Diss. ETH No. 20676

Novel Toolset for 2D Drawing and Animation

A dissertation submitted to
ETH Zurich

for the Degree of
Doctor of Sciences

presented by
Gioacchino Noris
MSc in Computer Science, ETH Zurich, Switzerland
born 14 May 1983
citizen of Switzerland

accepted on the recommendation of
Prof. Dr. Markus Gross, examiner
Dr. Robert Sumner, co-examiner
Prof. Dr. Marie-Paule Cani, co-examiner

2012

Abstract

This thesis investigates a set of novel digital tools for 2D Animation addressing the shortcomings of current digital support, representations and algorithms. Our goal is to produce animation tools that are intuitive to use, allow full control over the resulting drawing when desired, and provide the artist with immediate visual feedback of the animation as it progresses.

In contrast to previous work, where automation has been often been the goal, we focus on building tools that keep the artist as a central agent. We target automation of the most tedious tasks where the need for artistic interpretation is minimal, and otherwise aim for computer-assisted solutions geared to provide a similar experience as with traditional workflows augmented with algorithmic computation.

We start by investigating the problem of representing drawings digitally, analyzing existing representations, and highlighting their major shortcomings. We then present a hybrid representation that combines the advantages of vector and raster images, and propose the use of a novel vector description for lines and areas.

We then address the problem of vectorization of line drawings. This problem is challenging due to ambiguities in regions where lines are drawn close to each other or intersect. We propose a two-step, topology-driven approach that first exploits the pixel gradient information in a clustering process to generate an initial stroke graph from which the topology of the drawing is learned, and then applies a "reverse drawing" procedure where plausible junction configurations are considered and a heuristic optimum is selected.

Segmentation is a key step in organizing digital drawings into semantic groups ready for editing and animation. Done manually, this can be a very labor intensive task. We propose a scribble-based interface that guides a novel

energy minimization resulting in the labeling of the drawing strokes. In contrast to previous methods, we exploit both geometric and temporal information available with modern drawing devices.

In the realm of applications, we address the task of inbetweening, which is the creation of animation frames between pairs of key frames in order to create the illusion of a continuous animation. Drawings are represented as stroke graphs. Given two input key frames, a mapping between the graphs is derived, and spiral trajectories for graph nodes and additional salient points are computed. Strokes are then interpolated, leading to an initial set of inbetween frames. We propose a set of tools to modify the mapping, deal with simple topological mismatches, and redraw animation trajectories.

Finally, we propose a technique to control temporal noise in sketchy animation. Sequences of sketches typically present notable temporal artifacts in the form of visual flickering due to the lack of temporal consistency in the way sketched lines vary from the visually perceived boundaries and interior lines. We propose a two-step method that applies a temporal filter bi-linearly. By combining motion extraction, stroke correspondence, and inbetweening, temporal consistency can be enforced at the stroke level. We first apply this to selected key frames in the input animation to generate a so-called "noise free" sequence, and then to pairs of frames from the input sequence and the noise free sequence to obtain the desired temporal noise level specified by the user.

Compendio

Questa tesi di dottorato tratta lo studio di un nuovo set di strumenti ideati per facilitare l'automatizzazione e il supporto digitale per l'animazione bidimensionale (2D). Presentiamo un insieme di algoritmi e di tecniche specificatamente studiati per risolvere problemi considerati importanti in questa area di ricerca.

In contrasto con precedenti lavori di ricerca, dove l'automatizzazione era lo scopo principale, il nostro intento é quello di produrre strumenti intuitivi da usare, che diano all'artista totale controllo sul risultato, e che rispondano in tempo reale agli input permettendo una esperienza di lavoro scorrevole ed efficace.

Il concetto centrale é quello di mantenere l'artista al centro delle operazioni: tutto ciò che richiede interpretazione artistica é lasciato all'utente, mentre il potere computazionale viene utilizato per svolgere mansioni tediose, ripetitive, o che richiederebbero troppo tempo per essere svolte manualmente. Il nostro studio diventa quindi quello di ideare strumenti di supporto soggetti al controllo artistico.

In principio, proponiamo uno studio di come le immagini vengono rappresentate in campo digitale. Questa rappresentazione é determinante per quanto riguarda l'elaborazione dei contenuti, e quindi la possibilitá di costruire strumenti che supportino l'animazione 2D. Nel nostro lavoro identifichiamo una serie di problemi dovuti a come le immagini sono codificate, e proponiamo una rappresentazione alternativa ibrida, che combini i vantaggi di quelle esistenti (griglia di pixel, e immagini vettoriali). Discutiamo in fine una rappresentazione matematica unica per linee ed aree.

Ci dedichiamo poi al problema della vettorializzazione, ovvero la costruzione in digitale di un disegno su carta. Questo problema é ostico

principalmente in regioni dove i tratti del disegno si intersecano, o sono disegnati talmente vicini da risultare confusi. Proponiamo un sistema a doppia mandata, incentrato sullo studio della topologia del disegno. In primis, l'informazione contenuta nel gradiente di ogni pixel della griglia viene utilizzata in un processo di agglomerazione (clustering), che termina con la costruzione di un grafico che cattura la topologia del disegno. Grazie a questo grafico, applichiamo una procedura chiamata "Reverse Drawing", dove plausibili configurazioni di tratti vengono analizate, e quelle considrate migliori scelte, risultando in una soluzione euristica ottimale.

Affrontiamo poi il problema della segmentazione. Segmentare, o separare in gruppi distinti, é un processo chiave nell'organizzazione di un disegno in livelli separati. Fatto manualmente, questo processo é spesso molto costoso in termini di tempo ed energie, e spesso impone un procedere che si allontana dal semplice disegnare, richiedendo tuttavia particolare attenzione e spesso conducendo ad errori. Proponiamo dunque una interfaccia basata su scarabocchi (scribbles) atta a guidare una minimizzazione energetica che termina con l'assegnazione, per ogni tratto del disegno, ad un particolare gruppo o livello. In contrasto con lavori precedenti, il nostro metodo propone l'utilizzo sia di informazione geometrica che temporale, disponibile oggigiorno grazie a dispotivi di disegno digitali.

Nell'ambito delle applicazioni, ci occupiamo del problema chiamato "Inbetweening", ovvero la creazioni di immagini che stiano "in mezzo" a immagini chiave. Dato un disegno di partenza, queste immagini presentano modifiche graduali dei contenuti fino ad ottenere un disegno di arrivo, creando cosí l'illusione di una continuitá. Procediamo cercando un assegnazione di corrispondenza dei tratti provenienti da due immagini chiave (di partenza e di arrivo), e per ogni punto saliente in corrispondenza generiamo una traiettoria a spirale. I tratti vengono poi interpolati seguendo tali traiettorie, permettendo di generare le immagini desiderate. Il nostro metodo comprende strumenti per genstire semplici situazioni dove la topologia delle immagini chiave non corrisponde, come pure per ridisegnare traiettorie e aggiungere punti salienti.

Come ultimo lavoro, proponiamo una tecnica per controllare la quantitá di rumore visivo — chiamato "rumore temporale" — presente nelle animazioni dallo stile grafico caotico e ricco di tratti, come nel caso di bozze e schizzi. Sequenze di schizzi presentano artefatti temporali percepiti come sfarfallamento dell'immagine. La ragione é che i bordi degli oggetti sono disegnati da linee in costante mutamento, con spostamenti considerevoli e spesso non coerenti nel procedere con la sequenza di immagini. Proponiamo un metodo che applica un filtro temporale bi-lineare. Combinando l'estrazione del movi-

mento, il calcolo della corrispondeza fra tratti, e la creazione di immagini "nel mezzo", possiamo costrignere i tratti a muoversi fluidamente secondo una coerenza temporale. Tale processo viene prima applicato ad immagini chiave, generando cosi' una sequenza detta "senza rumore", e in un secondo momento, lo stesso processo é applicato a coppie di immagini: una proveniente da questa sequenza "senza rumore" e l'altra dalla sequenza originale. Il risultato, é una animazione dove la quantitá di rumore temporale tende ad un valore desiderato.

For $\mathcal{D}.\mathcal{N}$.

Acknowledgments

Hereby, I place on record my deepest appreciation to all the people who helped me during this Ph.D. Study, all the coauthors of the publications, and everyone that contributed, in any form, to the completion of this work.

First and foremost, I want to thank my mentor and advisor **Prof. Dr. Markus Gross**. Through his work and passion, he created a wonderful environment for research and study. His trust and guidance have been substantial to the successes obtained during these past four years.

I extremely grateful and indebted to the group of close collaborators that made the projects presented in this thesis possible:

- **Dr. Robert Sumner**, from Disney Research Zurich, for his guidance, his positive and passionate approach to research, and for the atmosphere of sympathy and friendship he casts on people around him.
- **Dr. Maryann Simmons** and **Dr. Brian Whited**, from the Walt Disney Animation Studios, for being my companions from day one and for supporting and motivating me over the years. Their help has been invaluable.
- **Dr. Alexander Hornung** and **Dr. Stelian Coros**, from Disney Reserch Zurich, for their day-to-day support, for the many research discussions, and for sharing the struggles and joys of our paper submissions.
- **Dr. Daniel Sýkora**, from the Czech Technical University of Prague, for his passion and dedication to hand drawn animation, his deep mathematical knowledge, and for all the tricks he thought me during our delightful collaborations.

My sincere gratitude to the staff at Disney Research Zurich, **Michelle Berchtold**, **Linda Breu** and **Dr. Stephan Veen**, at ETH Zurich, **Denise Spicher**, and at The Walt Disney Company, **Dayna Meltzer** and **Susan Harden**.

I would like to thank all the artists that contributed to this work with their art pieces. In particular, many thanks to **Maurizio Nitti**, from Disney Research Zurich, for his unlimited availability and support.

A thank you to all the people from the **Computer Graphics Laboratory** at ETH Zurich and from **Disney Research Zurich**. Special regards to **Dr. Thabo Beeler** who shared the office with me, and has been a friend and an example.

Finally, I am grateful to my to my family and my girlfriend **Giulia Ghielmetti**, for their unconditional support and affection, and for making every day of my life happier.

Contents

CHAPTER 1

Introduction

© Disney

"All you can do sometimes is just press harder on your pencil to try and make the drawing express what your're feeling in your heart, and you hope that the audience can feel it as they're looking at it."

– Glen Keane

Traditional Animation is a fascinating form of art with roots dating back to the early nineteen hundreds. Throughout the twentieth century, the Western world witnessed its proliferation; feature-length animations were presented in the movie theaters, and TV cartoons became daily entertainment for the masses.

In its purest form, traditional 2D animation is a prohibitively labor intensive process. Each frame of an animated production is drawn, cleaned up, and inked by hand. This process results in millions of drawing for a feature length film.

The advent of computers marks an important step in the evolution of animation. The transition from cel animation [Laybourne, 1998] to computer assisted 2D animation however, has not been as easy as some may have hoped. In 1978, Edwin Catmull writes [Catmull, 1978]:

> *In the last few years several systems have been written for aiding in the conventional twodimensional animation process. [...] While there has been some success and a great deal of optimism, the promise of higher output and quality using a computer has not been realized. The transition from simple drawings optimized for use on the computer to the complicated and detailed drawings of quality conventional animation has been much harder than expected.*

Important progress has been made in the past thirty five years. Digital tools are used for many aspects of modern 2D Animation pipelines, offering a wide range of tools, like management of asset creation and tracking, generation of visual elements and special effects, color management for lighting of scenes, and digital compositing.

Despite this progress however, the core challenge — drawing animation frames — is still very labor intensive. To date, the state of the art in digital tools is still not sufficient to automate the creation of high quality 2D Animation. Let us consider why this is the case.

One observation is that digital drawing hardware and software solutions do not match the fluidity, natural feel, and expressiveness achieved with pencil and paper. This is reflected by the fact that still today artists often prefer to draw on paper and have the drawings scanned and vectorized afterwards. A second observation is that, even as drawing tools improve and more drawings are done digitally, in practice this fact improves the ability to use automation for the rest of the process only minimally.

The reason for this is the implicit nature of 2D drawings and artwork. 2D drawings are in general just an unorganized collection of drawn strokes. It

is the viewer of the drawing that implicitly perceives the structure and implied 3d embodiment of these marks. This is something natural for a human viewer to achieve, but in general, a daunting if not impossible task to achieve automatically with a computer algorithm.

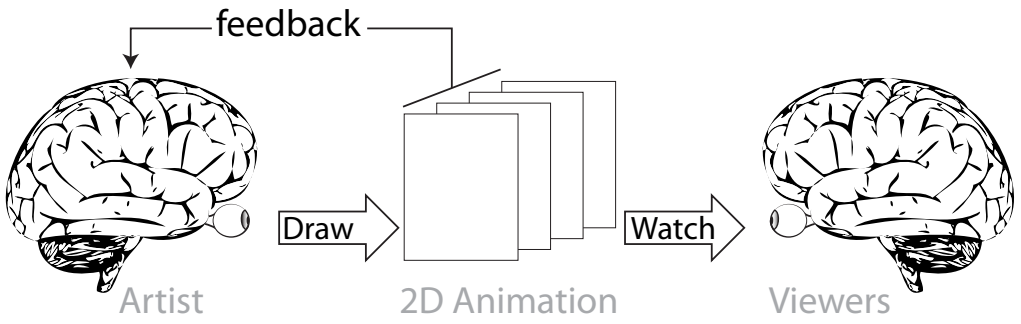

Figure 1.1: *2D Animation can be seen as the dispatch of a coded message. The message is the story portrayed in the animation. The story is encoded by the artist (by drawing) as a sequence of images. The audience receives the sequence of images and (by watching) deciphers it, each mind applying its own filter and obtaining a unique interpretation. Both the encryption and decryption rely on the human visual system and high-level cognition.*
On its own, the sequence of images holds enough information for a human audience to recognize the elements within the drawings, perceive the illusion of movement, and possibly build an emotional connection with the characters. As noted by [Catmull, 1978] however: "The problem of making a program infer the original object from its projections is akin to extremely difficult artificial intelligence problems."

This concept is illustrated with a metaphor in Figure 1.1. The artist on the left and the viewers on the right are humans, provided with a visual system and cognitive capabilities. Drawing and watching are processes in which a message (such as a story or emotions) is first captured into animation frames, and then interpreted while watching. The actual medium — the animation frames — carry the message in an "encrypted" form, and to be understood require, if not the actual animator's mind, at least human-like capabilities.

Before entering in the details of this work, we would like to make a few remarks about 2D Animation, and the difficulties one encounters in trying to support it with algorithms and digital tools.

While analyzing tools designed for artists, there are two important criteria to consider: the control over the result and the simplicity and immediacy

of the interaction. In both aspects, traditional animation tools set very high expectations. Right after the first few lines are drawn, an artist has an authentic feeling of how the drawing is going to be perceived by the viewers. The same applies to an animation, where by watching the sequence of drawings, the artist immediately perceives the animation as the viewers will. This feedback loop presents a powerful dynamic, in which the artist can judge his or her work while creating it, and even be inspired by it, possibly ending up with stronger art than originally planned.

Moreover, by giving the artist complete control over the drawing content, 2D Animation encourages artistic freedom and experimentation. As a discipline, animation developed a number of principles ([Johnston and Thomas, 1995]) and techniques, and nowadays, even when a drawing represents a scene of the real world, most rules that govern reality are bent or broken. Artists can (and often do) cheat on perspective, on physical behavior of objects, on their appearance, and they sometimes even define new law systems ([O'Donnell, 1980], [Gould, 1993]), with gags that become iconic to the genre.

A major risk when creating digital tools within such an environment is the one of imposing boundaries that could restrict either the control of the artists over the result, or the variety of results that can be produced. This is however a difficult mission, as by not introducing boundaries one reduces the prior knowledge that can be exploited to solve the problems at hand.

Overall, 2D Animation is a arduous domain. As engineer or researcher, one faces a number of challenges, such as: the inability to infer the 3D structure from the 2D images, the lack of prior knowledge of the content of the drawing and of the dynamics of the animation, the need for solutions that provide full artistic control to the users. This work is an attempt to tackle the core challenges of this domain, providing both the methodology and a set of solutions to progress toward the goal of computer assisted 2D animation.

1.1 Contributions

In this thesis we make the following contributions:

- The design of a novel toolset for the automation and assistance of the production of 2D Animation. We state a set of design principles, design a structure for the toolset, and develop a number of key components, with an outlook for further development.
- A study of the problem of defining an appropriate digital representation for drawings and animation data. We discuss the short comings of

existing representations and propose a set of ideas to design a hybrid vector-raster, line-area description that takes advantage of the (mutually exclusive, yet complementary) features of raster and vector image representations, and propose a unified description of shapes (such as lines or areas) to provide a novel editing platform for drawing elements.

- A vectorization system for line drawings designed to reconstruct high-quality centerline strokes, with particular care for junction regions and legacy artwork. Our method include two phases. First, a pixel-walk simulation exploits the gradient information contained in the raster image to create bands of moving pixels that converge at the stroke centerlines. Second, a reverse drawing procedure evaluates possible configuration scenarios at junctions and applies an empirically driven criterion to pick an adequate solution.

- A new scribble-based interface for user-guided segmentation of digital sketchy drawings. We introduce a novel energy minimization formulation in which both geometric and temporal information from digital input devices is used to define stroke-to-stroke and scribble-to-stroke relationships. Despite the problem being NP-Hard, we employ a simple heuristic that leads to a good approximation and permits an interactive system to produce accurate labellings even for cluttered sketchy drawings.

- An computer assisted solution for inbetweening of clean line drawings, with a focus on interactivity and artistic control. The system works by establishing a correspondence between the strokes of two given keyframes, and automatically computes morphing trajectories that respect arc motion and connectivity of the strokes. A set of interactive tools are designed to allow the user to modify the trajectory and account for differences in the drawing topology.

- A system to control the temporal noise in sketchy animation. The input animation is processed in two phases. First, a set of keyframes is selected, and for each pair of keyframes, an image-based motion extraction is used to establish a stroke-to-stroke correspondence and drive an inbetweening which results in the creation of a "noise-free sequence". Second, for each frame in the input sequence and its corresponding frame in the noise-free sequence, the same steps (motion extraction, correspondence, and inbetweening) are used to create a "variable noise sequence" which has the desired amount of temporal noise.

1.2 Design Principles

Upfront, we state a few design principles that have guided us throughout this work:

- *Keep the artist in the loop.* 2D animation is a tough research domain as it deals with the lack of computer readable information. As we will see in Chapter 2, despite almost four decades of research, the core steps of the production pipeline are still very labor intensive. We believe the main reason is that most attempts only considered fully automated solutions, ultimately cutting the artist out of the loop. This methodology has two consequences. First, it makes the problems much harder, due to the necessity to mathematically model very complex processes related to visual perception and object recognition. Second, it collides with the next design principle.
- *Seek artistic control.* 2D animation is a form of art. Human expression is its essence. Any solution assisting 2D animation should not hinder expressibility by imposing limits on what can be created. Any automated process should be controllable, allowing control over the extent to which solutions are carried, accepting guidance in solving the problems, and allowing for correction of parts of the results.
- *Close to industry.* The development of tools can benefit by collaborating with industry professionals. Real production scenarios, in terms of efficiency requirements and complexity of the data, provide a base line to define the performance of the tools. Feedback from professionals is the key in identifying the production bottlenecks and learning which tasks require more artistic control, which in turns suggests where automation and computer-assisted solutions should be targeted.

1.3 Organization

The goal of this thesis is to study and realize a novel toolset to support 2D animation. In this section we briefly discuss the general structure, and then dive into the work packages. The proposed toolset is organized into different categories, as shown in the overview Figure 1.2.

As “input” to the toolset, i.e. the interface from the real and digital worlds, we consider two scenarios:

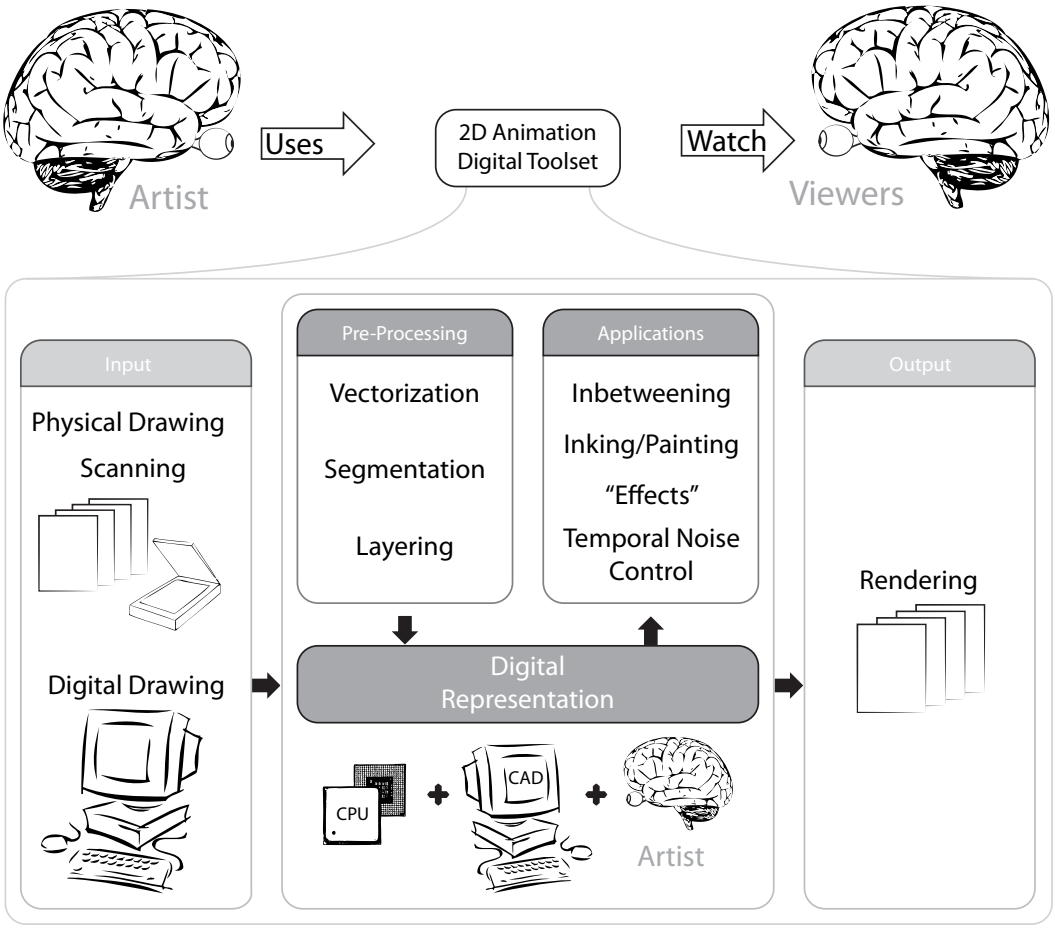

Figure 1.2: *2D Animation toolset Overview. We envision a system that supports the artist, providing automation and assistance under artistic control. By keeping the artist in the loop, the problem is no longer the full understanding of the animation content, but rather the design of proper tools. We have identified a number of core pieces of technology, and addressed them throughout the thesis. This figure serves as an overview to understand the overall design.*

Digital Representation	
1. Drawing Representation	(Chapter 3)
Pre-Processing	
2. Vectorization	(Chapter 4)
3. Segmentation	(Chapter 5)
Applications	
4. Inbetweening	(Chapter 6)
5. Temporal Noise Control	(Chapter 7)

Table 1.1: *This thesis is organized in five work packages, tackling different aspects of the toolset.*

- In order to be compatible with legacy artwork, such as drawings made prior the digital era or any sketch or drawing that is made on real paper for convenience or comfort, physical drawings can be scanned with the appropriate equipment, and then the resulting raster image vectorized to the appropriate digital representation.
- Alternatively, a digital drawing program can be used to directly produce drawings in the appropriate digital format.

The core of the toolset is then partitioned into “pre-processing”, “digital representation” and ”applications”.

The *digital representation* acts as a core piece of technology that binds the other components. The challenge is to design a unique representation for drawings that is easy to create and supports a variety of applications. Raw raster images, such those produced by common scanners and drawing packages, are inconvenient when it comes to any form of advanced editing. A number of abstract elements, such as lines, strokes, and areas, become key elements to widen the domain of applications of a digital toolset. By *pre-processing* raw images with the help of the user, important vector elements that match the intended application can be extracted and organized into a proper structure. The industry value of a digital tool or framework is measured by its impact on efficiency, quality of the results and working conditions. Guided by discussions with professionals, we propose a few *applications*, which are special either because of the impact on production efficiency or because they enable new styles of art.

1.3.1 Work Packages

The development of a full toolset is a challenging and ambitious task, even for the time-frame of a Ph.D. Thesis. Work packages, targeting a specific tool or category of tools, have been defined early on. They represent work milestones, distributed over a period of three and a half years, which resulted in the submission of research papers to different conferences and journals. This thesis includes 5 work packages, as shown in Table 1.1.

What follows is a description of each work package, including a brief motivation and description of the work that has been done.

1. **Drawing Representation**. We study the existing digital representations of drawings in the context of rendering and editing, and define a set of requirements for an ideal representation targeting 2D Animation. We then proceed by presenting two conceptual contributions. With the splat-vector hybrid we propose the combination of splats with vector centerlines to combine the visual richness of raster data with the editing capabilities of a vector representation. With the lines-areas hybrid we propose the combination of centerline with boundary lines to describe lines, areas, and combinations of the two with a unique representation, avoiding hard transitions from different vector descriptions.

2. **Vectorization**. We are interested in the problem of vectorizing line drawings. This problem is important as it represents the link between all the existing legacy artwork drawn on paper and digital tools that manipulate the stroke centerlines, such a computer-assisted inbetweening. As such, this tool can be considered an enabling technology. However there is also scientific interest. While extensive research has been done in other areas such as Computer Vision and Medical Imaging on topics like edge detection and vascular imaging, existing solutions for the vectorization of line drawings are not reliable enough for practical use. In particular, problems occur when strokes overlap (generating junctions) or are drawn close to one another (nearby strokes).

3. **Segmentation**. We address the problem of segmentation: the grouping (or clustering) of strokes composing sketches and line drawings. Among other applications, segmentation enables the addition of depth to drawings, both in terms of the visibility and the deformation of individual components. Explicit layering is error prone, limited in its use, and requires one to know the use of the drawing in advance. We instead aim for an interactive tool that allows the extraction of part of the drawing on the fly, requiring minimal user input and providing an in-

tuitive interface. We opt for a scribbles-based interface that guides an optimization combining geometric and temporal information.

4. **Inbetweening**. We develop a computer-assisted solution for tight inbetweening, tackling one of the most important bottlenecks of 2D production. Inbetweening, the problem of creating transitional drawings between pairs of key drawings, is one of the most laborious and time consuming tasks of the pipeline. Artistic control, in terms of automatic only to a desired extent, plays a special role here, as the amount of artistic interpretation required to properly generate inbetween strokes varies greatly from area to area within the same shot. We propose a solution which automatically generates a first solution and allows the user to progressively refine the result using a set of interactive tools.
5. **Temporal Noise Control**. We want to design a method to control the temporal noise present in 2D animation, targeting a sketchy drawing style. This type of animation is traditionally used only in early stages of production, and then progressively replaced with cleaner styles later on. The goal of this project is to tackle the problem of temporal noise — the perception of line flickering in sketchy animation — with the goal of enabling this rich style to be used in final productions. Rather than plain suppression, we aim for gradual control of the amount of noise in a scene, suggesting the use of line flickering as a controllable visual effect to enhance story telling.

1.3.2 Document Structure

- Chapter 1, which you are reading right now, gives an introduction to the goals and organization of this thesis.
- Chapter 2 sets the background for this topic, by diving through the state of the art in automation and assistance of 2D animation. We also discuss the production pipelines of classic 2D animation, highlighting their challenges and needs.
- Chapters 3, 4, 5, 6, and 7 present the work packages listed in Table 1.1.
- Chapter 8 summarizes the thesis and discusses the major contributions, giving an outlook for future research.

1.3.3 Publications

This thesis is based on the following accepted peer-reviewed publications:

B. WHITED, G. NORIS, M. SIMMONS, R. W. SUMNER, M. GROSS, J. ROSSIGNAC. BetweenIT: An Interactive Tool for Tight Inbetweening. In *Proceedings of Eurographics, volume 29, number 2, pages 605-614, Norrköping, Sweden, May 2010.*

G. NORIS, D. SÝKORA, S. COROS, B. WHITED, M. SIMMONS, A. HORNUNG, M. GROSS, R. W. SUMNER. Temporal Noise Control for Sketchy Aniamtion. In *Proceedings of the 9th International Symposium on Non-Photorealistic Animation and Rendering (NPAR'11), pages 93-98, Vancouver, Canada, August 2011.*

G. NORIS, A. HORNUNG, R. W. SUMNER, M. SIMMONS, M. GROSS. Topology-Driven Vectorization of Clean Line Drawings. To appear in *ACM Transactions on Graphics*, and be presented at *SIGGRAPH 2013*.

G. NORIS, D. SÝKORA, S. COROS, M. SIMMONS, B. WHITED, A. HORNUNG, A. SHAMIR, M. GROSS, R. W. SUMNER. Smart Scribbles for Sketch Segmentation. To appear in *EG Computer Graphics Forum*, and be presented at *Eurographics 2013.*

CHAPTER 2

Background

This Chapter presents general fundamentals and related work in the area of 2D animation, with particular focus on challenges faced in professional production pipelines. This Chapter considers the problem as a whole, and leaves to the following ones the discussion of specific areas.

In Section 2.1, we give a high level description of the production pipeline for classic 2D Animation. In Section 2.2, we highlight the challenges of computer assisted cartooning, and define three core problems we believe hold the key for important advancement in this field.

2.1 Classic 2D Animation Pipeline

This section describes the characteristics of 2D feature animation productions [Fekete et al., 1995, Johnston and Thomas, 1995, Hahn, 2008, Whitaker et al., 2009]. A common abstraction is to consider this process as two phases: *pre-production*, characterized by a relatively low cost effort, with a small team of highly creative individuals that define the story, the characters, and the desired visual look, and the *production*, where a larger team is involved, and the final 2D animation is produced.

2.1.1 Pre-Production

Pre-Production is a planning phase where ideas are developed and the work organized before the actual production starts. It usually involves *a small group of writers and visual artists led by a director and a producer* [Whitaker et al., 2009].

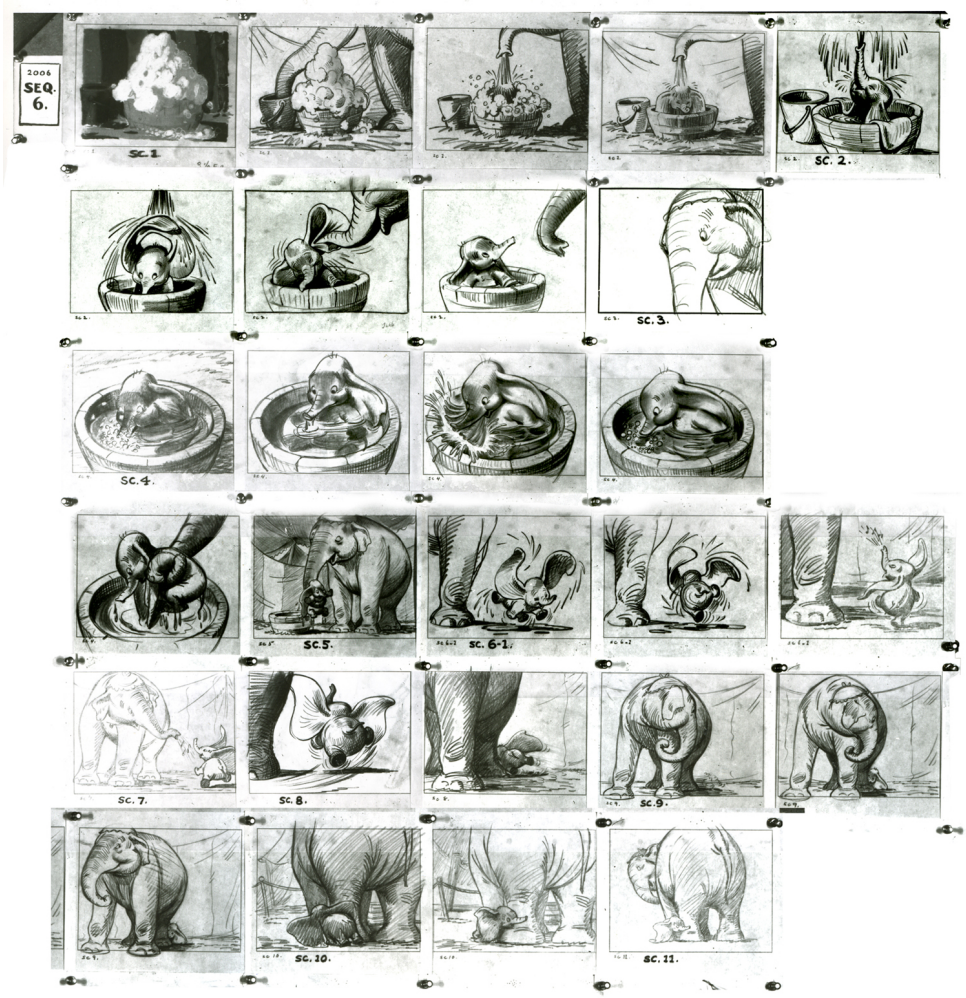

Figure 2.1: *Storyboard sequence from Disney's animated feature "Dumbo". Storyboards are a visual representation of the story, and help defining the story. "It is generally accepted that no production should proceed until a satisfactory storyboard is achieved and most of the creative and technical problems have been considered." [Hahn, 2008] © Disney*

The process begins with the creation of a storyboard (see Figure 2.1), which provides a visual representation of the story. The first complete storyboard was created by Walt Disney in 1933 during the production of "Three Little

Pigs" [Miller, 1957]. It is a form of pre-visualization where the story is organized as a sequence of illustrations or drawings, with the goal of visualizing the story and finding potential problems before they occur in later stages of production. Storyboards may be created with the help of Screenplays, textual descriptions of the story, possibly written as an adaptation of existing pieces, which include the movement, actions, expression, and dialogues of the characters.

In parallel with storyboarding, there is the character design. This process involves developing the appearance and features of the characters, considering the role of the character in the story and how the audience should perceive it.

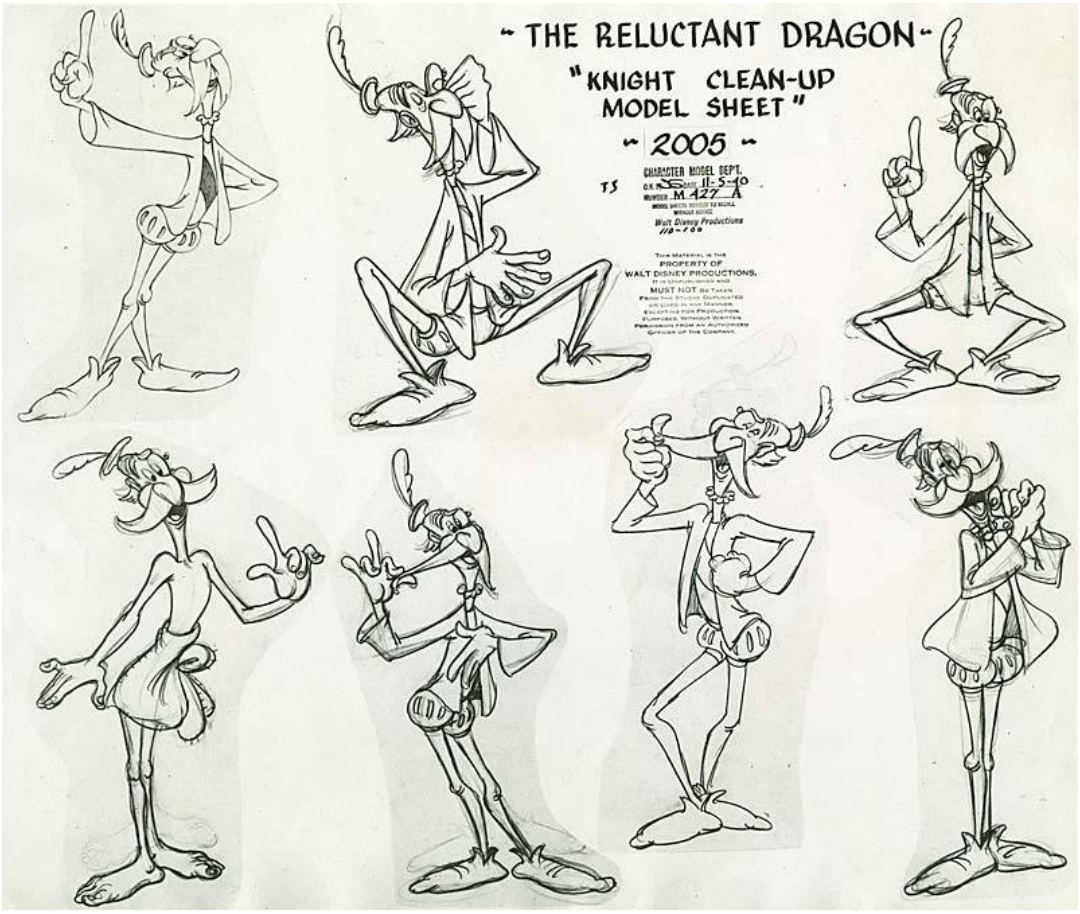

Figure 2.2: *Model Sheet of Disney's character "Sir Giles". This type of document describes the main features of a character, allowing multiple artists to draw the character consistently. © Disney*

One important aspect in productions involving multiple artists is the standardization of the appearance, poses, and gestures of the characters. This is

often achieved by the creation of "Model Sheets" (see Figure 2.2) — documents containing the details of each character — as well as "Characters Line-ups" (see Figure 2.3), which are depictions of different characters together in order to compare the scale of the characters against one another. At the end of the pre-production, a character may be assigned to a supervisor artist, typically an experienced animator, with the task of ensuring standardization throughout rest of the production.

Figure 2.3: *Characters line-ups are used to assess the relative size of characters. (Image courtesy of Alison Action)*

Nowadays, pre-visualization may also be used to get an early impression of the look and feel of the scene in terms of motion and timing. For this purpose mock-ups called "Animatics" may be produced. Animatics generally display a sequence of still drawings or rough animations to which dialog, sound or music are added in order to assess the effectiveness of the combination of the visual and auditory elements of the scenes.

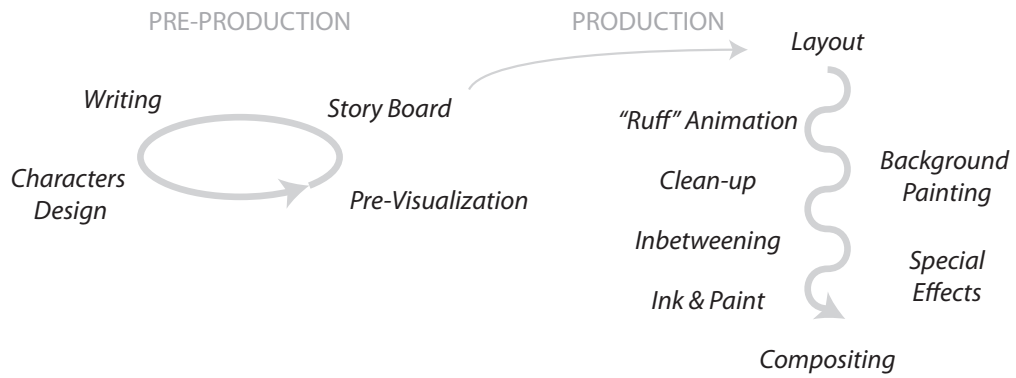

Figure 2.4: *The Traditional Animation Pipeline is characterized by two phases. In pre-production, a small team works to define all aspects of the story and finally produce the storyboard. In the second stage, the team grows and the actual production of the feature animation starts. Production includes a number of stages, and although often presented as a pipeline, dependencies may vary depending on a scene's complexity and content.*

2.1.2 Production

Production is characterized by a number of stages and roles, as shown in Figure 2.4. While presented as a pipeline, the dependencies are often defined by the content of the actual scenes. For example, a scene dominated by water effects - such a ship in a storm - may follow a very different production path than a scene where characters dance in front of static scenery [Noris, 2008].

In general, the following stages apply:

In "Layout" [Byrne, 1999], the staging for each scene is designed, including establishing the setting, choosing and placing character and prop elements, and specifying camera motion and cuts. This process is vital as it defines the rest of the production related to each scene.

Figure 2.5: *A painted background from Disney's feature animation "Pinocchio".*

The "Background" refers to a scenery typically drawn on a separate layer in front of which the animation takes place (see Figure 2.5). The role of background artists is to illustrate the sceneries according to the scene description and the overall style of the production, maintain continuity to replicate mood, lighting and details from one background to the next. Backgrounds are typically static, although a number of effects — such as trails of smoke or flags waving in the wind — may be applied afterwards during the effects phase. In early animation productions, backgrounds were manually painted on multiple big glass sheets, while today they are drawn digitally, using both 2D and 3D packages depending on the content of the background.

Character and effects animation is then done in multiple stages.

First, animators produce the subset of drawings that lay down the core of the action, called the "keys". These extreme drawings are often "ruff" versions that capture the spirit, flow, and arcs of the animation. "Clean up" artists are responsible for taking the ruff drawings and producing clean lines that remain true to the original intent.

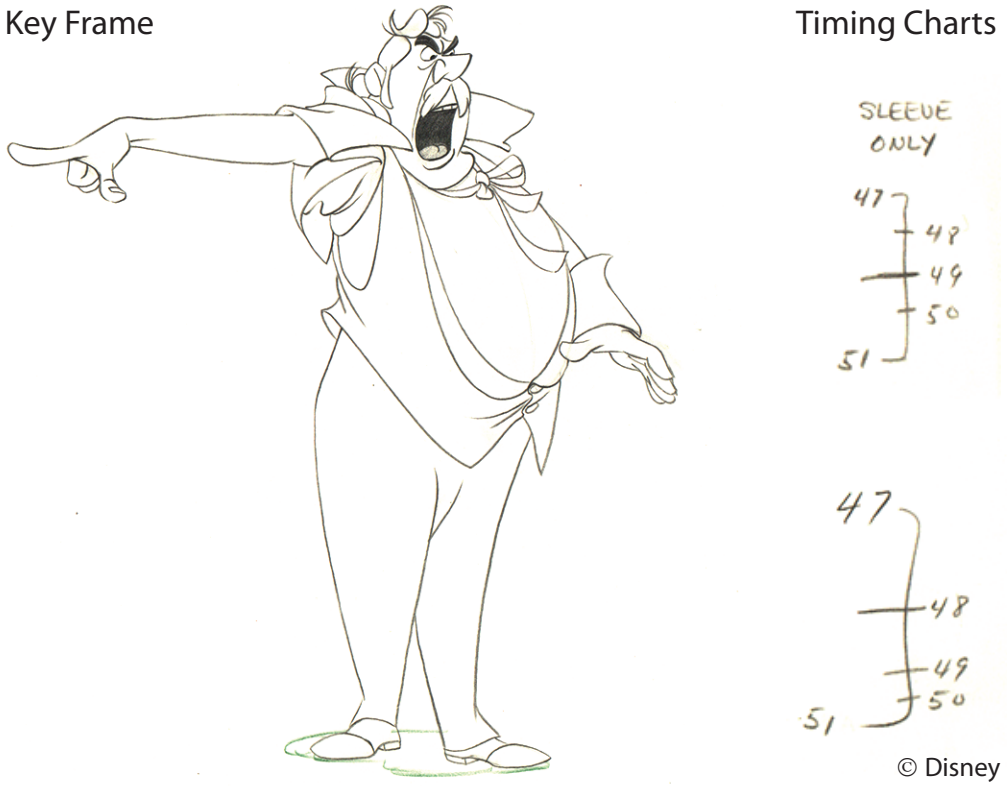

Figure 2.6: *A clean-up key from Disney's feature animation "Peter Pan". Timing charts are used to specify how inbetweens should be created with respect to the motion captured by the keys. In the case where different elements follow different animations, multiple charts may be used.*

Each key drawing has one or more *timing charts* (See Figure 2.6) associated with it. The animator uses these charts to specify how many drawings should be produced between keys, and at what intervals. It is the job of the "inbetweening artist" to draw the requested intermediate drawings to produce seamless motion. These transition drawings must remain true to the motion specified in the keys and should not detract from the action. The artists flip through the drawings as they work to ensure continuity and flow.

The drawings then go to ink and paint where regions are filled in with color, and lines may be retraced with colored ink. Originally, this would be done by tracing each frame on thin transparent sheets called "cels", which would then be colored.

Finally, compositing would close the process, by combining the different characters and background into a unique image. Traditionally, this would be made by stacking the different layers and taking a picture. A notable technological milestone is represented by the "Disney's Multiplane Camera", introduced in 1933, which allowed multiple layers to be independently lit and moved in three dimensions, creating realistic perspective movement and a more realistic depth impression [Telotte, 2010].

2.1.3 Computer Assisted Production

While some of the underlying mechanisms have benefited from the advent of digital technology, a 2D production today follows much the same basic work flow as traditional animation [Johnston and Thomas, 1995]. Computer-assisted solutions appeared in the late 1980s ([WDAS and Pixar, 1989], [Robertson, 2001], [Robertson, 1994]) and introduced a number of new features, such as raster based region filling algorithms, and roles, such as scanner operator and special effects artist.

The later stages of the production pipeline, namely *ink and paint*, *special effects* and *compositing*, witness the introduction of important features:

After inbetweening, drawings are scanned and sent to digital ink and paint for coloring. *The cleaned-up drawings are painted on a digital tablet that floods selected areas of the character with color.* [Hahn, 2008]. Diligent work is done in making sure regions are closed to prevent color from leaking. The state of the art in research proposes more robust coloring algorithms capable of dealing with gaps or color multiple closed regions with minimal user interaction [Sykora et al., 2009b], and we believe these improved tools could become valuable assets for the production frameworks.

Computer generated effects can also be produced, by computing gradients and similar coloring effects onto 2D shapes to create ground shadows or skin tones (see Figure 2.7), by completely generating rendered 3D scenes or objects to be used as backgrounds or solid objects, or by handling natural elements such as water, fire, smoke, or atmospheric effects [Hahn, 2008].

Digital compositing [Steve, 2006] enables sophisticated layer blending

Figure 2.7: *Effects such as shadows and skin tones can be generated from simple 2D shapes that overlay a character or a background.*

[Porter and Duff, 1984], as well post processing effects such as color correction, or color management using palettes [ToonBoom, 2010].

Despite the introduction of computer assistance, the production costs of feature animation have increased over the past twenty years ($23 million for "Beauty and the Beast" in 1991, $45 million for "The Lion King" in 1993, $105 million for "The Princess and the Frog" in 2009 [Amazon, 2012]). We believe this has the following reasons. First, the complexity of the scenes has increased as more effects were possible thanks to computer graphics and the studios pushed for richer visuals (see Figure 2.8). Second, new digital tools had a limited impact on drawing, which remains a considerable effort (over a million inbetweened drawings per full length production [Johnston and Thomas, 1995]).

Figure 2.8: *Two frames from Disney's animated features "Alladin" (1992) and "The Princess and the Frog" (2009). Notice the increase of visual complexity and number of animated characters.*

To better understand how digital tools could have a more substantial impact on the production of feature animation movies, in the next Section, we discuss the challenges that characterize computer assisted cartooning.

2.2 Challenges of Computer Assisted Cartooning

In this section we highlight the challenges related to the production of 2D Animations. This topic has been studied repeatedly in the past decades [Catmull, 1978], [Baudelaire and Gangnet, 1986], [Patterson and Willis, 1994], [Madeira et al., 1996].

2.2.1 Challenges

Computer assisted 2D animation is characterized by a number of challenges which are due to the encoding of the animation through the medium of drawings.

Drawings as Medium. A drawing presents a medium that is difficult to read for a machine. There are a number of challenges in relation to the lack of depth information, misleading depth cues, and the use of simplistic digital representations.

1. **Missing depth.** The major challenge in working with 2D animation is related to the loss of information (see Figure 2.9).

   The principal difficulty is that the animators' drawings are really two-dimensional projections of the three-dimensional characters as visualized by the animator, hence information is lost. [Catmull, 1978]

   Depth Ambiguity. Line drawings represent objects by their silhouettes and visual features. Strokes often overlap with depth discontinuities. There are however intrinsic ambiguities in the understanding of depth [Kawabata, 1997], as well as limits in the representation of 3D shapes using lines [Cole et al., 2009]. As a result, reconstruction of depth from line drawings is often not feasible.

2. **Misleading depth cues.** The extraction of depth information from drawings is hard as misleading depth cues may result from inaccuracies or deliberate artistic choices.

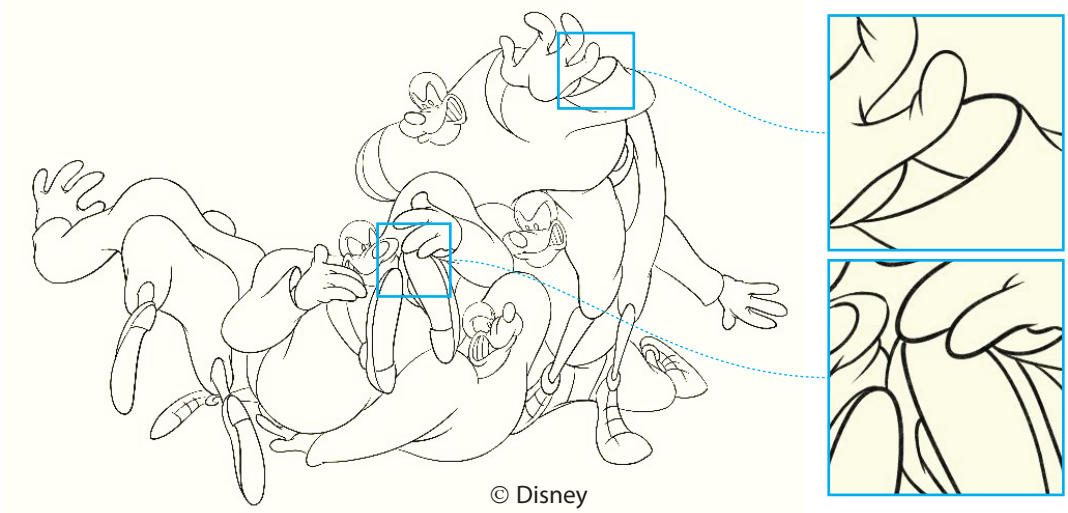

Figure 2.9: *Drawings are 2D projections of 3D characters and objects. The resulting loss of depth information is one of the principle challenges of computer assisted cartooning. Reconstructing the depth algorithmically is very hard. Locally (see the close-ups), stroke structures can be quite intricate and sometimes difficult to understand even for a human viewer. Globally, the interaction of characters and objects results in complex occlusions that make the task of object recognition very hard, even when strong prior is available (which in cartoon drawings is often not the case).*

The meaning of a line. As shown by [Cole et al., 2012], even in cases where artists draw well-defined 3D objects, most drawings contain a certain number of lines that do not necessarily have a clear importance with respect to the 3D context. Culture, as well as artistic style and historical evolution of the medium may influence where lines are drawn and how they are perceived by the audience. Discriminating between reliable and unreliable feature lines may be difficult.

3D correctness. The representation of 3D objects in a scene is not necessarily correct with respect to perspective rules or relative proportions of different objects. Incorrect perspective and proportions can be found spatially within one image, and temporally, considering a sequence of images. As a result, the connection of 2D drawings with 3D representations is weakened. For instance, the calibration of a camera using

proxy geometry or the reconstruction of 3D geometry (such as [Öztireli et al., 2011]) may not be possible in the general case.

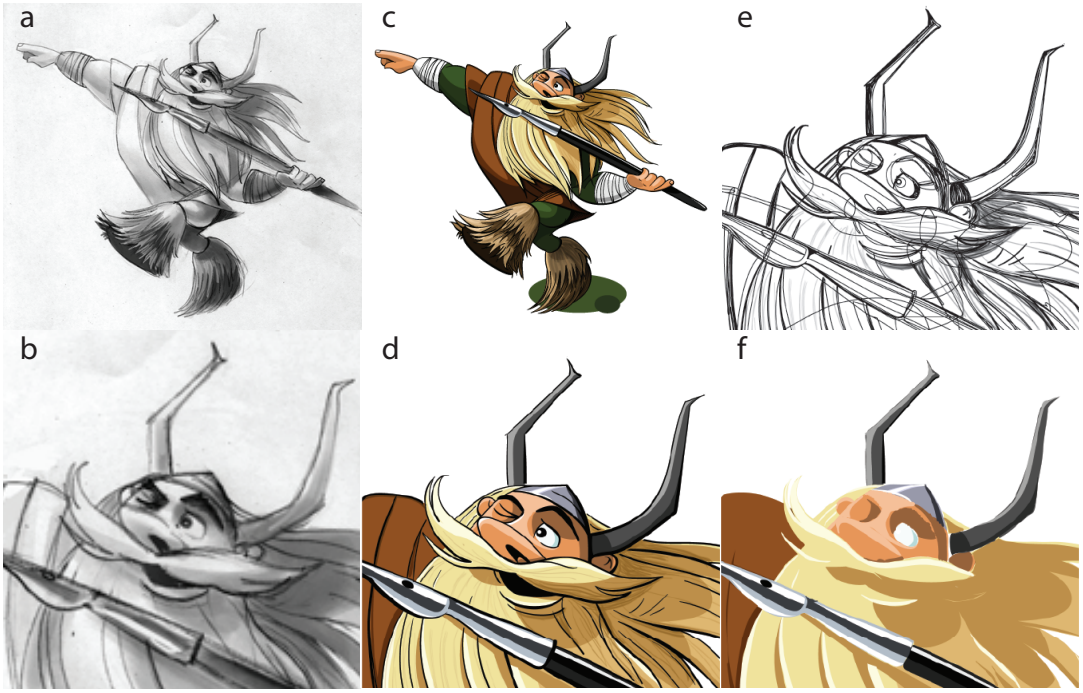

Figure 2.10: *Raster images (a, b) capture fine visual details, but offer no mathematical description of the content other than the pixel grid. Vector images (c, d) describe elements in vector form, allowing to modify individual elements, but offer less visual customization. Additionally, the vector elements (e, f) are typically unstructured sets of strokes and areas, and require extensive manual work to be organized into proper semantic groups. (Sketch by Catherine Dedova, vector illustration by Laroslav Lazunov)*

3. **Lack of appropriate semantic abstractions.** Existing image representations and vector objects provide limited support by encoding data inconveniently.

   Low-level representations. Raster and vector images offer mutually exclusive features, as shown in Figure 2.10. In the context of 2D Animation, where line drawings are animated, neither representation is sufficient.

   In order to be edited, Pixel grids require vectorization to extract stroke networks, with poor results at junction regions or

where strokes are drawn next to one another (See Chapter 4 and Figure 4.1).

When instead vector graphics tools are used to generate the drawings, strokes are often collected in unstructured sets and require extensive manual work to be explicitly organized in proper layers, depending on the use. In general, the useful semantic information is not explicitly represented and needs to be extracted.

Line vs. Area. Lines and Areas are often represented with different mathematical descriptions. In a drawing however, such distinction does not always match the drawing content. A line of growing thickness can eventually become an area. A group of line defining a closed boundary are in essence defining an area. Current vector representations however lack a unique mathematical construct to represent both areas and lines and seamlessly transition from one to the other.

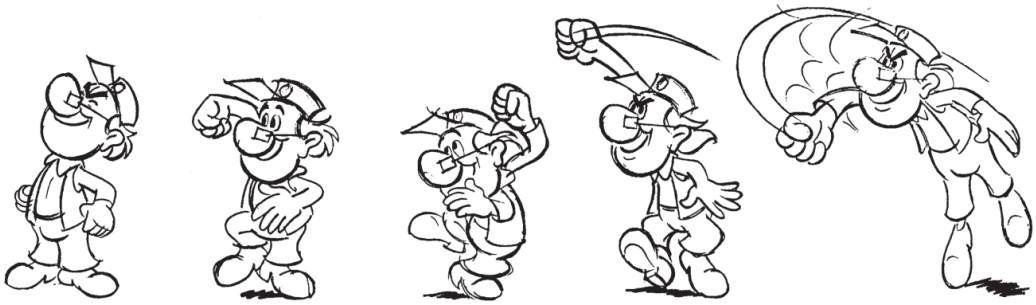

Figure 2.11: *Even a simple animation where a character grabs something in the air can be a challenging domain. Occlusions make the task of finding correspondences between the drawing very difficult. (Image source [Whitaker et al., 2009]).*

Sequence of drawings. A sequence of drawings composing a scene presents a second layer of complexity due to the introduction of a discrete temporal dimension, which can have positive or negative effects with respect to the challenges expressed above. Indeed, a coherent understanding the sequence of drawings is necessary to process a scene. We consider the following challenges:

1. **Topological changes of depth discontinuities.** Object transformations, such as rotations outside the image plane or shape

deformations, typically create topological changes in the discontinuities of the depth (see the character's right hand in Figure 2.11). Discontinuities have a major impact on the difficulty of the correspondence problem, as discussed later.

2. **Unpredictable transformations.** Despite the existing attempts to define the rules of cartoon physics and dynamics ([O'Donnell, 1980, Keane, 1987]), there is no set of rules that applies to the general case. Characters and objects may be subject to extreme, sudden transformations (see Figure 2.12), which in extreme cases may not be trivial to understand even by the audience.

3. **When is an animation correct?** Animation is often an iterative process, where a set of guidelines and principles [Johnston and Thomas, 1995] are applied to try and improve the result. There is however no notion of correctness for a given sequence, as each animator will produce animations based on his or her unique style, and keep improving small details if time allows it. It is therefore not clear what makes an animation correct, and how an algorithmic system could judge that a result converged to the right solution.

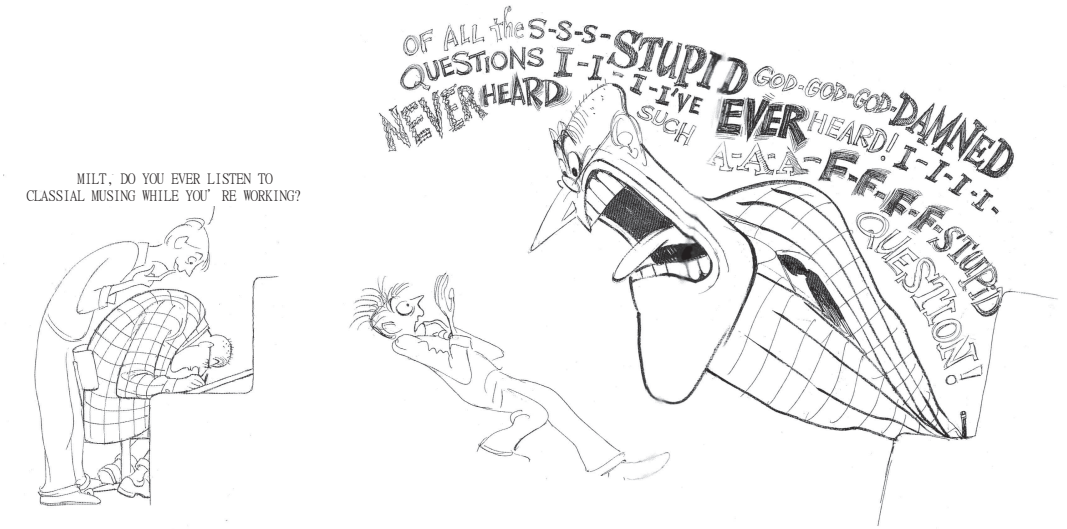

Figure 2.12: *Capturing mathematically the physical laws of cartoons is not simple, as characters and object are subject to extreme transformations, which do not obey real world rules. (Image source [Richard, 2002]).*

2.3 Core Problems

In a visionary perspective, the ultimate goal of computer science applied to 2D Animation would be the conception of a computational model able to reach a semantic understanding of the subject similar to the one of human beings. It would mean, that computers would be able to fully understand animation and drawings, and therefore assist (and potentially replace) the artist in their creation. This however is far from what can be realistically achieved in the near future, and if we ever reach that, it is likely through a collective effort across different computer science disciplines.

In this work, we consider the effort of identifying a small number of key sub-problems which present a convenient balance between their complexity and the impact of their solution to the advancement of the field. During this work, we encountered different instances of what we believe constitutes three core problems in the assistance and automation of 2D animation. Notice that these problems incorporate the challenges discussed in the previous Section.

Our insight is that solving these problems holds the key to revolutionary changes in this field. We do not pretend to solve these problems fully within the scope of this work. Instead, we provide some tools and ideas to approach them, with the hope to facilitate future research and development efforts. Figure 2.13 lists a number of techniques or areas of interest which are relevant to these problems, and points to the appropriate Chapters in this work.

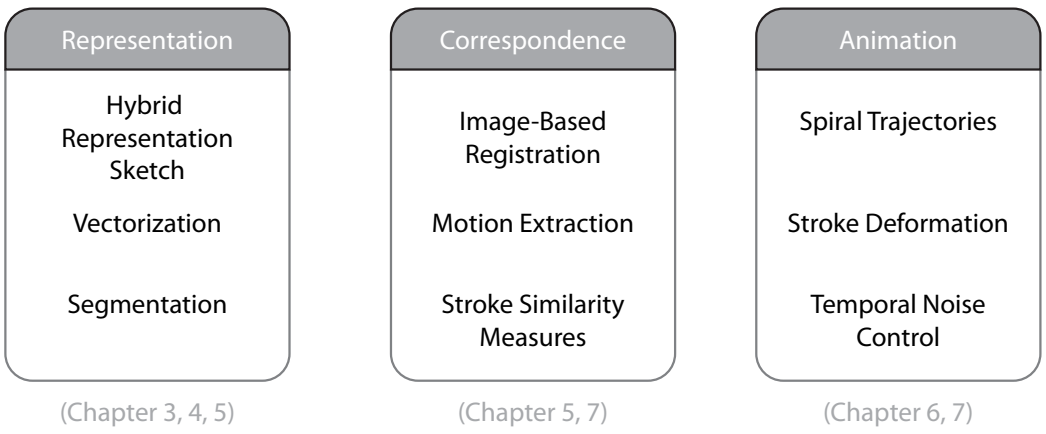

Figure 2.13: *In this work we identify and address three core problems in the field of computer assisted 2D animation: the representation, the correspondence, and the animation problems. For each problem we list relevant techniques and point the reader to the appropriate chapters.*

The representation problem. Given a drawing, or a sequence of drawings, we consider the problem of extraction and organization of data. In particular, we consider the extraction and representation of strokes and areas, of visual appearance, and of temporal evolution (i.e. motion or visual change).

The complexity of this problem lies in finding the appropriate mathematical abstractions with respect to the desired properties and the requirements of usable implementations.

We consider this problem as fundamental, as the lack of appropriate representations creates a number of dependencies between separate entities of a production pipeline, and limits the benefits of computer-assisted or automated solutions for 2D animation.

The correspondence problem. Given a set of two or more drawings, consider the problem of finding the correspondence of the drawing content. If drawings are represented as sets of curves, the problem is to find a stroke-to-stroke mapping between two or more sets. Similarly, if drawings are represented with areas, a mapping between the areas, and possibly between their boundaries, has to be found.

The complexity of the problem depends partly on the complexity of the data. Both the extent of the topological changes in depth discontinuities, as well as the complexity of the transformation happening over time — the animation — directly influence the difficulty of this problem. Additionally, the appropriate properties of the mappings have to be derived.

This is a fundamental step as it enables a variety of applications, such as inbetweening, automatic color propagation, and local motion extraction. These applications can have a great impact in the cost of a 2D animation production, and represent the main source of interest for applied research in this field.

The interpolation (or animation) problem. Given a set of drawings with correspondence defined between their elements, consider the problem of interpolating the drawing content in a way that respects the geometric and temporal constraints, as well as the principles of animation.

The complexity of the problem lies in capturing the animation principles (e.g. motion with arcs [Johnston and Thomas, 1995]) and enabling interactive artistic control, whose need is emphasized by the lack of a clear definition of correctness in animation. Addressing this problem

requires a good understanding of traditional animation and how animators work on paper.

This problem is sensitive as it deals with bringing technology very close to where the magic, or art, of animation happens. The simplicity and full artistic control of drawing on paper must be respected in order to provide artist with tools they trust and find satisfying. Automation should happen only where the amount of artistic interpretation is minimal. Interaction metaphor as well as hardware interfaces can have an important influence on the quality of the result.

CHAPTER 3

"Sketching" a Digital Representation

In this Chapter, we address the the problem of defining a digital representation suitable for the needs of 2D Animation. First, we explain the role of a digital representation in the context of this thesis. Then, we proceed by describing the existing representations, highlighting their advantages and disadvantages. Finally, we propose a sketch for a novel representation, called "hybrid" that aims at combining the rendering capabilities of raster images and the editing capabilities of vector images into a unique vector-raster description.

This Chapter presents a collection of ideas developed during the course of this Ph.D. study. We present some proof-of-concept solutions, though with a stronger focus on the conceptual contributions and ideas, considering it as a sketch for future research.

3.1 A Middle Ground between Applications and Pre-Processing

A digital representation plays a central role in the support of a production environment. The organization of data influences what is possible to achieve with such data, as well as the difficulty to reach a particular goal.

Two important aspects of a production pipeline have to be considered when designing a digital representation. First, one needs to consider the source of the data. Especially in cases where data comes from real world, a translation — or pre-processing — of the data will be necessary. In the case of 2D Animation, this would include digitization and formatting. Second, the whole purpose of digital support is to provide solutions for specific applications. Which applications are of most importance may influence how data is organized.

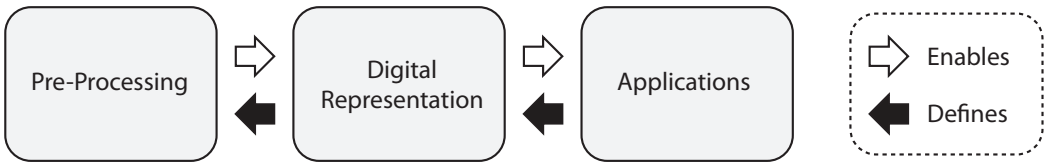

Figure 3.1: *Applications, Digital Representation, and Pre-Processing tools are dependent on each other. There is a "Enabling" forward dependency where Pre-Processing tools extract information from drawings to format it into a desired Digital Representation, and in turn such representation enables a set of applications. Specularly, there is a "Defining" backward dependency where the goal application define a number of requirements that a Digital Representation should meet, which in turn require particular Pre-Processing of the raw drawing data.*

In our work, we reflect these thoughts by defining three categories of tools (as introduced previously, in Figure 1.2): pre-processing, representation, and applications. The dependencies between these categories are further discussed with Figure 3.1. Consider the following perspectives:

- Pre-Procesing raw data makes it possible to format a drawing into specific Digital Representations. In turn, digital representations enable a number of applications. We call this the "Enabling" dependency.
- On the other hand, Applications define a set of requirements that a digital representation should meet. A digital representation requires data

to be organized in a way that does not necessarily match the raw input of scanners or digital drawing tools, resulting in the need for tools to extract and transform data appropriately. We call this the "Defining" dependency.

Having a clear plan right from the beginning is however not always possible. We had to proceed by studying the two ends first, with the hope to find a common ground in the middle.

In specific, we started by studying a few applications scenarios (Inbetweening in Chapter 6, and Temporal Noise Control in Chapter 7), that are considered important with respect to the production pipeline (See Chapter 2). During the process we learned a number of requirements that an ideal Digital representation should meet. Then, we have worked on important Pre-Processing tools (Vectorization in Chapter 4 and Segmentation in Chapter 5), learning what we could achieve in terms of processing of raw drawings.

This Chapter summarizes what we have learned. In order to present our ideas, in Section 3.2 we give an overview of existing digital representations, highlighting advantages and disadvantages with respect to capturing the visual richness of drawings and enabling editing. In Section 3.3 we list a number of requirements that we believe are important for a digital representation. Finally, in Section 3.4, we collect our ideas into the sketch of a novel digital representation for drawings.

3.2 Existing Digital Representations

The most common representations for digital drawings - raster and vector graphics - have complementary but mutually exclusive properties. On the one hand, raster images capture line art details down to the pixel level, but all edits are also restricted to the pixel level. On the other hand, vector graphics define a semantic abstraction of the content that allows for sophisticated editing operations [Coyne et al., 2007], but the abstraction process, which is generally a compression as it reduces the amount of data being stored, often loses some of the fine-scale detail.

In digital production environments, the choice of raster or vector graphics solutions has a great impact on the possible uses of the drawing. Raster solutions typically allow greater control over the look of the drawing, offering a variety of brushes and effects which mimic various real world techniques. Exemplar uses are concept art, characters and vehicles design, and

other forms of static drawings where the visual impact is predominant. Vector solutions are preferred for anything that deals with digital typography, paging, and printing, such as posters, brochures, magazines, as well as technical illustrations and schematics, or within solutions providing morphing or animation capabilities, such as flash animation [Adobe, 2012a].

In 2D Feature Animation the separation of these two representations is a fundamental issue. Even during the current digital age, 2D animation drawings are often created using pencil sketches on paper. This traditional way is still preferred by many artists over fully digital solutions due to the higher drawing quality and richness of detail. These line drawings are then scanned and vectorized for further processing in the digital movie production pipeline. Advanced 2D animation tools, such as automatic inbetweening, inking, and painting are forced to convert between the two representations to perform different types of operations. This conversion process generally decreases quality and sacrifices many properties of the original drawings, such as stroke texture and subtle details.

Additionally, while vector solutions can support animation to a certain extent, the underlying vector elements are not unified. Lines are typically represented as centerlines with thickness and a color associated to either the control points or the whole line. Areas are typically represented by their boundaries, again with color, or texture information. Discriminating between lines and areas is not always trivial, leading to compound objects. Separate descriptions require separate editing tools, adding an extra burden both to the development efforts and to the user learning and working experience.

3.2.1 Brush Models

Brush models are a key feature of any drawing program, and should be taken into account when defining a digital representation for drawings. Here we discuss the two main categories of brush models.

Raster-Based Brushes:

The raster canvas defines a persistent, constant-sized medium that stores the effect of the user interaction with the brush. Raster based brushes exploit this by utilizing efficient per-pixel operations that modify the drawing as needed, accessing data locally in space and time.

A well known raster approach is the 'stamping' method. Predefined stamps, i.e. bitmaps describing texture patterns, are consecutively composed on the canvas by spatially following the stroke trajectory. To sample the trajectory to

the raster grid, variations of the classic line algorithm [Bresenham, 1998] may be used. Different forms of dynamic behaviors, such as change in stamp's size, opacity, orientation, and others are typically available in today's digital painting frameworks [Adobe, 2012c, Autodesk, 2012], allowing to create a wide range of effects [Hall, 2010].

Data-driven approaches aim to faithfully represent real world brushes. Sophisticated models include simulations of physical brushes, where the behavior of bristles and filaments is captured and simulated in order to reproduce natural behaviors while painting on a canvas [Saito and Nakajima, 1999, Chu and Tai, 2002, Baxter and Lin, 2004, Baxter and Govindaraju, 2010].

Vector-Based Brushes:

With vector solutions, information is stored as part of a vector model, and the raster grid is typically considered only at the rendering stage. This brings major advantages in the ability to edit the drawing, but makes the vector data grow with the drawing's complexity, potentially causing performance issues both while editing and rendering.

Similar to PostScript [Warnock, 1982], Skeletal brushes [Hsu et al., 1993, Hsu and Lee, 1994] represent a well known approach for vector representations. Strokes are captured by their centerline, a given thickness, and some rules on how joints should behave. Given a particular pattern or texture image with prescribed backbone, custom appearance can be obtain by deforming such an image to follow the stroke.

Another approach [Ando and Tsuruno, 2009] proposes a method called Segmental Brush Synthesis, where a set of predefined strokes textures is taken and applied to segments composing the brush stroke. This method supports more complex visual appearance at interactive rate.

3.2.2 Representation Models

We now consider actual representations, discussing the two most critical aspects: rendering and editing.

Rendering

In this section we discuss how popular digital representations handle rendering. The subject of editing capabilities, which is of great importance for this Chapter, will follow in Section 3.2.2.

With raster images, rendering is a trivial operation where the pixels composing the image are shown on the screen. When multiple layers of raster images are overlaid, blend modes can be used to regulate the computation of the final color [Valentine, 2012]. In the general case, however, most of the visual complexity is captured by the image content, rather than by the rendering process.

Vector representations propose different approaches to rendering:

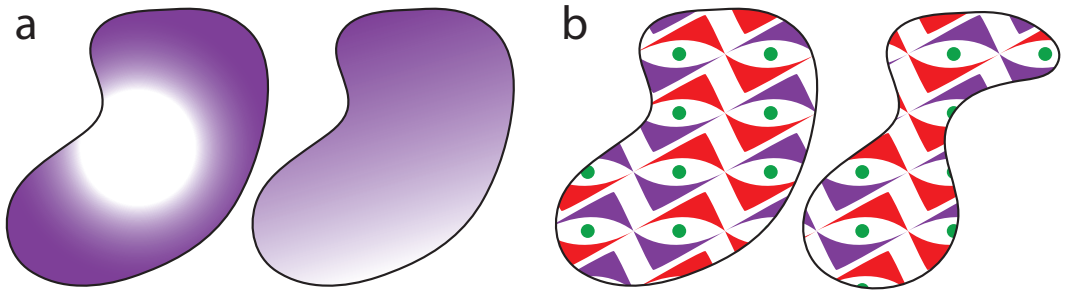

Figure 3.2: *Appearance customization of closed paths in Adobe Illustrator: radial and linear gradient modes (a), and pattern mode (b). Notice how the appearance of the interior is not affected by the actual shape of the boundary, but rather just by its bounding box.*

As an extention to PostScript, Adobe Illustrator takes advantage of open and closed paths and proposes a vector-based representation featuring a visually seamless integration of skeletal-brush lines and closed-regions, defined by their boundaries and rendered either as flat shaded surfaces, or using simple gradient and pattern modes (See Figure 3.2). The major visual short coming of this representation is the limited support for complex pixelated textures or gradients, which may require a huge number of closed-regions with complex shapes (See comparisons proposed in [Zhang et al., 2009]).

Gradient Meshes overcome some of these region problems by allowing to specify gradient functions over simple, well-structured meshes (See Figure 3.3). Methods like [Sun et al., 2007a] automatically compute optimized gradient meshes to match an input image. To allow intuitive editing, Diffusion Curves ([Orzan et al., 2008a]) abstract from the need of a mesh, and instead use curves with associated gradient colors, allowing to represent soft gradients and blur with intuitive lines and less editing burden.

While these methods allow for greater complexity of appearances, they are not designed to represent the visual complexity allowed by raster solutions.

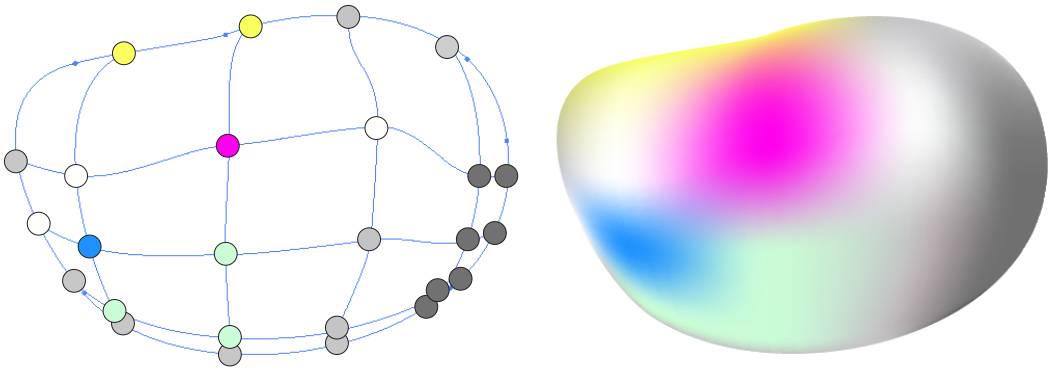

Figure 3.3: *Gradient Meshes allow to reach more complex visual appearance by computing smooth gradients between colors applied to the nodes of arbitrary meshes.*

Harmony [ToonBoom, 2010] addresses this problem by combining texture with vector data as shown in Figure 3.4. In the vectorization process, the skeletal lines (red) are extracted, as well as the boundary lines (blue) of strokes and areas. Additionally, the original drawing pixels are used as texture, which is masked by the boundary lines.

Editing

As discussed in Chapter 2, one of the issues with existing vector representation is the dichotomy of Strokes and Areas as distinct elements. In this Section we discuss its impact on editing.

First, let's consider the following definitions:

- **Line Editing.** Line editing refers to deformation and manipulation techniques that apply to open paths. This includes the morphing technique proposed in Chapter 6, as well as the manipulation tools available in professional products, such as Adobe Illustrator.
- **Area Editing.** Area editing refers to both boundary manipulation (close path editing) and space deformation algorithm, such as the ARAP techniques used in Chaper 7.

Figure 3.5 depicts two types of representations that are used in professional illustration and animation programs. We call them the "Separate Representation" (Illustrator [Adobe, 2012b]) and the "Mask Representation" (Harmony [ToonBoom, 2010]).

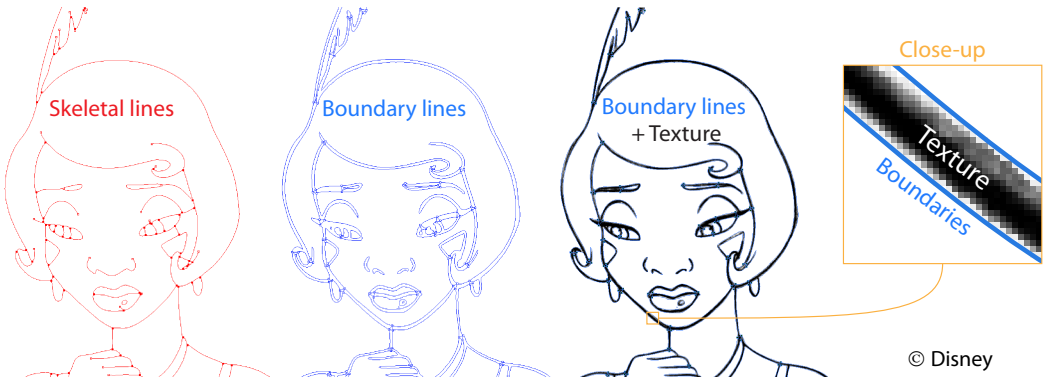

Figure 3.4: *In Harmony [ToonBoom, 2010], drawing are scanned and vectorized, extracting skeletal lines (red) and boundary lines (blue). The boundary lines are used to define a mask which is applied to the raster data. The subtle pencil or brush details are therefore captured.*

- The **Separate Representation** defines strokes as open paths, associated with a thickness profile. This gives a flexible editing representation as the open path can easily be deformed either with manual interaction or using morphing or interpolation tools. Limitations in the appearance customization are discussed in [Ando and Tsuruno, 2009].

  Areas are represented as closed paths. This gives a convenient way to edit the area boundaries. As we discussed in Section 3.2.2 however, the appearance, both in terms of its customization and how it is affected by boundary deformations, is quite limited.

  Individually, these representations are suitable for editing, as they allow to conveniently alter the shape of either lines or areas. The problem arises when a particular shape is a combination of lines and areas (See the *combination* case in Figure 3.5). In this case, separate representations are difficult to edit, and discontinuities may be visible at the transition point.

- The **Mask Representation** treats both lines and areas the same way, by considering a set of boundary lines that define a mask which is applied to the raster data. Conceptually, this representation has two important advantages over the alternatives: it captures the visual complexity of raster images, and has a uniform vector representation for lines and areas.

  The drawback however, is that the editing is very limited. By editing the boundaries, only the mask is affected. To really change lines or

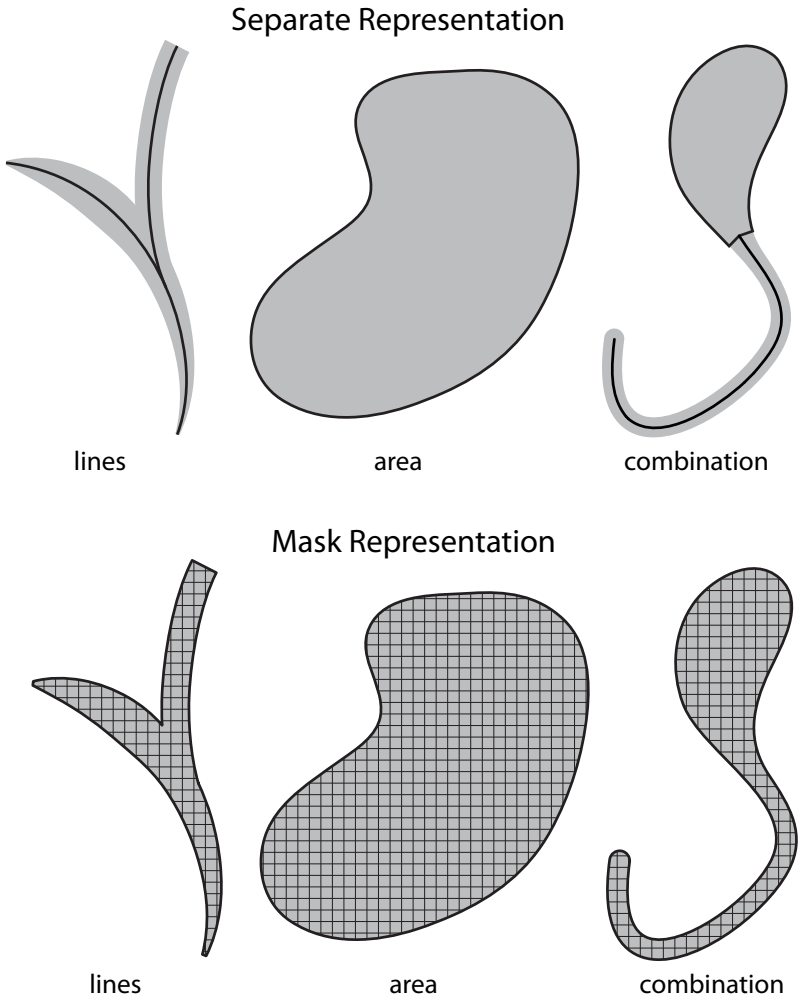

Figure 3.5: *In the **Separate Representation**, lines and areas have distinct vector descriptions. Lines are represented by open paths while areas are defined by their boundaries using closed paths. Drawings however present cases (combination) where this distinction is not appropriate. This has two undesired effects. First, one is forced to chose a hard transition point and this is not necessarily trivial, and second, the result is a compound object which is difficult to edit ensuring a smooth continuous edit across the transition.*
*Instead, with the **Mask Representation**, lines, areas, and combinations all have the same boundary-based representation. Editing however is extremely limited, as the actual appearance is given by the raster image underneath, which cannot be edited by vector means.*

areas, one should intervene at the raster level, ultimately neutralizing the advantage of vector data. Harmony allow the user to drop the raster data in favor of a color fill of the interior of the mask. Indeed, this enables the editing of the drawing at the vector level, but also loses the subtle visual details of the raster data. We also argue that, in the case of lines, editing using the boundary lines is more complex than editing using the centerlines.

Existing vector representations present a trade-off between editing and rendering capabilities. While the separate representation offers more support for editing, the visual complexity of lines and areas does not match the richness of raster images. On the other hand, the mask representation offers excellent visual fidelity to the drawings, but editing is almost impossible.

A summary of the requirements for an ideal digital representation is presented in the next Section.

3.3 Requirements

Now that we discussed existing representations and their limitations with respect to supporting 2D Animation, we want to proceed and list a number of requirements. We believe that, by meeting these requirements, a representation would be suitable for drawing and for 2D Animation, and improve substantially the productivity in professional environments.

In our opinion, an optimal drawing representation should:

- **Interface to both Physical and Digital Drawings.** In order to be usable, a drawing representation should be constructible both from scanned drawing — to be compatible with legacy artwork — as well as from digital drawing tools. This involves vectorization for the former, and input processing for the latter.
- **Embed Raster Data.** Raster images allow great control over the look of the drawing. Depending on the drawing style, subtle texture details can play an important role. Capturing these details purely with synthesis is not trivial. This issue is emphasized when considering a broad spectrum of drawing styles, such as paintings, charcoal or carbon sketches, etc.
- **Support Line Editing.** Line editing includes manual editing tools, such as manipulation of control points, or redrawing of (parts of) the center-

line (Illustrator [Adobe, 2012b]), as well as interpolation of key-framed strokes graphs (Chapter 6).

- **Support Area Editing.** Area editing includes editing of the boundary, by manipulating the control points, or by morphing to different shapes (Illustrator [Adobe, 2012b]), as well as space warping techniques, using mesh embedding, and As-Rigid-As-Possible methods (Chapter 7).
- **Describe Lines and Areas with a Unique Vector Description.** We discussed this issue in Section 3.2.2. The problem lies in representing drawing elements, regardless of their shape, with a unique vector description convenient for editing.

3.4 A "Hybrid" Representation

We now propose a novel hybrid representation that addresses the requirements expressed in the previous Section.

3.4.1 Vector-Splats Hybrid

The first idea of the hybrid representation is the combination of splats and vector elements to gain the advantages of both raster and vector representations. The concept is similar to the Mask Representation we have discussed in the previous Section, where the raster data is part of the final output, but instead of a fixed grid, we use splats mapped to guiding centerlines, allowing them to move according to deformations applied to the vector elements.

The hybrid vector- and splat-based representation builds on three components: the stroke graph, the splats, and the mapping between these two.

Stroke Graph. For the basic representation of stroke centerlines we use a network of standard parametric curves, similar to existing vectorization approaches. We distinguish between two scenarios:

- Single-Layer (Figure 3.6). In this scenario, we consider a network of strokes that live on the same plane. This is often the case for paper drawings that have been scanned and then vectorized, or for semantic elements (objects or characters) of digitally produced drawings.
- Multi-Layers (Figure 3.7). In this scenario, multiple planes of stroke networks are considered. This is often necessary when characters and background elements need to be animated independently.

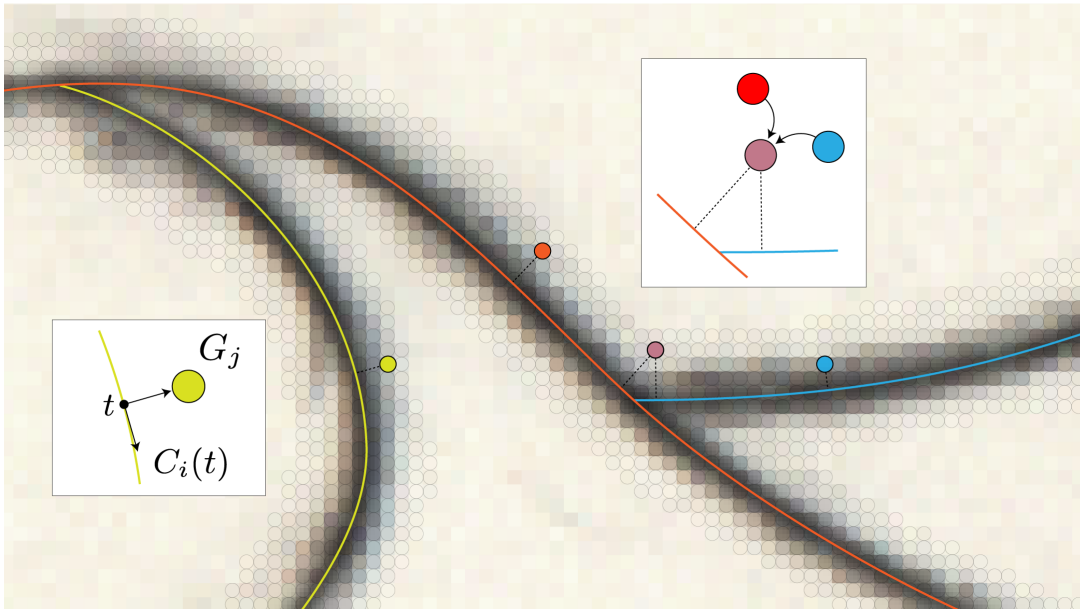

Figure 3.6: *Single-Layer Vector-Splats Hybrid. This image shows the use of a single layer to represent multiple strokes, such as in the case of vectorized drawings. Each splat is associated with the closest centerline. Splats that are close to multiple centerlines are cloned, mapping one splat-clone per centerline.*

Each centerline is defined by a spline curve $C_i(t), t \in [0,1]$. In the experiments of this chapter, we used Catmull-Rom splines, but any other parametric curve type such as B-Splines or subdivision curves is possible [Farin, 2001]. The network of stroke centerlines is the *stroke graph* of the drawing.

Splats. The square pixels of a raster image are transformed into a higher-order representation using radial basis functions, based on EWA splatting. At the center of each pixel p_j we place a corresponding Gaussian G_j (please refer to [Zwicker et al., 2002] for details) which transforms a single pixel value into a more flexible continuous graphical entity that supports arbitrary affine deformations and rendering with higher order interpolation schemes. This is a desirable property when editing the shape of lines (e.g., bending a straight line) or rendering zoomed views of a drawing.

Mapping. For each pixel representative G_j, coordinates $\vec{c}_{ij} := (t, \vec{d})$ relative to its stroke centerline C_i are stored, where t is the curve's arc-length parameter at the point $C_i(t)$ closest to the center of G_j, and $\vec{d}$ is the corresponding signed distance in the normal and tangential direction. The tangential component is

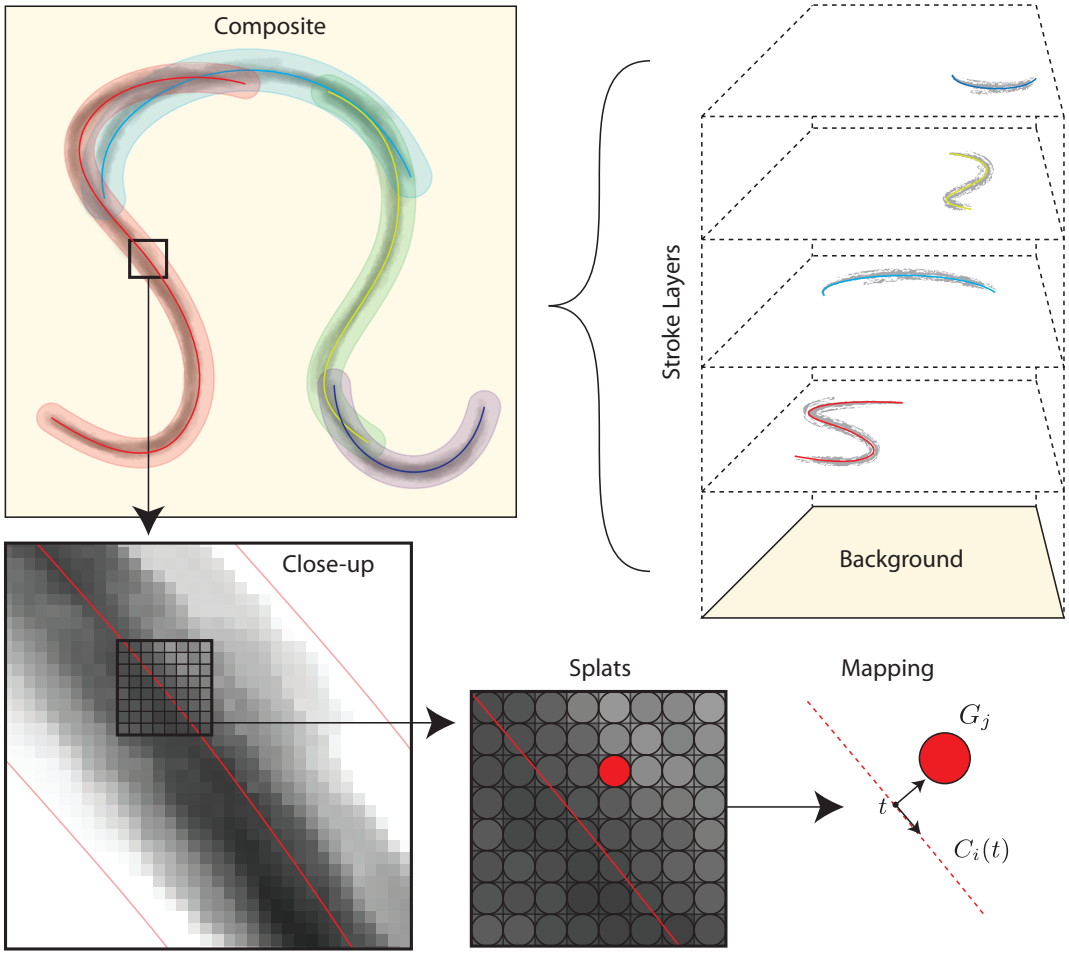

Figure 3.7: *Multi-Layer Vector-Splats Hybrid. This image shows the use of multiple layers to keep strokes independent. For each stroke, a mapping between the splats and the centerline is derived.*

nonzero only for G_j located around stroke endpoints. Note that for pixels p_j at junctions between several strokes C_i, where a unique mapping is not possible, we store coordinates $\vec{c}_{ji}$ for each such stroke.

As a proof of concept, we present the editing of a drawing in a single layer scenario (see Figure 3.8). The drawing on the right has been scanned and vectorized using the method proposed in Chapter 4. Our vectorization method already identified relevant pixels. Alternatively, background extraction techniques [Mcivor, 2000, Piccardi, 2004] can be used. The resulting splats are mapped to the nearest centerlines. Finally, the centerlines of the index finger are modified, and the deformation is propagate to the splats, producing the final render on the right.

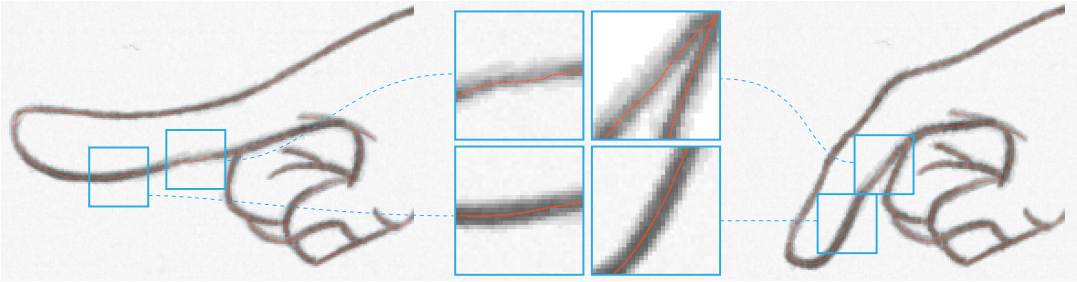

Figure 3.8: *Finger Editing. This result was produced while working on the Vectorization system proposed in Chapter 4. The drawing has been scanned and vectorized with the proposed technique. Splats mapped to the centerlines are used to represent pixels. This image shows the original drawing on the left, and the modified drawing on the right. Closeups of the same regions are shown in the center. Notice how the texture detail is preserved.*

3.4.2 Lines-Areas Hybrid

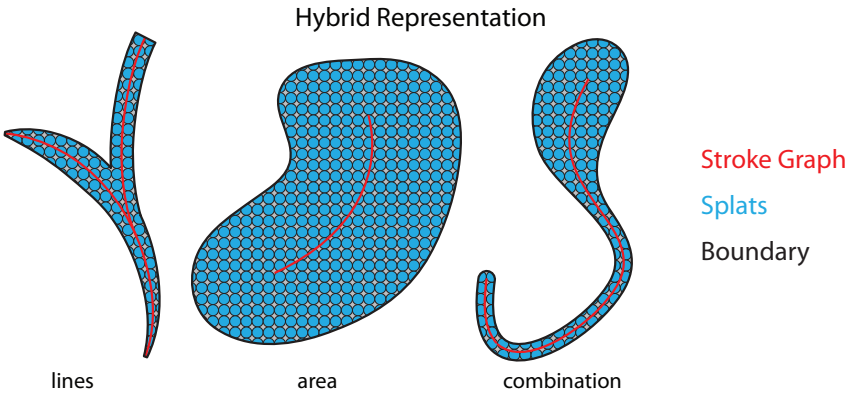

Figure 3.9: *This figure shows the Hybrid representation applied to different shapes. The representation consists of a stroke graph (red), a set of splats (blue), and the a boundary (black).*

The second idea of the hybrid representation is the use of a unified vector description for lines and areas by combining open and closed paths (see Figure 3.9). This Section is speculative, as it gathers ideas that have not been fully tested with working implementations.

Given a shape, the hybrid representation consists of a stroke graph (set of open paths, shown in red), a set of splats representing the raster data (shown

in blue), and a boundary line (close path, shown in black) that marks the transition to the background. A mapping similar to the one between splats and centerlines is derived for the boundary line, where control points are expressed in tangential and normal coordinates from the closest point on the centerline.

Let us consider the following aspects:

- Stylus Input Processing. This case reflects the interface to digital drawing tools.
- Scanned Drawing Processing. This case covers the creation of the hybrid representation in case of scanned drawings.
- Editing Scenarios. Here we list a few editing scenarios and describe the processing flow within the components of the representation.

Stylus Input Processing

Digital drawing solutions have the advantage (over scanning paper drawings) that the creation of the drawing can be fully captured as vector data. The user interacts with the program using either a mouse or a stylus (Figure 3.10a), and the program provides a set of drawing tools. Most sketch and painting programs [Adobe, 2012c, Autodesk, 2012] include a variety of pencil and brushes tools (Figure 3.10b).

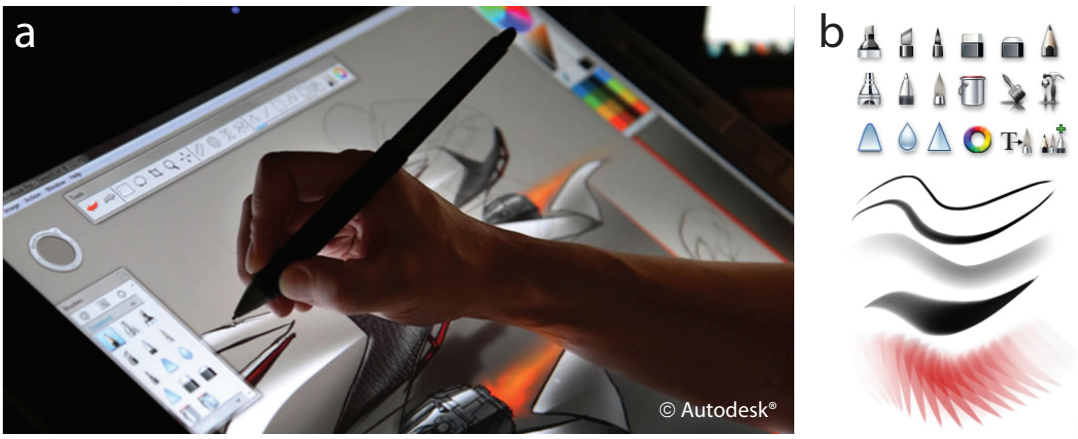

Figure 3.10: *A user interacting with a digital drawing program using a stylus (a). A number of pencils and brushes offered by Autodesk® SketchBook Pro. (Image source [Autodesk, 2012])*

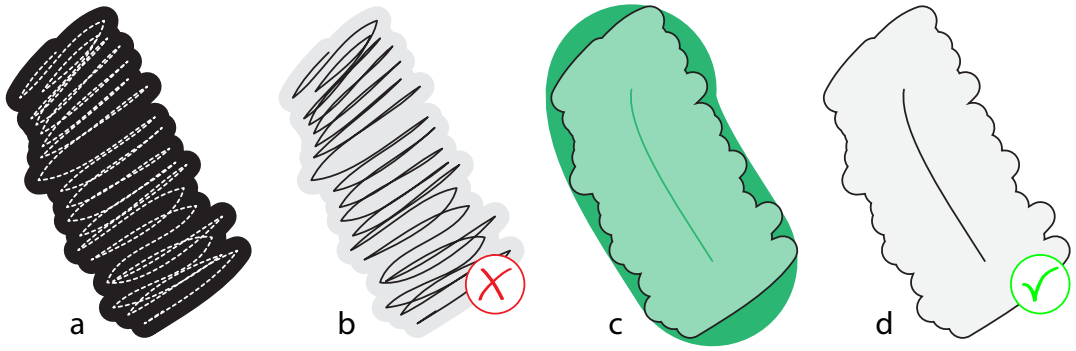

Figure 3.11: *Input processing. When using a pencil or a brush, areas are often the result of long knotty strokes (a). Using the stylus input unchanged will likely lead to a poor representation of the area (b). Skeleton methods, such as the Scale Axis Transform [Giesen et al., 2009], allow to obtain skeletal lines representing a smooth approximation of the area (c). Conceptually, this allows us to generate the desired describing centerline needed for the hybrid representation (d).*

Two particular aspects are important to generate the hybrid representation: the stylus trajectory and the brush properties. The former defines the path of the brush, which in the case of strokes, corresponds to the desired centerline. The latter defines both the raster data and the boundary lines.

There is however an important problem to solve, which conceptually is another instance of the line-area dichotomy, as discussed in Chapter 2. The problem arises when drawing areas using a stylus. As shown in Figure 3.11a, areas are often the result of tangled strokes that zigzag over the region multiple times, until the desired space is colored. The exact stylus trajectory, however, is not relevant, as the same area could be obtained in many different ways. The question is how to process the input to obtain more meaningful data.

The observation is that the generated areas have defined boundaries which appear more descriptive than the stylus trajectory. However, areas can often have a guiding direction, and for consistency with the hybrid representation definition, we are interested in generating a centerline.

The problem of deriving a skeletal lines from boundary lines — the medial axis problem — has been studied since the seventies [Blum, 1967]. One of the main challenges is the sensitivity of the result to small perturbations of the boundary, resulting in undesired skeletal lines. A few experiments are shown in Figure 3.12.

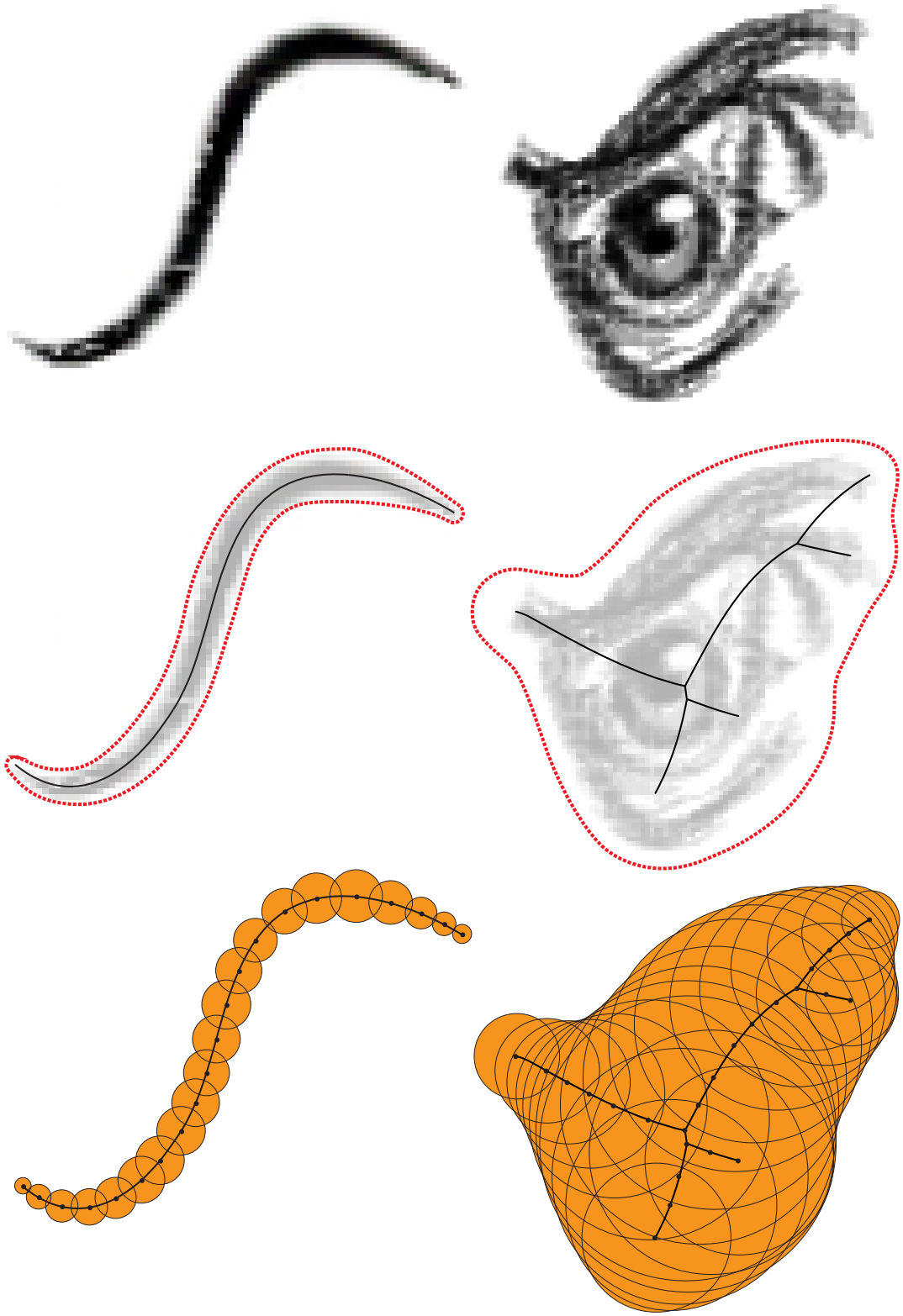

Figure 3.12: *Medial Axis Experiments. This figure shows Medial Axis experiments produced with Mesecina [Miklos, 2011]. The raster data is shown on the top. Boundary lines (red) and reconstructed skeletal lines (black) are shown in the middle. Medial axis disks are shown on the bottom.*

[Giesen et al., 2009] proposes the Scale Axis Transform, as a method to remove undesired skeletal lines by expanding the medial axis disks, ultimately smoothing the boundary. This could be a viable solution, but further experimentation is needed.

Scanned Drawing Processing

An in-depth discussion about vectorization is proposed in Chapter 4. Here we briefly consider the scenario where a drawing is scanned and vectorized to extract both boundary curves as well as centerlines. This can be achieved with a number of techniques, as discussed in Section 4.2. We demostrated in Figure 3.8 a working example where our hybrid representation has been constructed from a scanned drawing. Notice however that the boundary lines have not been extracted. Since the separation between the splats and the background is given, boundary lines could be extracted using edge detection techniques [Senthilkumaran and Rajesh, 2009].

Editing Cases

Let us consider the following editing scenarios:

1. **Line Editing.** In this scenario, we consider the editing of a centerline. This could be achieved by manual deformation of the control points, by a space deformation, by a morphing to a target modified centerline. The processing flow starts with the centerline deformation, followed by the propagation of the deformation to the splats and the boundary lines.
2. **Area Editing.** There are two distinct cases to consider:
   - **Space Deformation.** In this scenario, a particular area is subject to a deformation field. The processing flows starts with the individual deformation of the components (centerlines, splats, boundary lines), and follows with the re-computation of the mapping between splats and centerlines, and boundary lines and centerlines.
   - **Boundary Deformation.** We envision two possible means of editing the boundary. In once case, editing the boundary will redefine the centerline. This can be achieved with variations of medial axis, as discussed previously in this Section. Once the centerline is obtained, the mappings can be recomputed to restore the relationships between the representation components. The second case is the one where one wants to keep the centerline, but modify the

boundary. An instance of this case is when one wants to change the thickness profile of a stroke. The question is how to propagate the deformation of the boundary to the splats.

In this work, we only experimented with Line Editing (See Figure 3.8), and Space Deformation (See Chapter 7). Additional experiments should be conducted for the Boundary Deformation, both in terms of testing variations of the Medial Axis to generate centerlines from the boundary as well as find mappings that allow to alter the splats based on boundary alterations.

3.5 Conclusions

In this Chapter we have discussed the topic of digital representations, with focus on defining a representation suitable for drawings and 2D Animation. We have presented a few ideas we explored during this PhD, including the combination of Splats and vector data as a way to embed raster data and therefore capture the drawing visual appearance faithfully, as well as the concept of using a unified vector description for any type of shape, avoiding the problems that occur when lines and areas are distinct.

The main goal of this Chapter was to make the reader aware of the existing representation problems, and stimulate to think of possible solutions, possibly taking the proposed ideas as a base for further research.

CHAPTER 4

Line-Drawing Vectorization

Vectorization provides a link between raster scans of pencil-and-paper drawings and modern digital processing algorithms that require accurate vector representations. Even when input drawings are comprised of clean, crisp lines, inherent ambiguities near junctions make vectorization deceptively difficult. As a consequence, current vectorization approaches often fail to faithfully capture the junctions of drawn strokes.

In this Chapter, we propose a vectorization algorithm specialized for clean line drawings that analyzes the drawing's topology in order to overcome junction ambiguities. A gradient-based pixel clustering technique facilitates topology computation. This topological information is exploited during centerline extraction by a new "reverse drawing" procedure that reconstructs all possible drawing states prior to the creation of a junction and then selects the most likely stroke configuration.

For cases where the automatic result does not match the artist's interpretation, our drawing analysis enables an efficient user interface to easily adjust the junction location. We demonstrate results on professional examples and evaluate the vectorization quality with quantitative comparison to hand-traced centerlines as well as the results of leading commercial algorithms.

4.1 Introduction

Raster and vector representations form the foundation upon which nearly all two-dimensional graphics is built. Raster images can represent extremely rich detail but do not encode the kind of semantic information that promotes editing. Vector images, on the other hand, abstract image content as mathematical primitives such as lines and arcs that facilitate editing but can limit detail. While converting from a vector to a raster representation is a straightforward sampling operation, the complementary procedure of vectorization is significantly more difficult since it involves inferring high-level abstractions from low-level pixel content.

Hand-drawn 2D animation represents one particularly important application area of vectorization. The expressiveness, efficiency, and tactile control afforded by real pencil and paper are yet to be matched by digital drawing tools, and therefore 2D animations are often still hand-drawn on paper and then scanned. The content in the resulting raster images cannot be easily edited or used with higher level algorithms that require a stroke-based vector representation, such as computer-assisted inbetweening [Whited et al., 2010]. A similar problem exists even when digital drawing tools are used, since artists often build up lines with many short strokes, leading to an unorganized collection of tiny unconnected segments that are not amenable to further high-level processing. In this domain, an automated vectorization approach is essential as a feature animation can contain hundreds of thousands of individual drawings.

Loose and sketchy drawings contain a great deal of ambiguity which makes automatic vectorization extremely difficult. At the other end of the spectrum, "clean" drawings are defined by crisp, distinct lines and thus present the ideal input for vectorization. However, even in this case, ambiguities at stroke junctions make vectorization deceptively difficult. Due to its fixed structure and limited resolution, the regular grid of a raster image is ill-suited to represent regions where many strokes come together, overlap, cross, or join. As a consequence, current vectorization algorithms often fail at accurately representing junctions.

The poor quality of junction reconstruction is due in part to the local nature of existing vectorization algorithms: extracted lines result solely from the information contained in a fixed-size pixel neighborhood. In practice, a larger scope is often necessary to understand the inherent structure of the strokes defining a junction. In this sense, one can consider these stroke relationships as *non-local*. Motivated by this observation, we propose a non-local vector-

ization algorithm that employs the analysis of a drawing's topology in order to extract high-quality centerlines and junctions from clean drawings.

The first step of our approach analyzes the input image to derive the stroke topology. Here a gradient-based pixel clustering technique is employed that facilitates the extraction of the correct topology in under-sampled regions of the drawing. This topological information is exploited during centerline extraction by a "reverse drawing" procedure that reconstructs all possible drawing states prior to the creation of a junction and selects the most likely stroke configuration. If the automatic result does not match the artist's interpretation, our drawing analysis enables an efficient user interface to easily adjust the junction location. We demonstrate results on professional examples and evaluate the vectorization quality with quantitative comparison to centerlines hand-traced by an expert artist as well as with side-by-side comparisons to output from leading commercial methods.

Our system fits naturally into current pipelines to enable vector processing of scanned drawings. We make the technical contributions of the gradient-based pixel clustering procedure for accurate topological analysis as well as the reverse drawing procedure for producing the most plausible junction configurations. For either hand- or digitally-drawn input, our work provides a bridging technology that converts drawings into a format designed for further editing, automatic inbetweening, or other advanced vector-based processing algorithms.

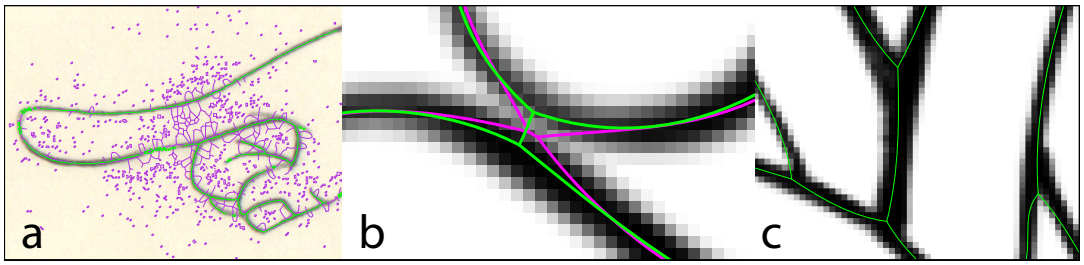

Figure 4.1: *Vectorization Challenges. Noise (a) and spatially adjacent strokes (b) require fine tuning of threshold parameters in existing approaches. Even for clean, high resolution input images, the superposition of strokes near junctions and sharp corners results in inaccurate centerline placement (c). (Results of Adobe Live Trace for different threshold settings shown in purple and green.)*

4.2 Related Work

Existing vectorization methods can be roughly classified into two groups based on whether they are designed to process image or line data. Techniques for the vectorization of general images make the assumption that the image content can be represented by a collection of boundary curves, together with smooth interpolating functions between the curves. In one family of approaches, the image is first segmented into regions by, for example, triangulation or using quad-dominant gradient meshes, and then the region interiors are filled with smooth gradients [Lecot and Levy, 2006, Sun et al., 2007b, Xia et al., 2009]. Alternatively, using diffusion curves, the smooth interior can be computed by solving a Poisson equation with the curves as boundary constraints [Orzan et al., 2008b]. Zhang and colleagues[Zhang et al., 2009] present an approach specifically tailored for temporally coherent cartoon animations, while Sýkora and coworkers[Sykora et al., 2005] vectorize regions of cartoon frames for the purpose of compression. Both address final cartoon frames with all colored foreground and background layers.

A related problem is the extraction of curve skeletons from 2D shape boundaries using variational methods [Cornea et al., 2007]. Additional hybrid methods seek to find centerlines in images. In the field of medical imaging, blood vessel extraction requires identifying and reconstructing the tubular structures from images and scans. A range of techniques has been developed [Kirbas and Quek, 2000], from pattern recognition, to model-based and tracking-based methods. In this domain, Whited and colleagues[Whited et al., 2009] present a semi-automatic centerline extraction from networks of strokes that also works in more general images (e.g. river networks from satellite imagery). While effective for vectorizing general image content, none of the above methods are designed to work with line drawings and do not sufficiently address the accurate extraction of centerlines and junctions.

The second group of methods is primarily concerned with vectorization of line drawings such as technical layouts. Prominent approaches are based on tracing [Freeman, 1974], thinning [Lam et al., 1992], or methods utilizing contours or projections such as the Hough transform [Liu and Dori, 1998]. Due to the focus on technical images, many of these methods are restricted to fitting straight line segments to input drawings [Janssen and Vossepoel, 1997]. Exceptions include the method by Chang and Yan[Chang and Yan, 1998], which fits Bezier curves, and the method by Zou and Yan[Zou and Yan, 2001], which focuses on issues such as jaggy line boundaries and junction points. Hilaire and

Tombre[Hilaire and Tombre, 2006] also address robustness and describe fitting of higher order primitives such as arcs in addition to line segments. Their method mainly addresses issues found in binary technical drawings and cannot be easily generalized to freehand sketches. Bartolo et al.[Bartolo et al., 2007] describe an approach based on Gabor and Kalman filtering in order to convert rough scribbles into a vectorized representation. When boundaries are well defined, skeleton methods [Lakshmi and Punithavalli, 2009] produce good vector centerlines that could be used to represent line drawings. However distorted skeleton centerlines appear at junctions.

Finally, commercial tools for vectorization of line art include Toon Boom Harmony, Adobe Live Trace, CorelDRAW, VectorEye, VectorMagic, and AutoTrace.

In many situations, existing methods and techniques provide high-quality results. However, strokes drawn very close together and junctions areas are usually poorly reconstructed. In most cases, this limitation arises from an algorithm that employs local operators without considering the overall structure of the drawing. Such information is needed to accurately reconstruct stroke centerlines and junction points, and to perform more sophisticated editing operations such as morphing and inbetweening. In our work we specifically address these open challenges.

4.3 Overview of Approach

The goal of vectorization is to extract stroke centerlines and a network of vector curves and junctions from an input raster image of a line drawing.

Current vectorization techniques face two major challenges. The first problem, illustrated in Figure 4.1a,b, is insufficient *local* discrimination of individual strokes due to noise and spatial proximity of strokes. The second problem, which is of a *global* nature, is the difficulty of obtaining accurate estimation of centerlines at junctions. It is a global problem because it requires information about the drawing topology and stroke configuration (see Figure 4.1b,c and also Figure 4.5). Both problems compromise centerline estimates and result in bad vectorization quality using existing techniques.

The algorithm we propose for vectorization of line drawings addresses these problems with a novel bottom-up analysis, which translates into three successive processing phases; each step of the algorithm increases the level of

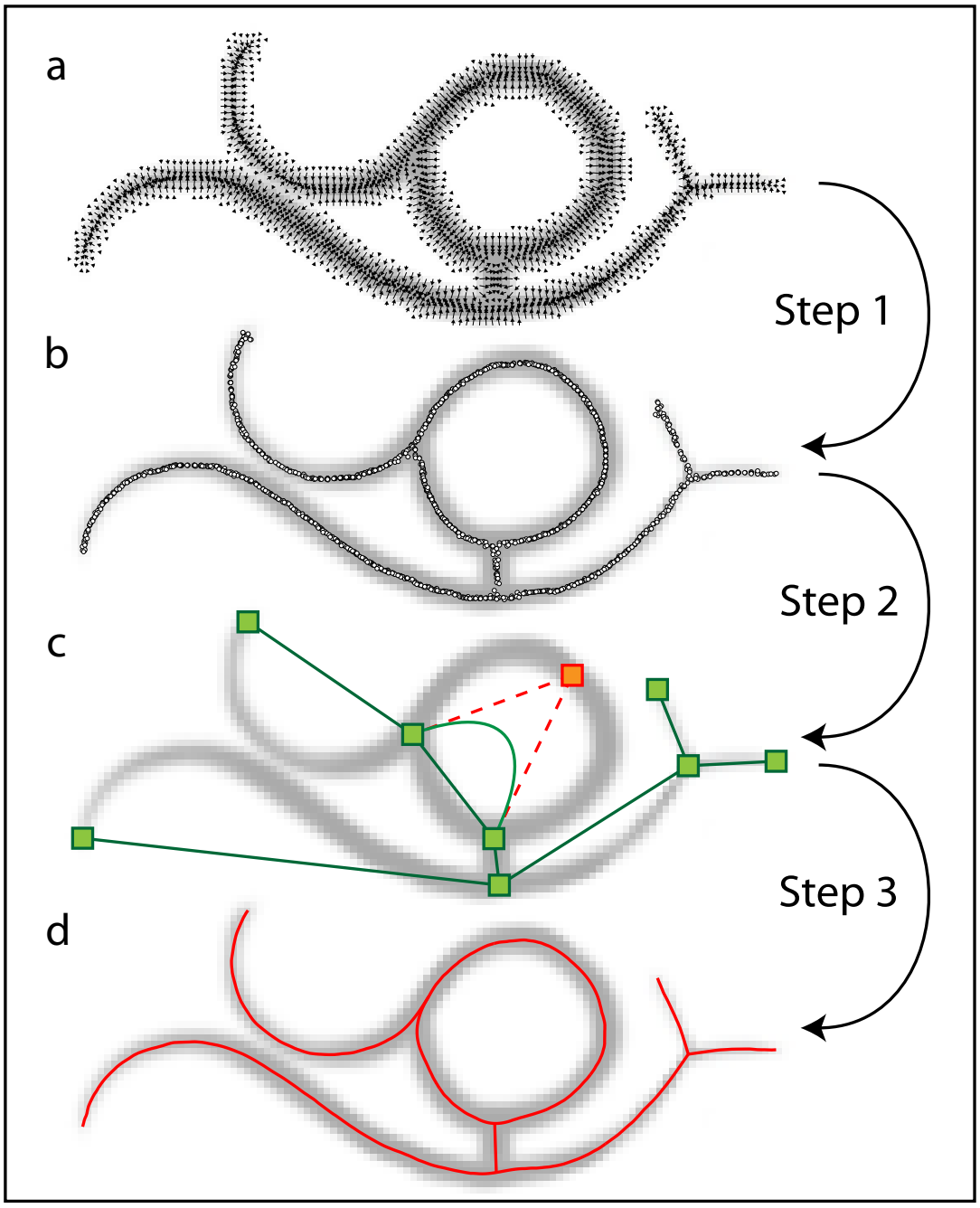

Figure 4.2: *Method Overview. Step 1: First, our algorithm disambiguates the input pixels using a gradient-based clustering process. Step 2: From the clusters, the topological skeleton of the drawing is extracted. Step 3: By utilizing the topological information, our reverse drawing procedure extracts accurate centerline estimates and junction positions.*

abstraction of the representation, until accurate centerlines can be reconstructed.

Step 1: Stroke Disambiguation by Clustering: Our first observation is that, in line drawings, the *color gradient* at each input pixel often provides a good local estimate of the center of a nearby stroke centerline (see Figure 4.2a). We show that a clustering approach, which moves pixels along the gradient field based on the notion of gradient "confidence", enables effective local disambiguation of strokes (Figure 4.2b) and compares favorably to existing skeletonization techniques.

Step 2: Topology Extraction: After clustering, the pixels are connected to form a *cluster graph* (Figure 4.2b). The second phase of our algorithm then analyzes this cluster graph to compute the underlying topological skeleton of the drawing (Figure 4.2c). This skeleton represents the individual stroke segments, stroke endpoints, and junctions between stroke segments. The proposed procedure is based on the computation of minimum spanning trees as an efficient solution for global topology extraction even on large drawings with complex cluster graphs.

Step 3: Centerline Reconstruction and Reverse Drawing: Using the topology of the cluster graph, the centerlines of the drawing can be extracted (Figure 4.2d). Particular care is taken in inherently ambiguous regions like junctions. The novelty of this approach is a topology-driven identification of such ambiguous regions, followed by an exploration of all the possible stroke configurations in a process we refer to as *reverse drawing*. We first score pairs of incident stroke segments at a junction and then select the most likely configurations, based on the assumption that smoothly joining stroke segments are generally more likely to belong to a continuously drawn stroke than segments joining at sharp angles. Notice that this step can be applied independently of the previous steps to improve results obtained for instance with robust skeletonization or thinning algorithms.

The algorithmic details of these three phases, from local stroke disambiguation to topology-aware reconstruction of centerline configurations, are described in Section 4.4. Our results and evaluation in Section 4.5 demonstrate that our approach resolves the limitations of existing techniques and results in high-quality vectorization.

4.4 Algorithm

4.4.1 Clustering for Stroke Disambiguation

Pixels $\vec{p}_i$ of a raster input image can be roughly classified into two categories, depending on their respective image gradient ∇_i: small gradients do not carry sufficient information about the stroke center, while large gradients provide a more confident guess about the centerline location. Accordingly, we classify each pixel $\vec{p}_i$ as either stationary: $\mathcal{S} = \{\vec{p}_i | \|\nabla_i\| < \epsilon\}$ or moving: $\mathcal{M} = \{\vec{p}_i | \|\nabla_i\| \geq \epsilon\}$ by thresholding the gradient norm. The norm threshold value ϵ should be set to be above the gradient levels of the image noise. All results presented in this paper have been produced with ϵ equal to 10% of the maximal gradient length.

The basic idea of our stroke clustering is that confident pixels $\vec{p}_i \in \mathcal{M}$ move towards the centerline by following the direction ∇_i[1]. Although local noise may influence the trajectory of individual pixels, as long as the gradient noise level is below ϵ, the pixels converge and cluster towards the centerline (see Figure 4.3).

For all pixels $\vec{p}_i \in \mathcal{M}$, the motion vector is set to $m_i = \delta t \nabla_i$, where δt is a constant speed factor, in our implementation equal to 10% of the width of a pixel. This has two consequences: first, the pixels move in compact bands, and second, centerlines are located where the two opposing bands meet. The stopping condition for each moving pixel is then naturally given by the motion coherence in its local neighborhood; for each pixel $\vec{p}_i \in \mathcal{M}$ the nearest neighbors $\mathcal{N}_i = \{\vec{p}_j | \|\vec{p}_j - \vec{p}_i\| \leq 1\}$ are collected. By looking at the sign of the dot product of the gradients $\nabla_i \cdot \nabla_j$, neighboring pixels $\vec{p}_j \in N_i$ are classified either as belonging to the same band (positive dot product), or the opposing band (negative dot product). The stopping condition is then defined as having one or more pixels of the opposing band that traveled past the location of pixel $\vec{p}_i$, which can be expressed as $(\vec{p}_j - \vec{p}_i) \cdot \nabla_i < 0$.

The clustering process terminates when the number of moving pixels drops below 1% of the initial set $\mathcal{M}$. Outliers, such as remaining background and isolated pixels, can be eliminated by removing those pixels that remained stationary through the whole clustering process or that have less then 2 neighbors within a 1-pixel radius. This clustering procedure results in a *contraction* of the input pixels around the approximate location of the stroke centerline.

Notice that at this stage it is possible to get an estimate of the stroke thickness by considering the distance that boundary pixels traveled. In our implemen-

[1] Gradient directions are kept constant throughout the clustering.

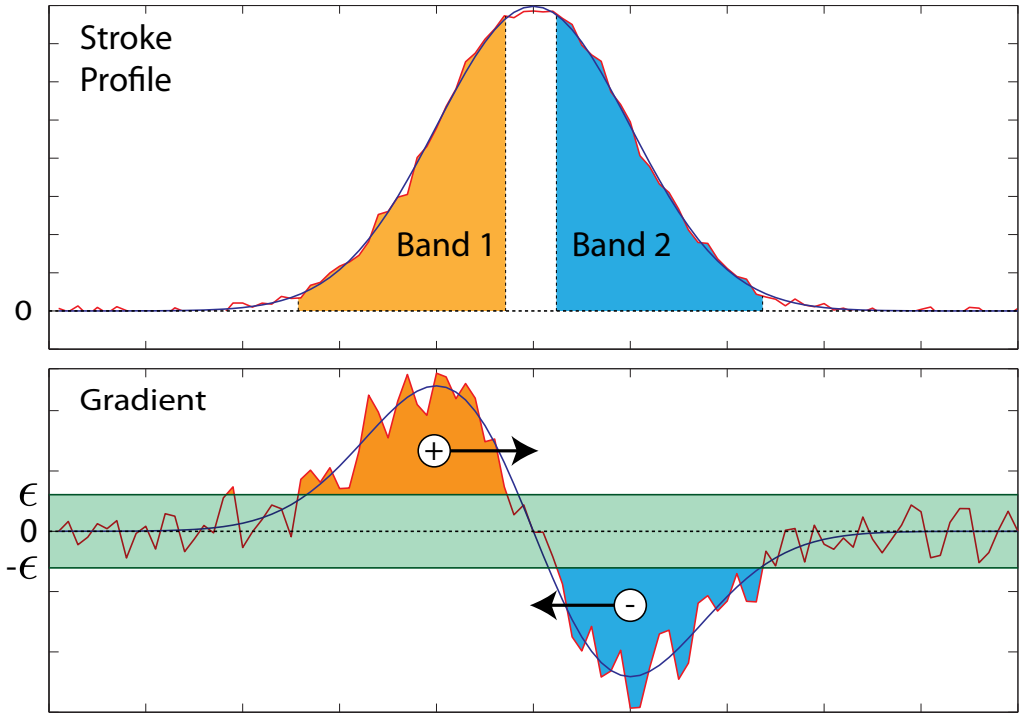

Figure 4.3: *The gradient threshold ϵ defines two bands of pixels with opposite gradient directions.*

tation we do not explicitly mark pixels as belonging to the boundary, but rather we store the traveled distance of each pixel as the approximate local stroke radius r_i and take a conservative estimate by setting it to be the maximum r_j of all $\vec{p}_j \in \mathcal{N}_i$. This estimate of the local stroke thickness will be utilized in the subsequent steps of our algorithm.

4.4.2 Topology Extraction

We are interested in extracting the topology of the drawing. After convergence, the cluster is a point set that densely samples the proximity of the drawing stroke centerlines. Instead of applying techniques from geometry reconstruction, we treat this point set as a graph, and rely on well-known efficient graph algorithms to extract a skeleton that explicitly contains the topology of the drawing.

The procedure is illustrated in Figure 4.4. A graph structure (the *cluster graph*) is constructed by connecting each clustered pixel $\vec{p}_i$ to each neighbor $\vec{p}_j$ within the local stroke thickness (see Figure 4.4b). A weighted edge e_{ij}

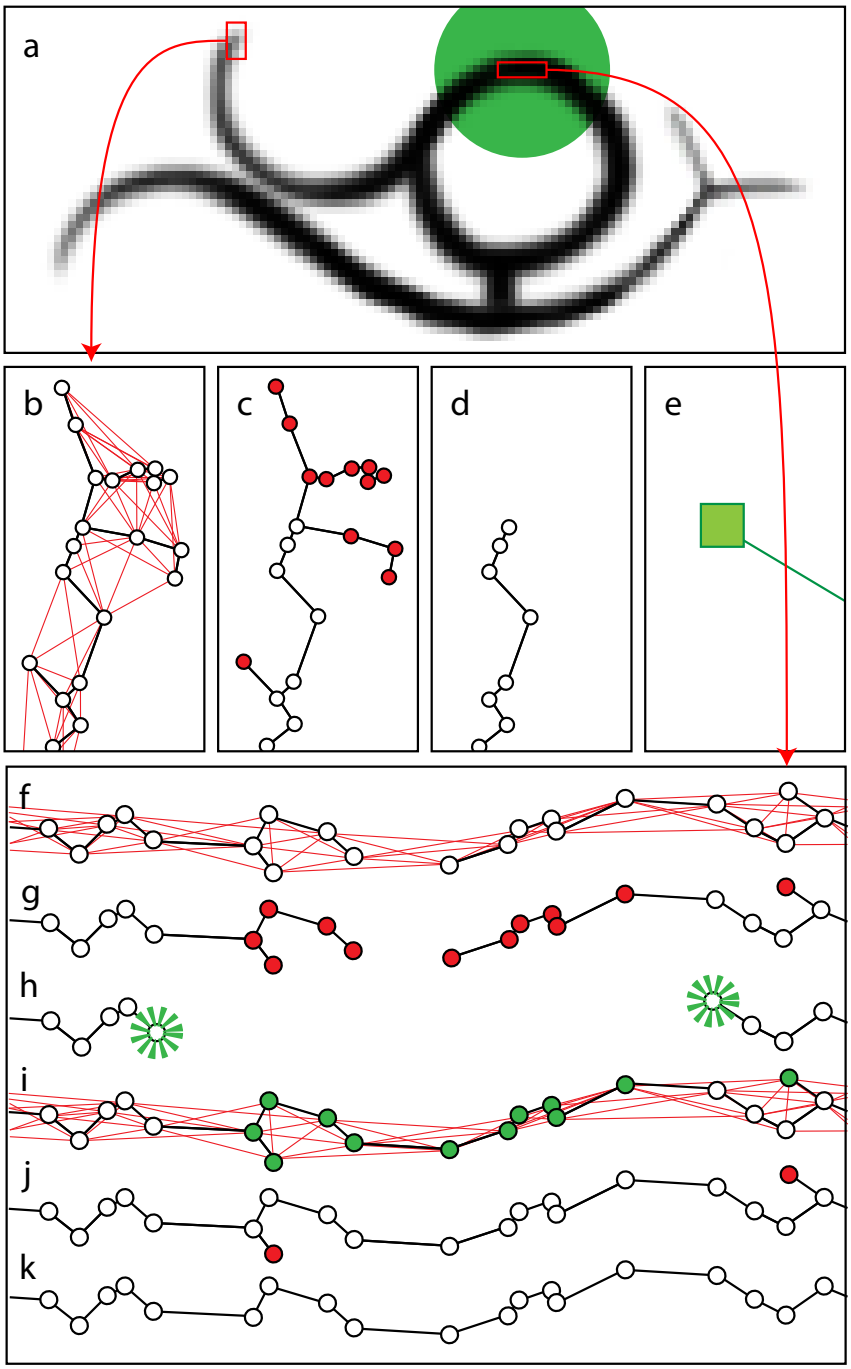

Figure 4.4: *Topology Extraction and Loop fixing (a).* ***Topology Extraction*** *(b-e). An minimum Spanning Tree (MST) is computed (b). Branches of length smaller than the stroke thickness are removed (c), resulting in a skeletonized version of the cluster (d) from which the final topology of the skeleton is extracted (e).* ***Loop Fixing*** *(f-k). For drawings containing loops (a), the MST computation breaks the loop in at least one location (f). The leaf pruning (g) widens the gap. To restore the link, for each of the remaining leaves (h) a local MST is computed (local scope: the green circle in a). (i) The local and global MSTs are merged, and remaining leaves pruned (j), ultimately restoring the loop (k).*

is added for each pair $(\vec{p}_i, \vec{p}_j)$. The weight $\omega(e_{ij})$ of an edge is simply the Euclidean distance $\omega(e_{ij}) = D(\vec{p}_i, \vec{p}_j)$.

The topological skeleton (endpoints, junctions, connectivity) of the drawing is then computed by topology-preserving coarsening of the cluster graph (Figure 4.4b-e). First, a minimum spanning tree (MST) of the graph is computed [Kleinberg and Tardos, 2005]. Due to the dense pixel clustering, the MST is characterized by a number of main branches with many very short branches ("twigs") which contribute to the stroke width/detail, but not ultimately to the topological structure we are seeking. In order to isolate the main branches, the leaves of the MST are iteratively pruned (Figure 4.4c). To avoid pruning the entire graph, we keep track of the length of the branches being removed, and terminate the iteration if deleting an additional node will make the total length of the removed branch greater than the local stroke thickness.

By definition, any loop in the drawing will be cut by the global MST. Figure 4.4f-k illustrates a procedure to reliably detect and close these cuts through the construction of a local MST around each leaf node. Consider the cut produced by the global MST (4.4f). The leaf pruning will erode both sides, widening the cut up to approximately $2r_i$ (4.4g). For each leaf node (4.4h), we compute local MST (4.4i) and then apply leaf pruning (4.4j). Loop connectivity is restored by taking the union of the global MST and the local ones (4.4k). Notice that this procedure will not affect actual end points, as the local MSTs followed by the leaf pruning will produce the same initial leaf nodes as the pruned global MST.

We now mention a few implementation details. First, an MST cut generates two leaf nodes, but it is sufficient to apply the above procedure to one of the two. Checking if a leaf is still a leaf after each iteration can save computation time. Second, prior the computation of the local MST, we zero the weights of the edges that are in the global MST. This will force the local MST to pick the same edges and only expand within the gap, avoiding the introduction of triangular structures or undesired loops. Third, in order to account for variation in our stroke thickness estimate, we set a conservative range for the local MST of $4r_i$ (green circle in Figure 4.4a).

The final topological skeleton of the drawing can then be obtained by collapsing all nodes of valence 2 in the graph. Nodes of valence 1 then correspond to stroke endpoints, nodes of valence ≥ 3 to stroke junctions, and the graph edges represent the topological stroke segments in the drawing (see Figure 4.4e).

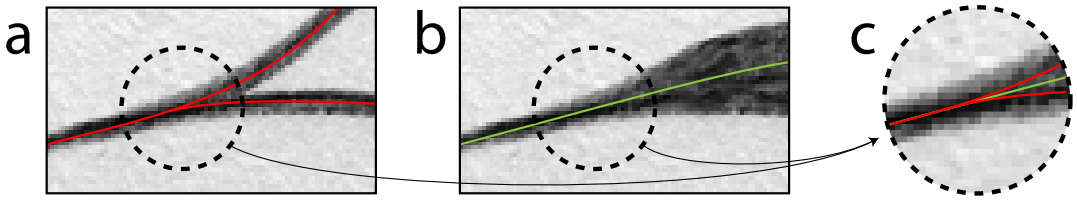

Figure 4.5: *Local Ambiguity. A junction (a) and a stroke with varying thickness (b) cannot be distinguished by considering the local appearance only (c).*

4.4.3 Centerline Extraction and Reverse Drawing

The main challenge for reconstructing accurate centerline estimates is ambiguities which cannot be resolved by purely local methods and hence lead to reconstruction artifacts in existing approaches. Figure 4.5 illustrates such an ambiguity where two strokes converge and cannot be distinguished from a stroke with varying thickness by considering only a local window. Moreover, even when it is clear that multiple strokes meet at a junction, one has to choose among a number of possible configurations (see Figure 4.6).

In order to address these challenges, our algorithm performs two steps. First, a set of base centerlines is traced, connecting all end points and junctions according to the drawing topology. We call the processing up to this point *base method*, as it produces centerlines which in nature are similar to the results obtained with prior art (see Figure 4.18). Second, our reverse drawing procedure (see Figure 4.7) utilizes the drawing topology to identify ambiguous regions (e.g., junctions and sharp corners) and then corrects the centerline estimates by choosing the most likely centerline configuration among all possible ones.

Base centerlines

Base centerlines are constructed by computing the source to destination shortest path on the full cluster graph. As source and destination points we pick the junctions and endpoints defined by the drawing topology. Hence, each topology edge generates a base centerline. The stroke thickness is derived locally from the selected path nodes.

Two special cases have to be considered: an edge connecting a junction to itself (loop) and two junctions connected by more than one edge. For both special cases, in our implementation we split the edges adding dummy valence-2

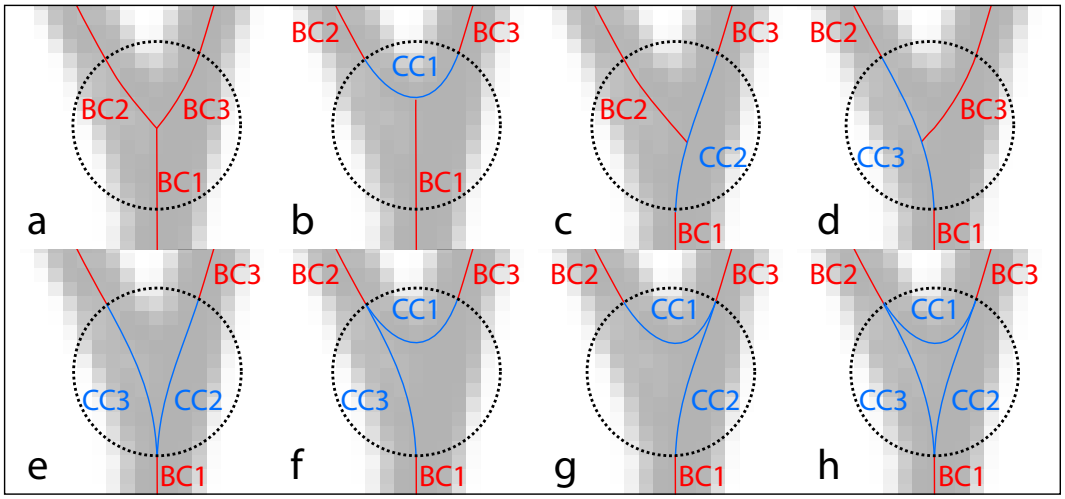

Figure 4.6: *Junction Configurations. This image shows all possible configurations combining the base centerlines (BCs) and continuous centerlines (CCs) inside the ambiguous region (black dotted circles). Case 'h' is included for completeness, but occurs rarely in practice.*

junctions (see red node in Figure 4.2c), forcing the shortest path to take the individual necessary routes.

The extracted shortest paths are then smoothed by applying a data-driven smoothing operator which moves the path along the local curve normal towards the center of mass of the clustered stroke pixels. For each point $\vec{p}_i$ of a centerline path, a Gaussian weighting function is used to assign weights to the cluster pixels $\vec{p}_j$ in the neighborhood. The refined position $\vec{p}_i$ is then given as

$$\vec{p}_i \leftarrow \vec{p}_i + ((\vec{c}_i - \vec{p}_i) \cdot \vec{n}_i)^T \vec{n}_i, \text{ with} \tag{4.1}$$

$$\vec{c}_i = \frac{\sum_j w_j \vec{p}_j}{\sum_j w_j}, \text{ and } w_j = e^{-\frac{D(\vec{p}_i, \vec{p}_j)}{2\sigma^2}}, \tag{4.2}$$

where $\vec{n}_i$ is the normalized approximation of the local curve normal and $\sigma = r_i$ to adapt the weighting scheme to the local stroke thickness (Figure 4.8). We compute approximate vertex normals by averaging the normals of the adjacent line segments, and then interpolate these normals linearly over the segments.

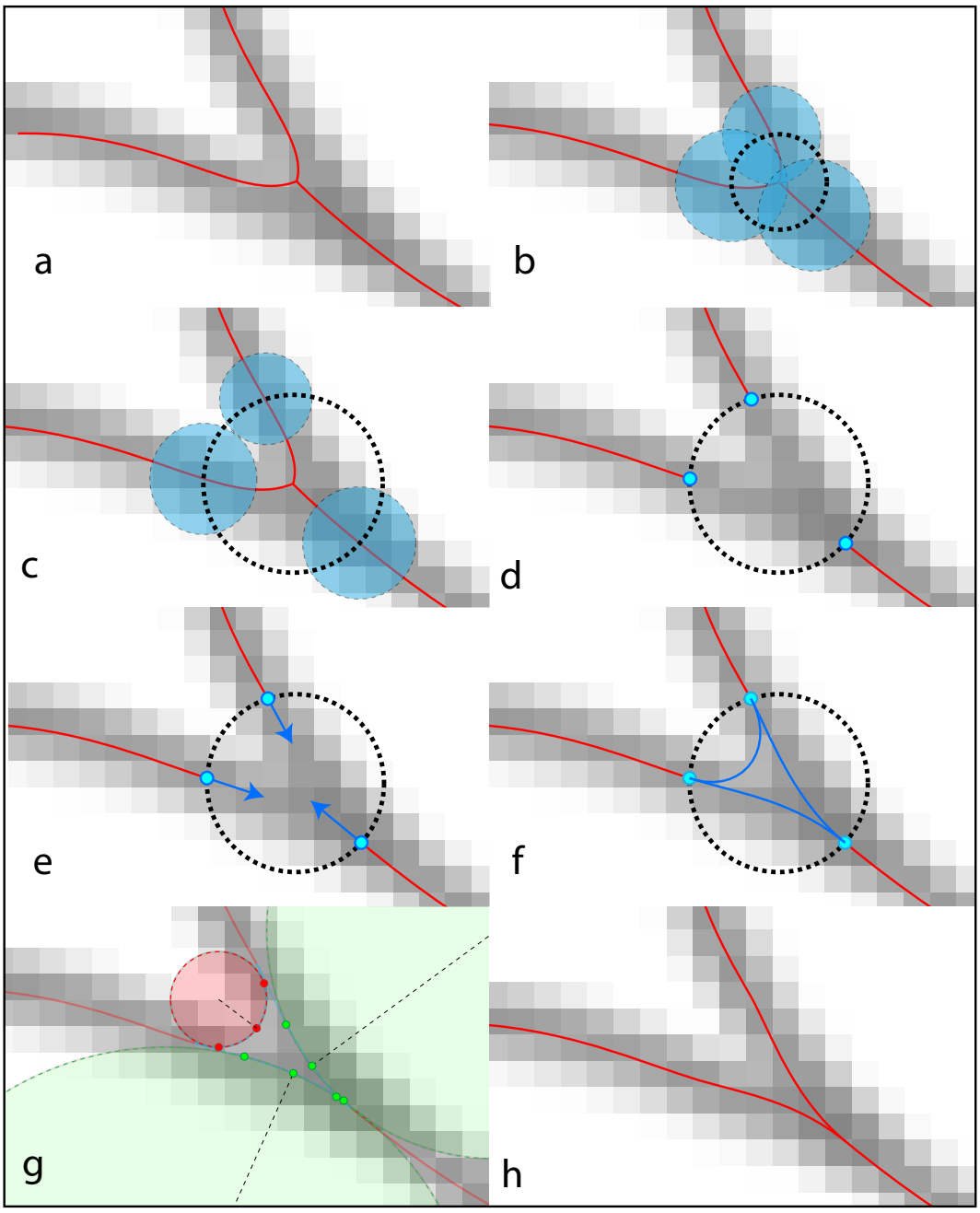

Figure 4.7: *Reverse Drawing. For correct centerline estimation in the proximity of overlapping strokes (e.g., at junctions) (a), our algorithm first identifies the ambiguous region (b,c) and removes the corresponding centerline estimates (d). From the intersections of the ambiguous region with the base centerlines (e), continuous candidate centerlines (CCs) are computed (e, f). Then the stroke-curvatures of the CCs are evaluated (g), and the final centerline configuration is selected (h).*

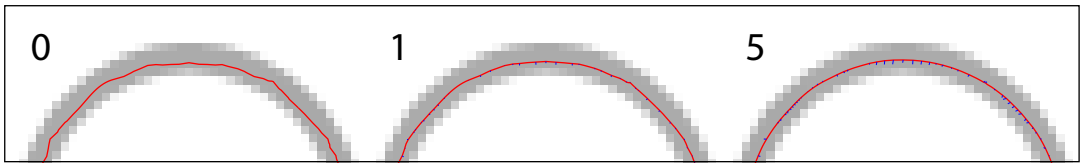

Figure 4.8: *Smoothing. This image shows the result of the smoothing of centerline paths for 0, 1, and 5 iterations. The movement of points is marked in blue.*

Reverse drawing

An overview of the reverse drawing procedure is illustrated in Figure 4.7. The first step consists of identifying ambiguous regions where strokes overlap.

Ambiguous region estimation: For each junction, we iteratively grow a circle AR at the junction position until the strokes no longer overlap. The process is summarized by the steps:

1. Create a circle AR centered at the junction.
2. Intersect AR with each adjacent base centerline BC_j.
3. At each intersection, generate a circle S_j of radius equal to the local stroke thickness (Figure 4.7b, blue circles).
4. If any pair of S_j intersects, increase the radius of AR and repeat from 2. Otherwise stop. (Figure 4.7c).

Continuous centerline construction: Given the ambiguous region at a junction, the base centerlines inside this region are removed and replaced by all possible configurations of continuous centerline candidates (CCs) at that junction (Figure 4.6b-h). Given the position and curve tangent of the intersections between the ambiguous region and a pair of base centerlines, a CC is computed as a cubic Hermite spline (Figure 4.7e,f). The normalized curve tangents are scaled to one third of the distance between the two points.

Stroke-Curvature: When considering the curvature of strokes, we observe that different stroke thicknesses (Figure 4.9a,b) result in different perceived curvatures even when the centerline is the same. As shown in Figure 4.9c, we define the *stroke-curvature* α by sampling the stroke centerline at three points with distance r (the local stroke thickness) and fitting a circle, computing $\alpha = 2 \arcsin\left(\frac{r}{2c}\right)$.

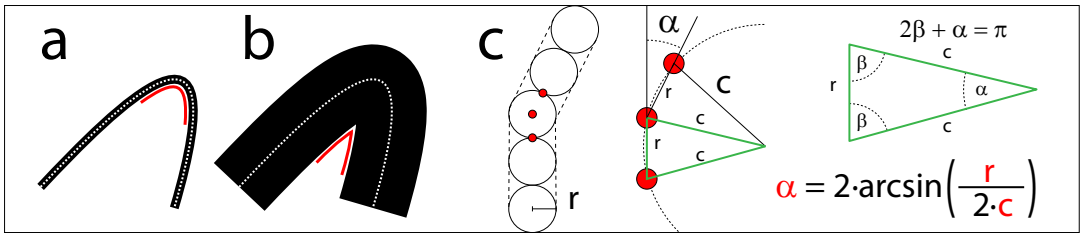

Figure 4.9: *Stroke-Curvature. (a) and (b) illustrate the apparent difference in visual smoothness of strokes with different thickness but the same centerlines. (c) shows the geometric reconstruction of the stroke-curvature α from the local stroke radius r and curvature radius c.*

To compute the stroke-curvature α_i of a continuous centerline CC_i, we sample CC_i uniformly with the sample distance r according to the local stroke thickness, and then set α_i as the maximum stroke-curvature over the whole curve (see Figure 4.7g).

Centerline selection: Our goal is to connect the base centerlines (*BCs*) around an ambiguous region by either picking continuous centerlines (*CCs*) only (e.g., Figure 4.6f-h), or a combination of *BCs* and *CCs* (e.g., Figure 4.6b-e).

Figure 4.10: *Examples of valence-4 junctions.*

In application contexts where fixed alphabets or specific drawing patterns are defined, junction classes (T, Y, X, etc) can be associated with predefined centerline configurations. However, in line drawings, the number of such classes is virtually infinite. Figure 4.10 shows just a few of the many valence-4 junctions that can be found in line drawings, each one with its own nuances. This makes it impractical to build a comprehensive classification.

Instead, we opt for a fixed selection scheme that can operate on any kind of junction (see Algorithm 1). Figure 4.11 illustrates the major steps for a valence-4 junction.

This selection scheme favors straight centerlines over curved ones. The stroke-curvature threshold t discriminates acceptable continuous centerlines from undesirable sharp turning connections, which are usually associated

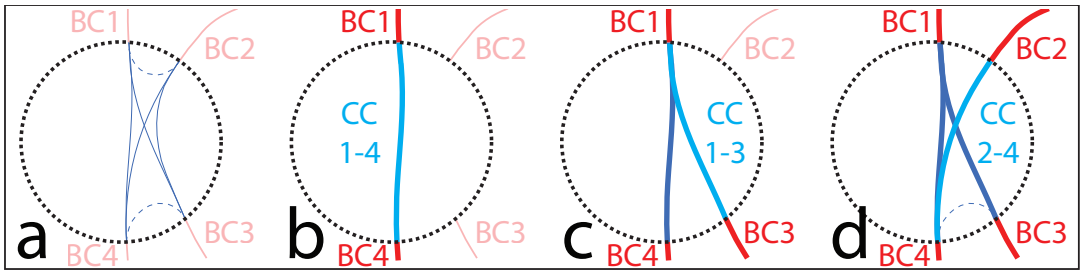

Figure 4.11: *Centerline Selection. CCs are generated (a), and for each, the stroke-curvature is computed. Any CC (dashed) with curvature exceeding the threshold is excluded. Sequentially (b,c,d), the CCs with the smallest curvature are selected until all BCs are connected (d).*

Algorithm 1: *Centerline Selection Scheme.*

Input:A junction with *BCs* **and** *CCs* **OutputA set of accepted** *BCs* **and** *CCs*

for all CC_i **do** Compute α_i
  if $\alpha_i > t$ **then** reject CC_i as *sharp turn*
  end if
end for Put the remaining *CCs* in a list Sort list by ascending α_i
while the list is not empty **do** Extract the straightest CC_i from the list and accept it
  if all *BCs* are connected **then** Terminate
  end if
end while
for all disconnected BC_i **do** Extend BC_i linearly until it crosses any accepted centerline (either BC_i or CC_i) Accept BC_i
end for

with cases where base centerlines stop at the junction (Figure 4.6a-d). In our implementation t is equal to 50°, an optimal value according to our empirical validation (see Section 4.5 and Figure 4.13).

Spikes: A spike is generally formed by two strokes, drawn in approximately opposite directions, that overlap in the region of the tip. Spikes are a special case of valence-3 junctions where one branch is relatively short. In terms of topology, a junction exists where the overlap starts, and the tip forms an end point, as illustrated by Figure 4.12a. After such a structure is detected in the topology extraction, the reverse drawing procedure treats it as any other junction by selecting the straighter *CCs*, which results in the appropriate representation of the spike.

Name	Input Type	Image Resolution	Processing Time
Louis	Scan Clean	2048^2	3m 10s
Dracolion	Digital	1024^2	29s
Dr. Facilier	Scan Clean	1024^2	46s
Mr. Darling	Scan Rough	2048^2	3m 50s
Moose	Scan Digital	2048^2	3m 27s
Mouse	Digital	1024^2	1m 1s
Muten	Digital	1024^2	24s
Sheriff	Digital	1024^2	55s

Name	SSA		
	Valence: 3	> 3	$< 3/tot\ in\%$
Louis	95.3%	88.8%	1.6%
Dracolion	95.5%	100%	2.2%
Dr Facilier	96.0%	75.0%	4.8%
Mr. Darling	97.4%	97.4%	9.5%
Moose	98.6%	98.6%	8.2%
Mouse	95.8%	90.9%	8.0%
Muten	96.0%	N/A	0%
Sheriff	94.5%	94.5%	19.8

Name	CE		SPE	
	vs. ALT	vs. Base	vs. ALT	vs. Base
Louis	176 %	125 %	159 %	138 %
Dracolion	279 %	177 %	145 %	144 %
Dr Facilier	168 %	120 %	181 %	157 %
Mr. Darling	186 %	140 %	270 %	156 %
Moose	140 %	106 %	168 %	134 %
Mouse	355 %	158 %	136 %	113 %
Muten	212 %	125 %	379 %	288 %
Sheriff	187 %	123 %	145 %	115 %

Table 4.1: *Numerical results and ground truth evaluation for different input drawings. See Section 4.5 for a detailed discussion (ALT: Adobe Live Trace, Base: base version of our algorithm, where no reverse drawing is applied).*

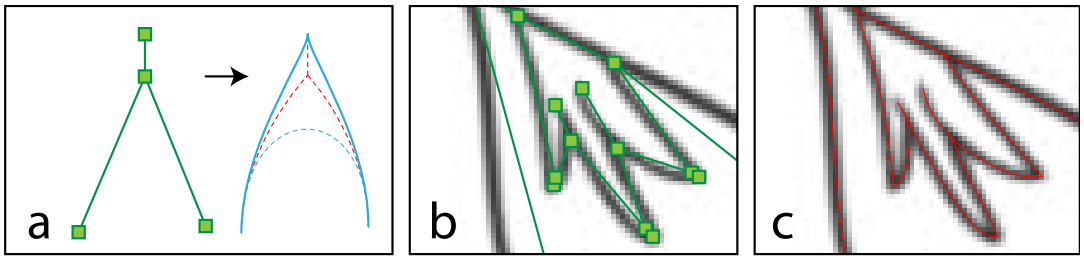

Figure 4.12: *Spikes can be considered a special case of a junction, where one of the branches is very short (a). (b-c) show respectively the topology and the resulting centerlines for several spikes.*

4.5 Validation and Results

We evaluate our approach on a variety of clean line drawings, including production drawings from 2D short and feature animations. (Figure 4.17). Of the paper examples, we include recently created drawings, as well as an archival piece which has degraded in quality over time and is therefore difficult to vectorize using existing techniques due to the age-related artifacts in the paper texture. We provide numerical and visual comparisons to two standard commercial tools, Adobe Live Trace [Adobe, 2012b] and Harmony [ToonBoom, 2010], as well as thinning algorithms, the Stentiford and Zhang-Suen (implemented in Wintopo [SiSoft, 2010]).

4.5.1 Evaluation

In order to evaluate the accuracy of our centerline reconstruction and to compare it to existing methods, we used a data set consisting of eight drawings (four scanned, four digitally drawn and then rasterized). In total, the drawings contain more than a thousand topologically relevant points (59.8% junctions, 40.2% endpoints), and about two thousand centerlines. For the digitally created images, accurate ground truth centerlines are available. For the scanned drawings, we asked an artist to manually trace the centerlines and highlight the correct topological configuration at junctions. The results of the evaluation are summarized in Table 4.1. We evaluate the following criteria:

Selection Scheme Accuracy (SSA): Algorithm 1 requires as input a stroke-curvature threshold parameter t. To pick an appropriate value, we proceeded as follows. For all images in our data set, an artist manually labeled all candidate centerlines as either *smooth* or *sharp turns*, obtaining a data set with more

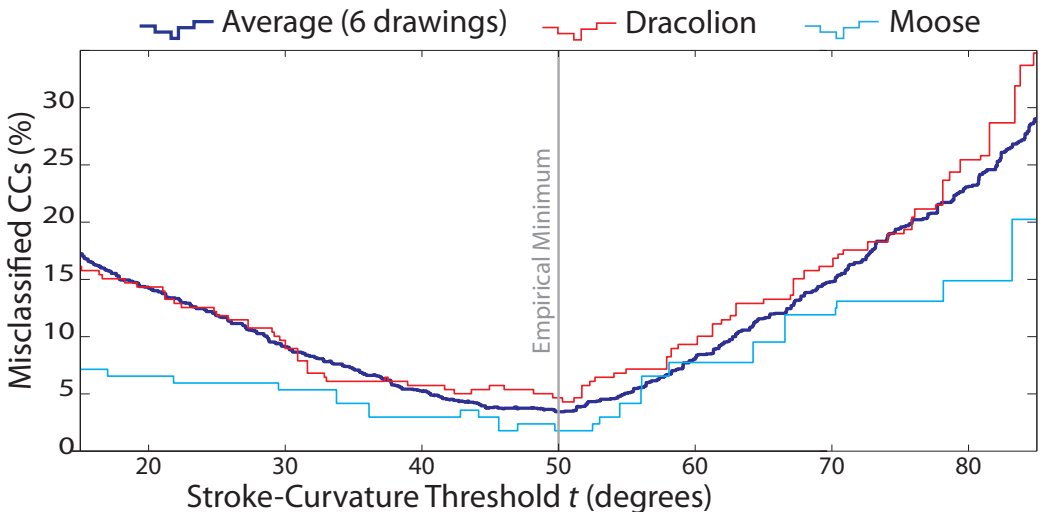

Figure 4.13: *Stroke-Curvature Thresholding. To establish an empirically optimal value of t, we built a data set containing more than 2000 CCs, manually labeled as either smooth or sharp turns. This image displays the classification obtained with Algorithm 1, plotting percentage of misclassified CCs versus the input threshold parameter t. For cross-validation, we show the average misclassification for 6 out of the 8 drawings, which suggests an optimal threshold value t, and finally we assess the quality of the prediction on the two remaining 2 drawings.*

than 2000 classified *CCs*. Then, as illustrated in Figure 4.13, we evaluated the percentage of misclassified *CCs* against different thresholds t and picked the empirical optimum at $t = 50°$. To validate this choice, we then performed a leave-one-out cross-validation (repeated for all drawings). On average, our choice produces an optimum prediction error of 1.99°, which corresponds to an error of 0.25% in terms of misclassified *CCs*. Overall, with the chosen threshold value $t = 50°$, our algorithm produces results that match the artist's classification in 95.5% of the cases. For further inspection, we broke down the values based on valence. Junctions with valence greater than 3 occurred less frequently (5.7% of total), and were more difficult to classify (accuracy 93.4%).

Centerline Error (CE): We evaluate centerline quality by computing the average minimum distance of dense sample points on the extracted centerlines to the ground truth centerlines (see Figure 4.14). The error with our approach has an average of only 4.13% of the average stroke thickness. We perform the same computation for Adobe Live Trace, resulting in an average of 7.97%, and for the base version of our algorithm (where no reverse drawing is ap-

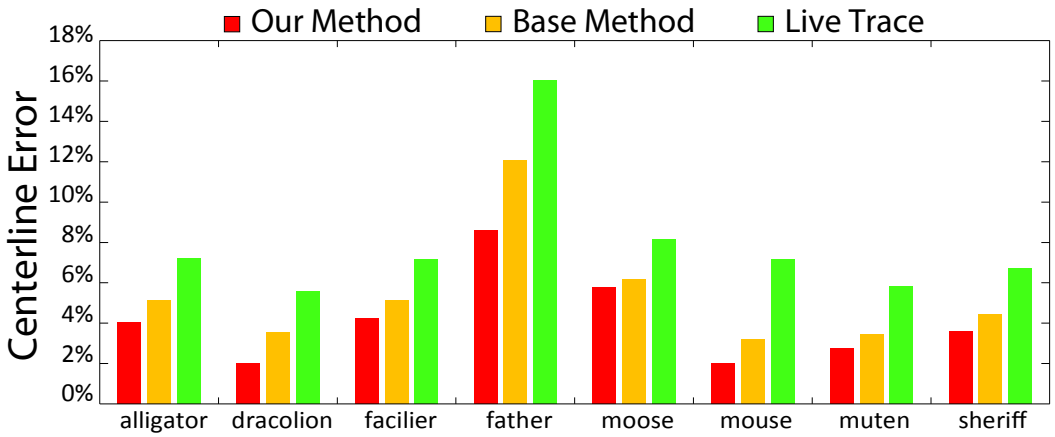

Figure 4.14: *Centerline Error. This image shows a comparison of the centerline error for the results obtained with Adobe Live Trace, our method, and the base version of our method, where no reverse drawing is applied. The values are expressed as percentage of the average stroke thickness in each drawing. Our method consistently produces a better score.*

plied) which shows an average error of 5.38%. For the result in the specific drawings, refer to the second to last column in Table 4.1. Our method shows consistent improvements of the centerline quality for all examples.

Salient Points Error (SPE): Junctions and endpoints are critical elements in the vectorization of line drawings. The quality of a reconstruction can be assessed by considering both the correctness in the placement of these points as well as the correctness in the overall topology. Given a drawing, consider two sets of salient points G and M, from the ground truth and the method to be evaluated respectively. We then compute:

$$D = \sum_{i \in M} \min_{j \in G} dist(i,j) + \sum_{j \in G} \min_{i \in M} dist(i,j) \tag{4.3}$$

A good reconstruction should produce a set M that is as similar as possible to G. In the case of correct topology, the contribution of a pair $(i \in M, j \in G)$ will express the quality of the salient point placement. However, if a method does not capture the correct topology (by either missing a junction or detecting more junctions than there actually are) mismatched pairs will penalize the score with additional accumulated distance. Figure 4.15 shows the comparison of our method with Adobe Live Trace and the base version of our method. To assess the performance across our data set, we normalized the values D considering twice the number of salient points $2 \cdot |G|$ in each drawing.

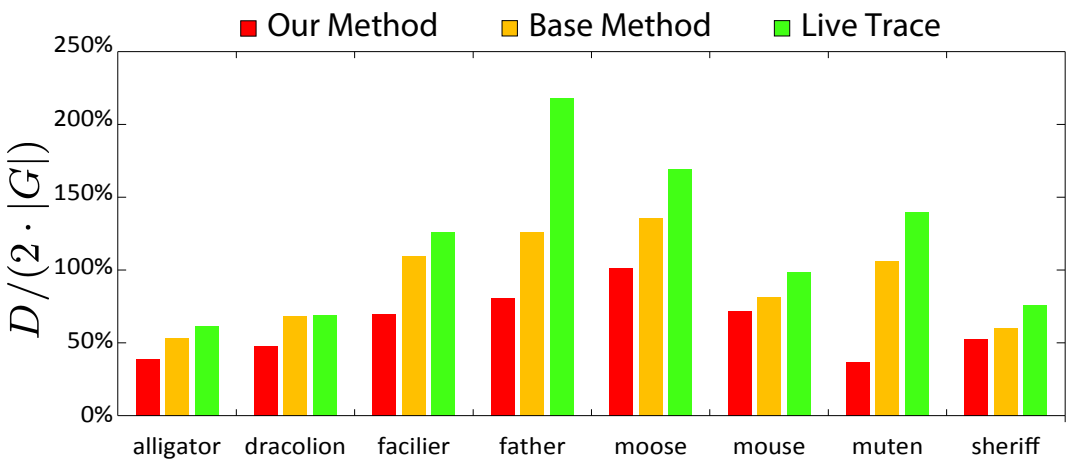

Figure 4.15: *Salient Points Error. This image shows a comparison of the salient points error for the results obtained with Adobe Live Trace, our method, and the base version of our method, where no reverse drawing is applied. The values are expressed as percentage of the average stroke thickness in each drawing. Our method consistently produces a better score.*

4.5.2 Input Resolution and Clustering Robustness

The clustering step described in Section 4.4.1 samples the input image to generate bands of moving pixels that stop when they meet. We experimented with super-sampling as a way to cope with very low input resolutions. Even 1-pixel wide strokes, if super-sampled appropriately with a smooth filter (in our tests: bi-cubic filter, 4 samples per pixel), can be reconstructed accurately, leading to robustness comparable to competing approaches.

As shown in Figure 4.16, the centerline error relative to the ground truth (logarithmic scale) drops exponentially with increasing resolutions. Differences between our method and Adobe Live Trace become more evident at higher resolutions.

4.5.3 Result Images

For a general overview of the capabilities of our method, we present a selection of vectorization results in Figure 4.17, taken from the eight drawings used in the previous section for error evaluation. Notice how several ambiguous regions are properly handled, and in most cases the proper junction

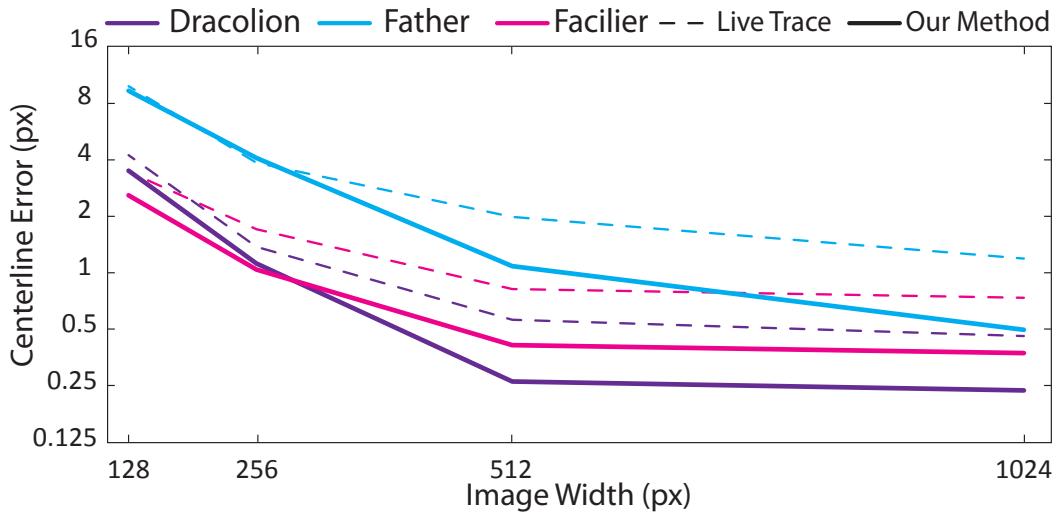

Figure 4.16: *Effect of increasing the input resolution. Both our method and Adobe Live Trace (dashed) exhibit an exponential reduction of the error. Differences between the two methods become more evident at higher resolutions.*

configuration is selected. These results combine the advantages of both the pixel clustering and the reverse drawing.

Figure 4.18 illustrates the benefits of pixel clustering on its own. Figure 4.18a,b show the vectorization results obtained with Adobe Live Trace and Sisoft Wintopo. Figure 4.18c shows results from only the Base portion of our algorithm, where centerlines are traced from the pixel clusters, but no reverse drawing is applied. Notice how both medial axis and thinning methods (a,b) rely on the creation of boundaries, usually obtained through color thresholding. However, with low thresholds, nearby strokes are not distinguished properly, and with high values parts of the drawings are lost. Pre-sharpening the images can alleviate these problems, but the proper kernel size has to be used; in our experiments, this approach required manual tuning to achieve good results. In contrast, our method successfully separates nearby strokes, while avoiding stroke losses.

Finally, Figure 4.19 provides visual comparisons of our complete method to leading commercial vectorization implementations, Toon Boom Harmony [ToonBoom, 2010], Adobe Live Trace [Adobe, 2012b], and SoftSoft Wintopo, respectively. Notice how the existing techniques have difficulties in discriminating nearby strokes, which results in merged centerlines for separate strokes and an incorrect drawing topology. Moreover, junction points are placed at inaccurate positions. Our method finds a more natural place-

Figure 4.17: *Collection of vectorization results generated with our method. The Mr. Darling image is not fully clean (the paper quality is degraded and lines not sharp), and thus represents a borderline case.*

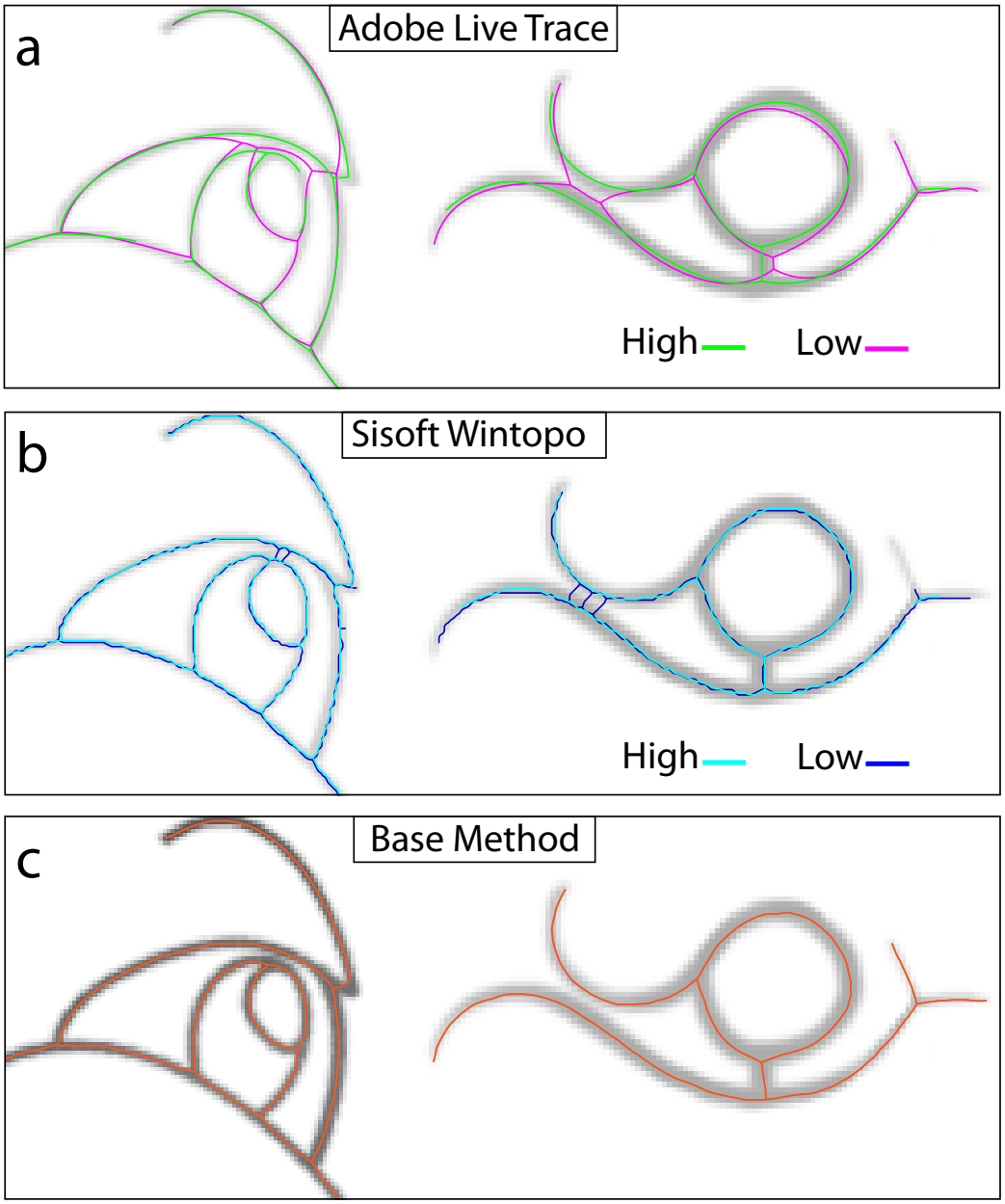

Figure 4.18: *Pixel Clustering Comparison. This image shows a comparison between Adobe Live Trace (a), Sisoft Wintopo (b), and the result of our base algorithm (c), where only the base centerlines have been extracted, and no reverse drawing has been applied. In (a) and (b) different color threshold parameters have been used. Notice how in order to get the nearby strokes separated, the color threshold must be set to high values, losing other parts of the drawing. Our method successfully separates the strokes without such losses.*

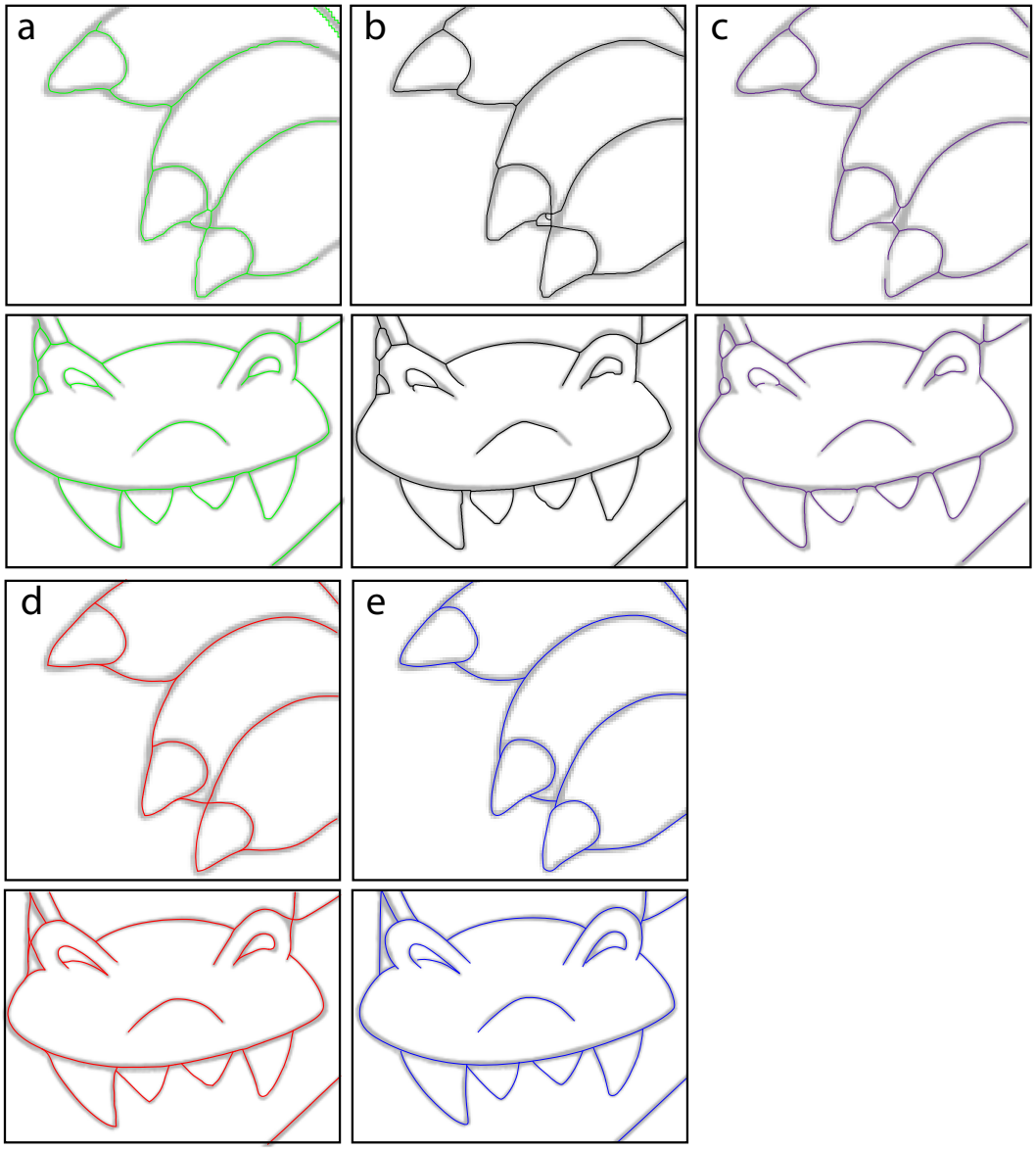

Figure 4.19: *General Comparison. This image shows close-up comparison between Adobe Live Trace (a), SiftSoft Wintopo (b), Toon Boom Harmony (c), our method (d), and the ground truth (e).*

ment. For these comparisons, we attempted to tune the parameters of the software packages to obtain the best possible results. Our method uses the standard parameter values described in the previous sections and requires no per-drawing tuning.

4.5.4 Processing Time

The processing time of our method for each of the input images is provided in Table 4.1, measured on a desktop PC[2]. The time necessary to process an image depends both on its resolution as well as the number of strokes and junctions. Due to the more complex topological analysis of a drawing, our algorithm requires more processing time than tools such as Live Trace or Harmony. However, the timings are still in the range of only a few seconds to minutes, and the necessary time spent in post-production to correct the centerline estimates and junctions is significantly reduced compared to the previously available solutions (see Figure 20).

For the Louis example (Figure 4.17), we compared the timing of manual post-processing of the output to fix erroneous junctions for Live Trace and our method: it took an artist 12 times longer to produce comparable results for the Live Trace example.

4.5.5 Limitations and Future Work

An important application for this system is 2D animation. As is, our system can be used in tandem with inbetweening techniques, such as [Whited et al., 2010]. However, the vectorization quality could be improved by considering the information contained in subsequent frames, improving the decision making (Algorithm 1) and recovering from errors in the topology. The difficulty however is that distinct elements moving independently may drastically change both the topology and the junction configurations, making these extensions very challenging.

As for any raster based method, image quantization and noise play a crucial role. In the case of low resolution input, details are hard to extract (Figure 4.21a). In our tests with variable resolution (see Figure 4.16) we observed a similar response for our method and Adobe Live Trace, but in order to not break - i.e. have enough moving pixels - the proposed pixel clustering

[2]Mac Pro, Quad-Core 2.66 GHz, 4GB RAM

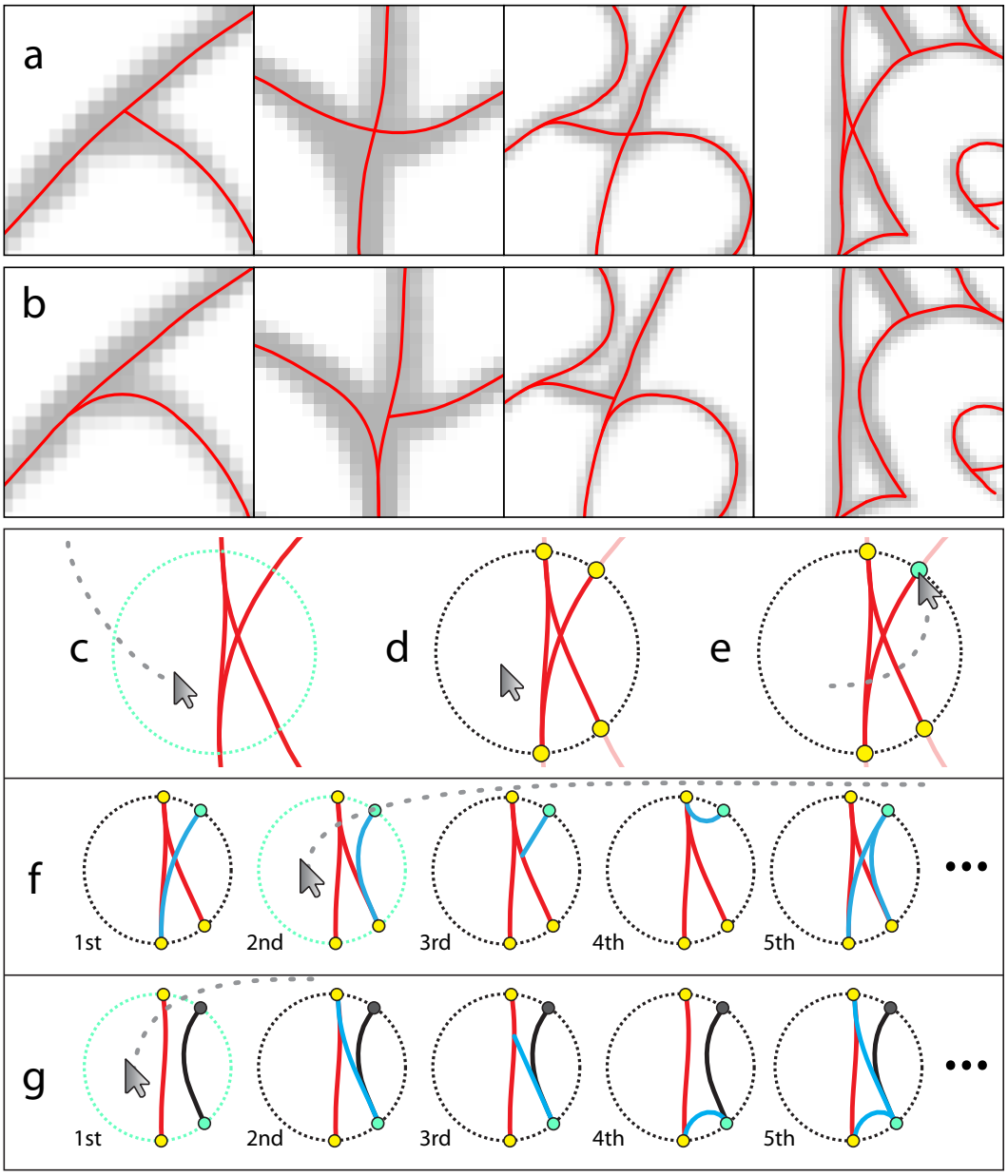

Figure 4.20: *System output (a). With a user interaction of only a few seconds per junction, different configurations can be obtained (b). The interaction steps are shown in (c-g). First, the user selects the junction (c), and activates the editing mode (d). BCs entry points (yellow dots) are used by the user to make the desired changes. Once an entry point is clicked (e), the system shows a set of valid configurations to choose from (f). The first criterion is that the entry point must be connected: 1 connection is favored over 2, 2 over 3, etc. Straighter CCs are the first choice, then linearly prolonged BBs, and finally, rejected CCs. If a choice influences the connectivity of another entry point, that point is automatically selected, and possible configurations displayed (g). Previous choices are marked in black. This process requires at most one choice per entry point.*

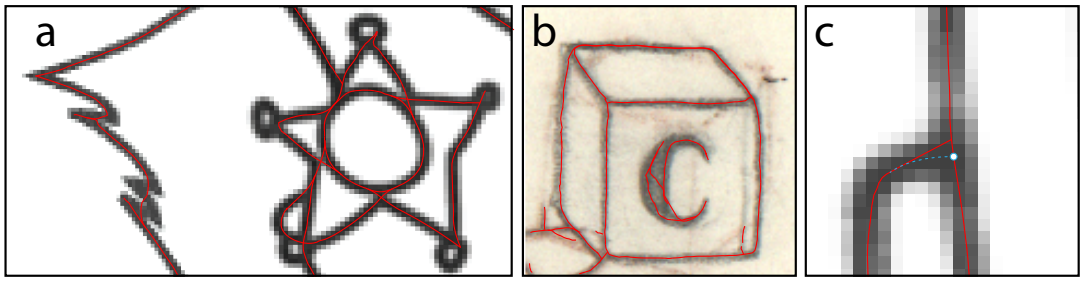

Figure 4.21: *Limitations. With the current method, small details (a) may not be captured by the topology extraction. The method is also not designed to work with solid areas (b). (c) The linear extension of the base centerlines does not always produce the perfect junction location (higher order extrapolation shown in blue).*

requires super-sampling. Noise may lead to strokes being torn apart. Additionally, we observe in Figure 4.16 the decay of the improvement rate as the resolution increases. This is partly due to the error getting closer to zero, but also to the difficulty in exploiting the additional information.

While Algorithm 1 in most cases produces good results, there may be cases of technical drawings, with grid structures or specific texture-like patterns, where a failure of the algorithm appears in many junctions, making manual fixing very time consuming. Here, a possible approach would be to apply machine learning to update the guiding criteria. The current spike detection relies on the presence of particular topological structures, and might also be improved by using machine learning in conjunction with the proposed stroke-curvature measure to be able to explicitly extract sharp corners

Finally there are a number of minor limitations. As shown in Figure 4.21b, our method is not designed to work with solid areas. Additionally, as explained in Section 4.4.3, base centerlines that are selected by Algorithm 1 for the final configuration are extended *linearly* inside the ambiguous regions. Figure 4.21c shows a case where this linear method results in a less accurate junction location than the one obtained with higher order extrapolation.

4.6 Conclusion

We have described a novel vectorization technique for clean line drawings which produces a high-quality representation suitable for vector processing. Our approach consists of two techniques: a gradient based pixel cluster-

ing that helps disambiguate difficult cases, and a reverse drawing procedure which exploits the drawing topology to make educated choices when dealing with junctions. Since these techniques are independent, the reverse drawing can be applied to improve the result of existing techniques that provide the drawing topology.

We have demonstrated the application of our method to a variety of professional examples. Our results show how our approach improves the vectorization of junctions and nearby strokes which represent the major shortcomings of state-of-the-art solutions when it comes to clean line drawings. Such accurate junction and centerline recovery makes stroke-based editing operations such as automatic inbetweening more feasible in a digital pipeline.

Possible future directions include addressing the limitations of the current method by exploring the vectorization of more sketchy and noisy drawings, considering image pre-filtering using LoG [Chen et al., 1987]. Moreover, we are interested in studying the semantic information present in sketchy drawings, considering strokes not only as sets of pixels, but as objects with mathematical properties (e.g., trajectory, direction) with the goal of exploiting such semantics in a clustering approach.

CHAPTER 5

A Scribble-based Segmentation Tool

In this Chapter, we present *Smart Scribbles*, a scribble-based interface for user-guided segmentation of digital sketchy drawings.

In contrast to previous approaches based on simple selection strategies, Smart Scribbles exploits richer geometric and temporal information. This enables better analysis of the similarity between the strokes making up a drawing, as well as the relationship between the strokes and the input Scribbles.

We introduce a novel energy minimization formulation in which both geometric and temporal information from digital input devices is used to define stroke-to-stroke and scribble-to-stroke relationships. Although the minimization of this energy is, in general, a NP-hard problem, we use a simple heuristic that leads to a good approximation and permits an interactive system able to produce accurate labelings even for cluttered sketchy drawings.

We demonstrate the power of our technique in several practical scenarios such as sketch editing, as-rigid-as-possible deformation and registration, and on-the-fly labeling based on pre-classified guidelines.

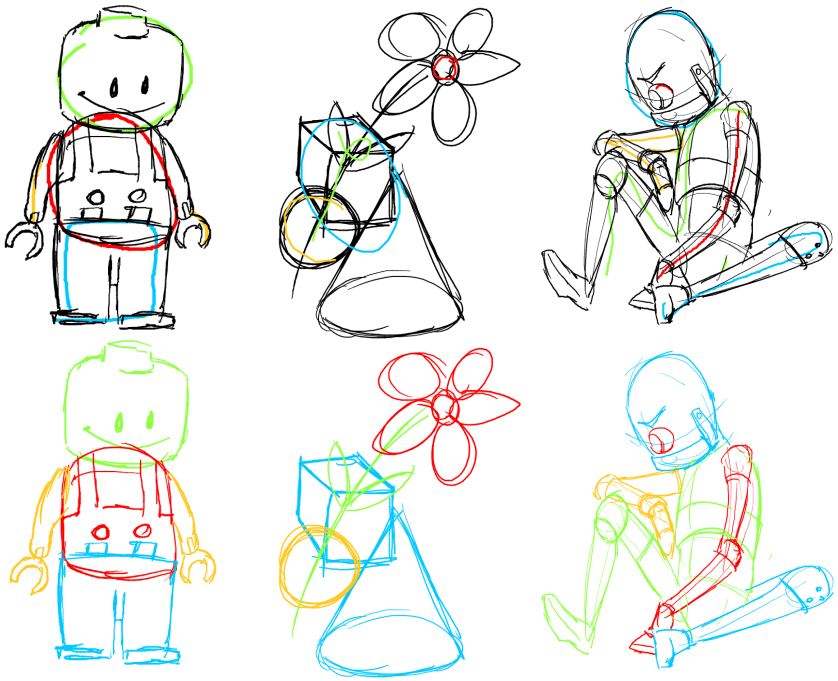

Figure 5.1: *Sketch segmentation: For each example pair, Smart Scribbles on the top produce the segmentation on the bottom.*

5.1 Introduction

Sketchy drawings are prevalent across a wide range of applications and domains. In early development phases, rough drawings are used, for example, for concept art in product design, and for storyboards in animation environments, and are favored both for the speed of generation, and the expressiveness of the results. A sketchy style also has a place in finished art – providing a level of visual richness not found in "clean" line representations, i.e. drawings constructed from crisp, distinct outlines and minimal interior detail.

Modern digital devices and graphics software solutions offer powerful stylization, deformation, morphing, and animation capabilities for 2D drawings. However, in order to perform this type of high-level task, a certain degree of understanding of the content of the drawing is required. This is a challenging problem due to the significant gap between the ability of a human to discern structure in a drawing and the capability of an algorithm to derive it from low level stroke information. This is true even for clean line drawings, and most existing approaches rely on the presence of a human user to provide sufficient information to guide the task.

The problem of extracting structure from drawings becomes substantially more difficult for sketchy input, and this is one reason it is far less common to find a consistently sketchy style in full-length animations or automatic support for sketchy input in high-level editing packages. One important category of drawing abstraction is segmenting the drawing into logical parts. To-date, there is no efficient method available for automatic segmentation in this domain. In contexts where a breakdown of the drawing is required, segmentation is typically achieved by design: the drawings are created in different layers, one for each logical component. This approach is too limiting in practice: it requires a priori knowledge of the use of the drawing, is cumbersome (especially when different tasks require different segmentations), and is an error-prone process, even for experienced artists.

We seek a semi-automated solution to segmenting sketchy drawings that is fast enough for interactive use, but also predictable and easy to use – making it accessible to even the most novice user.

To this end, we propose the concept of *Smart Scribbles* as an accurate and simple way for the user to specify semantically meaningful stroke clusters within a drawing (see Figure 5.1). In contrast to previous methods that use scribbles as positional constraints for various image editing tasks [Boykov and Jolly, 2001, Levin et al., 2004, An and Pellacini, 2008, Sykora et al., 2009b], our formulation considers more detailed geometric (position, orientation, curvature) and temporal information (time of creation), analyzing how strokes in the drawing relate to each other, and how well the Smart Scribbles match them. In addition, we introduce the concept of locality control as a way of conveniently trading off the scribbles' areas of influence for accuracy. This allows our system to produce desired results with minimal user intervention even for cluttered sketches.

We evaluate our approach on a collection of digitally drawn sketches of varying complexity, and demonstrate the application to various tasks including sketch editing, ARAP deformation and registration. As our solution is fast to compute, our method enables tight integration of these tasks within an interactive digital drawing session.

5.2 Related Work

Relevant prior art can be divided into three main categories: sketch labeling interfaces, scribble-based image segmentation and classification of vector fields in scientific visualization.

User-guided labeling of strokes in hand drawn images plays a central role in many sketch-based editing systems. In Lank et al. [Lank and Saund, 2005], the authors present an approach for inferring user intent from the local velocities, accelerations and curvatures of the selection lasso. More recently, Wolin et al. [Wolin et al., 2007] presented a technique for labeling groups of strokes from a vectorized sketch where the system attempts to automatically fragment continuous strokes into logical pieces to assist the user. Both of these techniques ultimately utilize a region-based selection approach. ScanScribe, a system developed by Saund and colleagues [Saund et al., 2004], presents the user with an intuitive selection paradigm that allows for the creation of objects from collections of pixels and supports further grouping into composite objects. The system is able to automatically segment the image into basic primitives, such as linear curve fragments, and then group them into more complex objects, such as rectangles, based on an alignment metric computed between nearby fragments (or finding perceptually closed paths as proposed in [Saund, 2003]). Two limitations of this automatic technique are 1) limited complexity of objects detected by the system and 2) the inability to handle sketchy overlapping curve fragments, thus requiring more traditional and tedious lasso/selection-box methods for more complex drawings.

The approach presented in this paper leverages previous works on interactive image segmentation in order to optimize the labeling process based on user scribbles. Boykov et al. [Boykov and Jolly, 2001] developed such an approach based on graph cuts for segmenting images and finding optimal boundaries between objects. In [Levin et al., 2004], Levin and colleagues present a similar framework based on a least-squares optimization for colorizing gray-scale images by roughly labeling regions with colored scribbles. More recently, An et al. [An and Pellacini, 2008] developed an interactive energy minimization framework for propagating color edits to similar regions throughout the image. Our approach is most similar to LazyBrush [Sykora et al., 2009b] a graph cut based system for sketch painting, i.e., the selection of regions in sketchy drawings. The main difference is that this system cannot provide the labeling of the strokes that bound each painted region. From this point of view, our framework can be seen as a generalization of LazyBrush, since it extracts meaningful boundaries first and then builds regions inferred from those boundaries. Because this process removes clutter from the input drawing, it greatly improves the accuracy of selection and reduces the amount of user interaction needed to obtain clean results.

Our approach also bears some resemblance to sketch-based clustering of vector fields in scientific visualization [Wei et al., 2010]. Here the aim is to allow the user to sketch 2D curves and use them as a query to retrieve 3D field lines whose view-dependent 2D projection is most similar to the input sketch. The

curvature along the input sketch is used to measure the similarity between two curves using the edit distance [Wagner and Fischer, 1974]. In our approach, curvature is also used to distinguish between different shapes. However, the main advantage of our work is that we formulate an energy minimization problem where, in addition to shape similarity, we also take proximity, orientation, temporal information, and smoothness of the final labeling into account. As a result, our system can produce reasonable clustering even in cases when the the shape of the input sketch is very rough or incomplete.

5.3 Method

The method we present allows users to intuitively segment digital sketches into semantically meaningful regions. The input to our framework consists of a digitally hand-drawn sketch and a small set of rough *scribbles*. The input sketch is composed of a set of *strokes*, which are piecewise linear curves represented by sets of 2D vertices recorded from a digital input device such as a tablet. For every vertex of a stroke we additionally store its time of creation. This lets us readily compute the drawing speed, and, in addition, helps differentiate strokes which are spatially close but are drawn at different moments in time.

The input scribbles, which we term *Smart Scribbles*, are special strokes that indicate the user's intent to segment a particular portion of the drawing. Two criteria related to the scribble primitives are critical in order to ensure a useful and intuitive system. First, Smart Scribbles should not have to closely follow the target region. However, if desired, the user should be able to precisely select localized regions. We call this property *locality control*. The second criterion specifies that the time of creation of the Smart Scribble should not influence the segmentation results.

We observe that generally speaking, processing strokes as a whole is very difficult. A single stroke can be arbitrarily complex: it can cross or overlap with itself multiple times, and/or it can densely cover an area the artist wished to fill in. For this reason, we break strokes and scribbles into linear *segments* by densely resampling the input. Any property defined locally over the stroke can easily be transferred to the segments.

We formulate the task of sketch clustering as an optimization problem, where the goal is to label each stroke in a way that minimizes an energy function. The remainder of this section describes in detail each of the steps used by our method. The energy function is defined in Section 5.3.1. It relies on a smoothness and data term which are described in separate Sections 5.3.1 and 5.3.1.

Finally in Section 5.3.2 we discuss the minimization method used to compute the final solution to the stroke labeling.

5.3.1 Energy function

We formulate the task of stroke labeling as an energy minimization problem. Figure 5.2 depicts the concept of our design. The input consists of a set of stroke segments S and a set of scribble segments R associated with a set of labels L. The goal is to find a labeling, i.e., an assignment ϕ of the labels in L to every segment in S, that minimizes the following energy function E:

$$E(\phi) = \sum_{i,j \in S} V_{i,j}(\phi_i, \phi_j) + \lambda \sum_{i \in S} D_i(\phi_i) \tag{5.1}$$

where $V_{i,j}$ is a smoothness term that captures the cost of the labeling with respect to the similarity between two stroke segments i and j. The data term D_i measures the affinity between scribbles and strokes. The parameter λ controls the relative influence of the smoothness and data terms.

Smoothness Term

The smoothness term is defined as:

$$V_{i,j}(\phi_i, \phi_j) = \prod_{g \in G} \delta(g(i,j), \sigma_g) \tag{5.2}$$

when $\phi_i \neq \phi_j$, otherwise it is zero. G is a set of similarity terms:

$$\begin{aligned} prox(i,j) &= ||\vec{p}_j - \vec{p}_i|| \\ dir(i,j) &= 1 - |\vec{d}_i \cdot \vec{d}_j| \\ curv(i,j) &= 1 - min(c_i, c_j)/max(c_i, c_j) \\ time(i,j) &= |t_j - t_i| \end{aligned}$$

where i and j are two segments, and p, d, c and t are the position, direction, radius of curvature, and time of creation associated with each segment. The fall-off function δ is defined as:

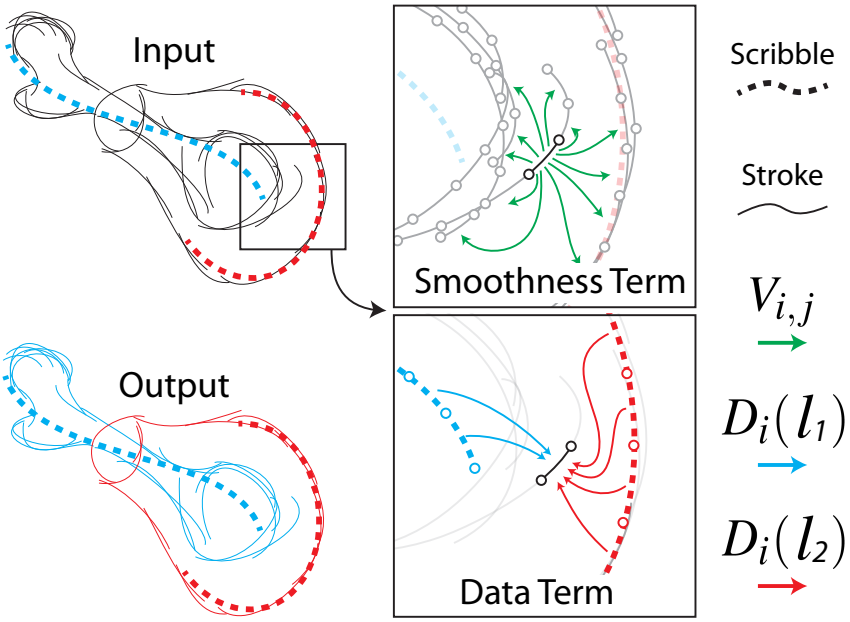

Figure 5.2: *Energy definition overview. The input consists of a set of strokes (black) and scribbles (red and blue dotted lines). The output consists of a labeling of all strokes (the labeling is indicated here by the red/blue color assignment to the strokes in the output).* ***Smoothness Term:*** *For a segment i and a neighbor segment j, $V_{i,j}$ expresses the energy of assigning the same label to both i and j, based on how similar they are.* ***Data Term:*** *Given a labeling $\phi_i = l_*$ (assigning label l_* to segment i), $D_i(\phi_i)$ expresses the energy of the labeling, which is a function of the similarity of segment i to all scribbles associated with l_*.*

$$\delta(g(i,j),\sigma_g) = \exp\left(-\frac{g(i,j)^2}{\sigma_g^2}\right) \tag{5.3}$$

These terms have intuitive meanings. Segments that are close together, are parallel, belong to strokes with similar curvature and have been generated close in time are likely to belong to the same semantic region. They should therefore get larger similarity scores. Notice that consecutive segments of a stroke are likely to have high similarity with respect to all the defined terms.

Data Term

The data term is defined as:

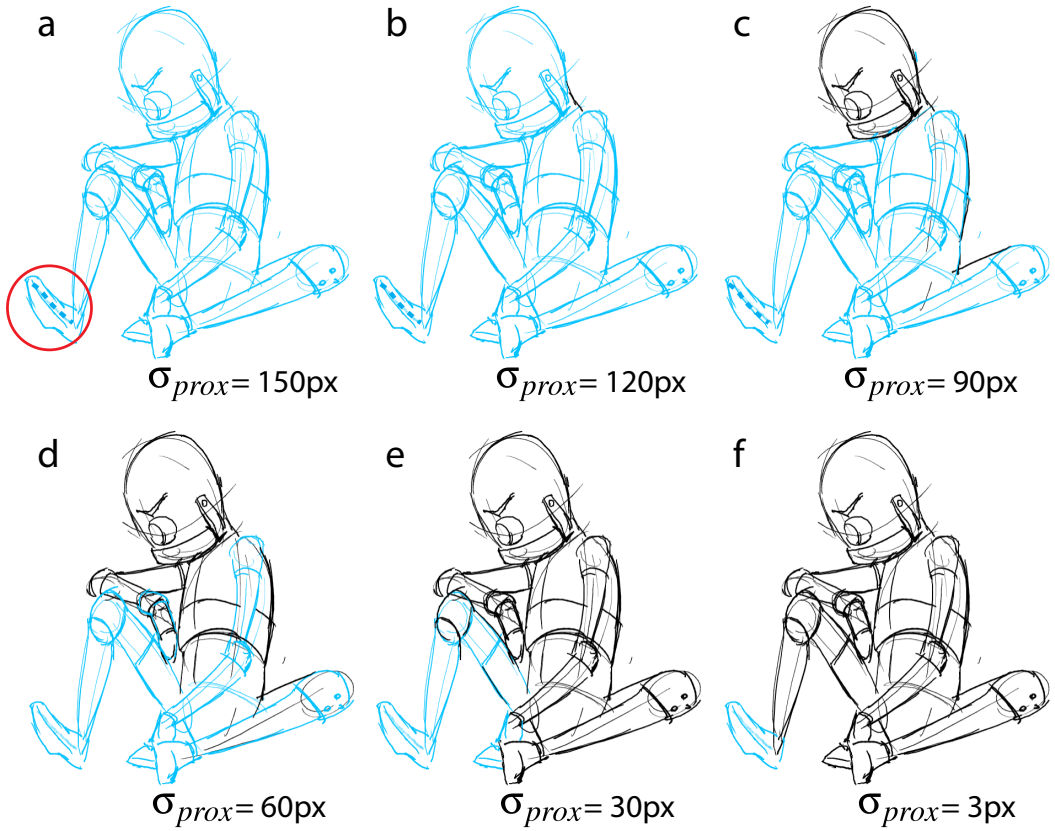

Figure 5.3: *The effect of the locality control by varying σ_{prox}: A blue scribble is drawn on the foot (circled in red). On the right, the value of σ_{prox} is progressively increased. Notice how the selection becomes progressively more local as the influence of the blue label gets overruled by the background label (shown in black).*

$$D_i(\phi_i) = 1 - \max_{r \in R(\phi_i)} A(i,r) \tag{5.4}$$

where $R(\phi_i)$ denotes a set of scribble segments r with label ϕ_i. The affinity $A(i,r)$ is defined as:

$$A(i,r) = \prod_{g \in G_{data}} \delta(g(i,r), \sigma_g) \tag{5.5}$$

Here, as with the smoothness term, we measure the similarity between segments rather than strokes. However, as scribbles have no associated time information, we reduce the set of similarity terms to $G_{data} = \{prox, dir, curv\} \subset$

G. Additionally, we alter the definition of curvature to become oriented: $curv(i,j) = ||\vec{c_i} - \vec{c_j}||$. This allows extra control in separating curves with the same curvature but different orientation (e.g. the tangled lines in Figure 5.7).

One of our main goals is to allow users, if desired, to have precise local control over the strokes that get affected by each scribble. To illustrate this, we consider a scenario where the user only draws one scribble, as shown in Figure 5.3a. In this case, because no concurrent label exists, all strokes are selected. This behavior, though reasonable, is not in line with a user's expectations of having local control.

To address this, we introduce an artificial background label $b \in L$, in addition to the labels prescribed by the user. This new label has a constant influence on each stroke segment i regardless of the existence of any particular user-defined scribble, i.e., $A(i,b) = B$. The background label therefore serves as a lower bound for computing the max component in the data term (5.4).

Furthermore, we control the locality of each scribble r by modifying its proximity fall-off δ (5.3) as follows:

$$\delta(prox(i,r), \sigma_{prox}) = \frac{1}{\sigma_{prox}} \exp\left(-\frac{prox(i,r)^2}{\sigma_{prox}^2}\right). \tag{5.6}$$

Here σ_{prox} follows the desired locality (i.e., is large for global influence and small for local influence) and the normalization term $1/\sigma_{prox}$ ensures the integral over the fall-off function stays equal for different values of σ_{prox} (i.e., amplitude is high for small values and low for large ones). In other words, the overall energy remains constant, while its spatial spread is controlled. When sigma σ_{prox} becomes very low, the response of the fall-off function (5.6) for distant stroke segments also becomes very low and can therefore be easily overridden when computing the max value in (5.4). This is illustrated in Figure 5.3b-f, where, in the case of strokes that are distant to the local scribble, the threshold B becomes active and overrides the influence of the blue label.

There are several possible ways to control the parameter σ_{prox}. One natural way is to use the speed of the scribble based on the experimentally demonstrated linear relationship between speed and perceived locality [Accot and Zhai, 1997]:

$$W = \frac{\beta \cdot L}{T - \alpha}. \tag{5.7}$$

Here W is the selection radius, L is length of the scribble, T is time spent on drawing it, and α and β are empirically measured constants. This rule

was used to control the selection locality in systems having limited modality [Lank and Saund, 2005]. Alternatively, one could consider the use of pen-pressure, or - in the case of binary modality - a simple key toggle to switch between two locality values.

5.3.2 Optimization method

As shown in [Boykov et al., 1998], minimizing the energy function defined in Equation 5.1 is equivalent to solving a multi-way cut on a specific weighted graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{S, L\}$ is a set of vertices and $\mathcal{E} = \{\mathcal{E}_s, \mathcal{E}_l\}$ is a set of edges (See Figure 5.4). The graph vertices $\mathcal{V}$ consist of stroke segments S and label terminals L. Each stroke segment $i \in S$ is connected to all other stroke segments $j \in S - \{i\}$ via edges $\mathcal{E}_{i,j}$ having weight $w_{i,j}$ equal to the smoothness term $V_{i,j}$ when $\phi_i \neq \phi_j$. In addition, auxiliary edges $\mathcal{E}_{i,l}$ connect stroke segments $i \in S$ to label terminals $l \in L$. Each $\mathcal{E}_{i,l}$ has weight $w_{i,l} = \lambda(1 - D_i(l))$, where λ is the parameter defined in Equation 5.1 .

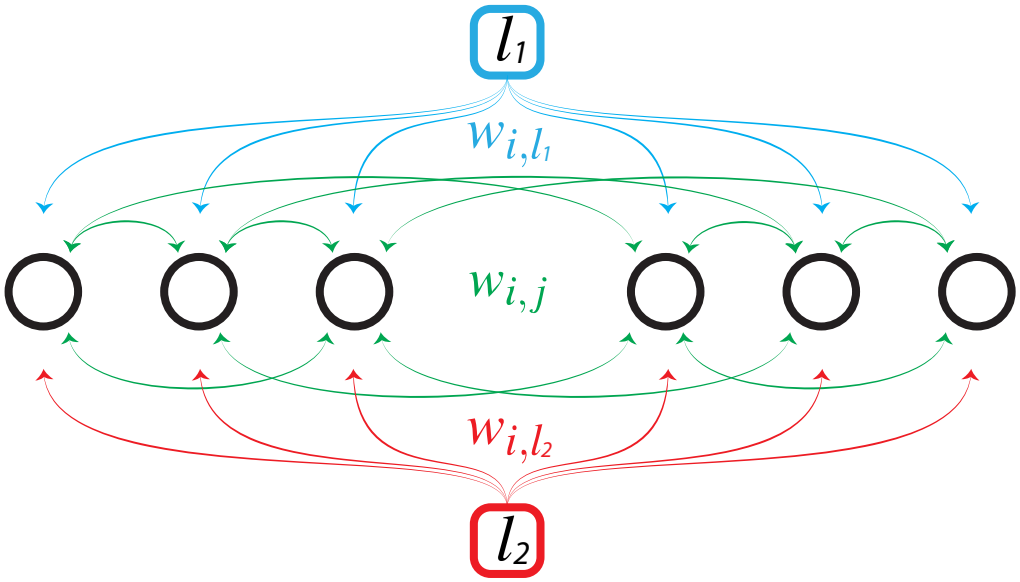

Figure 5.4: *Graph Construction. Stroke segments are shown as black circles. Terminal labels (in this example l_1 and l_2) are shown as colored squares. The graph edges $w_{i,j}$ reflect the smoothness terms $V_{i,j}$ between the stroke segments $i, j \in S$, while the data terms $D_i(l)$ for stroke segment $i \in S$ and label $l \in L$ are captured by the weights $w_{i,l}$.*

The multi-way cut problem with two terminals is equivalent to a max-

flow/min-cut problem for which efficient polynomial algorithms exist [Boykov and Kolmogorov, 2004]. However, for three or more terminals the problem is NP-hard [Dahlhaus et al., 1992]. To obtain a good approximate solution we use a simple divide-and-conquer heuristic previously proposed in [Sykora et al., 2009b] to gradually simplify the N-terminal problem into a sequence of $N-1$ binary max-flow/min-cut sub-problems. This approach provides results similar to more advanced techniques (such as α-expansion or α/β-swap [Boykov et al., 2001]), but is significantly faster and therefore better suited for interactive applications.

5.4 Applications

The labeling produced by our approach can be utilized to generate input to perform region as well as stroke segmentation (see Figure 5.6a). Once the labels for the segments of each stroke are computed, we can automatically obtain an area mask of the enclosed region using the LazyBrush [Sykora et al., 2009b] algorithm (see Figure 5.5). To this end, we first render all segments assigned to a specific label to a raster image. This image is used both as an input gray-scale image (Figure 5.5a) and as foreground soft scribbles (red in Figure 5.5b) for input to LazyBrush. In addition, we use a default background hard scribble around the image boundary (blue in Figure 5.5b). Given this input, LazyBrush produces the desired area mask (Figure 5.5c). As compared to other naive methods (like convex-hull or flood-fill), this approach works with concave regions and is robust to small gaps.

For more complex sketches, the user may need to specify additional Smart Scribbles (Figure 5.5d) to classify interior strokes and use them as additional background soft scribbles for LazyBrush (Figure 5.5e). These new scribbles enable the area computation method to produce masks that contain holes (Figure 5.5f).

We note that similar masks (Figure 5.5i) can be produced with the original LazyBrush algorithm directly. However, knowing the segmentation of the areas is not sufficient to provide a labeling of the individual strokes, because strokes at the boundaries between different area masks cannot be consistently assigned to one mask or another (Figure 5.5g). Moreover, with the original LazyBrush algorithm, the user must be more careful, since the optimization only takes into account the position of the scribbles. In contrast, our framework also considers orientation, curvature, and time (compare Figure 5.5d and h).

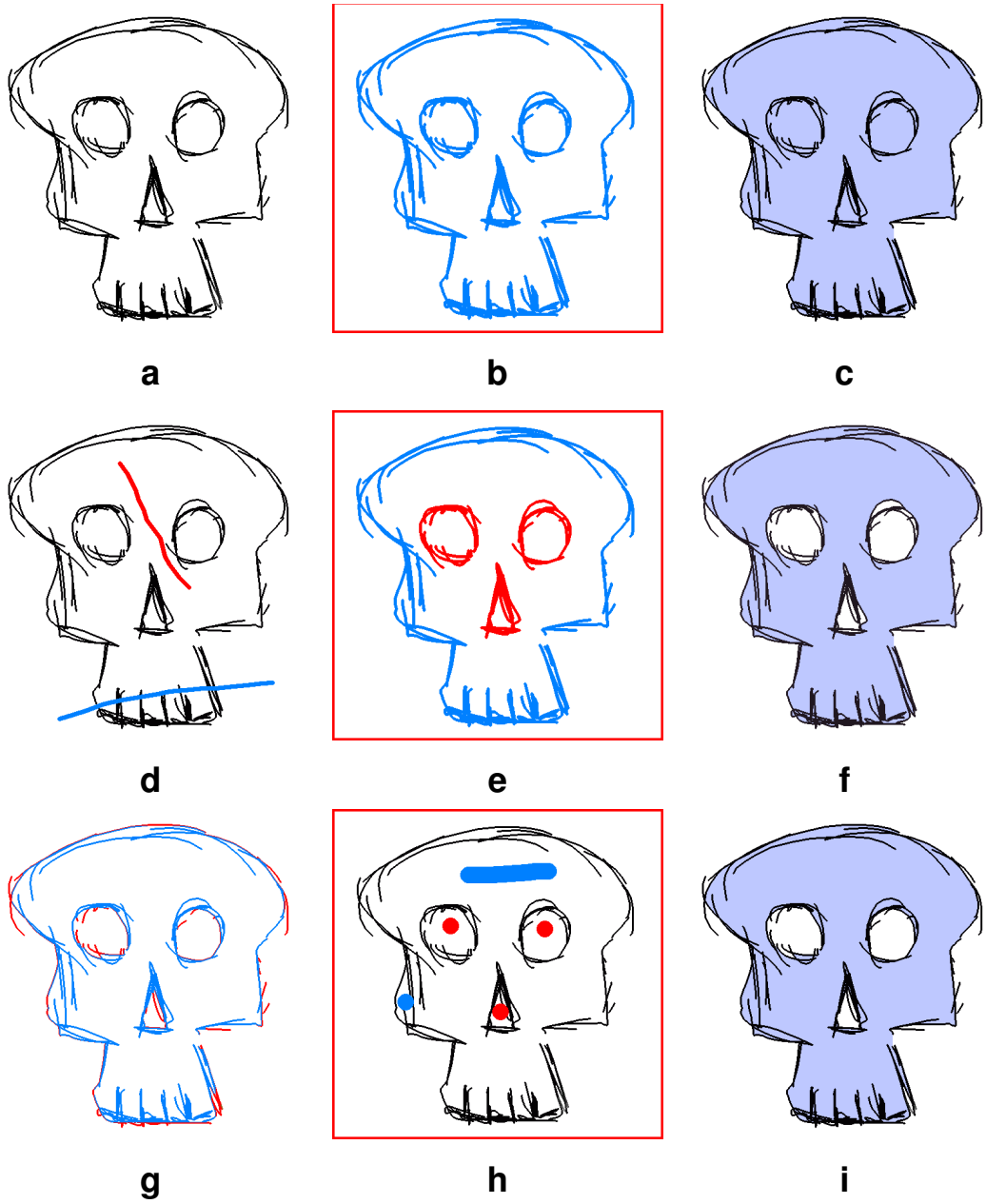

Figure 5.5: *Area mask computation: strokes of an input sketch (a) can be used as LazyBrush soft scribbles (b) to automatically fill the drawing (c). Additional Smart Scribbles (d) can be used to segment the strokes (e) for better control of the paint fill (f). Using the original LazyBrush algorithm (h) to paint the figure also produces a good result (i), however, the strokes cannot be classified based on the painting alone (g).*

The ability to easily label both strokes and areas empowers a large variety of applications. One can easily alter the individual drawing style for all the strokes that have the same label. It is also possible to accurately separate the different parts of a sketch, specify their depth ordering [Sykora et al., 2010] and then deform them independently using as-rigid-as-possible shape deformation techniques [Igarashi and Moscovich, 2005] (see Figure 5.6b). These operations can help, for instance in the context of image registration [Sykora et al., 2009a], to produce better alignment.

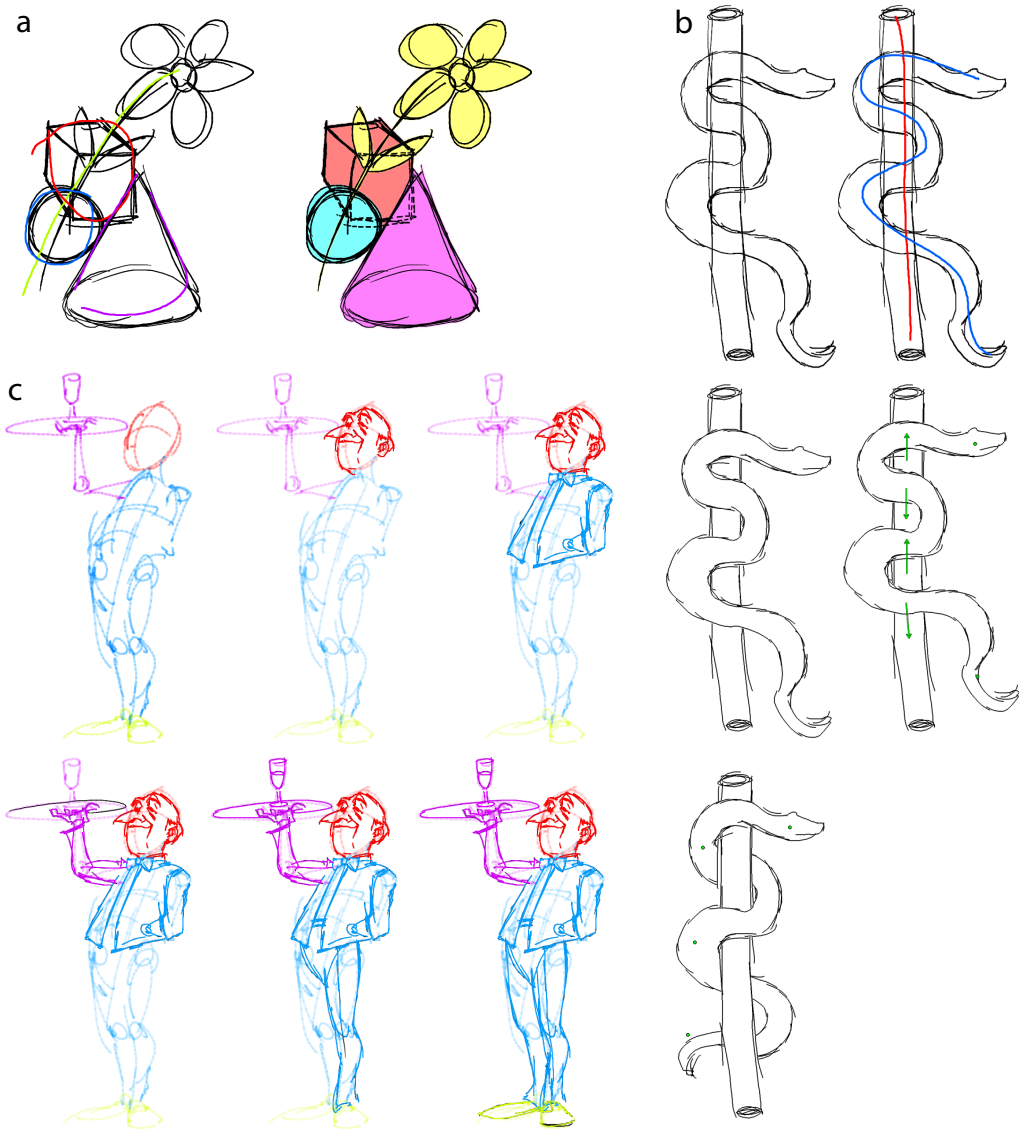

Figure 5.6: *Applications. Opaquing of the segmented clusters (a), ARAP deformation and opaquing with depth inequalities (b), on-the-fly labeling (c).*

When artists start a drawing, they typically begin with a simple, high-level sketch that depicts a set of of primitive shapes (see examples and references in [Gingold et al., 2009]) that are called volume or scaffold lines. If available, we can use these aiding structures as Smart Scribbles to segment the final detailed sketch (see Figure 5.6c). This would let the artist focus on drawing without having to switch between different brushes. As-rigid-as-possible deformation, for instance, could then instantly be used to correct the shape of semantically meaningful sketch regions.

5.5 Results

We demonstrate the effectiveness of our algorithm on a variety of input sketches. All results were generated using the parameters in Table 5.1.

Figure 5.7 shows a collection of simple input sketches and scribbles, together with the color-coded stroke labeling output by our system. These results show that desirable sketch segmentations can be obtained using very different scribbling strategies. We note that the input scribbles do not have to closely match the sketch in order for our algorithm to work well—approximate similarity in terms of position, orientation and curvature is sufficient.

Figs. 5.1 and 5.8 show results from more complex input sketches. To correctly segment these images, users typically start with rough, fast strokes, and then refine the output locally using slower, more accurate strokes. Our method robustly handles scenarios where strokes that are close together and almost parallel belong semantically to different regions (as shown on the waiter's legs and snake and pole example in Figure 5.8). In these cases, the time metric plays an important role in the labeling process.

Our framework does not require artists to draw the input sketches in any particular manner. It is possible that strokes representing the same region can be drawn at very different moments in time. This happens, for instance, when artists first draw silhouettes for the whole scene, and then proceed to refine the drawing. This can diminish the advantage of taking timing into account in the similarity metric. Correct segmentations can still be obtained, but more scribbles may be required. Alternatively the similarity metric can be adjusted to apply a smaller weight to the time parameter, or it can be removed as is done for the scribble metric.

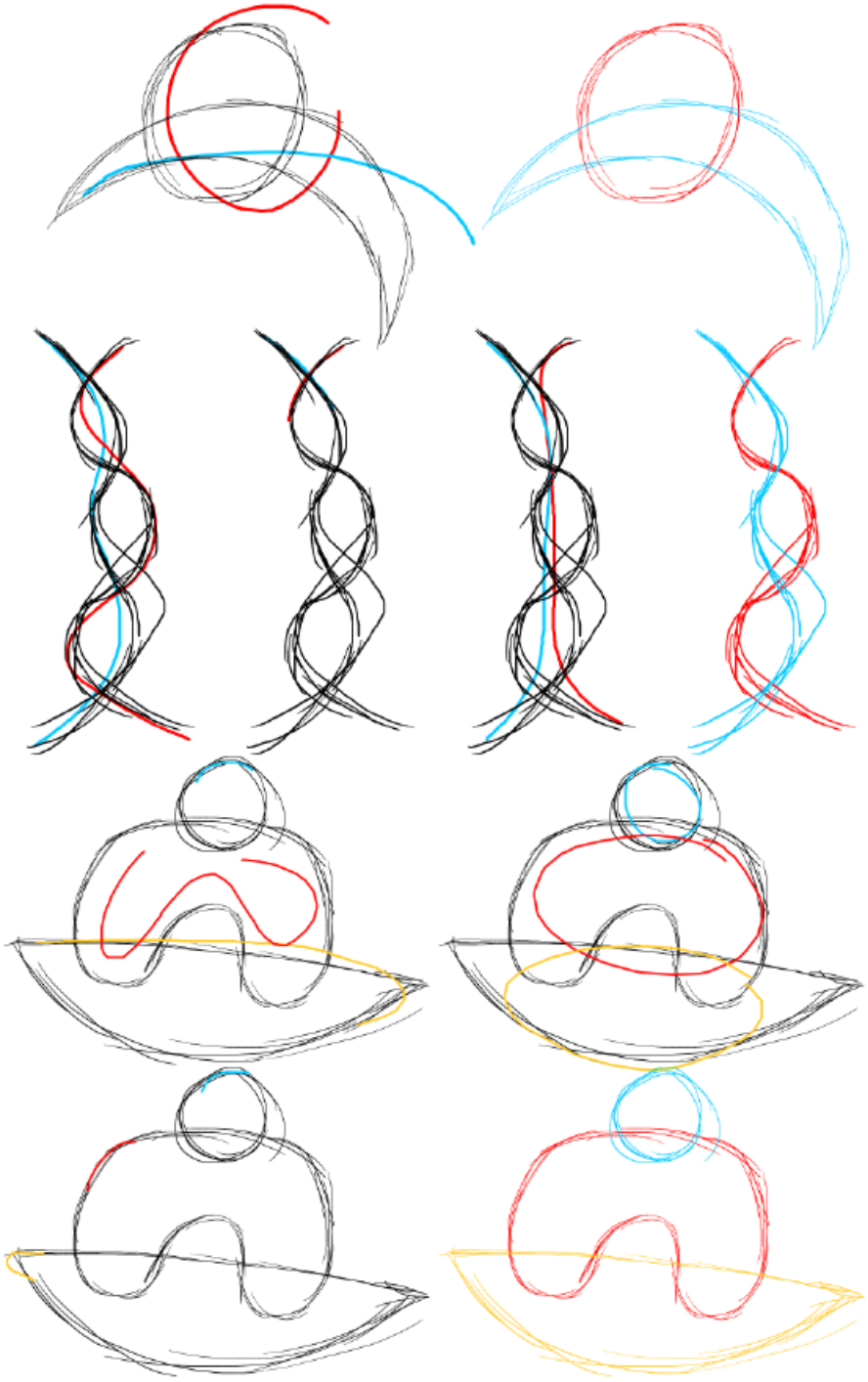

Figure 5.7: *Results for simple sketches: several different inputs can produce the same segmentation.*

Figure 5.8: *Example Results: For each example, the colored Smart Scribbles are shown on the input drawing and the adjacent image shows the resulting color-coded labelings.*

Parameter	Value	Unit
λ	4	
$\sigma_{prox\ smooth}$	100	px
$\sigma_{dir\ smooth}$	0.5	
$\sigma_{time\ smooth}$	1000	ms
$\sigma_{curv\ smooth}$	0.1	
$\sigma_{prox\ data}$	$[10\ ;\ 90]$	px
$\sigma_{dir\ data}$	0.1	
$\sigma_{curv\ data}$	0.25	
B	0.0001	
artboard width	1200	px
artboard heigth	1200	px

Table 5.1: *Parameter settings for the user study and all examples in this paper and the accompanying video.*

5.6 User Study Report

In order to test the efficiency and ease of use of our method, we conducted a user study comparing Smart Scribbles to our implementation of several commonly-used selection tools, namely point, box, and lasso (these tools are typically included in professional vector graphics software such as Adobe Illustrator or Inkscape). As shown in Figure 5.9, 35 participants took part in our user study (8 women and 27 men with ages ranging from 18 to 62). There were 5 artists, 9 hobbyists, and 21 people without drawing experience. None of them used Smart Scribbles before therefore all can be considered as novel users. Five had no experience with Adobe Illustrator or Inkscape, 14 had little experience, 15 were normal users, and 1 was an expert.

5.6.1 Learning Phase

The study began with a short tutorial introducing the selection tools. Here, users were allowed to test all four selection tools on a variety of simple drawings. There were examples to illustrate that there is no prescribed set of gestures for Smart Scribbles, the only recommendation was to try to roughly follow the shape and try to be as close as possible to the desired object. After this learning phase, the users were asked about their experience. The questions and answers are shown in Figure 5.10.

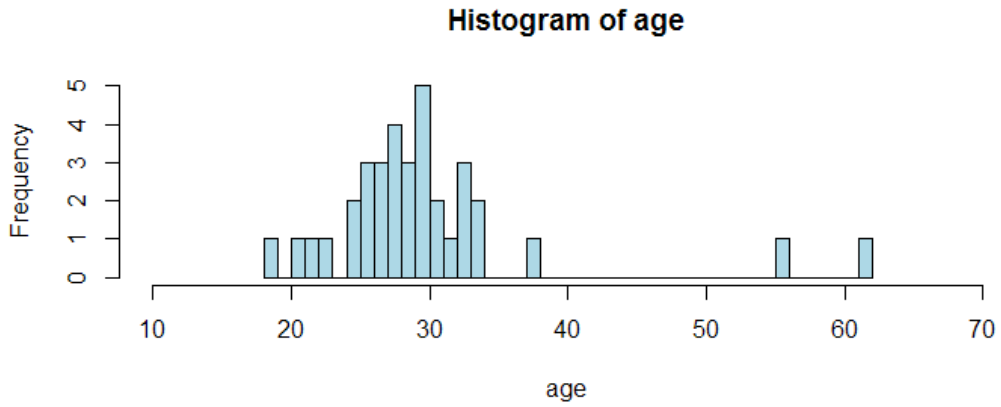

Figure 5.9: *Age histogram for the 35 participants of our user study.*

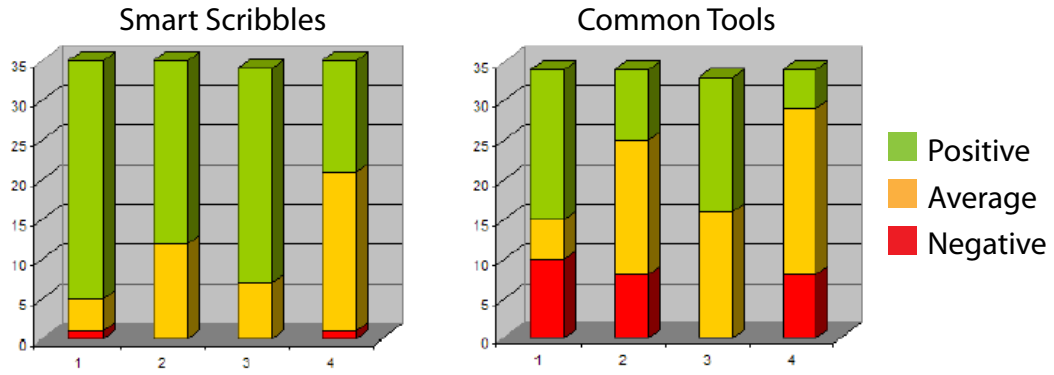

Figure 5.10: *Tools Evaluation. These plots show the questions and answers given after testing the tools in the first phase of the study. The numbers in the horizontal axis refers to the following questions:*
1. Did you like the interaction metaphor? (No, don't know, yes)
2. How good was this tool, or set of tools, for coloring complex drawings? (Badly suited, fair but not great, seems good)
3. Given enough time to practice, do you feel this is a tool, or set of tools, you could become efficient with? (no, maybe with effort, yes)
4. If yes, how long do you think it would take? (long time, some time, little time)

5.6.2 Comparison Phase

The participants were next shown pairs of identical drawings, one of which had already been labeled, while the other had not. Their task was to reproduce the stroke selection for the unlabeled drawing. In order to complete this task, the participants first used our Smart Scribbles (with parameters set according to Table 5.1) and then the common tools (point/box/lasso). This process was repeated with four different sets of drawings, for a total of eight labellings per participant. The order in which the eight scenarios were presented was random.

Participants were randomly divided into two groups. First initial group of 17 people worked on a selection of 4 simple drawings (skull, house, combo, and snake, see Figure 5.11) while second group (18 people) had 2 simple (skull and house) and 2 complex drawings (abstract and characters).

During the performance gain test our system measured interaction time and accuracy of the final labeling. Additionally, for the second group we measured mouse mileage. Mouse and keyboard were used to perform interactions with the system. There was a possibility to invoke undo/redo, zoom in/out and center viewpoint to the current position of the mouse cursor. In addition to that all participants were asked to respond to questions of which aim was to assess their subjective feeling about how these tools are suitable for the task.

Overall distributions of times and mouse mileage measured during the experiment are depicted in Figure 5.12. There is a notable performance gain when comparing Smart Scribbles to common tools ranging from 1.23x to 2.36x (median speed-up). Paired t-tests (see Table 5.2) have indicated that this gain is significant for 4 out of 6 drawings considering tight confidence level of 99.5%. Except couple of outliers the accuracy of labelling was typically close to 100% which indicates most of the participants were careful and tried to fulfill the task properly. Since the parameters of Smart Scribbles were fixed during the test we can also claim that a wide variety of users is able to produce target labeling within notably lower time as compared to common tools without necessity of tedious personalized parameter tuning.

The qualitative results of this phase are shown in Figure 5.13.

5.6.3 Locality Control Phase

When we asked participants whether they like interaction metaphor of Smart Scribbles 30 responded yes, 4 did not know, and 1 said no. For common

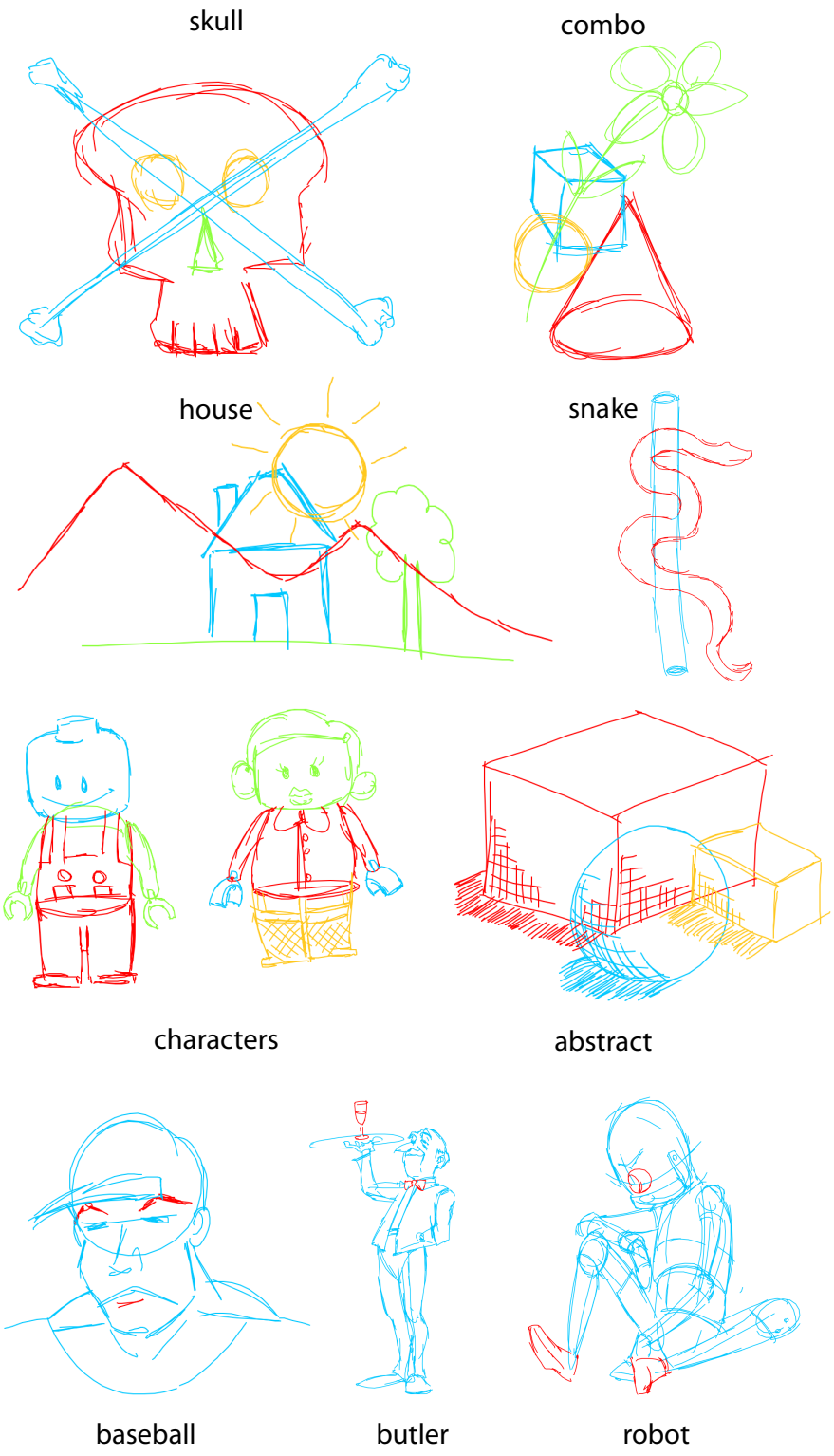

Figure 5.11: *Drawings and labeling used in the user study.*

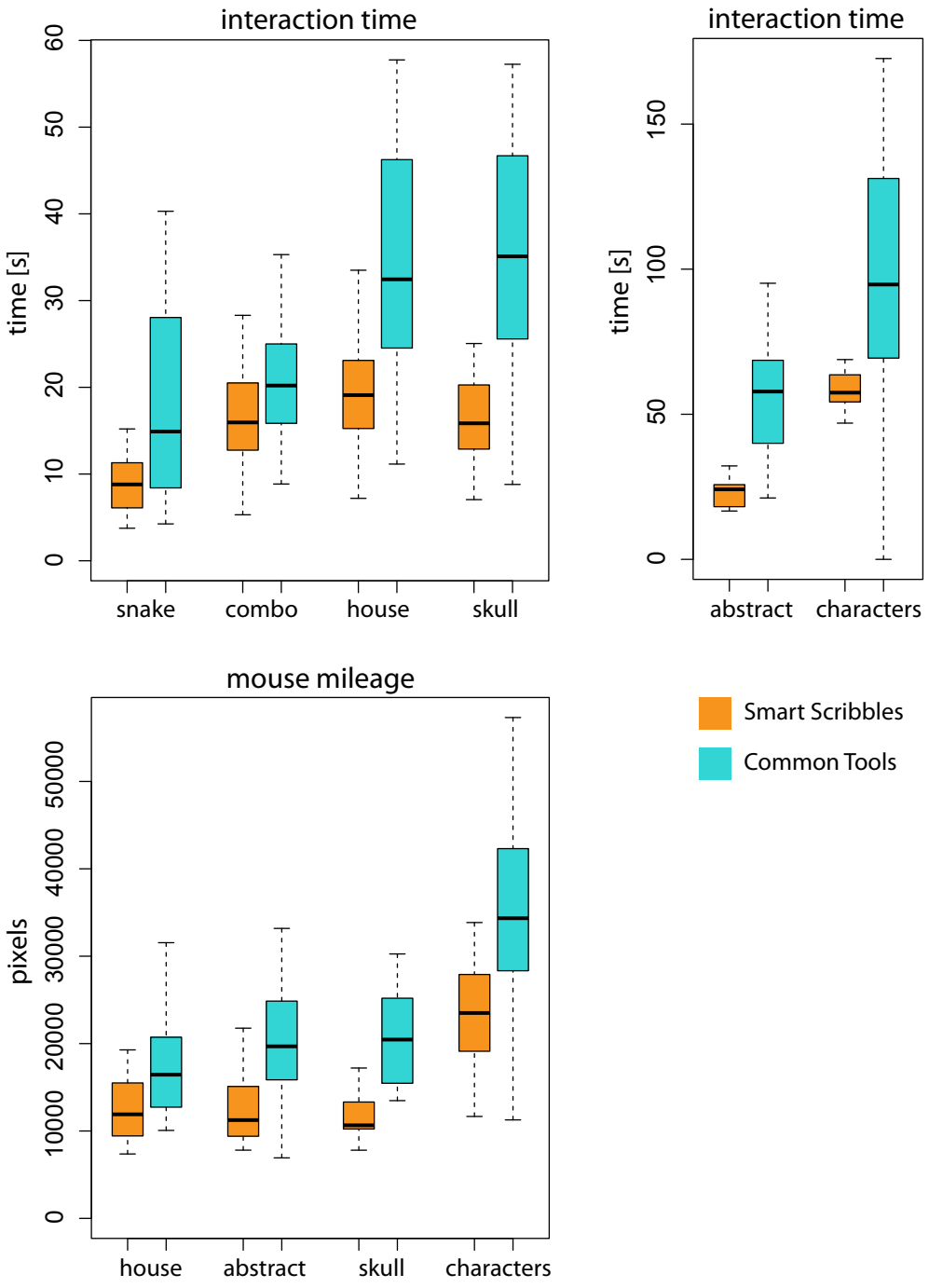

Figure 5.12: *Interaction times (top) and mouse mileage (bottom) of participants for different drawings using Smart Scribbles (orange) and common tools (blue).*

drawing	speed-up	$t(df)$	p-value
combo	1.23x	-1.8798	0.07847
snake	1.53x	-2.4807	0.02461
skull	1.83x	-8.3931	0.00000
house	1.85x	-5.2488	0.00001
abstract	2.36x	-5.5759	0.00003
characters	1.65x	-3.8896	0.00118

Table 5.2: *Median speed-ups and results of paired t-tests comparing times spent on labeling different drawings using Smart Scribbles and common tools.*

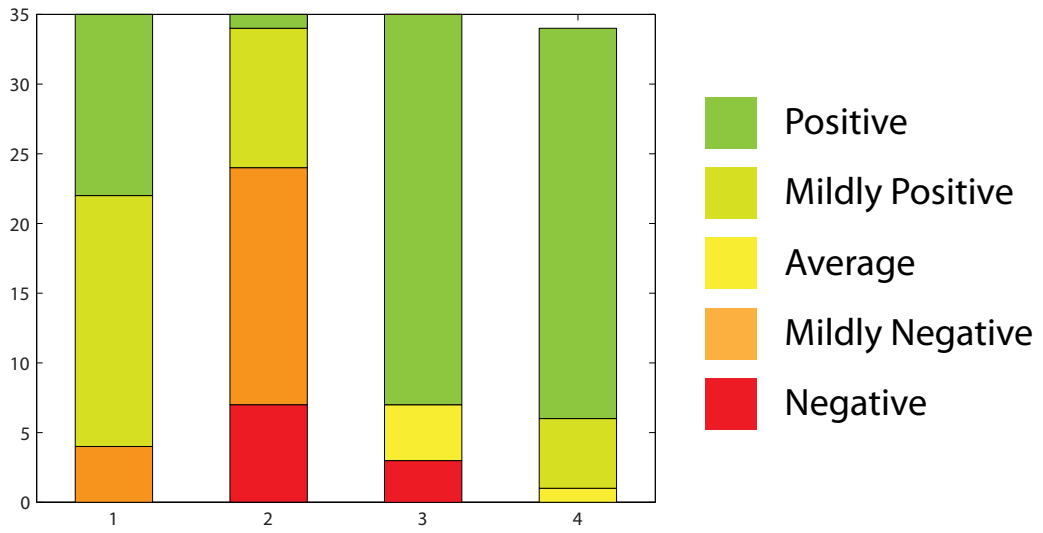

Figure 5.13: *Tools Comparison. This plot shows the questions and answers given after the performance tests in the second phase of the study. The numbers in the horizontal axis refers to the following questions:*
1. Did the scribbles let you solve the tasks efficiently? (No, sometimes, mostly, definitely)
2. Did the common tools let you solve the task efficiently? (No, sometimes, mostly, definitely)
3. Which tool did you prefer? (Common tools, no preference, scribbles)
4. Should scribbles be added to programs like illustrator, would it help people with their work? (No, I don't know, possibly, likely)

tools answers were: 19 yes, 5 did not know, and 10 said no. Moreover, 23 participant were convinced that Smart Scribbles are good tool for segmenting drawings with complex depth contrary to only 9 who preferred common tools. Based on these two questions a paired Wilcoxon signed rank test indicated the preference of Smart Scribbles over the common tools is significant with the confidence level of 99.5% ($V = 102$, $p = 0.00159$ and $V = 363$, $p = 0.00083$ respectively).

In addition to the performance improvement tests and tool preferences we also let the participants to experiment with continuous locality control (speed driven) and binary switching between local and global influence of Smart Scribbles. They were asked to reproduce labeling of details in three different drawings (baseball, butler, and robot, see Figure 5.11). In the first round participants controlled locality using mouse speed (linear relationship between speed and locality was used as proposed in [Accot and Zhai, 1997]). In the second round shift key was used to switch between two different modes (global as a default and local with key pressed).

After the test partecipants were asked four questions, as shown in Figure 5.14. First, they were asked whether they noticed the effect of the mouse speed on continuous locality control. Most participants (28) noticed the effect and only 7 were not sure. Then we asked whether they prefer continuous control using mouse speed or binary control with shift key. A majority (31) was for binary switching, only 2 people preferred speed-based control, and 2 have no preference. One of the reasons for this result might be the fact that users typically do not like to have some kind of time limitation when interacting with the computer. This observation is in line with another question we asked, i.e., whether they think speed is a good way to instruct the computer on what you want they do. Only 6 said yes, 27 participants answered: *No, don't want to be slow*, and 2 *No, it's hard to control*.

5.7 Limitations and Future Work

The selection of good parameters for the similarity terms and the energy function requires some effort. As can be observed in the parameter sensitivity graphs in Figure 5.15, the system is robust when parameters are perturbed one at a time. This is due to the correlation that exists between the similarity terms. However, it is possible that the perturbation of multiple parameters can lead to significant changes in the result. We also tested the usefulness of the information provided by the time of creation of strokes. After removing this information from the similarity terms, and after re-tuning the remaining

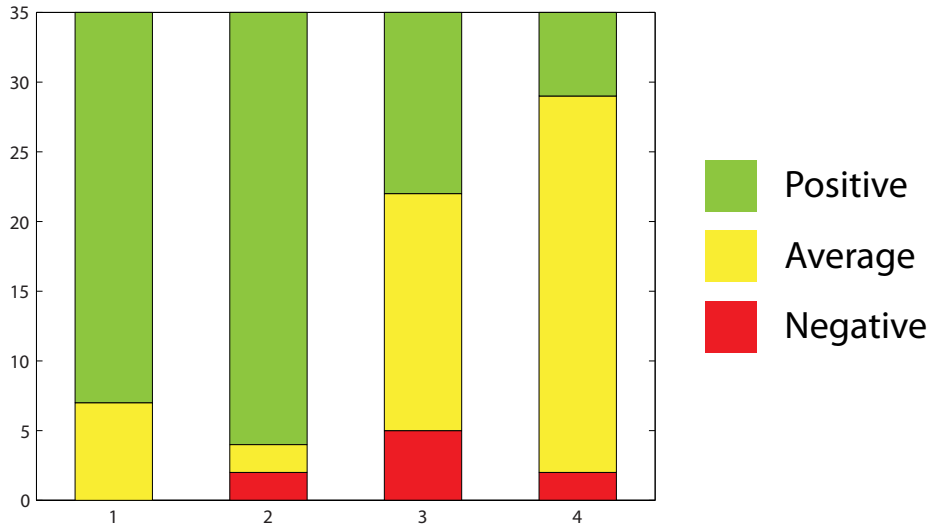

Figure 5.14: *Tools Comparison. This plot shows the questions and answers given after the performance tests in the second phase of the study. The numbers in the horizontal axis refers to the following questions:*
1. Have you noticed any difference between local and normal scribbles? (No, maybe, yes)
2. Did you prefer using speed-based or SHIFT-based scribbles? (Speed-based, no preference, Shift-based)
3. Given enough time to practice, do you think you could control speed well? (No, possible, likely)
4. Do you think using speed is a good way to instruct the computer on what you want to do? (No, hard to control, No, I don't want to be slow, yes)

parameters, we achieved the results shown in Figure 5.16. In our experience, disregarding this temporal information reduces the effectiveness of our method, as overlapping strokes require more effort to be separated. In the future we plan to develop a system that allows automatic parameter tuning based on a database of ground truth data.

Although we aim to produce accurate labeling with minimal user effort, detailed selection is necessary when ambiguities exist. One such ambiguous case occurs when an object is occluded by another object and parallel strokes from each are very close together or even overlaid. In this case, only the time constraint can provide a distinctive metric to obtain correct labeling. However, when the time is not available or when the user does not preserve temporal coherency of strokes, our approach requires additional user guidance.

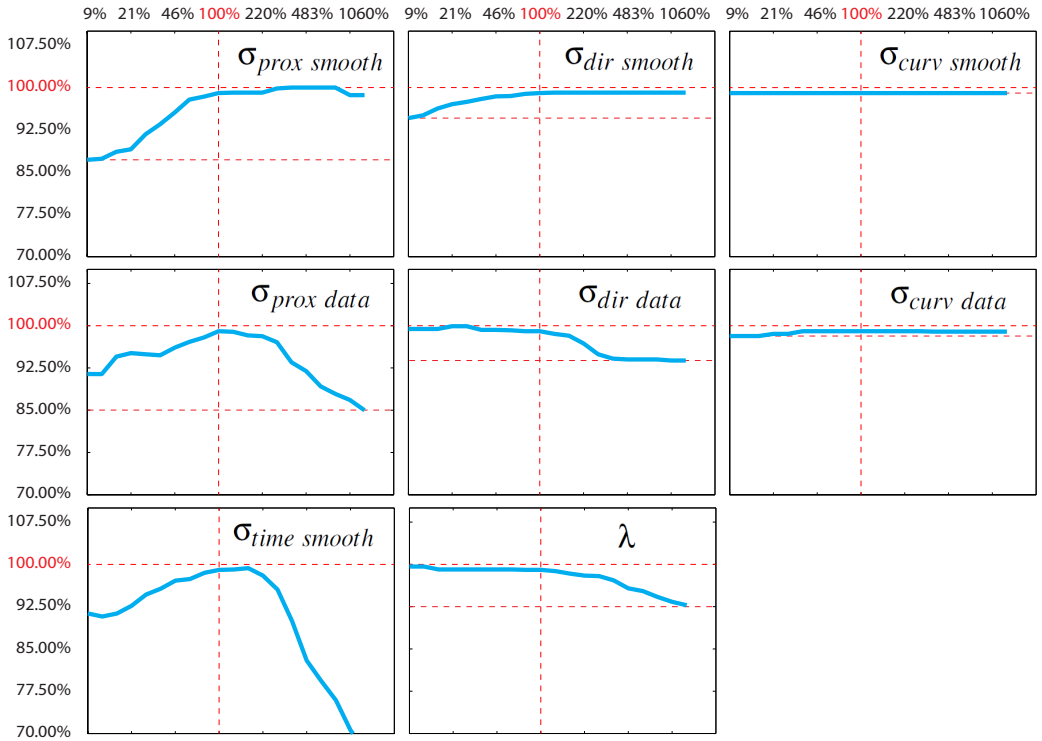

Figure 5.15: *Single parameter perturbation. Given a database of 8 drawings, each with 5 different sets of scribbles drawn to match a desired segmentation, we measure the the segmentation accuracy obtained with perturbations of the empirically chosen settings of Table 5.1. In each graph, the horizontal axis shows the multiplication factor for one of the parameters in exponential scale. The vertical axis shows how the correctness of the segmentation evolves. The graphs suggest slight changes to the parameters that would lead to a better fit for this database. Additionally, they show that the system is mostly sensitive to time and proximity information, while direction and curvature hold less influence.*

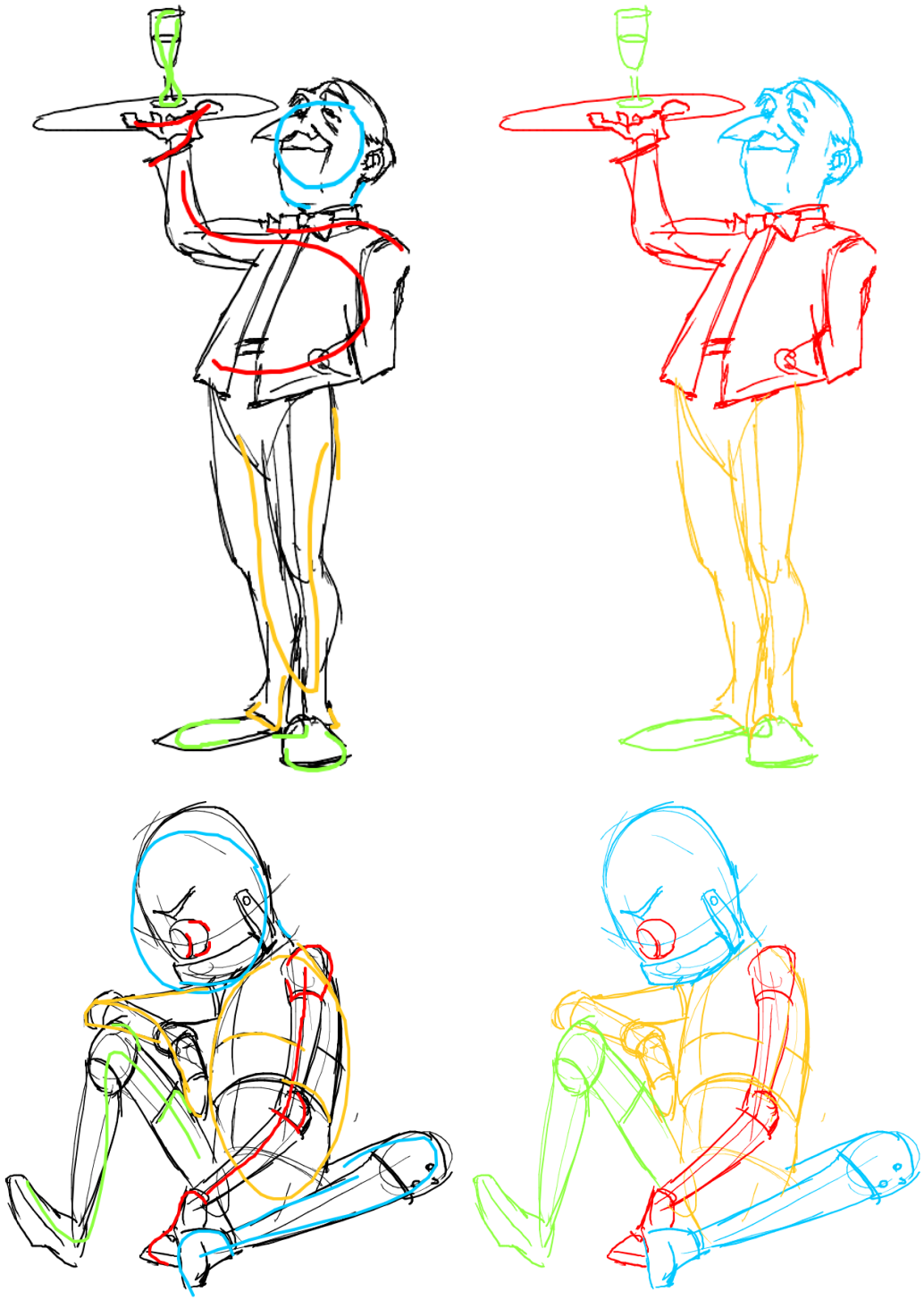

Figure 5.16: *No-Time-Test: this image shows how the system works when no temporal information is used. Notice how the cluttered regions require more scribbles to produce a proper segmentation.*

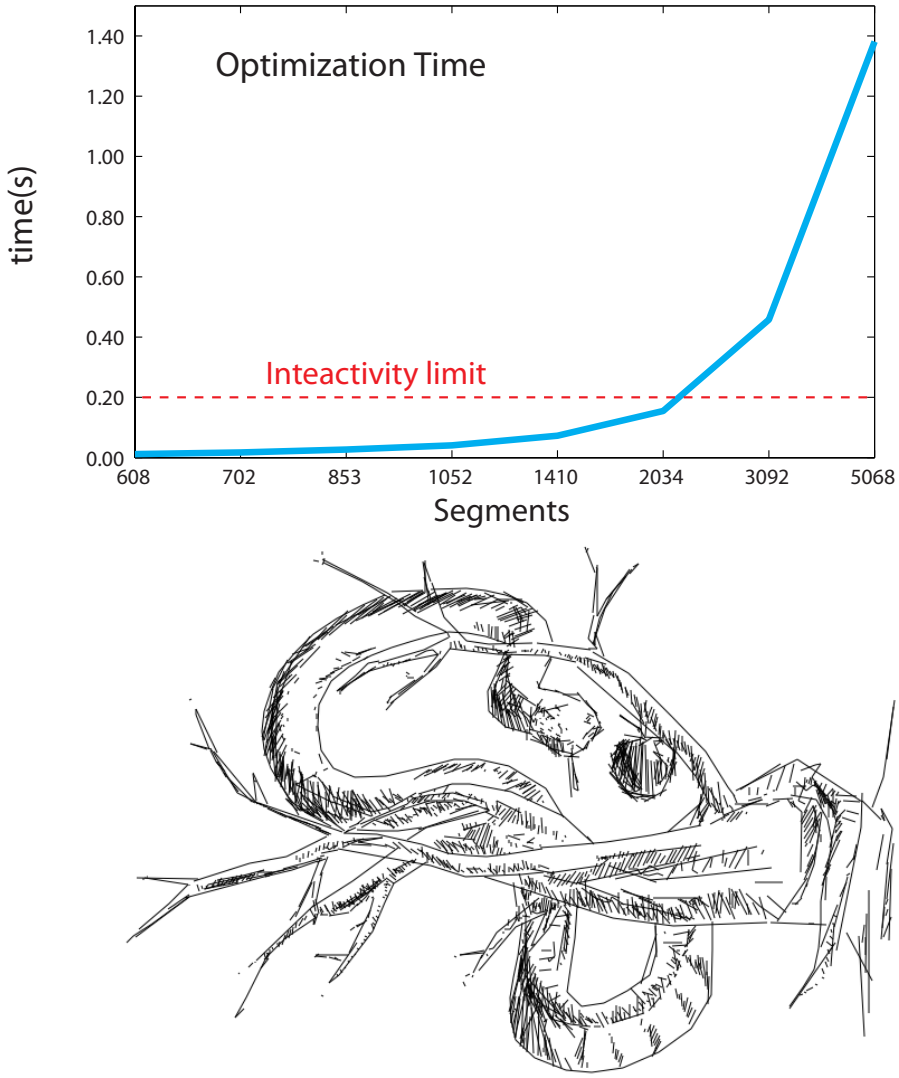

Figure 5.17: *Performance Limit. The graph on the top shows the computation time for the optimization in our implementation. The data was generated by progressively subsampling a complex drawing. Assuming a interactivity limit of* 0.2 *seconds, our implementation can optimize the labeling for up to 2000 segments. The corresponding subsampled drawing is shown on the bottom.*

The proposed graph-cut energy minimization strategy is generally very fast and produces the labeling at interactive rates. However, in the worst case, when a large number of stokes are close to each other as defined by our similarity measure, the number of edges in the graph can grow quadratically with the number of strokes and the computation can become prohibitively slow (see Figure 5.17). The problem can be alleviated by subsampling the strokes and processing disconnected components individually. Another problem is related to the non-polynomial complexity of the core max-flow algorithm we use [Boykov and Kolmogorov, 2004]. In certain situations where the cost of the minimal cut is very high and the graph topology is complex, the number of augmentation paths can grow very quickly along with the computation time. This issue can be solved by a recently proposed incremental breadth-first search solution [von Goldberg et al., 2011] that works in polynomial time and is typically notably faster than [Boykov and Kolmogorov, 2004].

The use of previously labeled drawings as Smart Scribbles offers another avenue for future work. They could be used on-the-fly to label new sketches as they are created, thus simplifying further interactions. This approach could be used, for instance, as an extension to the recently presented ShadowDraw system [Lee et al., 2011], where each sketch in the database would be augmented with Smart Scribbles. In this way the segmentation would be provided automatically as new drawings are created. A similar use case arises in the context of sketchy animations where image registration [Sykora et al., 2009a] can be used to transfer already labeled strokes and treat them as Smart Scribbles for the next frame. This can help, for instance, to better control temporal noise [Noris et al., 2011]. Smart Scribbles could also potentially be used to improve the accuracy of drawing simplification methods [Grabli et al., 2004, Barla et al., 2005, Shesh and Chen, 2008], as a typical problem with the current, fully automatic approaches is that they do not take into account any semantic information such as provided by our approach.

5.8 Conclusions

We have presented Smart Scribbles, a scribble-based interface for sketch-segmentation. Our method is fast, supports multi-label segmentation, and acts as an enabling technology for a variety of applications in the context of drawing, editing, and animation.

In the long term, we envision a next-generation drawing application, where drawing, editing, and animation are tightly integrated, and where the sim-

plicity of the interaction is the key. This work represents a step in this direction; a bridge between classic drawing and digital editing.

CHAPTER 6

Inbetweening

The generation of inbetween frames that interpolate a given set of key frames is a core component in the production of a 2D feature animation. As discussed in Chapter 2, this task is still very labor intensive. In this Chapter, we aim at reducing the cost of the inbetweening phase by offering an intuitive and effective interactive environment that automates inbetweening when possible while allowing the artist to guide, complement, or override the results. *T*ight inbetweens, which interpolate similar key frames, are particularly time-consuming and tedious to draw.

We focus on automating these high-precision and expensive portions of the process. We have designed a set of user-guided semi-automatic techniques that fit well with current practice and minimize the number of required artist-gestures. We present a novel technique for stroke interpolation from only two keys which combines a stroke motion constructed from logarithmic spiral vertex trajectories with a stroke deformation based on curvature averaging and twisting warps. We discuss our system in the context of a feature animation production environment and evaluate our approach with real production data.

This work was a collaboration with a second PhD Candidate, Dr. Brian Whited. The technical contributions of Section 6.3.4 are not to be considered as contributions of this PhD dissertation.

6.1 Introduction

Inbetweening is a cornerstone of the 2D animation pipeline. An animator produces drawings at *k*ey frames that capture the heart of the animated motion. The *i*nbetweener then draws the frames between each pair of keys that fill in the intermediate motion. A typical feature-length animation requires over one million inbetween drawings, with ratios of 3-4 inbetweens for each key [Johnston and Thomas, 1995].

The manual production of inbetween frames is both difficult and time-consuming, requiring many hours of intensive labor by skilled professionals. Furthermore, its cost is a significant part of the total production budget. Consequently, researchers have long sought an automatic solution. Despite a large body of work in this area spanning over four decades, no definitive solution exists. This is in part because the problem is ill-posed—there is no concise set of rules for producing high-quality inbetweens, especially in cases of complex motion and changing occlusion. The artist relies on his or her perspective of the 3D world in which the animated characters are embedded, as well as on an artistic vision for achieving the desired aesthetics. It would not be sufficient to derive the three dimensional world from the drawings, even if it were possible, as most often the characters have their own, undefined, laws of physics and deformation.

We approach the challenging task of automatic inbetweening with two important observations derived from studying the traditional animation process. First, *t*ight inbetweens are among the most laborious and time consuming to produce. Tight inbetweens are those drawn between two key frames that are very similar in shape. These inbetweens are tedious to draw because they require the greatest amount of technical precision and the least amount of artistic interpretation. By focusing on tight inbetweening, we pinpoint an expensive portion of the inbetweening process while restricting the problem to one that is more tractable from an algorithm standpoint. Our second observation is that an effective system should automate as much as possible while effortlessly deferring to artist control whenever needed. We never take the artists out of the loop. Instead we seek to make them more productive by automating the tedious and time consuming tasks so they can focus their efforts on areas where their creative talents and expertise are required.

Based on these observations, we have developed *BetweenIT*, an intuitive and effective interactive environment that automates tight inbetweening when possible while allowing the artist to guide, complement, or override the results. BetweenIT operates on vectorized pairs of consecutive key frames that were drawn in the program or obtained from scanned drawings. These key

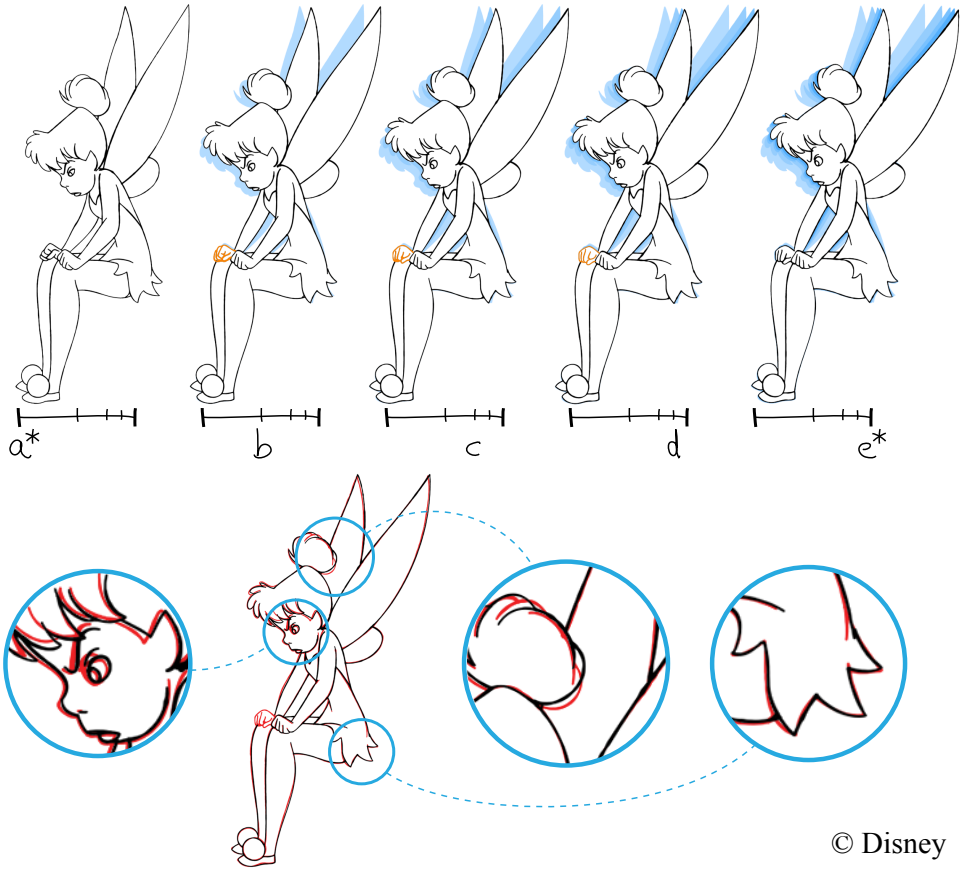

Figure 6.1: *Tinker Bell: Frames (a) and (e) are the given key frames and (b-d) (except for the right hand) were generated automatically. The "shadows" of previous frames are used to visualize their evolution. The topology of the right hand is not compatible across the two keys and therefore not suitable for automation: the portion indicated in red was hand-drawn.*

frames are segmented automatically into strokes. The strokes of one key frame are then matched with corresponding strokes of the subsequent key frame. Finally, interpolated strokes are generated for each requested inbetween frame. Most often the automatic tools provided by BetweenIT are sufficient to achieve the desired results. In the remaining cases, the aesthetic concerns of the artist as well as semantic ambiguities or topological discrepancies between consecutive frames will call for user intervention. BetweenIT offers an intuitive and effective workflow that allows such changes to be efficiently expressed. Supported options include: adjusting the shape of an individual stroke in a frame; adjusting the trajectory that a given vertex follows across frames; resolving ambiguities in the correspondence; and controlling

the behavior of occluded strokes that appear or disappear. All interactions can be specified by the artist graphically via a drawing interface.

6.2 Background

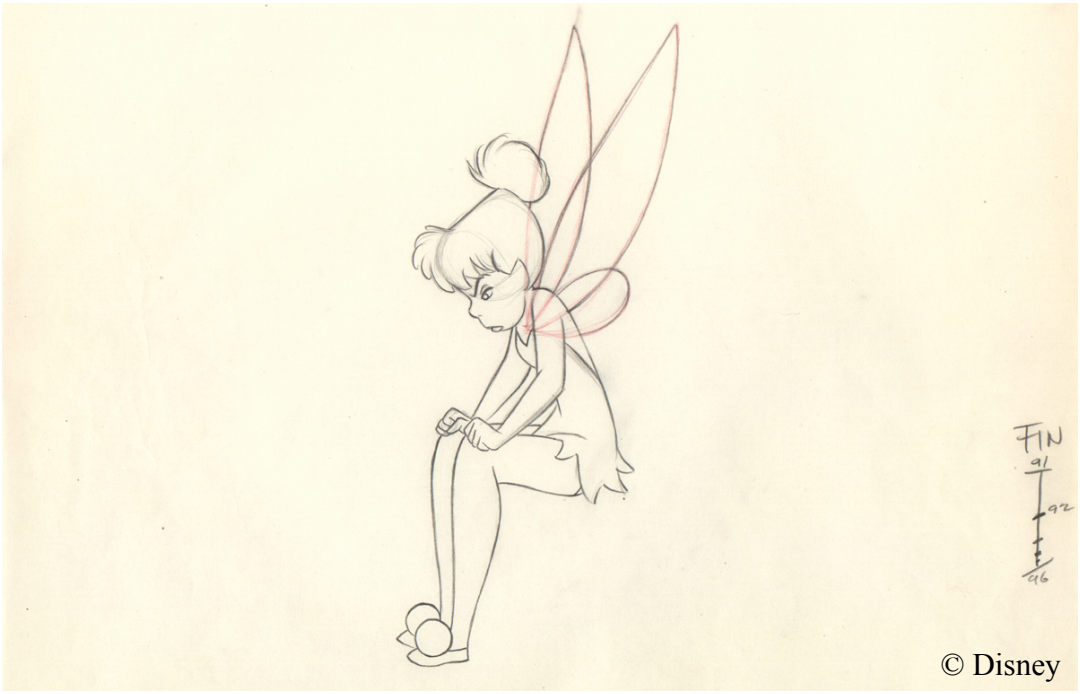

Figure 6.2: *Example archival image: The artist has indicated the occluded lines in red for reference. The associated timing chart is seen in the lower right.*

While some of the underlying mechanisms have benefited from the advent of digital technology, a 2D production today follows much the same basic workflow as traditional animation [Johnston and Thomas, 1995]. The process begins with a storyboard, which provides a visual representation of the story. In layout, the staging for each scene is designed, including establishing the setting, choosing and placing character and prop elements, and specifying camera motion and cuts.

Character and effects animation is then done in multiple stages. First, animators produce the subset of drawings that lay down the core of the action, the "keys." These extreme drawings are often "ruff" versions that capture the spirit, flow, and arcs of the animation. A "clean up" artist is responsible for taking the ruff drawings and producing clean lines that remain true to the original intent.

Each key drawing has one or more *timing charts* associated with it (e.g. see Figure 6.1). The animator uses these charts to specify how many drawings should be produced between keys, and at what intervals. It is the job of the inbetweening artist to draw the requested intermediate drawings to produce seamless motion.

When inbetweens are needed for dramatically different key drawings, advanced artistic skill and interpretation are required to produce satisfactory intermediate frames. In these cases, a further "breakdown" might be done, where the artist produces those frames within the range that present special drawing problems. *Tight* inbetweens require less artistic interpretation – but considerable technical skill to lay down the lines accurately.

The inbetweened frames represent a significant portion of the drawings, budget, and time for a production. A typical feature animation averages about four drawings per frame (for different characters, props, etc.). An 80 minute feature with 24 frames per second requires 460,800 drawings per production. If a quarter of these frames are done by the animators, then that leaves 345,600 inbetweened drawings. Since multiple drawings are often produced before arriving at the final version, the final tally can add up to over a million inbetweened drawings per full length production [Johnston and Thomas, 1995].

One critical consideration during animation and inbetweening is arcs of motion. Natural motion always follows an arc of some sort and therefore to achieve fluid and lifelike animation, artists must capture these natural arcs. This is one of the most challenging aspects of inbetweening since making a drawing on an arc is much more difficult than one placed linearly between the keys [Johnston and Thomas, 1995].

Our approach focuses on automating the process of inbetweening animated frames post clean-up for characters, props, and effects. We introduce a novel interpolation scheme which automatically derives natural arcs of motion from pairs of consecutive keys. Our system fits into the current 2D pipeline with minimal change.

6.2.1 Related work

The problem of automatic 2D inbetweening dates back more than forty years to the inception of computer graphics as a field of research [Miura et al., 1967]. To date, it has remained unsolved. The many challenges in automatic inbetweening have been well defined and include the

information loss and ambiguity inherent in a 2D projection of a 3D character, and the topology variations between key frames that result from changes in occlusion. An automated approach must address several challenges comprising: establishing correspondence in the presence of occlusions and topological changes [Catmull, 1978], computing stroke trajectories through time [Reeves, 1981], and avoiding the unnatural distortion that often occurs when there is a rotational component in an object's movement [Kochanek et al., 1982].

Many of the early methods are stroke-based. They require the user to identify a correspondence between the strokes of consecutive key frames (e.g., [Miura et al., 1967, Burtnyk and Wein, 1971, Levoy, 1977, Durand, 1991]) and do not handle occlusions or topological changes. Correspondences in Reeves's method [Reeves, 1981] are indicated by a collection of curves called moving points that are sketched by the user and connect the key frames to specify both trajectory and dynamics. A cubic metric space blending algorithm is used to find the positions of the remaining points, but requires heuristics to complete the patch network defined by the key frame strokes and moving point trajectories. Our method employs a semi-automatic correspondence algorithm that infers correspondence for the entire key from a small number of user gestures. Interpolation is automatic and designed to follow natural looking motion arcs. However, our method supports an optional trajectory redrawing interface similar to Reeves's moving points that allows customized trajectories to be specified by the artist.

Kort [Kort, 2002] presents a user-guided inbetweening system that identifies correspondences and computes inbetweens automatically but allows the user to correct undesired correspondences, trajectories, or timing. The method is restricted to the class of animations in which occlusions are resolved via an invariant layering. Our method is not restricted to layers and our workflow allows occlusion ambiguities to be resolved with help from the artist via a simple and efficient user interface.

Another layer-based approach [de Juan and Bodenheimer, 2005] targets the reuse of previously created 2D animations. Unlike the previously discussed methods, their system is image-based, rather than stroke-based. Characters in completed animation frames are segmented from the background and divided into layers. Inbetweening is accomplished via radial-basis function (RBF) interpolation of the layer contours [Carr et al., 2001], followed by morphing of the interior texture. The RBF interpolation requires the shapes to be properly aligned, and artifacts can result from misalignments. The MeshIK system of Sumner and colleagues [Sumner et al., 2005] includes a boundary-based interpolation scheme that does not require alignment, but it does not

address interior texture morphing. As shown by Baxter, Barla, and Anjyo [Baxter et al., 2009a], compatible embedding enables a maximally rigid interpolation [Alexa et al., 2000, Fu et al., 2005, Baxter et al., 2009b] that naturally blends both a shape's boundary and its interior texture. Similar methods have also been applied for cartoon capture and reuse [Bregler et al., 2002]. Nevertheless, methods based on texture blending are susceptible to blurring artifacts unless extreme care is taken to align internal texture features. Because we use a stroke-based algorithm, our method does not suffer from blurring problems. Sýkora and colleagues [Sykora et al., 2009a] use a similar imaged-based approach to register and morph cartoon frames. Their approach is designed to handle large pose changes and not the detailed interpolation of precise key strokes which we are interested in.

A final class of inbetweening systems employs skeleton-based methods for key frame interpolation. In the early work of Burtnyk and Wein [Burtnyk and Wein, 1976], key frame components are embedded in skeletal structures which are animated directly to deform the embedded shape. The inbetweening work of Melikohv et al.[Melikhov et al., 2004] uses a skeleton concept to deform the texture surrounding hand-drawn lines. They focus on simplicity so that non-artists can use the system. The vectorized strokes in our system encode the skeleton and the varying thickness of the drawn lines, preserving the original hand-drawn appearance.

Fekete and colleagues [Fekete et al., 1995] evaluate automatic inbetweening in the context of a paperless 2D animation system and identify several advantages and disadvantages. The advantage of reduced hand-drawn inbetweens and greater animation reuse is tempered by the need to specify correspondences, struggle with awkward timing specification, and build template models. Our BetweenIT system addresses all of these concerns: correspondences are mostly automatic and require little user guidance (Section 6.3), timing changes are specified in a natural fashion (Section 6.4), and templates are not required. BetweenIT leads to a quantifiably reduced amount of interaction compared to the number of strokes drawn when inbetweens are done by hand (Table 6.1).

6.3 Core Algorithms

This section details the automatic algorithms, which include graph building, matching, and interpolation. For tight inbetweening, these algorithms often produce adequate results automatically. The handling of more difficult cases that involve artist intervention is discussed in section 6.4 where we present

the entire workflow, including all user interaction. Our solutions were designed with the goal of an artist-friendly workflow: the focus is on algorithms that quickly produce a plausible result which can then be guided or edited by the artist.

6.3.1 Representation

A single drawing in our system is stored as a graph of strokes. The graph nodes, which we refer to as *salient points*, are the junctions at which strokes meet or end, and the graph edges are the strokes themselves. Each salient point contains pointers to its incident strokes, which are ordered counterclockwise around the salient point. A single stroke S is represented as a piecewise linear curve with n vertices. Each vertex i has an associated thickness measure T_i, for $i \in 1 \ldots n$.

6.3.2 Stroke matching

The input to the inbetweening process is a pair of consecutive keys which have been segmented into a stroke graph. The user initiates the automatic matching algorithm in one of two ways: by selecting a pair of corresponding strokes, one from each key frame, or by selecting a region via a selection lasso. When the lasso is used, the most similar pair of strokes within the selected region is chosen automatically. This initial pair provides the seed for our Correspondence Tracing Algorithm (CTA). In each key, starting from the selected stroke, the CTA traverses the graph in both directions in a depth-first order, respecting the circular order of the incident strokes around nodes. The traversal is performed simultaneously on both keys. The recursion stops when a incompatibility in the connectivity is detected or when two corresponding strokes are too dissimilar.

Our similarity metric computes an error E between two strokes A and B based on two different factors, E_L and E_A. E_L is the difference in arc-length between the two strokes. E_A is the area of the region bound by the two strokes when they are brought into endpoint alignment. E is then computed as $(E_A + E_L^2)/(L(A) + L(B))^2$ where L is arc-length, and compared against a constant T_E to determine if two strokes are similar. We have found that $T_E = 1$ works well and use it for all examples presented in this paper. Other similarity measures may be used [Baxter and ichi Anjyo, 2006, Veltkamp, 2001, Sebastian et al., 2003] that take into account proximity, orientation, curvature measures, and other shape descriptors. We have opted

for a simple approach which works well in practice and is fast. No metric is perfect in all cases and for tight inbetween situations, the choice of particular method is not a major factor.

The inbetweening operates on the subgraph of strokes matched by CTA. In the tight inbetweening problem, the two keys often have identical topology and similarly shaped strokes. In such situations, CTA establishes stroke-to-stroke correspondence for a whole connected component of the key. In more complex situations, especially where the keys have different topologies, the artist has the opportunity to provide further correspondence seeds, run CTA again, and compute inbetweens for missing portions of the keys. When the lasso selection is used, successive CTA traversals are executed, each time computing the best seed among the remaining strokes not covered by the previous traversals. This process continues until no stroke pairs with $E < T_E$ remain.

6.3.3 Vertex Correspondence

By default, smooth (subdivided) versions of the key strokes are re-sampled using uniform spacing in arc-length in order that matched strokes on each key have the same number of vertices. This re-sampling defines the default vertex-to-vertex correspondence. The artist alters this correspondence by identifying corresponding salient points, one on each stroke of a matching pair. These salient points are inserted as new endpoints, hence splitting the strokes. More complex correspondence algorithms could be substituted [Sebastian et al., 2003], but we have found in practice that having a simple,fast, and predictable technique is often more desirable.

6.3.4 Composite Interpolation

Our interpolation scheme is motivated by the basic principle of action arcs in 2D animation as described in the classic *Illusion of Life*: "most movements will describe an arc of some kind...One of the major problems for the inbetweeners is that it is much more difficult to make a drawing on an arc than one halfway inbetween...No one has ever found a way of insuring that the drawings will all be placed accurately on the arcs, even when experienced people are inbetweening the action, and it is one of the most basic requirements for the scene"[Johnston and Thomas, 1995]. In addition to this desire for overall natural motion arcs, inter-stroke continuity must be maintained. The approach also must be efficient enough to fit into an interactive framework.

To this end, we have developed a novel stroke interpolation scheme which combines a *stroke motion* constructed from logarithmic spirals with *stroke deformation* based on curvature averaging and twisting warps.

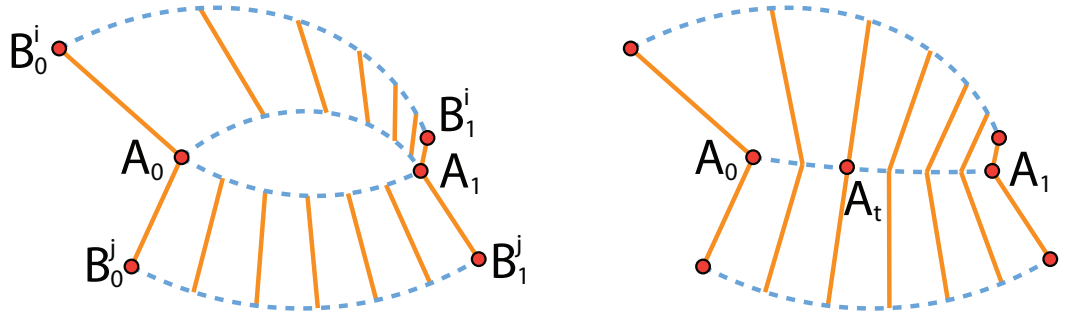

Figure 6.3: *Stroke motion: The left figure shows the two spiral paths resulting from calculating the spiral parameters individually for the strokes incident on A_0. On the right they have been computed as a weighted average of the parameters calculated for each incident stroke.*

Stroke Motion

Input strokes may be undergoing a motion that involves not only translation, but also rotation and scaling. Hence, we have developed a solution that provides natural, arched motions when the corresponding key features are congruent (related by a rigid body motion) or similar (related by an affinity that is a combination of rotation, translation and uniform scaling). The stroke motion is a map between time t (which evolves from 0.0 to 1.0), and a set of similarities. A similarity may be defined by two points and their images. Therefore, we define a stroke motion by the movement of its endpoints.

In choosing the particular type of trajectory for our endpoint motion, we rejected circular arcs since they are too limiting, and rejected cubic Bezier and bi-arc segments since they sometimes produce unacceptable inflection points or self-intersections. Instead, we opted for logarithmic spirals. A logarithmic spiral is a pleasing curve found in nature (e.g. shells, weather systems, spiral galaxies) [Thompson, 1992]. For our application, logarithmic spirals yield better results than Archimedean spirals, especially when used to approximate screen motions of shapes that approach or move away from the viewpoint and hence grow or shrink in a non-linear manner due to perspective (Figure 6.3 left-top).

Given an initial position A_0 and final position A_1 of an endpoint A, we compute its position A_t at time t using a logarithmic spiral motion $l(t)$ which is the combination of a rotation $r(\alpha)$ by angle α and a uniform scaling $s(\rho)$ by a factor ρ, both with respect to a fixed point (spiral center) F. We need to consider all strokes incident on A when computing the appropriate values for α and ρ. The process involves the following steps:

1. A weight is assigned to each incident stroke S^i based on its relative arc-length: We compute the sum L^i of the arc-lengths of the stroke at time $t = 0$ and at $t = 1$. The weight w^i is set as follows, where L_T is the total of all L^i:
$$L_T = \sum_i (L(S_0^i) + L(S_1^i)), \quad w^i = \frac{L^i}{L_T}. \tag{6.1}$$

2. Each incident stroke S^i, has one endpoint with initial position A_0 and final position A_1. Let B_0^i, B_1^i be the position of the remaining endpoint. We compute the angle α^i between $\vec{A_0B_0^i}$ and $\vec{A_1B_1^i}$ and the scale factor $\rho^i = \frac{||\vec{A_1B_1^i}||}{||\vec{A_0B_0^i}||}$.

3. The final angle and scale factor are a weighted average: $\alpha = \sum w^i \alpha^i$ and $\rho = \sum w^i \rho^i$.

4. The fixed point F is computed by solving the linear system
$$F\vec{A}_1 = r(\alpha)s(\rho)F\vec{A}_0 \tag{6.2}$$
In the case of pure translation, the determinant of the system is zero (and the fixed point is at infinity). We handle this case accordingly.

The position A_t at time t of endpoint A from the initial key moved by this logarithmic spiral motion is
$$A_t = l(t)A_0 = F + r(\alpha t)s(\rho^t)(F\vec{A}_0) \tag{6.3}$$

This process is carried out for each endpoint (see Figure 6.3 right). For any given stroke, the stroke motion is then defined by the motions of its endpoints.

Stroke Deformation

The stroke motion described above produces the positions A_t and B_t of the two endpoints A, B, of a stroke at time t. We must now compute the evolving shape of the stroke that connects these endpoints. The morph should

smoothly blend between the key stroke shapes while preserving tangent continuity between adjacent strokes that are smoothly connected in both keys. We achieve this with a three-step deformation involving *intrinsic shape interpolation*, *curve fitting*, and a *tangent aligning warp*.

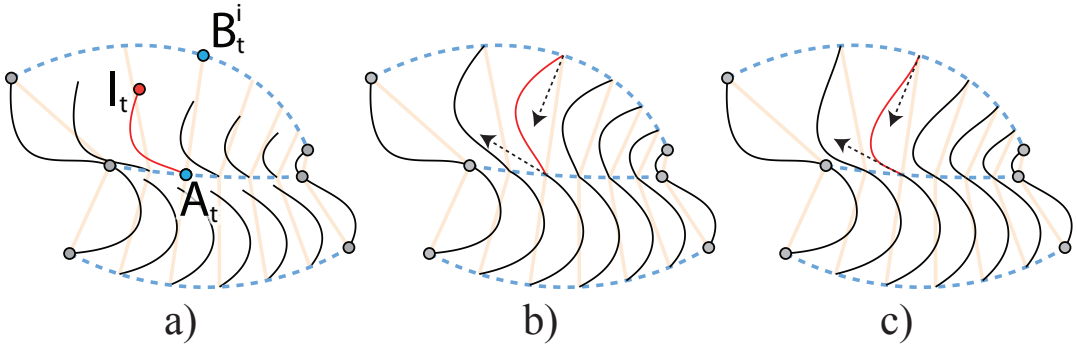

Figure 6.4: *Stroke deformation: (a)* ***I****ntrinsic shape interpolation. (b)* ***Curve Fitting****: transforms I to align the position I_t with B_t^i. Note the discontinuity at the endpoint where the red stroke meets the black one. (c)* ***Tangent Alignment warp****: enforces continuity at endpoints.*

Intrinsic Shape Interpolation: Each stroke is represented in terms of its edge lengths, vertex angles and vertex thicknesses. The shape of a stroke at time t is constructed from a linear interpolation of this intrinsic description as proposed in [Sederberg and Greenwood, 1992], which is a discrete version of curvature interpolation [Surazhsky and Elber, 2002]. Note that if we start the construction at A_t, the resulting interpolated curve I ends at some position I_t, which does not necessarily coincide with B_t (Fig.6.4 a). We avoid using the optimization described in Sederberg et al. [Sederberg and Greenwood, 1992] to fix the discrepancy as it can produce unacceptable results, as outlined by the authors. We also interpolate the thickness parameter at each vertex linearly between keyframes.

Curve Fitting: Each stroke I is then rotated by by angle $\angle(\vec{A_tB_t})(\vec{A_tI_t})$ and scaled uniformly by by $||\vec{A_tB_t}||/||\vec{A_tI_t}||$ about A_t so that I'_t coincides with B_t (Fig. 6.4 b). These first two steps produce pleasing morphs that preserve the continuity of the stroke graph but fail to preserve smoothness of connections between adjacent strokes. This shortcoming is addressed by the next step.

Tangent Aligning Warp

In order to be consistent with the logarithmic spiral, we assume that the angle of the tangent at an endpoint varies exponentially over time. Therefore, we

compute the desired endpoint tangent direction of an interpolated stroke by linearly interpolating the polar representation of the tangent in the two keys. Each interpolated stroke is then warped (Fig. 6.4 c) so that it matches the desired tangents at its endpoints. To do this, we use a variation of the Twister warp [Llamas, 2003]. First, assume the angles between the endpoint tangents of I' and the desired tangents at A_t and B_t are a and b. Assume that L is the total arc length of the stroke I'. For every vertex position P of I', L_P is the arc-length from P to A_t along I'. The Twister warp moves P to $P + (P_A - P) + (P_B - P)$, which simplifies to $P_A + (P_B - P)$, where P_A is the image of P by a rotation of angle r_A around A_t and P_B is the image of P by rotation of angle r_B around B_t, where r_A and r_B are defined as follows:

$$r_A = a\cos^2((\frac{\pi}{2})(\frac{L_P}{L})), \quad r_B = b\cos^2((\frac{\pi}{2})(\frac{L - L_P}{L})) \tag{6.4}$$

The proposed interpolation scheme produces aesthetically pleasing results that ensure tangent continuity across smooth junctions between strokes and preservation of features that are present in both keys. In addition, it is an approach suitable for interactive use, since it does not require numeric iterations or other optimizations.

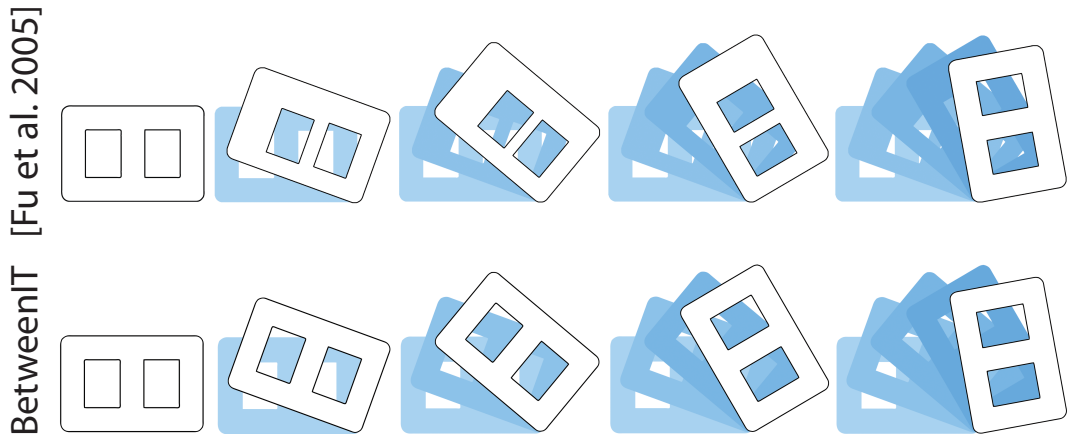

Figure 6.5: *The work of Fu and colleagues [Fu et al., 2005] resolves translation invariance via linear interpolation of at least one vertex on each connected component which can lead to uncoordinated movement. Our method ensures that all connected components move together in a coordinated fashion.*

6.3.5 Comparison with prior art

In developing our interpolation algorithm, we have evaluated several existing techniques. Leading contemporary methods formulate the interpolation as an optimization problem over the entire drawing that maximizes the rigidity of the inbetween shapes to avoid scaling and shearing artifacts [Fu et al., 2005, Surazhsky and Elber, 2002, Alexa et al., 2000]. Since the optimization is formulated in the differential domain, the result is invariant to global translation. Consequentially, these methods cannot be applied as-is because the input to our application is made up of multiple connected components which must move as a semantic unit when the translation invariance is resolved—an issue not addressed by existing work. Figure 6.5 demonstrates this problem using our implementation of [Fu et al., 2005]. Using our approach, when different objects of one key frame are all moved by the same rigid body motion or affinity to new poses in another key frame, then their trajectories are consistent through inbetweening(Fig. 6.5, 6.6. If the components are transformed each by a different affinity, there is no fundamental reason to believe that they need to follow a consistent path. In either scenario, our workflow allows the artist explicitly specify the path, if desired.

The most similar work to BetweenIT is that of Baxter and Ichi [Baxter and ichi Anjyo, 2006], which describes a system for computing an N-way interpolation of several input keyframes at the stroke level. Correspondences are computed automatically and editable by the user when mismatches occur. The major difference is that they treat lines individually, whereas our technique utilizes a graph structure which preserves connectivity in the inbetweens.

Our logarithmic spirals ensure that all portions of the drawing (including separate connected components) move in a coordinated fashion while respecting the important principle of motion arcs from traditional hand-drawn animation. Our three-step deformation method deforms the stroke interiors while ensuring continuity at endpoints. All steps execute interactively from start to finish in contrast to the aforementioned algorithms which require a precomputation step.

6.4 Workflow

We have developed a natural workflow for user-guided inbetweening in which various interactive operations allow the user to optionally guide and/or override the algorithmic tools described in the previous section. As

with other parts of the system, the interface design was driven by conversations with artists. The system is moded, with keyboard shortcuts to switch between modes (correspondence editing, salient editing, trajectory editing, occlusion editing, breakdown editing).

The most notable aspect of the GUI is the interaction metaphors. The artists favor stroke-based interaction over clicking, and thus we have designed interactions within BetweenIT around the concept of *guide strokes*. The semantics of a guide stroke, and hence the resulting action, depends on the current mode and are described in more detail below. These guide strokes offer an artist-friendly alternative to clicking on pairs of strokes or salient points for the different operations, since clicking is often inconvenient when using a pen and tablet input device. It should be noted that all interactions provide instant feedback to the artist with the exception of the occlusion editing, which can incur a few seconds for complex drawings due to its current naïve quadratic implementation.

Correspondence Editing: If not satisfied with the output of the automated CTA stroke matching, the user can trim or extend the set of matched stroke pairs. To remove strokes from the matched set, the user simply draws a guide stroke along them on one of the keys. To extend the matched set, the user draws a guide stroke, providing a new seed for the next round of CTA. This new CTA traversal will not override previous inbetweens and will stop once a stroke with an existing correspondence is found.

Salient Point Editing: To fine tune the correspondence, the user can specify additional salient points indirectly through guide strokes. Assuming the strokes from the adjacent keys are displayed in the same view, the user draws a guide stroke that passes through (approximately) the matching stroke pair. This action splits each stroke by inserting a new salient point. The salient point on the first stroke is the point closest to the starting point of the guide stroke and the salient point on the second stroke is computed similarly. The initial two strokes are thus broken into four. The same guide stroke serves to define the trajectory of vertex corresponding to the salient point as it moves between the keys.

The user may also desire to merge adjacent strokes, thereby deleting a salient point. This operation is useful when a stroke in one frame should correspond to two adjacent strokes in the other key frame which have been split by a junction with another stroke. This operation is performed by connecting two adjacent strokes with a guide stroke.

Timing Specification: In practice, the timing for a set of inbetweens is specified prior to the inbetweening step and loaded in with the input key draw-

ings. For total flexibility, however, the user may adjust the time parameterization before and after inbetweens have been computed. If they are already computed, they will update on the fly for immediate feedback.

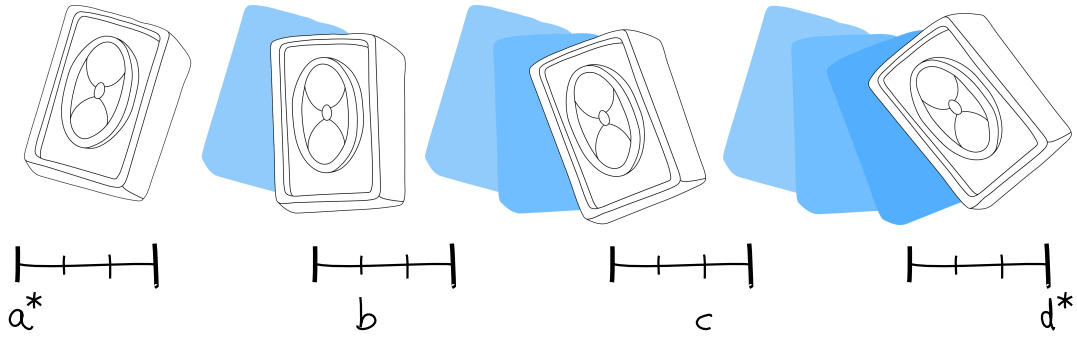

Figure 6.6: *Speaker: Frames (a) and (d) are the given key frames and (b-c) are generated inbetweens.*

Trajectory Drawing: Any of the automatically computed vertex trajectories may be edited as follows. The user draws a guide stroke to indicate the new, desired path. The ends of the trajectory are fixed, as they are specified by the key frames. It would be too restrictive, however, to require that the guide stroke coincide with the initial and final vertex positions. In cases where the drawn path does not match, the guide stroke is automatically retrofit through a similarity transformation (translation, rotation, and uniform scaling) to meet the end constraints. The adjacent inbetweens that are affected by such a change are recomputed immediately. When in playback mode, the result is reflected in the animation in real-time, giving instant feedback and allowing the user to redraw the trajectory repeatedly until the desired result is obtained.

Occluded Lines Drawing: In situations where a stroke is partially or totally occluded in one key frame, but not in the next or previous key frame, the graph connectivity may be locally incompatible. The user may extend a stroke past its occlusion point by drawing a guide stroke to indicate the occluded portion of the stroke.

The added stroke is then intersected with other strokes in the same frame, creating substrokes. Each substroke has an independent visibility toggle. By default, the substrokes toggle visibility at each intersection, which works well in the common simple case, such as in Fig. 6.9. When more control is necessary, the user may select specific substrokes to manually toggle the visibility. The visibility status is transferred automatically to subsequent correspond-

ing substrokes through time. For even further flexibility, the produced inbetween substrokes may also be toggled individually with a tap of the pen on the stroke.

Breakdown Insertion: When two key frames or subsets of key frames are too dissimilar to be classified as “tight," the user has the option of inserting an additional “breakdown” key frame for either the entire frame, or just the subset that is not tight. The user simply draws the strokes of the inbetween and then treats them as key strokes. In this way our system allows the user to produce manual stroke-level breakdowns only where needed.

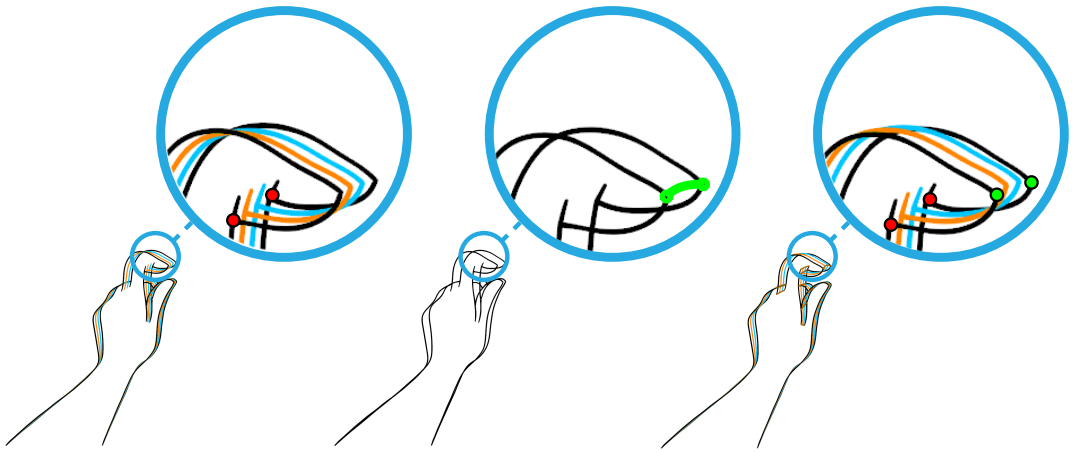

Figure 6.7: *Hand – User interaction: A trajectory is specified for the finger tip. The arc and resulting salient points are shown in green. The original automatically generated salient points are shown in red.*

6.5 Results

The goal of BetweenIT is to automate the production of inbetweens where possible. As the level of complexity in the drawings increases, we support intuitive and direct input from the artist to guide the automation. We expect the level of interaction or “touch time”[Catmull, 1978] to reflect the complexity of the drawing problems presented by the keys. The following examples are sample results of utilizing BetweenIT on real production data. A timeline noting the relative position of the frame is shown beneath each example. Keys are denoted with an asterisk, all other frames were generated by BetweenIT. In each example, the previous frame(s) in the sequence are indicated as a “shadow” to show the progression of the interpolation.

In the following, we present results for the purpose of evaluating the usability of BetweenIT and the quality of the inbetweens it produces. First, we show a collection of inbetweens generated by BetweenIT given archival key frames. We compare the automatically generated results visually to the hand-drawn inbetweens from the original artwork. We also present sample results from a artist evaluation of BetweenIT by the effects department.

Quantitatively we measure performance in terms of stroke count as well as time, where possible. All of the examples shown produced automated results in roughly 1 second on average. In cases where the user desires, or is required, to guide the system, he/she draws a series of "guide strokes" and/or indicates edits with mouse clicks, as described in Section 6.4. In the following we use interaction count (the sum of the guide strokes and input mouse clicks) versus the total strokes in the frame as a rough measure of efficiency gain. For example, if each of a pair of keys has n strokes and there are k inbetweened frames, then the artist would have to hand draw kn strokes to create the inbetweens. If the system produces fully automated results that are acceptable, that is a 100% gain in efficiency. If the user needs to perform i interactions to guide the system, the gain is $\frac{kn-i}{kn}$. Also note that even when this measure reported 0% gains, the use of BetweenIT may still save time, since the guide strokes need not be precise and may be drawn faster than the strokes that make up the final frame. Table 6.1 summarizes the results.

Archival examples: Figure 6.6 illustrates an example of pure rigid body motion. In this case, the results shown were produced fully automatically.

In Figure 6.7 we illustrate a next possible level of interaction. Here the automatic technique has produced a reasonable solution, but the user has chosen to slightly alter the trajectory of the tip of the finger with a single drawn arc. The inbetween is automatically updated after the arc is drawn, giving the user interactive feedback. In addition to specifying the arc, this interaction transparently produces additional salient points at the tip of the finger.

The Goofy hand example in Figure 6.10 presents a more challenging topological problem. For example, the automatic algorithm was unable to correctly establish correspondence for all strokes on the back of the hand. The desired result was achieved after the user manually specified 3 correspondences to guide the matching. The rightmost image in Figure 6.10 is a visualization of the automatically generated trajectories, illustrating the smooth arcs generated by the interpolation algorithm.

The duck sequence in Figure 6.8 shows two successive frame ranges. Note that unlike the previous examples the specified timing is not linear. Figure

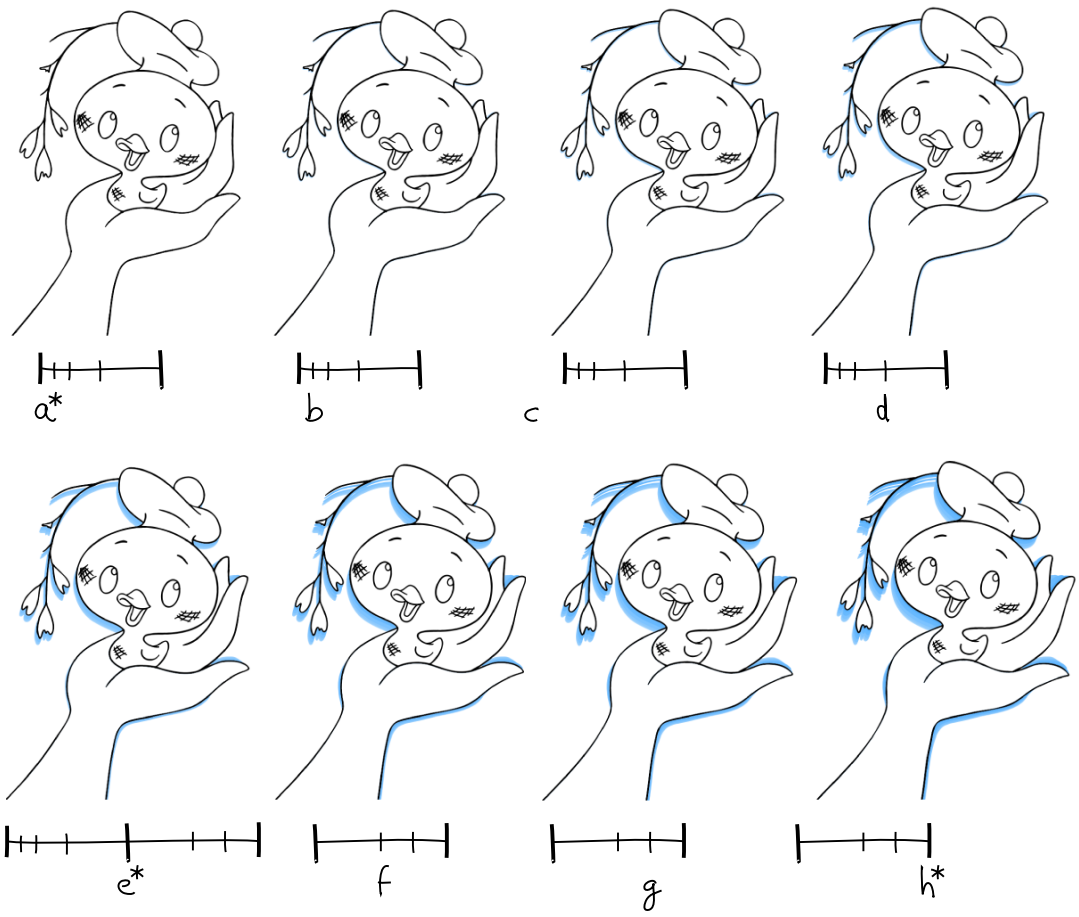

Figure 6.8: *Duck: Frames (a), (e), and (h) are the given key frames and (b-d) and (f-g) are generated inbetweens.*

6.9 illustrates an occlusion fix for this example, where a single stroke drawn by the user indicates the desired shape of the occluded region.

Figure 6.1 presents a step higher in the complexity range. Here there are many topological as well as occlusion challenges. The rotating right hand is the most notable drawing problem and can not be handled automatically. In this case a breakdown drawing for only the hand has been inserted for frames $b - d$, and the remaining portion of the frames were produced by the algorithm. Our framework allows the user to select desired areas and do the drawing themselves seamlessly within the automated context. This is important because there will always be portions of the drawings that the artist will want to do themselves.

Finally, in Figure 6.11 we show an example taken from [Kort, 2002]. Kort's technique attempts to solve occlusion issues algorithmically, but is only suc-

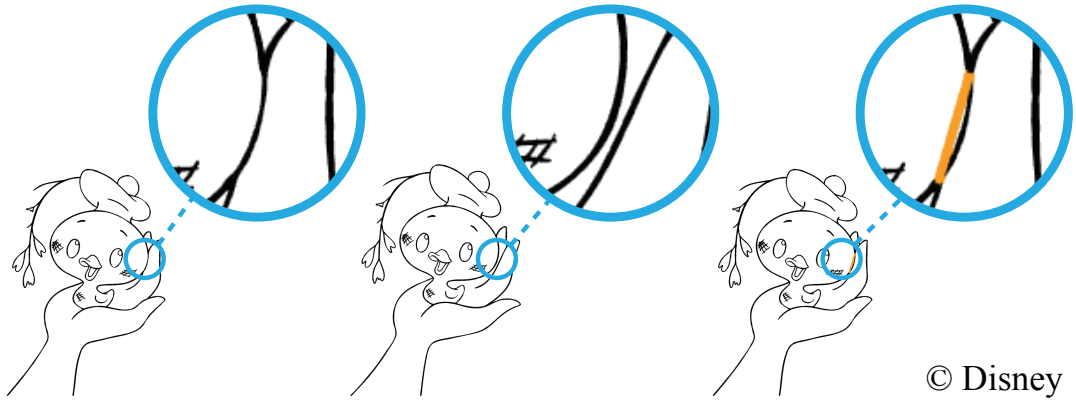

Figure 6.9: *Duck – User interaction: The first two images show an expanded region in each of the keys, a and e, where the finger is initially occluded and then moved from behind the duck. The user input occlusion stroke is shown on the right. Resulting inbetweens are shown in 6.8.*

Example	# Inb. Strk	Interactions	% Eff. Gain
Speaker	48	0	100
Goofy hand	84	13	85
Duck	770	12	98
Tinker Bell	456	24	95
Tone	145	24	83

Table 6.1: *Number of interactions required to generate the inbetweens vs. total number of hand-drawn inbetween strokes.*

cessful in the simplest of cases, and therefore the ears present a failure case. In contrast our system requires user interaction for all three stages: correspondence, feature identification, and occlusion (totaling 29 interactions) but was able to successfully generate coherent inbetweens, shown in the bottom row of 6.11.

For the purposes of evaluating our system, in addition to looking at potential efficiency gain, we can compare the results of the algorithm to hand-drawn inbetweens. We do not expect in general an exact match, as even between artists, different drawings are produced, but such a comparison can give some indication of the success of our approach. Figures 6.12,6.1 show the result of using our tool against original hand-drawn inbetweens by overlaying the two.

All of the above results were presented to a lead clean up artist to judge qual-

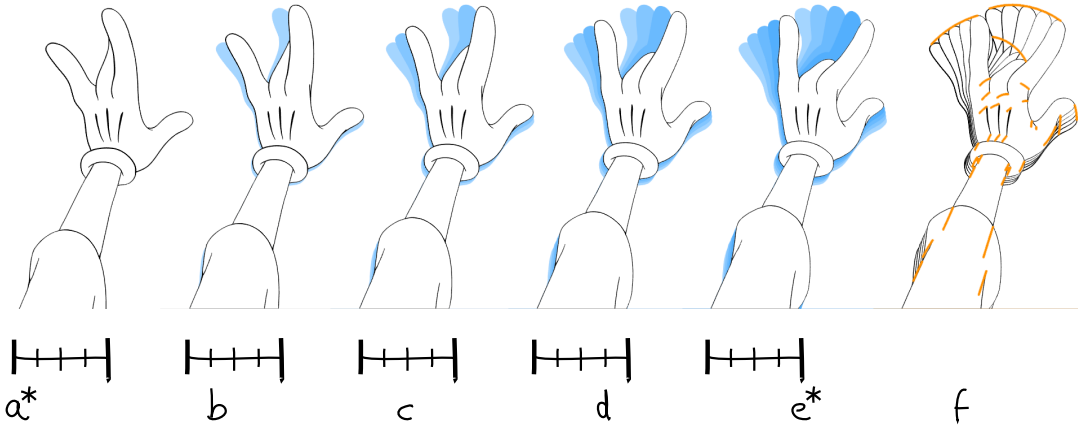

Figure 6.10: *Goofy hand: Frames (a) and (e) are the given key frames and (b-d) are generated inbetweens. The rightmost figure superimposes the frames and visualizes the trajectories. © Disney*

ity – as she would evaluate any of the inbetweens produced in her department. For the duck example and the Goofy hand example there was one note each regarding the shape of a single inbetweened stroke. To produce the desired shape for the duck it was sufficient to add one additional user-defined salient point. The note for Goofy's hand requesting a change of the location of the knuckle bend is shown in the left image in Figure 6.13. To implement this fix, the drawn fix stroke was added as a breakdown to the middle inbetween frame to produce the final results shown in the bottom row of Figure 6.13.

Trial Evaluation: In addition to comparing the generated inbetweens to existing hand-drawn examples, BetweenIT was evaluated by the effects department who used the tool to inbetween tone shapes. Tones, highlights, and shadows represent only a portion of the types of inbetweening performed in effects, but a time-consuming portion, and one well-suited to automation. Figure 6.14 shows an example tone . The frog character was loaded in as a reference background on which to draw the tone shown on the right side of her face. Note that the tone region is masked out against the character in the final composite (shown in the middle figure) and therefore the shape of the left side of the tone is not important.

Another example showing only the inner outline of a tone for a character's face is shown in figure 6.15. This shot was comprised of 5 key frames and 29 inbetweens. The rightmost figure in 6.15 shows all 34 of the tone frames. A hand-drawn version of all 29 inbetween frames took approximately an entire day to complete, while the version produced by BetweenIT was completed

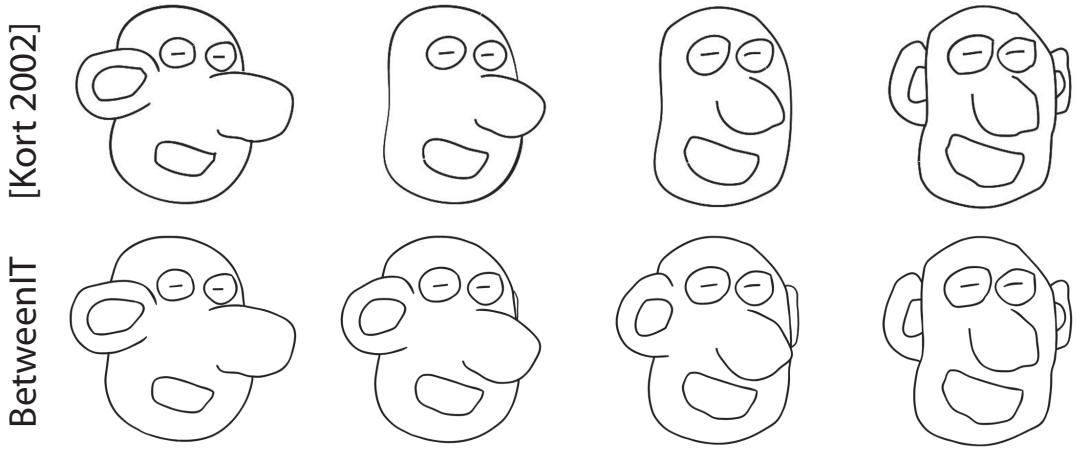

Figure 6.11: *Kort example: Comparison of results from [Kort, 2002], in top row, to results from BetweenIT shown in the bottom row.*

end-to-end in roughly 30 minutes. These numbers are just a indication of efficiency: the artist completed this example after just a single afternoon to get familiar with BetweenIT. We expect the efficiency gain to increase with more exposure to the tool. Our stroke efficiency measure for this example is 83% (20 salient points and 4 trajectories drawn – see Table 6.1), which maps to the task being completed 16 times faster than by hand.

6.6 Conclusion

We have presented the BetweenIT system for the user-guided automation of tight inbetweening. For cases where the user is not satisfied with the automated results, BetweenIT provides a context in which the user can guide the system in a natural way to produce quality results efficiently. The inbetweening is driven by a novel solution for stroke interpolation along natural arcs from only two keys.

We demonstrate workflow and results using BetweenIT on a collection of real production examples of varying complexity. The results generated are comparable to hand-drawn examples in many of our test frames. Where user intervention was necessary, the required input was small relative to the resulting reduction in the number of overall drawn strokes. After a single introductory session with the tool, an effects artist was able to efficiently produce inbetweened tones on multiple shots that were included in the final feature production.

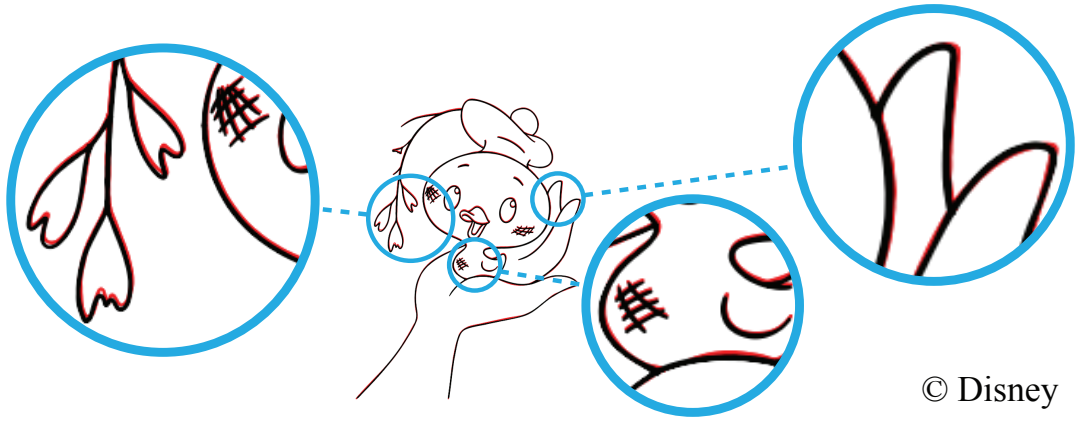

Figure 6.12: *Duck: An example frame comparing the results shown in Figure 6.8 by overlaying a hand-drawn frame in red.*

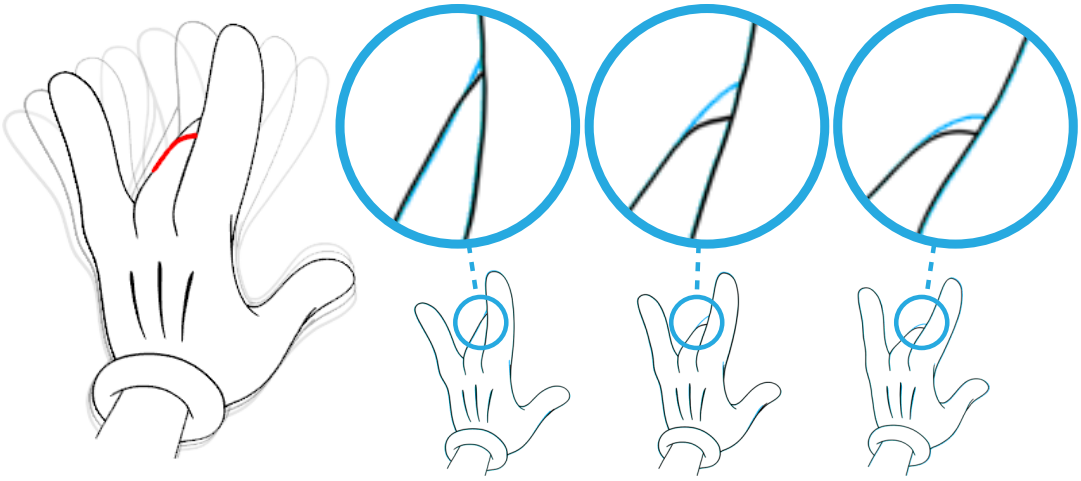

Figure 6.13: *Goofy hand: The left figure shows an artist's desired change (in red) to the knuckle from Figure 6.10. The remaining figures show the results after inserting a breakdown stroke at the middle frame. The expanded area compares the new inbetween to the one before the fix (blue). © Disney*

Figure 6.14: *Frog Tone: This example contains 9 key frames and 22 generated inbetweened frames. The leftmost figure shows a background reference image with the tone shape superimposed, the middle figure is the composited frame, and the rightmost image shows all 31 tone frames superimposed - color-coded by key frame. © Disney*

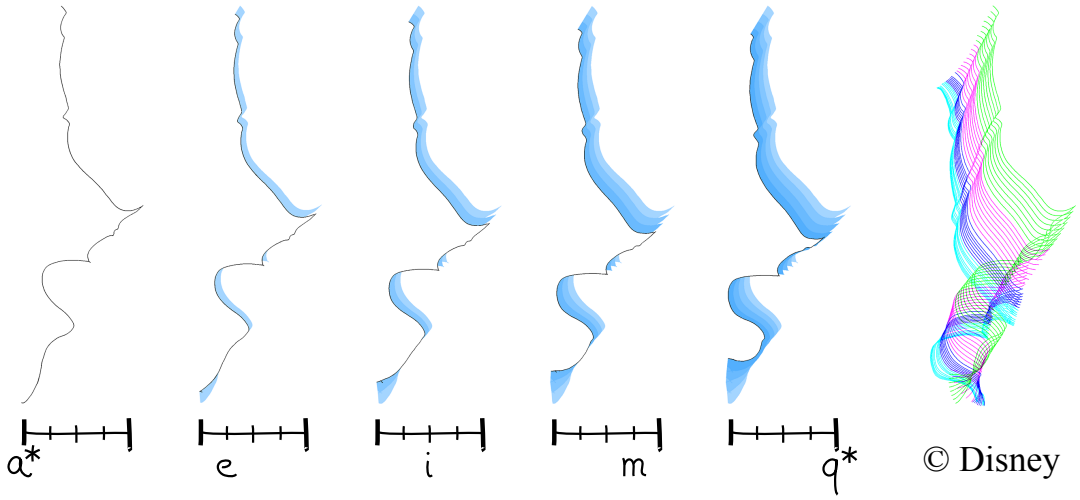

Figure 6.15: *Face Tone: Frames (a) and (q) are two key frames for the tone shape and (e,i,m) are a subset of the 15 generated inbetweens (b-p). The rightmost figure superimposes tone shapes for all 34 frames, with the (a-q) shown in green.*

While our hope is that such a tool will increase the efficiency of the inbetweening phase, and thus reduce the cost of a production, such a tool, if successful, could also broaden the scope of animation that can be practically supported. Scenes with very slow motion are expensive to inbetween and are therefore sometimes avoided. Our solution could make the incorporation of such scenes more feasible.

There are many avenues for improvement in our current system. We have primarily focused on the interpolation algorithms: the correspondence, and feature identification modules could benefit from further exploration. High-quality vectorization is also an open area.

CHAPTER 7

Temporal Noise Control

In this Chapter, we propose a technique to control the temporal noise present in sketchy animations. Given an input animation drawn digitally, our approach works by combining motion extraction and inbetweening techniques to generate a reduced-noise sketchy animation registered to the input animation. The amount of noise is then controlled by a continuous parameter value.

Our method can be applied to effectively reduce the temporal noise present in sequences of sketches to a desired rate, while preserving the geometric richness of the sketchy style in each frame. This provides the manipulation of temporal noise as an additional artistic parameter, e.g. to emphasize character emotions and scene atmosphere, and enables the display of sketchy content to broader audiences by producing animations with comfortable noise levels.

We demonstrate the effectiveness of our approach on a series of rough hand-drawn animations.

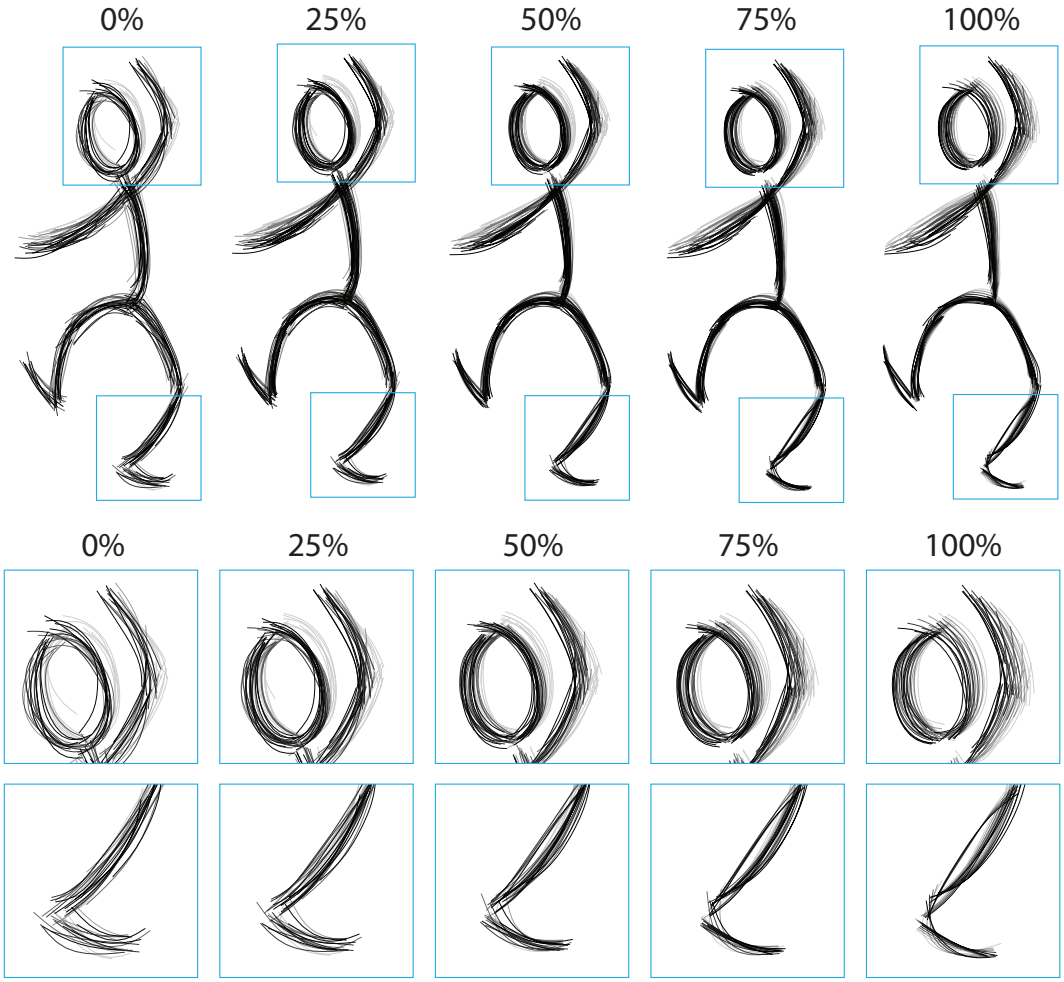

Figure 7.1: *Balancing scene. We compare different noise reduction values from* 0% *(input animation) to* 100% *(noise-free).*

7.1 Introduction

Compared to traditional cleaned-up drawings, sketches present a kind of visual richness, where both silhouette and interior lines are composed of many rough strokes. This style allows another dimension of expressiveness - emotion, action, and other features can be conveyed through the "sketchy" drawings.

The richness provided by the sketchy style can be considered to be a form of geometric noise. Despite its positive benefits in still images, geometric noise becomes temporal noise in sequences of sketches and is generally unpleasant to view. The industry solution to this problem is to remove the geometric

noise. In production environments, early versions of animation (both 2D and 3D) are often composed of sequences of rough sketches. Later in the pipeline, these are systematically replaced either with clean-line drawings or with renderings of 3D scenes, which typically present cleaner visuals. Animations completely made of sketches are less common and generally confined to short sequences or small productions[1].

Our goal is to preserve the expressiveness inherent in sketchy drawings while removing unpleasant temporal issues. We propose to reduce *temporal* noise while keeping *geometric* noise by supporting interpolation at a finer level, down to the individual sketchy strokes. Instead of appearing and disappearing, the strokes transition smoothly from frame to frame. Enforcing these constraints manually in typical production environments would be impractical: establishing a fine-scale correspondence of strokes within the sequence of sketches and generating the proper animation path is too labor-intensive.

A key insight of our approach is that we can first construct a noise-free animation using only a representative subset of the input frames such that effectively all temporal noise is removed. Then, the desired amount of noise can be continuously varied on top of this noise-free animation. Another key idea is the use of motion extraction to allow local stroke searches within the global motion - enabling an automated solution to the fine-scale stroke correspondence problem.

7.2 Related Work

Research efforts related to sketchy or hand-drawn styles of illustration can be divided into two main topics: simplification and generation (static images as well as temporally coherent animations).

In the area of automated simpification/beautification, several techniques have been developed that reduce the number of lines in a drawing by pruning possibly redundant lines while still conveying the notion of the original shape [Wilson and Ma, 2004, Grabli et al., 2004]. Instead of solely performing stroke reduction, Barla et al. [Barla et al., 2005] propose a perceptually motivated technique for synthesizing representative lines. Input strokes are first grouped using a greedy clustering algorithm that pairs strokes according to

[1] Notable examples include Frédéric Back's short "L'homme qui plantait des arbres" and a brief series of sketches in the "Colors of the Wind" sequence in Disney's animated feature *Pocahontas*.

screen-space proximity. A geometric reconstruction step then follows to produce a single line for each group. The described approach is for static drawings, though they propose incorporating a temporal aspect of perceptual grouping- "common fate", expressed by grouping line segments with similar velocity to produce coherent animation. Shesh et al. [Shesh and Chen, 2008] later extended this approach to handle time coherence by building a simplification hierarchy of strokes and using opacity blending to interpolate between a pair of strokes and its simplified version to create smooth transitions.

Generation of static sketchy or pen-and-ink style illustrations from input 2D images/photographs and from 3D models is a popular topic in the field of non-photorealistic animation and rendering (NPAR) (e.g. [Winkenbach and Salesin, 1994, Salisbury et al., 1997, Coconu et al., 2006]). Far less attention has been focused on developing temporally coherent results suitable for animation. Existing techniques [Curtis, 1998, Bourdev, 1998, Kalnins et al., 2002, Kalnins et al., 2003] focus on rendering stylized silhouettes of animated 3D models. After silhouettes are extracted from the 3D model the main challenge is to generate coherent "sketchiness" along the silhouette strokes over time, e.g. through assigning temporally coherent parameterizations to strokes in the image plane [Kalnins et al., 2003, Bourdev, 1998] or alternatively achieving coherence via a particle system seeded along the silhouette[Curtis, 1998] or through stroke texture representations [Benard et al., 2010] designed for temporal coherence. The key issue with all of these approaches is that they require an underlying 3D model or a clean 2D image with known stroke correspondences. In our case the challenge is that we have a set of unordered strokes in each frame which are not necessarily moving coherently and we need to determine how to best match and interpolate them over time.

7.3 Method

The method we introduce offers artistic control over the level of temporal noise in hand-drawn animations. Mismatches in the individual strokes used to define the same silhouette in consecutive frames can be a major source of temporal noise, particularly in rough sketches. Generally speaking, we seek to maintain the global motion of an input animation, as well as the overall drawing style, while manipulating the temporal noise level. Smooth output animations are typically preferred, but we note that high-frequency noise can be an effective artistic tool. The animation of a character who is scared, for

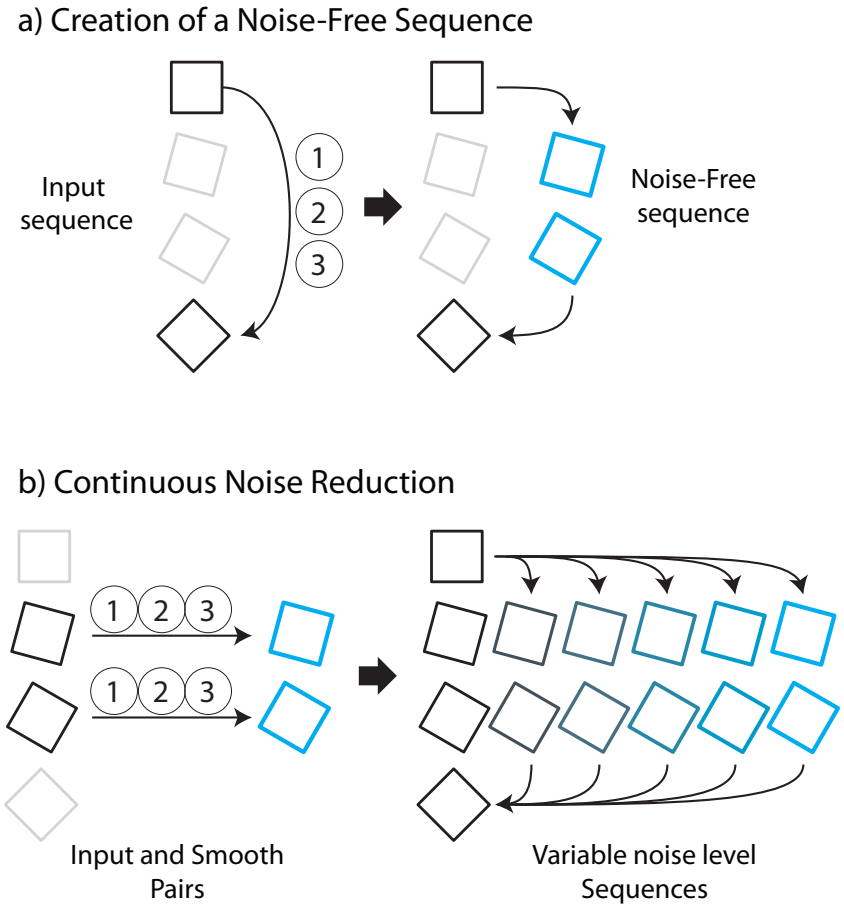

Figure 7.2: *Method Overview. Our method works in two phases. (a) The input sequence is processed to create a Noise-Free sequence. To do so, three steps are performed: (1) Motion extraction (2) Frames Deformation and Stroke-to-Stroke correspondences, and (3) Interpolation. (b) For each pair of input vs. noise-free frames, an arbitrary number of sequences with increasing noise reduction can be generated, again using the same three steps.*

instance, could benefit from a certain amount of temporal noise to emphasize emotion.

Before describing our method, we introduce the notation used throughout the paper. Our algorithm operates on sequences of frames, where each frame $\mathcal{F}$ contains a set of strokes that appear in the animation at the same moment in time. Each stroke s is a piece-wise linear curve defined by a sequence of vertices. The i-th stroke in a frame $\mathcal{F}$ is given by $\mathcal{F}(i)$.

A motion field is a function $\mathcal{M} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ that tracks the movement of every point on the 2D canvas. In particular, $\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}$ is the motion field that describes the relative motion between frames $\mathcal{F}_1$ and $\mathcal{F}_2$. We define $\mathcal{D}(\mathcal{M}, \mathcal{F})$ to be the deformed frame that is obtained by taking the vertices of every stroke in $\mathcal{F}$ and displacing them according to $\mathcal{M}$.

7.3.1 Overview

Our approach takes as input a sequence of frames from a hand-drawn animation. The core algorithm works in two passes (see Figure 7.2). First, the input sequence is processed to create a *noise-free* animation (see Figure 7.2a). This is done by sampling the original input animation to choose representative frames. These frames are smoothly interpolated to create a new animation, and, for the time-being, all other frames not in the representative subset are ignored. The output of this stage is a set of smooth, automatically-created inbetween frames.

In the limit if we choose only the first and very last frame of the animation as the representative set, replacing frames from the original sequence with these newly synthesized inbetweens would effectively produce a noise-free animation. However, much of the finer scale motion and sketchy details would also be removed. On the other hand, if the representative set includes all of the original frames, then no interpolation is performed and we have the original, noisy content.

Our goal is to allow an artist to precisely control the level of temporal noise. This is enabled by the second pass of our algorithm, which smoothly interpolates the original noisy animation with the smooth inbetweens created during the first pass (see Figure 7.2b).

Both passes must solve the same problem of creating smooth inbetween frames. In other words, given two frames of animation, we must establish a fine-scale stroke correspondence and interpolate smoothly between each stroke pair.

7.3.2 Representative Frame Sampling

The first phase of our algorithm generates a noise-free sequence which ideally should resemble the original animation as much as possible. The sampling scheme (i.e. choice of representative frames) is crucial to the quality of the resulting animation. Our method is comprised of two main components: the sampling strategy and timing control.

We propose two selection strategies for choosing representative frames. *Uniform sampling* selects representative frames distributed at equal intervals specified by a window size w. *Keyframes* uses important or extreme frames that are manually selected from the original animation input.

Once the representative frames are selected, our system assumes a uniform division of time units inside each interval, resulting in an approximation of the input timing. Our experience is that, by respecting the input animation keyframes, this approximation is acceptable. However, for particular scenes it might be desirable to have finer control. This can be achieved by altering the timing values used to generate the inbetweens in [Whited et al., 2010]. It is assumed that the keyframe specification and timing information, if needed, can be provided as input, along with the original animation. This is suitable for a classic animation environment, where both keyframes and timing information are captured by timing charts.

7.3.3 Creating Smooth Inbetween Frames

Algorithm 2: *CreateInbetweens*

Input: $\mathcal{F}_1, \mathcal{F}_2$**: animation frames**
Input: t**: interpolation parameter,** $0 \leq t \leq 1$
Output: $\mathcal{F}_t$**: an inbetween**
$\hat{\mathcal{F}}_1 \leftarrow \mathcal{D}(\mathcal{M}^{\mathcal{F}_2}_{\mathcal{F}_1}, \mathcal{F}_1)$
$\mathcal{S} \leftarrow computeStrokeCorrespondencePairs(\hat{\mathcal{F}}_1, \mathcal{F}_2)$
$\mathcal{F}_t \leftarrow \{\}$
for all $(i, j) \in S$ **do**
    $s \leftarrow interpolateStrokes(\mathcal{F}_1(i), \mathcal{F}_2(j), t)$
    $\mathcal{F}_t \leftarrow \mathcal{F}_t \cup \{s\}$
end for

Algorithm 2 describes the steps to create smooth inbetween frames. Both passes of our method use this algorithm. The input consists of a pair of

representative animation frames, $\mathcal{F}_1$ and $\mathcal{F}_2$, and an interpolation parameter $t, 0 \leq t \leq 1$. Our goal is to create a new frame $\mathcal{F}_t$ using solely the strokes from frames $\mathcal{F}_1$ and $\mathcal{F}_2$. This process ensures continuity in the strokes that are output as t varies between 0 and 1, and thus results in smooth animations.

The first step towards creating smooth inbetween frames consists of identifying the pairs of strokes from $\mathcal{F}_1$ and $\mathcal{F}_2$ that represent the same features in the drawing at different moments in time. Every stroke from one input frame is matched to the most likely candidate stroke from the other frame, as expressed by a stroke correspondence measure. We refer to this process as finding the stroke-to-stroke correspondences (see Section 7.3.3).

Strokes that are used to define the same element in a drawing (e.g. a portion of the silhouette) can be far apart spatially from frame to frame. Similarly, strokes that represent different elements of the drawing can become spatially close. Computing appropriate stroke-to-stroke correspondences in this setting is therefore a very difficult task, since proximity is not a reliable measure of similarity. To mitigate this problem we compute the stroke correspondence measure after deforming the strokes of $\mathcal{F}_1$ according to the motion field $\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}$ (see Section 7.3.3). More precisely, we compute the stroke-to-stroke correspondences between the frames $\hat{\mathcal{F}}_1 = \mathcal{D}(\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}, \mathcal{F}_1)$ and $\mathcal{F}_2$. In general, the spatial distances between the strokes in the deformed version of $\mathcal{F}_1$ and $\mathcal{F}_2$ are significantly smaller, so computing stroke-to-stroke correspondences in this setting is less prone to mis-matches. The stroke-to-stroke correspondence step results in a set of pairs (i, j), where each pair indicates a correspondence between stroke i of $\mathcal{F}_1$ and stroke j of $\mathcal{F}_2$. All pairs of corresponding strokes are then interpolated to create the inbetween frames $\mathcal{F}_t$ (see Section 7.3.3).

The remainder of this section outlines the method used to create the motion fields, the process of deforming the input frame, the criteria used to compute the stroke-to-stroke correspondences, and the stroke interpolation method.

Motion Extraction

An important part of our processing pipeline consists of computing the relative motion, or motion field, between two frames $\mathcal{F}_1$ and $\mathcal{F}_2$. We use a slightly modified version of the As-Rigid-As-Possible (ARAP) method [Sykora et al., 2009a]. The original approach assumes the mask of the registered image is known beforehand. This allows the control lattice, which represents the motion field $\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}$, to be adapted to the topology variations of the underlying shape.

In our problem domain, we are dealing with a more complicated scenario, as the input consists of an unordered set of strokes without any connectivity information. However, it is still important to take into account the topology of the input sketch, as opposed to using a uniform grid. To overcome this difficulty we compute a rasterized distance field, as illustrated in Fig. 7.3, which we use as a mask. The computed distance field closes small gaps between the input strokes. In addition, the distance field provides a better cue for image registration (similar to *chamfer matching* [Borgefors, 1988]).

In addition to the distance field, we use a hierarchical coarse-to-fine approach to compute the motion field. We first build a multi-resolution pyramid of images by recursively reducing the image size by a factor of two. Each image level has a corresponding control lattice whose resolution also differs at each level by a factor of two. We run the registration algorithm on the lowest resolution image first, with the coarsest control lattice. We then render a pixel accurate representation of the motion field which is used to initialize the positions of the control points of the finer lattice used for the image at the next resolution level. The process continues until all images in the image pyramid have been processed. We have observed that this hierarchical approach speeds up the convergence of the motion field extraction algorithm and increases the method's robustness under large motions.

As noted in [Sykora et al., 2009a] the image registration algorithm can potentially get trapped in a local optimum. These cases are typically rare but when they occur we let the user drag-and-drop selected control points in order to guide the algorithm towards a better solution. This operation can be implemented by fixing the positions of the selected control points and changing their associated weights to large values as in [Wang et al., 2008].

Frame Deformation and Stroke Correspondences

The motion field $\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}$ represents the relative motion between frames $\mathcal{F}_1$ and $\mathcal{F}_2$. This provides a way of estimating the location of each stroke from $\mathcal{F}_1$, if it had been drawn at the time represented by frame $\mathcal{F}_2$. The computation of the deformed frame is straightforward. The vertices v (which are simply 2-dimensional points on the digital canvas) defining the strokes in $\mathcal{F}_i$ are displaced according to the motion field: $v \leftarrow v + \mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}(v)$.

The stroke correspondence step is used to compute a measure of how well two strokes from different frames will interpolate. Intuitively, the better aligned and spatially close the two strokes are, the better their correspondence measure should be. For our work, we define the correspon-

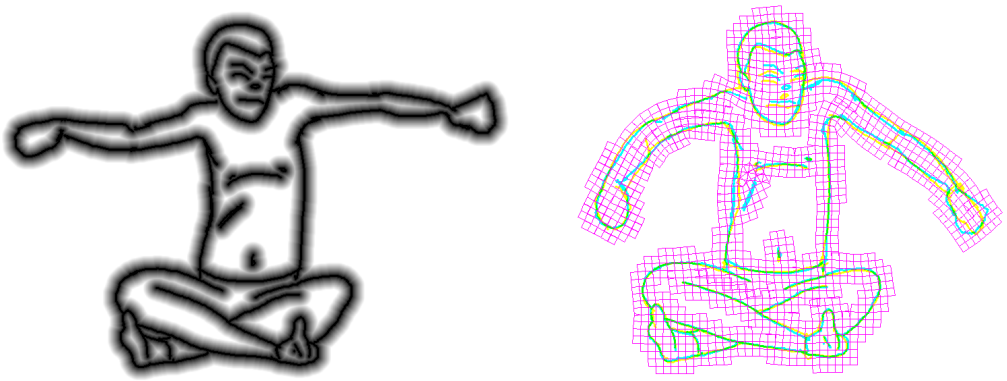

Figure 7.3: *Control lattice for the ARAP image deformation: distance field computed from rasterized strokes (left), ARAP registration using the control lattice based on the distance field (right).*

dence measure between two strokes s_1 and s_2 as $h(s_1, s_2) * h(s_2, s_1)$, where $h(A, B)$ is a component of the Hausdorff distance. More precisely, $h(A, B) = max_{a \in A}(min_{b \in B}(d(a, b)))$ and $d(a, b)$ is the Euclidean distance between points a and b, i.e. the vertices of strokes A and B. We note that this is one of many choices of similarity measures [Seah and Tian, 2000, Veltkamp, 2001, Lie et al., 2010]. However, in our experiments, we found the Hausdorff distance to work well and be more robust than other measures.

Stroke Interpolation

The stroke correspondence algorithm is used to find all pairs of strokes that need to be interpolated to create the inbetween frames. We use the three-step deformation method introduced in Whited et al. [Whited et al., 2010] to smoothly interpolate pairs of strokes. The strength of this approach over other techniques [Fu et al., 2005, Baxter and ichi Anjyo, 2006, Baxter et al., 2009b] is that in addition to interpolating the stroke curvatures, it also captures the affinity of the global motion (rotation, translation and uniform scaling).

This interpolation method computes animation trajectories as spirals based on the relative orientation of the strokes, as measured using information from $\hat{\mathcal{F}}_1$ and $\mathcal{F}_2$.

Flipping a stroke will result in different animation spirals. During the stroke interpolation the two frames being interpolated may be far apart, and in case of large rotation, the relative orientation of strokes may be unreliable. To

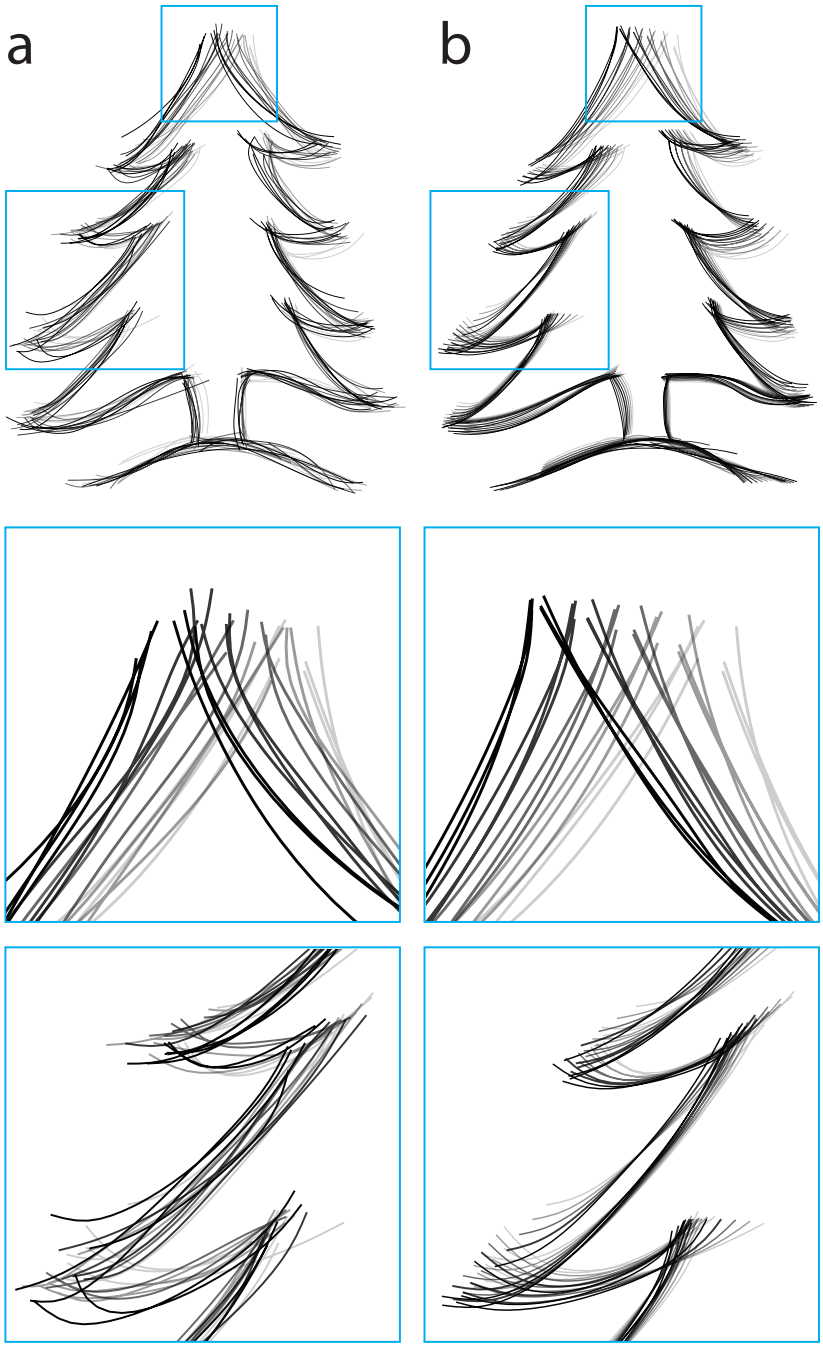

Figure 7.4: *Tree scene. We compare the input animation (a) with the noise free result (b). Close-ups are marked by blue squares.*

improve the result, we slightly modify the original algorithm by storing the relative orientation of strokes during the Stroke Correspondence search. This is when the frames are deformed to overlap, and therefore the relative orientation of strokes is predictable.

7.4 Results

We tested the effectiveness of our algorithm on hand-drawn animation sequences containing between 30 and 90 frames. In order to evaluate our motion extraction algorithm, the animation of the *Tree* shown in Figure 7.4 was generated procedurally (see Figure 7.5). All other animations were created manually with a digital drawing tool. The *Square* animation, shown in Figure 7.6, presents a simple case with an almost rigid motion. The use of temporal noise as an artistic tool is investigated using the *Face* animation, which is shown in Figure 7.11. Lastly, the *Balancing* animation, Figure 7.1, illustrates another challenging example handled by our framework.

For the examples in this Section, to emphasize the temporal progression, consecutive animation frames are displayed with decreasing opacity. For the *Square*, *Balancing*, and *Face* scenes, we used a uniform representative frame sampling, with a window size of 7 frames. The *Tree* scene uses a set of 8 selected keyframes and the resulting window sizes are between 4 and 7 frames. Temporal reduction values are displayed as percentages, where 0% is the input animation and 100% is the Noise-Free animation.

Our algorithm was developed in C++ and runs as a single thread. On a standard workstation, the execution of Algorithm 2 takes up to 10 seconds per pair of frames, with the motion extraction and the search for correspondences being the most time-consuming tasks.

7.4.1 Ground Truth Comparison

To evaluate our motion extraction method we created a "ground truth" animation of the Tree in Maya. Starting with a planar textured mesh of the undeformed tree (see Figure 7.5), the mesh was deformed by using the bend deformer tool and keyframing the curvature. The deformed tree texture was then used as a reference over which an artist sketched each frame of the animation.

Figure 7.7 compares the extracted motion field with the ground truth generated in Maya. In general, the extracted motion captures both the global

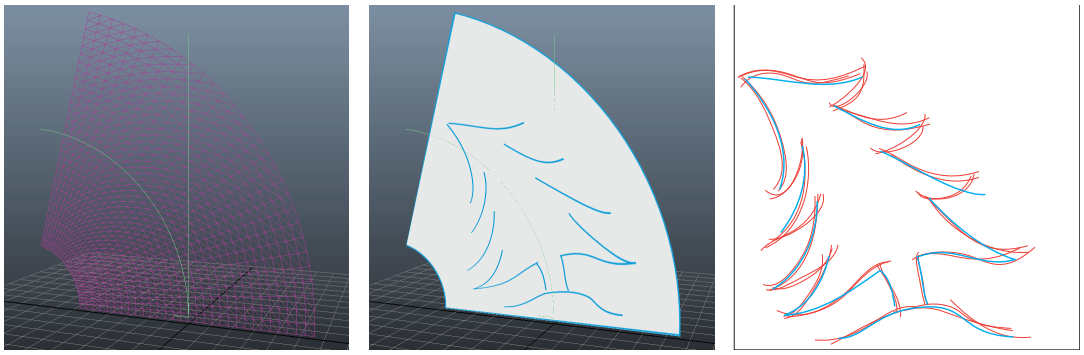

Figure 7.5: *Generation of the motion ground truth. A planar grid is generated and deformed with Autodesk Maya (left). A reference image is textured on the mesh (center), generating a reference animation over which the final tree animation is sketched (right). The global motion is captured by the mesh deformation.*

animation motion and local deformations due to the geometric noise. As a result, it provides a very precise stroke alignment, which greatly simplifies the task of finding stroke to stroke correspondences.

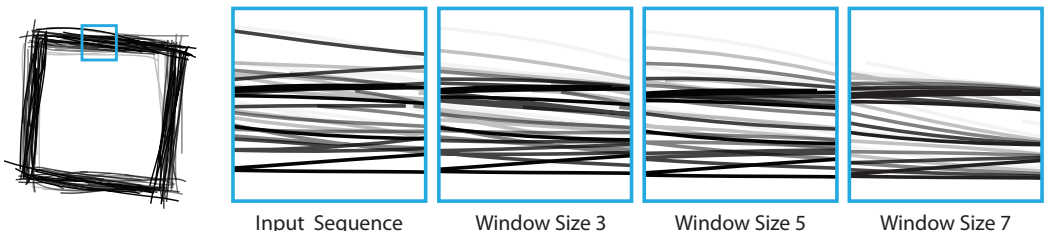

Figure 7.6: *Square Scene. This image shows the effect of increasing window sizes. Each close-up shows 5 animation frames overlayed with decreasing opacity. When the noise is high, the lines look evenly distributed and unstructured (Input Sequence). With low temporal noise, lines appear to follow a structure and tend to be clustered (Window Size 5 and Window Size 7).*

7.4.2 Neighborhood Averaging Comparison

We compare our results with those obtained through neighborhood interpolation, which proceeds as follows. For each sample point p_1 of a stroke j in a frame i, we collect the two nearest neighbor points p_2 and p_3 in frames $i-1$

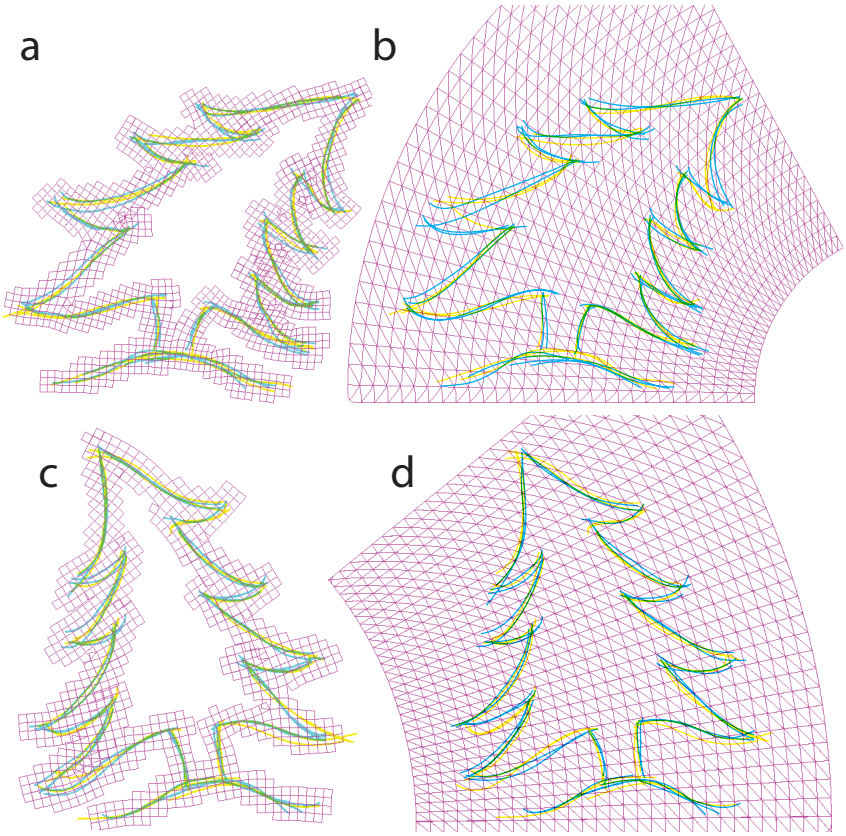

Figure 7.7: *Motion Extraction Comparison. The extracted motion (a and c) is compared with the ground truth motion (b and d). The blue and yellow frames represent $\mathcal{D}_i^j$ and $\mathcal{F}_j$, with $i = 22, j = 26$ (a,b) and $i = 29, j = 33$ (c,d). The extracted motion is precise and simplifies the search of stroke to stroke correspondences.*

and $i+1$. p_1 is then updated to $p_1 \leftarrow (p_1 + p_2 + p_3)/3$. An example result is shown in Figure 7.8. Since each sample point p_k is free to move independently, divergent attractions result in breaks of the line continuity. This usually happens when sample points of one stroke are influenced by different sets of strokes.

This result shows that a simple averaging approach is not desirable. In general, we observe that when the motion is large, this approach produces obvious artifacts, losing important features of the frame. This motivates the use of motion extraction to alleviate the effects of large animation motions. Even when the motion is small, kinks and undesirable deformations are present. This motivates the consideration of strokes as atomic entities, therefore shift-

ing the correspondence from the individual points samples to the stroke level. Our approach benefits from both considerations.

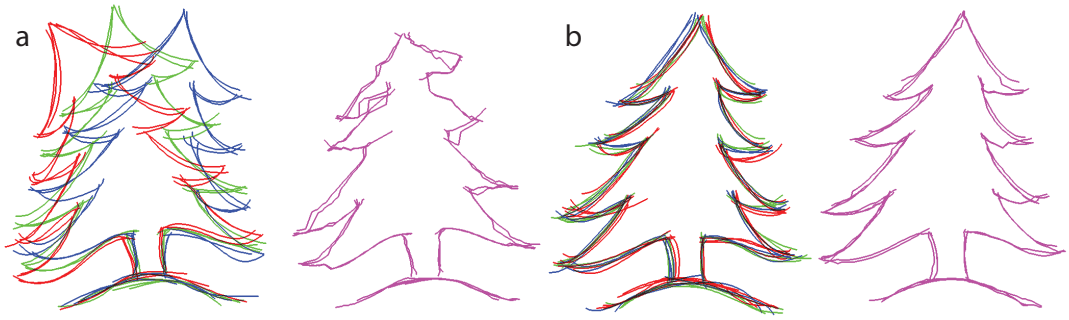

Figure 7.8: *Neighborhood Averaging. A window of three frames (red, green, blue) is used to compute a per point neighborhood average (purple). When the motion is large (a), the resulting frame is strongly degenerated. Even for slow animations (b), this approach leads to undesirable kinks and deformations.*

7.4.3 Sampling and Timing Control

As discussed in Section 7.3.2, we implemented two selection strategies for choosing representative frames, *uniform sampling*, and manually designated *keyframes*. Figure 7.9 compares results of using the different selection strategies. Notice that using the input animation keyframes, as opposed to uniformly sampling the frames, greatly reduces the motion error.

Figure 7.10 shows the effect of different timing charts applied to the same set of frames where the timing values used to generate the inbetweens in [Whited et al., 2010] are altered. Notice how the spaces are affected by the choice of the timing functions - most notably at the tip of the tree.

7.4.4 Temporal Noise as an Artistic Tool

The second phase of our method allows the generation of sequences with varying temporal noise. By manipulating the noise reduction level in different parts of the animation, an artist has the ability to use the noise as an additional storytelling element. Noisy animations can be used to portray certain feelings, such as anger or fear. A proof of concept is shown in Figure 7.11.

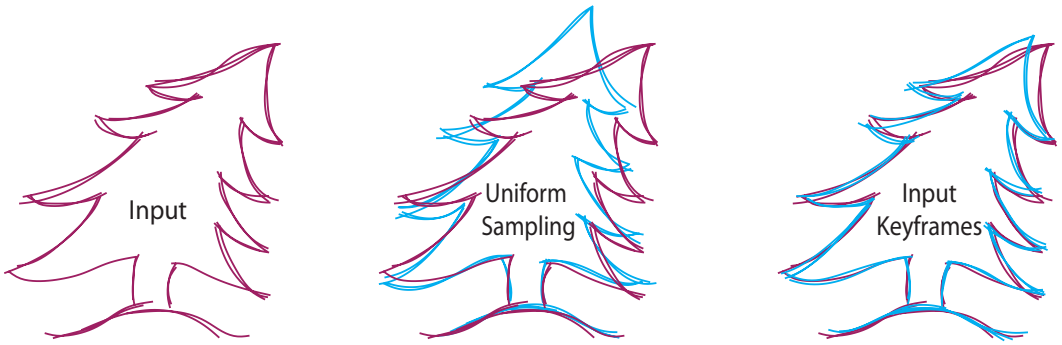

Figure 7.9: *Sampling Strategy. This image shows the maximum motion error (visualized as misalignment) obtained in the Tree animation using two different sampling strategies: Uniform Sampling, with a window size of 7 frames, and Keyframes, with 6 keyframes marked at the important animation times.*

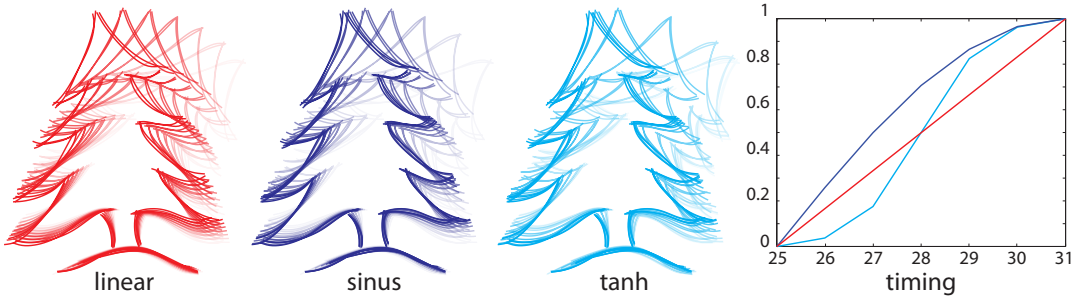

Figure 7.10: *Timing Control. This image shows the effect of different timing functions applied to the same set of frames.*

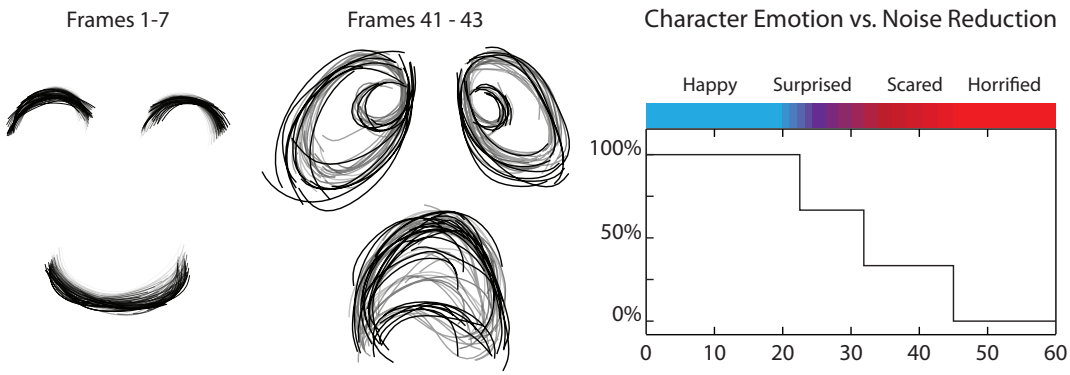

Figure 7.11: *Face. This scene presents a character with an emotional evolution from happy to horrified.*

7.5 Conclusion

We have presented a novel technique for noise manipulation in sketchy animations and demonstrated its use on a series of hand-drawn inputs. Our approach not only makes the production of larger scale sketchy animations feasible through automated noise reduction, but also widens the scope of artistic control to include noise as a first-class creative device.

7.5.1 Limitations and Extensions

One limitation of this work is due to the intrinsic difficulty of handling occlusions, topology changes, and disconnected components moving independently in a 2D environment. For the motion extraction step, distinct objects with different motions can create divergent motion fields and occlusions can force unnatural compressions; both cases are difficult to capture with a ARAP model. In order to handle these scenarios in complex scenes, the input would need to be first segmented into coherent layers.

Additionally, complex curved strokes that self intersect may cause problems in the correspondence and interpolation steps. A simple solution, similar to what is proposed in Barla et al. [Barla et al., 2005], is to split these strokes into separate line segments. However, every segment would then behave independently, which may not be desirable.

Another limitation of our work lies in the selection of keyframes. As shown in Figure 7.9 the set of selected representative frames has a significant impact on the output animation. In particular, as the complexity of the animation increases (i.e. higher frequency motions), a larger number of keyframes is needed to preserve the motion. This effectively limits the number of smooth inbetween frames that we can create to reduce temporal noise.

Our frame inbetweening technique needs to be extended to provide smooth interpolation across multiple keyframes. Although the interpolating motions computed by the logarithmic spiral technique [Whited et al., 2010] are smooth between consecutive keyframes, the approach has the limitation that continuity is not necessarily preserved across longer sequences. However, multiple techniques have been proposed to preserve this continuity at the keyframes, while still interpolating them. In one approach [Powell and Rossignac, 2008], subdivision is used to smoothly interpolate 3D poses by blending consecutive screw motions. A more recent approach [Rossignac and Vinacua, 2011] also produces continuous motions as a blend-

ing of consecutive "steady affine motions". In 2D, both of these algorithms may be applied to logarithmic spirals with little modification.

7.5.2 Relative Stroke Orientation

As presented in Section 7.3.3, we use [Whited et al., 2010] to create smooth blends for pairs of matched strokes. This interpolation method computes animation trajectories as spirals based on the relative orientation of strokes. Flipping a stroke will result in different animation spirals. During the stroke interpolation the two frames being interpolated may be far apart, and in case of large rotation, the relative orientation of strokes may be unreliable. To improve the result, we slightly modify the original algorithm by storing the relative orientation of strokes during the Stroke Correspondence search. This is when the frames are deformed to overlap, and therefore the relative orientation of strokes is predictable.

We mention a rare failure case. Curved object boundaries make the relative orientation of strokes harder, and mismatches may occur. Figure 7.12 shows an extreme example where two strokes lying on a curved boundary are matched together. The relative orientation computation would produce the wrong result (a) in contrast to the the desired solution (b).

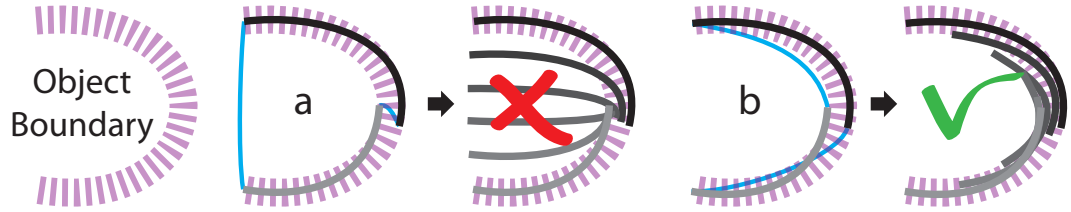

Figure 7.12: *Curved boundaries of objects can, in extreme cases, deceive the computation of the relative orientation of strokes, resulting in the wrong motion trajectories (a) opposed to the desired ones (b).*

CHAPTER 8

Conclusions

This chapter concludes the thesis by summarizing and discussing the major contributions and suggesting future research directions.

8.1 Discussion

In this thesis we have proposed a set of digital tools to support the production of 2D Animation. Our focus has been to conceive tools that keep the artist as a central agent supported by computer-assisted solutions. In order to favor artistic control, we have applied full automation only for tasks with little need for artistic interpretation. We have collaborated extensively with professionals from the Walt Disney Animation Studios, learning about the industry problems and incorporating in our tools feedback and suggestions from artists and engineers.

We started our work by studying the production pipeline of traditional 2D Animation, as well as learning about the existing digital solutions. We identified a number of challenges related with computer assisted cartooning, and defined three core problems — representation, correspondence, and interpolation — that in our opinion hold the key to revolutionary changes in this

field. We then proceeded by considering possible tools to address these problems. Our work proposes three categories of tools: digital representation, pre-processing, and applications.

We have studied the problem of representation, analyzing the existing digital representations and proposing a novel "hybrid" representation. We have highlighted the advantages and shortcomings of existing raster and vector representations, and defined the requirements for a representation suitable for 2D Animation drawings. We have proposed a novel, Vector-Splats hybrid to combine the visual richness of raster with the editing capabilities of vector images. Raster data is captured by splats, and rendered using EWA Splatting. Splats are mapped to centerlines with a simple tangent-normal frame from the closest point. Deformations of the centerlines are propagated to the splats. Additionally, we investigate the issue of representing both areas and lines with a unique vector description. We propose the use of both centerlines and boundary lines, and define of relationship between them based on different editing scenarios.

We have presented a system for the vectorization of clean line drawings. This problem is challenging as when lines are drawn close to each other, or intersect, local information becomes ambiguous. Our observation is that a non-local scope is necessary to make educated choices in how to deal with these ambiguities. We developed a two-step method that does a first initial reconstruction, which is used to learn the drawing topology and identify the ambiguous regions, and then addresses these regions by reconstructing plausible configurations and picking a heuristic optimum. As demonstrated in the results, our method improves the vectorization of junctions and nearby strokes, features which existing vectorization solutions fail to reconstruct faithfully.

We have then developed an interactive segmentation tool for the labeling of strokes. This tool makes it possible organize the drawing strokes into semantic groups, preparing the drawing for further processing. Given a drawing, the user sketches scribbles associated with labels over the region of interest, and the system solves a energy minimization to generate a labeling of the strokes. In contrast to previous work, our method exploits both geometric and temporal information. Similarities between the scribbles and the drawing strokes, as well as among the drawing strokes, are used to define the energy terms of our formulation. A user study was conducted to learn about the experience of novice users, as well as compare our method with common selection tools. The results of the study show that our method is generally faster than existing tools, and registered positive qualitative feedback with respect to the ease of use and learning curve of our solution.

Inbetweening represents one of the most labor intensive tasks in the production of traditional animated features. We propose a computer-assisted solution for tight inbetweening. In our method, drawings are represented as graph of strokes. Given a pair of key frames, a mapping between the graphs is derived. Inspired by the animation principle of moving with arcs, we compute logarithmic spiral trajectories for the junctions and salient points, and then interpolate strokes ensuring smooth continuity across junctions. We provide the user with a set of tools to correct for mismatches in the key frames topology, deal with simple occlusion cases, and manipulate the trajectories. Our method minimizes the number of required artist-gestures, and a trial evaluation showed great productivity improvements over manual solutions.

Finally, we have proposed a method for control of temporal noise in sketchy animations. Sketches present a particular visual style where perceived boundaries and interior lines are approximated by many rough strokes. In a sequence of sketches, the lack of temporal coherence of this geometric variation results into the perception of flickering lines, an effect we call temporal noise. In order to control the amount of noise of a given sequence, our method performs two steps. In the first phase, selected key frames from the input animation are processed to generate a so called "noise-free" sequence. In the second phase, we process each pair of corresponding frames from the input sequence and the noise-free sequence, generating sequences of decreasing temporal noise. Finally, the user can select the desired temporal noise level. Our method enables the production of longer scale sketchy animations by automatic reduction of noise, and widens the scope of artistic control by exposing the temporal noise as creative device.

8.2 Future Work

In this thesis we have proposed tools to address different problems in relation to the support of 2D animation. This final Section outlines some areas of future work in the context of the solutions presented in the individual thesis Chapters. We end with a more global vision of future directions for advancement of the field based on the findings of this research effort, in reference to the core problems introduced in Chapter 2.

For the digital representation of drawings, we have proposed in Chapter 3 the concepts of vector-splats and line-areas hybrids. Further investigation is necessary to understand the proper relationship between boundaries and centerlines, as well as how the deformation of boundaries should influence the interior, both in terms of centerline and splats.

To address the correspondence problem, we consider the embedding of 2D drawings into deformation fields driven by a 3D canvas. We touched on this topic in Chapter 7, by embedding strokes into deformation mesh. Our meshes however were only 2D. 3D geometry introduces the necessary depth information that is lost in 2D drawings, making it possible to solve otherwise complicated occlusion problems. The deformation of a given mesh defines the necessary correspondence across the animation. Recent efforts such as [Schmid et al., 2011] propose the embedding of 2D strokes into 3D canvas based on proxy geometry to explore new the visual style of 3D animation. The Walt Disney Animation Studio recently produced a short called "Paperman", where 2D strokes are animated from underlaying proxy geometry. We believe this is a very interesting and promising research path. Not only does it solve some of the major challenges of computer assisted 2D Animation, but it also provides a bridge between 2D and 3D artists and tools, allowing both fields to benefit from each other.

For the interpolation problem, we have proposed in Chapters 6 and 7, techniques for the interpolation of clean line drawings and sketchy drawings. Our work however only considered the geometric and temporal aspects of the interpolation, but we have not studied how the visual details, such as stroke textures, but also more painterly effects, should evolve during the animation.

Conceptually, we believe that the production of 2D Animation should be only as labor intensive as the artistic interpretation requires it. Everything else should be automated. We experienced that the principle of keeping the artist in the loop is the key to support 2D Animation. We consider the use of 2D-3D hybrid systems as a very viable path for future development. These systems maintain the freedom of 2D drawings, but introduce the missing information necessary to process the data algorithmically and truly gain from automation. The complexity may lie in the need to animate the story twice (with the proxy geometry first, and in drawings second), as well as balancing the distribution of the complexity of the animation.

We are confident that these problems, if addressed with the right methodology, can be addressed and solved, and by looking at the most recent developments, we think that we might be standing at the verge of a revolutionary change of this field.

APPENDIX A

Curriculum Vitae

Personal Information

First Name:	Gioacchino
Last Name:	Noris
Address:	Disney Research, Zurich Clausiusstrasse 49 8092 Zurich, Switzerland
Phone:	+41 79 375 48 30
Email:	chino@disneyreserach.com
Homepage:	http://graphics.ethz.ch/∼gnoris/
Nationality:	Swiss
Data and place of birth:	May 14th 1983 in Lugano, Switzerland

Education

since November 2008	joint Ph.D. student at the Swiss Federal Institute of Technology (ETH) and Disney Research Zurich (DRZ).
Sept. 2006 - Sept. 2008	M.Sc. Visual Computing, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.
Sept. 2003 - Sept. 2006	B.Sc. Computer Science, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

Employment

Nov. 2009 - Jun. 2012	Research Assistant, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.
Nov. 2006 - Sept 2008	Assistant, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

Publications

ACM Transactions on Graphics, to appear	G. Noris, A. Hornung, R. W. Sumner, M. Simmons, M. Gross: *Topology-Driven Vectorization of Clean Line Drawings*
Computer Graphics Forum, to appear	G. Noris, D. Sýkora, S. Coros, M. Simmons, B. Whited, A. Hornung, A. Shamir, M. Gross, R. W. Sumner: *Smart Scribbles for Sketch Segmentation*
SIGGRAPH 2012	T. Beeler, B. Bickel, G. Noris, P. Beardsley, S. Marschner, B. Sumner, M. Gross: *Coupled 3D Reconstruction of Sparse Facial Hair and Skin*
NPAR 2011	G. Noris, D. Sýkora, S. Coros, B. Whited, M. Simmons, A. Hornung, M. Gross, R. W. Sumner: *Temporal Noise Control for Sketchy Aniamtion*
Eurographics 2010	B. Whited, G. Noris, M. Simmons, R. W. Sumner, M. Gross, J. Rossignac: *BetweenIT: An Interactive Tool for Tight Inbetweening*
ETH Zurich, master thesis 2008	G. Noris: *Computer-Assisted Cartooning: Inbetweening*

List of Figures

List of Tables

Bibliography

[Accot and Zhai, 1997] Accot, J. and Zhai, S. (1997). Beyond fitts' law: Models for trajectory-based hci tasks. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 295–302.

[Adobe, 2012a] Adobe (2012a). Flash.

[Adobe, 2012b] Adobe (2012b). Illustrator.

[Adobe, 2012c] Adobe (2012c). Photoshop.

[Alexa et al., 2000] Alexa, M., Cohen-Or, D., and Levin, D. (2000). As-rigid-as-possible shape interpolation. *Proceedings of SIGGRAPH 2000*, pages 157–164 catmull.

[Amazon, 2012] Amazon (2012). Box office mojo.

[An and Pellacini, 2008] An, X. and Pellacini, F. (2008). Appprop: All-pairs appearance-space edit propagation. *ACM Transactions on Graphics*, 27(3):40.

[Ando and Tsuruno, 2009] Ando, R. and Tsuruno, R. (2009). Stroke by example using segmental brush synthesis. Pacific Graphics 2009 Poster Papers.

[Autodesk, 2012] Autodesk (2012). Sketchbook pro.

[Barla et al., 2005] Barla, P., Thollot, J., and Sillion, F. (2005). Geometric clustering for line drawing simplification. *In Proceedings of the Eurographics Symposium on Rendering*, pages 183–192.

[Bartolo et al., 2007] Bartolo, A., Camilleri, K. P., Fabri, S. G., Borg, J. C., and Farrugia, P. J. (2007). Scribbles to vectors: preparation of scribble drawings for CAD interpretation. *SBIM*, pages 123–130.

[Baudelaire and Gangnet, 1986] Baudelaire, P. and Gangnet, M. (1986). Advances in computer graphics i. chapter Computer-assisted animation: An overview, pages 469–498. Springer-Verlag New York, Inc., New York, NY, USA.

[Baxter et al., 2009a] Baxter, W., Barla, P., and ichi Anjyo, K. (2009a). Compatible embedding for 2d shape animation. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):867–879.

[Baxter et al., 2009b] Baxter, W., Barla, P., and ichi Anjyo, K. (2009b). N-way morphing for 2d animation. *Comput. Animat. Virtual Worlds*, 20(2):79–87.

[Baxter and Govindaraju, 2010] Baxter, W. and Govindaraju, N. (2010). Simple data-driven modeling of brushes. *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 1–8.

[Baxter and ichi Anjyo, 2006] Baxter, W. and ichi Anjyo, K. (2006). Latent doodle space. *Computer Graphics Forum*, 25(3):477–486.

[Baxter and Lin, 2004] Baxter, W. V. and Lin, M. C. (2004). A versatile interactive 3d brush model. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, PG '04, pages 319–328, Washington, DC, USA. IEEE Computer Society.

[Benard et al., 2010] Benard, P., Cole, F., Golovinskiy, A., and Finkelstein, A. (2010). Self-similar texture for coherent line stylization. *NPAR 2010: Proceedings of the 8th International Symposium on Non-photorealistic Animation and Rendering*, pages 91–97.

[Blum, 1967] Blum, H. (1967). A Transformation for Extracting New Descriptors of Shape. In Wathen-Dunn, W., editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge.

[Borgefors, 1988] Borgefors, G. (1988). Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:849–865.

[Bourdev, 1998] Bourdev, L. (1998). *Rendering Nonphotorealiztic Strokes with Temporal and Arc-length Coherence*.

[Boykov and Jolly, 2001] Boykov, Y. and Jolly, M.-P. (2001). Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. *Proceedings of Internation Conference on Computer Vision*, pages 105–112.

[Boykov and Kolmogorov, 2004] Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137.

[Boykov et al., 1998] Boykov, Y., Veksler, O., and Zabih, R. (1998). Markov random fields with efficient approximations. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–655.

[Boykov et al., 2001] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239.

[Bregler et al., 2002] Bregler, C., Loeb, L., Chuang, E., and Deshpande, H. (2002). Turning to the masters: Motion capturing cartoons. *ACM Transactions on Graphics*, 21(3):399–407.

[Bresenham, 1998] Bresenham, J. E. (1998). Seminal graphics. In *Seminal graphics*, chapter Algorithm for computer control of a digital plotter, pages 1–6. ACM, New York, NY, USA.

[Burtnyk and Wein, 1971] Burtnyk, N. and Wein, M. (1971). Computer generated key frame animation. *Journal of the SMPTE*, 80:149–153

[Burtnyk and Wein, 1976] Burtnyk, N. and Wein, M. (1976). Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *CACM*.

[Byrne, 1999] Byrne, M. T. (1999). *Animation – The Art of Layour and Storyboarding*.

[Carr et al., 2001] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. (2001). Reconstruction and representation of 3d objects with radial basis functions. *Proceedings of SIGGRAPH 2001*, pages 67–76.

[Catmull, 1978] Catmull, E. (1978). The problems of computer-assisted animation. *Proceedings of SIGGRAPH 1978*, pages 348–353.

[Chang and Yan, 1998] Chang, H.-H. and Yan, H. (1998). Vectorization of hand-drawn image using piecewise cubic bézier curves fitting. *Pattern Recognition*, 31(11):1747–1755.

[Chen et al., 1987] Chen, J. S., Huertas, A., and Medioni, G. (1987). Fast convolution with laplacian-of-gaussian masks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9:584–590.

[Chu and Tai, 2002] Chu, N. S.-H. and Tai, C.-L. (2002). An efficient brush model for physically-based 3d painting. *Computer Graphics and Applications, Pacific Conference on*, 0:413.

[Coconu et al., 2006] Coconu, L., Deussen, O., and Hege, H.-C. (2006). Real-time pen-and-ink illustration of landscapes. *NPAR 2005: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 27–35.

[Cole et al., 2012] Cole, F., Golovinskiy, A., Limpaecher, A., Barros, H. S., Finkelstein, A., Funkhouser, T., and Rusinkiewicz, S. (2012). Where do people draw lines? *Communications of the ACM*, 55(1):107–115.

[Cole et al., 2009] Cole, F., Sanik, K., DeCarlo, D., Finkelstein, A., Funkhouser, T., Rusinkiewicz, S., and Singh, M. (2009). How well do line drawings depict shape? In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, volume 28.

[Cornea et al., 2007] Cornea, N. D., Silver, D., and Min, P. (2007). Curve-skeleton properties, applications, and algorithms. *IEEE Trans. Vis. Comput. Graph.*, 13(3):530–548.

[Coyne et al., 2007] Coyne, M., Duce, D., Hopgood, B., Mallen, G., and Stapleton, M. (2007). The significant properties of vector images. Survey, System Simulation Ltd. and Oxford Brookes University.

[Curtis, 1998] Curtis, C. J. (1998). Loose and sketchy animation. *Technical Sketch SIGGRAPH 1998*.

[Dahlhaus et al., 1992] Dahlhaus, E., Johnson, D. S., Papadimitriou, C. H., Seymour, P. D., and Yannakakis, M. (1992). The complexity of multiway cuts. *Proceedings of ACM Symposium on Theory of Computing*, pages 241–251.

[de Juan and Bodenheimer, 2005] de Juan, C. N. and Bodenheimer, B. (2005). Re-using traditional animation: Methods for semi-automatic segmentation and inbetweening. *Proceedings of Symposium on Computer Animation 2006*, pages 100–102.

[Durand, 1991] Durand, C. X. (1991). The toon project: Requirements for a computerized 2d animation system. *Computers and Graphics*, 15,2:285–293.

[Farin, 2001] Farin, G. (2001). *Curves and Surfaces for CAGD*. Morgan-Kaufmann.

[Fekete et al., 1995] Fekete, J.-D., Bizouarn, E., Cournarie, E., Galas, T., and Taillefer, F. (1995). Tictactoon: a paperless system for professional 2d animation. *Proceedings of SIGGRAPH 1995*, pages 79–90.

[Freeman, 1974] Freeman, H. (1974). Computer processing of line-drawing images. *ACM Comput. Surv.*, 6(1):57–97.

[Fu et al., 2005] Fu, H., Tai, C.-L., and Au, O. K.-C. (2005). Morphing with laplacian coordinates and spatial-temporal texture. *Proceedings of Pacific Graphics 2005*, pages 100–102.

[Giesen et al., 2009] Giesen, J., Miklos, B., Pauly, M., and Wormser, C. (2009). The scale axis transform. In *Proceedings of the 25th annual symposium on Computational geometry*, SCG '09, pages 106–115, New York, NY, USA. ACM.

[Gingold et al., 2009] Gingold, Y., Igarashi, T., and Zorin, D. (2009). Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics*, 28(5):148.

[Gould, 1993] Gould, S. (1993). Looney tuniverse: Ther is a crazy king of physics at work in the world of cartoons. *New Scientist Magazine*, 1905.

[Grabli et al., 2004] Grabli, S., Durand, F., and Sillion, F. (2004). Density measure for line-drawing simplification. *Pacific Conference on Computer Graphics and Applications*, pages 309–318.

[Hahn, 2008] Hahn, D. (2008). *The Alchemy of Animation*. Disney Editions.

[Hall, 2010] Hall, S. (2010). *Best Digital Brushes for Photoshop: A unique directory of over 4,000 digital brush effects, and how to achieve them*. Peachpit Press, Berkeley, CA, USA, 1st edition.

[Hilaire and Tombre, 2006] Hilaire, X. and Tombre, K. (2006). Robust and accurate vectorization of line drawings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(6):890–904.

[Hsu and Lee, 1994] Hsu, S. C. and Lee, I. H. H. (1994). Drawing and animation using skeletal strokes. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 109–118, New York, NY, USA. ACM.

[Hsu et al., 1993] Hsu, S. C., Lee, I. H. H., and Wiseman, N. E. (1993). Skeletal strokes. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*, UIST '93, pages 197–206, New York, NY, USA. ACM.

[Igarashi and Moscovich, 2005] Igarashi, T. and Moscovich, T. (2005). As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics (*.

[Janssen and Vossepoel, 1997] Janssen, R. D. and Vossepoel, A. M. (1997). Adaptive vectorization of line drawing images. *Computer Vision and Image Understanding*, 65(1):38–56.

[Johnston and Thomas, 1995] Johnston, O. and Thomas, F. (1995). *The Illusion of Life*.

[Kalnins et al., 2003] Kalnins, R. D., Davidson, P. L., Markosian, L., and Finkelstein, A. (2003). Coherent stylized silhouettes. *Proceedings of SIGGRAPH 2003*, pages 856–861.

[Kalnins et al., 2002] Kalnins, R. D., Markosian, L., Meier, B. J., Kowalski, M. A., Lee, J. C., Davidson, P. L., Webb, M., Hughes, J. F., and Finkelstein, A. (2002). Wysiwyg npr: drawing strokes directly on 3d models. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 755–762.

[Kawabata, 1997] Kawabata, N. (1997). Depth perception in simple line drawings. *Perceptual and motor skills*, 85(3 Pt 1):1043–1057.

[Keane, 1987] Keane, G. (1987). Dynamics of animated drawings. Class Handout.

[Kirbas and Quek, 2000] Kirbas, C. and Quek, F. K. (2000). A review of vessel extraction techniques and algorithms. *ACM Computing Surveys*, 36:81–121.

[Kleinberg and Tardos, 2005] Kleinberg, J. and Tardos, E. (2005). *Algorithm Design*.

[Kochanek et al., 1982] Kochanek, D. H., Bartels, R., and Booth, K. S. (1982). A computer system for smooth key frame animation.

[Kort, 2002] Kort, A. (2002). Computer aided inbetweening. *NPAR 2002: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 125–132, http://doi.acm.org/10.1145/508530.508552.

[Lakshmi and Punithavalli, 2009] Lakshmi, J. K. and Punithavalli, M. (2009). A survey on skeletons in digital image processing. *Proceedings of the International Conference on Digital Image Processing*, pages 260–269.

[Lam et al., 1992] Lam, L., Lee, S.-W., and Suen, C. Y. (1992). Thinning methodologies - a comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(9):869–885.

[Lank and Saund, 2005] Lank, E. and Saund, E. (2005). Sloppy selection: Providing an accurate interpretation of imprecise selection gestures. *Computers and Graphics*, 29(4):490–500.

[Laybourne, 1998] Laybourne, K. (1998). *The Animation Book: A Complete Guide to Animated Filmmaking–From Flip-Books to Sound Cartoons to 3- D Animation*.

[Lecot and Levy, 2006] Lecot, G. and Levy, B. (2006). Ardeco: Automatic region detection and conversion. *EGSR'06*, pages 349–360.

[Lee et al., 2011] Lee, Y. J., Zitnick, C. L., and Cohen, M. F. (2011). Shadowdraw: real-time user guidance for freehand drawing. *ACM Transactions on Graphics*, 30:27.

[Levin et al., 2004] Levin, A., Lischinski, D., and Weiss, Y. (2004). Colorization using optimization. *ACM Transactions on Graphics*, 23(3):689–694.

[Levoy, 1977] Levoy, M. (1977). A color animation system: based on the multiplane technique. *Proceedings of SIGGRAPH 1977*, pages 65–71.

[Lie et al., 2010] Lie, D., Chen, Q., Yu, J., Gu, H., Tao, D., and Seah, H. S. (2010). Stroke correspondence construction for vector-based 2d animation inbetweening. *Proceedings of Computer Graphics International 2010*.

[Liu and Dori, 1998] Liu, W. and Dori, D. (1998). A survey of non-thinning based vectorization methods. *SSPR/SPR*, pages 230–241.

[Llamas, 2003] Llamas, I. (2003). Twister: a space-warp operator for the two-handed editing of 3d shapes. *Proceedings of SIGGRAPH 2003*, pages 663–668.

[Madeira et al., 1996] Madeira, J. S., Stork, A., and Gross, M. H. (1996). An approach to computer-supported cartooning. *The Visual Computer*, 12:1–17. 10.1007/BF01782215.

[Mcivor, 2000] Mcivor, A. M. (2000). Background subtraction techniques. *Proceedings of Image and Vision Computing, Auckland, New Zealand, 2000.*

[Melikhov et al., 2004] Melikhov, K., Tian, F., Seah, H. S., Chen, Q., and Qiu, J. (2004). Frame skeleton based auto-inbetweening in computer assisted cel animation. *CW '04: Proceedings of the 2004 Intl. Conference on Cyberworlds*, pages 216–223.

[Miklos, 2011] Miklos, B. (2011). Mesecina: A software to visualize the medial axis and related computational geometry structures.

[Miller, 1957] Miller, D. (1957). *The Story of Walt Disney*.

[Miura et al., 1967] Miura, T., Iwata, J., and Tsuda, J. (1967). An application of hybrid curve generation: cartoon animation by electronic computers. *AFIPS '67 (Spring): Proceedings of the April 18-20 1967, spring joint computer conference*, pages 141–148.

[Noris, 2008] Noris, G. (2008). Computer Assisted Cartooning. Master's thesis, Eidgenössische Technische Hochschule Zürich, Zürich, Switzerland.

[Noris et al., 2011] Noris, G., Sykora, D., Coros, S., Whited, B., Simmons, M., Hornung, A., Gross, M. H., and Sumner, R. (2011). Temporal noise control for sketchy animation. *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, pages 93–98.

[O'Donnell, 1980] O'Donnell, M. (1980). The cartoon laws of physics. *IEEE Institute*, 18:12.

[Orzan et al., 2008a] Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., and Salesin, D. (2008a). Diffusion curves: A vector representation for smooth-shaded images. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, volume 27.

[Orzan et al., 2008b] Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., and Salesin, D. (2008b). Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics*, 27(3).

[Öztireli et al., 2011] Öztireli, A. C., Uyumaz, U., Popa, T., Sheffer, A., and Gross, M. (2011). 3d modeling with a symmetric sketch. EG.

[Patterson and Willis, 1994] Patterson, J. W. and Willis, P. J. (1994). Computer assisted animation: 2d or not 2d? *The Computer Journal*, 37(10):829–839.

[Piccardi, 2004] Piccardi, M. (2004). Background subtraction techniques: a review. *Systems, Man and Cybernetics*, 4:3099–3104.

[Porter and Duff, 1984] Porter, T. and Duff, T. (1984). Compositing digital images. *SIGGRAPH Comput. Graph.*, 18(3):253–259.

[Powell and Rossignac, 2008] Powell, A. and Rossignac, J. (2008). Screwbender: Smoothing piecewise helical motions. *IEEE Comput. Graph. Appl.*, 28:56–63.

[Reeves, 1981] Reeves, W. (1981). Inbetweening for computer animation utilizing moving point constraints. *Proceedings of SIGGRAPH 1981*, pages 263–270.

[Richard, 2002] Richard, W. (2002). *The Animator's Survival Kit*. Faber and Faber, 1st edition.

[Robertson, 1994] Robertson, B. (1994). Disney lets caps out of the bag. *Computer Graphics World*, 17(7):58–64.

[Robertson, 2001] Robertson, B. (2001). Digital toons. *Computer Graphics World*, 18(8):58–64.

[Rossignac and Vinacua, 2011] Rossignac, J. and Vinacua, A. (2011). Steady affine motions and morphs. *ACM Transactions on Graphics (to appear)*.

[Saito and Nakajima, 1999] Saito, S. and Nakajima, M. (1999). 3d physics-based brush model for painting. In *ACM SIGGRAPH 99 Conference abstracts and applications*, SIGGRAPH '99, pages 226–, New York, NY, USA. ACM.

[Salisbury et al., 1997] Salisbury, M. P., Wong, M. T., Hughes, J. F., and Salesin, D. (1997). Orientable textures for image-based pen-and-ink illustration. *Proceedings of the ACM SIGGRAPH Conference (SIGGRAPH-97)*, pages 401–406.

[Saund, 2003] Saund, E. (2003). Finding perceptually closed paths in sketches and drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4):475–491.

[Saund et al., 2004] Saund, E., Fleet, D., Larner, D., and Mahoney, J. (2004). Perceptually-supported image editing of text and graphics. *ACM Transactions on Graphics*, 23(3):728–728.

[Schmid et al., 2011] Schmid, J., Senn, M. S., Gross, M., and Sumner, R. W. (2011). Overcoat: an implicit canvas for 3d painting. *ACM Trans. Graph.*, 30:28:1–28:10.

[Seah and Tian, 2000] Seah, H. S. and Tian, F. (2000). Computer-assisted coloring by matching line drawings. *The Visual Computer*, 16(5):289–304.

[Sebastian et al., 2003] Sebastian, T. B., Klein, P. N., and Kimia, B. B. (2003). On aligning curves. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(1):116–125.

[Sederberg and Greenwood, 1992] Sederberg, T. W. and Greenwood, E. (1992). A physically based approach to 2d shape blending. *Proceedings of SIGGRAPH 1992*, pages 25–34.

[Senthilkumaran and Rajesh, 2009] Senthilkumaran, N. and Rajesh, R. (2009). Edge detection techniques for image segmentation — a survey of soft computing approaches. *International Journal of Recent Trends in Engineering*, 1(2).

[Shesh and Chen, 2008] Shesh, A. and Chen, B. (2008). Efficient and dynamic simplification of line drawings. *Proceedings of Eurographics, Computer Graphics Forum*, 27(2):537–545.

[SiSoft, 2010] SiSoft (2010). Wintopo. *wintopo.com*.

[Steve, 2006] Steve, W. (2006). *Digital Compositing for Film and Video*. Focal Press, 2nd edition.

[Sumner et al., 2005] Sumner, R., Zwicker, M., Gotsman, C., and Popovic, J. (2005). Mesh-based inverse kinematics. *ACM Transactions on Graphics*, 24(3):488–495.

[Sun et al., 2007a] Sun, J., Liang, L., Wen, F., and Shum, H.-Y. (2007a). Image vectorization using optimized gradient meshes. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA. ACM.

[Sun et al., 2007b] Sun, J., Liang, L., Wen, F., and Shum, H.-Y. (2007b). Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics*, 26(3):11.

[Surazhsky and Elber, 2002] Surazhsky, T. and Elber, G. (2002). Metamorphosis of planar parametric curves via curvature interpolation. *Intl. J. of Shape Modeling*, 8(2):201–216.

[Sykora et al., 2005] Sykora, D., Burianek, J., and Zara, J. (2005). Video codec for classical cartoon animations with hardware accelerated playback. *Proceedings of International Symposium on Visual Computing*, pages 43–50.

[Sykora et al., 2009a] Sykora, D., Dingliana, J., and Collins, S. (2009a). As-rigid-as-possible image registration for hand-drawn cartoon animations. *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, pages 25–33.

[Sykora et al., 2009b] Sykora, D., Dingliana, J., and Collins, S. (2009b). Lazybrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum*, 28(2):599–608.

[Sykora et al., 2010] Sykora, D., Sedlacek, D., Jinchao, S., Dingliana, J., and Collins, S. (2010). Adding depth to cartoons using sparse depth (in)equalities. *Computer Graphics Forum*, 29(2):615–623.

[Telotte, 2010] Telotte, J. P. (2010). *Animating Space: From Mickey to WALL-E*. University Press of Kentucky.

[Thompson, 1992] Thompson, D. W. (1992). *On Growth and Form*.

[ToonBoom, 2010] ToonBoom (2010). Harmony.

[Valentine, 2012] Valentine, S. (2012). *The Hidden Power of Blend Modes in Adobe Photoshop*.

[Veltkamp, 2001] Veltkamp, R. C. (2001). Shape matching: similarity measures and algorithms. *Shape Modeling and Applications, SMI 2001 Intl. Conference on.*, pages 188–197.

[von Goldberg et al., 2011] von Goldberg, A., Hed, S., Kaplan, H., Tarjan, R. E., and Werneck, R. F. (2011). Maximum flows by incremental breadth-first search. *ESA*, pages 457–468.

[Wagner and Fischer, 1974] Wagner, R. and Fischer, M. (1974). The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173.

[Wang et al., 2008] Wang, Y., Xu, K., Xiong, Y., and Cheng, Z.-Q. (2008). 2d shape deformation based on rigid square matching. *Computer Animation and Virtual Worlds*, 19(3–4):411–420.

[Warnock, 1982] Warnock, J. (1982). Postscript. Marketed by Adobe Systems.

[WDAS and Pixar, 1989] WDAS and Pixar (1989). Caps: a computer animation production system.

[Wei et al., 2010] Wei, J., Wang, C., Yu, H., and Ma, K.-L. (2010). A sketch-based interface for classifying and visualizing vector fields. *Proceedings of IEEE Pacific Visualization Symposium*, pages 129–136.

[Whitaker et al., 2009] Whitaker, H., Sito, T., and Halas, H. (2009). *Timing for Animation*. Focal Press.

[Whited et al., 2010] Whited, B., Noris, G., Simmons, M., Sumner, R., Gross, M. H., and Rossignac, J. (2010). Betweenit: An interactive tool for tight inbetweening. *Computer Graphics Forum (Eurographics 2010 Proceedings)*, 29(2).

[Whited et al., 2009] Whited, B., Rossignac, J., Slabaugh, G., Fang, T., and Unal, G. (2009). Pearling: Stroke segmentation with crusted pearl strings. *Pattern Recognition and Image Analysis*, 19(2):277–283.

[Wilson and Ma, 2004] Wilson, B. and Ma, K.-L. (2004). Rendering complexity in computer-generated pen-and-ink illustrations. *NPAR 2004: Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering 2004 Annecy, France, June 7-9, 2004*, pages 129–137.

[Winkenbach and Salesin, 1994] Winkenbach, G. and Salesin, D. (1994). Computer-generated pen-and-ink illustration. *Proceedings of SIGGRAPH 1994*, pages 91–100.

[Wolin et al., 2007] Wolin, A., Smith, D., and Alvarado, C. (2007). A pen-based tool for efficient labeling of 2D sketches. *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 67–74.

[Xia et al., 2009] Xia, T., Liao, B., and Yu, Y. (2009). Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Transactions on Graphics*, 28(5):1–10.

[Zhang et al., 2009] Zhang, S.-H., Chen, T., Zhang, Y.-F., Hu, S.-M., and Martin, R. R. (2009). Vectorizing cartoon animations. *IEEE Trans. Vis. Comput. Graph.*, 15(4):618–629.

[Zou and Yan, 2001] Zou, J. J. and Yan, H. (2001). Cartoon image vectorization based on shape subdivision. *Computer Graphics International*, pages 225–231.

[Zwicker et al., 2002] Zwicker, M., Pfister, H., van Baar, J., and Gross, M. H. (2002). Ewa splatting. *IEEE Trans. Vis. Comput. Graph.*, 8(3):223–238.