves in the first subdivision, when considered together, are the equivalent of the whole $K(t)$ from which they were originally derived.

What should strike us here as important is that at every subdivision step, the point on the CURVE coinciding with the half-way parametric mark, is also the point that is uniquely related to the local geometric entity that defines the shape of the CURVE.

Let us explain this using an example of a proper second-order CURVE K. Let the point $\mathbf{P}_C$ be the intersection of the tangent lines through the anchor-points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ of K. Then the three points $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$, and $\mathbf{P}_C$ form the vertices of the local geometric entity that defines the shape of K. This is the same entity as the constraint-triangle ΔABC from our past discussions. For the convenience of our deliberations here, let us refer to the three points $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$, and $\mathbf{P}_C$ simply as A, B, and C.

Proposition 8.1 *Let ΔABC be any proper triangle on the plane. Then a second-order Bézier* Curve *K can be inscribed in ΔABC using the principles of de Casteljau's recursive midpoint rule, based on a two-fold geometry beginning on the edges $\overline{AC}$ and $\overline{CB}$, so that the resultant collection of midpoints m_i at each recursive step converges to the* Curve *K.*

A single step in de Casteljau's method applies a two-fold geometry to arrive at the one *true* point of the underlying quadratic Curve K. That is, the recursive rule begins with the three primary (guiding) points A, B, and C, and determines the first-order midpoints p and q, on the line-segments $\overline{AC}$ and $\overline{CB}$ respectively. Then, from the resultant line-segment $\overline{pq}$, we obtain the second-order midpoint m_0 as illustrated in Figure 8.1(a). The point m_0 lies on K and coincides with the half-way parametric mark at $t = 0.5$, and that $\overline{pq}$ is the tangent line to the underlying Curve K at m_0. Consequently, the midpoint rule splits the constraint-triangle ΔABC into two-halves at m_0, the left-half ΔApm_0 and the right-half Δm_0qB, in which de Casteljau's method is again applied in both sub-components to obtain m_1 and m_2 as shown in Figure 8.1(b). Correspondingly, the points m_1 and m_2 lie on the parametric marks of $t = 0.25$ and $t = 0.75$, and that $\overline{ij}$ and $\overline{kl}$ are tangents to the Curve K at m_1 and m_2, respectively.

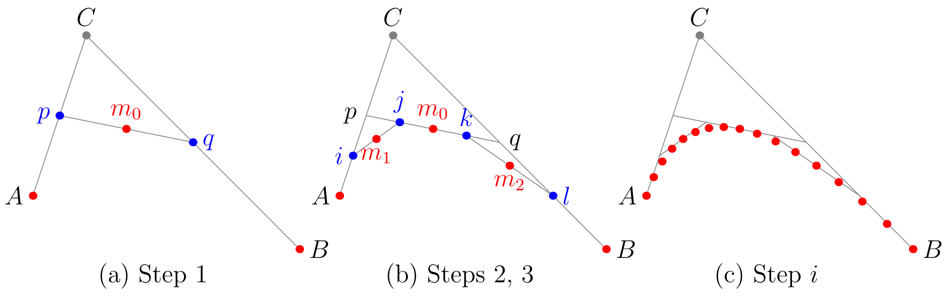


Figure 8.1: The recursive midpoint rule applied on points A, B, and C.

By recursively carrying out this binary-partitioning rule for infinitely many times over the sub-components derived at each recursive step, and then retaining all midpoints m_i at every subdivision, we get the smooth and true Bézier Curve K.

De Casteljau's recursive midpoint rule, while elegant in its approach, is computationally expensive and is not suitable for application domains requiring fast responses. In particular, pin-pointing m_i on the true curve and deriving its relationship with respect to the primary points A, B, and C, is a tedious process, especially if m_i sits deep down in the binary-partitioned step. This hassle of the recursive step, however, is overcome if we extend the tangent line at m_i until it cuts the edges $\overline{AC}$ and $\overline{CB}$ of ΔABC at p_i and q_i, respectively. Based strictly on the line-segment $\overline{p_iq_i}$ over m_i, we get its parametric mark t from the *distance-ratio* on three separate occasions as stated in Equation 8.1, for a second-order Bézier CURVE.

Parametric distance-ratio t ♠

$$t = \|\overline{Ap_i}\|/\|\overline{AC}\| = \|\overline{Cq_i}\|/\|\overline{CB}\| = \|\overline{pm_i}\|/\|\overline{p_iq_i}\| \tag{8.1}$$

Take Figure 8.1(b) for example, m_1 is the midpoint between i and j (which, respectively, are the midpoints between A and p, and p and m_0). Extending the line-segment $\overline{ij}$ until it cuts the original guides at $\overline{AC}$ and $\overline{CB}$ gives us $t = 0.25$ by Equation 8.1 as depicted in Figure 8.2(a). Figure 8.2(b) shows another example with $t = 0.83$.

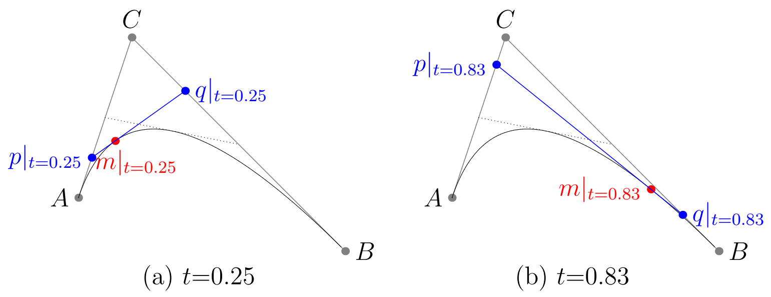


Figure 8.2: De Casteljau's recursive midpoint rule in direct relations to the parametric distance-ratio t.

In general, de Casteljau's recursive midpoint rule is applicable across all orders of the Bézier curve family where more than one extremum exists in $K(t)$. That is, for a Bézier CURVE of order n, a single step in de Casteljau's method would apply an n-fold geometry on its local geometric entity to get to a corresponding point m_i on $K(t)$. The reverse is also true when determining the exact parametric position t of a point $p \in K(t)$. Setting geometry aside, for the time being, we will see in the next section that the mathematical expression for any rational (and non-rational) Bézier CURVE comes as a direct consequence of de Casteljau's midpoint rule. Determining any point p is as trivial as evaluating the mathematical expression with t, and that determining the exact parametric position t of $p \in K(t)$ is the same as solving the expression for t.

8.2 Quadratic Bézier CURVES

A Bézier CURVE $K(t)$ of degree n is built up from the backbone *elevation* expression

$$1^n = [(1-t)+t]^n, \tag{8.2}$$

which controls the trajectory overture of the CURVE from one anchor-point $\mathbf{P}_{\theta_A}|_{t=0}$ to the other anchor-point $\mathbf{P}_{\theta_B}|_{t=1}$. The number of terms on the right-hand side of the expanded Equation 8.2 determines the number of *control*-points (minus the two anchor-points) required to describe $K(t)$.

For example, a third-order Bézier CURVE requires two control-points $\mathbf{P}_1$ and $\mathbf{P}_2$, in addition to the two anchor-points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$. These four points act as coefficient markers, so that the expanded elevation expression gives way to the cubic equation

$$K(t) = (1-t)^3\mathbf{P}_{\theta_A} + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_{\theta_B}. \tag{8.3}$$

One can see that this expression is analogous to the Bernstein polynomial of degree three. The anchor-points are treated as unique constants and are located at the extreme ends of the expanded equation, so that when $t = 0$, $K(t)$ returns only the $\mathbf{P}_{\theta_A}$ part of the expression; and likewise, when $t = 1$, $K(t)$ returns only the $\mathbf{P}_{\theta_B}$ part. As a consequence, the anchor-points sit on the CURVE $K(t)$ and the control-points do not. The latter serves only to guide the formation of $K(t)$ to within the convex hull of the closed polygon $\mathbf{P}_{\theta_A}$, $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_{\theta_B}$.

Definition 8.2 *A second-order Bézier* CURVE *$K(t)$ comprises one control-point $\mathbf{P}_C \equiv (x_C, y_C)$ and two anchor-points $\mathbf{P}_{\theta_A} \equiv (x_A, y_A)$ and $\mathbf{P}_{\theta_B} \equiv (x_B, y_B)$, which together makes up the constraint-triangle ΔABC. The* CURVE *is completely inscribed in ΔABC and is defined by its quadratic equation*

$$K(t) = (1-t)^2\mathbf{P}_{\theta_A} + 2t(1-t)\mathbf{P}_C + t^2\mathbf{P}_{\theta_B}. \tag{8.4}$$

Here, $KS(t)$ is equivalent to the point $(x(t), y(t))$, and we can decompose Equation 8.4 into its individual axis components

$$x(t) = (1-t)^2 x_A + 2t(1-t)x_C + t^2 x_B, \text{ and}$$
$$y(t) = (1-t)^2 y_A + 2t(1-t)y_C + t^2 y_B.$$

We can relate the validity of Equation 8.4 back to de Casteljau, using the notion of the parametric distance ratio t with respect to the edges $\overline{AC}$ and $\overline{CB}$.

Lemma 8.3 *Let $m \equiv (x_m, y_m)$ be a point on $KS(t)$. Then m is the product of de Casteljau's recursive midpoint rule by its relation to ΔABC, so that for the parametric position t that m is located on, $m|_t$ is exactly $(x(t), y(t))$.*

Proof. Let $p = (x_p, y_p)$ and $q = (x_q, y_q)$ be the two respective intersection points on the edges $\overline{AC}$ and $\overline{CB}$ of ΔABC, obtained by extending the tangent line at m. Then by Equation 8.1, the parametric position $t \in [0.0, 1.0]$ of m is the distance-ratio that ensures that p lies $t\|\overline{AC}\|$ distance away from A, and q lies $t\|\overline{CB}\|$ distance away from C, such that

$$x_p = t.(x_C - x_A) + x_A, \qquad y_p = t.(y_C - y_A) + y_A; \text{ and} \tag{8.5}$$
$$x_q = t.(x_B - x_C) + x_C, \qquad y_q = t.(y_B - y_C) + y_C. \tag{8.6}$$

The point m is the second-order midpoint of de Casteljau's rule, and it lies $t\|\overline{pq}\|$ distance away from p. Then

$$\begin{aligned}
x_m &= t.(x_q - x_p) + x_p \\
&= t.((t.(x_B - x_C) + x_C) - (t.(x_C - x_A) + x_A)) + x_p \\
&= x_A.(1 - 2t - t^2) + x_C.(2t - 2t^2) + x_B.t^2 \\
&= x_A.(1 - t)^2 + x_C.2t(1 - t) + x_B.t^2 \\
&= x(t)
\end{aligned}$$

Similarly, we obtain

$$\begin{aligned}
y_m &= t.(y_q - y_p) + y_p \\
&= t.((t.(y_B - y_C) + y_C) - (t.(y_C - y_A) + y_A)) + y_p \\
&= y_A.(1 - 2t - t^2) + y_C.(2t - 2t^2) + y_B.t^2 \\
&= y_A.(1 - t)^2 + y_C.2t(1 - t) + y_B.t^2 \\
&= y(t)
\end{aligned}$$

Thus, $m|_t \equiv (x_m, y_m) = (x(t), y(t))$. ■

Lemma 8.4 *The line $\overline{pq}$ with respect to the constraint-triangle ΔABC of a second-order Bézier* Curve *$K(t)$ is the tangent line to $K(t)$ at $m|_t$.*

Proof. From the individual axis components of $KS(t)$, we differentiate $x(t)$ and $y(t)$ with respect to t, so that

$$\begin{aligned}
\frac{dx(t)}{dt} &= 2((t-1)x_A + (1-2t)x_C + tx_B), \text{ and} \\
\frac{dy(t)}{dt} &= 2((t-1)y_A + (1-2t)y_C + ty_B).
\end{aligned}$$

This evaluates the tangent expression for the Curve $K(t)$ at the parametric position $t \in [0, 1]$ to

$$\frac{dy(t)}{dx(t)} = \frac{(t-1)y_A + (1-2t)y_C + ty_B}{(t-1)x_A + (1-2t)x_C + tx_B}.$$

The first midpoint m_0 identified by de Casteljau's rule occurs at $t = 0.5$ (see the example from Proposition 8.1). Substituting $t = 0.5$ into the tangent expression above gives

$$\frac{dy}{dx}|_{t=0.5} = \frac{y_B - y_A}{x_B - x_A},$$

which is exactly the gradient of the baseline $\overline{\mathbf{P}_{\theta_A}\mathbf{P}_{\theta_B}}$. And since $\overline{pq}$ is parallel to $\overline{\mathbf{P}_{\theta_A}\mathbf{P}_{\theta_B}}$, then the gradient of the line $\overline{pq}$ is the same as the gradient of $\overline{\mathbf{P}_{\theta_A}\mathbf{P}_{\theta_B}}$.

By Lemma 8.3, any point $m|_t \in K(t)$ is the product of de Casteljau's recursive midpoint rule by its relation to ΔABC. In other words, for any m lying on the CURVE $K(t)$, there exist a sub-triangle $\Delta ABC'$ where m is the first midpoint, so that the line $\overline{p'q'}$ related to $\Delta ABC'$ is the tangent line of $K(t)$ at m. The line $\overline{p'q'}$ is related to line $\overline{pq}$ at $m|_t$ by the notion of the distance ratio stated in Equation 8.1, so that $\overline{p'q'} \subset \overline{pq}$ at $m|_t$. Hence, it follows that the line $\overline{pq}$ is the tangent line to $K(t)$ at $m|_t$. ■

The Bézier CURVEs we have dealt with so far are the *non*-rational Bézier CURVEs. They are also sometimes known as *integral* Bézier CURVEs, and they have very rigid shape structures due to their formation that is based simply on the derivatives at the anchor-points. As was with our previous techniques, we also want to be able to tweak an arc and control it to our advantage here, based on some information about its environment. We can do this cleanly by adding weights to $K(t)$ and then rationalize the CURVE while keeping the tangents at the endpoints intact. This we explain in the next section.

8.3 Rational Quadratic Bézier CURVEs

A non-rational Bézier CURVE is made *rational* by scaling the elevation expression (Equation 8.2) of the control-points in $K(t)$ with values which we denote as weights w_i. For example, based on our earlier expression in Equation 8.3, a third-order rational Bézier CURVE $K(t)$ is given as

$$K(t) = \frac{(1-t)^3 w_0 \mathbf{P}_{\theta_A} + 3t(1-t)^2 w_1 \mathbf{P}_1 + 3t^2(1-t) w_2 \mathbf{P}_2 + t^3 w_3 \mathbf{P}_{\theta_B}}{(1-t)^3 w_0 + 3t(1-t)^2 w_1 + 3t^2(1-t) w_2 + t^3 w_3}. \quad (8.7)$$

As a rule of thumb, the elevation expression must always sum to one; otherwise the CURVE will change with the coordinate system. And so if any of the weights are altered, we need to normalize the blending of the elevation expression by dividing through their total value. Clearly, when all the weights are set to 1, we get the original non-rational cubic Bézier Equation 8.3.

One other way to view this, is that a rational Bézier CURVE is essentially the *superclass* of its non-rational version, in terms of an object-oriented concept. There are many other aspects of the rational Bézier CURVE discussed in the literature pertaining to its tangency and curvature, its reparameterization, as well as its relation to the Bernstein polynomial [43, 113, 120].

However, what concerns us most in our deliberations here is the manipulation of the weights w_i, particularly the weight w_1 related to control-point $\mathbf{P}_C$ of a rational quadratic Bézier CURVE. This directly contributes to our solutions in the active-smoothing of handwritten TRACEs, which we shall discuss in further details in the later part of this chapter. It is by getting the correct ratio in w_1 that we can perform an efficient segmentation and identify the crucial G^1 continuity points in a TRACESEGMENT TS. And it is through the by-product of this notion that error measure of an approximation can be reduced to a minimum.

Definition 8.5 *A second-order rational Bézier* CURVE $K(t)$ *comprises the three vertices of its constraint-triangle* ΔABC *and three weights* w_0*,* w_1*, and* w_2 *that correspond to the vertices* $\mathbf{P}_{\theta_A}$*,* $\mathbf{P}_C$*, and* $\mathbf{P}_{\theta_B}$*, respectively. The curve is completely inscribed in* ΔABC *and is defined by its quadratic equation*

$$K(t) = \frac{(1-t)^2 w_0 \mathbf{P}_{\theta_A} + 2t(1-t) w_1 \mathbf{P}_C + t^2 w_2 \mathbf{P}_{\theta_B}}{(1-t)^2 w_0 + 2t(1-t) w_1 + t^2 w_2}. \tag{8.8}$$

A second-order rational Bézier CURVE is sufficient to represent a conic. Let us refer to Equation 8.8 and set the weights $w_0 = 1$ and $w_2 = 1$. Then, the curve $K(t)$ is

- a hyperbola if $w_1 > 1$;
- a parabola if $w_1 = 1$; and
- an ellipse if $w_1 < 1$.

This behaviour of the rational curve provides us with a good reason to dwell solely in its quadratic domain. Furthermore, converting a second-order rational Bézier CURVE to its cubic non-rational equivalent, is easier than the other way around [120]. This ensures the portability of our techniques if it becomes necessary to export the final smoothed arcs as cubic Bézier CURVEs to support, for example, the generation of true-type fonts. The alternative of directly using cubic Bézier CURVEs in our application domain may not necessarily produce desirable approximation results during active-smoothing, and may at times fail altogether.

8.4 Constraint-Triangle ΔABC and Area Relations

There is a subtle, yet powerful relationship between the quadratic rational Bézier CURVE $K(t)$ and its constraint-triangle ΔABC. We point this out in terms of the area construed above the baseline of ΔABC and underneath the CURVE. For convenience, let us once again refer to the three points $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_B}$, and $\mathbf{P}_C$ that make up the vertices of ΔABC simply as A, B, and C.

Theorem 8.6 *Let* $\overline{AB}$ *be the baseline of the constraint-triangle* ΔABC*. Let* M *be the midpoint of* $\overline{AB}$*, and let* L *be the intersection point of the rational*

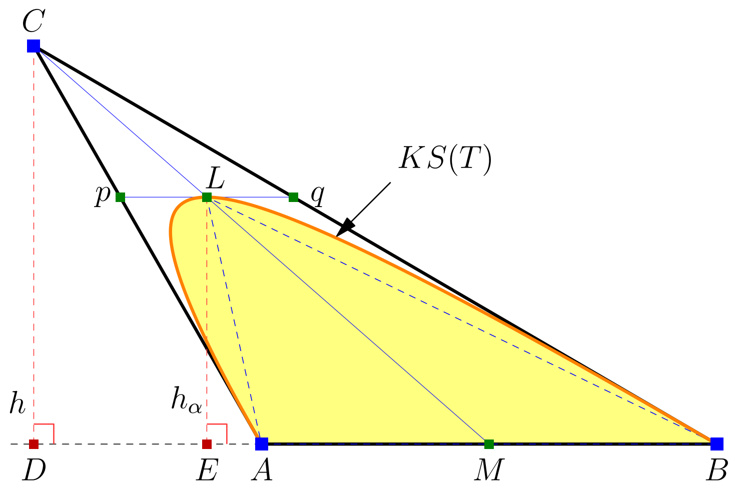


Figure 8.3: Area under the CURVE $K(t)$ inside the constraint-triangle ΔABC.

quadratic Bézier CURVE $K(t)$ *and the line-segment* $\overline{CM}$*, so that*

$$\alpha = \|ML\|, \text{ and } \beta = \|MC\|.$$

Also, let λ *be the area of the constraint-triangle* ΔABC*, and let* $f(\lambda)$ *be the area underneath* $KS(t)$ *inscribed in* ΔABC*, expressed as a function of the area of* ΔABC*. Then,*

$$f(\lambda) = \frac{\alpha\beta}{\beta^2 - \alpha\beta + \alpha^2}\,\lambda. \tag{8.9}$$

Proof. The area underneath the CURVE $K(t)$ completely covers the area of ΔABL and portions of the area in ΔALp and ΔBLq, where p and q are points on the edges $\overline{AC}$ and $\overline{CB}$, respectively, such that the line $\overline{pq}$ is parallel to $\overline{AB}$ and passes through the point L. These three areas can all be expressed in terms of λ, which is the area of the constraint-triangle ΔABC, where.

$$\lambda = \frac{1}{2}\|AB\|h.$$

Let $\overline{CD}$ be a line perpendicular to $\overline{AB}$, intersecting at the point D, so that $h = \|CD\|$. Similarly, let $\overline{LE}$ be another line perpendicular to $\overline{AB}$, intersecting at the point E, so that $h_\alpha = \|LE\|$. Then by similar triangles ΔMLE and ΔMCD,

$$\frac{h_\alpha}{h} = \frac{\alpha}{\beta}, \text{ so that } h_\alpha = \frac{\alpha}{\beta}h.$$

This leads to the expression of the area of ΔABL in terms λ.

$$\text{Area}(\Delta ABL) = \frac{1}{2}\|AB\|h_\alpha = \frac{\alpha}{\beta}\,\frac{1}{2}\|AB\|h = \frac{\alpha}{\beta}\lambda.$$

Also, since $\overline{CM}$ bisects $\overline{AB}$ at M, it bisects $\overline{pq}$ at L, so that $\|AM\| = \|MB\| = \frac{1}{2}\|AB\|$, and $\|pL\| = \|Lq\| = \frac{1}{2}\|pq\|$. By similar triangles ΔCpL and ΔCAM, we can express $\|pL\|$ in terms of $\|AB\|$ as follows:

$$\begin{aligned}\frac{\|pL\|}{\|AM\|} &= \frac{\|CL\|}{\|CM\|}\\ \frac{\|pL\|}{\frac{1}{2}\|AB\|} &= \frac{\beta-\alpha}{\beta}\\ \|pL\| &= \frac{\beta-\alpha}{2\beta}\|AB\|.\end{aligned}$$

Then the area of ΔALp, which is equal to the area of ΔBLq, can be expressed in terms of λ, as follows.

$$\begin{aligned}\text{Area}(\Delta ALp) &= \frac{1}{2}\|pL\|h_\alpha\\ &= \frac{1}{2}\ \frac{\beta-\alpha}{2\beta}\|AB\|\ \frac{\alpha}{\beta}h\\ &= \frac{\alpha(\beta-\alpha)}{2\beta^2}\ \frac{1}{2}\|AB\|h\\ &= \frac{1}{2}t\ \lambda, \text{ where } t = \frac{\alpha(\beta-\alpha)}{\beta^2} < 1.\end{aligned}$$

From de Casteljau's recursive midpoint rule, $\frac{\alpha}{\beta}$ is the ratio of the trajectory of the Curve inside its constraint-triangle ΔABC where the extremum point L lies. The midpoint rule splits the Curve at L into two portions, constrained by the sub-triangles ΔALp and ΔBLq, so that the extremum points in both triangles are also within the bounds of the split ratio $\frac{\alpha}{\beta}$, giving us the same situation of finding the area underneath $KS(t)$ with the original ΔABC. This allows for a recurrence relation in terms of the area of ΔABC with the area underneath $KS(t)$.

Then, letting $f(\lambda)$ be the area underneath $KS(t)$, expressed as a function of the area of ΔABC, we get:

$$\begin{aligned}f(\text{Area}(\Delta ABC)) &= \text{Area}(\Delta ABL) + f(\text{Area}(\Delta ALp)) + f(\text{Area}(\Delta BLq))\\ f(\lambda) &= \frac{\alpha}{\beta}\lambda + f(\frac{t}{2}\lambda) + f(\frac{t}{2}\lambda)\\ f(\lambda) &= \frac{\alpha}{\beta}\lambda + 2f(\frac{t}{2}\lambda).\end{aligned}$$

Expanding the terms,

$$\begin{aligned}
f(\lambda) &= \frac{\alpha}{\beta}\lambda + 2[\frac{\alpha}{\beta}(\frac{t}{2}\lambda) + 2f(\frac{t}{2}(\frac{t}{2}\lambda))] \\
f(\lambda) &= \frac{\alpha}{\beta}\lambda + 2[\frac{\alpha}{\beta}\frac{t}{2}\lambda + 2[\frac{\alpha}{\beta}(\frac{t}{2})^2\lambda + 2f((\frac{t}{2})^3\lambda)]] \\
&\dots \\
f(\lambda) &= \frac{\alpha}{\beta}\lambda + 2(\frac{t}{2})\frac{\alpha}{\beta}\lambda + 2^2(\frac{t}{2})^2\frac{\alpha}{\beta}\lambda + 2^3(\frac{t}{2})^3\frac{\alpha}{\beta}\lambda + \dots \\
f(\lambda) &= \frac{\alpha}{\beta}\lambda\ [1 + t^2 + t^3 + \dots].
\end{aligned}$$

Since $t < 1$, this series converges to

$$\begin{aligned}
f(\lambda) &= \frac{\alpha}{\beta}\ \frac{1}{1-t}\ \lambda \\
f(\lambda) &= \frac{\alpha\beta}{\beta^2 - \alpha\beta + \alpha^2}\ \lambda.
\end{aligned}$$

■

Corollary 8.7 *A non-rational quadratic Bézier* CURVE $K(t)$ *has the ratio of lengths* $\alpha : \beta = 1 : 2$. *Then, by Theorem 8.6, the area underneath the* CURVE $K(t)$ *bounded by its constraint-triangle* ΔABC *is* $\frac{2}{3}\lambda$.

The implication of Theorem 8.6 allows us to *directly* compute the area in constant time; that is, without even having to instantiate the CURVE. Normally, one would first need to assemble the CURVE and then digitally integrate it with, say, the Trapezoidal rule [12] over a number of resolution steps m. The finer the resolution, the more accurate the area becomes, but at a price of a larger number of steps.

In retrospect, we can determine λ, before computing the area under $KS(t)$, without having to figure out the height h of ΔABC. That is, given three non-collinear points (x_A, y_A), (x_B, y_B), and (x_C, y_C), the area of ΔABC is simply

$$\text{Area}(\Delta ABC) = \lambda = \frac{1}{2}|(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)|. \quad (8.10)$$

Corollary 8.8 *Let* $w_0 = w_2 = 1.0$. *Then the weight* w_1 *of the rational quadratic Bézier* CURVE $K(t)$ *stated in Definition 8.2 is the ratio of the lengths* $\overline{ML}$ *and* $\overline{LC}$, *so that in terms of* α *and* β,

$$w_1 = \frac{\|ML\|}{\|LC\|} = \frac{\alpha}{\beta - \alpha}. \quad (8.11)$$

Proof. Without affecting our discussions, since rational Bézier CURVEs are projective invariant, let us reposition the control-points $\mathbf{P}_{\theta_A}$ and $\mathbf{P}_{\theta_B}$ so that they lie on opposite sides of the x-axis with the midpoint M being the coordinate origin, so that $(x_A, y_A) \equiv (-1, 0)$, $(x_B, y_B) \equiv (1, 0)$, and $(x_M, y_M) \equiv (0, 0)$. This can be done by performing a simple Euclidean transformation on ΔABC (and subsequently, on $KS(t)$, but not necessarily); a translation followed by a rotation.

Evaluating $KS(t)$ at $t = 0.5$ returns $KS(0.5) = \frac{w_1}{1+w_1}\mathbf{P}_C$. Clearly, $KS(0.5)$ and L are the same points (from the proof of Lemma 8.4), and furthermore, the points L and $\mathbf{P}_C$ are on the same line through the coordinate origin M. Thus it follows that the length of $\overline{ML}$ is exactly $\frac{w_1}{1+w_1}$ times the length of $\overline{MC}$. That is,

$$\begin{aligned} \|ML\| &= \frac{w_1}{1+w_1}\ \|MC\| \\ \alpha &= \frac{w_1}{1+w_1}\ \beta. \end{aligned}$$

Rearranging this expression, we get

$$w_1 = \frac{\alpha}{\beta - \alpha}.$$

■

8.5 The Significance of the Weight w_1

The weight w_1, related to the control-point $\mathbf{P}_C$, plays a very significant role in controlling the behaviour of the desired rational Bézier CURVE that we are after. By setting the other two weights w_0 and w_2 to 1.0, we give ourselves freedom through w_1 to pursue the single extremum point in an *appropriate* TRACESEGMENT TS, which is directly affiliated to the CURVE $K(t)$. That is, we use the fact from Corollary 8.8 to deduce the point $L \in TS$ in which to obtain the smooth approximate of TS through $K(t)$. This rightly assumes that the point L is the extremum point in the underlying second-order curve as we pointed out earlier in Lemma 7.18.

An appropriate TRACESEGMENT TS has all its internal points $p_i \in TS$ inside the constraint-triangle ΔABC. Finding the point L, which equivalent notation is $\mathbf{P}_{\theta_L}$ in our standards to signify that the point also lies on the CURVE $K(t)$, given TS is exactly the same routine that we previously explained in Adaptation I in section 7.3, while discussing the adaptive midpoint inellipses:

Pick the point $p_i = \mathbf{P}_{\theta_L}$ that lies on $\overline{CM}$. If no such point exists, then pick the point $\mathbf{P}_{\theta_L}$ that is the intersection of the line-segment $\overline{p_i p_{i+1}}$ and the line $\overline{CM}$.

We then compute the weight w_1, and simply generate the approximated CURVE $K(t)$ for TS. We give three examples in Figure 8.4;

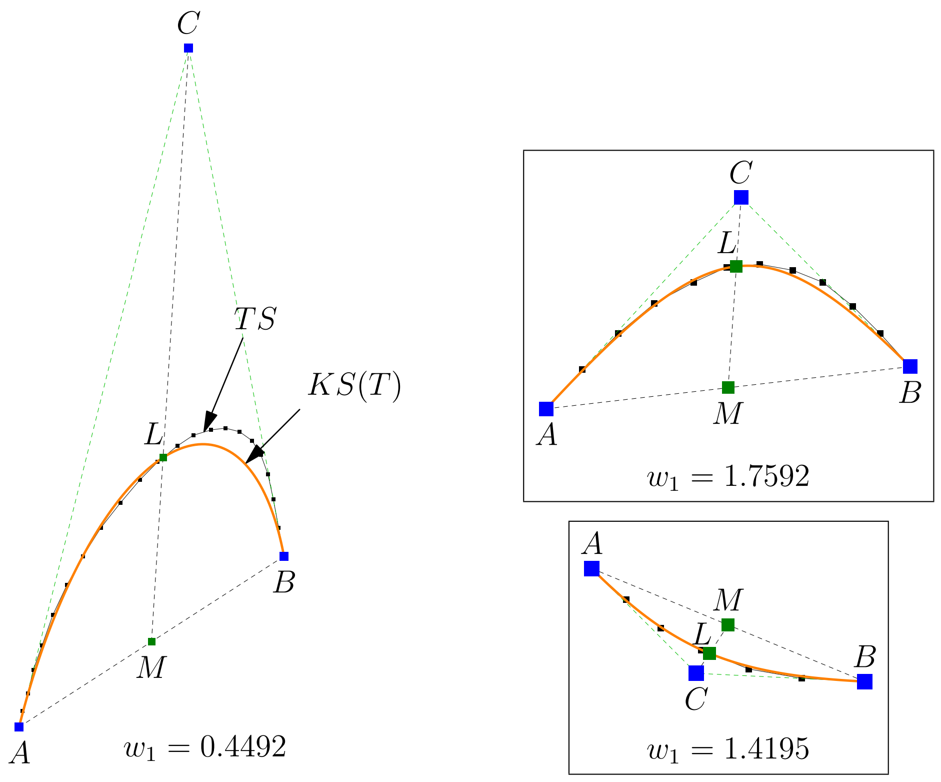


Figure 8.4: Determining w_1 from based on ΔABC and TS.

Of course, this estimate may not necessarily be desirable, as the only thing that we can guarantee from this exercise is that $K(t)$ will pass through the three points $\mathbf{P}_{\theta_A}$, $\mathbf{P}_{\theta_L}$, and $\mathbf{P}_{\theta_B}$. We also know from our previous deliberations that if the gradient at $L \in TS$ is parallel (or almost parallel) to the gradient of the baseline $\overline{AB}$, then chances are that the measure of error between $K(t)$ and all the points in TS would be minimal. Otherwise, the need to refine $K(t)$ arises if the measure of error is far from satisfactory.

Instead of performing the error measurement in the ways that we proclaimed earlier, which involves all the points in TS, we devised an alternative strategy that is even more efficient, by making use of the conotation implied above with the weight w_1. We will show in the next section that it is sufficient to determine the quality of the estimated Bézier Curve $K(t)$ by measuring only two other *significant* points, related to the constraint-triangle ΔABC, and adhering to de Casteljau's midpoint properties.

8.6 Selective Error Measurement

Unlike the ellipses, the Bézier CURVEs do not have an obvious centre reference from which to serve as a strong basis for error computation as we saw in section 7.4. But, it is guided and construed by the constraint-triangle ΔABC.

More importantly, the CURVE itself is a direct product of de Casteljau's recursive midpoint rule. As a consequence of this, we should also be able to use de Casteljau's recursive idea to *guide* a series of well-placed rational second-order Bézier CURVEs to desirably estimate an appropriate TRACESEGMENT TS. We do this by 'looking ahead' at two partition error-corridors ϵ_R and ϵ_S as indicated in Proposition 8.9.

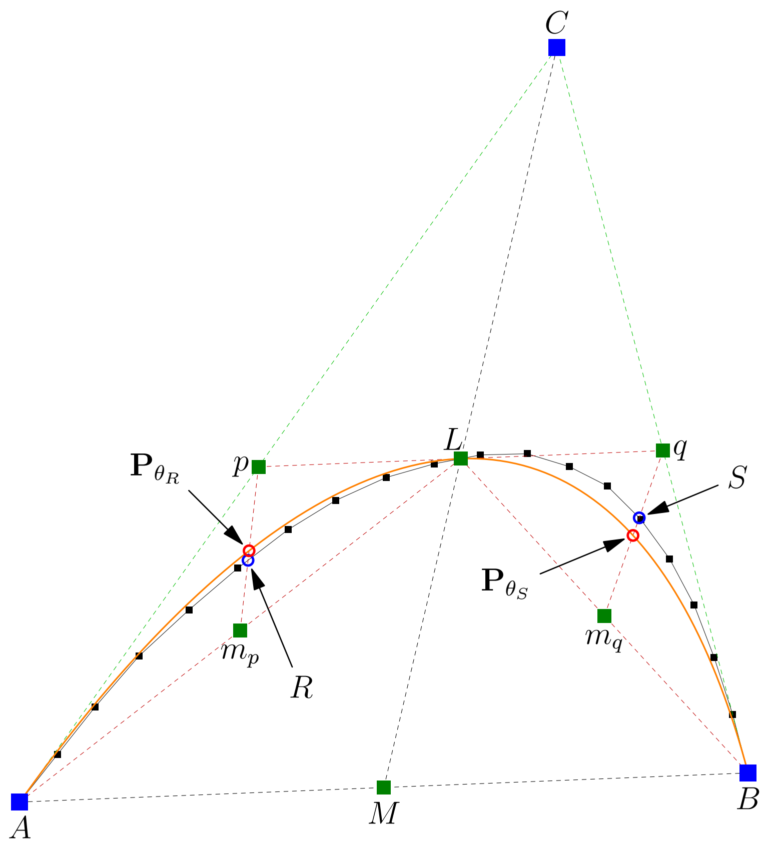


Figure 8.5: Measuring error at the partition corridors.

Proposition 8.9 *Let us refer to the constraint-triangle ΔABC of an appropriate* TRACESEGMENT *TS and all the marked points M, L, p, and q, from which are stated by Theorem 8.6 and its Corollary 8.8, and where $K(t)$ is the approximated* CURVE *for TS obtained from computing the weight w_1. Let m_p and m_q be the midpoints of $\overline{AL}$ and $\overline{LB}$, respectively. Also, let R be the intersection be-*

tween $\overline{pm_p}$ and TS, and let S be the intersection between $\overline{qm_q}$ and TS. Then, for $\mathbf{P}_{\theta_R} = K(0.25)$ and $\mathbf{P}_{\theta_S} = K(0.75)$, the partition error-corridor at $\mathbf{P}_{\theta_R}$ and $\mathbf{P}_{\theta_S}$ are $\epsilon_R = \|R\mathbf{P}_{\theta_R}\|$ and $\epsilon_S = \|S\mathbf{P}_{\theta_S}\|$, respectively, and that, for a predetermined tolerance corridor value ϵ, the Curve *$K(t)$ is a* desirable *approximation of TS if and only if both the following conditions are met:*

$$\epsilon_R \leq \epsilon, \textit{ and } \; \epsilon_S \leq \epsilon.$$

The proposition implies that the TraceSegment TS inside ΔABC is *convex*, and as such, TS can be sufficiently approximated by a set of well-positioned second-order Curves (from Proposition 3.12). Conventionally, after obtaining the first estimated Curve $K(t)$, we must inevitably test the fitness quality of $K(t)$ to TS by measuring the error-distance for every point $p_i \in TS$ with respect to the co-related point $p_{\theta_i} \in K(t)$. Then, based on this measured error, we decide whether or not to split $K(t)$ at the halfway mark, at $t = 0.5$, and repeat the procedure in the newly bisected left and right sub-segments until the computed error is satisfactorily below a certain tolerance value.

Thus it follows that in the worst-case scenario, determining the set of second-order curves to desirably estimate a TraceSegment TS of n points requires $O(n \log n)$ steps. Proposition 8.9, however, suggests a more expedient way to achieve the same results. This is by using the inherent properties that de Casteljau's recursive midpoint rule has over the second-order curve, as indicated by Lemma 8.3.

The points $\mathbf{P}_{\theta_R}$ and $\mathbf{P}_{\theta_S}$ are, in essence, the extremum points of the broken-down Curve inside the respective constraint-triangles ΔALp and ΔBLq. The recursive midpoint rule by de Casteljau dictates that R and S are equivalent to $\mathbf{P}_{\theta_R}$ and $\mathbf{P}_{\theta_S}$, respectively, so that both ϵ_R and ϵ_S are zero (or close to zero). If and only if both these conditions are true, then the rest of the points in TS, which are representative of the points in a second-order Curve $K(t)$, will follow suit in the trajectory alignment as we stated previously in Definition 3.4. In other words, if both ϵ_R and ϵ_S are within the bounds of the predetermined tolerance corridor value ϵ, then all other points in TS are also within the bounds of ϵ with respect to their corresponding points in $K(t)$.

In circumspect, the point $L \in TS$ (or the point $p_L \in TS$ closest to L) then becomes the most likely candidate for the crucial G^1 continuity point, in which to split TS into two sub-segments for further consideration when neither of the conditions are met. If we look at Proposition 8.9 from the viewpoint of the Douglas-Peucker (DP) algorithm, then we notice the similarities between the two methods. However, there is a considerable difference in the issues of efficiency between Proposition 8.9 and the DP algorithm.

Once again, as we pointed out in the previous section, we do not need to physically instantiate the Curve $K(t)$ in order to perform the error measurements. That is, once we have found the point L, we can immediately deduce

w_1 based on the constraint-triangle ΔABC, and obtain our expression for $K(t)$. Identifying the points R and S is a straight-forward geometrical computation on ΔABC and TS, and that the points $\mathbf{P}_{\theta_R}$ and $\mathbf{P}_{\theta_S}$ are the results of evaluating the CURVE's expression by substituting $t = 0.25$ and $t = 0.75$, respectively, into $K(t)$.

As part of the solution to the active-smoothing routine, we can show that identifying the point L does not require traversing $O(n)$ points in a TRACESEGMENT TS – in fact, we only need $O(1)$ time to accomplish this. Furthermore, the significance of the areal analysis in Theorem 8.6 is more pronounced in the preconditions to the active-smoothing routine. Both of which details we will elaborate in the next chapter. For now, we want to show that the solution presented in this chapter, where $KC = \{K_1(t), \ldots, K_m(t)\}$ is the smooth estimate for TS, is also a desirable approximation of TS within the bounds of ϵ.

8.7 Bézier Segmentation

The Bézier segmentation of an appropriate TRACESEGMENT TS splits TS into sub-segments of appropriate TRACESEGMENTS TS_i, where each $TS_i \subseteq TS$ has a corresponding *desirable* approximation of a rational quadratic Bézier CURVE $K_i(t) \subseteq KC$ over it, so that the COMPOSITECURVE KC fully estimates TS in its entirety. Given a tolerance error-bound value ϵ, the COMPOSITECURVE KC is a desirable estimate of TS, if and only if every point in $p_i \in TS$ is within ϵ distance of its corresponding point $\mathbf{P}_{\theta_i} \in K$.

Consequently, this process of Bézier segmentation is the natural follow-up to section 8.6. That is, we simply put the appropriate TRACESEGMENT $TS = \{p_1, \ldots, p_n\}$ and its constraint-triangle ΔABC as inputs to Proposition 8.9. Then for a predetermined value of ϵ, we compare the computed error measurements for ϵ_R and ϵ_S. If either ϵ_R or ϵ_S is larger than ϵ, we then split TS at the point L (or the point $p_L \in TS$ closest to L) and recursively apply Proposition 8.9 on the sub-segments $TS_1 = \{p_1, \ldots, p_L\}$ and $TS_2 = \{p_L, \ldots, p_n\}$. The point $p_L \in TS$ is marked as a crucial G^1 continuity point.

This recursive midpoint splitting of the Bézier segmentation terminates when TS_i is relatively flat, or when a desirable rational quadratic Bézier curve fulfils the two conditions in Proposition 8.9. We give the pseudo-codes of the Bézier segmentation in Algorithm 12.

To graphically illustrate the Bézier segmentation of a proper TRACESEGMENT TS, we highlight the recursive steps through Figure 8.6 to Figure 8.11. In **Round 0**, the measures of $\|R\mathbf{P}_{\theta_R}\|$ and $\|S\mathbf{P}_{\theta_S}\|$ are both greater than ϵ, which means the segment fails the two conditions in Proposition 8.9. So TS must then be split at the identified point L, leading to **Round 1**. There, the underlying curve between A and B is a desirable approximation, and so no further actions

Algorithm 12: `BézierSegmentation` (ACTIVETRACE $TS_j^{\mathrm{M}} \subseteq T^{\mathrm{M}}$, ϵ)

```
Output: A set of well-placed, desirable rational quadratic Bézier CURVEs
        in $KC$ that approximates $TS_j^{\mathrm{M}} = \{p_1, \ldots, p_n\}$, and marks the
        $G^1$ continuity points identified in $TS_j^{\mathrm{M}}$.
1  Initialize $K \leftarrow \{\}$;
2  Set $TSS \leftarrow$ MakeSegmentAppropriate($TS_j^{\mathrm{M}}$);          /* Algorithm 8 */
3  foreach $TS_i$ in $TSS$ do
4      Add $K \leftarrow$ RefineBezSegment($TS_i$, $\epsilon$);           /* Algorithm 13 */
5  return $K$;
```

Algorithm 13: `RefineBezSegment`(<*Appropriate*>ACTIVETRACE T^{M}, ϵ)

```
Output: A set of desirable rational quadratic Bézier CURVEs in $KC$ that
        approximates $T^{\mathrm{M}} = \{p_1, \ldots, p_n\}$, and marks the $G^1$ continuity
        points identified in $T^{\mathrm{M}}$.
1  Initialize $K \leftarrow \{\}$;
2  Initialize $\Delta ABC$ from $T^{\mathrm{M}}$;
3  Determine $p_L, p, q, m_p, m_q$ from $\Delta ABC$ and $T^{\mathrm{M}}$;        /* Theorem 8.6 */
4  Set $\alpha, \beta \leftarrow$ Length $\overline{ML}$, Length $\overline{MC}$;
5  Set $R, S \leftarrow$ Intersection($\overline{pm_p}$, $T^{\mathrm{M}}$), Intersection($\overline{qm_q}$, $T^{\mathrm{M}}$);
6  Set $(w_0, w_1, w_2) \leftarrow (1.0, \frac{\alpha}{\beta-\alpha}, 1.0)$;
7  Set $\mathbf{P}_{\theta_R}, \mathbf{P}_{\theta_S} \leftarrow KS(0.25), KS(0.75)$;
8  Set $\epsilon_R, \epsilon_S \leftarrow$ Length $\overline{R\mathbf{P}_{\theta_R}}$, Length $\overline{S\mathbf{P}_{\theta_S}}$;
9  if $\epsilon_R > \epsilon$ or $\epsilon_S > \epsilon$ then
10     Mark $p_L$ As crucial $G^1$ continuity point;
11     Reset $\frac{dy_L}{dx_L} \leftarrow$ compute $\tan(\theta_L)$;          /* Proposition 7.21 */
12     Add $K \leftarrow$ RefineBezSegment($\{p_1, \ldots, p_L\}$, $\epsilon$);
13     Add $K \leftarrow$ RefineBezSegment($\{p_L, \ldots, p_m\}$, $\epsilon$);
14 else Add $K \leftarrow$ GetQuadCurve($KS(t)$);
15 return $K$;
```

need to be taken. In **Round 3**, the sub-segment between A and B fails the ϵ condition, and the sub-segment would need to be split at L. The recursive steps are carried out in **Rounds 4** and **5**, until we obtain the COMPOSITECURVE that desirably approximates TS, with an average error measure of 0.66601 pixels per point.

Round 0:

$w_1 = 0.4492$, $\epsilon = 1.24$

$\|R\mathbf{P}_{\theta_R}\| = 7.4275$, $\|S\mathbf{P}_{\theta_S}\| = 15.0867$

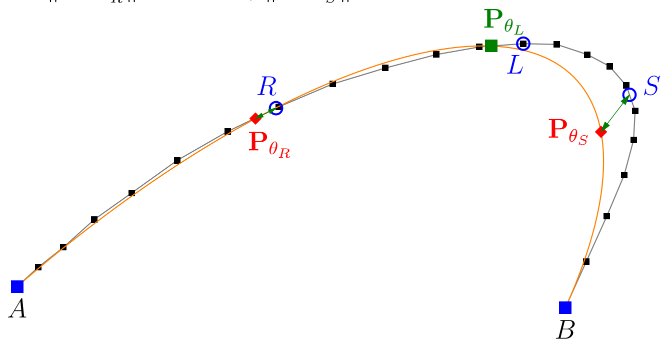


Figure 8.6: Smoothing an appropriate TRACESEGMENT TS via Bézier segmentation.

Round 1:

$w_1 = 1.0557$, $\epsilon = 1.24$

$\|R\mathbf{P}_{\theta_R}\| = 0.9088$, $\|S\mathbf{P}_{\theta_S}\| = 0.7448$

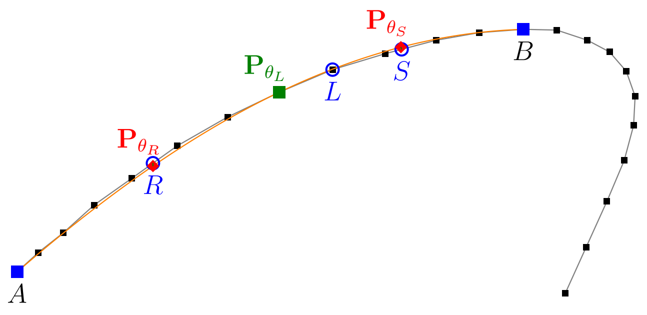


Figure 8.7: Smoothing an appropriate TRACESEGMENT TS via Bézier segmentation.

Round 2:

$w_1 = 1.4628$, $\epsilon = 1.24$

$\|R\mathbf{P}_{\theta_R}\| = 2.9141$, $\|S\mathbf{P}_{\theta_S}\| = 3.4844$

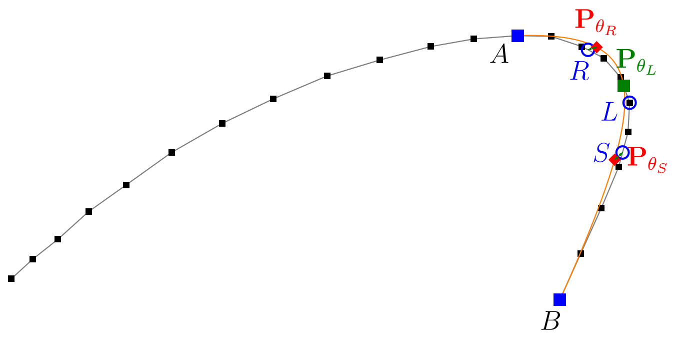


Figure 8.8: Smoothing an appropriate TRACESEGMENT TS via Bézier segmentation.

Round 3:

$w_1 = 0.7821$, $\epsilon = 1.24$

$\|R\mathbf{P}_{\theta_R}\| = 0.4283$, $\|S\mathbf{P}_{\theta_S}\| = 0.2919$

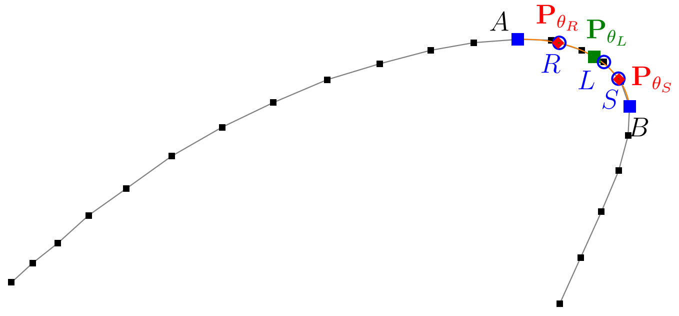


Figure 8.9: Smoothing an appropriate TRACESEGMENT TS via Bézier segmentation.

Round 4:

$w_1 = 2.1071$, $\epsilon = 1.24$

$\|R\mathbf{P}_{\theta_R}\| = 0.6589$, $\|S\mathbf{P}_{\theta_S}\| = 0.9635$

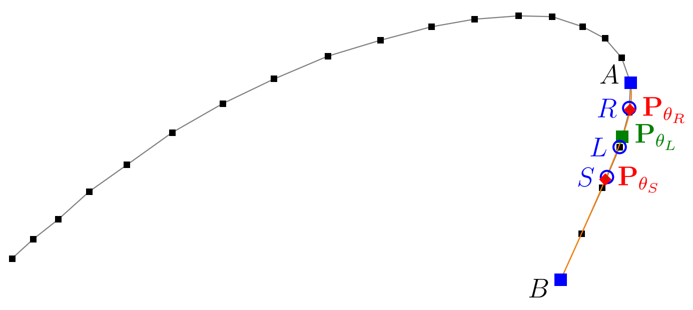


Figure 8.10: Smoothing an appropriate TRACESEGMENT TS via Bézier segmentation.

Bézier Segmentation Completed.

Average error per point: 0.66601 (pixels) $< \epsilon = 1.24$

Total points kept: 4 (of 23)

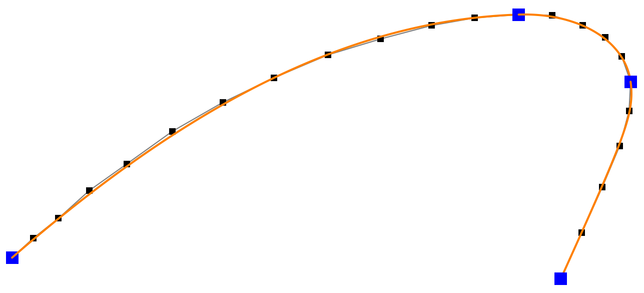


Figure 8.11: Smoothing an appropriate TRACESEGMENT TS via Bézier segmentation.

8.8 Climate Mapping with Smooth Contours

To review the reliability, quality, and accuracy of our Bézier segmentation method, we applied our technique to the temperature maps, as part of our ongoing collaboration with the Meteorologisches Institut der Universität Freiburg.

The challenge was to desirably approximate a set of indeterministic input points of similar temperature readings scattered over a certain geographical area with smooth contours. This would, among other things, establish the climatic trend of the represented region. For example, we want to connect all areas on the map that have a reading of 18°C. But because of the irregular locations of the sensors mounted all over the prescribed areas, we cannot get a proper smooth contour that is legible enough for further evaluations. Clearly, this problem setting is similar to ours, accept that the input points are not time-ordered.

Our solution was to first order these points so that they can be used in our Bézier segmentation method, and we presented this through the work of Matuschek [73]. It involves several filter techniques that ensure data integrity before the points can be made to follow a certain orderly manner. Once this has been established, the points are fed into the Bézier segmentation method. There we saw that it proved to be an efficient method that generated temperature contours that are fit for climatic maps. We present the results in Figure 8.12 (before smoothing) and Figure 8.13 (after smoothing).

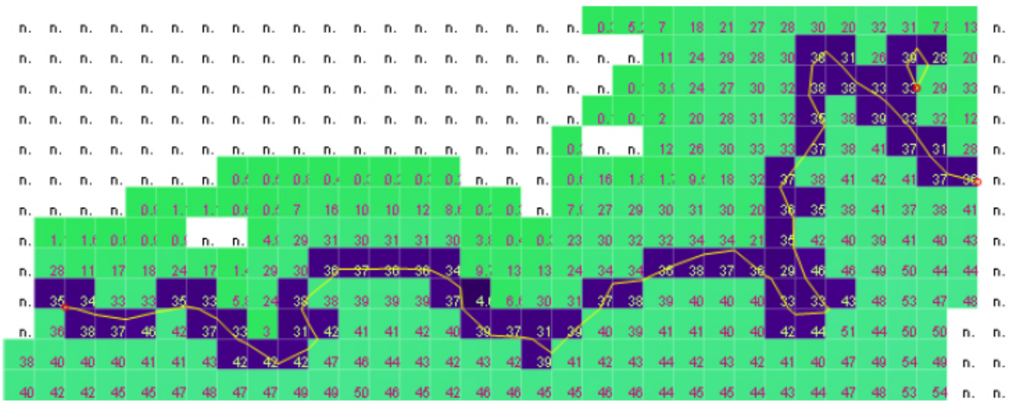


Figure 8.12: Unsmoothed temperature contours.

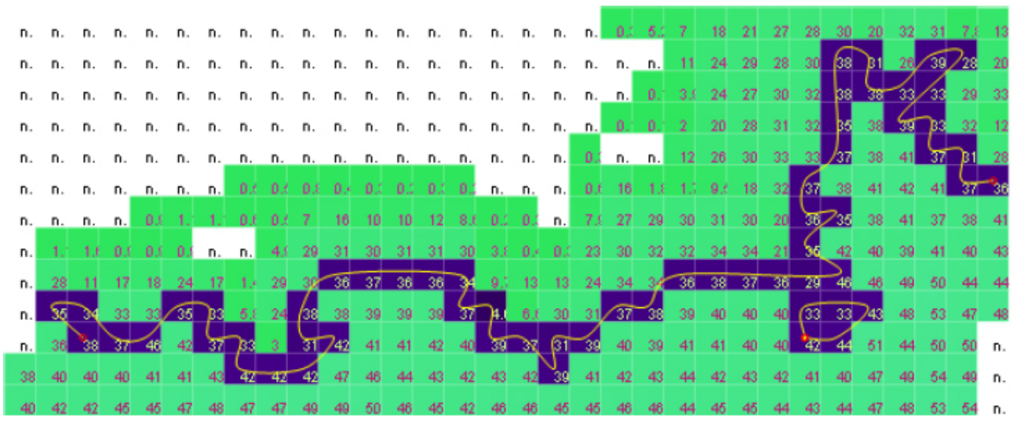


Figure 8.13: Smoothed temperature contours.

CHAPTER 9

The Active-Smoothing Solution

On route to this chapter, we have met and thoroughly discussed several classical routines for handling discrete curves over a set of indeterministic points, and, upon deeper ruminations, found that they can be made even more efficient. These increased efficiencies then become the very reasons we adduce for coming up with the concept of *active-smoothing* – turning the 'passive' process of approximating desirable Curves on input handwritten Traces 'active'. When the routines are properly assimilated and established in a supporting application program, one can inevitably expect to immediately see his/her digital handwritings as confluent curves even before the pen is lifted off the tablet screen. One's style and flair in the written portrayals are also maintained, but not exaggerated.

Active-smoothing here is the culmination of combining a shrewd strategy with the performance of efficacious computations to produce output renditions of high-quality smooth Curves that do not deplete at any resolutions, *while data is still being received.* As we progress through the chapter, we will see that the process of active-smoothing relies significantly on making coordinated judgements on local information that leads to the economical savings over many expensive (and extensive) calculations.

Looking back to section 3.5, we have, in fact, already introduced the main routine of the active-smoothing solution; through the 'active-sampling' routine in Algorithm 1. What 'active-sampling' essentially does is to categorise the *type* of a new incoming point, as it is being sampled, as either `Normal`, `Inflection`, or `SharpEdge`. The sections in this chapter are the fundamental expansions to this 'active-sampling' procedure, each with a clear and contumacious objective, drawing its concepts from the past chapters in their enhanced efficiency contributions.

9.1 Maintaining *Appropriate* Segments

We saw that it was only through an *appropriate* TraceSegment that a proper approximation can be made for it with a set of CurveSegments without losing desirable points. Recall from Definition 7.20 that a TraceSegment TS is considered appropriate if and only if all the points $p_i \in TS$ lie inside the constraint-triangle ΔABC. We showed the 'passive' routine to make TS appropriate in Algorithm 8, which takes an average of $O(n \log n)$ time to run, given a TraceSegment containing n points.

Clearly, this runtime is not feasible for the 'active' routine.

We propose a faster solution, which takes into account the main properties of the constraint-triangle ΔABC and reacts immediately at every instance of a new incoming point.

Lemma 9.1 *Let $p_k \in TS$ be the latest `ActivePoint` to be classified by the 'active-sampling' routine as either `Normal`, `Inflection`, or `SharpEdge`. Let $p_j \in TS$ be the last crucial point identified by the active-smoothing routine, where $j < k$, and that p_j is a point with either a G^0, G^1, or G^2 continuity joint and is not `Normal`. Then the TraceSegment $TS' = \{p_j, \ldots, p_{k-1}\} \subseteq TS$ is appropriate.*

Proof. This implies that the points $p_{j+1}, \ldots, p_{k-1}$ are all `Normal`. Suppose that TS' is not appropriate. Then there exists a point $p_i \in TS'$ that lies outside the constraint-triangle ΔABC, formed by the intersection of the gradient lines at p_j and p_{k-1}. There are two cases to consider:

Firstly, without loss of generality, let us assume that p_{k-1} lies to the right of the tangent line through p_j. We can reflect this to argue the converse. Then here, we can safely assume that all the intermediate points between p_j and p_{k-1} also lie to the right of the tangent line through p_j. Furthermore, all these intermediate points lie to the left of the tangent line through p_{k-1}, except p_i, which lies outside ΔABC. If this is the case, then the only time that this can happen is when there was a trajectory change at p_{i+1}, which would have caused the 'active-sampling' routine to report p_{i+1} as an `Inflection`, or a `SharpEdge` point, where $j < i < i+1 < k-1$. In consequence, p_{i+1} would have been the last crucial point identified instead of p_j.

Secondly, the point p_i lies on the opposite side of $\overline{AB}$ with respect to the vertex $\mathbf{P}_C$. In such a case, all the intermediate points between p_j and p_{k-1} are outside ΔABC – particularly on the opposite side of $\overline{AB}$ with respect to $\mathbf{P}_C$. This happens when the trajectory of TS' has made a complete 180° turn at p_{k-1} from the trajectory at p_j, and that the point p_{k-1} would have been marked as a crucial G^1 continuity point by the active-smoothing routine. In consequence, p_{k-1} would have been the last crucial point identified instead of p_j.

Either of these cases contradicts the assumption, and hence the TraceSegment TS' is appropriate. ■

Thus, by Lemma 9.1, it is sufficient that we check only the previous point p_{k-1} and the reference $\mathbf{P}_C$ to ensure that the current TraceSegment TS' that we are actively monitoring is appropriate. This requires that we maintain updated references for the tangent line through p_k and the baseline $\overline{AB} = \overline{p_j p_k}$ of ΔABC, for every newly sampled point p_k that was added and classified as `Normal` by the 'active-sampling' routine.

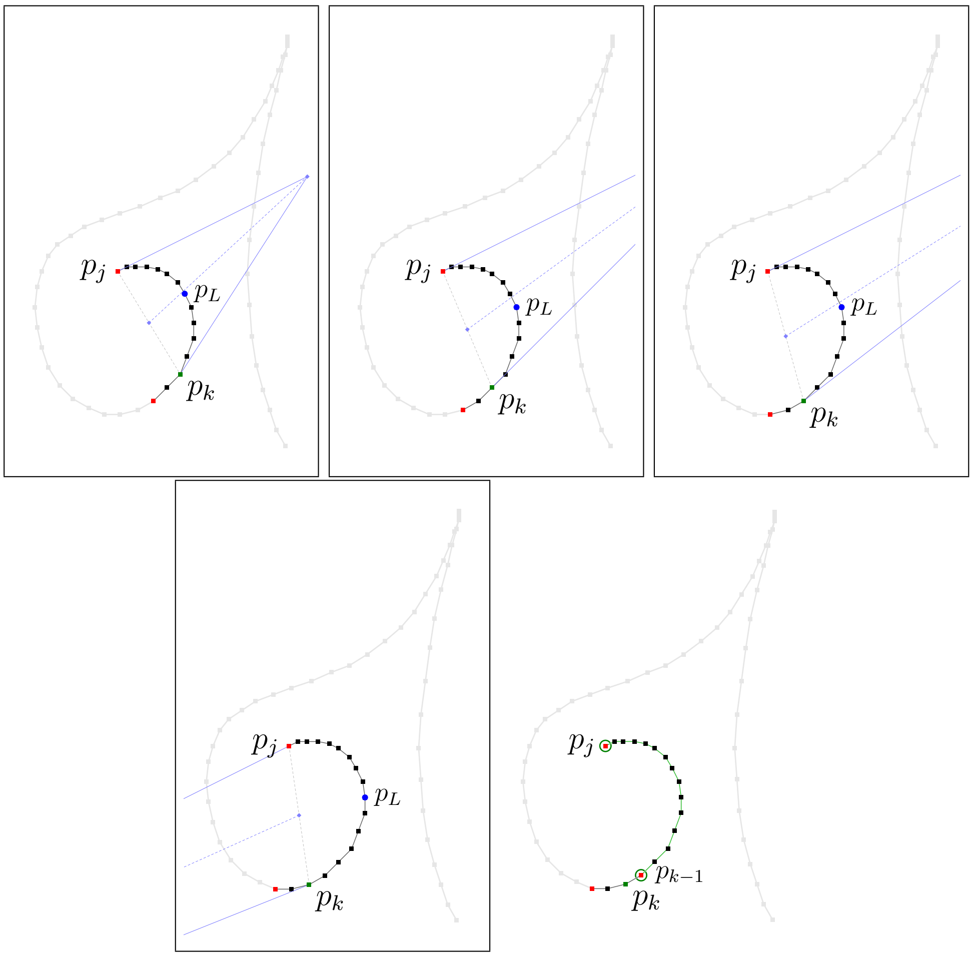


Figure 9.1: Active Bézier segmentation, while maintaining farthest point reference. Top three figures: A new fully instantiated `ActivePoint` p_k, makes up the TraceSegment $\{p_j, \ldots, p_k\}$, which is an appropriate segment. Bottom left: The new point p_k makes the segment $\{p_j, \ldots, p_k\}$ not appropriate. Bottom right: By Lemma 9.1, the segment $\{p_j, \ldots, p_{k-1}\}$ is an appropriate segment, so that it is now ready for the Bézier segmentation routine.

The repercussions of this active routine is that, if TS' becomes *in*appropriate at p_k, then the active-smoothing solution can consider and mark p_{k-1} as a crucial G^1 continuity point. Subsequently, we can begin the next step to approximate a CompositeCurve K for the independent *appropriate* TraceSegment $TS' = \{p_j, \ldots, p_{k-1}\}$.

Corollary 9.2 *The appropriate TraceSegment* $TS' = \{p_j, \ldots, p_{k-1}\} \subseteq TS$ *is composed of second-order curves.*

Proof. This is an immediate consequence of Proposition 3.12. ■

9.2 Maintaining Farthest Point Reference

For reasons of efficiency, we accord the active-smoothing solution with the Bézier segmentation technique that we described earlier in section 8.7. Of which, the technique's selective error measurement places a deep emphasis on the point $\mathbf{P}_{\theta_L}$ on the TraceSegment TS in order to compute the weight w_1 to use in the rational quadratic Bézier expression $KS(t)$. The point $\mathbf{P}_{\theta_L}$ is an extremum point of TS, and it is the farthest point away from the baseline of the constraint-triangle ΔABC. It is, as indicated in Proposition 8.9, the intersection of the bisector line $\overline{CM}$ with TS, where M is the midpoint of the baseline $\overline{AB}$.

The runtime of finding $\mathbf{P}_{\theta_L}$, given an appropriate TraceSegment TS of n points, is inherently linear. However, this can be easily improved to $O(1)$ time if we take advantage of the preconceived structure of the 'active-sampling' routine. We do this by simply maintaining a reference to a point in TS that is the closest to $\overline{CM}$.

Again, without loss of generality, we shall assume that the TraceSegment $TS = \{p_1, \ldots, p_n\}$ is flowing from left to right, where the anchor-point $\mathbf{P}_{\theta_B} = p_n$ lies to the right of the tangent line through the anchor-point $\mathbf{P}_{\theta_A} = p_1$.

Proposition 9.3 *Let* $p_L \in TS$ *be the 'farthest reference' point* on, *or immediately to the right of the bisector line* $\overline{CM}$, *so that* p_{L+1} *is strictly to the right of* $\overline{CM}$. *Then given a new incoming point* p_k *with a corresponding new constraint-triangle* $\Delta ABC'$, *where* p_k *is not marked as a crucial point,* p_{L+1} *is the new 'farthest reference' point if and only if the current* p_L *lies to the left of the new bisector line* $\overline{C'M'}$.

For example, we start with the base case $TS = \{p_1, p_2\}$. Then p_2 is the reference point p_L lying to the right of $\overline{CM}$. When a new point p_3 is added into TS, we only need to check that p_2 remains to the right of the new bisector line $\overline{C'M'}$. Otherwise, p_3 is the new farthest reference point p_L. This goes on until the newly inserted point p_k into TS is marked as crucial by the active-smoothing routine. We give another example of graphically in Figure 9.1.

Thus, we determine the point $\mathbf{P}_{\theta_L}$ that is farthest away from the baseline $\overline{AB}$, only when it becomes necessary, by simply computing the intersection of the two line-segments $\overline{CM}$ and $\overline{p_L p_{L+1}}$.

9.3 Maintaining Areas and Lengths

Often, one may find it useful to keep track of the area underneath the Trace-Segment TS and above the baseline of the constraint-triangle ΔABC. The trapezoidal rule satisfactorily provides a good measure of this, involving all the points $p_i \in TS$. However, this may turn out to be an expensive exercise, as the baseline $\overline{AB}$ changes for every new point p_k added into TS; that is, one would have to recompute the area again from the initial point p_j for every new p_k.

The same argument goes in keeping track of the length TS and reporting the measure between any two points, say, p_j to p_k in TS, where $j < k$. One would have to sum the lengths of the Edges $\overline{p_j p_{j+1}}, \ldots, \overline{p_{k-1} p_k}$, which takes $O(n)$ time.

Despite these inherently linear time operations, we can perform a piggy-back strategy to hook onto the 'active-sampling' routine to give us the area and length measure in $O(1)$ time. We do this by adding two additional variables to the `ActivePoint` structure to store the cumulative area κ_{Area} and the cumulative length κ_{Len} at each point p_i.

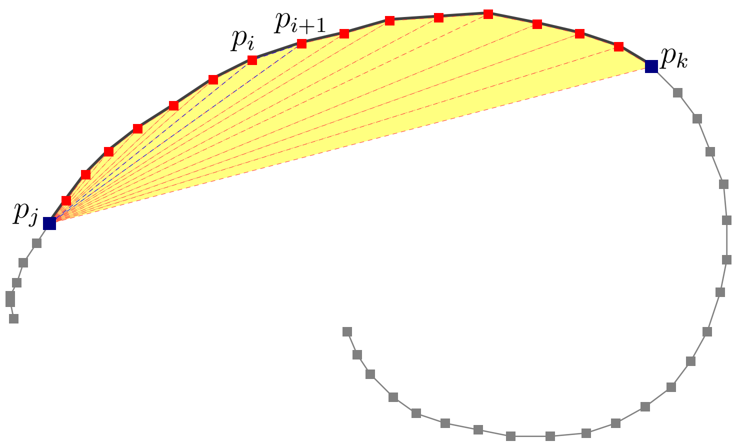


Figure 9.2: Cumulative triangle areas between p_j and p_k.

Lemma 9.4 *Let p_j be the last crucial point identified by the active-smoothing routine, and let p_k be the latest point added into TS. Then the area underneath the TraceSegment TS inside the constraint-triangle at the point p_k from the start point p_j is*

$$\kappa_{\text{Area}}(p_k) = Area(\Delta p_j\ p_{k-1}\ p_k) + \kappa_{\text{Area}}(p_k - 1).$$

Proof. The area underneath the TraceSegment TS is the sum of a series of triangles given as

$$\text{Area}(\Delta p_j p_{(j+1)} p_{(j+2)}) + \ldots + \text{Area}(\Delta p_j p_{(k-1)} p_k) = \sum_{i=j+1}^{k-1} \text{Area}(\Delta p_j p_i p_{(i+1)}).$$

■

Lemma 9.5 *Let p_k be the latest point added into TS. Then the length of the TraceSegment TS at the point p_k is*

$$\kappa_{\text{Len}}(p_k) = \|p_k \; p_{k-1}\| + \kappa_{\text{Len}}(p_k - 1).$$

Proof. The length the TraceSegment TS is the sum of the lengths of the Edges $\overline{p_0 p_1}, \ldots, \overline{p_{k-1} p_k}$. ■

Thus, by Lemma 9.4 and Lemma 9.5, we need to perform two additional computations to set the values of $\kappa_{\text{Area}}(p_k)$ and $\kappa_{\text{Len}}(p_k)$ at p_k, as an extra step to the 'active-sampling' routine.

So to obtain the area underneath TS at any point p_i, where $j < i \leq k$, we simply query the value of $\kappa_{\text{Area}}(p_i)$. Similarly, to determine the length of the TraceSegment between any two arbitrary points p_r and p_s, where $r < s$, we compute the straight-forward difference $\kappa_{\text{Len}}(p_s) - \kappa_{\text{Len}}(p_r)$.

We note here that both these measures are estimates for the actual underlying Curve KS, which may or may not be corresponded to TS. However, $\kappa_{\text{Area}}(p_i)$ provides a quick comparative measure when juxtaposed with the *actual* area underneath KS, which can also be computed in $O(1)$ time we stated earlier in Theorem 8.6.

9.4 The Active-Smoothing Algorithm

Ever since the beginning of the thesis, we advocated and insinuated that the active-smoothing solution should be one that is most efficient, reliably accurate, least time consuming, and most important of all, simple. And since the introduction of the geometrical elements of the digital ink in chapter 3, we have guided, discussed, and presented our thoughts to culminate here, where we shall now assemble all that we learnt to approximate a set of indeterministic points desirably with smooth curves, actively.

In fact, the essence of simplicity in our active-smoothing solution is overwhelming. We sum up the ideas necessary to conceive the algorithm in the following three steps.

- Perform only directed $O(1)$ time computations at each newly instantiated `ActivePoint` p_k (computations include the first and second derivatives, and the curvature at p_k). Do this while also maintaining two purposeful reference points p_j and p_L; where p_j is the last identified crucial point in the current active Trace T (section 7.5), and p_L is the farthest point reference in the current active TraceSegment TS (section 9.2).

- Since p_k contains cumulative measures with respect to areas and length (section 9.3), and that the intermediate points between p_j and p_{k-1} always make up an appropriate TS, we only need to react when we encounter either one of these two situations:

  – The point p_k is identified by the `ActiveSampling()` routine as either `SharpEdge` or `Inflection` – in which case we run the Bézier segmentation routine on the segment $\{p_j, \ldots, p_k\}$; or

  – The active segment $\{p_j, \ldots, p_k\}$ is no longer appropriate – in which case we run the Bézier segmentation routine on the segment $\{p_j, \ldots, p_{k-1}\}$.

  Note that it takes $O(1)$ time to determine whether or not $\{p_j, \ldots, p_k\}$ is appropriate (section 9.1).

- Finally, we tidy up the segment between p_k and the final point p_n, once the pen is lifted off the transducer device. We complete the active-smoothing routine by performing the final Bézier segmentation on the segment $\{p_j, \ldots, p_k, p_{k+1}, p_n\}$.

The first two steps make up the bulk of the active-smoothing process, and as we mentioned at the beginning of the chapter, they form the extension to the `ActiveSampling()` routine we discussed in section 3.5. Thus, putting them all together, we present the pseudo-codes to the active-smoothing algorithm in Algorithm 14. The algorithm focuses on the point p_k, as described by the three steps above. The extended steps added to the original `ActiveSampling()` routine are highlighted in blue. We illustrate this visually in Figure 9.3 continuing from the Bézier segmentation process in Figure 9.1.

9.5 Performance Analyses

To fully appreciate the significance of the active-smoothing routine, we compared our technique to the Adobe Acrobat's annotation tool. What we found was very encouraging.

It takes the entire 100% processor resources for Adobe Acrobat to sample and render the annotations on the PDF documents while the user is still writing, and

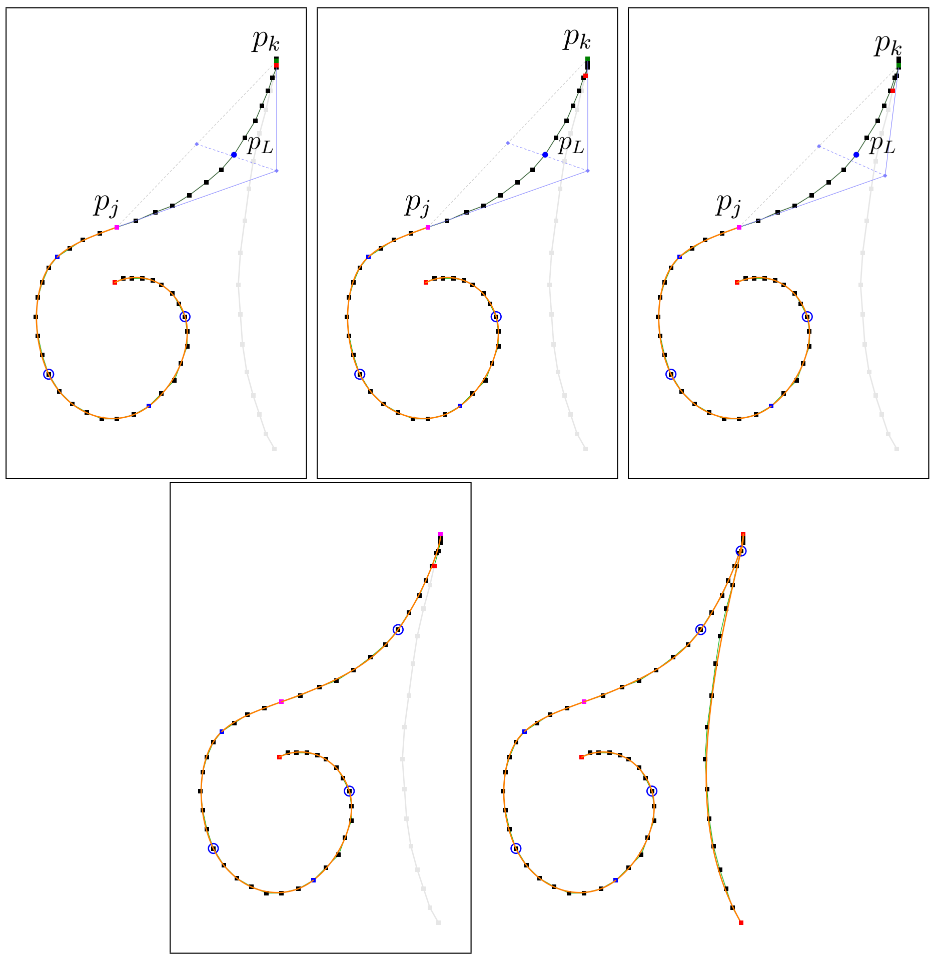


Figure 9.3: Active-smoothing. A continuation of the process started in Figure 9.1. Here we show that once a segment is appropriate, it is immediately smoothed by the active-smoothing process.

then smooth the Traces after the user finishes. In contrast, it takes our active-smoothing routine no more than 18% of the processor resources to smooth *and render the smoothed* handwritings while the user is still writing on the document. This is a reduction of over 80% of the processor time to service our algorithm compared to servicing Adobe's; the amount of which can definitely be put to other useful purposes, if the need arises.

Relating this back to our problem statement in chapter 2, this release of processor resources can be utilised to service the other input modals (i.e. video, audio, etc.) in the open document type in Lecturnity, without compromising on the quality of the recorded freehand writings.

To end of this chapter, we present four other examples to show the output quality renditions of the active-smoothing routine. We illustrate them in Figures 9.4 to 9.7.

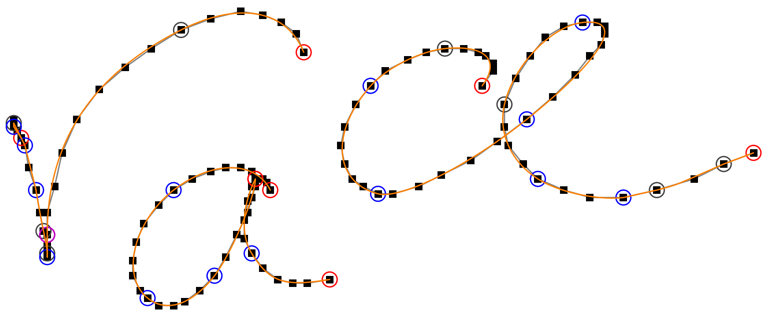


Figure 9.4: Active-smoothing example output quality. A point with a circle around it denotes a crucial point; a red circle marks a sharp-edge vertex, a blue circle marks a G^1 continuity point, and a purple circle marks an inflection point.

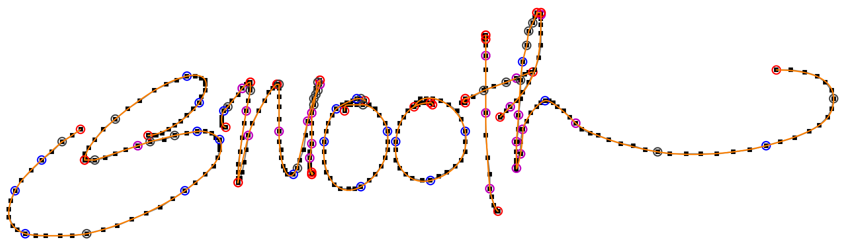


Figure 9.5: Active-smoothing example output quality.

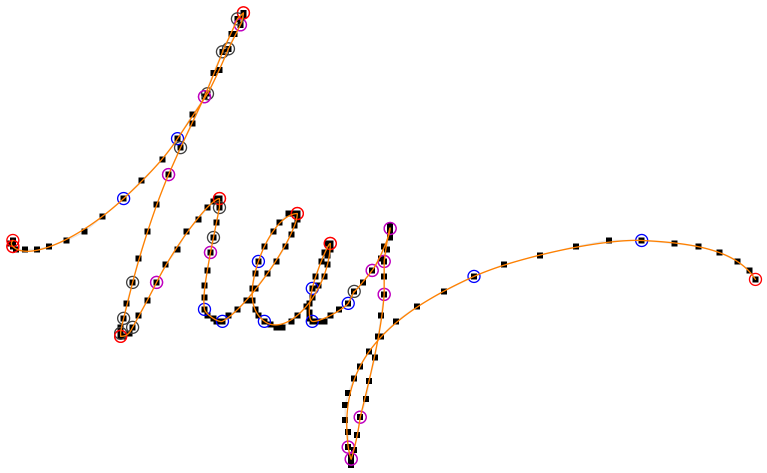


Figure 9.6: Active-smoothing example output quality.

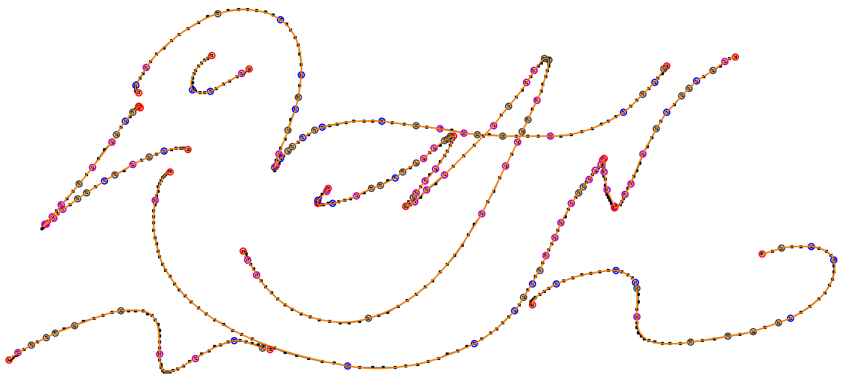


Figure 9.7: Active-smoothing example output quality.

Algorithm 14: `ActiveSmoothing.addPoint` (`ActivePoint` p_n)

Output: (*Actively* to screen) The smoothed approximate of a Trace T.
Data: **Global variables:** (i) `ActivePoint` p_j: The last crucial point identified; (ii) `ActivePoint` p_L: Farthest point reference; (iii) `Trace` T: The current Trace that is being actively smoothed.

```
1  Set Type($p_n$) ← SharpEdge;
2  if $T$ is empty then
3      Add $T \leftarrow p_n$;
4      Set $p_j, p_L \leftarrow p_n$;
5  else
6      Set dy/dx($p_n$) ← $(p_n.y - p_{n-1}.y)/(p_n.x - p_{n-1}.x)$;
7      Set $\kappa_{\text{Len}}(p_n) \leftarrow \|p_n\ p_{n-1}\| + \kappa_{\text{Len}}(p_{n-1})$;
8      if $p_{n-2}$ DO NOT exist then Set dy/dx($p_{n-1}$) ← dy/dx($p_n$);
9      else
10         Set Type($p_{n-1}$) ← Normal;
11         Set dy/dx($p_{n-1}$) ← $(p_n.y - p_{n-2}.y)/(p_n.x - p_{n-2}.x)$;
12         Set $\kappa_{\text{Area}}(p_n) \leftarrow$ Area($\Delta p_j\ p_{n-1}\ p_n$) $+\kappa_{\text{Area}}(p_{n-1})$;
13         if $p_{n-3}$ exists then
14             Set $p_k \leftarrow p_{n-2}$;
15             Set ddy/ddx($p_k$) ← $(p_{k+1}.y - p_{k-1}.y)/(p_{k+1}.x - p_{k-1}.x)$;
16             Set $p_k.\hat{k}$ ← Compute average curvature;   /* Equation 3.5 */
17             if $|p_k.\hat{k}| >$ SHARP_TOL then Set Type($p_k$) ← SharpEdge;
18             else if sign($p_k.\hat{k}$) $\neq$ sign($p_{k-1}.\hat{k}$) then
19                 Set Type($p_k$) ← Inflection;
20             Initialize $\Delta ABC$ ← Constraint-triangle($p_j..p_k$);
21             Set $M$ ← Midpoint($\overline{AB}$);
22             if $p_{L+1}$ is to the left of $\overline{CM}$ then Reset $p_L \leftarrow p_{L+1}$;
23             if Type($p_k$) = Normal then
24                 Set $\overline{AB} \leftarrow \overline{p_j\ p_{k+1}}$;
25                 Set $p_C$ ← Intersection tgt. lines at $p_j$ and $p_{k+1}$;
26                 if $p_C$ and $p_k$ on opposite sides of $\overline{AB}$ then
27                     Set Type($p_k$) ← MarkG1;
28             if Type($p_k$) $\neq$ Normal then
29                 Set $\mathbf{P}_{\theta_L}$ ← Intersection $\overline{CM}$ and $\overline{p_L\ p_{L+1}}$;
30                 Set $K$ ← FastBezierSegmentation($p_j..p_k$, $\mathbf{P}_{\theta_L}$, $\epsilon$);
31                 Render CompositeCurve $K$;
32                 Reset $p_j, p_L \leftarrow p_k$;
33                 Update $\kappa_{\text{Area}}(p_k)$, $\kappa_{\text{Area}}(p_{k+1})$, $\kappa_{\text{Area}}(p_n)$;
```

CHAPTER 10

Random Access Navigation

Let us now look into one of the fundamental application of the digital ink technology. The focus of this chapter is on the ease of navigation, particularly the real-time "active visible scrolling" of recorded freehand writings in ink documents. Metaphorically, this ideology resembles the thumb-flipping of a book. The contents we present in this chapter have been communicated and published at the ACM's World Wide Web International Conference [84]. In the six upcoming sections, we shall describe the importance of random access navigation with respect to the freehand writings.

10.1 Replaying Ink in Real Time

The electronic pen's digital ink offers a convenient way for visually communicating ideas in the vicinity of digital screens, which growing trends indicate it to be a favourite amongst *technocrats*, and the likes, ever since its conception. This is particularly true (and important) in many modern and networked learning environments, where using this method of communication is seen as a natural way of exchanging viewpoints between teachers and students. Its a digital upgrade, if you like, from the messy blackboards and chalks, to the cleaner whiteboards and markers, to the *cooler*, state-of-the-art digital-boards and styluses. This trend of moving towards natural environments in the digital world encourages freehand writings to be the bridging tool that encompasses portable and synchronizable standards of manipulating ink data. Apart from allowing writers to showcase the flair and style of their austerity, there are a few other benefits that comes with this package that must be highlighted.

Take for example this lecture hall scenario. Lecturers typically deliver their lessons through slides prepared from customary softwares like MS PowerPoint and Adobe Acrobat. Essentially, as their classes progress, they will write, highlight, underline, draw and formulate on those slides to put forth their points to the students. And by opening "empty" slides, just so that they can spontaneously write on them, actually facilitates discussions amongst students, helps to illustrate complex mathematical proofs, and demonstrates diagrammatic facts more clearly. Studies show that the combination of handwritten annotations, the development of thoughts through freehand writings, and the verbal explanations of the lecturers, are indeed crucial in assisting students' understanding of subjects [47, 91, 138]. The evidence confirm that simply capturing the static end-result of the lecturer's annotations for recalling the subjects at a later time, is indeed inferior than to unambiguously maintain the whole sequence and dynamics of those freehand annotations when recorded for interactive replay.

User interfaces that allow replaying of real-time multimedia documents usually include command buttons to quickly *fast-forward* and *rewind* their data sets for systematic browsing. We see this very often in many learning softwares that try to cater to both students and teachers in assisting them to effectively retrieve desired segments of those documents [138, 91]. Random access navigation provides an additional advantage on top of this simple fast-forward and rewinding of real-time information. It gives users the ability to straight away focus on the intended portions of their documents without having to wait a second longer for these processes to arrive at the stipulated point of time, albeit the processes being a little faster than the normal replay mechanism.

10.2 Random Access Navigation

Random access navigation on real-time multimedia documents provides an additional advantage on top of the simple *fast-forward* and *rewinding* of real-time information.

10.2.1 Streaming Freehands

We term the process of replaying ink data streams in exactly the same recorded sequence, or a sequence that is as close as possible to the users' original writing-motions as *streaming freehands*. Throughout our discussions, we shall assume that all recorded ink data are properly sampled in the highest granularity, that is they are captured by reliable mediums so that the full dynamics of users' writing styles are preserved as accurately as they appeared on screen when recorded.

10.2.2 Accessing Objects

Müller and Ottmann [92] defined *random accessibility* as the retrieval of all necessary objects, which must be made completely available in an uncompressed or dereferenced way, at any time instance, such that the entire *access time* of these objects is less than the media-specific threshold period τ. Here, this threshold period refers to the hardware-specific duration of accessing raw data (on disk, for example) and rendering them onto the screen. Putting it simply, given any instance of time on the 'active' slider-bar, we must have a complete list of all associated *visible* objects for that time instance to be made available for display on screen – so fast, that users get the impression of *immediacy.* This is usually tolerated for up to a few hundred milliseconds, as is expected from the threshold period τ.

Furthermore, as characterised for general multi-stream domains, random accessibility also insists on the *master-slave* synchronisation between the audio stream (master) and the digital-board action streams (slave). In our case, this is seen as the seamless blending of users' voices to their freehand writings during recording, for a more solid synchronisation model when replayed.

We discuss the InkML data representation model in section 10.3, but for now, we will express each 'ink trace' as an *ink-object* to parallel the arguments above. Accessing a particular ink-object at a stipulated time event t_q for use in random-access navigation, demands the instantaneous rendering of that ink-object, *plus* all other ink-objects since the time event t_0 that are deemed visible prior to t_q.

10.2.3 Timing Significance

The operation of streaming freehands involves a set of time-evolving ink-objects with inherent time dependencies. This is a direct consequence of the recording process, in which provisions were made in the first place to allow users to manipulate their handwritings in any way they like. For example, it is natural for users, while writing, to erase, duplicate, enlarge, change colours, or move their traces around the electronic board to take advantage of the facilities offered by the control software. Each of those manipulations coincide at certain time instances on the slider-bar, and the recording process responds to these by either creating new ink-objects, deleting present ones, or modifying the context details of current selected ink-objects to reflect the users' handling of their writings and sketches.

From here on, we discuss only the *discrete* changes of the context details of the ink-objects as pointed out by Salzberg and Tsotras [110]. However, we also note that the *continuous* modifications of these contexts would further require a separate set of scalable data structures that can be used in both temporal and bitemporal development environments [23].

Streaming freehands, in addition to the above manipulations, is what makes these dynamic ink documents unique, in contrast to just statically render the final end-product. Although we will not see much difference in timing comparisons of random accessibility between the two entities for small data volumes, enlarging the scope to deal with situations where huge data volumes is inevitable, results in a substantial amount of time that makes it necessary to re-evaluate the applied concepts of "immediacy" to ensure user-practicality is addressed.

10.3 Ink Markup Language (InkML)

W3C's InkML standard [108] allows the description of precise digital ink information that is not only interchangeable between systems, but also allows the convenient 'archival' and 'streaming' of ink data between applications. Amongst many of its properties, it is important to note that;

- InkML supports the representation of handdrawn ink, on top of capturing the pen positions over time;
- InkML allows the specification of extra information for additional accuracy such as pen tilt and pen tip force (pressure) for the enhanced support of applications such as character and handwriting-style recognition, and authentication; and
- InkML provides the means for extension – by allowing application-specific information to be added to the ink files to support issues of compatibility.

10.3.1 Structuring InkML

In addition to the *xy* coordinate channels, we include a time-stamp "**T**" in the *regularChannels* to register time instances for each sampled coordinate. We utilised the following *traceFormat* for all recorded data in our experiments and findings:

```
<traceFormat>

  <regularChannels>

    <channel name="X" mapping="*">
    <channel name="Y" mapping="*">
    <channel name="T" mapping="*">

  </regularChannels>

  <intermittentChannels>

    <channel name="Eraser" type="boolean"
    default="False">

  </intermittentChannels>
```

```
</traceFormat>
```

Our application-specific navigation technique for random accessibility of ink documents, concentrates mainly in the archival mode of InkML. We post-process the recorded elementary ink data to extract the basic instructions marked up in the file to reconstruct the exact sequence of the original handwritten traces. We then port these instructions into segmented *object modules* to coincide with the actual slider-bar time reference.

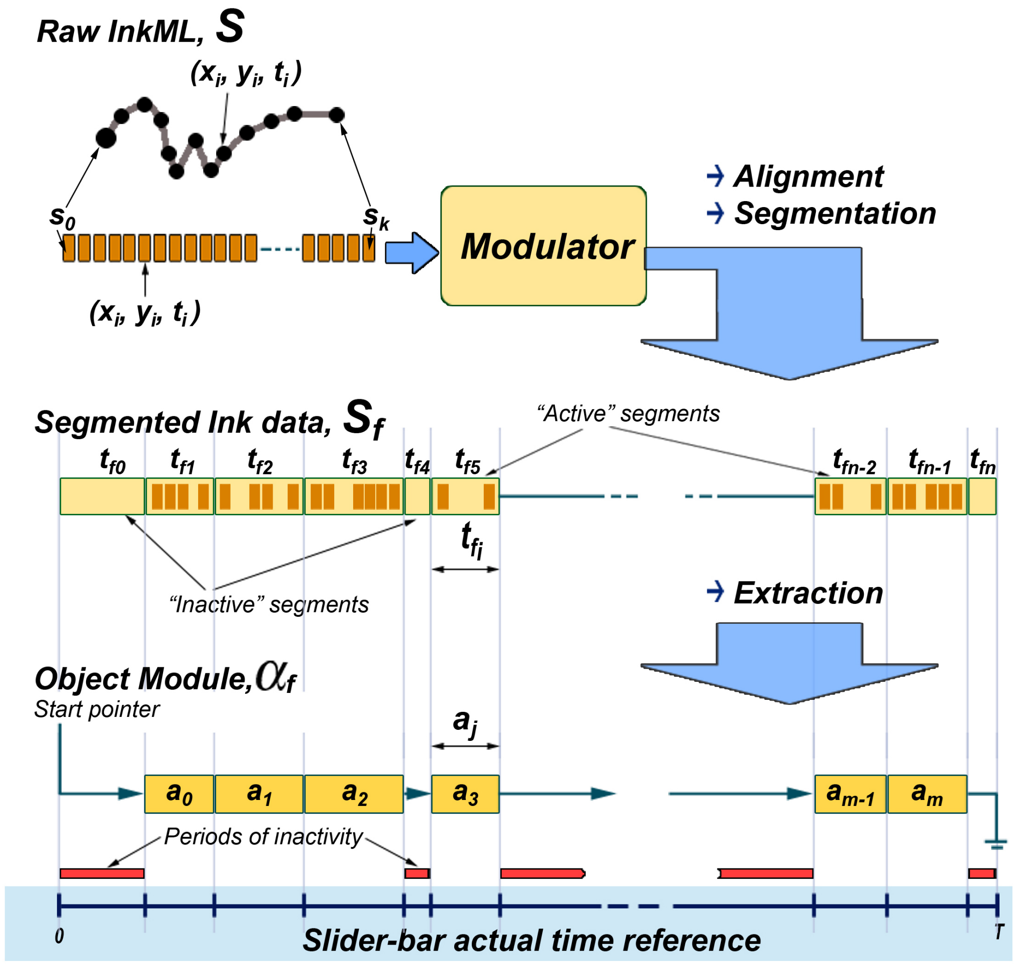


Figure 10.1: Modulated alignment of the resultant *object modules* α_f to the segmented time scale of the slider-bar.

10.3.2 Object Modules

Extricating the fundamental information from the InkML file returns a set $S = \{(x, y, t) < E >\} = \{s_0, s_1, ..., s_k\}$, where tuples $s_i = (x_i, y_i, t_i)$ refer to the *regularChannels*' data corresponding to the xy coordinates and time stamps t of the i^{th} sequence. The *intermittentChannels*' information $< E >$ are instructions for erasing ink traces whenever this boolean variable is set to $E =$ "True". Lines are drawn between two successive tuples s_i and s_{i+1} if the variable $E =$ "False".

This set S is translated onto the slider-bar's *actual* time reference-scale and segments $S_f = \{t_{f0}, t_{f1}, ..., t_{fn}\}$ are built upon it to collectively assemble the aligned s_i tuples. Each segment t_{fi} is constrained to a period of not more than a pre-defined tolerance of t_σ msec, and may result in an empty segmented content.

A segment t_{fi} in S_f is considered "active" if there exists ink data within itself, and is considered "inactive" otherwise.

Our *object modules* are derived from all active segments in S_f to result in $\alpha_f = \{a_0, a_1, ..., a_m\}$, such that $\alpha_f \subseteq S_f$ and $m \leq n$. We illustrate our "modulator" method and an example of object modules juxtaposed to a slider-bar's actual time reference scale in Figure 10.1.

Here, we further define that an object module a_j, is a primitive Java graphics object that contains the underlying symbolic rendering instructions represented by the segment. Rendering an object module a_j to screen on the Java platform is carried out by executing `g.draw(aj)`.

10.4 Replay Scenarios

We tailored our experiments to specifically test documents with huge amounts of ink data – a minimum data size of 60 minutes worth of continuous writing. We prepared three data management scenarios based on our object modules targeted at the following objectives for navigating with random accessibility:

- to observe and record the reaction (rendering) times against data volumes;
- to scrutinise the change in data volumes with respect to the change in reaction times;
- to pick out the general relationship (if any) between the data volumes and the reaction times; and
- to recommend a suitable technique for handling massive data volumes for practical applications involving freehand writings that exploits the InkML standard.

Our three scenarios differ in their aspects of representation and access to data for the same set of ink information. Their performances are expected to have significant impacts on the reaction times during random-access navigation.

10.4.1 Scenario I – Raw InkML

All data are lifted and constructed directly from the InkML file, and no object modules are created in this scenario. We kept the primitive $S = \{(x, y, t) < E >\}$ tuples throughout the process of random accessibility, as depicted in Figure 10.2. This scenario quickly renders ink traces by reading the instructions contiguously from the file but ignoring the time stamps t. New lines are drawn by joining a pair of coordinate points, or erased according to the pre-recorded sequence in the file.

At query point t_q, the program identifies the time reference t_i on the slider-bar to its nearest corresponding tuple $s_j = (x_j, y_j, t_j)$, and executes all ink instructions starting from s_0 to s_j.

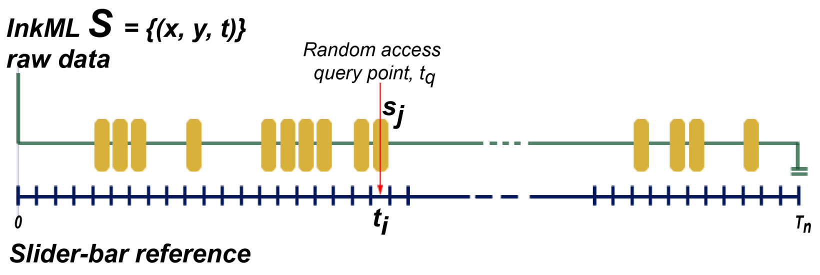


Figure 10.2: Scenario I – Rendering from raw InkML.

10.4.2 Scenario II – Applied Object Modules

Object modules described in Section 10.3.2 are utilised for this scenario. The fragmented data S_f is converted into α_f by picking out all active segments.

At query point t_q, the program identifies the last *complete* object module a_{j-1} and starts rendering from module a_0 to a_{j-1}. Afterwhich, it drops back to the 'raw InkML' method specified in Scenario I to finish the remaining set of data contained in ε_q, as illustrated in Figure 10.3.

10.4.3 Scenario III – Cumulative Modules

Similar to Scenario II, object modules are also implemented here. However, instead of simply keeping active segments of S_f within each module, we defined a

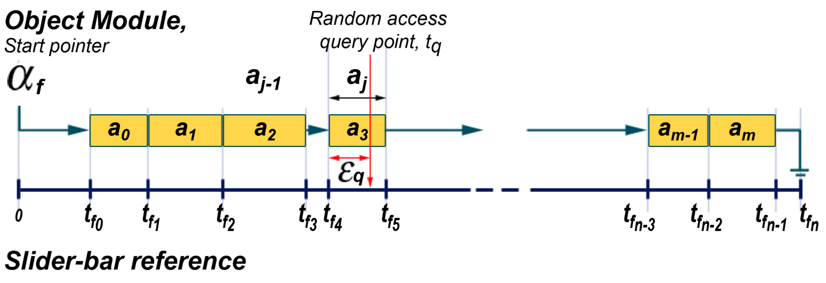


Figure 10.3: Scenario II – Applied data modules.

cumulative property for each module a'_j, such that it collectively stores all previous modules running up to its position. Equation 10.1 describes this cumulative property of α'_f, and is reflected pictorially in Figure 10.4.

$$a'_j = a_0 + a_1 + ... + a_j = \sum_{c=0}^{j} a_c \tag{10.1}$$

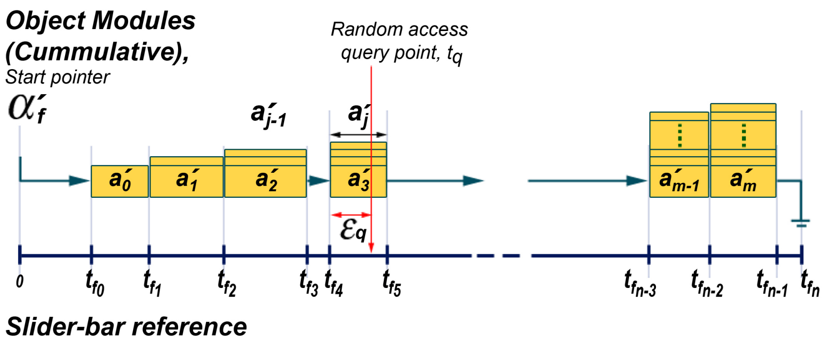


Figure 10.4: Scenario III – Cumulative data modules.

Since our definitions insist that each data module a_j results in the creation of primitive Java graphics objects, we added this 'redundancy' information in a'_j, accumulated over each preceding module, to experiment with the trade-offs between memory space and the processor's execution time.

At query point t_q, the program identifies the last *complete* module a'_{j-1} and renders it straight away (as all associated *visible* objects prior to t_q have already been pre-calculated and contained inside a'_{j-1}). Then, it again drops back to the 'raw InkML' method to construct the remaining set of information within ε_q.

10.5 Measurables and Data Contents

Data volumes are measured from the total number of coordinates constructed on screen at any stipulated time instance referenced by the slider-bar. This *primary* representation is always the same for all the three scenarios above if the same set of InkML files are used. We base our evaluations on this basic standard, so as to provide a realistic benchmark for comparing data management styles in our three scenarios.

However, we must also note that the modulated representations in α_f may contain different voluminal capacity in each a_j, as they are the symbolic representations of the affected coordinates, and therefore need to be measured in different units. But to keep things on a more tolerable level, we will use the measurement described above for data volumes to be categorised as 'number of coordinates'.

When discussing *reaction* times, we refer to the duration the program takes to relay complete rendering instructions to the screen. This is again referenced to the time instance indicated on the slider-bar, and more importantly, to the associated data volume.

All contents in our InkML setup files observe the following guidelines as part of the experimental procedure;

- non-stop writing from the beginning to the end of recording (to test for the worst case scenario);
- at some point in time during recording, writing pauses for some period before continuing; and
- at some point in time during recording, some writings are erased off the screen before continuing.

10.6 Results and Observations

The experiments were conducted on a Wacom PL-400 connected to a Pentium X86 processor running at 800MHz. We engaged 10 sets of InkML files for our experimental setup, each with a data volume of over 60 minutes worth of continuous writings. We tested the integrity of the InkML files on their reactions to the three data management scenarios built for the random-access navigation's "active visible scrolling" ideology.

We collected and averaged the data observations for all volumes rendered, and the *reaction* times for those volumes, at fixed intervals of 50,000 milliseconds – referenced by the active slider-bar actual time-scale. At each of these intervals, the program notes the query point time-reference t_q, and executes the random-access navigation routine. That is, it renders all stored information, depending on the data management model, beginning from t_0 up to t_q. The mean data volumes V and *reaction* times T are then plotted against the stipulated intervals for each of the three data management scenarios, and is depicted in Figure 10.5. Plotting the same *reaction* times against the volumes is illustrated in Figure 10.6. Since we fixed the random-access intervals at a constant rate for our observations, we can also monitor this rate of change of the volumes δV, and the rate of change of the *reaction* times δT. This is shown later in Figure 10.7.

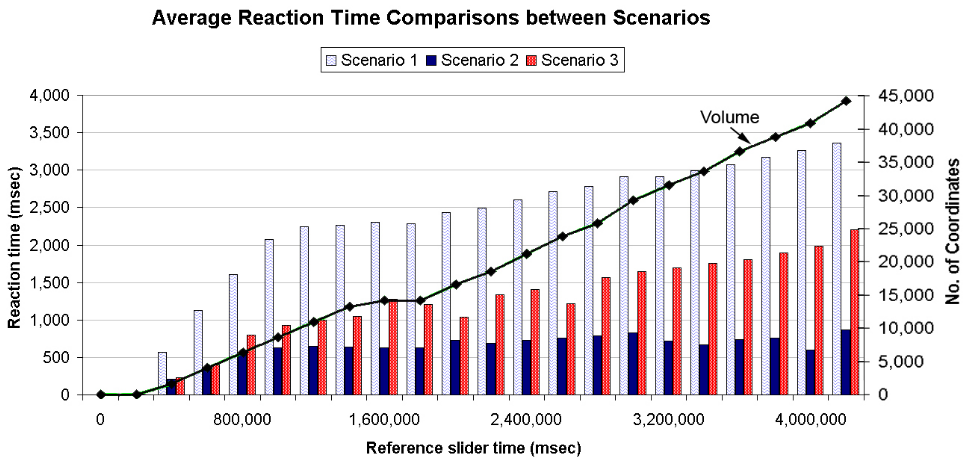


Figure 10.5: Average reaction times (bar graphs) between scenarios I, II, and III aligned to the slider-bar's time-reference scale. The corresponding handwritten volume at each time-reference is represented by the line graph.

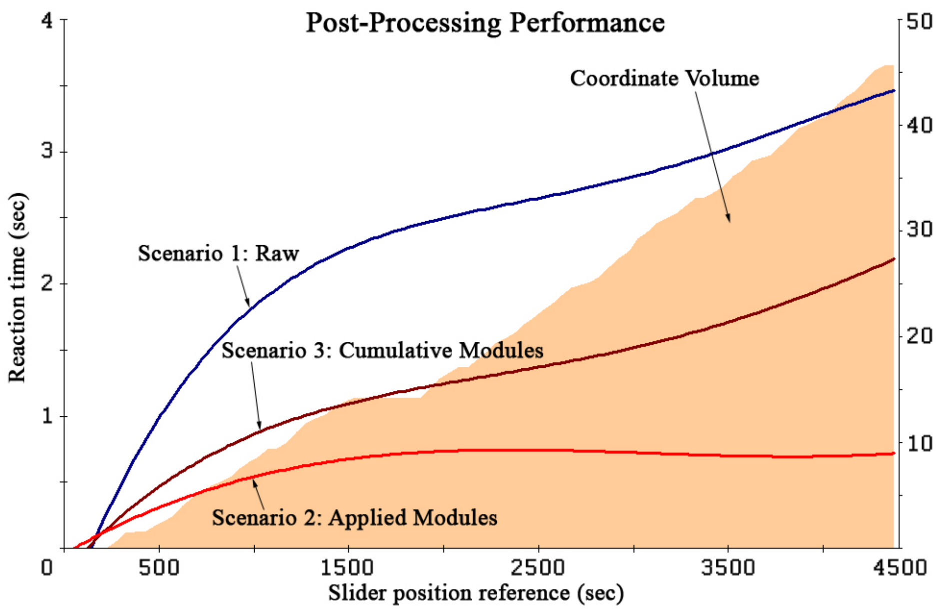


Figure 10.6: Performance: Average reaction time taken by scenarios I, II, and III to render the volume of data coordinates.

10.6.1 Analysis

When we structured the formulation of Scenario III, we anticipated a data management model that is to superceed the random accessibility performances of the other two scenarios in terms of the speed of access. This cumulative method of storing multimedia objects was proved to be speed-efficient and sound for retrievals of multi-stream entities used in domains that engage the “random-access replay” facility [92]. However, the results in Figure 10.5 tells otherwise.

For randomly navigating through huge data volumes containing digital ink, the method of *applied object modules* described in Scenario II maintained a reaction time of less than 1,000 msec on average. This is a comfortable fraction compared to the other two scenarios for our InkML setup files. This is discussed in further details in the “Performance” section.

Writing continuously for over one hour, and observing the data content-guidelines in section 10.5, gave an average of 45,000 coordinate volume capacity in all our InkML files. The growth in volume of each InkML file over time is depicted in the line graph in Figure 10.5, and its growth rate is strictly dependent on the speed of the person writing.

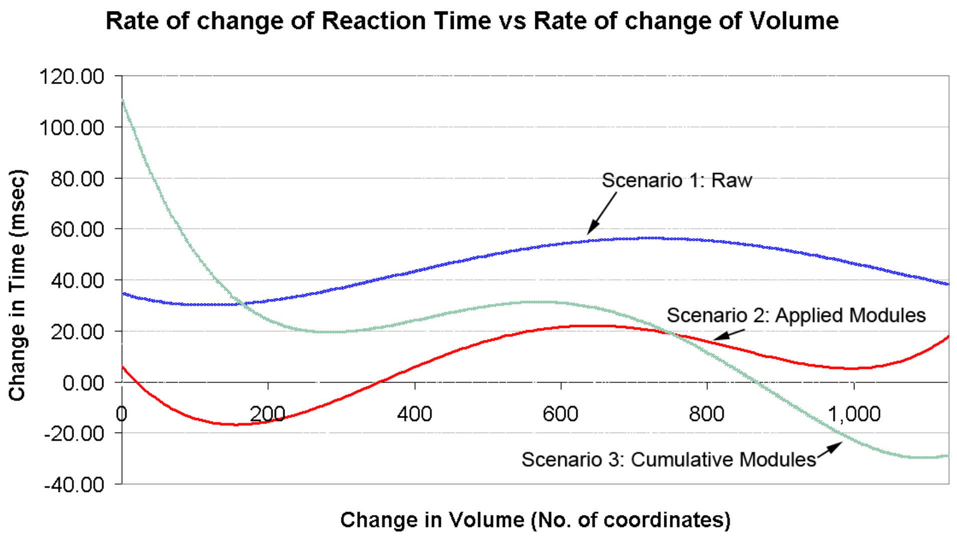


Figure 10.7: Rate of change of reaction times δT versus rate of change of volumes δV.

The factors that are thought to affect the reaction times over the data volumes, that highlights the distinctive graphs in Figure 10.6 for each of the three scenarios, is given in the “Correlations” section. That section also explains the graph in Figure 10.7, which is used to derive our “Recommendations” of handling massive volumes of ink data for practical “active visible scrolling” navigation, and its synchronisation to the master/slave configuration for general multi-stream applications.

10.6.2 Performance

The chart in Figure 10.5 is a repercussion of the practical aspect of this experimental setup. The data volumes (line graph) and their *reaction* times (bar graph) are aligned to every sampled interval on the slider-bar's time-reference scale. Navigating by dragging the slider-bar's indicator forwards and backwards took that much of average reaction times as shown on the chart.

Figure 10.6 is a graph that highlights the reaction times compared directly to the data volumes. This theoretical aspect is aimed to measure the generalised reaction time-performance, based on the three data management scenarios.

For data volumes that are above the 3,500 coordinate-capacity level, it took at least 1,000 msec to render the ink information fully on screen using the data model described in Scenario I. This volume is raised to a 10,000 coordinate-capacity level before it took Scenario III the same amount of reaction time to render its data contents. Scenario II, on the other hand, did not overshoot the 1,000 msec mark at all for rendering the entirety of its data volumes.

All three scenarios exhibit the essence of a logarithmic complexity in their reaction time-performance. However, we can confirm from Figure 10.5 and Figure 10.6, that, primarily, the data management model used in Scenario II is the most efficient method for handling random-access navigation. Its rate of convergence is 3.792 times faster then Scenario I's and is 2.029 times faster than Scenario III's. Its ability to keep the reaction times below the 1,000 msec mark is a tremendous advantage for the "active visible scrolling" technique in relation to the governing media-specific threshold period τ for issues of *immediacy*. Moreover, this scenario is also more economical than Scenario III in terms of memory manipulations – unlike Scenario III, Scenario II's data modules do not have the cumulative property.

10.6.3 Correlations

Apart from observing the surface performance of managing huge ink information, we also scrutinised the underlying strengths (or shortcomings) of the three data management scenarios. This is done through the analysis of the two most prominent rates of change; δT and δV.

The graph in Figure 10.7 estimates $\frac{\delta T}{\delta V}$ for all three scenarios. Their overall correlated average is given in Table 10.1. We explain the relevance of this $\frac{\delta T}{\delta V}$ in the "Recommendations" section.

As expected (from the analyses in the previous sections), Scenario II has

Scenario	**Avg. Correlation** $\frac{\delta T}{\delta V}$
Scenario I	0.10696
Scenario II	0.00734
Scenario III	0.13929

Table 10.1: Overall average correlations of the three scenarios.

the smallest average correlation value of 0.00734, and that the performance of its data management model is construed to be the most stable (with the least fluctuations near the zero level) depicted by the results in Figure 10.7. The average $\frac{\delta T}{\delta V}$ is calculated from the mean values of the each raw data obtained from δT divided by δV.

Referring back to the descriptions of *object modules* α_f, we can conclude that the rate of change of reaction times in α_f for Scenario II is small enough for the comfortable and practical random-access navigation of huge data volumes. Its performance at this rate is sustainable for a projected volume of up to 80,000 coordinate capacity.

10.6.4 Recommendations

Based on the findings in the "Correlations" section above, we know that the factor which governs the relationship between the data volumes and the *reaction* times is $\frac{\delta T}{\delta V}$. We also know that the rendering program is able to immediately determine the amount of volume of coordinate data V at any stipulated time instance referenced by the active slider-bar.

Then, by defining an evaluation function $f(V, \frac{\delta T}{\delta V})$, we should be able to resolve an accurate estimate as to how much reaction time t_R it will take to render a volume V given the relationship $\frac{\delta T}{\delta V}$, as shown in Equation 10.2.

$$t_R = f(V, \frac{\delta T}{\delta V}) \tag{10.2}$$

We recommend the value of t_R be used as a threshold gauge for addressing the concepts of "immediacy" to ensure practicality for end-users. A further post-processing of an InkML file is needed if it is found that at some point in time, after going through the modulation process described in Section 10.3.2, the corresponding data volume will result in a *reaction* time of $t_R > a\tau + b$; where τ is the media-specific threshold period, and a and b are some pre-defined constants.

What remains to be done in the next post-processing step, if the above condition is met, is up to the developers to decide. Although, here are some checkmarks when tackling this problem:

- Converting the resultant screen-shot of the ink canvas constructed from a set of ink instructions to that of a static image for future references, will loose all vectored information (i.e. scaling the application window size up and down may distort this reference image badly, and may no longer be compatible with the current set of ink data).

- Breaking up one massive InkML file into manageable sequential *chunk*-files require some sort of a reference mechanism, so that opening a *chunk*-file down the sequence of *chunks* will not disturb the flow of freehand writings between any two *chunks*.

- Any breakups of the original InkML file that has been configured for the general multi-stream master/slave configuration, must ensure the time-stamps in the InkML file be systematically re-aligned to the other associated event-files before channelling the outputs of each entity to the various mediums.

Part III

Gesturing & Feature Interactions

CHAPTER 11

Gesticulations and the Gesture Continuum

Motivated by the treatment of digital handwriting as 'first-class' data type, we shall now reintroduce the notion of the digital ink here, which is used not only to enter and represent contents but also to operate *gesture commands*. A gesture command is the resultant process of invoking gesture-like movements with the pen, a *gesticulation*, that resemble special shapes drawn in a certain distinctive way, recognised by an application program as certain predefined instructions for it to carry out.

The power of these *gestures* lies in their ability to convey expressive and meaningful strokes as commands to the system without having to switch between other input devices when using the pen explicitly. Cutler and Turk [21], Barrientos and Canny [7], and Wexelbrat [133] noted in their reports the following functional roles of human gestures; communicate meaningful information (semiotic), manipulate the environment (ergotic), and discover the environment through substantial experience (epistemic). Cutler and Turk further expanded their definitions to include communication tasks when interpreting the gestural inputs from the stylus. These tasks specify the commands and parameters for:

- Navigating through 2D space;
- Specifying items of interest;
- Manipulating objects in the sketching environment;
- Changing object values; and
- Issuing task specific commands.

In his sociological field of study, Kendon [57] remarked an important part of kinetics research showing just how gesture phrases are organized in relation to speech phrases. We can parallel his arguments and reasonings to relatively coincide with pen gestures (as natural human gestures) and the instantiated sketch objects (as dictionized speech contents). Kendon stated that there is a consistent patterning in how gesture phrases are formed in relation to the phrases of speech - just as, in a continuous discourse, speakers group tone units into higher order groupings resembling a hierarchy, so gesture phrases may be similarly organized.

Gestures that are put together to form phrases of bodily actions have the characteristics that permit them to be recognized as components of willing communicative action. There are five kinds of gestures that make up the *gesture continuum*:

- *Gesticulation* - spontaneous movements of the hands and arms that accompany speech.
- *Language-like gestures* - gesticulation that is integrated into a spoken utterance, replacing a particular spoken word or phrase.
- *Pantomimes* - gestures that depicts objects or actions, with or without an accompanying speech.
- *Emblems* - familiar gestures accepted as a standard.
- *Sign languages* - referring to the complete linguistic system, such as the American Sign Language.

As the list progresses, association with speech declines, language properties increases, spontaneity decreases, and social regulation increases. This gesture continuum clearly states a useful background support for deriving systems that are able to receive continuous pen gestures as inputs, along with a string of referenced objects.

So far, we are aware that the digital ink derives its name from the data class that represents any kind of information created when using a digital pen to draw strokes on a screen of a pen-based input device. While the strokes received on the pen-computer's surface get recorded as digital ink, its consequential markings are reflected noticeably onto the screen simultaneously. Any digital ink that was written is either kept in its raw form or goes through a process that translates it into recognizable texts or graphic objects. Often, in many pen-based applications, the ink goes further to be tried as a possible pen-gesture. If and when the ink is confirmed as a gesture command, its visual trace on the screen is immediately removed upon the execution of the respective gesture command.

Digital ink in its raw representation as processed by the respective hardware and software holds far greater amounts of information than are immediately apparent. These include, among others; the type of pen tip, the amount of pressure used to create a stroke, the height of the pen above the screen surface, opacity value, colour, and timing data. As a result, the digital ink is acknowledged in serious enterprises such as one for secure handwritten signatures [53, 42, 22], as well as in everyday norms of handwriting email messages [32], sketching design posters [17], and annotating digital documents [37, 92]. As more electronic devices with pen interfaces continue to become available for entering and manipulating information, efforts have been made to ensure that supporting pen-applications are effective at leveraging this method of input. Handwriting is an input modality that is very familiar for most people since everyone learns to write in school. So, there is a high tendency for people to use this mode of input and control not only in the classroom scenario described above but for a great variety of different applications as well.

11.1 The Gesture Entity

Let us restate the gesture recognition problem, which was identified by Rubine [107] in 1991, to fit the structure of our own work with the TRACEs. Given an input TRACE T, determine the Gesture class g_κ to which T belongs. In other words, we want to identify the Gesture g_κ whose members are most like T.

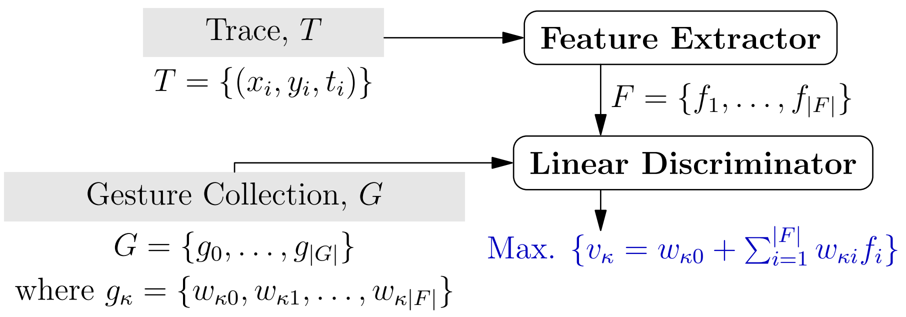
Gesture recognition problem. ♠

Definition 11.1 *A* Gesture g_κ *is made up of a collection of constant, trained weights* $w_{\kappa i}$ *for* $0 \leq i \leq |F|$*, where* $|F|$ *is the number of features, and is identified by its unique identity* κ*.*

We denote the term *G*esture by a capital 'G' in the above definition to differentiate it from the generic meaning of a 'gesture', that was previously explained in chapter 11. A Gesture is never related to a TRACE, and we shall maintain that these two are separate and unique entities. However, they are comparable, albeit not directly, but through the measured set of pre-agreed and predefined features F extracted from the TRACE, put together by way of a linear discriminator. We shall explain the role of the linear discriminator in identifying certain TRACEs as Gestures in section 11.3. For now though, let us first proceed in describing the various features that can be found in any given TRACE.

11.2 Features

Metaphorically, in generative linguistics, *features* are defined as any of various abstract entities that combine to specify the underlying phonological, morphological, semantic, and syntactic properties of linguistic forms. Together, they



Trace, T
$T = \{(x_i, y_i, t_i)\}$
Feature Extractor
$F = \{f_1, \ldots, f_{|F|}\}$
Linear Discriminator
Gesture Collection, G
$G = \{g_0, \ldots, g_{|G|}\}$
where $g_\kappa = \{w_{\kappa 0}, w_{\kappa 1}, \ldots, w_{\kappa |F|}\}$
Max. $\{v_\kappa = w_{\kappa 0} + \sum_{i=1}^{|F|} w_{\kappa i} f_i\}$

Figure 11.1: Linking a TRACE T to a known Gesture g_κ by means of the Linear Discriminator.

act as targets of linguistic rules and operations. Similarly, the features extracted from a TRACE T relay geometrical as well as algebraic information that can be used as parameters which uniquely identify a particular Gesture g_κ.

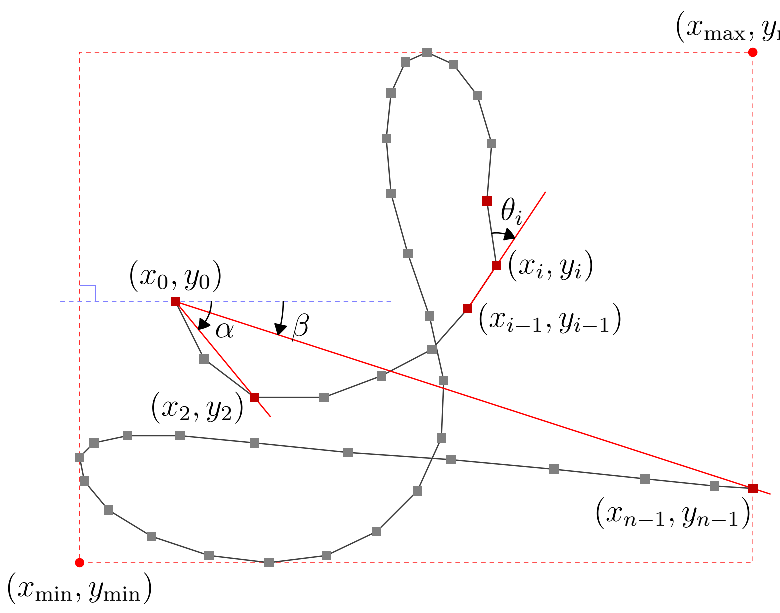


Figure 11.2: Identifying with features by Rubine.

Definition 11.2 *A* feature $f_i \in F$ *is a measured value, representative of a prominent characteristic of any given* TRACE T*, that serves to distinguish it from other features in* F*.*

In order that there be enough *distinctive* features to provide the apparent differentiation across all Gestures in a particular collection, the choice of features should unique in its own right, with respect to the TRACE. Furthermore, each feature should be incrementally computable in constant time per input point, which should then allow for arbitrarily large TRACEs to be handled as efficiently as small ones. Also, a small change detected in the input TRACE should result in a correspondingly small change in the feature.

Rubine introduced a set of 13 standard features, which mathematical functions measure the angular information and lengths related to a TRACE T [107]. We list them down as follows from f_1 to f_{13}, using Figure 11.2 as guide. Let $\partial x_{i\_0} = x_i - x_0$ and $\partial y_{i\_0} = y_i - y_0$.

$$f_1 = \cos\alpha = \partial x_{2\_0} / \sqrt{\partial x_{2\_0}^2 + \partial y_{2\_0}^2} \tag{11.1}$$

$$f_2 = \sin\alpha = \partial y_{2\_0} / \sqrt{\partial x_{2\_0}^2 + \partial y_{2\_0}^2} \tag{11.2}$$

$$f_3 = \sqrt{(x_{\max} - x_{\min})^2 + (y_{\max} - y_{\min})^2} \tag{11.3}$$

$$f_4 = \arctan \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}} \tag{11.4}$$

$$f_5 = \sqrt{\partial x_{(n-1)\_0}^2 + \partial y_{(n-1)\_0}^2} \tag{11.5}$$

$$f_6 = \cos\beta = \frac{1}{f_5}\ \partial x_{(n-1)\_0} \tag{11.6}$$

$$f_7 = \sin\beta = \frac{1}{f_5}\ \partial y_{(n-1)\_0} \tag{11.7}$$

Let $\Delta x_i = x_{i+1} - x_i$ and $\Delta y_i = y_{i+1} - y_i$. Also, let $\theta_i = \frac{\Delta x_i \Delta y_{i-1} - \Delta x_{i-1} \Delta y_i}{\Delta x_i \Delta x_{i-1} + \Delta y_i \Delta y_{i-1}}$.

$$f_8 = \sum_{i=0}^{n-2} \sqrt{\Delta x_i^2 + \Delta y_i^2} \tag{11.8}$$

$$f_9 = \sum_{i=1}^{n-2} \theta_i \tag{11.9}$$

$$f_{10} = \sum_{i=1}^{n-2} ||\theta_i|| \tag{11.10}$$

$$f_{11} = \sum_{i=1}^{n-2} \theta_i^2 \tag{11.11}$$

Let $\Delta t_i = t_{i+1} - t_i$.

$$f_{12} = \max_{i=0}^{n-1} \frac{\Delta x_i^2 + \Delta y_i^2}{\Delta t_i^2} \tag{11.12}$$

$$f_{13} = t_{n-1} - t_0 \tag{11.13}$$

Features f_1 and f_2 are the cosine and sine of the initial angle α, and highlights how rightwards and upwards the TRACE is at the beginning, respectively. Rather than applying the actual angle, the sine and cosine of α is used instead to avoid the discontinuity as the angle phases through 360° and wraps back to 0°. Features f_3 and f_4 deal with the information about the bounding box. The former describes the length and the latter describes the angle of its diagonal line. This diagonal length is also referred to as the 'size' of the bounding box.

The distance between the starting and ending points of the TRACE is given by f_5, followed by f_6 and f_7 which convey the cosine and sine of the angle between the same starting and the ending points. The cosine of which tells us about the horizontal distance between the two points, whereby if the ending point lies on the left of the starting point, f_6 will be negative. On the other hand, the sine of the angle tells us about the vertical distance, which will make f_7 negative if the ending point lies below the starting point. The total length of the TRACE from its starting to ending points is given by f_8, and the collective total angle traversed is summed up in f_9. While the absolute value of the total angle traversed is defined in f_{10}, the sum of the squared value of those angles is described by f_{11}. Here, f_{11} also tells us about the shallow 'sharpness' of the TRACE, which is needed to distinguish between smooth and sharp gestures.

The last two features f_{12} and f_{13}, calculates the sum of the maximum speeds squared, and the duration it takes to draw the TRACE, respectively. Involving the time factor in the process for identifying gestures help to increase the likeliness of correctly interpreting intended gestures, and indicates that TRACEs are dynamic objects and are not simply static pictures.

11.3 Classifying a Gesture

Once provided with a set of features $F = \{f_1, \ldots, f_{|F|}\}$ extracted from a given TRACE T, we can proceed to classify T as one of the known Gestures g_κ from the collection G. Earlier, we depicted this process of classifying T as a Gesture in the flow diagram in Figure 11.1. The linear discriminator (LD) is the component that decides which of the Gesture class g_κ that T is closest to. Associated with each g_κ is a linear evaluation function v_κ over the features F, given by

$$v_\kappa = w_{\kappa 0} + \sum_{i=1}^{|F|} w_{\kappa i}\ f_i, \quad \text{for } 0 \leq \kappa \leq |G|. \tag{11.14}$$

The classification of T is straight-forward; T is the Gesture $g_\kappa \in G$ whose linear evaluation value v_κ is the maximum over all others in G.

Since the LD will always classify T as one of the Gestures in G, then there has to be a way of possibly rejecting g_κ, in cases where the classified Gesture is either ambiguous or an outlier. That is, a classified Gesture g_κ is *ambiguous* if there is a near tie for the maximum per class evaluation in v_κ, and it is an *outlier* if the computed value of v_κ is numerically distant from the rest of the data.

Rubine's solution to reject an ambiguous classification was to rely on the strength of a probability function. We reformulate it as follows:

$$P(g_\kappa|T) = \frac{1}{\sum_{j=0}^{|G|} e^{(v_j - v_\kappa)}}, \forall j \neq \kappa. \tag{11.15}$$

In other words, given a Trace T, we get an estimated probability that T was classified correctly by the LD as g_κ with Equation 11.15, which works well in practice if the rejection rate is set to $P(g_\kappa|T) < 0.95$.

Statistically, an outlier classification can be determined using the Mahalanobis distance [68], which takes into account the covariance of the variables in calculating distances. Specifically, we are interested in deducing the number of standard deviations a Gesture g_κ is away from the mean of its chosen class κ.

$$\delta^2 = \sum_{i=1}^{|F|} \sum_{j=1}^{|F|} \mathbf{M}_{ij}^{-1}(f_i - \overline{f}_{\kappa i})(f_j - \overline{f}_{\kappa j}) \tag{11.16}$$

Obvious outliers can then be eliminated if we reject any classification whose computed distance is $\delta^2 > 0.5|F|^2$. We describe the inverted common covariance matrix $\mathbf{M}_{ij}^{-1}$ in further details in the next section.

11.4 Training the Weights

In his approach, Rubine adopted a well-known closed formula to produce optimal classifiers under certain strict normality assumptions on a per-class distribution of feature values [107]. So that with the assumptions, the training problem here simply becomes one that is to determine the weights $w_{\kappa i}$, given a set of example Traces related to a Gesture class g_κ, without applying iterative techniques.

Let $f_{\kappa e i}$ be the i^{th} feature of the e^{th} example of the Gesture class κ, for $0 \leq e < |E_\kappa|$, where $|E_\kappa|$ is the number of training examples of class κ. The sample estimate of the mean feature vector per class $\overline{f}_\kappa$ is the average of the features in the class:

$$\overline{f}_{\kappa i} = \frac{1}{|E_\kappa|} \sum_{e=0}^{|E_\kappa|-1} f_{\kappa e i}. \tag{11.17}$$

The sample estimate of the covariance matrix $\mathbf{M}_{\kappa ij}$ of class κ is computed as in Equation 11.18. This is then averaged to yield the common covariance matrix $\mathbf{M}_{ij}$ over all classes in G.

$$\mathbf{M}_{\kappa ij} = \sum_{e=0}^{|E_\kappa|-1} (f_{\kappa ei} - \overline{f}_{\kappa i})(f_{\kappa ej} - \overline{f}_{\kappa j}) \tag{11.18}$$

$$\mathbf{M}_{ij} = \frac{\sum_{\kappa=0}^{|G|-1} \mathbf{M}_{\kappa ij}}{-|G| + \sum_{\kappa=0}^{|G|-1} |E_\kappa|} \tag{11.19}$$

Following which, the sample estimate of the common covariance matrix is then inverted to yield $\mathbf{M}_{ij}^{-1}$, and from which the weights for the Gesture class g_κ are computed.

$$w_{\kappa j} = \sum_{i=1}^{|F|} \mathbf{M}_{ij}^{-1} \overline{f}_{\kappa i}, \text{ for } 1 \leq j \leq |F| \tag{11.20}$$

$$w_{\kappa 0} = -\frac{1}{2} \sum_{i=1}^{|F|} w_{\kappa i} \overline{f}_{\kappa i} \tag{11.21}$$

CHAPTER 12

Gesturing versus Normal Writing

The digital ink domain is unlike any others that can have all its related information easily organised and cross-referenced, and presented in front of users to allow them direct manipulation of the data. We recap that a TRACE simply refers to a trail of digital ink data made between a successive pair of pen-down and pen-up events representing a sequence of contiguous ink points – the xy coordinates of the pen's position. Sometimes, we may find it advantageous to also include timestamps for each pair of the sampled coordinates, if the sampling property of the transducer device is not constant. A sequence of TRACEs accumulates to become meaningful graphics, forming what we (humans) perceive as characters, words, drawings, or commands. Each TRACE can be categorised in terms of the timings noted for its duration, lead and lag times. Furthermore, the TRACEs on the 2D plane group together to form what we perceive as *freehand writings* or *freehand drawings.*

12.1 Temporal Relationships in Digital Freehands

Each TRACE, resulting from a pair of successive pen-down and pen-up events, written on an interactive digital screen can be categorised in terms of the timings noted for

- the *duration* of the TRACE,
- its *lead-time*, and
- its *lag-time.*

Lead-time refers to the time taken before a TRACE is scribed, and lag-time refers to the time taken after the TRACE is scribed.

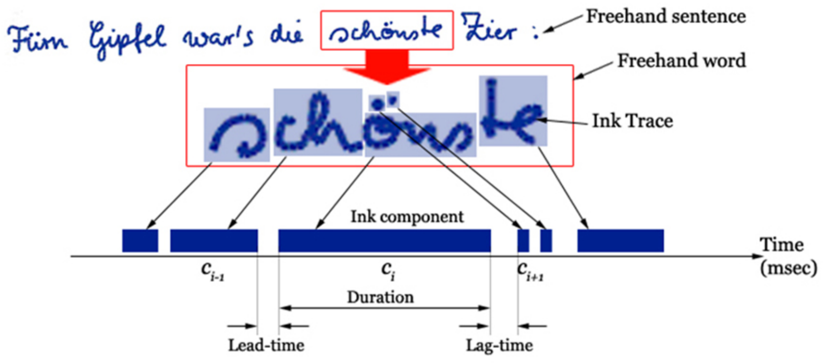


Figure 12.1: Splitting up freehand writing into its ink components on the time line.

12.1.1 Freehand Writing

For a set of contiguous ink components $S_j = \{c_0, c_1, \ldots, c_n\}$ in a freehand sentence made up of n TRACEs, we note that the lag-time for the ith component is exactly the same as the lead-time of the $(i+1)$th component; i.e. $lag(c_i) = lead(c_{i+1})$. Consequently, the timings that separate one set of ink components apart from another are the first lead-time $lead(c_0)$ and the last lag-time $lag(c_n)$ in S_j. These times are significantly longer than their in-between neighbours c_1 to c_{n-1}.

Most people write rather fast such that the time intervals between intermediate ink components in one freehand word is very short. If we observe a complete freehand sentence made up of a group of freehand words, we can categorise each ink component within those words into one of the following four groups:

- **Beginnings** – Ink components found at the start of a freehand word;
- **Endings** – Ink components found at the end of a freehand word;
- **In-betweens** – Ink components found in the middle of a freehand word; and
- **Stand-alones** – Disjointed ink components

The groups differ in the demarcations of their lead and lag times, and as such, provide for a way a pervasive system can identify them. Other forms of freehand writings include mathematical equations, alphabets or characters of various languages, and signatures.

12.1.2 Freehand Drawing

Similar to freehand writings, freehand drawings are also made up of composite sets of ink TRACEs. Here, the diagrams formed on screen are symbolically represented ideas of the writers' that we term as freehand shapes. Often, when using the digital whiteboard as a medium of conveying concepts to audiences, we include hand-drawings (illustrations) made up freehand shapes alongside any writings we may already have.

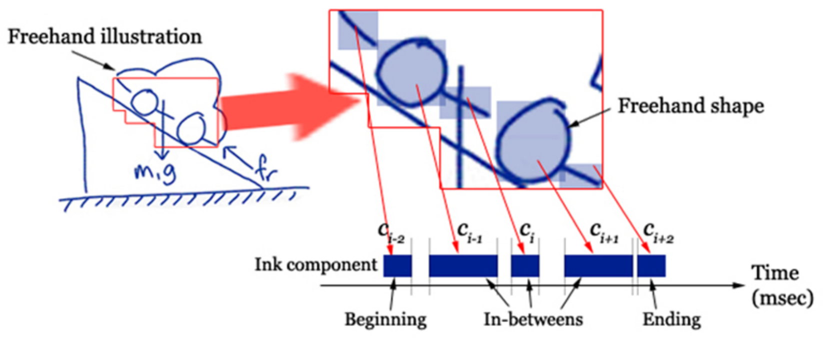


Figure 12.2: Splitting up freehand drawing into its ink components on the time line.

Collectively, a set of freehand shapes is the equivalent of freehand words. As such, the ink components formed by the freehand shapes, when observed by their lead and lag times, can also be classified as *beginnings*, *endings*, *in-betweens*, or *stand-alones*. Figure 12.2 highlights this preconception. Completing this ontology, we further categorise other symbolic forms such as flow charts, state transition diagrams, and artistic and graphical illustrations as freehand drawings

12.1.3 Standalone Components and Gestures

The temporal relation that singles out stand-alone components (and freehand gestures) from their continuous writing and drawing counterparts is the "longer-than-average" lead and lag times of a single-stroke ink TRACE, as shown in Figure 12.3. In this case, there is a pause period between samplings of ink components that results in significantly longer lead and lag times.

As we will demonstrate in a later section, these stand-alone components do not occur as often as we would have predicted. And even when it does occur, the probability of the stand-alone component being a symbolic TRACE is higher than it being a command Gesture. It is not very often that we see people gesturing to a control system in the middle of writing a sentence or drawing a diagram. So we can anticipate, rather convincingly based on the lead and lag times obtained,

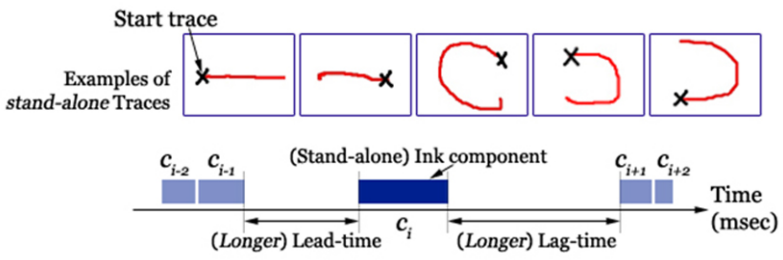


Figure 12.3: Stand-alone ink components that can be interpreted as a possible gesture.

that the latest ink component might be more of an instance of a Trace, rather than a Gesture.

In pursuit of standardising a suitable time-out period, we decided to set the upper limit of both the lead and lag times to 1,500 msec, in retrospect after some initial testing. That is to say, if an observed (lead or lag) time exceeds this period, we immediately consider it as “significantly long” and record it as 1,500 msec.

12.2 Ink States

Advanced recognition techniques have been introduced to the digital ink domain, and while a number of works have gone into handwriting and sketch recognition [107], there are among these works that use the same techniques to attempt to recognise hand-drawn pen gestures [51]. Hence we now have the ‘writing’ mode as well as the ‘gesture’ mode to help out with the current interface issues. We point out here that by ‘gesturing to the system’ we mean the actual gesticulated actions resulting from the pen movements that represent a command, and not merely the ‘tap and drag the icon’ movements. The prospects of gesturing commands in ink-domain applications are encouraging. So much so that many authors think that this is the epitome of the final interface for pen-based applications. Although we agree to a certain extent, that this mode of ubiquitous interfacing may form the fundamentals of future (more enhanced) hardware devices, we do not anticipate this to be the trend in our present context. The Perceptual User Interface (PUI) guidelines should be adopted and used to create the first steps of the bridging between fully packed screens of user interface widgets to that of a ubiquitous one.

Based on the four categories of the temporal observations guarding each ink entity, our state transition diagram in Figure 12.4 illustrates the relations (in states) that an ink component can take. Our previous works show that there exist anticipatory action-events between a trace and a gesture, and that those events are primarily associated with the *ending* and *stand-alone* transitions [83]. A trace has a high probability of becoming a gesture when it is categorised as an *ending*, and is almost always a gesture when it is a *stand-alone*. It remains a trace when it is either a *beginning* or an *in-between*. Similarly, a gesture can be degraded back to a trace should the user decide it to be so even if the system has anticipated a gesture judging from the ink component being categorised as an *ending* or a *stand-alone*.

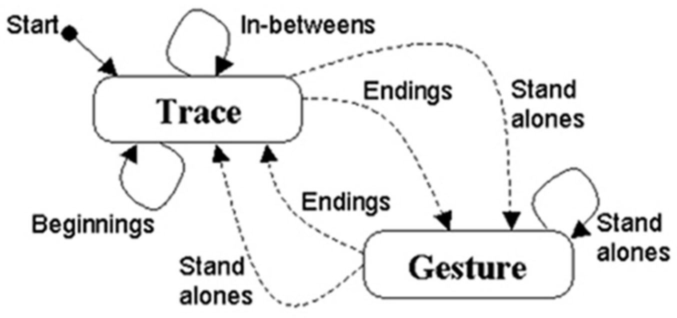


Figure 12.4: State transitions between an ink trace and a gesture.

The two main states, trace and gesture, and the current context of the writings on the digital boards can be immediately scrutinised. We are able to know

- when and where the user is engaged in continuous writings;
- when and where the user is sending commands to the system; and
- when the user is not interacting with the boards at all.

12.3 Sampling Freehand Styles

We tasked 25 people at our institute to write sample scripts on a Wacom PL-550 tablet. These scripts were purposely designed to underline the various forms of symbolic Traces discussed in the previous section as a mixture of both freehand writings and freehand drawings. They constitute paragraphs from a book, mathematical equations, flow charts, state transition diagrams, technical drawings, artistic illustrations, and personal signatures.

Our participants all have their own distinctive styles of writing, differing in the way their strokes and curves are written. Eight of them were left-handed, while the rest were right-handed. They each have different speeds of writing that seem to dictate the legibility and clarity of their written compositions. These writings of several thousand traces are saved as InkML files (based on W3C's InkML standards [108]) that are then further scrutinised into one of the four ink component categories as beginnings, endings, in-betweens, or stand-alones.

12.4 Ink Component Categories

Every single ink component retrieved from the InkML files is observed by their lead and lag times to be classified into one of the four predefined categories. We did this manually for the first 1,000 TRACEs in order to obtain the estimated timing distributions – the mean μ and standard deviations σ of the lead-lag times for all four categories – which are necessary to automate the entire process of scrutiny. The final values of which are shown in Table 12.1.

msec	μ **Lead-time**	μ **Lag-time**	σ **Lead-time**	σ **Lag-time**
Beginnings	1297.95	205.86	635.32	358.66
Endings	218.63	1303.45	370.90	602.34
In-betweens	194.31	200.11	356.78	372.56
Stand-alones	1227.87	1236.92	733.94	764.50

Table 12.1: Tabulated mean and standard deviations of the four categories with respect to the lead- and lag-times.

The values above are then entered into our analyser program that uses the following heuristics to automatically categorise all other ink components. Where the categories Cat(b) refers to *beginnings*, Cat(e) refers to *endings*, Cat(i) refers to *in-betweens*, Cat(s) refers to *stand-alones*, and κ as the scaling factor:

Classify component $[c_g(t_{lead}, t_{lag})]$:

$$\Rightarrow \text{Cat}(b), \text{ if } t_{lead} \geq \mu_{b\_lead} - \kappa\sigma_{b\_lead} \text{ AND } t_{lag} \leq \mu_{b\_lag} - \kappa\sigma_{b\_lag};$$
$$\Rightarrow \text{Cat}(e), \text{ if } t_{lead} \leq \mu_{e\_lead} + \kappa\sigma_{e\_lead} \text{ AND } t_{lag} \geq \mu_{e\_lag} - \kappa\sigma_{e\_lag};$$
$$\Rightarrow \text{Cat}(i), \text{ if } t_{lead} \leq \mu_{i\_lead} + \kappa\sigma_{i\_lead} \text{ AND } t_{lag} \leq \mu_{i\_lag} + \kappa\sigma_{i\_lag};$$
$$\Rightarrow \text{Cat}(s), \text{ if } t_{lead} \geq \mu_{s\_lead} - \kappa\sigma_{s\_lead} \text{ AND } t_{lag} \geq \mu_{s\_lag} - \kappa\sigma_{s\_lag};$$

Here, we use the standard statistical scaling factor $\kappa = \frac{2\sigma}{\sqrt{n}}$, where n is the total number of samples.

There are also times when the above heuristics are not able to categorise the ink components due to the overlapping of ranges. The range-diagram in Figure 12.5 we produce in-lieu of this, illustrates this problem. So, whenever the Classify routine hits this overlapped areas, we got the analyser program to stop and prompt for user-guidance.

Over time and after giving numerous suggested guidance, we noticed certain patterns emerging at every occurrence of our manually adjusting the misclassifications, based on the knowledge of what we saw when data were being collected.

- That, if the previous ink component was categorised as *in-betweens*, then the current ink component should also be *in-betweens*.

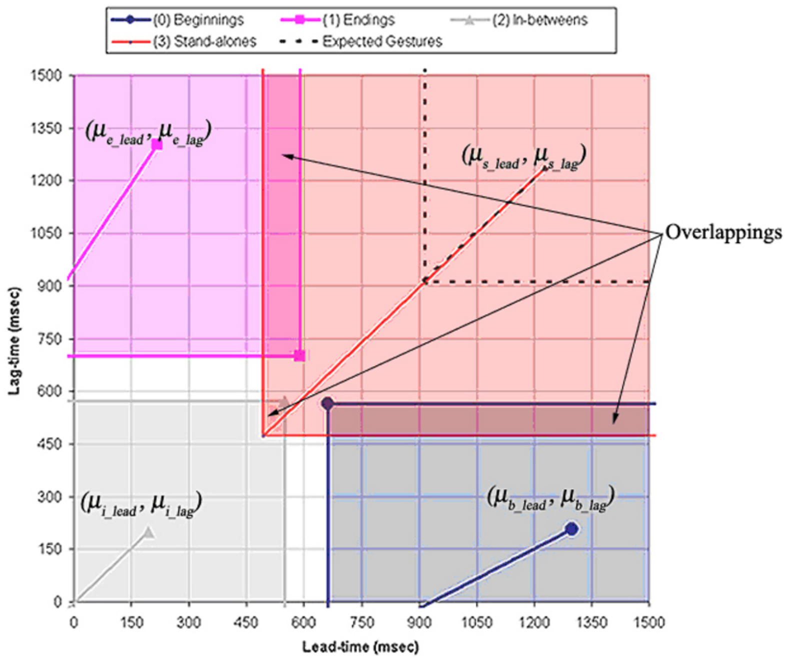


Figure 12.5: Range-diagram depicting the four overlapping categories.

- That, if the previous ink component was categorised as *beginnings*, then the current ink component should be *in-betweens*.
- That, if the previous ink component was categorised as *stand-alones*, then the current ink component should be *beginnings*.

Naturally, we incorporated the three additional rules above into our analyser program, kicking off if and only if the main classifier heuristics fail. This dramatically drops the number of failures and misclassifications.

12.5 Correlating the Lead- and Lag-Times

The classification process in section 12.4 tags each ink component to one of the four designated categories. We noted a strong correlation between the lead and lag times amongst the ink components with respect to them being either *beginnings*, *endings*, *in-betweens*, or *stand-alones*.

12.5.1 Regional Areas

A follow-up to Figure 12.5, the scatter-plot diagram in Figure 12.6 illustrates the actual spread of all the tagged ink components in coordinate relation to their individual lead- and lag-times.

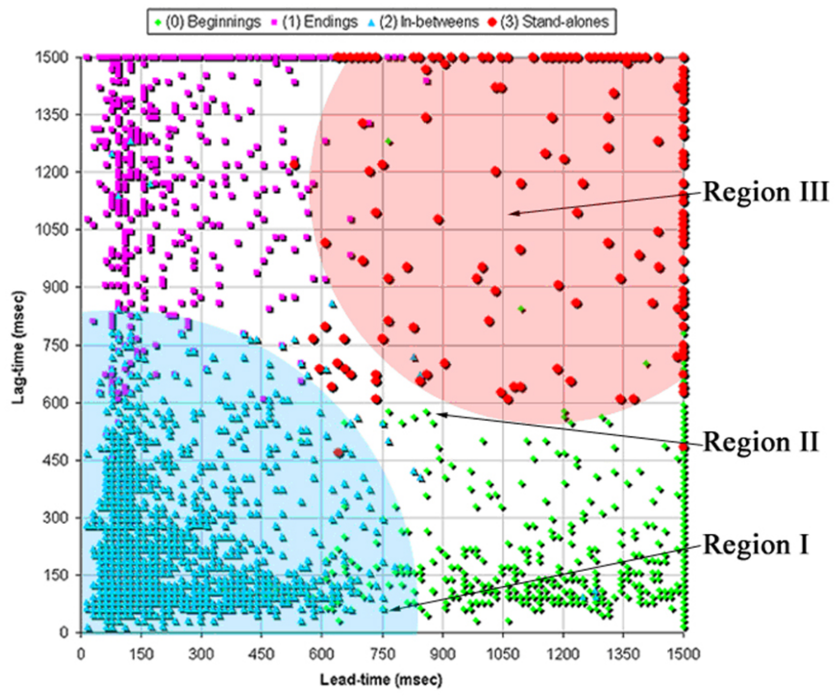


Figure 12.6: Scatter plot of 14,570 pairs of categorised ink components.

We observe three main regions of clustering from the above scatter-plot – the ‘near-neighbours’ in Region I (made up mainly of the *in-betweens*), the ‘extremes’ in Region II (made up mainly of the *beginnings* and *endings*), and the ‘solitaries’ in Region III (made up mainly of the *stand-alones*). The tightness of the clustering decreases as we traverse the plot from Region I towards Region III; that is, where Region I is very tightly clustered, the obvious sparseness in Region III suggests a huge variety of styles written for the stand-alone Traces.

The outline of each regional curve is the estimated resulting decision surface separating the samples between the stand-alones (Region III) and the non-stand-alones (Regions I and II). The attached correlation coefficients r to Regions I, II and III are 0.7699, 0.4819, and 0.1230 respectively. Here, r is the measure of strength of the linear relation between the lead and lag time variables where the magnitude of r indicates the strength of a linear relation, and is calculated with Equation 12.1, where X refers to the lead-time and Y refers to the lag-time variables. For example, a value r close to zero means that the linear association is weak.

$$r = \frac{S_{XY}}{\sqrt{S_{XX}}\sqrt{S_{YY}}}, \tag{12.1}$$

where $S_{XY} = \sum(x - \overline{x})(y - \overline{y})$, $S_{XX} = \sum(x - \overline{x})^2$, and $S_{YY} = \sum(y - \overline{y})^2$.

12.5.2 Categorical Distributions

From all 14,750 pairs of samples (of ink components), we tabulated their relative numbers and found the percentage distributions for all four categories. Figure 12.7 depicts each of those categories, while separating the individual distributions of the lead and lag times. The graph confirm the scatter-plot diagram in Figure 12.6 and prepares the essential grounds for our further investigation in finding out the chances that an ink component is a symbolic TRACE, given its lead- and lag-times.

The stand-alone distribution graph is expected to be of a significant interest to us, as it tells the slim probability whenever an ink component is not a symbolic TRACE. We know that this graph alone is not sufficient enough to strongly predict any user intentions each time we receive a stand-alone ink component. However, combined with the other three graphs, we help reduce the number of processes needed to continuously track all freehand writings on the perceptual user interface environment.

12.5.3 The Bivariate Data

When two traits are observed for the individual sampling units and each trait is recorded in some qualitative categories, then the joint behaviour of two random variables, X and Y from each of the traits, is determined by the cumulative distributive function,

$$F(x, y) = P(X \leq x, Y \leq y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f(u, v)\ dv\ du,$$

derived directly from the fundamental theorem of multivariate calculus [56, 103]. It follows that

$$f(x, y) = \frac{\partial^2}{\partial x \partial y} F(x, y).$$

The graphs obtained in Figure 12.7 hint that our distributions of the lead- and lag-times across all four categories are varying "Normally", with $\mu_x = 462.0059$, $\mu_y = 462.2614$, $\sigma_x = 682.0767$, and $\sigma_y = 682.6080$, where x and y refer to the lead- and lag-time variables respectively.

We find it advantageous to know the joint relation between the lead- and lag-times of each ink component for making obvious generalisations in the future. A lookup probability table constructed from the current samples of categorised

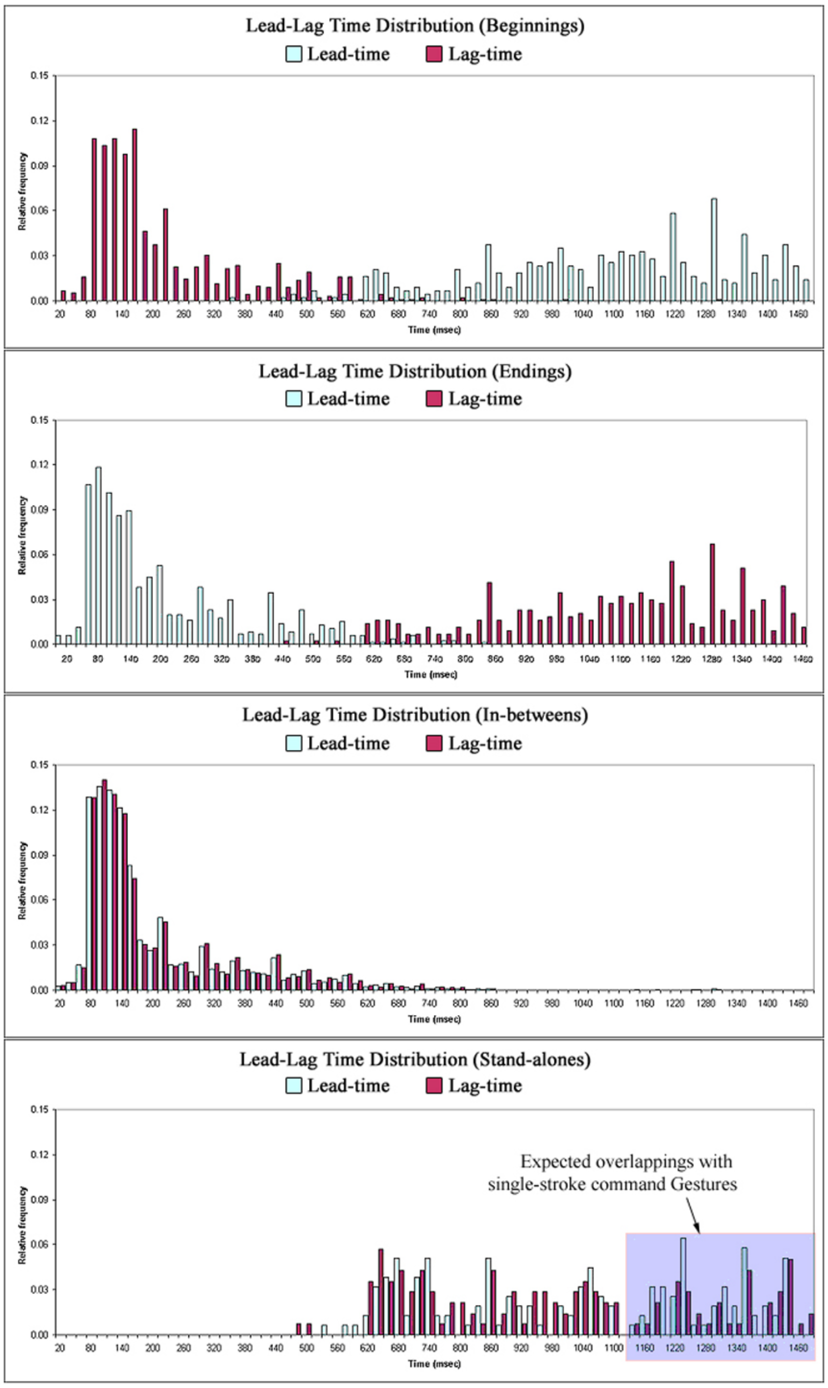


Figure 12.7: Lead-lag time distributions for *beginnings*, *endings*, *in-betweens*, and *stand-alones*.

ink components, following a bivariate normal density, is expected to expedite our mission of finding the quick probability for a symbolic Trace to remain in its own entity. Equation 12.2 describes this density function, and Figure 12.8 illustrates the lookup table graphically.

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp[-\frac{z}{2(1-\rho^2)}] \tag{12.2}$$

$$\text{where } z = \frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}$$
$$\text{and } \rho = cor(x,y) = \frac{\sigma_{xy}}{\sigma_x\sigma_y}.$$

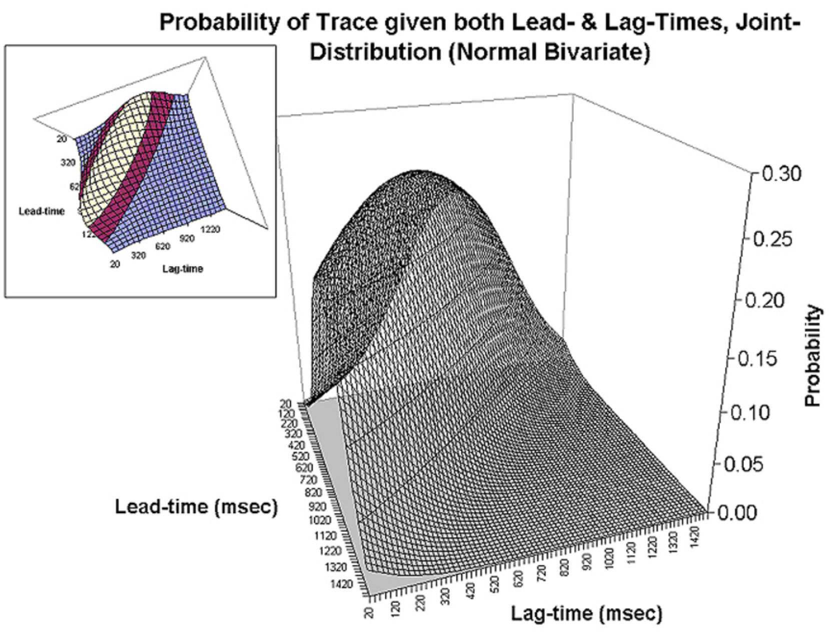


Figure 12.8: Joint distribution via the normal bivariate density.

For every section on the surface in Equation 12.2 that gives a high probability that an ink component should be a symbolic Trace, there are other sections on the surface that depicts otherwise. These sections confirm our scatter-plot diagram in Figure 12.6, and are the composite distributions of all the graphs in Figure 12.7.

The next section details the use of the graph in Figure 12.8 through the formal formulation of our statistical hypotheses in making proper inferences of the population samples.

12.6 Statistical Acceptance

Broadly speaking, the goal of testing statistical hypotheses is to determine or conjecture about some features of the population, which is strongly supported by the information obtained from the sample data. Based on key statistical concepts in the context of determining the best solution for our problem definition, we establish the alternative hypothesis H_1 and its nullifying opposite H_0 as stated below.

H_0 : "That an ink component is **not** a symbolic Trace."

H_1 : "That an ink component is a symbolic Trace."

Where testing each hypothesis is carried out as follows, where X and Y refer to the lead- and lag-time variables respectively.

$$\mathrm{H}_0 : \overline{X} \geq \mu_x, \ \overline{Y} \geq \mu_y$$

$$\mathrm{H}_1 : \overline{X} < \mu_x, \ \overline{Y} < \mu_y$$

12.6.1 Test Statistic

Our test statistic refers directly to the joint-distribution described in Equation 12.2 (and its Figure 12.8 equivalent), the Z-test forms the rejection criteria needed for a test level of significance $\alpha = 0.05$ (and that $\alpha/2 = 0.025$). Figure 12.9 demonstrates this notion from two cross-sectional views of the normal bivariate density curve.

We accept H_1, if and only if $R : |Z| > Z_{\alpha/2}$, where R refers to the rejection regions highlighted in Figure 12.9, and $Z_{\alpha/2} = Z_{0.025}$ is the boundary of acceptance on the joint distribution of the normal bivariate density graph shown in Figure 12.10. Otherwise, we reject H_1 and claim H_0 that an ink component $c_g(X = t_{lead}, Y = t_{lag})$ shall not be considered as a symbolic Trace.

12.6.2 Inferring the Bivariate Data

Plugging in the values of $\mu_x = 462.0059$, $\mu_y = 462.2614$, $\sigma_x = 682.0767$, and $\sigma_y = 682.6080$ for every time-step of X and Y from 0 to 1500, we obtain the entire region of acceptance (and rejection) on our bivariate surface. The darkened regions in Figure 12.10 are the areas for 'rejection' while the lighter region is the area for 'acceptance'. It is made to correspond directly to our lookup table, with the input parameter $c_g(t_{lead}, t_{lag})$ and retrieving from the table an output probability of whether any ink components should be considered as symbolic

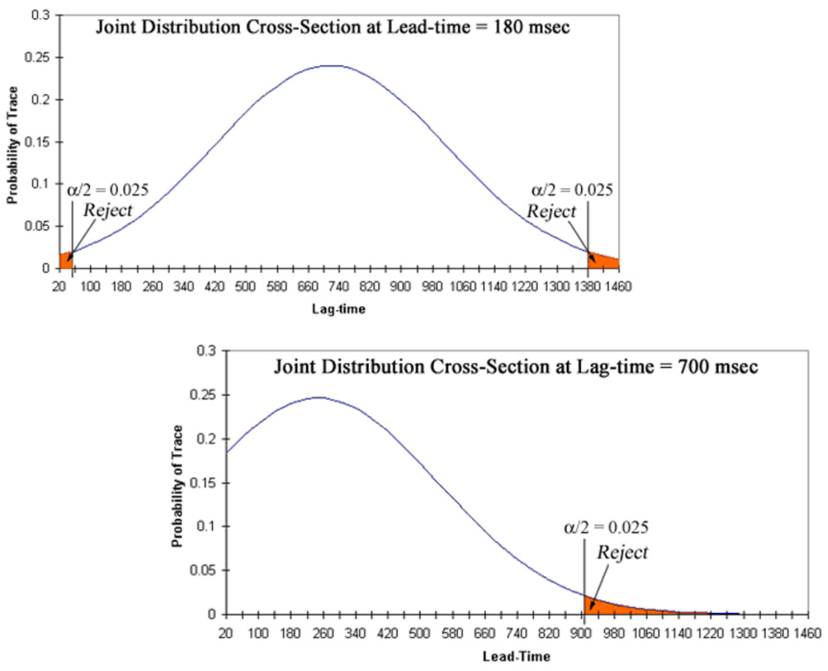


Figure 12.9: Cross-sectional view of the joint distribution of the normal bivariate density at lead-time = 180 msec and lag-time = 700 msec respectively.

Traces given its lead and lag times (i.e. $P(\text{Trace}|c_g(t_{lead}, t_{lag})))$, with an attached H_1 (strong acceptance) or H_0 (strong rejection).

We give two examples in Figure 12.10. The first, with an input parameter of $c_1(1377, 1281)$ gets a probability value of $P = 7.38 \times 10^{-08}$ and recommended for 'rejection'. This means that the likelihood of the ink component c_1 being a symbolic Trace is very slim, and further tests should be made to check if we could indeed upgrade it to a command Gesture. The second, with an input parameter of $c_2(309, 1011)$ receives $P = 0.1164$ and recommended for 'acceptance'. This is a clear-cut case that the ink component c_2 should definitely be considered as a symbolic Trace.

12.6.3 Chances of Getting a Gesture

We mentioned in section 12.1 that the probability of occurrence of stand-alone ink components is very small. We also made the assumption that all command Gestures be categorised collectively as stand-alones to fit our timing definitions. Our preliminary analysis of the sample data indicates the findings shown in Table 12.2.

We iterate again that it is not very often that people choose to Gesture in the

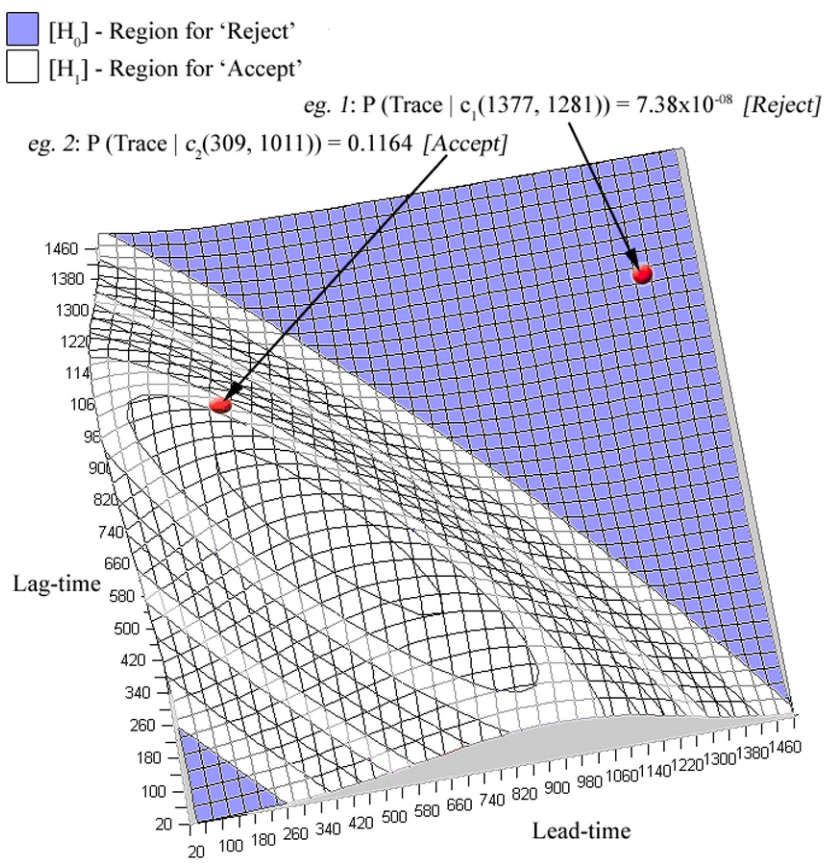


Figure 12.10: Incorporating the normal bivariate density to assist with the null and alternative hypotheses.

midst of writing sentences (or drawing pictures) on the board. They only Gesture to invoke commands to correct mistakes, access files, edit current writings, and browse between canvas screens. Table 12.2 proves this claim, as we see that command Gestures make up only 1.21% of all stand-alone components (which makes up 22.49% of all sampled ink components).

The hypotheses graph depicted in Figure 12.10 highlights areas for rejection that seem a little too large. However, based on the facts in Table 12.2 and of our dissection of the scatter-plot diagram in Figure 12.6, we believe that the lookup table obtained from the hypotheses graph is true and concise enough to make reliable decisions for all future ink components.

Total ink components sampled	14,750	(100.00%)
Stand-alone components (incl. of command Gestures)	794	(5.38%)
Command Gestures ONLY	179	(1.21%)

Table 12.2: Breakdown of ink components with focus on stand-alones.

CHAPTER 13

UI-on-Demand and Within-Reach

By now, we have fully investigated the lead- and lag-times of digital freehand writings and drawings, and found that there exist a direct correlation of the lead-lag times (categorised as *beginnings*, *endings*, *in-betweens*, and *stand-alones*) and the probability of the ink component being a symbolic TRACE. Our final lookup probability table, which is the resultant joint distribution of the lead and lag times through the normal bivariate density function, gives very reliable outcomes of accepting or rejecting the chances of ink components as symbolic TRACEs.

The digital ink environment is unlike the desktop metaphor that we know and have grown used to. A digital board is more personal than a monitor, in a way that it allows us to directly touch it and see the immediate reaction through the digital pen. In classrooms where huge wall-mounted digital boards are utilised, it is impossible to have a bird's eye view of the whole environment when we are so near their proximity to impose the conventional desktop interfaces. Let us now look at an alternative solution that seeks to combine three classical UI-design concepts involving adaptable, adaptive and perceptual ideas for an on-demand-and-within-reach interface that monopolizes the single-stroke digital ink inputs. By maintaining an active background tracking system on the temporal and spatial information of every input ink data, we can anticipate convincingly what users are trying to do to achieve their goals when working on the boards, and provide them with assistance within predictable contexts. We want to maintain within the methodologies that users must always feel in control no matter how adaptive the assistance from the system may be.

13.1 Perceptual User Interface (PUI)

Our strongest constraint when working towards an appropriate interface for these huge boards is to maximise on a single input modal, namely the digital pen.

Recently, the digital ink has received more attention as progress is made in optimising its data structure and storage methodologies [123, 96, 20]. A shift in the writing paradigm has led many to believe that we can leave the raw input ink as ink representing itself [1, 6]. As, in the first place, content readability would depend greatly on the user's ability to write clearly (and that very good handwriting recognition technology is vital to not misinterpret any freehand writings on the boards). What's more, because of this perceptual thinking, many have looked at the physical properties of the ink features and derive from them natural gesticulated commands [107]. Few works show the successful merging of the 'writing' mode and the 'gesture-command' mode on a single, non-segregated platform, without having escape sequences or using the button on the digital pen to switch between the two modes [51]. However, those works are not meant for the interface-interaction techniques with huge wall-mounted digital boards, and that the user interfacing issues of such have yet to be fully exploited. Here, it follows that Turk and Robertson's [123] Perceptual User Interface (PUI) approach seems to be the bridge we are looking for.

PUIs seek to combine an understanding of natural human capabilities with the actual device medium and the related machine perception and reasoning. The goal is to make the user interface more natural and compelling by taking the advantage of the ways people naturally interact with each other and with the world. In our case, we want the user to feel most comfortable using the electronic board as if it was a chalkboard. And then, without having to always search for a command from the pull-down menus (at some fixed screen location), indicate as naturally as possible by gesticulating anywhere on the board, of 'what we think the command might look like'. This is as opposed to forcing users to memorise 'what gestures are associated to which commands'. This manner of communicating with the hardware, according to the PUI standards, is expected to be transparent and passive, and the machine in turn should perceive this relevant human way of communicating, and react by generating graphical or visualization outputs that would be naturally understood by the user.

13.1.1 Writing Space versus Interactions

The abundant writing surface allowed by huge wall-mounted digital boards (see Figure 1.4) is harder to perceive for user-machine interactions as compared to the comfortably sized tablets and office whiteboards. Freehand writings are an entity on their own, containing the flair and style of expressions of the writer, which do not necessarily follow an ordered sequence or format when ample space is provided. To date we see a number of digital ink-based studies that concentrate on managing contents and editing 'informally' [1, 51], which came about as a result of this entity. Whether it is grouping a set of freehand words into a continuous flow of sentence, or combining a set of drawings to form the physics of represented objects, the programs rely on a lot of command buttons in toolboxes and pull-down menus to affect their causes. And since all of these applications are tailored for the desktop setup (including the tablets), the authors need not mention any proximity problems.

Try putting these same applications on the huge wall-mounted digital boards and we can expect users to complain about its lack of suitable interface.

The point is, if we can get the system to anticipate correctly what its users are trying to do to achieve their goals, by providing assistance-on-demand and within-reach just by analysing the input ink elements, then we would be one step closer to realizing the most appropriate user interface that properly combines issues of adaptability, adaptivity, and those mentioned in the PUI standards, for our huge wall-mounted digital boards. As George Santayana (1863 – 1952), a Spanish-born philosopher once said, “Science is nothing but developed perception, interpreted intent, common sense rounded out and minutely articulated.”

13.1.2 Adaptability versus Adaptivity

Interfaces that are *adaptable* allow users to customise the application to suit their comfort needs and this is seen strongly in Shneiderman’s stand for “direct manipulation” [108, 115]. By putting everything on screen in a certain orderly fashion and relying on the human hand-and-eye coordination, the interface design stresses on the convenience of users to achieve their goals through their own actions. The interface provides a top-down global approach environment for the users, and a lot of efforts are placed in this “art” of direct-manipulation on the designers to support complex information management tasks [75]. In addition, users can customise the default interface widgets for an overall screen outlook that they can be most comfortable with.

Adaptive interfaces on the other hand, model the individual user’s interest, preferences and usage characteristics of the application to allow the interface to tailor itself to each other. The interface takes a proactive stance in assisting users and often involves background agents. It operates best when there is indeterminism about user actions and events within the environment and when there is no viable prospect to organise all data at hand. We agree with Maes [116] that agents are not an alternative for direct manipulation, but are the complimentary metaphors; very good application interfaces are needed for agents to be effective as they are not substitutes for the interfaces.

But by combining adaptivity and adaptability, we do find added advantages for intermediate and advanced users as reported by Bunt *et al.* [11] in their user-analysis studies that involved the application of Fitt’s and Hick’s laws and the GOMS model. Particularly, they mention that “ reduced interface is bound to be more efficient” is what triggered our own research in developing an appropriate interface for the huge wall-mounted digital boards in classroom situations

13.1.3 Lack of Interfacing Mechanisms

The Wacom tablets and the Tablet PCs place the desktop computer screens literally on the desktops and allow for one extra mode of input; the digital pen. This hardware innovation solves the problem of referencing between separated slides and physical writing areas by placing both (windows) adjacent to each other. Annotating on the slides or documents is made more accessible as described clearly by Bargeron and Moscovich [6]. Further more, by using appropriate software, all presentation data can be saved and retrieved for viewing at a later time.

Wall-mounted digital boards, utilised in the education and corporate sectors, are also almost always connected to an operating system that impose the 'desktop' interface to their users across all board-application programs. There are, however, a few others with no clear-cut user interfacing mechanisms and rely strictly on the actual hardware devices to simulate a typical whiteboard. We can parallel arguments of *adaptability* and *adaptivity* of the interfaces to the current literature because all these wall-mounted boards are being handled from another console; usually a desktop setup. When this happens, we feel that the wall-mounted boards are just the more expensive versions of the beamer-projectors, and it seems wasteful that we do not fully utilise the functions available on the digital screens. As a matter of fact, the screens themselves are able to receive direct inputs from users albeit their awkwardness in size. We attributed this to the lack of proper interfaces for those screens, or if a good one exists, then it is not completely dependable on just the digital pen input, which may be most convenient for users of this device.

Unlike desktop computers, wall-mounted digital boards are meant to be an active presentation medium for a congregation of audience in a large room. While there is always a desktop console that can control the wall-mounted boards, through the mouse and keyboard, we want to free up this mode of interfacing and let the user (who is usually the presenter) communicate directly with the screen using just a digital pen. Judging by the dominant metaphor that what we have today is a mismatch for the computer environment we are dealing with tomorrow, it is difficult to place this huge screen scenario (alone) into any pronounced categories of user interface debates on concepts of *adaptivity* and *adaptability*. For instance, the sphere of influence of the hand-and-eye coordination needs to be enlarged, and perhaps include the body, to follow the claims of direct manipulation of adaptable interfaces. Furthermore, while the audience have it easy watching from a distance, the presenter does not: standing so close to one part of the huge screen (or screens) often leads to the interface widgets he or she may require to be out of reach, or worse still, because the presenter cannot see those widgets, he or she may assume that such actions or commands represented by those widgets do not exist within the board application. We point out here that this may affect the flow of the lecture presentation. In this case, we may be left to rely on the adaptivity of the interface for the wall-mounted boards to proactively assist the user while at the same time ensuring that the user still feels in control of the whole process.

13.2 Active Background Tracking

Fewer widgets on screen may turn out to be more effective as hypothesized by Bunt *et al.* [11] especially for these types of situations. For if we want to reproduce the physical chalkboard on the digital boards, and still be able to take full advantage of all the capabilities that the digital system can offer, we may need to compromise on the number of interfacing widgets so that the instructor can focus on his or her own lesson delivery and pay less attention to the background technology. In fact, one of the goals of creating a practical interface for the boards is to give as much writing space as possible to the users, and to adopt 'natural' hand-gestures whenever appropriate to invoke only the necessary commands and tools given the context, while engaging in a lecture presentation.

Part of our solution of minimizing the numerous interface combinations of toolboxes and command buttons is to spruce up the background environment with active and reactive mechanisms – placing interface agents as collateral to borrowing processing time from the system. Since Jameson [54] claimed that there are no conclusive proofs that adaptable interfaces are always superior, our system's active background components are also designed to tolerate adaptive user characteristics and tasks, as well as perceptual events received from the ink environment of the huge wall-mounted digital boards, solely from the pen-based input modal [88].

13.2.1 The Background Model

There are five components that make up the active background model to track ink inputs perceived from the board environment as shown in Figure 13.1. The Trace Agent, Gesture Agent and the UI Controller all receive the same information from the digitiser. The Trace Agent classifies each ink component into one of the four temporal categories and determines statistically the state of the ink. The Gesture Agent uses the Trace Agent's verdict and its own percept of the ink environment to determine the kind of gesture (command) to tell the foreground application, should the ink component not be a trace. The UI Controller provides the appropriate graphical visualization of the user input, drawing its instructions from the foreground application by leaving trace marks on the screen or showing on-demand menu options near the user's current position with respect to the boards.

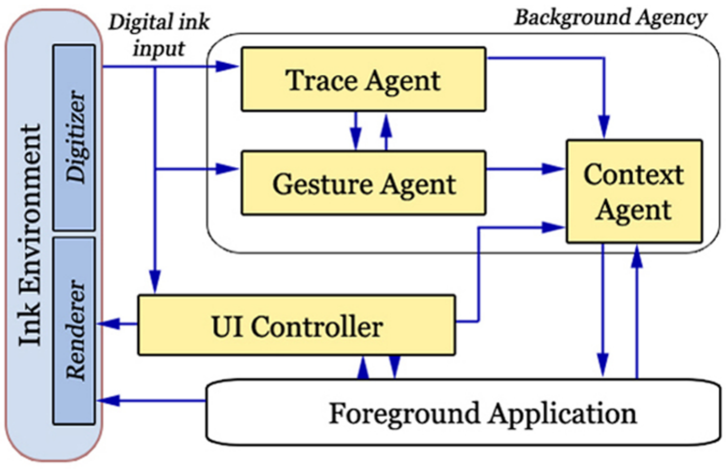


Figure 13.1: Model of the active background involving the ink environment, an agency, and a foreground application.

A Context Agent monitors feedback interactions between the user and the system. This agent communicates internally with both the Trace and Gesture agents to determine all spatial and temporal ink activities on the ink environment. It also requests further information from the UI Controller and the Foreground Application to confirm the ink states and to double check its own data collection of previous events to better anticipate current and future ink activities [87]. Among the agent's more notable properties are:

- Determining the consistency of the freehand writings;
- Determining the frequency of gestures with respect to the writings;
- (Assisted) Acting and reacting to the foreground applications; and
- Deciding the most appropriate context given any (writing) situation.

In terms of coming up with the appropriate context for a particular digital board application, the agent works on the initial information of the current ink state. Here, traces are straight-forwardly associated with writings (or drawings) and then depending on the user's editing choices, the agent sets the 'context' to allow specifically for changing pen attributes, canvas attributes, or any other annotating activities. Single-stroke gestures are tagged to the more specific menu items for browsing the boards (and/or slides) contents, accessing files, invoking help, as well as performing editing commands such as copy, cut, paste, insert, delete, etc. On the forefront, this resembles very much like the popup menu ideology, but with the exception of dynamic menu contents reacting on the same set of gestures and traces. In our case, the contents are dependent on the current user situation.

13.2.2 Strategy

Turk and Robertson [123] proposed a strategy for designing an effective interface that we find satisfies issues of adaptability, adaptivity, and perceptuality. He concentrated on three key components:

- Leveraging human capabilities;
- Exploiting the available technologies; and then
- Accessing and manipulating all information as we see fit.

This is in relation to the Seeheim model pointed out by Morse [90], which highlights the "functional core" that directs the "presentation component" and the "dialogue controller".

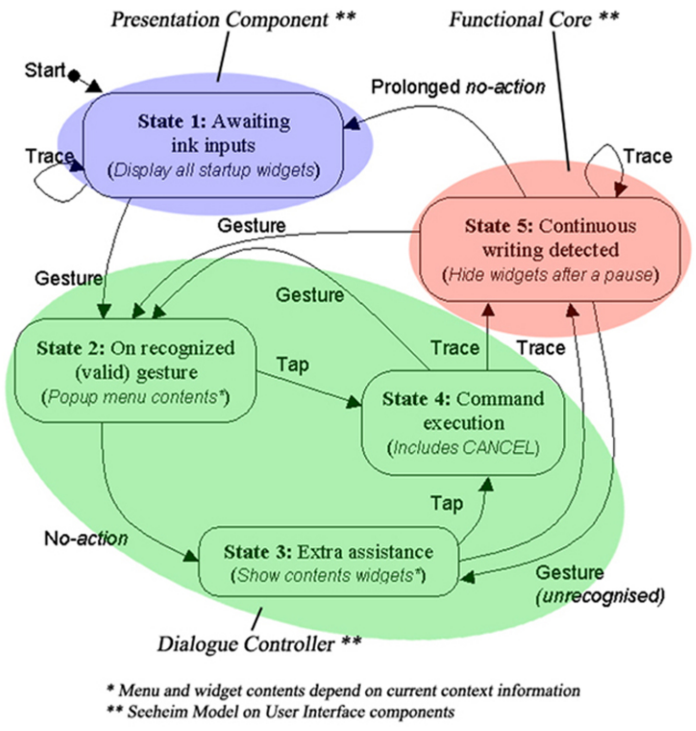

Presentation Component **
Functional Core **
Start
Prolonged *no-action*
State 1: Awaiting ink inputs (*Display all startup widgets*)
Trace
Trace
Gesture
State 5: Continuous writing detected (*Hide widgets after a pause*)
Gesture
Gesture
State 2: On recognized (valid) gesture (*Popup menu contents**)
Tap
Trace
Trace
State 4: Command execution (*Includes CANCEL*)
Tap
No-*action*
State 3: Extra assistance (*Show contents widgets**)
Gesture (*unrecognised*)
Dialogue Controller **
* Menu and widget contents depend on current context information
** Seeheim Model on User Interface components

Figure 13.2: Tracking ink events and reactionary stratagem.

The state transition diagram in Figure 13.2 encompasses the above concepts, as well as incorporating our active background-tracking model that optimally manages a way of communicating with the user in a purely digital ink environment. Five transition states are based upon the input ink states received (either as traces or gestures), or the lack of it (either as tapping on the board, or no-action, or prolonged no-action) with the help of a pervasive timer mechanism for this reactionary stratagem.

As a rule of thumb, we leaned towards the play-safe strategy; for whenever there is a "doubt" (a level of *un*assuredness beyond a certain threshold value [83]), the user must be given a choice to see and select from the full-range of commands offered by the foreground application. This is prominently reflected in State 1. States 2 and 3 are on-demand menus and inter-face options proffered by the system based on the current written ink context. Only the selected (anticipatory) widgets are displayed on screen while all other options that are deemed "not-in-context" are packed under a command button marked "More >>". Referring to the fact that we want to provide the instructors with an uncluttered writing canvas, the interface widgets will disappear once the system detects continuous writings. This strategy works on all our board scenarios including those that we use for purely "writing" applications, as well as those that integrate lecture slides in them (see Figure 13.3). The emphasis of which is directed towards a single digital pen input by the user to put him or her in control of the class and the technology within.

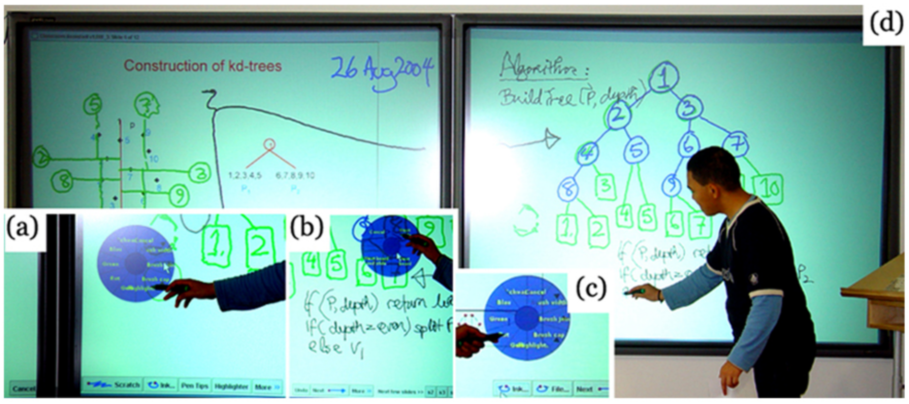


Figure 13.3: Huge wall-mounted digital boards as teaching medium. Insets: (a) dynamic-content pie menu for pen attributes, (b) dynamic-content pie menu for browsing slides/board-canvas, (c) bottom toolbar widgets within reach and in-sight of user, (d) normal freehand writings left as trace marks on the ink environment.

13.2.3 Deriving Interfaces from Inputs

To complete the visualization of the on-demand assistance, we bank-in on the last known coordinate position of the user's latest written Trace (or Gesture).

Popping up dynamic-content pie menus and/or other static command buttons in that vicinity seems most appropriate and convenient for the user. Earlier research on pie menus proved an advantage for users of digital screens [13], as people tend to better remember the cyclic position, and as such expedites the selection process. Our command button (toolbar) interface is logged to the bottom of the screens, appearing on-demand with only an array of anticipated widgets (see Figure 13.3 for examples). The combined lengths of the number of widgets will not exceed the arm's length of the instructor to ensure that all widgets are within reach and within the instructor's line of sight.

13.3 Findings and Evaluations

To measure the response of users who utilize the huge wall-mounted digital boards, based solely on the digital pen inputs, we collected and analysed all ink information received in classroom lessons for a period of one semester week. The digital boards were set up to record the amount of writings by instructors, and to see how much of those writings are 'meant' and 'interpreted' as command gestures or for invoking on-demand menu options. The data are then used to infer how well our support system reacts in providing valid and useful assistance to instructors. While the system records all the above performance data, we also stationed a human observer for each lesson that utilized the digital boards. The sole purpose is to track "human errors", where we define this to be the changing of the instructor's mind on a 'set action'. For example, after purposely calling up an on-demand menu, the instructor decides that the menu is not needed after all (subsequently, the Cancel option was selected), or the instructor decides that he or she really wanted to access a different option that is completely out of context (subsequently, the "More >>" button was selected).

13.3.1 Inks, Commands, and the UIs

There was a total of 125,866 ink inputs received and processed by the digital boards during our observation period. All lessons had PowerPoint presentation slides embedded on the screen. Instructors were able to browse the contents of the slides and the boards, and write anywhere on the screen they please. We noted that over 94% of the digital inks collected were freehand writings (and drawings), and only 5.15% were anticipated by the system as gestures for commands and for invoking the hidden (on-demand) interface. Figure 13.4 depicts the further breakdown of the command gestures and the extent the correct anticipation led to the invoking of the appropriate UIs. "Human errors" amounted to 12.43% of the system's anticipation process. This is as opposed to the system's own error in judgment where it either reacted with a wrong command or interpreted the wrong context, which were all due to the misinterpretation of ambiguous single-stroke gestures. This is a prevalent problem that we highlighted in our previous works [87, 83] and is also explained by various authors [51, 90], especially in cases where the legibility of the instructor's freehand writings are by themselves questionable.

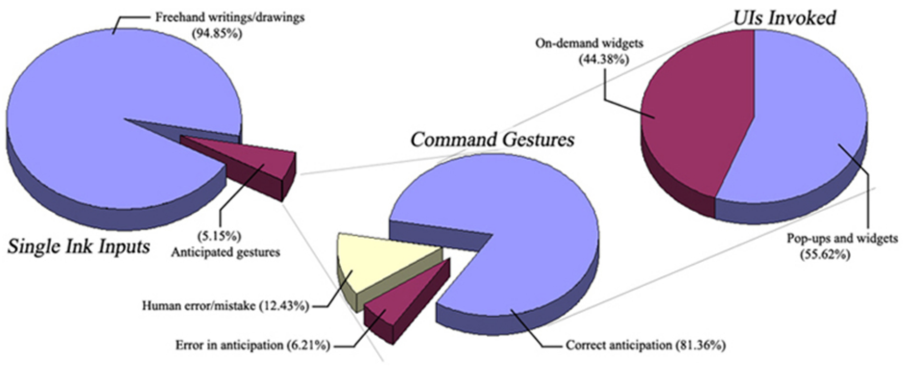


Figure 13.4: Deriving interfaces from single-stroke digital ink inputs.

Our data shows that of all the correct (assisted) interpretation made by the system in provid-ing the interface-on-demand, more that 55% of the time involved popup pie menus. The rest of the time required the appearances of toolbar widgets near the instructors. Our observers noted that the instructors took the advantage of the absence of tool-widgets and naturally utilized the boards as if they were chalkboards – writing everywhere and anywhere they please. The instructors did not seem to mind the lack of visible technology, as to them the system seems to conjure up the interface whenever it is that they need to use it. Making choices from the anticipated menu options gave users the impression that the boards have always been in their control.

13.3.2 Users' Choices

To obtain a better picture of how users take to having their menu options anticipated by the system in whatever context they are currently working on, and then presented to them on-demand, we also made provisions to monitor the kinds of widget options users click. The graph in Figure 13.5 depicts this finding.

The first two bars highlight the frequency of users' choices whenever any options were selected from the bottom toolbar. Our system was able to correctly anticipate the context in which the users are working more than 86% of the time. That is, within the limited number of command buttons allowed for the arm span of the users, the system is able to place at least one button that the user can tap for immediate action. Only 13.45% of the time do users have to locate their commands buried inside the "More >>" button. The last three bars show the number of times users click on either the static pull-down menus, the on-demand widgets (bottom toolbars) and the dynamic-content popup-pie menus respectively. Over half the time we see users utilizing the convenience of the popups and 42.02% of the time on the on-demand widgets.

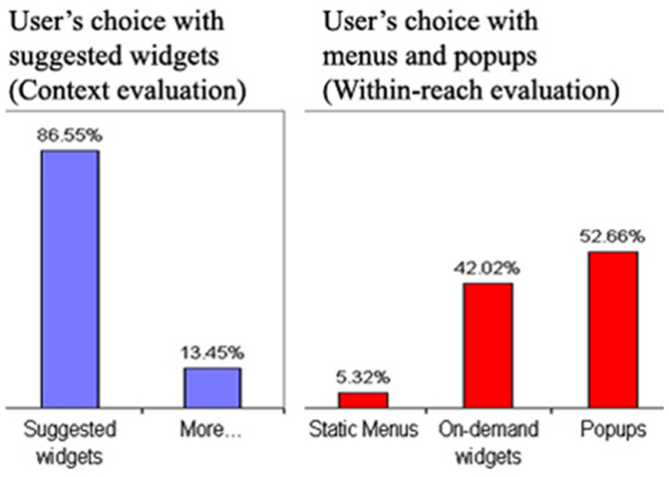


Figure 13.5: Users' choices on widget options.

CHAPTER 14

*Dis*oriented Gestures for TableTop Environments

We have yet to find a multi-user interaction model developed for the tabletop displays that handles the notion of "who-is-doing-what-and-when-and-where", without having to rely on external sensing devices or imposing hardware to be worn by the users. The support mechanisms in place for current tabletop systems stress considerably more, among others, on ubiquitous table rules for appropriately displaying and orienting screen-items (widgets), effective usage guidelines for practical UI supports, and convenient tools for multiple document management [28, 100, 124]. These supports are for general manipulation purposes; explicitly tailored for tabletop users such as the picking, moving, shoving, and rotating of active widgets on the tabletop, and the defining of users' public and private work areas.

14.1 The TableTop Environment

In 1993, Wellner [132] presented and discussed the technique of merging physical papers with digital documents and other onscreen widgets onto a common desktop surface and make them all manoeuvrable directly by hand. This technology of "computerised reality", or some-times known as "mixed-reality", has since then caught on, progressing from a simple, single-user-setup that is attached to a single projector, a camera, and a computer, to that of a more complex multi-user-arrangement that is capable of facilitating both collaborative and cooperative activities while aiding users in their quest to solve more important tasks. Most of these interactive tabletops are designed to conform with the Computer Supported Cooperative Work (CSCW) standards, which have the main goal of using technologies of computer networking and associated hardware, software, services, and techniques to support (and to a certain extent, understand) the way people work in groups [38, 61, 74, 106, 72].

Research for the tabletop displays and the related designs of CSCW environments to encompass multiuser interactions made way for further developments in both hardware innovations and software interfacing models. We see new and hi-tech tabletop equipments presented to the scientific community almost every year since 1994. Many are coupled with properly positioned sensing devices such as cameras and motion detectors, while few others come with human-contact-type sensors [59, 28, 100, 124]. The tabletop displays themselves have recently evolved with built-in sensor architectures smart enough to recognise multiple hand positions and allow synchronous inputs [100, 135]. These are held together by control and cognition algorithms that provide the backbone support for all activities on the tabletop systems. Such assistance form the software programs bridges the interfacing problems between the tabletops' four-corners and the simultaneous interface for multiple users [60, 61].

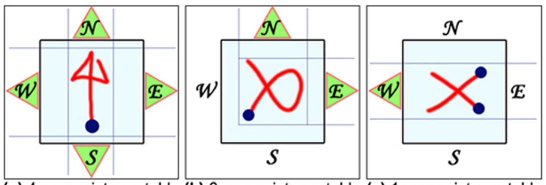
However, we often note that no significant efforts can be found in the literature to determine which of the user around the table is currently and actively interfacing the tabletop; suffice to know that "someone" is touching and manipulating the onscreen widgets. For even with the help of cameras, techniques to differentiate the users are complex, and subsequently, the cost of such complexities may affect the overall multi-user tabletop-interaction profoundly in terms of reaction times (latency) [127, 29].

Our solution in attempting to correctly recognise which of the users seated around the table-top display (and as a consequence get the interface program to work or react in favour of the identified user) lie in monitoring closely the input command gestures received by the table-top operating environment. This is as opposed to placing biometric sensors on the users, or investing on special tabletop sensors that send signals through users upon their touch.

14.2 The *Dis*oriented Pen-Gestures Approach

In the example from Figure 14.1(a), the "arrow" drawn on the tabletop is an "up-arrow" to a user seated in the South corner, but a "right-arrow" to a user in the East corner. Depending on who drew it, our method will tell apart, whether the gesture is a `STH_UP_ARR` or an `EST_RT_ARR` accordingly. Because of this attached corner-identity in the recognised gesture-command, the foreground application can simply make use of this information to (perhaps) continue processing and managing the tabletop UI for better communication controls in favour of the user at the identified corner.

Our approach to identify users seated around the tabletop display without relying on external sensing devices is to bank-in purposefully on the characteris-



(a) 4-corner interpretable (Single-stroke) **(b)** 2-corner interpretable (Single-stroke) **(c)** 1-corner interpretable (Multi-stroke)

Figure 14.1: Similar in appearance but differ in by-corner-interpretation of single and multi-strokes.

tics of the gestures received. That is, we emphasise our observation on the way users gesture commands to the system with their digital pens (or their fingers). We discuss our method in this thesis to show that we can relate these gesturing-techniques to any of the four corners of the table by virtue of the "orientation properties" contained in every gesture made on the tabletop environment. All pen-gestures are made up of various identifiable "trace-features" in spatial and temporal domains [83]. This set of trace-features is used, in conjunction with a primed set of related trained-weights in classical linear gesture-classification algorithms, to recognise input traces as possible gestures.

By carefully selecting a set of key features from the spatial domain and combining them with another set of key features from the temporal domain, our method draws strong conclusions of how dis-oriented a particular trace is. These are mainly the variance in angular and velocity-type features that we use - within the "dis-orientations" - to relate a trace to one of the corners of the table. This is the corollary of the original linear gesture-classifier theorem [107]; that a gesture is recognisable through features that describe the nature of its trace out-line by means of multimodal distribution, meaningful/reusable semantics, relative dependency, and real value variables. Consequently, the classification of the correct gesture type, as well as the identification of the particular table corner it came from, is derived simultaneously using the same algorithm.

14.3 Pen-Gestures as Interaction Protocols

A standard way of communicating to the control system hosting an Ink Environment [87] is to involve the pen-gesture technology as interaction protocols. Where in the absence of a keyboard and a mouse, the digital pen is used directly on the digital screen surface to mark the start of the communication, select onscreen items as parameters, and conclude with a final mark of completion. Most of the time, we tend combine all of the three steps above into a single- or multi-stroke gesture-command to achieve the same effect.

The terminology of "gesture-command" insists that the pen-gestures applied on the environment are to be interpreted as system commands, which are to be acted upon immediately. A few or all of the trace-features contained in the gesture can be reused to affect this action visually through appropriate animations on the selected screen item for the users' benefit. The gestures therefore, should be identifiable from anywhere within the predefined environment enclosure.

Similarly, we describe our tabletop environment as an extension of the Ink Environment. And as such, when a gesture-command is drawn on an active onscreen widget, its reaction to the gesture is always somewhat related. An active onscreen widget here is one that we can animate expressively on the tabletop environment, as opposed to passive widgets which remain anchored to and unmovable at stipulated screen positions.

For instance, we can gesture a unidirectional arrow directly on an active document on the tabletop. The entire protocol is realised upon recognition of the gesture-command as, say, `WST_LT_ARR`, and the active document as the object parameter to the protocol. In addition, the tabletop environment also knows that the gesture comes from the user seated on its West corner. To complete the visual effect, the document is animated on the tabletop following the properties of the "arrow" gesture - drawing its speed, acceleration, trajectory, and direction from the trace-features contained in the gesture-command. Note that this is not the same as tapping the document, then meticulously dragging it across the table, and letting it go only when we are satisfied with its new position. In effect, there are other means to do this and are explained by several authors [100, 109, 124].

14.4 Significance of Trace-Features

Rubine [107] identified 13 unique features used in determining the gesture-type of an input trace for his gesture trainer and classifier algorithms. We have thoroughly discussed this earlier on in chapter 11. These trace-features purportedly describe the measurements of the traces' various lengths, bounding box properties, angles, durations, and speeds. They were selected to accommodate both small and large changes to the feature values that in turn will influence the stability of the trained-weights in the linear discriminator.

14.4.1 The Linear Discriminator

The identification of a gesture-type depends heavily on the goodness of variance on the values of the trained-weights w_{ci} in association to a related trace-feature f_i. In a linear gesture-classifier, the set of trace-features $f_1, \ldots, f_F$ extracted from an input trace g is applied to the linear evaluation function over the values of the trained-weights. The classification of the gesture G is the maximum v_c of all the known gesture values.

$$v_c = w_{c0} + \sum_{i=1}^{F} w_{ci}\ f_i \text{ , for } 0 \leq c \leq C \tag{14.1}$$

Here, C is the total number of known gesture-types and F is the total number of features extracted from g. The training problem then boils down to determining the weights w_{ci} from a sample set of traces that are meant to be interpreted as gestures.

14.4.2 Centre of Gravity and Angular Velocity

We implemented Rubine's training algorithm and retained all 13 features in our method. We also added seven more features that measure a trace based upon its angles and speeds, with respect to the trace's centre of gravity (COG). By definition and by our implementation, the COG is considered as both spatial and temporal. It tells us a point within the trace's bounding box of where it is most heavy and most balanced when viewed on the 2D plane. Temporally, the concentration of points (which led to the COG itself) is in fact the result of the input sampling rate of the digital screen device (when used as a tabletop), coupled to the users' way of manoeuvring the pen while gesturing; slowing down at some locations (which will pick up more sampled-points per square inch) and hurrying up at others.

Angular and velocity-type features, when observed from the trace's COG, are our dis-oriented features. We found them to have the most significant variance for their associated trained-weights c_{ci}, when it comes to deciding which corner of the table a gesture was conceived. Our study of analysing how independent an individual feature is with respect to other features (involving the original 13 features) in returning the correct classification of a gesture-type confirms this. Our observations point out that angular (spatial) and velocity-type (temporal) features can supersede all others, and are thus key features in their role of identifying gestures from tabletop corners

14.4.3 The *Dis*oriented Features

Our seven additional features are derived from an input trace as illustrated in Figure 14.2. The center of gravity (COG) $C_{\mathrm{G}} = (x_G, y_G)$ is a point on the 2D plane that is the median of all $\{x_0, \ldots, x_{n-1}\}$ and $\{y_0, \ldots, y_{n-1}\}$. We list our feature equations as follows, enumerating them after the first 13 originals from Rubine's.

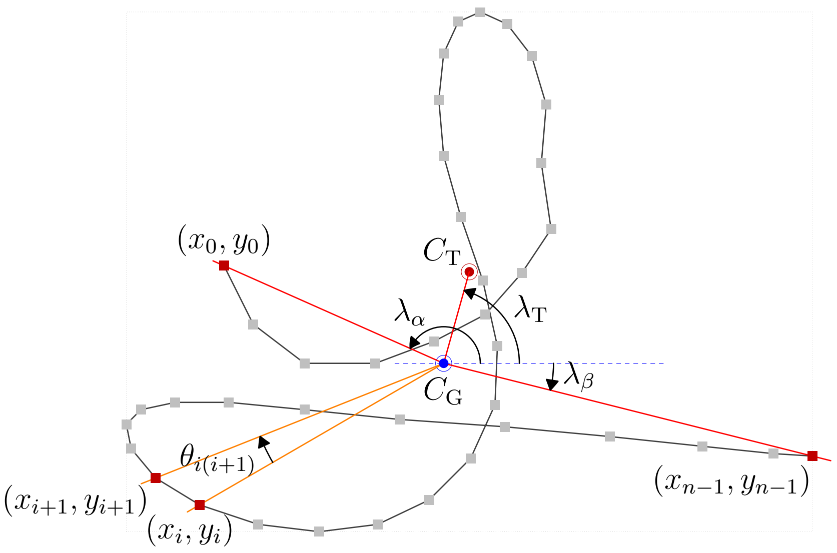


Figure 14.2: Deriving features with respect to the COG $C_{\mathrm{G}} \equiv (x_G, y_G)$, in addition to Rubine's original 13 features discussed earlier in Figure 11.2. The centre of the TRACE's bounding-box is marked by $C_{\mathrm{T}} \equiv (x_T, y_T)$.

Let $\partial x_{i\mathrm{G}} = x_i - x_{\mathrm{G}}$ and $\partial y_{i\mathrm{G}} = y_i - y_{\mathrm{G}}$.

$$f_{14} = \cos \lambda_\alpha = \partial x_{0\mathrm{G}} / \sqrt{\partial x_{0\mathrm{G}}^2 + \partial y_{0\mathrm{G}}^2} \tag{14.2}$$

$$f_{15} = \sin \lambda_\alpha = \partial y_{0\mathrm{G}} / \sqrt{\partial x_{0\mathrm{G}}^2 + \partial y_{0\mathrm{G}}^2} \tag{14.3}$$

$$f_{16} = \cos \lambda_\beta = \partial x_{(n-1)\mathrm{G}} / \sqrt{\partial x_{(n-1)\mathrm{G}}^2 + \partial y_{(n-1)\mathrm{G}}^2} \tag{14.4}$$

$$f_{17} = \sin \lambda_\beta = \partial y_{(n-1)\mathrm{G}} / \sqrt{\partial x_{(n-1)\mathrm{G}}^2 + \partial y_{(n-1)\mathrm{G}}^2} \tag{14.5}$$

$$f_{18} = \cos \lambda_{\mathrm{T}} = \partial x_{\mathrm{TG}} / \sqrt{\partial x_{\mathrm{TG}}^2 + \partial y_{\mathrm{TG}}^2} \tag{14.6}$$

$$f_{19} = \sin \lambda_{\mathrm{T}} = \partial y_{\mathrm{TG}} / \sqrt{\partial x_{\mathrm{TG}}^2 + \partial y_{\mathrm{TG}}^2} \tag{14.7}$$

Let $\Delta t_i = t_{i+1} - t_i$ and $\cos \theta_{ij} = \frac{\partial x_{i\mathrm{G}} \partial x_{j\mathrm{G}} - \partial y_{i\mathrm{G}} \partial y_{j\mathrm{G}}}{\sqrt{(\partial x_{i\mathrm{G}} \partial y_{i\mathrm{G}})^2 (\partial x_{j\mathrm{G}} \partial y_{j\mathrm{G}})^2}}$.

$$f_{20} = \max_{i=0}^{n-1} \frac{\cos \theta_{i(i+1)}}{\Delta t_i^2} \tag{14.8}$$

Features f_{14} to f_{19} are a series of related pairs, taking into consideration the cosine and sine of angles between C_{G} and (x_0, y_0), C_{G} and (x_{n-1}, y_{n-1}), and

C_G and C_T respectively. As was pointed out by Rubine, we favoured using the cosine and sine of the angles rather than using the angles themselves to avoid discontinuity as it passes through 2π and 0. These preliminary features detect the levels of dis-orientation a trace has that are centred on its heaviest point, where many of the sampled points within the trace are normally concentrated. These concentrations usually happen at bends and corners around the trace, and whenever users slow down in their gesturing.

Our primary observations indicate that these features have a tendency to show a significant variance depending on the corner from which the gesture was drawn, even when the outline of the gesture looks very much alike from all four corners of the table. We need to detect enough of these variances to ensure our derivation of the table-corner, from just an arbitrary input trace, has a high probability of being correct.

14.5 TableTop Gesture Groups

We can find two distinct groups when manipulating gestures for the tabletop; the *family* group, and the *corner* group.

14.5.1 Family Group

We define a *family* of a gesture-type as a set of gestures whose trace-outlines look very much alike when viewed from more than one corner of the table, but may have different interpretations (of their names and system commands) with respect to the corner they are associated with. Figures 14.1(a) and (b) illustrate this idea. For the same gesture outline in Figure 14.1(a), we interpret it as `STH_UP_ARR`, `EST_RT_ARR`, `NTH_DN_ARR`, and `WST_LT_ARR` respectively, going anti-clockwise around the table starting from the South corner. Similarly in Figure 14.1(b), we interpret the gesture as `NTH_ALPHA` and `EST_TWRL_DN` from the North and East corners respectively.

Note that from the way an 'alpha'-gesture is drawn, we do not include interpretations for the West and South corners as they would prove cognitively unpopular [88, 135]. This is the same reason why Figure 14.1(c) should only be interpreted from the West corner. Figure 14.3 gives a few more examples of the family of gestures that we studied for this paper. A 90-degree rotation of one family gesture results in another family gesture of the same set. A complete set of family gesture contains all family gestures of the same type for all four corner-orientations.

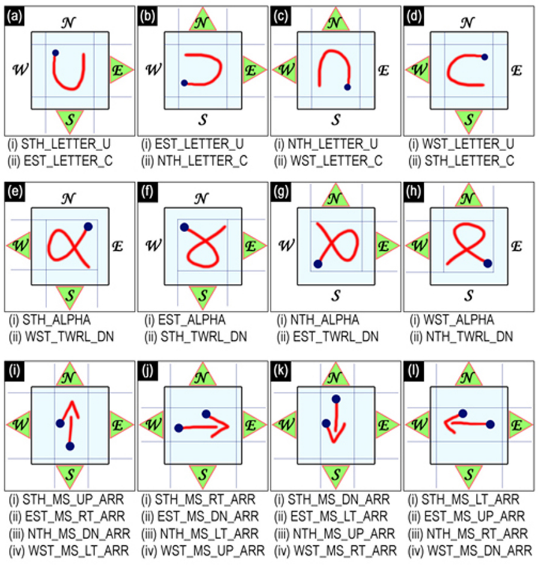


Figure 14.3: Three sets of complete single- and multi-stroke families of gestures: Set 1 – (a), (b), (c), (d); Set 2 – (e), (f), (g), (h); Set 3 – (i), (j), (k), (l).

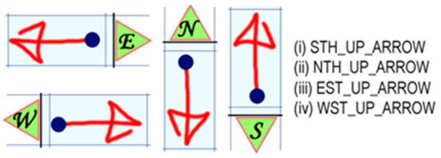


Figure 14.4: The `UP_ARROW` corner group.

14.5.2 Corner Group

A *corner* group refers to a set of gesture-types that has the same command interpretation from more than one table corner. Usually, the outlines of these gestures are not the same when viewed from a single corner of the table. A corner group is a consequence of picking one gesture from each family belonging to a complete family-set, and ensuring that the orientation information in the gesture name is exclusive. Figure 14.4 shows an example of an `UP_ARROW` corner group, which was derived from the complete set of family gestures in Figure 14.1(a).

14.6 Variance Comparison in Trained-Weights w_{ci}

We mentioned earlier that we used Rubine's classical training and recognition algorithms in this study. The only differences here are that we are working with a tabletop environment, and with seven additional feature definitions.

The trained-weights w_{ci} of all the known gesture-types are amassed together and then observed categorically by their family and corner groups. Particularly, we are interested in how much each trained-weight of a single gesture-type varies with respect to the same weight in all other gesture-types. The higher this variance, the more significant the trained-weight is, as it will have a bigger influence in the decision making equation of the linear discriminator.

14.7 Training by Table Corners for Family

In order that we have a complete set of family gestures, as illustrated in Figure 14.3, we need to ensure that we create training examples from the correct corners. As a matter of fact, we need to train at least 15 examples from one corner of the table for one particular orientation of a single gesture-type, and then repeat this operation on all four corners of the table to complete the family set. The gestures trained for each corner must coincidentally have the corner information imbued at the start in its name (i.e. `NTH_`, `STH_`, `EST_`, `WST_`).

We note here that the total number of gesture-types C for the tabletop is at least four times as many as the number of gesture-types for conventional digital screens (four being the number of corners that our tabletop equipment has).

14.8 Observation of the Trained-Weights w_{ci}

The results we obtained when observing the variances of the trained weights w_{ci} in association to their trace-features f_i are presented as follows.

Based on a given library of gesture-types C, we extract the values of each trained-weight w_{ci} for all $c \in C$ and $1 \leq i \leq 20$, and categorically study the variance of their standard deviations. Single-stroke gestures are viewed separately from the multi-stroke gestures, as the latter is made up of a few more identical trace-features f_i for each stroke-component that make up the full multi-stroke gesture.

We do not, however, discuss in this chapter our methodologies of multi-stroke gesture recognition that utilises and combines the fundamental trace-features on each gesture-component used in the current single-stroke recognition technique. We have developed a technique of compounding single-stroke 'seeds' that pave the way to identifying similar geometrical objects through a more abstract level of interpretation. However, this is still an ongoing work and has yet to be concluded. For now though, we believe that it is sufficient to state that multi-stroke gestures require several more trace-features than single-stroke gestures for efficient recognition of the gesture-types as well as identifying the corners they come from.

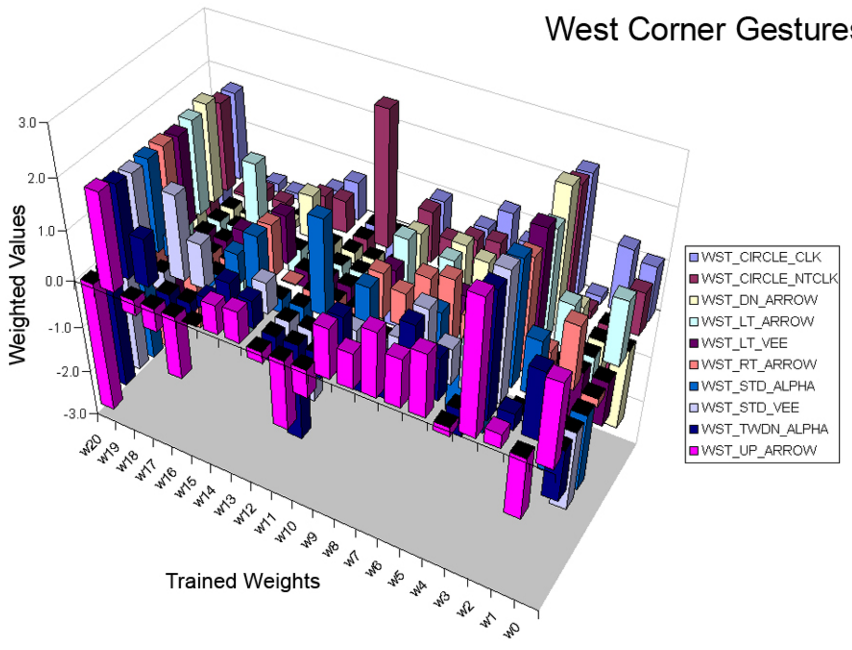


Figure 14.5: Variations of trained-weights w_i for single-stroke gestures arranged by corner-group gesture-types.

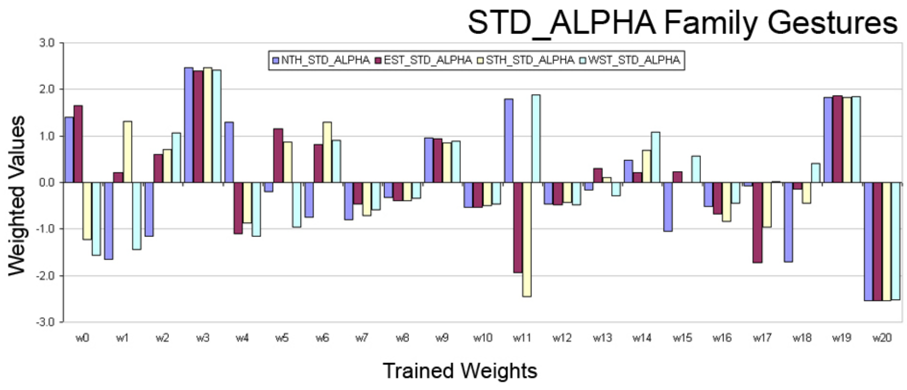


Figure 14.6: Variations of trained-weights w_i for single-stroke gestures arranged by family gesture-types.

We find that the trained-weights w_{ci} that vary the most have associated trace-features f_i that are composed of angular and velocity-type components. This is obvious for the single-stroke gestures as depicted in the peaks of the graphs in Figures 14.5 and 14.6; the former arranged by family-group gesture-types, and the latter by the corner-groups.

14.9 TableTop Monopoly

We programmed our own prototype of the Monopoly board game to test out the stability of our hypothesis in section 14.2. Our manual and partially-automatic Monopoly game was made to run on the most basic of tabletop environments, and involves a good mixture of active (e.g. dice, cards, money, player pieces) and passive (e.g. street boxes) onscreen widgets.

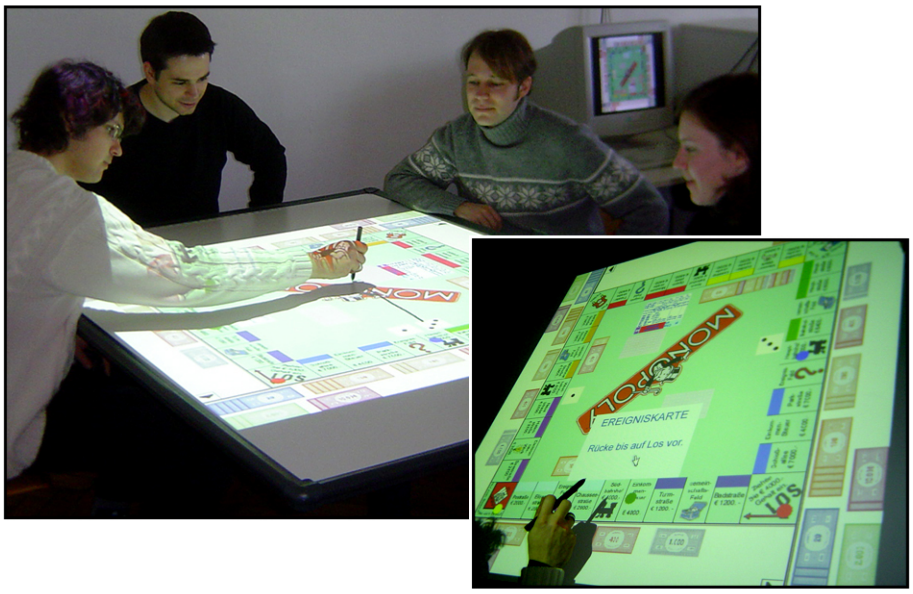


Figure 14.7: Tabletop Monopoly, developed in-house.

14.9.1 Handling 'Out-of-Turn' Actions

Monopoly is a turn-based game, and each new turn is preceded by the rolling of the dice. Depending on where on the street boxes the player lands, he/she can decide to make a purchase, auction, or pay rent to the owner of the street box property. Here, we are most interested in the last bit – the paying of rent. From the rules of the Monopoly game, the owner of the particular street box must demand a payment from his/her opponent who has just landed on the owner's property. This action must be done 'out-of-turn', that is, the owner of the stipulated property needs to ask for rent payment before the next rolling of the dice, or the owner forfeits the amount due. Around the tabletop, the owner of the property may not necessarily sit in the next turn of the current player (Monopoly changes players' turns in clockwise round). And because our Monopoly is programmed for manual play, the computer does not know which players own what properties.

This is where the advantage of our method comes in. We made use of up to eight easy-to-draw family of gestures in the game, and instructed the players to use them as communication protocols to affect their commands to the game controller. That is for example, a player performs a gesture to roll the dice, then gestures on his/her piece to move it across the board in the number of steps indicated by the dice, and gesture once more to indicate his/her desire to purchase or auction the street box property that he/she has just landed on. In the same manner, the owner of the street box property gestures anywhere on the tabletop to demand payment from his/her opponent. The computer deciphers this gesture and translates it to obtain the 'demand-rent' protocol-command as well as derive which corner of the table the gesture came from. Assuming that the owner of the property does not change places throughout the duration of the game, the computer is able to properly service the owner as per required. Figure 14.8 details this scenario graphically.

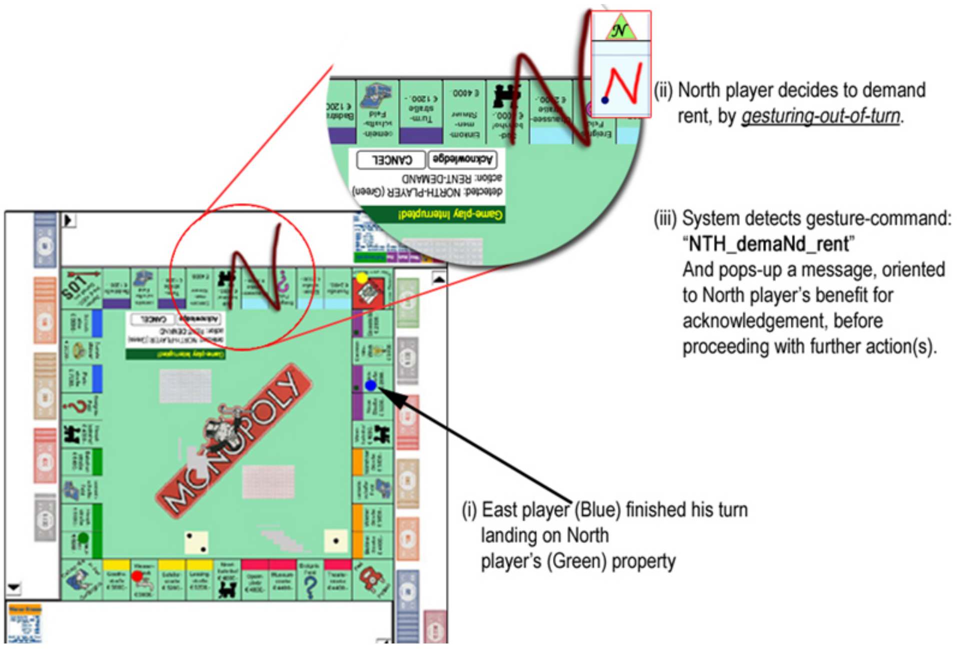

(ii) North player decides to demand rent, by *gesturing-out-of-turn*.
(iii) System detects gesture-command: "NTH_demaNd_rent" And pops-up a message, oriented to North player's benefit for acknowledgement, before proceeding with further action(s).
(i) East player (Blue) finished his turn landing on North player's (Green) property
MONOPOLY
Game-play Interrupted!
detected: NORTH-PLAYER (Green)
action: RENT-DEMAND
Acknowledge
CANCEL

Figure 14.8: North player gesturing 'out-of-turn', after East player landed on North player's property.

Other 'out-of-turn' actions in the Monopoly game that take advantage of the *dis*-oriented gestures include buying houses and hotels, redeeming mortgaged properties, and selling/buying owned properties between any two players.

14.9.2 Correctness of Identification

Rubine, in his original paper, suggested several ways of rejecting false positives that trigger a misclassification of an ambiguous gesture-type [107, 36]. This happens when there is a near tie for the maximum per-gesture-type evaluation in the linear discriminator. And as the corner information of the gesture-type is tied strongly to the features f_1 to f_{20}, our implementation of Rubine's method yielded the same results of at least 98% recognition rate of the correct gesture-type as well as the table corner. This is the corollary of the gesture-classification algorithm working.

Now, with respect to our Monopoly table-game, we collected results from a log file that we save at each run of the game. We ran the game 20 times; 14 involving four players, four involving three players, and two involving two players. The log file kept count of the number of times a gesture was made while a player was in-turn, and it also noted the number of times out-of-turn gestures were made. The results are shown in Table 14.1.

A typical complete game of Monopoly averages 27% out-of-turn gestures, and out of that, more than 98% were correctly classified (both gesture-type and table-corner).

Gesture Action	**Frequency**	**Correct**	**Misclassified**
In-turn	73.00%	99.21%	0.79%
Out-of-turn	27.00%	98.87%	1.13%

Table 14.1: Breakdown of gesture-actions for the Monopoly table-game.

14.9.3 Generality of Methods

Our method of determining the table-corner from which a gesture comes from is derived directly from the pen-gesture technology that involves conventional digital screen environments, which we presented these at the TableTop conference [85]. These screens are usually upright, sometimes wall-mounted, but always having only one orientation and only cater for one anonymous pen-input at a time. The tabletop display, on the other hand, makes use of a conventional digital screen, laid out flat on a table, and can only receive asynchronous pen-inputs.

The key point brought forward in this chapter is that users of our tabletop environment will be required to gesture to the system with their digital pens, in order to register their intention and start the protocol for communication. As long as users continue to use gestures as their means of controlling their communication and interfacing on the tabletop, our method should still remain valid. We just need to ensure that proper training of the gesture-types is carried out promptly irregardless of the medium used for gesturing (digital pen, fingers, etc.).

Also, if new tabletop hardwares that can handle multiple inputs simultaneously, can subtly separate all incoming information and pipe complete gestures for serial processing, then the tabletop software controlling the environment can be tweaked so that synchronous inputs can be processed the same way as the current asynchronous ones. Since the raw-gesture information are then intact, we will be able to determine which corners all incoming gestures come from.

Part IV

Conclusion

CHAPTER 15

Conclusion

In many ways, the first part of the thesis attempted to emulate Ockham's razor, in solving the problem of improving and *sustaining* the quality of digital freehand writings with symbolically represented curves. Where through the law of succinctness, William of Ockham, a 14th century English logician and Fransciscan friar, proclaimed in Latin,

"Entia non sunt multiplicanda praeter necessitatem."

This is roughly translated as "Entities must not be multiplied beyond necessity", and is often paraphrased, as we popularly know of it today, as "All other things being equal, the simplest solution is the best." [82] A superficially simple phenomenon may have a complex mechanism behind it. A simple explanation would be simplistic if it failed to capture all the essential and relevant parts.

What we found with the active-smoothing solution is an amalgamation of a series of subtly simple, yet important and non-trivial observations of the hidden characteristics of the Bézier family of spline curves. Our quest to smooth a set of time-ordered and indeterministically positioned points in a Trace on the 2D plane, took us from the classical cubic spline interpolation methods, to approximating the Traces exclusively with elliptic arcs, and ended with the frugally efficient and highly adaptable rational quadratic Bézier curves that completed our active-smoothing concept. Along the way, we pointed out several state-of-the-art approximation techniques that are highly intuitive, complex, and time consuming, which, unfortunately, may not produce the correct smoothing results that we are after when these techniques are applied to our domain of digital freehand writings.

Our conceived notions to smooth out the set of indeterministically positioned points lie not in consistently adding superfluous points to the current set, but rather to get rid of over 70% of the total points in any one Trace. The points retained are the *crucial* ones, and they not only dictate the overall shapes of the Traces they represent, but preserve them as well. Together with the measures of the first and second derivatives, and the measures of curvatures at those crucial points, we essentially have the basic ingredients to reconstruct a set of well-placed composite Curves that desirably approximate the original Traces, at any desired resolution. This was carried out specifically through the two new smoothing methods we presented, namely, the Elliptic Segmentation and the Bézier Segmentation. With these methods, we achieved highly precise curves that well-estimated the Traces, while strictly adhering to the Weierstrass approximation theorem, and significantly reducing the amount of storage space. It was directly from the two new methods that we brought out the exigent characteristics of the second-order curves, which then enabled us to reduce the $O(n)$ complex computations per input point down to a constant. These contributed to the final fragments we needed to assemble a very fast algorithm that we presented through the active-smoothing solution. Its performance, we saw, took less than 20% of the processor resources to smooth digital handwritings and render the resultant high quality, symbolicaly represented curves, while the primitive resolution sampling process from the transducer device is still ongoing.

The figures throughout the thesis, depicting the various examples of the active-smoothing process, are the product of incorporating our algorithm into *Asymptote* codes [41]. Asymptote is a vector graphics language that provides a natural coordinate-based framework for technical drawing. The labels and equations within the figures are typeset with LaTeX, and together they produce high-quality PostScript output. This is our attempt to prove our claims that the curves produced by our algorithms to smooth the pixelated Traces are of extremely high quality. We invite the reader to use his/her Adobe Acrobat program, while scrutinizing this thesis (in PDF), to zoom-in to the deepest level at 1800%, and to notice that the rendered curves never once degenerate into pixelated line-segments.

The prototypes of our algorithms were developed solely on the Java platform, but were later exported to MS Visual C++ to handle the demands of the Lecturnity software, and MS Visual C# to handle the climate mapping tool described in section 8.8. The stability of our algorithms in these programming domains is another indication of the stoutness and correctness of the concepts and routines within.

Once we have established the robustness of the smoothing routines, we went on to exploit the various branches that the intrinsic features within the Traces can be used to support. Not only were we able to differentiate between writing and gesturing on a common, unsegregated platform, but we were also able to determine the locations from which the Traces and Gestures originated. The former was portrayed through our work in developing "intelligent user interfaces" for the huge wall-mounted boards, and the latter based on the TableTop environment.

In conclusion, the combined parts of smoothing freehand written Traces with our new methods, both passively and actively, and the implementation of our gesture-based user applications, complete our proposed concepts and solutions to efficiently handle the digital ink.

CHAPTER 16

Non-Related Works

Throughout the course of my doctoral candidacy, I have also been involved with several other research projects offered by the Chair of Algorithms and Data Structures, that are completely not related to the topic presented in this thesis. These works centred around the common theme of “IP network algorithms”, broken down into two primary parts: IP packet classification, and IP conflict detection.

We have successfully communicated our ideas to the wider research community, through our publications, as well as through the completed works of the students working with us. The following lists the more significant contributions in chronological order.

- **Versioning Tree Structures by Path-Merging** (Frontiers in Algorithmics Workshop, 2008) [86]. We proposed *path-merging* as a refinement of techniques which we use to make linked data structures partially persistent. The technique’s strengths lie in their subtle, yet effective ways of merging non-essential versions of the original underlying methods of persistence to derive efficient time and space bounds. They are primed for handling applications where it makes sense to store only the substantially essential versions. The main ideas presented in this article came from the submitted work by Langner, which we describe the third listed item.

- **Is the Popular R*-tree Suited for Packet Classification?** (Network Computing and Applications, 2008) [65]. The benchmarks showcased in this paper depicted how well an R*-tree can be utilised for packet classification. The work describes two representative classification algorithms using the ClassBench tools suite.

- **Using Partial Persistence to Support Bursts of Operations in IP-lookup** (Bachelor Thesis, 2007) [62]. The original work here highlights the enhancement of the classical path-copying method, which makes linked data structures partially persistent, into what we term as the *path-merging* method. The new algorithm was tested on partially persistent red-black binary search trees, and gave efficient results in support of the *SlabDetect* algorithm, which resolves conflicts in a set of 1D packet filters.

- **A New Output-Sensitive Algorithm to Detect and Resolve Conflicts in Internet Router Tables** (InfoComm, 2007) [64]. This article contains the full description of the *SlabDetect* algorithm mentioned above. The algorithm runs in $O(n \log n)$ time to detect all conflicts in a given set of 1D packet filters, and reports at most $O(n)$ conflicts that are deemed essential.

- **Geometrical Algorithms for Packet Filter Conflict Detection** (Bachelor Thesis, 2007) [105]. This work is primarily based on Klee's measure problem to determine (and report) all overlaps from a given set of n iso-oriented rectangles which are not tightly covered. We developed a geometrical algorithm for use in the IP filter conflict detection environment by evolving the representation of overlapping rectangles in a Hasse diagram (by applying a transivity rule) and adapting it to a trellis structure. The implementation runs in $O(n\sqrt{n}\log n + k)$ time, where k refers to the number of parts of the broken up regions that are in conflict, which is $O(n^2)$.

Bibliography

[1] Anderson, R. J., Hoyer, C., Wolfman, S. A., and Anderson, R. A study of digital ink in lecture presentation. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2004), ACM, pp. 567–574.

[2] Àngel Jorba, and Tatjer, J. C. On the divergence of polynomial interpolation. *Journal of Approximation Theory 120*, 1 (2003), 85–110.

[3] Arge, L., de Berg, M., Haverkort, H. J., and Yi, K. The priority R-tree: A practically efficient and worst-case optimal R-tree. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2004), ACM, pp. 347–358.

[4] Backer, J., and Kirkpatrick, D. Finding curvature-constrained paths that avoid polygonal obstacles. In *SCG '07: Proceedings of the Twenty-Third Annual Symposium on Computational Geometry* (New York, NY, USA, 2007), ACM, pp. 66–73.

[5] Banks, M., and Cohen, E. Real time spline curves from interactively sketched data. In *SI3D '90: Proceedings of the 1990 Symposium on Interactive 3D Graphics* (New York, NY, USA, 1990), ACM, pp. 99–107.

[6] Bargeron, D., and Moscovich, T. Reflowing digital ink annotations. In *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2003), ACM, pp. 385–393.

[7] Barrientos, F. A., and Canny, J. F. Cursive: Controlling expressive avatar gesture using pen gesture. In *Proceedings of the 4th International Conference on Collaborative Virtual Environments* (2002), ACM Press, pp. 113–119.

[8] Barsky, B. A., and DeRose, T. D. Geometric continuity of parametric curves: Three equivalent characterizations. *Computer Graphics and Applications 9*, 6 (November 1989), 60–69.

[9] Baxter, L. K., and Asano, S. Graphical input tablet. United States Patent, Patent No. 3670103, Shintron Company, Inc. (Cambridge, MA, US), June 1972.

[10] Biswas, M. K., Dresevic, B., and Kallay, M. Method and apparatus for scale independent cusp detection. United States Patent, Patent No. 7146046, Microsoft Corporation (Redmond, WA, US), May 2006.

[11] Bunt, A., Conati, C., and McGrenere, J. What role can adaptive support play in an adaptable system? In *IUI '04: Proceedings of the 9th International Conference on Intelligent User Interfaces* (New York, NY, USA, 2004), ACM, pp. 117–124.

[12] Burden, R. L., and Faires, J. D. *Numerical Analysis*, 7th ed. Brooks Cole, Pacific Grove, CA 93950, USA, 2001.

[13] Callahan, J. R., Hopkins, D., Weiser, M. D., and Shneiderman, B. A. An empirical comparison of pie vs. linear menus. In *CHI '88: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1988), ACM, pp. 95–100.

[14] Catmull, E., and Rom, R. A class of local interpolating splines. *Computer Aided Geometric Design* (1974), 317–326.

[15] Chakerian, G. D. *Mathematical Plums.* Mathematical Association America, 1979, ch. 7. A Distorted View of Geometry.

[16] Chung, K.-L., Lin, F.-C., and Chen, W.-C. Cost-optimal parallel B-spline interpolations. In *ICS '90: Proceedings of the 4th International Conference on Supercomputing* (New York, NY, USA, 1990), ACM, pp. 121–131.

[17] Corel Corp. Corel®Grafigo™2. In *COREL* [online] http://www.ritescript.com/Products/ritePen.aspx, 2008.

[18] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 2nd ed. The MIT Press, Cambridge, Massachusetts 02142-1315, USA, 2001, ch. 18. B-Trees, pp. 434–454.

[19] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 2nd ed. The MIT Press, Cambridge, Massachusetts 02142-1315, USA, 2001, ch. 15. Dynamic programming, pp. 323–369.

[20] Crowley, J. L., Coutaz, J., and Bérard, F. Perceptual user interfaces: Things that see. *Communications of the ACM 43*, 3 (2000), 54–64.

[21] Cutler, R., and Turk, M. View-based interpretation of real-time optical flow for gesture recognition. In *Proceedings of the Third IEEE International Conference on Automatic Face & Gesture Recognition* (1998), IEEE Computer Society, pp. 416 –421.

[22] Cyber SIGN, Inc. Technology overview. In *Biometric signature verification* [online] http://www.cybersign.com/com/techoverview.htm, 2008.

[23] Danielsson, M., and Müller, R. A time-evolving data structure scalable between discrete and continuous attribute modifications. *Computer Science in Perspective* (2003), 98–114.

[24] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2000, ch. 2. Line segment intersection, pp. 19–44.

[25] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2000, ch. 10. More geometric data structures, pp. 211–234.

[26] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2000.

[27] DeRose, T. D., and Barsky, B. A. Geometric continuity, shape parameters, and geometric constructions for Catmull-Rom splines. *ACM Transactions on Graphics 7*, 1 (1988), 1–41.

[28] Dietz, P., and Leigh, D. Diamondtouch: a multi-user touch technology. In *UIST '01: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2001), ACM, pp. 219–226.

[29] Dominguez, S. M., Keaton, T., and Sayed, A. H. Robust finger tracking for wearable computer interfacing. In *PUI '01: Proceedings of the 2001 Workshop on Perceptive User Interfaces* (New York, NY, USA, 2001), ACM, pp. 1–5.

[30] Douglas, D. H., and Peucker, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2 (December 1973), 112–122.

[31] Evans, D. G., Schweitzer, P. N., and Hanna, M. S. Parametric cubic splines and geologic shape descriptions. *Mathematical Geology 17*, 6 (January 2005), 611–624.

[32] Evernote Corp. ritepen 3.0 product details. In *Ritescript Products* [online] http://www.ritescript.com/Products/ritePen.aspx, 2008.

[33] Faloutsos, C., Sellis, T., and Roussopoulos, N. Analysis of object oriented spatial access methods. *SIGMOD Record 16*, 3 (1987), 426–439.

[34] Flöry, S., and Hofer, M. Constrained curve fitting on manifolds. *Computer Aided Design 40*, 1 (2008), 25–34.

[35] Fornberg, B., and Zuev, J. The Runge phenomenon and spatially variable shape parameters in RBF interpolation. *Computers & Mathematics with Applications 54*, 3 (2007), 379–398.

[36] Frankish, C., Hull, R., and Morgan, P. Recognition accuracy and user acceptance of pen interfaces. In *CHI '95: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1995), ACM Press/Addison-Wesley Publishing Co., pp. 503–510.

[37] GRAHL Software Design. Annotate, edit and comment PDF files. In *PDF Annotator* [online] http://www.ograhl.com/en/pdfannotator/index.php, 2008.

[38] Grasset, R., Looser, J., and Billinghurst, M. OSGARToolKit: Tangible + transitional 3d collaborative mixed reality framework. In *ICAT '05: Proceedings of the 2005 International Conference on Augmented Tele-Existence* (New York, NY, USA, 2005), ACM, pp. 257–258.

[39] Guru, D. S., and Nagendraswamy, H. S. Symbolic representation of two-dimensional shapes. *Pattern Recognition Letters 28*, 1 (2007), 144–155.

[40] Guttman, A. R-trees: A dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1984), ACM, pp. 47–57.

[41] Hammerlindl, A., Bowman, J., and Prince, T. Asymptote: The vector graphics language. In *Asymptote* [online] http://asymptote.sourceforge.net/, 2008.

[42] Hangai, S., Yamanaka, S., and Hamamoto, T. Writer verification using altitude and direction of pen movement. *International Conference on Pattern Recognition 3* (2000), 483–486.

[43] Hansford, D. *The Essentials of CAGD*, 1st ed. AK Peters, October 2000, ch. 6. Bézier Patches, pp. 71–94.

[44] Hershberger, J., and Snoeyink, J. Speeding up the douglas-peucker line-simplification algorithm. Tech. rep., University of British Columbia, Vancouver, BC, Canada, Canada, 1992.

[45] Hurst, G. S. Electrographic sensor for determining planar coordinates. United States Patent, Patent No. 3798370, Elographics, Inc. (Oak Ridge, TN, US), March 1974.

[46] Hurst, G. S., and Park, J. E. Electrical sensor of plane coordinates. United States Patent, Patent No. 3662105, The University of Kentucky Research Foundation (Lexington, KY, US), May 1972.

[47] Hürst, W., Maass, G., Müller, R., and Ottmann, T. The "authoring on the fly" system for automatic presentation recording. In *CHI '01 extended abstracts on Human factors in computer systems* (2001), ACM Press, pp. 5–6.

[48] Hürst, W., Maass, G., Müller, R., and Ottmann, T. The "Authoring on the Fly" system for automatic presentation recording. In *CHI '01: CHI '01 extended abstracts on Human Factors in Computing Systems* (New York, NY, USA, 2001), ACM, pp. 5–6.

[49] Hürst, W., and Mohamed, K. A. *Enhancing Learning through Human Computer Interaction.* Information Science Reference, London, WC2E 8LU, January 2007, ch. II. Human Computer Interaction for Computer-Based Classroom Teaching, pp. 21–42.

[50] Hürst, W., and Müller, R. A synchronization model for recorded presentations and its relevance for information retrieval. In *MULTIMEDIA '99: Proceedings of the Seventh ACM international Conference on Multimedia (Part 1)* (New York, NY, USA, 1999), ACM, pp. 333–342.

[51] Igarashi, T., Edwards, W. K., LaMarca, A., and Mynatt, E. D. An architecture for pen-based interaction on electronic whiteboards. In *AVI '00: Proceedings of the Working Conference on Advanced Visual Interfaces* (New York, NY, USA, 2000), ACM, pp. 68–75.

[52] IMC AG. Rapid authoring tool LECTURNITY – Rapid learning. In *imc Advanced Learning Solutions* [online] http://www.lecturnity.de/, 2008.

[53] Jain, A. K., Griess, F. D., and Connell, S. D. On-line signature verification. *Pattern Recognition 35*, 12 (2002), 2963–2972.

[54] Jameson, A. Adaptive interfaces and agents. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications* (2003), 305–330.

[55] Jenks, G. F. Lines, computers, and human frailties. *Annals of the Association of American Geographers 71*, 1 (March 1981), 1–10.

[56] Johnson, R. A., and Bhattacharyya, G. K. *Statistics – Principles and Methods*, 3rd ed. John Wiley & Sons, Inc., Toronto, Canada, 1996.

[57] KENDON, A. An agenda for gesture studies. In *Semiotic Review of Books 7*, 3 [online] http://www.univie.ac.at/wissenschaftstheorie/srb/srb/gesture.html, 1996.

[58] KNUTH, D. E. *The Metafont book*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[59] KOBAYASHI, M., AND KOIKE, H. EnhancedDesk: integrating paper documents and digital documents. *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific* (July 1998), 57–62.

[60] KRUEGER, W., AND FROEHLICH, B. The responsive workbench. *IEEE Computer Graphics and Applications 14*, 3 (1994), 12–15.

[61] KRUGER, R., CARPENDALE, S., SCOTT, S. D., AND GREENBERG, S. How people use orientation on tables: comprehension, coordination and communication. In *GROUP '03: Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work* (New York, NY, USA, 2003), ACM, pp. 369–378.

[62] LANGNER, T. Using partial persistence to support bursts of operations in IP-lookup. *Bachelor Thesis, Albert-Ludwigs-Universität Freiburg* (March 2007).

[63] M, M. S., AND RAZA, S. A. Capturing outline of fonts using genetic algorithm and splines. *Proceedings of the Fifth International Conference on Information Visualisation* (2001), 738–743.

[64] MAINDORFER, C., MOHAMED, K. A., OTTMANN, T., AND DATTA, A. A new output-sensitive algorithm to detect and resolve conflicts in internet router tables. In *INFOCOM 2007: Proceedings of the 26th IEEE International Conference on Computer Communications* (May 2007), IEEE Press, pp. 2431–2435.

[65] MAINDORFER, C., AND OTTMANN, T. Is the popular R*-tree suited for packet classification? *Seventh IEEE International Symposium on Network Computing and Applications* (July 2008), 168–176.

[66] MALAVARD, L. C., AND MARTY, P. M. Graphical input system. United States Patent, Patent No. 3449516, IIT Research Institute (Chicago, IL, US), October 1969.

[67] MALAVARD, L. C., AND MARTY, P. M. Graphic data transcription system. United States Patent, Patent No. 3632874, Agence V, Nationale De Valorisation De La Recherche A. N. A. R. (Puteaux, France), April 1972.

[68] MANLY, B. F. J. *Multivariate Statistical Methods: A Primer*, 3rd ed. Chapman & Hall/CRC, Boca Raton, Florida 33431, USA, 2004.

[69] MANOCHA, D., AND CANNY, J. F. Detecting cusps and inflection points in curves. *Computer Aided Geometric Design 9*, 1 (May 1992), 1–24.

[70] MARINO, J. S. Identification of characteristic points along naturally occurring lines – an empirical study. *Cartographica: The International Journal for Geographic Information and Geovisualization 16*, 1 (June 1979), 70–80.

[71] MARK, D. M. Conceptual basis for geographic line generalization. In *Proceedings of the 9th International Symposium on Computer-Assisted Cartography* (Journals Division, 5201 Dufferin Street, Toronto, ON, Canada, M3H 5T8, April 1989), University of Toronto Press and Cartographica, pp. 68–77.

[72] Masoodian, M., McKoy, S., and Rogers, B. Hands-on sharing: collaborative document manipulation on a tabletop display using bare hands. In *CHINZ '07: Proceedings of the 7th ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction* (New York, NY, USA, 2007), ACM, pp. 25–31.

[73] Matuschek, O. Climate mapping tool. *Bachelor Thesis, Albert-Ludwigs-Universität Freiburg* (April 2008).

[74] McCowan, I., Gatica-Perez, D., Bengio, S., Moore, D., and Bourlard, H. Towards computer understanding of human interactions. *Machine Learning for Multimodal Interaction 3361/2005* (January 2005), 56–75.

[75] McGrenere, J., Baecker, R. M., and Booth, K. S. An evaluation of a multiple interface design solution for bloated software. In *CHI '02: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2002), ACM, pp. 164–170.

[76] McKay, E. *Enhancing Learning through Human Computer Interaction.* Information Science Reference, London, WC2E 8LU, January 2007.

[77] McKinley, S., and Levine, M. Cubic spline interpolation. Tech. rep., College of the Redwoods, Eureka, CA 95501, USA, 1998.

[78] McMaster, R. B. A statistical analysis of mathematical measures for linear simplification. *Cartography and Geographic Information Science 13*, 2 (April 1986), 103–116.

[79] McMaster, R. B. Automated line generalization. *Cartographica: The International Journal for Geographic Information and Geovisualization 24*, 2 (1987), 74–111.

[80] McMaster, R. B. The integration of simplification and smoothing algorithms in line generalization. *Cartographica: The International Journal for Geographic Information and Geovisualization 26*, 1 (1989), 101–121.

[81] Melkman, A. A. On-line construction of the convex hull of a simple polyline. *Information Processing Letters 25*, 1 (1987), 11–12.

[82] Menger, K. A counterpart of Occam's razor in pure and applied mathematics ontological uses. *Synthese 12*, 4 (December 1960), 415–428.

[83] Mohamed, K. A. Increasing the accuracy of anticipation with lead-lag timing analysis of digital freehand writings for the perceptual environment. In *ICCMSE '04: Proceedings of the International Conference on Computational Methods in Sciences and Engineering* (Leiden, The Netherlands, 2004), T. Simos and G. Maroulis, Eds., VSP/Brill Science Citation Index, pp. 387–390.

[84] Mohamed, K. A., Belenkaia, L., and Ottmann, T. Post-processing InkML for random-access navigation of voluminous handwritten ink documents. In *Proceedings of the 13th International World Wide Web Conference* (New York, NY, USA, 2004), ACM, pp. 266–267.

[85] Mohamed, K. A., Haag, S., Peltason, J., Dal-Ri, F., and Ottmann, T. Disoriented pen-gestures for identifying users around the tabletop without cameras and motion sensors. *TableTop 2006. First IEEE International Workshop on Horizontal Interactive Human-Computer Systems* (January 2006), 43–50.

[86] Mohamed, K. A., Langner, T., and Ottmann, T. Versioning tree structures by path-merging. In *Proceedings of the Second Annual International Workshop, Frontiers in Algorithmics FAW 08* (Berlin, 2008), LNCS Springer, pp. 101–112.

[87] Mohamed, K. A., and Ottmann, T. Fast interpretations of pen gestures with competent agents. In *Proceedings of the IEEE Second International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS)* (2003), IEEE Press, pp. PS09–04.

[88] Mohamed, K. A., and Ottmann, T. *Encyclopedia Of Human Computer Interaction*. IGI Global, Hershey, PA 17033, USA, December 2005, ch. Pen-Based Digital Screen Interaction, pp. 323–369.

[89] Mohamed, K. A., and Ottmann, T. Active-smoothing in digital ink environments. In *EMME '07: Proceedings of the International Workshop on Educational Multimedia and Multimedia Education* (New York, NY, USA, 2007), ACM, pp. 119–120.

[90] Morse, A., and Reynolds, G. Overcoming current growth limits in ui development. *Communications of the ACM 36*, 4 (1993), 72–81.

[91] Mukhopadhyay, S., and Smith, B. Passive capture and structuring of lectures. In *Proceedings of the seventh ACM international conference on Multimedia (Part 1)* (1999), ACM Press, pp. 477–487.

[92] Müller, R., and Ottmann, T. The "authoring on the fly" system for automated recording and replay of (tele)presentations, Special issue on "multimedia authoring and presentation techniques". *Multimedia Systems 8*, 3 (October 2000), 158–176.

[93] Opheim, H. Smoothing a digitized curve by data reduction methods. In *Eurographics '81: Proceedings of the International Conference and Exhibition* (Amsterdam, The Netherlands, 1981), J. L. Encarnacao, Ed., North-Holland Publishing Company, pp. 127–135.

[94] Pal, S., Biswas, P., and Abraham, A. Face recognition using interpolated Bézier curve based representation. *Proceedings of the International Conference on Information Technology: Coding and Computing 1* (April 2004), 45–49.

[95] Pavlidis, T. Curve fitting with conic splines. *ACM Transactions on Graphics 2*, 1 (1983), 1–31.

[96] Pentland, A. Perceptual user interfaces: Perceptual intelligence. *Communications of the ACM 43*, 3 (2000), 35–44.

[97] Pham, B. Conic B-splines for curve fitting: A unifying approach. *Computer Vision, Graphics, and Image Processing 45*, 1 (1989), 117–125.

[98] Plass, M., and Stone, M. Curve-fitting with piecewise parametric cubics. *SIGGRAPH Computer Graphics 17*, 3 (1983), 229–239.

[99] Pottmann, H., and Wallner, J. *Computational Line Geometry*, 1st ed. Springer, Heidelberg, 69126 Germany, August 2001.

[100] Rekimoto, J. Smartskin: an infrastructure for freehand manipulation on interactive surfaces. In *CHI '02: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2002), ACM, pp. 113–120.

[101] Ren, Y., Yang, C., Yu, Z., and Wang, P. A way to speed up buffer generalization by douglas-peucker algorithm. *Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International 5* (September 2004), 2916–2919.

[102] Reumann, K., and Witkam, A. P. M. Optimizing curve segmentation in computer graphics. In *Proceedings of the International Computing Syposium* (New York, NY, USA, 1974), A. Gunther, B. Levrat, and H. Lipps, Eds., Elsevier, pp. 467–472.

[103] Rice, J. A. *Mathematical Statistics and Data Analysis*, 2nd ed. Duxbury Press, Belmont, California 94002, 1995.

[104] Richter, S. Abstracting digital ink traces with efficient line simplification algorithms. *Studienarbeit Thesis, Albert-Ludwigs-Universität Freiburg* (September 2004).

[105] Ritter, G. Geometrical algorithms for packet filter conflict detection. *Bachelor Thesis, Albert-Ludwigs-Universität Freiburg* (January 2007).

[106] Rogers, Y., Hazlewood, W., Blevis, E., and Lim, Y.-K. Finger talk: collaborative decision-making using talk and fingertip interaction around a tabletop display. In *CHI '04: Human Factors in Computing Systems* (New York, NY, USA, 2004), ACM, pp. 1271–1274.

[107] Rubine, D. Specifying gestures by example. *SIGGRAPH Computer Graphics 25*, 4 (1991), 329–337.

[108] Russell, G., Chee, Y.-M., Seni, G., Yaeger, L., Tremblay, C., Franke, K., Madhvanath, S., and Froumentin, M. Ink markup language [online], Available: http://www.w3.org/TR/InkML/. *W3C Working Draft* (October 2006).

[109] Ryall, K., Forlines, C., Shen, C., and Morris, M. R. Exploring the effects of group size and table size on interactions with tabletop shared-display groupware. In *CSCW '04: Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work* (New York, NY, USA, 2004), ACM, pp. 284–293.

[110] Salzberg, B., and Tsotras, V. J. Comparison of access methods for time-evolving data. *ACM Computing Surveys (CSUR) 31*, 2 (1999), 158–221.

[111] Sarfraz, M. A rational spline with tension: some CAGD perspectives. *Proceedings of the 1998 IEEE Conference on Information Visualization* (Jul 1998), 178–183.

[112] Sarfraz, M., Asim, M., and Masood, A. Capturing outlines using cubic Bézier curves. *Proceedings of the 2004 International Conference on Information and Communication Technologies: From Theory to Applications* (April 2004), 539–540.

[113] Sederberg, T., and Farouki, R. Approximation by interval Bézier curves. *Computer Graphics and Applications, IEEE 12*, 5 (September 1992), 87–95.

[114] Sellis, T. K., Roussopoulos, N., and Faloutsos, C. The R+-tree: A dynamic index for multi-dimensional objects. In *VLDB '87: Proceedings of the 13th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1987), Morgan Kaufmann Publishers Inc., pp. 507–518.

[115] Shneiderman, B. A. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *IUI '97: Proceedings of the 2nd International Conference on Intelligent User Interfaces* (New York, NY, USA, 1997), ACM, pp. 33–39.

[116] Shneiderman, B. A., and Maes, P. Direct manipulation vs. interface agents. *Interactions 4*, 6 (1997), 42–61.

[117] Siepmann, D. Cardinal interpolation by polynomial splines: Interpolation of data with exponential growth. *Journal of Approximation Theory 53*, 2 (1988), 167–183.

[118] Sinden, F. W. Method and apparatus for parametric representation of handwritten symbols. United States Patent, Patent No. 6580826, Lucent Technologies Inc. (Murray Hill, NJ), June 2003.

[119] Smarandache, F. *Generalisations et Generalites*, nouvelle ed. Ed. Nouvelle, Fes, Marocco, 1984, ch. Chap. Generalizations of Ceva's Theorem and Applications, pp. 15–20.

[120] Sohel, F. A., Karmakar, G. C., and Dooley, L. S. A generic shape descriptor using Bézier curves. *International Conference on Information Technology: Coding and Computing 2* (April 2005), 95–100.

[121] Sunday, D. Algorithm 16: Polyline simplification. In *Geometry Algorithms* [online] http://geometryalgorithms.com/Archive/algorithm_0205/algorithm_0205.htm, 2006.

[122] Toraichi, K., Kwan, P. W. H., Katagishi, K., Sugiyama, T., Wada, K., Mitsumoto, M., Nakai, H., and Yoshikawa, F. On a fluency image coding system for beef marbling evaluation. *Pattern Recognition Letters 23*, 11 (2002), 1277–1291.

[123] Turk, M., and Robertson, G. Perceptual user interfaces (introduction). *Communications of the ACM 43*, 3 (2000), 32–34.

[124] Ullmer, B., and Ishii, H. The metaDESK: models and prototypes for tangible user interfaces. In *UIST '97: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 1997), ACM, pp. 223–232.

[125] Ünlü, A., Brause, R., and Krakow, K. Handwriting analysis for diagnosis and prognosis of Parkinsons disease. In *Proceedings of the International Symposium on Biological and Medical Data Analysis* (2006), N. Maglaveras, I. Chouvarda, V. Koutkias, and R. Brause, Eds., LNCS Vol 4345, Springer Verlag Heidelberg, pp. 441–450.

[126] Vaughan, J., Whyatt, D., and Brookes, G. A parallel implementation of the douglas-peucker line simplification algorithm. *Software: Practice and Experience 21*, 3 (1991), 331–336.

[127] von Hardenberg, C., and Bérard, F. Bare-hand human-computer interaction. In *PUI '01: Proceedings of the 2001 Workshop on Perceptive User Interfaces* (New York, NY, USA, 2001), ACM, pp. 1–8.

[128] Weisstein, E. W. Brianchon point. In *MathWorld – A Wolfram Web Resource* [online] http://mathworld.wolfram.com/BrianchonPoint.html, 2008.

[129] Weisstein, E. W. Conway triangle notation. In *MathWorld – A Wolfram Web Resource* [online] http://mathworld.wolfram.com/ConwayTriangleNotation.html, 2008.

[130] Weisstein, E. W. Inellipse. In *MathWorld – A Wolfram Web Resource* [online] http://mathworld.wolfram.com/Inellipse.html, 2008.

[131] Weisstein, E. W. Trilinear coordinates. In *MathWorld – A Wolfram Web Resource* [online] http://mathworld.wolfram.com/TrilinearCoordinates.html, 2008.

[132] Wellner, P. Interacting with paper on the digitalDesk. *Communications of the ACM 36*, 7 (1993), 87–96.

[133] Wexelblat, A. Gesture at the user interface: A CHI '95 workshop. *ACM SIGCHI Bulletin 28*, 2 (1996), 22–26.

[134] White, E. R. Assessment of line-generalization algorithms using characteristic points. *Cartography and Geographic Information Science 12*, 1 (April 1985), 17–28.

[135] Wu, M., and Balakrishnan, R. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *UIST '03: Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2003), ACM, pp. 193–202.

[136] Yang, H.-M., Lu, J.-J., and Lee, H.-J. A Bézier curve-based approach to shape description for Chinese calligraphy characters. *Proceedings of the Sixth International Conference on Document Analysis and Recognition* (September 2001), 276–280.

[137] Yap, C. K. Complete subdivision algorithms, I: Intersection of Bezier curves. In *SCG '06: Proceedings of the Twenty-Second Annual Symposium on Computational Geometry* (New York, NY, USA, 2006), ACM Press, pp. 217–226.

[138] Zupancic, B., and Horz, H. Lecture recording and its use in a traditional university course. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education* (2002), ACM Press, pp. 24–28.