COMPUTATIONAL DESIGN TOOLS FOR CARTOONING

PRADEEP KUMAR JAYARAMAN

PRADEEP KUMAR JAYARAMAN

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2017

Acknowledgments

I would like to convey my deepest appreciation to people who have helped me so far in this PhD study.

The love, support, and encouragement provided by my wife Aarthi, and parents Hemalatha and Jayaraman, have been instrumental throughout these years. My brother Ramkumar, encouraged me to undertake this PhD and has always been supportive in every possible way. I would like to take this opportunity to thank them for everything.

I am fortunate to have worked with two PhD supervisors. Prof. Philip Fu, my first PhD supervisor provided a wonderful environment with several opportunities for research and study. His support and guidance throughout these years have been substantial and crucial to the successes obtained so far, and kept this research on track. I have learnt and benefited much from him.

Prof. Jianmin Zheng adopted me as his student during the last year of my PhD. He has been extremely supportive and helped me in several problems, both in research and otherwise, and I am deeply thankful to him for that. His extensive knowledge in computer graphics and mathematics has been inspirational to me, and our many long discussions have been invaluable for my learning.

I am grateful to the coauthors that made my publications possible, especially Dr. Chih-Kuo Yeh and Prof. Tong Yee Lee from National Cheng Kung University, Taiwan. I have learnt much by collaborating with them and I'm thankful for the opportunities.

I am thankful to my Thesis Advisory Committee, Prof. Tat-Jen Cham and Prof. Lap-Pui Chau, for reviewing my research objectives and achievements, and providing valuable comments.

My fellow lab mates and colleagues made my life easier by providing valuable advise, help, support and company. I am thankful to all the pleasant conversations and smiles we shared.

Finally, I would like to thank Nanyang Technological University for the excellent funding and infrastructure to carry out research.

Contents

List of Figures

List of Tables

Abstract

Cartoons have a long and rich history in entertainment industry. While traditional methods to create 2D cartoons are prohibitively tedious, the use of computers has greatly simplified and improved the efficiency in cartoon production. Still, a majority of established software tools for cartooning use the computer merely as a canvas. Recently, there is an increased research interest in developing computer-assisted methods to simplify the tedious manual work performed by artists, thereby allowing them to focus on creative endeavours. Such works intelligently analyze artwork with computational models, and have been applied to problems such as line drawing vectorization, inbetweening, coloring, and animation. In this research, we mainly focus on three areas that have not received much attention: drawing and digitizing freeform artwork from paper medium, automatically generating shading on 2D line drawings, and semantically manipulating or animating existing cartoon images. These areas are unified under the roof of efficient computational cartoon techniques. We also present an application of the developed methods for interactive 3D reconstruction of organic objects from natural images.

We first explore drawing and digitizing 2D line drawings from a paper medium. We present a novel method that directly recognizes and vectorizes strokes drawn on paper using everyday pens/pencils, in front of a commodity webcam. Compared to offline methods where the paper is scanned and then the strokes are vectorized, our online method analyzes the entire drawing process and captures both spatial and temporal information of the strokes similar to a digital tablet. By this, we can reconstruct strokes as drawn by the user, without ambiguities caused by stroke junctions, intersections, etc. Our method may facilitate the development of various multimedia applications such as line drawing vectorization, video scribing, and pen input interface. We discuss potential future directions to scale up this work to enable usage in demanding applications.

Second, we consider the problem of generating shading on detailed 2D line drawings with wrinkles. Shading is a tedious process in comic/cartoon production given the volume of contents that artists have to prepare regularly over tight schedule. While we can automate shading with the presence of geometry, 2D artists are very unlikely to model the geometry for every single drawing which is actually also a very tedious task. In this work, we aim to

automate shading generation by analyzing the shapes, interactions, and spatial arrangement of wrinkle strokes in a clean line drawing. By this, artists can focus more on the design rather than the tedious manual work, and experiment with different shading under different light conditions. To achieve this, we propose a novel technique that contains three key technical contributions. First, we model five perceptual cues by exploring relevant psychological principles to estimate the local depth profile around strokes automatically. Second, we formulate stroke interpretation as a global optimization model that balances different interpretations suggested by the perceptual cues and minimizes the discrepancy. Third, we develop a wrinkle-aware inflation method to generate a height field surface that can support shading. We demonstrate various styles of shading effects including 3D-like soft shading and 2D manga style shading on a number of line drawings.

Third, we observe that a huge amount of 2D cartoon images are digitally available today. Hence, there is a significant artistic value if we can somehow re-use this content and enable end-users to manipulate or animate these images to create intriguing results with minimum effort. Regarding this, we undertake the problem of modeling the hairs in a given cartoon image with consistent layering and occlusion, so that we can produce various visual effects from just a single image. We propose a novel 2.5D modeling approach to deal with this problem. Given an input image, we segment the hairs of the cartoon character into regions of hair strands. Then, we apply our novel layering metric to automatically estimate the depth ordering among the hair strands, employ our hair completion method to fill the occluded regions, and create a 2.5D model of the cartoon hair. By this, we can produce various visual effects, such as windblown hair animations and local hair editing.

Finally, we present an application of the developed methods to the problem of interactively reconstructing high-relief 3D geometry from a single photo. We begin by constructing a 2.5D model by segmenting the image into regions, followed by layering and completion that can handle three common cases of occlusions. Next, users can interactively markup slope and curvature cues on the image to guide our constrained optimization model to inflate and lift up the image regions to 3D. Finally, we stitch and optimize the inflated layers to produce a high-relief 3D model. Compared to previous work, we can generate high-relief geometry that can plausibly support large viewing angles, and handle complex organic objects with varying shape profiles.

CHAPTER 1

Introduction

1.1 Background

Cartoons have a long and rich history in entertainment industry. Recently, there has been an explosion of cartoons and comics in the form of feature-length films, comic books, graphics novels, etc. Traditionally, creating 2D cartoons is a very prohibitively tedious process, where artists have to draw, ink, shade and color every single frame of an animated cartoon or comic book page by hand. The use of computers has greatly improved the efficiency in cartoon production; various software tools that mimic the traditional manual pipeline have been developed, with numerous capabilities such as, resolution-independent representation, stroke-based manipulation, intuitive coloring and editing, digital compositing, in-betweening between character poses, and creating special effects, which require tedious effort in the traditional approach.

Even though there exist decades of research in cartoon production and manipulation, there is still tremendous scope for improvement. It is not difficult to gauge that the cartooning industry would benefit better if computers could better assist in simplifying tedious steps in the production process. This however is not a simple process since computers cannot understand higher level semantics like humans. For example, it is very easy for humans to group a set of pixels into semantic regions, or estimate the 3D shape of an object merely from abstract 2D sketches. These are ill-posed problems for computers and decades of research have been devoted to attempts in solving such problems. On the other hand, computers have extensive computational power that can help in solving complex problems if represented in the right way.

Most of the established digital tools in cartooning pipeline employ computers merely as a drawing canvas to create and edit cartoon content. They have not taken into consideration machine intelligence and the extensive computational power of modern computers to solve higher level semantic problems that arise in cartooning. In order efficiently employ computers as tools in simplifying cartoon production, they should be able to perceive and understand

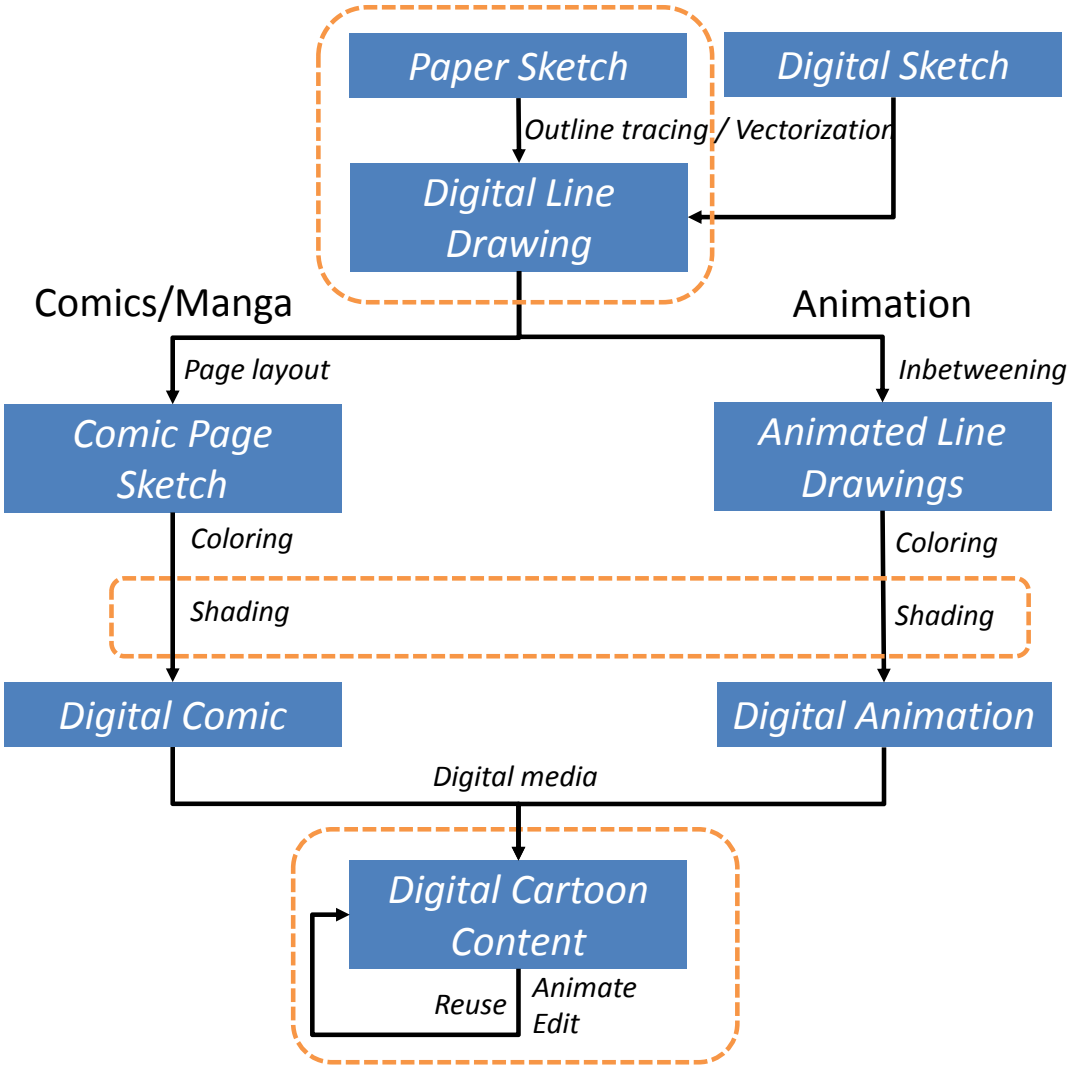


Figure 1.1: *A general pipeline of the cartooning process for comics/manga and animations. The dotted orange boxes indicate the focus of works explored in this thesis.*

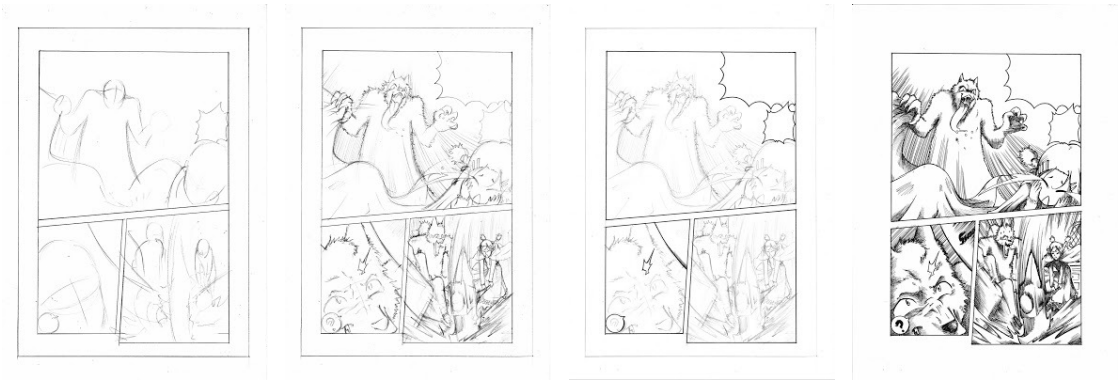
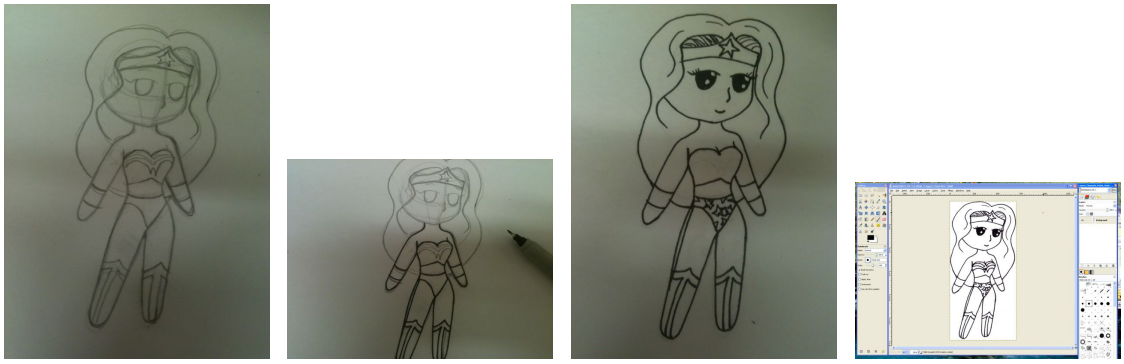


Figure 1.2: *Comic page pipeline [1] (a) Panel layout. (b) Crude sketch of contents. (c) Trace a cleaned up outline copy (inking). (d) Enrich by adding shading effects such as hatching and screening.*

the semantics that humans can easily infer from images or drawings, to perform simplify or automatically perform the corresponding works. With this objective, we primarily aim to explore computational methods and models that can make use of semantic information available in cartoon content, to enable computers to analyze and automate certain tedious tasks in areas related to cartooning. Regarding this, we propose three methods that can be used to simplify certain stages in cartoon production as explained in detail in the forthcoming paragraphs.

To design such computational tools, we first need to understand the cartoon production pipeline and the artists' task in each stage. Figure 1.1 shows such a typical pipeline that generally describes both comic and cartoon production scenarios. The overall pipeline for comics and cartoons is quite similar, particularly in the initial stages. To create a comic book page, the artists follow a sequence of steps as illustrated in Figure 1.2. This begins with sketching the panel layout on paper. Next, artists lay out characters in the scene using crude sketch lines to experiment with the look of each panel. After this, a traditional artist will use the sketch as a guide to create a clean copy into a paper with a pen. This process is usually referred to as *inking* and is commonly done by laying a paper on top of the sketch on a light table. Digital artists on the other hand, usually clean up the drawing, and scan the paper. When the drawing is required in vector graphics format, artists use the scanned sketch as a base layer and carefully trace a clean drawing on top of it using a stylus or mouse. This step requires a duplication of effort but is necessary for vectorization the



(a) (b) (c) (d)

Figure 1.3: *Typical sketching and digitization workflow [2]. The line drawing is first sketched on paper with a pencil (a), followed by inking with a pen to highlight the outline (b). The pencil sketch is then erased to obtain a clean line drawing (c). This is scanned into the computer (d) to obtain a raster image. Artists may directly work with this or choose to vectorize by tracing over with a stylus or mouse if necessary.*

line drawing. Furthermore, to enhance the appearance and better depict the objects in the scene, comic/manga artists usually perform shading and optionally coloring. Shading is primarily applied to achieve a 3D perception by highlighting the shadows or unlit regions as shown in Figure 1.4. To achieve this, artists traditionally lay screens on top of the clean line drawing and shape them using pen-like utility knives. This can be mimicked digitally using layer masks and is the most prevalent screening technique in recent times. However, regardless of whether artists use simple gray color or pattern based screening, the shading step remains a tedious process that has to be manually performed for each drawing. These steps are repeated for each panel in each page to create a complete comic book. For animation, the initial process of sketching and digitizing drawings is similar. After the various frames are digitized and inbetweened either manually or with software, artists then shade and color the frames; hatching or pattern based shading is seldom used in animations. Techniques such as onion skinning may be used to simplify coloring of multiple frames. Additionally, to keep the animation effort tractable, foreground objects are drawn in separate transparent sheets of paper called *cels*; the digital equivalent of this is a layer with transparent background. The background layer is often a panoramic static image and all the layers are composited and rendered using a physical or virtual camera to general the final animation.

Sketching and digitizing of strokes is usually the very first step for creating

any kind of cartoon content as discussed above. While traditional cartooning is based on paper based physical media, the advantages of computers have pushed more and more artists to shift to digital media. With the increasing ubiquity of drawing tablets and digital pens/styluses, many if not all artists have started to sketch on computers directly. These devices have many attractive advantages such as the ability to capture not only the drawn strokes, but also several important information such as pressure sensitivity, pen angle tilt, sketching speed, timestamps for stroke samples, etc. However, drawing on tablets is a significantly different experience for the user compared to intuitive drawing on paper. For this reason, several artists prefer to first sketch on paper and use a scanner to digitize the drawing as shown in Figure 1.3. Since this produces a raster image, often a digital vector drawing is created by manually tracing over the image with a stylus. Even though there are vectorization software that can extract strokes from the raster images, this is a very challenging and ill posed problem does not always provide satisfactory results. The efficiency of this step can be improved if the computer can help to save this duplicate effort of manual tracing.

Once the clean sketches are digitized, an important artistic step in enhancing the visual appearance of these sketches is shading. Shading can dramatically change the atmosphere of a line drawing, and artists currently achieve this by manually screening, hatching, or even black-inking a clean line drawing. Figure 1.4 illustrates the visual impact that shading can have on 2D drawings. However, this can be laborious due to the repetitive requirements of shading in each comic book panel or each animation frame, and saving labor and reducing costs is an important requirement in any industry. While such shading process can be automated to a large extent with the presence of 3D geometry, artists are very unlikely to model the geometry for every single drawing. Therefore, it would be desirable to infer certain geometry information directly from the artists clean sketches, and help to automate the shading, and save the artists from the tedious shading process, so that they can focus on the design.

Cartoon animations are typically created from scratch, by sketching and digitizing each frame, cleaning them up, coloring and compositing. With the huge amount of digital cartoon content available today, there is great value in reusing this content to achieve visually pleasing animation with minimum effort. Unfortunately, this is a very challenging problem since the image is in 2D, while one needs at least a 2.5D understanding of the image contents to animate

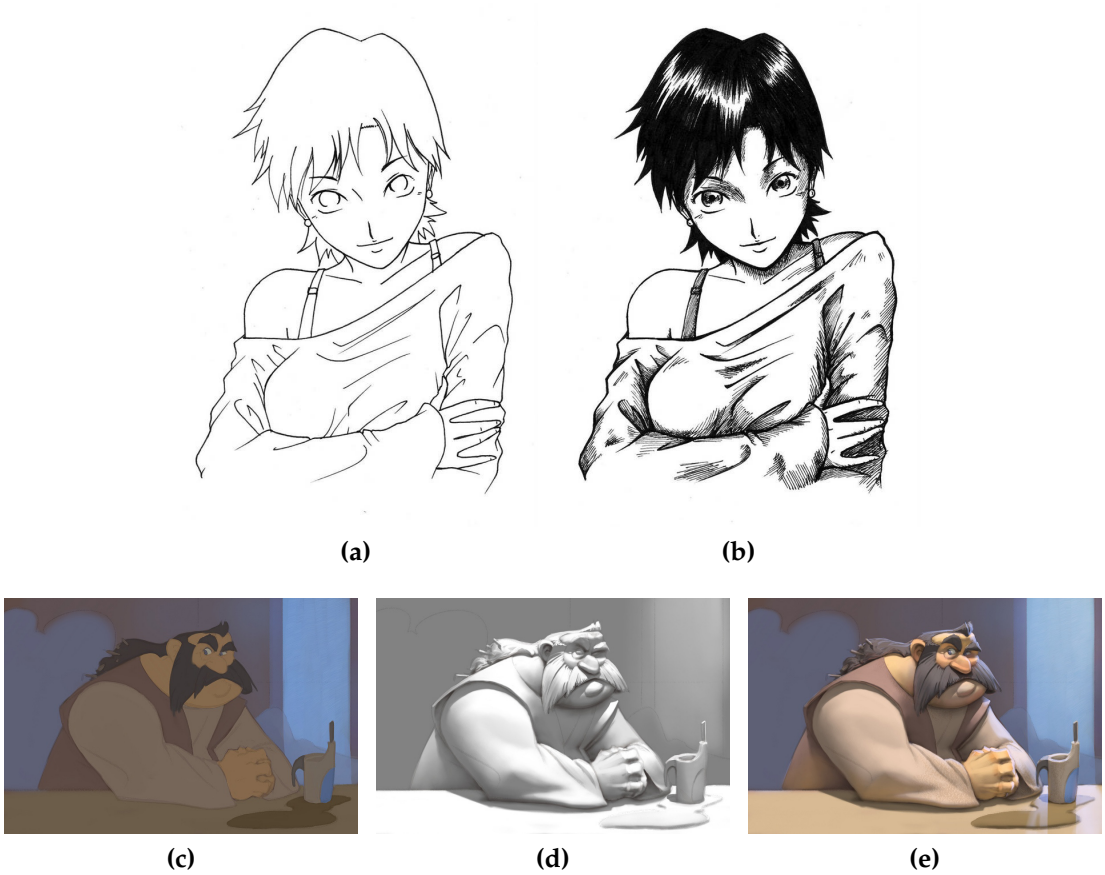


Figure 1.4: *Effect of shading. A clean line drawing (a) has been shaded in 2D manga style using hatching patterns in (b) [3]. Note the detailed shading close to wrinkles in the cloth. Shading can enrich flat colored drawing as in (c). Here, the artist has manually created the grayscale shading image (d) and multiplied it with the flat color image to obtain the final result (e) [4]. This shows that the presence of 3D geometry can aid the shading process.*

the contents; i.e., recover the various semantic regions in the image, estimate their relative depth orderings and complete the hidden/occluded portions of each region so that holes do not appear when animating. However, humans can easily perceive the relative depth of regions from a single image and fairly estimate the shapes of objects behind occlusions. Hence, extracting such perceptual information from the cartoon image will be useful for enabling the creation of intriguing animations or manipulations with less effort.

1.2 Objectives & Contributions

Research Objectives. Our main focus in this thesis is in formulating methods that can utilize semantic information to simplify tedious steps in cartoon production. In detail, the following are the research objectives for this thesis:

- To explore an online approach to vectorizing line drawings, that can directly capture and vectorize strokes as and when users sketch on paper, by analyzing the drawing process with a commodity camera.
- To automate shading generation on 2D line drawings by analyzing the strokes in clean sketches considering their shapes, their spatial arrangements and relationships to neighboring strokes and ensuring global consistency of geometric features in the line drawing.
- To support animation and manipulation of cartoon content present in a single image by extracting 2.5D information, and developing tools to enable users to perform these actions with less effort.

Contributions. A detailed background study on research related to problems and applications in cartooning along with analysis on related subproblems is first presented. We then develop three main novel methods that can improve the efficiency of cartoon production in certain stages of the pipeline as detailed below.

As a first contribution, we present a novel pen interface that can vectorize strokes drawn with everyday pens/pencils on a paper by analyzing the drawing process with a camera. Vectorization methods that are used to digitize the strokes drawn on paper are almost always offline methods. By offline, we

mean that the vectorization is carried out on a completed line drawing which is scanned into a raster image. Due to this, the problem of recovering strokes becomes ill-posed and ambiguous especially due to rasterization artifacts and junctions in the drawing, resulting in long strokes to be broken into multiple short segments that affect editability. Online methods such as drawing on a tablet capture the entire drawing process including temporal information of strokes, and do not face these problems; however, such a method does not exist for paper based drawings. This forces artists to manually trace the paper drawings using a mouse/stylus and a digital tablet. In this work, we experiment with an online method to vectorize clean line drawings drawn on paper. The primary advantage of our method is that it allows users to draw on paper with everyday pens/pencils and does not have an additional step of tracing to digitize the strokes. Our method observes the entire drawing process using a fixed camera (commodity webcam in our experiments), and analyzes the drawn strokes capturing both spatial and temporal information. This is challenging particularly due to occlusions caused by the hand and pens while drawing, and we propose methods to handle these problems. By this, we can avoid various ambiguities, enhance the vectorization quality, and recover the stroke drawing order as in the case of a digital tablet, but without the need for expensive hardware. This type of online method could potentially be applied in the very first step of cartooning where artists clean up their initial sketches. It can also been used in other multimedia applications, such as pen input interface or video scribing where the stroke drawing process can be rendered to produce interesting visual effects.

Our second contribution lies in the automatic shading of line drawings by estimating depth suggested by 2D lines and partial 3D reconstruction. The main challenge in shading 2D line drawings is the lack of 3D geometric information of the underlying shapes; this problem is particularly highly challenging in the presence of multiple wrinkle strokes in a drawing. We propose an automatic shading generation method that works by first analyzing the strokes in the drawing to recover their depth profiles, and guide an inflation method to lift the 2D line drawing into 3D. We model five perceptual cues: T-junctions, convexity, proximity, continuity and regularity gestalts to estimate the local depth profile around strokes. Since some perceptual cues could suggest conflicting information, we then formulate stroke interpretation as a global optimization model that simultaneously balances different interpretations suggested by the

perceptual cues and minimizes the interpretation discrepancy. Different from existing methods, our method is completely *automatic*, although users could provide their input if necessary. Further, we develop a wrinkle-aware inflation method to generate a 3D surface guided by the result of the previous step. This works in two steps where we first propagate the sparse depth profiles recovered across each stroke to the interior of the line drawing, followed by reconstructing the surface that matches these depth profiles. Compared to existing inflation methods, our method particularly focuses on interior strokes and therefore produces more detailed geometry that can generate perceptually valid and visually pleasing shading. Furthermore, our inflation method can reconstruct two different types of shadings: soft shading that imbues a 3D-like feel, and manga/cel shading that retains the 2D feel of the drawing. We explain these shading styles in detail and demonstrate our method on a number of input line drawings. This method could be potentially applied in the shading stage of cartoon production or in visually enhancing existing cartoons.

Our third contribution is in the 2.5D modeling of hairs in a single cartoon image with goal of animating or manipulating the hair strands. By inferring the relative depth orderings among various hair strands, and completing the occluded portions of the hair, we are able to generate wind blown hair animations and support local hair editing, given just a single image as input. Inspired by the ability of humans to easily perceive the depth ordering among adjacent regions, we propose a novel layering metric that can analyze T-junctions and overlapping area among adjacent regions to accurately estimate the depth orderings. These two cues complement each other and when combined are able to automatically infer the depth orderings for a wide variety of occlusion scenarios. After that, we apply our completion method to fill the occluded portions of each hair; this is necessary since holes could show up otherwise in the occluded hairs when animated. By this, we are able to generate a complete 2.5D model of the cartoon hair that can be animated or manipulated to generate various visual effects, e.g., we develop a simplified fluid simulation model to produce wind blown hair animations on a still cartoon character which is very common in cartoons. Our method is largely automatic and retains the 2D look and feel of the original image when animated. It has a huge potential in realizing the artistic value of existing cartoon content in static images by easily reusing them for creating visually intriguing

results.

Our fourth contribution extends the planar 2.5D model from the previous work into 3D using an interactive inflation approach. This results in a high-relief 3D model, that can depict a large range of depth within each layer, from just a single image of complex objects. We demonstrate results on a wide range of natural images, and discuss several future applications for the cartooning pipeline.

To summarize we make the following contributions in this thesis:

- We present a novel attempt in designing an online vectorization method that interactively recognizes and vectorizes hand-drawn strokes on paper in front of a commodity camera.
- We make the first attempt in modeling perceptual cues from Gestalt psychology with the goal of recovering a globally consistent 3D shape from a single 2D line drawing. We apply this to the problem of automatically shading cartoon line drawings.
- We propose a 2.5D modeling framework to animate the hairs in a cartoon image, where we develop a novel layering metric to estimate the depth ordering among adjacent regions, and completion method to fill in the occluded hair strands. Our method is largely automatic and enables animating or manipulating the content of existing cartoon images with minimum user input.
- We extend our previous work on 2.5D modeling to interactively support non-planar 2.5D layers, that enable high-relief 3D reconstruction of a wide range of organic objects from single input images.

1.3 Organization

The rest of the thesis is organized as follows. We first provide a detailed review of various computational models and methods developed by the research community to simplify cartoon/comic production in Chapter 2, while particularly focusing on works related to our applications. In Chapter 3, we present our novel camera-based vectorization method that can directly vectorize strokes drawn on paper in front of a commodity camera. In Chapter 4, we

propose our Gestalt psychology based 2D line drawing analysis framework that can automatically reconstruct a proxy 3D geometry to support shading. In Chapter 5, we describe our 2.5D cartoon hair modeling framework that can animate the hair of a cartoon character from just a single image as input. In Chapter 6, we present an application where our developed techniques are integrated to provide interactive high-relief reconstruction of organic objects from a single image or photo. Finally, Chapter 7 concludes the thesis and points out a few research directions for future work.

CHAPTER 2

Related Work

In this chapter, we first provide some broad background on the traditional cartooning pipeline. Next, we discuss how computer-assisted cartooning methods have helped to simplify various steps in the traditional pipeline, and survey some prominent previous works in this domain. After that, we focus more closely on related works specific to the problems considered in this research.

2.1 Traditional Cartooning

Cartooning is the process of creating cartoon images or videos, which are two-dimensional illustrations with non-realistic or semi-realistic visual appearance. The artists who create cartoons are called cartoonists, and may work with various formats such as animation, comic strips and graphics novels (see Figure 2.1). Regardless of the format in which artists choose to work, the overall workflow is by and large the same. In this section, we survey the traditional way of creating cartoons.

Traditional 2D cartooning mainly consists of the following stages [5, 6, 7].
Storyboarding. Traditional cartooning begins with a storyboard which is a form of pre-visualization and looks like a comic strip containing the script for cartoon in the form of crude sketches or images. For cartoon animations, the storyboard maybe complex and include animatics, which are simplified pictures/videos synced with the completed soundtrack. This allows animators to iterate and refine the script before delving into the expensive and time consuming final animation. For comics, storyboards are seldom used. They are usually relatively simple, containing rough sketches of characters, page layout, and speech-balloon placement.

Drawing. Drawings are primarly carried out in papers using black/colored pencils. For cartoon animation, only keyframes which are enough to convey the major actions are drawn by the lead animator. Once approved, these sketchy pencil drawings are cleaned up into clean line drawings, which would be used to generate the *inbetween* frames. For comics, the characters, scene and

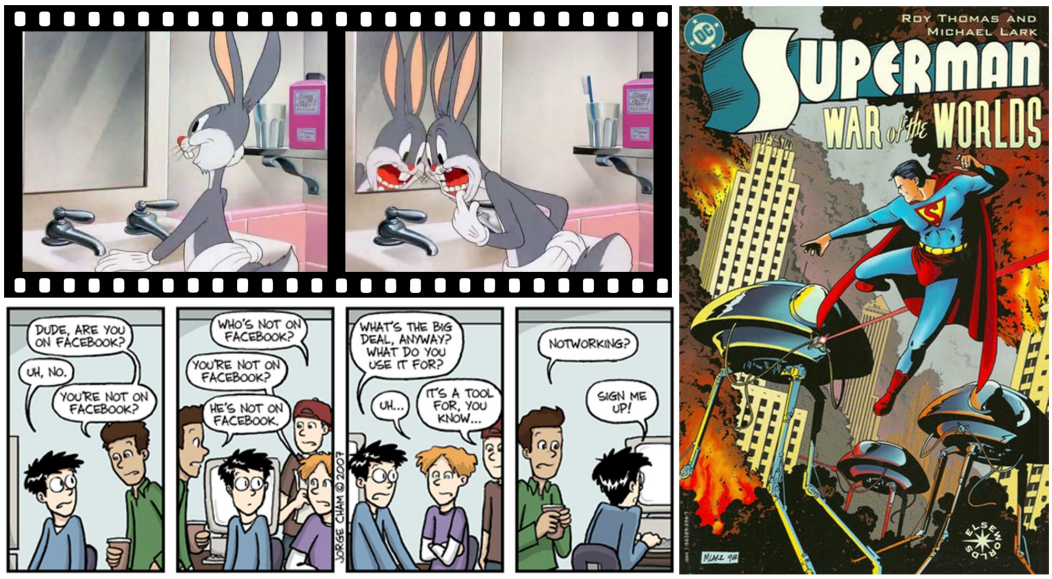


Figure 2.1: *Cartoons in various formats. Top left: "Bugs Bunny : The Wabbit who came to Supper (1942)", an animated cartoon (in public domain); Bottom left: "PhD Comics", a comic strip (© Jorge Cham); Right: "Superman: War of the Worlds", a graphic novel (© DC Comics).*

speech balloons are all sketched on paper, and cleaned up directly during inking.

Inbetweening. Inbetweening is an important step exclusive to cartoon animations. Recall that only keyframes are drawn initially; the goal of inbetweening is to fill in the frames in between the keyframes. This is one of the most tedious stages in traditional cartooning and is performed manually. This step needs careful attention to ensure seamless and smooth motion in the final animation, and is often carried out by less-experienced artists/animators.

Ink-and-Paint. *Inking* is the process of refining the outlines of the pencil drawings with black ink (usually India ink), while *Painting* is the act of adding color to drawings on paper or cels. These two processes are collectively known as ink-and-paint. For cartoon animation, the cleaned up line drawings are first inked and transferred to *cels*, which are transparent sheets of plastic, followed by painting with acrylics on the backside of the cels (so that the ink lines are visible). There are usually different cels for characters and backgrounds which can be composited to create the scene. For comics, inking is often just carried out on top of the pencil drawings which are later erased, or using a light table where a new sheet of paper is placed on top of the pencil drawing and traced over. This is followed by coloring with watercolor, acrylics, or various other

forms of paints.

Shading. To represent various 3D-like characteristics such as depth, shape and lighting of objects in a 2D drawing, artists use shading. For comics, artists use a wide variety of texture patterns such as hatching, cross-hatching and screentones (manga style) to achieve this, while considering the 3D shape of objects and rough location of the light source. Patterns are seldom used in animations, and artists utilize gray/dark shades of colors, to indicate shading regions. This is a difficult step that has to be repeated for each frame while maintaining consistency to prevent visual artifacts across frames.

Lettering. Lettering is exclusive to comics, and is the process of creating speech-balloons and the text within them. Traditionally, lettering is done by hand using two drawing instruments called the T-square and the Ames lettering guide. A T-square is an instrument used primarily as a guide for drawing horizontal lines on a draft table, while an Ames lettering guide is a plastic instrument that allows a letterer to draw rows of guide lines within which letters are written. Using these, letters are physically written on paper using nib pens or brushes with India ink using fonts that best match the mood of the comic. Careful attention is given to the placement of the bubbles to prevent occlusions of important foreground element.

Camera work & Animation. Animations are filmed using a rostrum camera, which consists of a movable lower platform on which the article to be filmed is placed, and an upper platform which contains a camera and can be moved to create effects such as camera pan. Once the entire drawing sequence has been transferred to cels, they are composited on top of each other, with the background cel at the bottom of the stack. The cels are pinned to the camera's lower platform to hold them in place and a piece of glass is lowered on top of them keep them flat. The camera then photographs the frame, and this process is repeated manually for each frame in the sequence.

2.2 Computer-assisted Cartooning

Computers are a versatile tool for creating art, and have revolutionized 2D hand-drawn cartooning. The use of computers as a standard art tool is common these days due to the extraordinary flexibility offered by digital tools,

over traditional methods in various stages of the pipeline such as inking, coloring, image representation, storage, inbetweening and animation.

Computed-assisted cartooning methods first began in the early 1970s when Burtnyk and Wein developed systems for automating time consuming tasks in keyframing and animation [8, 9, 10]. In the late 1980s [11], several new digital features started replacing traditional cel-based methods such as simple region-based filling methods for coloring, use of scanners to transfer pencil drawings and background images into the computer, digital compositing to overlay digital layers instead of cels etc. Since then, research on computer-assisted methods for cartooning has been very popular. Even though most of the underlying mechanisms have benefited from the advent of digital technology, the overall workflow is pretty similar to the traditional pipeline. We now survey each step in the computational pipeline along with the related research works. Here, we would like to particularly give a detailed overview of sketching and digitizing, and shading line drawings along with related subproblems which are the main focus of our applications in Chapters 3 & 4. Additionally, we also discuss the problem of generating 2.5D models from 2D cartoon content which is one of the problems considered in Chapters 4 & 5 with interesting applications.

2.2.1 Sketching & Digitizing Line Drawings

In this section, we focus on the sketching and digitization of line drawings so that they can be processed on a computer. The primary aspect to consider in sketching a drawing is the file format or representation used to describe the drawing. Hence, we first briefly highlight the two most commonly used representations while discussing the advantages of each. Next, we consider ways to digitize a line drawing, particularly as vector graphics that can be easily manipulated.

Line Drawing Representations

There are two major kinds of representations upon which almost all two-dimensional graphics is built—raster and vector graphics, which are illustrated in Figure 2.2.

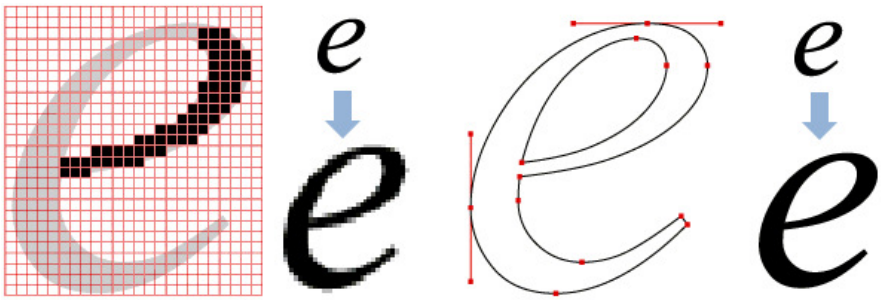


Figure 2.2: *Raster and vector graphics. Left: raster graphics are composed of a grid of pixels with no encoded geometric relationship, hence they are scale dependent and not easily manipulatable; Right: vector graphics are composed of geometric primitives such as Bézier curves (control points shown in red), allowing them to be intuitively manipulated and rendered at any resolution without loss of detail.*

Raster graphics are the most common form of image representation, where an image is approximated using a rectangular grid of pixels, each representing a different color. Raster graphics are capable of representing extremely rich detail, and are the most commonly used format for storing natural digital images, e.g., captured by cameras and scanners. However, while they are convenient for global image manipulations, localized editing of lines or shapes based on geometric and semantic constraints is tedious, since raster images contain no pixel-to-pixel geometric correspondence.

Vector graphics on the other hand, represent content using a combination of mathematical primitives such as points, lines and curves, which can support intuitive editing based on their underlying geometry, but are rather limited in representation detail particularly when it comes to colors and textures. This paves way for interesting properties such as minimal file size, resolution-independence, and easy and intuitive editing. Given two different kinds of representations, there arises a need to convert from one to another. Converting vector graphics into raster graphics is simply a matter of sampling the vector image based on the desired resolution, whereas the inverse problem of converting raster graphics to vector graphics is much more difficult, and is commonly known as *vectorization*. Vectorization is particularly important and relevant to digital cartooning to convert the scanned pencil drawings into an *editable* digital representation. Recent advances in raster-to-vector conversion [12, 13], and richer representation capabilities [14] are slowly making vector graphics as the primary choice for artists. We discuss about line drawing vectorization in detail in the forthcoming paragraphs.

Line Drawing Vectorization

Once the paper based drawings are scanned, they are converted into raster images where there is no geometric information of the lines as discussed above. Hence, they cannot be further edited intuitively, for example, one cannot modify the individual strokes in a raster image and working with pixels is not feasible. Vectorization of line drawings is therefore an important preprocessing step in cartooning. For example, manga (Japanese comics) artists traditionally create their cartoon drawings on paper with pencils, and then scan and vectorize their drawings for further digital processing and editing.

Early methods for line drawing vectorization can only deal with rather simple drawings such as technical layout plans and maps, where they approximate the drawings with straight line segments [15] or employ morphological operations such as thinning [16] and skeletonization [17, 18] to extract the pixel-wide contours. Hilaire and Tombre [19] also attempted to fit line segments and arcs in a least-squares sense to images of technical drawings. As such, these methods are generally incapable of handling freehand drawings, because such drawings are usually composed of large number of strokes with higher-order primitives such as curves.

Several research works have been proposed to handle the vectorization of higher-order primitives. Among them, Chang et al. [20] studied piecewise fitting of Bezier curves for vectorizing cartoon images. Bao et al. [13] proposed to trace near-constant-width lines in line drawings by estimating a dense orientation field that guides the tracing direction. Noris et al. [12] reconstructed robust centerlines in topologically-complex line drawings in the presence of certain junction ambiguities. Very recently, Favreau et al. [21] proposed an algorithm that explicitly optimizes between fidelity and simplicity of the curves used for vectorization, thereby yielding minimal number of curves for easy editability. Some works that estimate temporal information and tracing directions of strokes [22, 23] and extract curves to trace contours with connectivity and orientation [24, 25] may also be used in vectorizing line-drawings.

While recent methods mentioned above and commercial software like Adobe Illustrator [26] and WinTopo [27] could produce reasonable vectorized lines given in line drawing images, they often cause fragmentation and/or stroke distortion problems especially near line junctions due to various stroke recog-

nition issues. Moreover, none of them could faithfully reconstruct strokes with spatio-temporal coherence following how the artist creates a drawing, because they are offline methods and simply do not have access to the stroke-continuity information.

Pen Interface

Another relevant research area for line drawing vectorization is the development of pen interfaces to capture and reconstruct pen strokes in an environment that mimics how people use pens and papers, see [28] for a related survey. Generally, there are two major approaches:

Sensor-based approach develops pen interfaces [29, 30] by using a combination of specially-designed hardware which often involve pens and paper. The most common and largely succesful pen interface is the graphics tablet. The device consists of a flat input surface on which artist can draw using the pen stylus. The drawn results are usually displayed on the computer monitor, but recently some expensive graphics tablets come with a screen on the drawing surface for better usability. Even though there are many other special sensor based solutions; for example, Anoto Digital pens [31] require special paper with invisible non-repeating dotted patterns which can be recognized by the pen's special ball-point tip which is augmented with an infrared camera. The non-repeating patterns recognized by the pen over time allow the device to infer the spatial and temporal information of strokes written with normal ink. Another product, the Wacom Inkling [32] uses sonar-like technology; the pen emits an inaudible pulse of sound, which is picked up by two microphones on the receiver unit which is clipped on top of a sheet of normal paper to be used. The receiver can then use this data to infer the exact location of the pen in the paper-space. In general these methods require hardware support such as special paper with invisible dot pattern, camera-augmented pens, capacitive touch sensor, infrared camera, etc. This approach has not found widespread adoption mainly due to the need for special hardware devices which can be expensive. Graphics tablets are the only widespread sensor based devices, still, many artists prefer drawing on paper since the physical experience of drawing with a graphite pencil on paper is different from using a smooth tipped stylus on a smooth tablet surface. Tablets that come without a screen also have a difficult learning curve since one has to draw while looking at the

monitor rather than the tablet. Hence, artists often draw on paper first, and use tablets to sketch over the scanned paper drawings.

Camera-based approach captures video frames of the pen writing/drawing process through a camera, and attempts to either localize or track the pen tip in each frame to obtain the pen-tip trajectory and stroke. Lin et al. [33] and Tang et al. [34] proposed various video-analysis schemes to recover the spatio-temporal grouping of strokes in Chinese characters. They first extract strokes by thinning/skeletonizing the final video frame of the completed drawing, and then infer the stroke continuity by analyzing the evolution of strokes over time. However, their method does not track the pen tip and attempts to infer stroke connectivity by analyzing the relevant pixels in the stroke skeleton throughout the video. Hence, the method is highly sensitive to shadow and noise, and requires tedious fine-tuning before the operation. Since the occlusion problem (e.g. hand occluding the paper) is not handled, the analysis is performed offline, and there is no interactive feedback. Munich and Perona [35, 36] captured handwriting strokes by analyzing visual input from a conventional camera. The method was later extended to support handwriting recognition by Wienecke et al. [37]. They used a correlation-based template matching method to detect the pen tip. Hence, the method cannot handle planar rotations and perspective distortions. Pen-paper contact is estimated based on a simple local analysis of intensity, which can be easily affected by shadow or neighboring strokes. Since it assumes a small image region that confines the handwriting strokes, occlusions are handled with a fixed-size mask. More recently, Seok et al. [38] assumed a specific conical-shaped colored pen, and tracked the pen tip by simple color detection and geometric cues. While robust to orientation and illumination changes, this method restricts the color and shape of the pen tip. Though the method is online, it cannot handle the occlusion problem because it assumes that the handwriting strokes are unidirectional, and the hand and pen body would move away without hiding the drawn strokes in a fixed time. This assumption does not hold for freehand drawings since the user's hand may move around and occlude previous strokes. Moreover, [38] does not consider temporal information to disambiguate strokes to support meaningful stroke-based editing, and assumes specific color and shape constraints to detect the pen tip.

Some of the problems with previous offline and online approaches that are identified in this section, are handled by our proposed method which is ex-

plained in detail in Chapter 3. Our method experiments with a camera based line drawing digitization method, and is more general and can overcome a number of problems and assumptions taken by existing methods: it considers hand and pen occlusions over the paper; it can retain stroke continuity and drawing order; it is stable for local cluttering caused by neighboring strokes; it is an online method capable of delivering interactive feedback; and it has the potential for enabling meaningful stroke-based editing.

2.2.2 Comic Layout & Lettering

The first step in creating a comic or manga page is the panel layout design followed by speech balloon placement. Artists take various factors into account when designing the layout. First, the size of each panel corresponds to the importance of content. Cao et al. [39] proposed a probabilistic generative model to generate various comic page layouts from a set of images annotated with the desired importance. Second, a simple grid like layout is seldom used since it does not clearly indicate the correct flow among various panels that readers must follow; artists therefore prefer an irregular grid with staggered panels. Again, Cao et al. [40] proposed a probalistic model to generate panel layouts and speech balloon placements that effectively guide the attention of viewers across the page as desired by the artist. Note also that digital fonts have greatly simplified the lettering process by allowing artists to use hundreds of readily available fonts, and obviated the need for physical methods that were used traditionally.

2.2.3 Inbetweening

Inbetweening is a tedious manual process in the traditional cartoon animation pipeline where the character poses in between the keyframes are manually drawn to ensure seamless motions in the animation, and has been an important problem considered for automating with digital methods. The first step in computational inbetweening is digitizing the line drawing, and vectorizing it in case a paper based medium was used in the drawing of the keyframes. This is typically done using any of the procedures described in the previous paragraphs. After that, the main challenge is establishing correspondence among

strokes in the presence of occlusions and topological changes, since 3D information is not present in the drawings [5]. Earliest methods for inbetweening proposed linear interpolation between known correspondence points in each keyframe [8, 10]. Such methods fail in the presence of topological changes in the drawing, and look unnatural due to simple linear interpolation. Moreover, they require tedious manual intervention to specify correspondences since the topological changes are not handled in the formulation. Recent methods such as [41, 12] first analyze the raster image to determine chains of strokes, and the relationships among them. Based on this, they carry out linear or higher-order interpolation among corresponding strokes in each keyframe. Often, logarithmic spiral curves, a visually pleasing curve found in nature, are preferred since they neither cut short smooth rotations like linear interpolation nor contain inflection points like higher order interpolation curves. Such methods have been applied successfully to tight inbetweening where line drawing does not change drastically from one keyframe to another.

Inspite of these advances, many animators prefer to draw each frame, including keyframes and inbetweens manually since it allows complete freedom in expressing the desired motions. This has been a long standing problem since the procedure is laborious even when performed on a computer. Recently, [42] attempted to address this issue. When artists draw a new frame, their method computes the global similarity between past and current drawings to predict what the users might want to draw in the current frame, thereby saving tedious manual effort.

2.2.4 Coloring

After inbetweening, the clean line drawings are digitally colored. Earlier, this was carefully done with simple flood fill based methods [7]. Flood filling based coloring however requires special care especially in freehand drawings. In particular, even small gaps in the contour will cause the colors to leak outside the desired region. Moreover, floodfilling can be tedious in the presence of hatching patterns or sketchy lines in the drawing since this will cause a large number of small closed regions in the drawing. Recent methods are capable of fast and easy coloring using simple user-based scribbles that are propagated across intensity-continuous [43] and texture continuous regions [44] while addressing the above problems. Furthermore, coloring of

multiple frames in one pass is made possible by using techniques such as onion-skinning, or propagating color information by establishing frame-to-frame correspondence [45, 46].

2.2.5 Shading

Shading is usually considered as part of the coloring step, however, we treat it separately since its requirements are quite different from coloring. In detail, while coloring can be completely carried out with the information present in the 2D line drawing, shading requires an understanding of the 3D shape of the object and the rough location of the light source; such semantic information is not available readily in 2D drawings. This makes computational shading a very challenging problem, and very few attempts have been made at directly solving this issue.

Currently, artists simply use digital brushes to perform shading by imagining the position of the light source and the 3D shape of the object of interest. Different styles of shading such as soft shading where the shading is represented with gradients to provide a realistic 3D-like shading, or 2D cartoon shading discrete gray colors or manga screentone patterns are used are very common. Artists have to carefully consider the requirements of each shading style and manually repeat them for each drawing while maintaining a consistent look and feel. Also, there is no easy way to experiment with different shading styles.

While we can automate shading with the presence of geometry, artists are very unlikely to model the geometry for every single drawing. Hence, the key to automatic shading is to somehow obtain a depth value along each line as well as the interior of the objects bounded by these lines. In general, there are two approaches to achieving this: *depth assignment* where users can markup some cues to assign the depth to the 2D drawing, or *depth inference* where the computer can automatically infer the depth among various regions in the drawing by extracting perceptual information from the drawing and interpret this to estimate the depth. The problem of making use of this depth to reconstruct partial 3D geometry is also a related problem.

Depth Assignment

Depth assignment refers to manually assigning depth to 2D content. The direct approach asks users to manually assign depth values to segmented groups of pixels [47, 48]. This however requires tedious, trial based input especially for complex input; humans are better at estimating the relative depth orderings among neighboring regions, and this can be modeled by considering depth equalities and inequalities [49, 50, 51]. Such methods often provide an interface for users to markup depth inequalities for neighboring regions. These previous methods however do not consider the depth within each region, and can only be applied on simple cartoons without complex wrinkles, or detail strokes. Due to the amount of manual input, it may be impractical to apply them to cartoons with complex interior strokes, or a large number of regions. Hence, depth inference methods that extract and use perceptual information from drawings with minimal to no effort from the users are usually preferred.

Line Drawing Interpretation

Line drawing interpretation has gained much interest from both physiological perceptual studies and computer vision researches for decades. To interpret lines, both local features (properties of lines) and global features (interactions among lines) have been explored. In particular, 3D surface properties such as principal curvatures [52, 53, 54] and local depth/orientation discontinuities induced by lines [55, 56] have been extensively studied from a geometric perspective. By exploring the Gestalt psychology [57, 58], researches have also been conducted in interpreting higher-level semantics of lines by considering human perception. Such a higher-level understanding of lines has brought in a wide range of applications, e.g., curve completion [59], local layering [60, 61, 62], and line drawing simplification [63, 64].

However, applications of such perceptual understanding to problems in cartoon production is generally lacking. In our work, we also aim at interpreting lines guided by Gestalt psychology. But different from existing applications, we estimate the geometry of surfaces illustrated in 2D line drawings especially considering the wrinkle strokes, by analyzing and combining various perceptual cues so that the tedious shading process can be automated.

Depth Inference

Automatically inferring depth from natural images is a classical problem in computer vision. Several approaches that exploit photographic cues in natural images for estimating depth such as texture [65], shading [66, 67], and lens focus [68, 49], have been developed. As such, these methods are not applicable to cartoons since these assumptions about natural image properties do not hold for hand-drawn cartoons reliably. For example, shading in cartoons can be crude and physically incorrect, there may be hatching patterns instead of shadows to convey lighting, and so on. Textures are also largely absent in most cartoons, hence, feature-based approaches are hard to apply.

However, cartoons generally have well defined contours, and hence well-defined *T-junctions*, which are points where three curves from the boundaries of neighboring regions appear to intersect due to occlusion. From the *amodal completion law* in Gestalt psychology [58], it is known that when a curve stops on another curve, thus creating a T-junction, our perception tends to interpret the interrupted curve as the boundary of some object undergoing occlusion. There are some previous works which exploits T-junctions, but they mainly focus on natural images [69, 70]. Other approaches attempt to learn depth cues from images using a supervised approach and require extensive datasets for machine learning [71, 72], and cannot be directly applied to cartoon images. Recent work by Palou and Salembier [60] employed T-junctions along with convexity cues to resolve layering in natural images, but their results show only a few number of segmented regions, and the method deals with basic T-junction cases only (please refer to Chapter 5 for details on more complex junctions). Liu et al. [61] attempted to estimate depth ordering among regions of a cartoon video sequence by exploiting T-junctions and temporal information from nearby frames. They first identify 3-valence corner points as T-junctions, and aggregate them from multiple frames to remove noisy or incorrect identifications. To ensure temporally consistent depth labeling for image regions, they assign similar depths to regions based on the depth assigned in previous frames using a graph-cut formulation. Their approach however is not sufficient for static 2D input where there is no temporal information to guide the depth estimation. This is particularly common when T-junctions are ill-defined or occur on the interior of regions where the extent of lower and higher regions is not clear. Also, there are often interior strokes

in line drawing which do not contain any T-junctions, hence there is a need for formulating methods that go beyond T-junctions.

We have identified several perceptual cues in Gestalt psychology that have not been sufficiently explored. In Chapter 4, we explore and model many of these gestalts including convexity, proximity, continuity, and regularity along with T-junctions. We analyze cartoon line drawings using these cues to reliably estimate depths depicted by the 2D lines. We perform a global optimization to ensure consistent interpretation of all these gestalts. Once this depth information is available for each line in the drawing, it can be used to guide a 3D reconstruction method that can estimate some partial 3D information of objects in the drawing to support shading.

Estimating 3D Geometry from 2D Lines

Extensive researches have been done in reconstructing 3D or pseudo-3D geometry from an image. Rather than attempting to be exhaustive, we discuss those tailored for line drawings. To construct a 3D geometry from a line drawing, various works have been developed to construct objects or architectural structures from industrial or architectural line drawings by making strong assumptions on the input drawings and the shapes being constructed [73, 74]. Due to the strong assumptions, these methods generally are inapplicable to arbitrary drawings as seen in cartoons. To handle arbitrary line drawings as inputs, existing geometry reconstruction methods can be roughly classified into inflation methods and normal field estimation methods.

Inflation methods aim at reconstructing 3D geometry in an input drawing by lifting up each object region from 2D to 3D. To reconstruct detailed structures on an object region, user-specified constraints within each region the drawing may be adopted [75, 51, 76, 77, 78]. While these interactive methods can produce good results given sufficient user-specified constraints, they require tedious manual inputs, especially to handle crowded wrinkle strokes that are commonly seen in cartoons. Besides user-specified constraints, attempts have been made to automatically reconstruct surface geometry from line drawings by analyzing T-junctions among the boundary lines [79]. However, the analysis of interior strokes is generally lacking. Although some researches tried to handle interior strokes by region subdivision, the recovered geometry is still

unsatisfactory due to the lack of semantic understanding of interior strokes. In contrast, our method presented in Chapter 4 estimates partial geometry information by analyzing the semantics of all the strokes based on their shapes, spatial layout and interaction in the line drawing. Our inflation method can recover a delicate geometry even if the line drawing contains complex wrinkle strokes, and support the generation of shading to significantly reduce the manual effort.

Normal field estimation methods aim at reconstructing an image-space normal field over the input drawing using various image-space features, e.g. dominant curvatures of strokes [80], user-marked normals [81], 3D normals along isophotes [82], hatching strokes [83], and cross-sectional curvature lines [84, 85]. While most of these methods can well reconstruct the overall shape from the drawing, they are generally unable to handle interior strokes that indicate wrinkles or folds. To handle interior strokes, Johnston [80] proposed to incorporate manual specification to indicate the sign of normals along each stroke whenever needed. Bui et al. [83] proposed to estimate local geometry along each stroke based on nearby hatching strokes. In short summary, all existing methods adopt only simple local features (e.g., curvature) to estimate local geometry, and need additional manual specifications (e.g., label and hatching strokes) to achieve good results. In comparison, our method analyzes not only local stroke properties, but also global interactions among the strokes (folding and squeezing). Therefore, we are able to achieve a better detailed geometry that can support shading results even for drawings with complex wrinkles.

2.2.6 Animation

Digital software has completely replaced use of physical medium such as cels in compositing the final animation. Each foreground object can be stored in separate *layers*, and colored and composited over their respective backgrounds with simple alpha blending techniques. Moreover, pan and zoom operations which required physical movement of the rostrum camera become trivially possible with equivalent software-based operations. The final output can be a digital video file in standard formats (MP4, WMV, QuickTime, Flash etc.), or rendered onto cinematic films. This is done for each scene until the entire film is complete. Note also that there are several other related research areas such as deformation, physically-based animation, etc. We do not go into the details

of these areas as they are not related to the works presented in this thesis.

2.2.7 2.5D Modeling

So far, we have surveyed several key areas in the computational cartooning pipeline, while particularly detailing the key focus of the projects presented in this thesis. As we stated in the introduction, most cartoon animations are typically created from scratch. On the other hand, there is a huge amount of digital cartoon content readily and easily (e.g., cartoon images) available today. Hence, there is artistic potential in enabling users to reuse this content to achieve visually pleasing still-image animation or manipulation with minimum effort. The key to reusing such 2D cartoon content is in identifying various parts or regions in the image and estimating the relative depths so that we may have a model to support semantic editing. It is also necessary to retain the artistic style depicted in the original 2D content.

2.5D graphics create complex scenes by layering 2D graphics at different depths (see Figure 2.3). They have attracted great interests in a wide range of areas due to their simplicity and elegance for delivering 2D-like art forms for applications in cartoon design and manipulation, generating parallax effects in still images, desktop publishing design etc. Usually 2D meshes (planar) are used to represent different layers, where each layer is arranged in a scene in 2.5D i.e. stacked at different depth levels to produce visual perceptions such as occlusion, proximity, and depth in general.

Recent research has focused on the problem of creating 2.5D models to support various visual effects, as well as enrich the modeling techniques to allow for richer representations. Rivers et al. [86] combined 2D vector drawings in different views to create a 2.5D model that can be rotated arbitrarily. To handle these problems, the authors propose using linear interpolation in 2D for the strokes' shape, and 3D coordinates for their position and depth ordering. McCann and Pollard [87] generalized the layering concept by enabling 2D graphics to partially occlude one another, or even with themselves by local layering. While global layering just requires a list to store the ordering, they propose a list-graph data structure to handle local layering, where a graph containing a list at each node is used to store the layer ordering at that local region among the overlapping layers. Yeh et al. [88] enriched 2.5D modeling with assorted

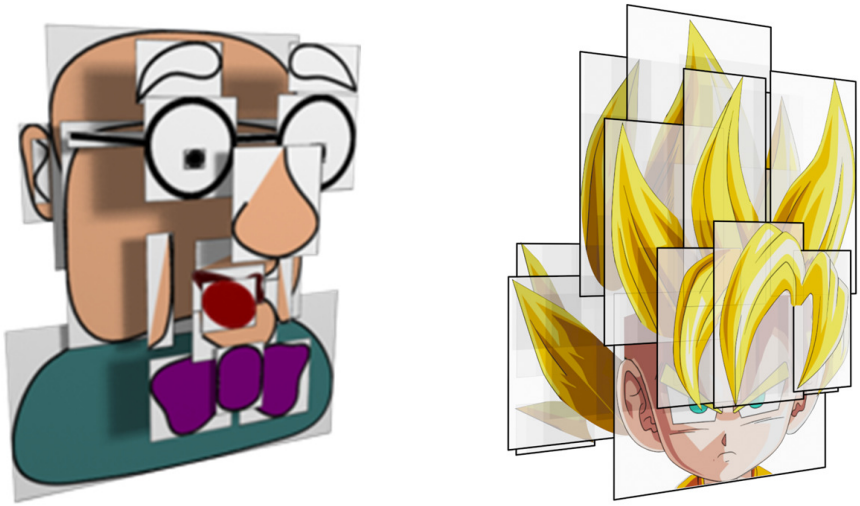


Figure 2.3: *Examples of 2.5D models. Planar layers are stacked on top of each other at different depths (figure on the left taken from [86]).*

effects by considering 2D graphics with both front and back sides. By this, they supported several novel operations such as rolling, twisting and folding that exposed the back side of the 2.5D model, and greatly improved the ability illustrate rich graphics.

While the above works generally focus on creating 2.5D models from scratch, the key to reusing existing 2D contents is in extracting the 2.5D information from images and videos to reconstruct models, which is the focus on our work in Chapter 5. The primary subproblems involved in recovering the 2.5D information from an image involves decomposing the foreground into regions i.e. segmentation, inferring depth ordering among them i.e. layering, and completing the occluded layers so that holes do not show up when manipulated. The upcoming sections discuss previous works for each of these subproblems.

Segmentation

Segmentation is a classic research area in computer vision with huge volumes of research works. Precise segmentation of cartoon images to decompose them into meaningful regions is a difficult problem. Common unsupervised approaches include watersheds [89], superpixels [90] and graph-based methods [91]. These can result in over-segmentation and often require region merging techniques. While unsupervised methods are attractive due to lack of need for user input, in most cases the segmentation quality is not acceptable for car-

toon images since important structural details especially near boundaries are often omitted due to various reasons ranging from image noise, non-uniform shading of colors, gaps in contours, decorative lines which do not represent region boundaries etc. Moreover, even though edge information provides strong hints for decomposition, extracted edges often contain gaps and are difficult to join reliably. Hence, there is a need for application-specific design of segmentation methods that can exploit domain knowledge and assumptions about the input cartoon images.

Unsupervised segmentation methods for cartoons include [92] that decomposes cartoons with homogeneous colors and well-defined contours; they extracted contours using the Laplacian of Gaussian filter, and performed connected-component analysis to decompose the image. However, the above assumptions of color homogeneity and strong contours do not always hold for cartoons of other styles, often leading to problems. To alleviate this, Zhang et al. [93] proposed an approach combining floodfilling, morphological operations and region merging. This however still needs tuning of user defined parameters to achieve optimal results.

On the other hand, supervised segmentation techniques that accept scribbles as user input to mark foreground and background regions have been designed to segment cartoons, and offer far more flexibility and control. Sýkora et al. [43] designed a scribble based coloring framework that uses graph cut to segment and produce region masks based on user defined scribbles. They first preprocess the image using a Laplacian of Gaussian filter to enhance the contours, and formulate the segmentation as a multi-way cut on a graph whose nodes and edges represent image pixels and their 4-connected neighbors respectively. The primary advantage of this method is the ease of segmentation even in the presence of strong shading or textures. Our segmentation method in the 2.5D modeling framework is supervised with minimum user input, since our animation and manipulation applications demand a high quality segmentation.

Depth Ordering

After segmentation, the next step in 2.5D modeling is to recover the depth ordering among the various regions. These methods can be either manual depth

assignment as discussed in Section 2.2.5, or automatic depth inference as in Section 2.2.5. We propose a novel layering metric based on Gestalt psychology to automatically infer the relative depth orderings among adjacent regions.

Layer Completion

The next step in 2.5D modeling requires the completion of each region in the model since holes show up otherwise when animated. Completing missing regions in images and videos, also known as *inpainting*, is a long-standing problem in computer vision with applications in object removal, restoration of old films/photographs, etc. Several works have been proposed to deal with this problem for natural images based on structural cues [94], textural cues [95, 96] and a combination of both [97, 98]. The main challenge for cartoon layer completion lies with the extreme differences in appearance between cartoon images and natural images. For example, cartoon images have minimal texture, strong contours, physically-incorrect geometry etc., whereas natural images contain strong textures, no contours and physically correct geometrical structures. This prohibits the direct use of the above techniques that are designed exclusively for natural images.

There is limited previous work that deals with layer completion for cartoons; one prominent work that explicitly addresses this problem for cartoon videos is by Zhang et al. [99], where the region in each frame is warped and aligned to consecutive frames where the occlusion is assumed to have reduced. More recently, Sýkora et al. [78] completed occluded layers in single images by propagating the manually assigned depth ordering from the boundary using a diffusion process. This is achieved by initializing a value of 1 in the pixels of the occluded layer boundary, and 0 in the occluding layer boundary. These values are then diffused to the neighboring pixels as in [100], and pixels with a value less than 0.5 are thresholded to obtain the completed region. While this works well for their application where the 2.5D model remains static, this method does not consider other kinds of occlusions, for e.g., a single object broken into two by a common occluder. Layer completion for cartoon images remains an application-specific and open research problem. In our work, we propose a completion method that can handle three common cases of occlusions seen in cartoon hair which is the focus of our application.

CHAPTER 3

Line Drawing Vectorization with Commodity Camera*

Vectorization of line drawings is an important pre-processing procedure in cartooning. Most of the existing methods are *offline*, and accept raster images of completed line drawings as input to generate a vectorized representation. Due to this, they do not have access to stroke-continuity information and hence result in broken or incorrectly merged strokes. In this chapter, we present a novel approach to the problem of clean line drawing vectorization. Rather than designing complex offline methods that work on completed drawings, we present a simple and efficient *online* approach, to recognize and vectorize strokes drawn on paper in front of a commodity webcam.

3.1 Introduction

The acquisition and vectorization of hand-drawn strokes on paper are beneficial to a wide range of multimedia applications: from handwriting recognition to cartoon production, video scribing and sketch-based modeling. This bridges the gap between paper and digital representations, and facilitates the development of computational methods for editing and interaction.

In cartooning, vectorization is an important preprocessing procedure, where hand-drawn paper drawings are converted to vector graphics based digital representations for further processing with computer software. In general, there are four major approaches for recognizing and vectorizing paper-based line drawings according to the kind of inputs being processed:

- *Manual approach* involves manual effort of tracing individual strokes in a scanned input image. While being accurate and flexible, this approach can be extremely tedious subject to the complexity of the drawing.

*This author was the primary contributor of this work [101], who conceived the research idea, performed literature survey, carried out the implementation, and conducted the experiments.

- *Image-based approach* [13, 12] takes a scanned image of a completed drawing as an input, and attempts to automatically analyze the drawing contours to infer the stroke geometry and produce the vector representation. Major challenges of this approach are the image noise and the ambiguities in reconstructing the strokes due to image resolution and line junctions.
- *Sensor-based approach* [29, 32] employs hardware devices to track the pen trajectory on paper to detect the pen strokes. This approach, however, requires specially-designed devices that are not common, e.g., sensors in the ballpoint tip of a pen, special patterns printed on paper, etc.
- *Camera-based approach* [36, 38] captures the video of the drawing process, and recognizes strokes on paper by tracking and analyzing the pen-tip trajectory by computer-vision methods. The two main advantages of this approach are 1) the use of original writing instruments, i.e., pens/pencils, so involving no learning overhead in practice, and 2) the availability of temporal information from the captured video.

Our goal in this work is to develop an interactive camera-based method to capture and vectorize hand-drawn strokes on paper through a commodity low-cost webcam that captures at 480p/720p resolution and 30 frames/second. By this, we can faithfully vectorize drawn strokes with improved stroke continuity, and also recover the temporal information and drawing order of the strokes. With the emergence of recent ubiquitous devices like the Google Glass [102], camera-based approach could be particularly useful to help recognize hand-drawn inputs in front of the embedded camera, and to facilitate the development of pen input interface.

Previous methods for vectorizing hand-drawn strokes either exist as online/offline approaches, which are constrained by the stroke length, size, and direction, or strictly offline methods, which process the scanned images of the final drawings. To the best of our knowledge, this is the first attempt of developing an online camera-based method to acquire and vectorize *freehand* line drawings.

However, such a problem is very challenging since we need to robustly track the tip of the pen in the video, accurately model the pen-paper contacts, handle the pen-paper as well as hand-paper occlusion, and complete the entire computation with interactive performance. To address these issues, we pro-

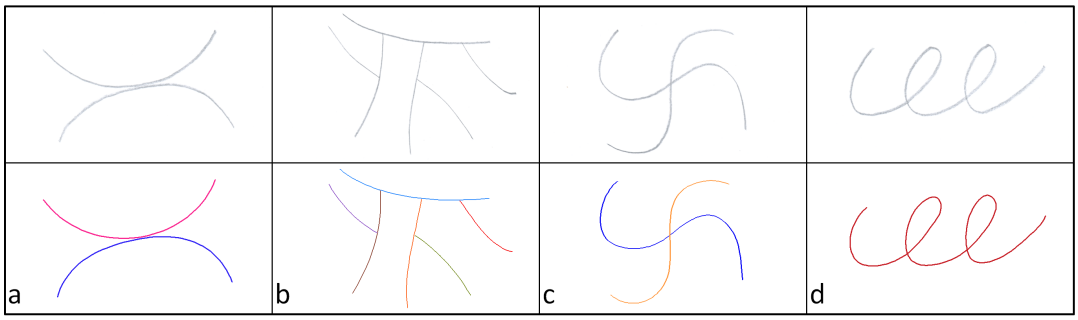


Figure 3.1: *Top row: four line drawings on paper, revealing four common stroke recognition ambiguities: (a) spatial adjacency, (b) junction ambiguities, (c) multiple-stroke intersection, and (d) self-intersection. Existing vectorization methods may misrecognize the original strokes as strokes of some other drawing continuity or break the original strokes into several shorter strokes at the junctions and intersections. Bottom row: our method can faithfully recognize and vectorize these strokes even in the presence of the ambiguities; color-coding is used to indicate individual recognized strokes.*

pose the following novel techniques, which are the major technical contributions of this work:

- First, we identify three distinctive features, i.e., variance, color, and shape, and integrate them to robustly locate the tip of pen/pencil while quickly rejecting negative samples. By this, we can obtain the trajectory of the pen-tip across the video frames as a function of time.
- Second, we model the pen-paper contact implicitly by analyzing the zero-crossings of the Laplacian of Gaussian filter response along the pen-tip trajectory, and extracting an appropriate subset of it to estimate the potential region of the drawn strokes.
- Lastly, we combine the spatio-temporal information to reconstruct strokes while preserving the stroke continuity and drawing order even in the presence of recognition ambiguities.

Our method has several advantages over existing offline vectorization methods, which work only with completed drawings. First, it is *online*, so it can capture and produce vectorized strokes with real-time feedback. Second, it involves only everyday pens/pencils and a simple camera rather than specialized pen/paper hardware. Third, our online method enables the capture of strokes with temporal information and drawing order, thus facilitating the development of various multimedia applications, and could potentially be

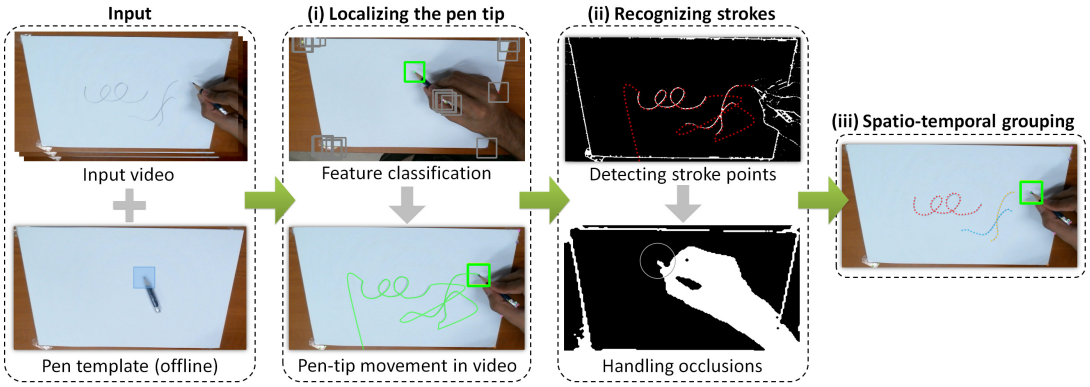


Figure 3.2: *Overview of our stroke recognition and vectorization method. From left to right: our input is a video stream and a simple pen-tip template; (i) potential patches are searched in video frames to localize the pen tip and track its trajectory; (ii) Laplacian of Gaussian filter response helps extract stroke points to estimate the actual drawn stroke; hand and pen occlusions are also tracked; and (iii) lastly, spatio-temporal grouping is performed to identify individual strokes (color coded).*

integrated with camera-based interaction device such as the Google Glass. Lastly, by using temporal information, we can effectively avoid various stroke recognition ambiguities that are common in existing vectorization methods, see Figure 3.1: spatial adjacency, junction ambiguities, multiple-stroke intersection, and self-intersection, see also [12]. From the results shown in Figure 3.1, we can see our method can avoid these recognition ambiguities and faithfully recover each stroke natural to the actual stroke drawn by the user while preserving the stroke continuity in the vectorization.

3.2 Overview

Figure 3.2 overviews our method with a running example. Our system setup includes a drawing space with a sheet of paper fixed on a desk, an everyday pen/pencil for drawing, a commodity webcam that oversees the drawing space from the top, and a desktop computer that processes the webcam's video stream and performs our method to recognize and vectorize the drawing strokes. In summary, our method consists of the following three major steps:

i) **Localizing the pen tip.**
Extracting discriminable features around the pen tip for efficient, rotation-invariant, and illumination-invariant per-frame detection of pen tip (Sec-

tion 3.3.2); *Pruning* the search space by corner detection to accelerate the performance (Section 3.3.3); and *Classifying* the extracted feature vectors using a cascaded approach to quickly identify and track the pen tip over time to obtain a meaningful pen-tip trajectory (Section 3.3.3).

ii) **Reconstructing Strokes.**

Extracting a subset of points along the pen-tip trajectory to estimate the footprint of the drawn strokes by analyzing the edge profile along the trajectory (Section 3.4.1); and *Resolving* occlusion problem caused by user's hand and pen over the paper (Section 3.4.2).

iii) **Spatio-temporal Grouping.**

Grouping the extracted points into a stroke by considering stroke connectivity, spatial proximity and temporal proximity information (Section 3.5); and *Vectorizing* the clustered stroke by fitting a cubic spline over the grouped points, and outputting the final result in SVG vector format.

Note that lighting is important in our method since it is primarily camera based. If the environment is too dark, then our technique may fail to accurately track the pen tip, and analyze the paper in the video frames. Therefore, we assume that the drawing space is well lit, and the lighting conditions do not vary significantly throughout the drawing process. In our implementation, the above computation takes only $\sim$0.072 sec. per video frame, showing that our method can support online stroke recognition and vectorization at interactive speed. However, since some drawn strokes may be occluded fully or partially by user's hand or pen, we can only reconstruct their unoccluded parts during the online computation. Our method keeps tracking the hand and pen locations (see Section 3.4.2) to determine if the occlusion has been removed, and then reconstructs and renders the previously-occluded strokes once they become visible and can be detected by our method. Hence, in a single video frame, our method may recover more than one stroke at a time.

3.3 Localizing the pen tip

3.3.1 Initialization

Our input consists of a video stream of the pen drawing process and a pen-tip template. To prepare a new pen-tip template, which is just an offline one-time step, it only takes a few seconds to mark a 60×60 image region on a frame of the video stream to enclose the pen tip. Here we only assume the pen is placed over the paper, on which the drawing would be done, and place no restrictions on the pen orientation as our method can later automatically normalize the template input to a canonical form that is rotation-invariant.

In detail, when extracting the pen-tip template, we also need a foreground-background segmentation [103] on the video frame to obtain a background-subtracted binary image, assuming that the pen is not part of the initial scene. Here we set a high adaptation time period in the background model to avoid too-early merging of foreground into the background.

3.3.2 Feature Extraction

Given a patch, which could be a pen-tip template or a 60×60 region in a video frame, we next need to extract distinctive features in the patch for supporting the pen-tip localization. The set of features we carefully selected in this work are *background confidence*, *intensity variance*, *color*, and *shape*. These features are simple-to-compute and yet discriminative to help identify the pen tip under rotation, illumination variation, and mild perspective distortions.

i) *Background Confidence ($\mathcal{B}_i$).* By applying foreground-background segmentation [103] on every video frame (including the one for the pen-tip template and also the video frames of the drawing process), we can obtain a background-subtracted binary image M whose background pixels take zero values. Hence, given a patch in M, we can count the number of background pixels in it, and compute the patch's background confidence ($\mathcal{B}_i$), i.e., the ratio of the number of background pixels to the total number of pixels in the patch.

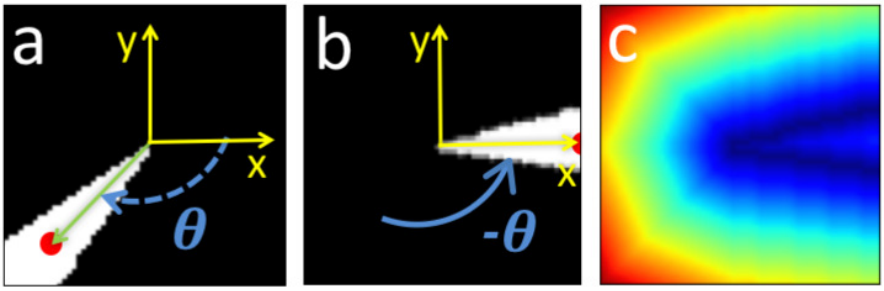


Figure 3.3: *Rotation-invariant shape feature. Given a binary patch extracted around a detected corner point in a video frame, we compute (a) the center of mass (red) of the largest foreground blob, rotate and normalize the patch into (b) a canonical shape representation, and then use (c) a Euclidean distance transform of the contour to compute the Chamfer distance.*

ii) *Intensity Variance (σ^2).* The second feature we employed is the statistical variance of pixel grayscale values. This feature helps to measure the uniformity of pixel intensities in a patch. Moreover, after we obtain the pen-tip template, we pre-compute its intensity variance, say σ^2, and define low and high intensity variance thresholds $0.5\sigma^2$ and $1.5\sigma^2$, respectively, to support the fast cascaded classifier later on.

iii) *Color ($\mathcal{C}_i$).* Third, we use the CIELab color-space to compute the color feature of a patch. We quantize the color by computing a normalized 2D histogram with 8×8 bins for each of the 3 channels, and flatten the histogram as a 192-dimensional color feature vector. By this, we can compute the color feature vector of the pen tip template, say $v_t^{(c)}$, as well as the color feature vector of any patch in a video frame, say $v_i^{(c)}$. Hence, we can further compute the color dissimilarity measure between them by Euclidean distance: $\mathcal{C}_i = \|v_t^{(c)} - v_i^{(c)}\|$.

iv) *Shape ($\mathcal{S}_i$).* Lastly, we compute the shape feature of a patch by devising an efficient rotation-invariant method. Note that from the video frames, since we create candidate patches to be centered at detected corner points (Section 3.3.3), we can be certain that the pen tip (if any) would always appear at the center of a patch. Hence, for each patch, we compute the shape dissimilarity as follows:

   (a) Find the largest connected foreground blob in the given patch, and compute its center of mass, see the red dot in Figure 3.3(a);

   (b) Compute its offset angle, say θ, from the positive x-axis in the image

space in clockwise direction, see again Figure 3.3(a);

(c) Rotate the patch image about its center (pen tip, if any) by $-\theta$ in order to normalize the patch to a canonical representation, see Figure 3.3(b);

(d) Lastly, compute shape dissimilarity measure $\mathcal{S}_i$ for a given patch against the shape of the pen-tip template using Chamfer distance: $\mathcal{S}_i = \sum_{x \in T} DT_i(x)/|T|$, where DT_i is the Euclidean distance transform of the i^{th} patch, T is the set of contour points in the pen-tip template.

3.3.3 Feature Classification

Pruning the search space

In typical object localization, a sliding window is often used to go over a given video frame and extract all possible patches (potentially hundreds of thousands) for feature matching. Clearly, this approach is infeasible to deal with a large search space for interactive applications, so we prune the search space by two key observations in our problem:

- First, we assume a fixed camera and paper. This is a reasonable assumption in most camera-based methods. Given that, we assume that the pen-tip relatively retains its scale while drawing, so we only consider patches of the same size as the pen-tip template.
- Second, since the pen tip is always a *sharp corner*, we perform corner detection [104], which is a very fast process, to extract fixed-size patches around detected corners to accelerate the search for the pen tip.

By these, we can reduce the cardinality of the search space to the number of corners detected in each video frame and greatly accelerate the pen-tip localization.

Cascaded Classifier

Given the set of patches $\{s_i\}$ extracted from the pruning method above, we exploit the four features described in the previous subsection, and devise a

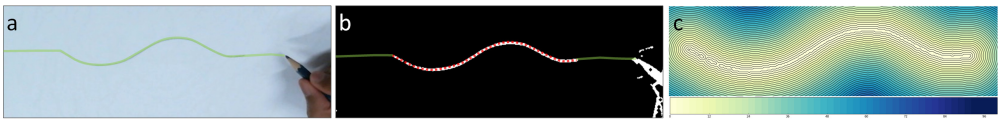


Figure 3.4: *Detecting drawn strokes: (a) captured pen-tip trajectory in green; (b) thresholded $\boldsymbol{L} \circ \boldsymbol{G}$ filter response reveals the drawn stroke; stroke points (red) are further extracted by using (c) level-sets defined around the detected stroke.*

fast cascaded approach to further classify patches in $\{s_i\}$. Note that this is a lazy approach in applying the features, so that we do not need to compute all the four features for every patch in $\{s_i\}$.

- In the first step, we compute the background confidence score of each candidate patch in $\{s_i\}$, and discard those outside the threshold $[0.5\mathcal{B}_t,\ 1.5\mathcal{B}_t]$, where $\mathcal{B}_t$ is the background confidence of the template.
- In the second step, we compute the intensity variance of the remaining patches, and filter out those with intensity variance outside the prescribed threshold range $[0.5\sigma^2,\ 1.5\sigma^2]$. The above features enable us to quickly reject almost all irrelevant patches that are not similar to the pen-tip template.
- In the third step, we compute the color dissimilarity measure $\mathcal{C}_i$ of each remaining patch using the color feature vectors of the patch and the pen-tip template. By this, we can filter and retain only the patches whose $\mathcal{C}_i$ is less than a threshold T_c, which is set to be 0.3 in all our experiments.
- The last step is similar to the third step, but now we compute the shape dissimilarity of the remaining patches, to retain only the patches with $\mathcal{S}_i$ less than a threshold T_s, which is again set to be 0.3 in all our experiments.

Since the fast cascaded filtering process above may still retain more than one patch in the end, we may need to further compute the average value of color and shape dissimilarity measures for each remaining patch, and select the one with the smallest average to locate the pen-tip patch.

3.3.4 Tracking

There may also be scenarios where the pen tip is not detected by the cascaded classifier above, e.g., missing the pen tip in the corner detection, and severe

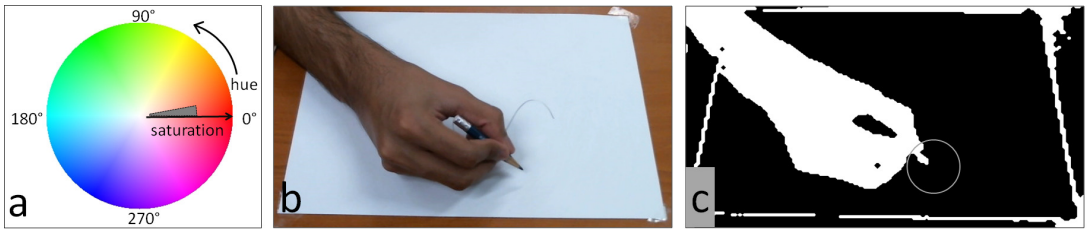


Figure 3.5: *Occlusion detection: (a) thresholding skin color in the hue-saturation space; (b) a sample frame from the video; and (c) masks obtained for the pen (gray circle) and for the hand (white).*

visual interference such as noise, shadow, and clutter. Hence, in addition to per-frame detection, we also employ the detected pen-tip location in the previous video frame to perform short-term tracking of the pen tip using the Lucas-Kanade optical flow method [105]. By this, we can improve the pen-tip localization, and avoid breaking the pen-tip trajectory due to the above mentioned reasons.

In detail, we initialize the tracker by each detected pen-tip location from the cascade classifier, and continue the short-term tracking until the cascade classifier finds a subsequent detection, at which time the tracker is reinitialized.

By combining feature-based detection and optical-flow based tracking, we can obtain a smooth trajectory of the pen-tip, say $\mathbf{T}$, over the drawing space as a function of time (e.g., see Figure 3.4(a)):

$$\mathbf{T} = \Big\{ (p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n) \mid p_i \in \mathbb{R}^2, t \in \mathbb{R} \Big\},$$

where $p_i = (x_i, y_i)$ denotes a spatial coordinate, and t_i a monotonically-increasing timestamp (frame index).

3.4 Reconstructing Strokes

3.4.1 Detecting Stroke Points

Once we capture the pen-tip trajectory across the video frames, our next step is to determine the set of stroke points on the paper, say $\mathbf{S}$ (see Figure 3.4(a)). In detail, we determine $\mathbf{S}$ by analyzing the response of the Laplacian of Gaussian ($\mathbf{L} \circ \mathbf{G}$) filter over the video frames. The $\mathbf{L} \circ \mathbf{G}$ filter corresponds to how the

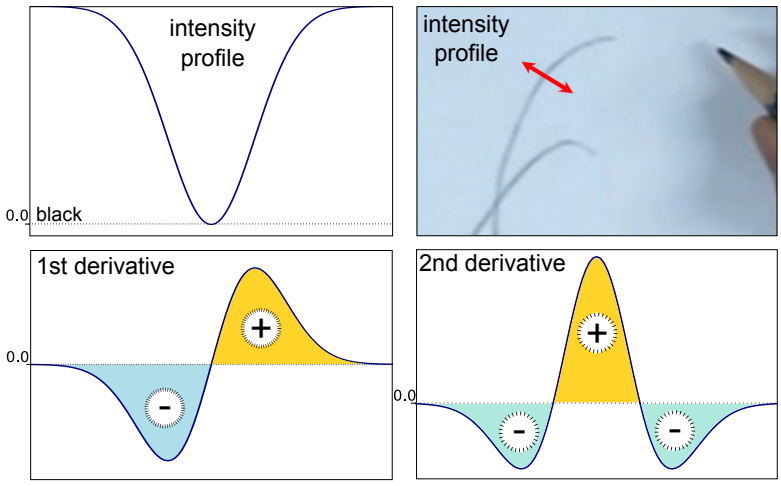


Figure 3.6: *Stroke intensity profile. Top: intensity profile across a stroke (perpendicular to stroke direction). Bottom: first and second derivatives of the intensity profile, respectively. The maxima of the second derivative corresponds to the centerline of the stroke.*

human visual system observes sudden changes in contrast and downplays subtle differences [106]. Mathematically, it is represented by the curvature, or the second derivative of the image intensity, see Figure 3.4(c).

The intensity profile of strokes in clean line drawings is generally in the form of a parabolic profile (see Figure 3.6(top)), and the maxima of the filter response corresponds to the centerline of edges, or in our case, the centerline of the drawn strokes (see Figure 3.6(bottom)). From each frame, say F, we derive a new image $\mathcal{L} = \max(0, \mathbf{L} \circ \mathbf{G}(F))$ by applying the $\mathbf{L} \circ \mathbf{G}$ filter on F, where we have clamped the negative filter response to zero. Next, we scale the image to the range $[0, 255]$. After that, since there are often some unwanted responses caused by noise, we further threshold $\mathcal{L}$ by setting pixels ≤ 15 (chosen empirically for our setup) to 0 to remove noise and weak responses, and obtain a set of edge pixels over the video frame, say $\{\mathbf{q}\}$. Next, we can define a level-sets around the edge pixels in $\mathcal{L}$ using a distance transform with the Euclidean distance metric: $D(\mathbf{x}) = \min_q \|\mathbf{x} - \mathbf{q}\|$, where $\mathbf{x}$ is each pixel in the video frame. The stroke points can be arbitrarily spaced based on the speed of the drawn stroke; hence we first resample them to ensure that the stroke points in the entire pen-tip trajectory are equally spaced; in practice the spacing is set to be unit distance. By this, we can extract a set of stroke points $\mathbf{S}$ as a subset of the

trajectory $\mathbf{T}$ lying close to the drawn strokes within a threshold:

$$\mathbf{S} = \{(p_i, t_i) \mid D(p_i) \leq \tau; \forall (p_i, t_i) \in \mathbf{T}\} \ ,$$

where τ defines the maximum level set below which points are on the drawn strokes; in practice, it is set to be 2.0.

3.4.2 Handling Occlusions

In the previous subsection, we presented an analysis method to obtain stroke points on the paper based on the $\mathbf{L} \circ \mathbf{G}$ filter response. However, it is important to note that not all edges in the filtered image $\mathcal{L}$ actually correspond to the drawn strokes. Typically, the user's hand and pen inside the video frames may occlude some of the drawn strokes, and affect the edge profile and the stroke point analysis. Hence, to obtain meaningful strokes points, we continuously track the positions of pen and hand in the video stream, and perform the stroke point analysis (Section 3.4.1) only on the non-occluded image region. In other words, we postpone the stroke point analysis until the occlusion is clear. In detail, we estimate the pen and hand occluding regions as follows:

- By the pen-tip localization method in Section 3.3, we can always locate the pen tip over the video frames. Hence, we can approximate the pen occluding region by a small circular region around the pen tip (radius equal to the size of the pen-tip template), see the gray circle in Figure 3.5(c).
- To estimate the hand occluding regions, we first convert the colors in each frame to the HSV color space. Then, we perform a pixel-level classification based on [107] to determine the skin color, and generate an initial mask of the hand. Note that the V channel is ignored in the computation to account for illumination invariance. By further applying morphological post-processing operations such as erosion and closing, we can obtain a clean mask of the user's hand as shown in Figure 3.5(c). Note also that we experimented our method with three users of different skin colors, and obtained the following threshold ranges in the classification model for hand tracking: $0 \leq \mathit{hue} \leq 10^\circ$ and $0.08 \leq \mathit{saturation} \leq 0.59$, see Figure 3.5(a). Other methods that combine color and texture features of human skin [108] may also be used here.

3.5 Spatio-temporal Grouping

Given a set of strokes points **S** obtained from the previous stage, our next task is to group them and form meaningful drawing strokes that are spatio-temporally coherent. To do so, we first need to remove redundant points with exactly the same spatial coordinates. Note that redundant points could be produced if the pen tip stays on the same image location for more than one video frame.

Three criteria are important in combining the stroke points to form longer strokes:

i) **Edge Connectivity.** This criterion explores the presence/absence of an actual stroke on paper between (p_i, t_i) and (p_j, t_j). Recall that we used the $\mathbf{L} \circ \mathbf{G}$ filter response to retain only a subset of the pen-tip trajectory that corresponds to drawn strokes in Section 3.4.1, moreover, we have the temporal information to determine the ordering of stroke points. Hence, this criteria is already satisfied since stroke points which are not part of the drawn strokes on paper are removed, and those that are close to each other spatially and temporally represent actual connected drawn strokes on paper.

ii) **Spatial Proximity**. The second criterion explores the spatial closeness between (p_i, t_i) and (p_j, t_j) since we should not group stroke points that are too far away from each other. To satisfy this criterion, we can measure the Euclidean distance between the two stroke points, and ensure that far away stroke points do not form a continuous stroke.

iii) **Temporal Proximity**. The third criterion is similar to the second one, but it explores the temporal proximity between (p_i, t_i) and (p_j, t_j), since we do not want to merge temporally non-continuous stroke points into a single stroke. Note that this is the most important criterion in reconstructing strokes that are free of the various ambiguities highlighted in Figure 3.1.

To group the stroke points based on these criteria, we adopt the agglomerative hierarchical clustering approach. In detail, we assign each stroke point to a separate cluster initially, and iteratively combine clusters to form a hierarchy using the single linkage function give by: $\min_{i \in A, j \in B} d(i, j)$ where A and B are clusters and d is the Euclidean distance. We finally form flat clusters using a

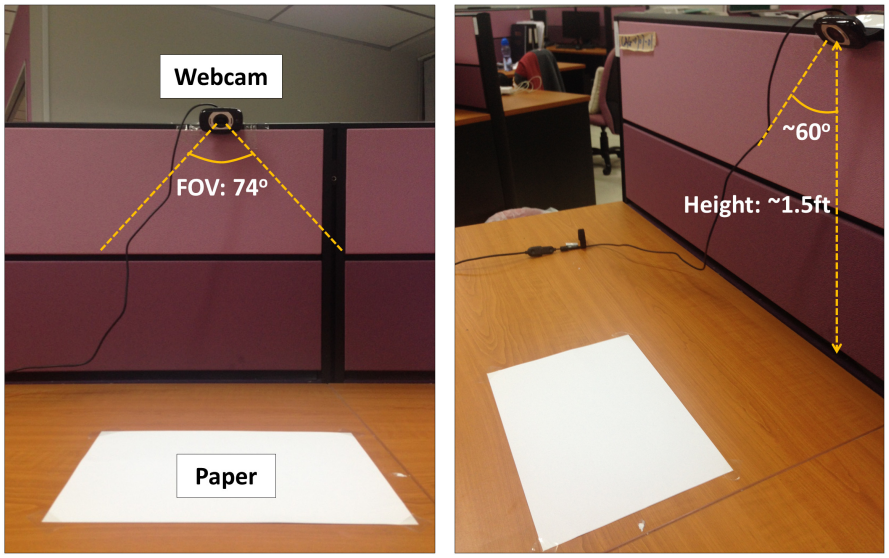


Figure 3.7: *System setup from two different views.*

cutoff threshold which is set to be 1.5∼2 in practice. Once we examine and group each pair of temporally-consecutive points to form longer strokes, we then vectorize each of them by cubic spline fitting.

3.6 Results and Evaluation

3.6.1 Implementation

We implement and run our method on a desktop computer with a 3.2GHz Intel i7 CPU, 12 GB memory, and 64-bit Windows 7. We employ the Logitech C615 webcam to capture video streams at 480p/720p and 30 frames/sec. See Figure 3.7: the webcam is fixed at ∼45cm above the drawing space opposite to the user at an evaluation angle of ∼60 degree and a field-of-view of 74 degree.

Estimating the homography transformation between the webcam view and paper is a one-time process: During the system initialization, we first identify the largest white blob in the image, and then use Hough line transform to extract straight lines around the blob's boundary edges. After that, we compute line intersections to obtain the paper corners, show them to the user for confirmation, and then estimate the homography matrix by assuming the paper dimension. We implement our software in Python 2.7, and use the following library APIs: OpenCV [109], NumPy, and SciPy [110] for vision-based

Table 3.1: *Evaluation of performance: average time taken to complete the various stages in our method.*

Stage	Method	Average Time (seconds)	
		CHEETAH	HELLO WORLD
1	Pen-tip localization	0.035	0.035s
2a	Detecting stroke points	0.012s	0.012s
2b	Handling occlusions	0.009s	0.008s
3	Spatio-temporal grouping	0.012s	0.012s

methods and numerical computation.

3.6.2 Evaluation: Performance

First, we perform an experiment to evaluate the performance of our method in delivering interactive online computation. To do so, we measure the time taken (in sec.) to process each video frame (resolution 720p) for each stage in the pipeline (see again Figure 3.2). Two drawing cases, CHEETAH and HELLO-WORLD (see Figure 3.10), with totally 7680 and 3150 frames, respectively, are used here. The average processing time taken by each stage is reported in Table 3.1. From the performance results, we see that the entire computation consistently takes around 0.07-0.08 sec. per frame, showing that it can attain interactive performance.

3.6.3 Evaluation: Accuracy

Second, we evaluate the accuracy of our results by comparing them with the actual strokes drawn on paper, as well as with the vectorization results produced from the commercial tool WinTopo Professional [27] (with the default option *One-Touch Vectorization* in our experiments). The six line drawings shown in Figure 3.10 are used in our evaluation and the following four criteria are used:

i) Number of Strokes. First, we compare the total number of actual strokes drawn on paper against the total number of strokes vectorized by our method, and also by WinTopo, see Table 3.2. From the results, we can see that our online method is able to accurately recognize all the user-drawn strokes without missing any stroke. In sharp contrast, WinTopo produces a lot more strokes

drawing	# actual strokes	# reconstructed		overlap %	avg. deviation
		ours	WinTopo		
CAT	44	44	74	88.77	1.568
GIRL	24	24	43	88.50	1.520
HELLO-WORLD	6	6	30	94.89	1.215
OLD-MAN	42	42	50	86.29	2.957
DUCK	17	17	35	87.34	2.135
CHEETAH	25	25	42	83.66	1.994

Table 3.2: *Evaluation of accuracy: the number of reconstructed strokes, overlap percentage, and average deviation.*

than the actual number of strokes drawn by user due to its deficiency in resolving various stroke recognition ambiguities, see again Figure 3.1.

ii) Overlap Percentage. Second, we evaluate how close our vectorization results are to the actual drawing. Here we first scan the paper drawing and use a simple binary segmentation to extract the actual strokes. Then, we transform and overlay our vectorized strokes onto the scanned image (through the recovered homography) and compute the overlap percentage: $|A \cap B|/|B|$, where A and B are sets of pixels covered by the scanned strokes and the vectorized strokes, respectively. Note that this measure aims to see how the vectorized strokes (actually one-pixel-wide lines) are contained within the actual strokes in the scanned image space.

From Table 3.2, we can see that our method can attain very high overlap percentage, but not yet perfect, because: 1) the homography recovered for computing the overlap percentage is not perfect, and 2) we use a discrete set of stroke points (captured in each frame) to reconstruct a stroke by spline-fitting. Since WinTopo vectorizes strokes by binary-thinning the scanned images, its vectorized results can always perfectly overlap with the original strokes. However, from Figure 3.10, we can still see how similar our vectorization results are with respect to the scanned drawings.

iii) Average Deviation. Third, we quantitatively measure the deviation of our reconstructed strokes from the scanned drawings in pixel units. Like overlap measure, this is done in the image space of the scanned drawings.

Let P and Q be the set of pixels belonging to the scanned strokes and our reconstructed strokes (1-pixel wide), respectively. We compute the average deviation between them as $dev(P, Q) = \sum_{q_i \in Q} \min \|q_i - p_j\| / |Q|$, where p_j is the

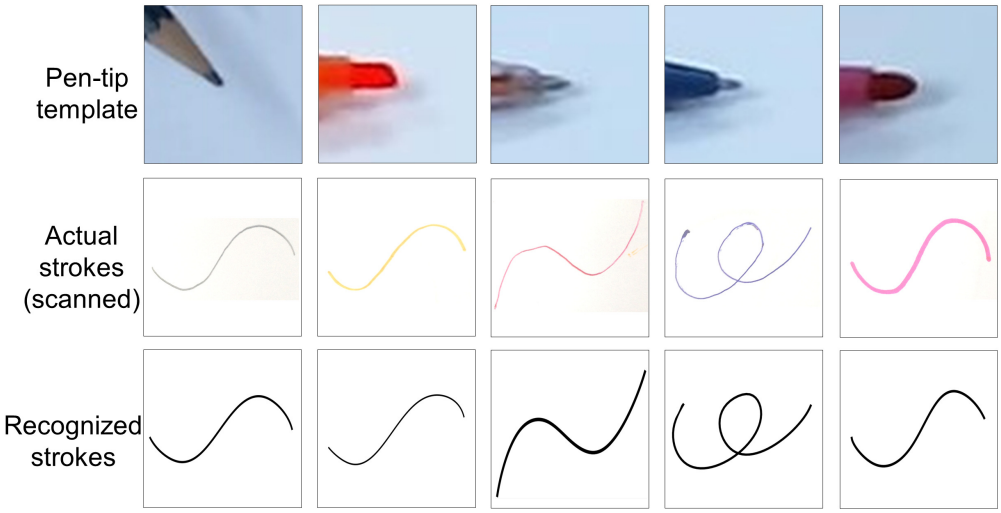


Figure 3.8: *Our method supports a variety of pens and pencils. Top row: different pen-tip templates; middle row: actual strokes drawn on paper; and bottom row: corresponding strokes recognized by our method with the pen/pencil.*

pixel in P that is the nearest to q_i in the image space. From the results shown on the rightmost column of Table 3.2, we can see that the average deviation is consistently around 1 to 2 pixels, indicating that our results are sufficiently close to the original strokes. Note that the resolution of the scanned images used in this experiment is 877×637 pixels.

iv) Visual Comparison. Lastly, we randomly colorize the recognized strokes for both our results and WinTopo, and perform a visual comparison. Two comparison examples are shown in Figure 3.9. From the zoomed views shown in the figure, we can clearly see that WinTopo suffers from stroke recognition ambiguities, and tends to break strokes at junctions, thereby producing excessive strokes. In contrast, our method, which employs temporal information, can accurately reconstruct the long strokes while retaining the original stroke continuity and avoiding various stroke recognition ambiguities.

3.6.4 Evaluation: Occlusion Detection

Next, we explore and illustrate how occlusion detection helps to improve the stroke recognition. A typical case is presented in Figure 3.11. During the drawing process, hand and pen occlusions could seriously interfere the stroke point analysis process described in Section 3.4.1. See the drawing in Figure 3.11(a) and the resulting $\mathbf{L} \circ \mathbf{G}$ response in Figure 3.11(b). If we analyze this response

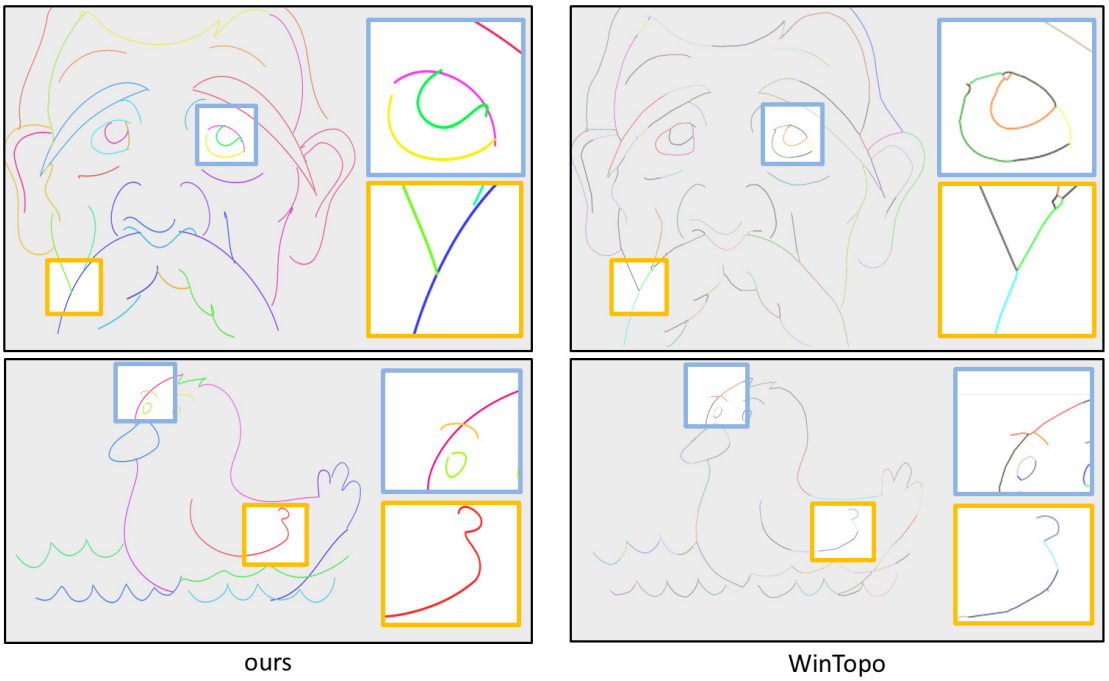


Figure 3.9: *Visual comparison of stroke reconstruction results: our method (left) and WinTopo (right).*

and extract the stroke points around the pen-tip trajectory at this moment (video frame #84), the analysis result would be error-prone.

Hence, we detect and track the hand and pen occlusion regions over the video frames, see Figure 3.11(c), and postpone the stroke point analysis and extraction till the occlusion is clear, see Figure 3.11(d). By this mechanism, we can improve the extraction quality while attaining full automation.

3.6.5 Evaluation: Vectorization Results

Figure 3.10 presents the vectorization results generated by our method and compares them side by side with scanned images of the original drawings. Notice the individual reconstructed strokes (each randomly-colored). They are free of the various stroke recognition ambiguities we mentioned in Section 3.1 (see also Figure 3.1); in addition, they can correspond nicely to the actual strokes drawn by the user. This demonstrates one key advantage of our online method in improving the stroke recognition quality.

Furthermore, we develop a video-scribing application, which renders the vectorized strokes on a virtual paper according to their spatio-temporal orders

Figure 3.10: *Vectorization results of six different line drawings generated by our method. In each box, we show the vectorization result with randomly-colored strokes on the right and a contrast-enhanced scanned image of the original drawing on the left. From left to right: the drawings are* OLD MAN, CAT, DUCK, GIRL, CHEETAH, *and* HELLO-WORLD*; our method recognizes 42, 44, 17, 24, 25, and 6 strokes from them, respectively.*

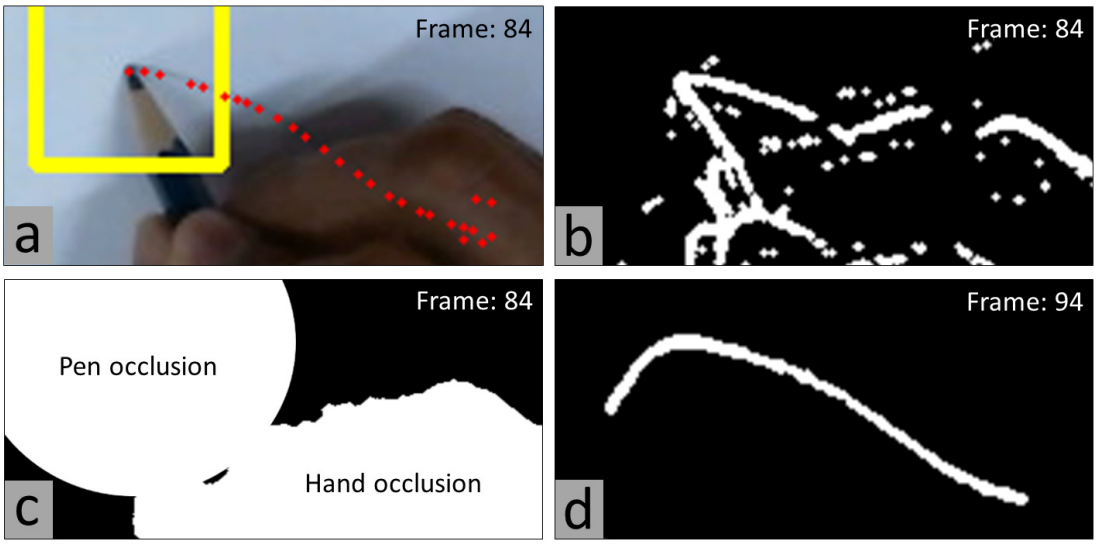


Figure 3.11: *Example of how occlusion could interfere the analysis of stroke points: (a) notice a number of stroke points are occluded by the hand and the pen-tip at the moment; (b)* $\boldsymbol{L} \circ \boldsymbol{G}$ *response affected by the occlusion; attempting to recognize stroke points in the presence of occlusion is error-prone; (c) occlusion mask computed by our method; and (d) valid* $\boldsymbol{L} \circ \boldsymbol{G}$ *response of the stroke after the occlusion is clear.*

Figure 3.12: *Video scribing. Our method enables us to easily re-render the vectorized strokes, which have temporal information, and produce a video scribing effect. The drawing video is spatio-temporal coherent with the original drawing by the user but goes without the pen and the user's hand.*

but without the pen and user's hand, see Figure 3.12 for image snapshots. By this, we can simulate and visualize the actual drawing process, and create visually-pleasing animation effects.

3.6.6 Variety of Pens and Pencils

Our method can support a variety of pens and pencils. To include a new pen/pencil to stroke recognition, the user may present picture of the pen tip in front of the camera, and then apply our interface to crop the picture and create a pen-tip template. After that, our method can employ the pen-tip template to automatically locate the pen-tip trajectory. Figure 3.8 shows a simple example where the top, middle and bottom rows in the figure show the pen-tip templates, scanned image of a drawn stroke, the vectorization results, respectively.

3.7 Summary

This chapter has presented a novel interactive camera-based method to capture and vectorize freehand line drawings on paper. There are three key contributions in this work. First, we propose a robust spatio-temporal tracking technique that can efficiently localize and track the pen tip in video frames. This technique combines the strength of a fast-cascade classifier with discriminable features and optical-flow tracking (which is less accurate) to achieve high performance and accuracy. Second, we extract stroke points in the video frames by considering not only the edge profiles around the estimated pen-tip trajectory but also the hand and pen occlusion to improve the stroke recognition accuracy. Lastly, we employ clustering to produce meaningful strokes that are spatio-temporally coherent with the actual strokes drawn by the users. As for the evaluation, we perform a number of experiments to examine different aspects of our method: computation performance, accuracy against actual drawings and an offline commercial tool for vectorization, as well as pen and hand occlusion. In the end, we use the method to produce assorted vectorization results of clean line drawings, as well as apply it to produce the video-scribing animation effect with some of our results.

Limitations

Our technique has certain limitations that we are worth discussing for future investigations. First, we assume that the paper is fixed in the drawing space to considerably simplify the problem. However, it is common for artists to rotate the paper for convenience when drawing. To support this, we need a procedure to estimate the paper orientation in each video frame to support further analysis by our method. Second, our current method can only support clean strokes, and is hence potentially applicable to the cleaning up or inking stage. Sketchy strokes that are crowded and drawn in close proximity, are common in the initial stages of cartooning when the artist is still experimenting with the design. However, since our method estimates the pen-paper contact by analyzing the pencil marks (or ink trail) on paper, this becomes unreliable when strokes are densely crowded. Third, there are some corner cases where our technique can fail, e.g., drawing a stroke so fast that it hinders the pen tip tracking, drawing light strokes that are too dull to stand out from the paper, and occluding the pen tip thereby preventing its tracking. Lastly, while our method resolves several ambiguities faced by offline vectorization methods with reasonable accuracy in vectorization, this can be further improved for demanding applications. A hybrid approach using both offline and online techniques can help with the above two problems. Such an approach has been recently explored for decomposing paintings into editable layers by analyzing the video of the painting process in an offline step [111]. However, our problem is more challenging due to need for accurate stroke correspondence between online and offline estimations, as well as dealing with occlusions, and is worth investigating in future. Such a hybrid approach can also help to improve the efficiency of complex offline vectorization algorithms. Detailed solutions that can potentially address these limitations are discussed in Chapter 7.

CHAPTER 4

Globally Consistent Wrinkle-Aware Shading of Line Drawings*

4.1 Introduction

Shading can dramatically change the atmosphere and style of a 2D drawing, and is an important step in both comics and animation production. Once the line drawing is digitized and/or vectorized, for e.g., using the technique presented in Chapter 3, the next step in the cartooning pipeline is to shade the drawing. Artists currently achieve this by imagining the 3D shape of the object being depicted and the light source location, and manually drawing the shading regions, followed by coloring them, or laying screentones based on the desired visual style. While such process can be automated with the presence of geometry, artists are very unlikely to model the geometry for every single drawing. Therefore, it would be desirable to infer certain geometry information directly from the artists' clean line drawing, automate the shading, and save the artists from the tedious shading process, so that they can focus on the design.

Early methods for geometric modeling of line drawings [112, 56, 113, 55, 114, 115] seem to be applicable for our need. However, these methods are designed mainly for line drawings of rigid polyhedral shapes in orthographic view. In our case, arbitrary 2D drawings generally depict freeform objects with no guarantee on the physical correctness.

On the other hand, sketch-based modeling methods [116, 117, 118] may be usable, but they are too demanding for our shading purpose as they aim to create a complete 3D geometry, requiring much user annotation. Note that traditional 2D cartooning is a highly labor intensive process with a very tight schedule. An approximate view-dependent proxy geometry would be suffi-

*This author was the primary contributor who conceived the research idea, performed literature survey, carried out the implementation, and conducted the experiments. This work is in collaboration with a research group in The Chinese University of Hong Kong.

Figure 4.1: *Even with just a few strokes, line drawings can illustrate nontrivial geometric features that are (perceptually) not difficult to recognize.*

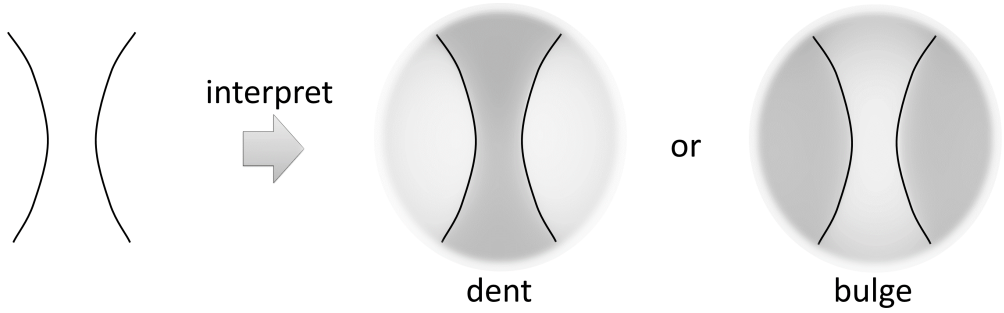


Figure 4.2: *Ambiguity in interpreting the two wrinkle strokes shown on the left. Locally, we may interpret the region inbetween the strokes as a dent (mid) or as a bulge (right). Note that white and black colors indicate near and far, respectively.*

cient to support the shading, similar to [78]. However, their method considers mainly the region contour, while we need to consider interior strokes, which depict intra-region bulges and dents around wrinkle strokes in complex drawings, so that we can compute and generate the shading.

Besides, some methods aim to recover normal field [80, 84, 85] or proxy geometry [75, 76] from line drawings; however, they are not very successful in analyzing the semantics of interior strokes, e.g., see the wrinkle strokes in Figure 4.1. As their interpretation of strokes is either user-driven or handled locally without a global consideration, the inferred local geometry may not be consistent over the drawing. In particular, we may have ambiguities in the local interpretation, see Figure 4.2. Such crowded wrinkle strokes are not rare in complex 2D drawings, unfortunately.

To automate shading production on line drawings with wrinkles, we develop a series of computational methods, as summarized in the following novel technical components:

- First, we adopt and model five different perceptual cues, including T-junction, convexity, continuity, proximity, and regularity, to interpret the local geometric meaning of wrinkle strokes by exploring relevant psychological principles. These cues analyze not only stroke shapes and connections but also their spatial arrangement and interaction in line drawings.
- Second, we formulate a global optimization to the stroke interpretation problem, aiming at maximizing the fulfilment of local interpretations and minimizing the inconsistencies across interpretations over the drawing.
- Third, we present a wrinkle-aware inflation method to estimate the partial geometry and produce shading on the input line drawing. Our inflation method can help generate two commonly-used shading styles: soft shading, which imbues a 3D-like feel, and 2D shading, which is commonly seen in traditional cartoon and manga (see Figure 4.3 and Figure 4.4).

To demonstrate the applicability and effectiveness of our method, we show various shading results on different kinds of line drawings, and conduct different experiments to evaluate and compare against state-of-the-art methods.

4.2 Overview

Our Goal

The input to our method is a set of strokes (including both interior and boundary strokes) in a vectorized clean line drawing, e.g., see Figure 4.3(a). These strokes are purposely drawn by hand to illustrate various geometric features such as bulge, dent, and silhouette, while each of the strokes is simply given as a 2D polyline, without any label that indicates the geometric features.

Given such input, our goal in this work is to analyze wrinkle strokes in line

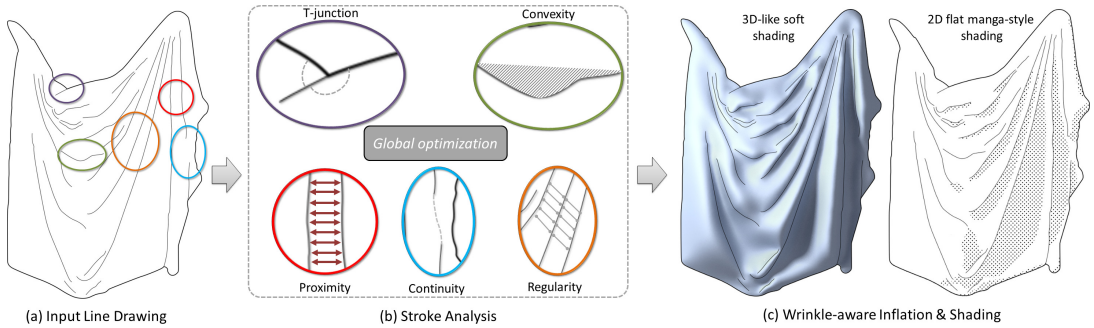


Figure 4.3: *Overview. Our method takes a 2D line drawing as input (a), and analyzes various perceptual cues to estimate local geometric information around each stroke (b). After that, we combine these local cues via an optimization model to maximize the satisfaction over the local inference and to obtain a globally-consistent stroke interpretation. Lastly, we perform wrinkle-aware inflation to generate a proxy geometry that can be used to generate shading (c).*

drawings, including how they spatially interact with one another, so that we can estimate a partial geometry information to automate the creation of various shading styles on line drawings.

The Challenges

To produce shading with wrinkles, we at least need to roughly locate where to shade and how much to shade. To support such computation, we have to estimate certain local geometry information around the strokes, even if we do not require physical correctness. This is a challenging problem as we are given only a single static line drawing without temporal information, as opposed to an animated sequence of drawings or a set of video frames in Zhang et al. [99] and Liu et al.'s work [61]). Our input strokes are simply 2D polylines without labels and user annotations. To resolve this, we can consider the following subproblems:

First, we have to estimate the local geometry around each stroke, specifically, i) what is the local gradient around the stroke, ii) whether the stroke is occluding or non-occluding, and iii) any specific geometric feature around the stroke, e.g., T-junction, etc. To support the analysis, T-junctions can help to estimate the local geometry [61, 62], but T-junctions are insufficient for resolving the problem, since most of them are around the object boundary, while wrinkles in line drawings often occur in the interior, see Figure 4.1 (left).

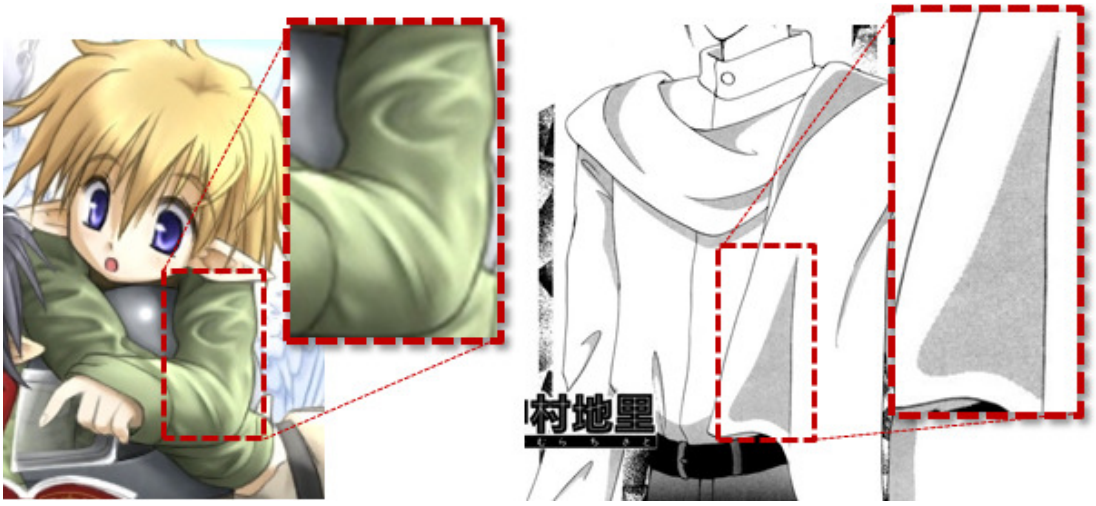


Figure 4.4: *Common shading styles. 3D-like soft shading makes use of gradients and soft shadows (left) [119], while 2D-flat shading (e.g., as in manga) has hard shading region boundaries. Also note how the shading region seldom crosses over strokes.*

Second, we need to integrate local geometry information estimated from individual strokes, so that local stroke interpretation can be spatially consistent with one another for generating a partial geometry to support the shading. If we consider only local cues around individual strokes, the estimated geometry may be ambiguous and/or featureless.

Lastly, we need to consider the shading style characteristics. In general, there are two types of shading (see Figure 4.4):

- *Soft shading* resembles 3D shading and is created with color gradients. Such a style has a smooth transition from dark to light across the shading region and provides a nice 3D perception; it is however harder to create manually in 2D space since the shading involves gradients.

- *2D shading* seen in traditional cartoon and manga is different from conventional 3D shading. Here, the shading region has hard boundary, with solid color or texture and usually does not go across the strokes in the drawing.

Our Approach

We approach the above subproblems through a novel three-stage computation pipeline as shown in Figure 4.3.

Given an input 2D line drawing (Figure 4.3a), our method analyzes not only T-junctions but also several other perceptual cues by consulting relevant principles from Gestalt psychology. These include cues on individual strokes (convexity), as well as cues on pairs and groups of strokes (proximity, continuity, and regularity).

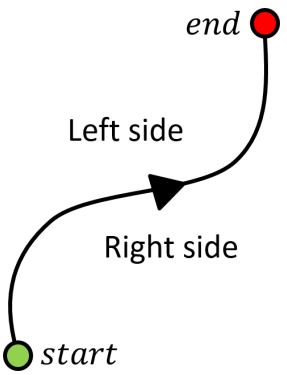
Next, to obtain a globally-consistent stroke interpretation over the perceptual cues, we formulate a novel optimization model to combine and balance the local stroke inference results (Figure 4.3b). Here, we define energy functions to quantify individual cues, and integrate them into a global energy function, so that we can maximize the perceptual consistency over the drawing.

Lastly, to produce shading, we develop a wrinkle-aware inflation method to estimate a partial geometry by constructing a mesh and estimating its z-coordinates (a height field) from the local depth profiles inferred around strokes (Figure 4.3c). We design our inflation framework such that the reconstructed mesh will satisfy the shading characteristics of the desired shading style, i.e., 3D-like soft shading or 2D flat shading as mentioned above.

4.3 Inferring Stroke Gradients

4.3.1 Simplified Stroke Model for Wrinkles

We consider two kinds of strokes in our input: *boundary* and *interior*. For boundary strokes, which enclose the object in drawing, we employ straightforward boundary conditions to model their geometric behavior, see Section 4.4. For interior strokes (each denoted as s_i), we use a simplified stroke model to describe their depth profiles. Note that like freeform 2D drawings, this model is not for physical correctness, but for estimating perceptual information, so that we can later estimate partial



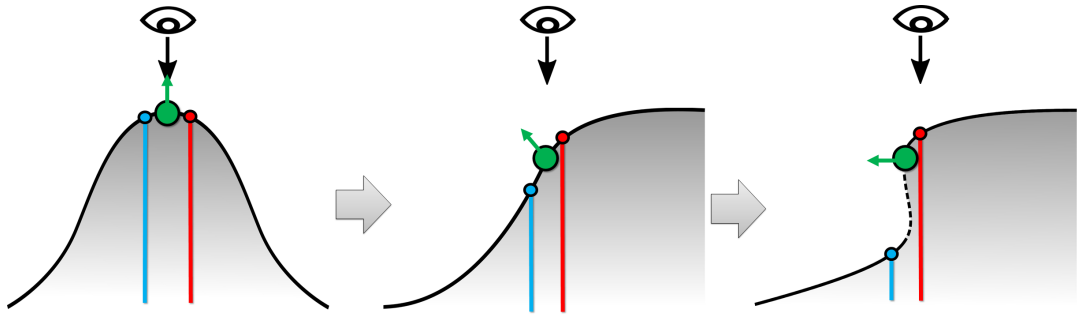


Figure 4.5: *These cross-sectional views illustrate transitions across depth profiles in the simplified stroke model, where the green dot refers to a point on the stroke. In this model, when $|\kappa_i|$ is small, the stroke is interpreted as a ridge or valley, and when $|\kappa_i|$ increases, the local depth profile gradually transits to a suggestive contour, and then to an occluding contour with local depth discontinuity.*

geometry for generating shading.

On a wrinkled surface, different surface regions are deformed to different extent. So, a line drawing can have ridge/valley strokes, suggestive strokes, as well as occluding strokes (see Figure 4.5), depending on the amount of depth difference, or gradient, across the stroke. Our simplified stroke model considers a smooth transition across these depth profiles, and employs a signed value, denoted as $\kappa_i \in [-1, +1]$ (for stroke s_i), to describe the transition. In particular, the sign of κ_i can indicate the gradient direction, or equivalently which side of the stroke is higher or lower in the depth profile while its magnitude can indicate the relative amount of depth difference. We also define the orientation reference (left and right sides) of each stroke by traveling along the stroke from its start to end point (see inset figure); here, κ_i is said to be *positive* if the *left side* of the stroke is *higher*, and vice versa.

Next, we explore five different perceptual cues to obtain local hints about κ_i (Section 4.3.2), and then formulate an optimization model to combine these local hints to infer a globally-consistent stroke interpretation (Sections 4.3.3).

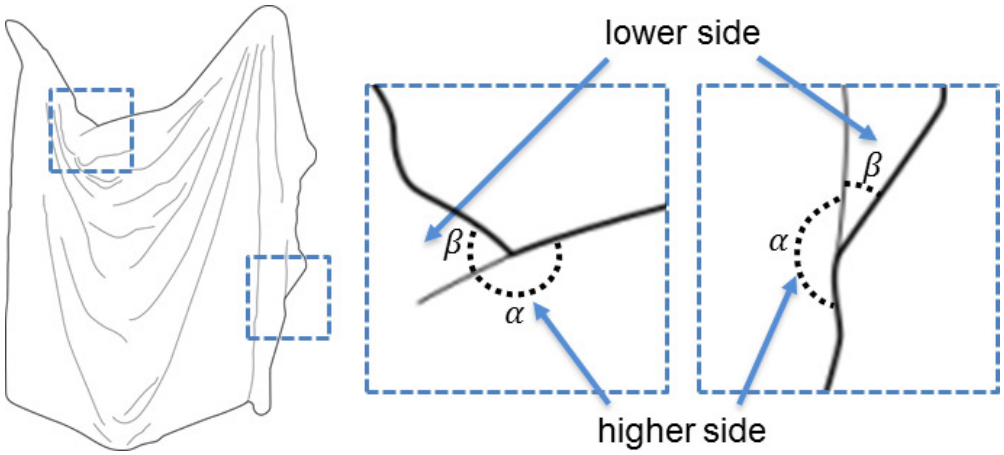


Figure 4.6: *Perceptual Cue: T-junction.*

4.3.2 Perceptual Cues

T-junction

A T-junction is formed when an endpoint of a stroke lies on another stroke. Perceptually, this suggests a local occlusion, see the *amodal completion law* [58]: we tend to interpret the interrupted curve as the boundary of some object undergoing an occlusion, i.e., the side next to the interrupted curve is lower than the side next to the other curve, see Figure 4.6.

We explored two metrics for T-junction based on the junction angles: [61] and [62]. We adopt the latter one since we found it to be more robust for our case in experiments. In detail, we compute the T-junction cue as:

$$T_i = \frac{1}{\pi/2}\Big[\ |(\alpha \bmod \pi) - \frac{\pi}{2}| - |(\beta \bmod \pi) - \frac{\pi}{2}|\ \Big], \tag{4.1}$$

where α and β are the junction angles (see again Figure 4.6) and mod is the modulus operator. The range of T_i is $[-1, +1]$: a value close to zero indicates a weak T-junction (e.g., when $\alpha \approx \beta$), while a value close to +1 (or -1) indicates a strong T-junction with the side on angle α (or β) being higher than the other. Note that for strokes that do not associate with any T-junction, T_i is zero.

Convexity

Convex shapes could be perceptually associated to figure or foreground regions in Gestalt psychology [120]. Several works have employed this cue to

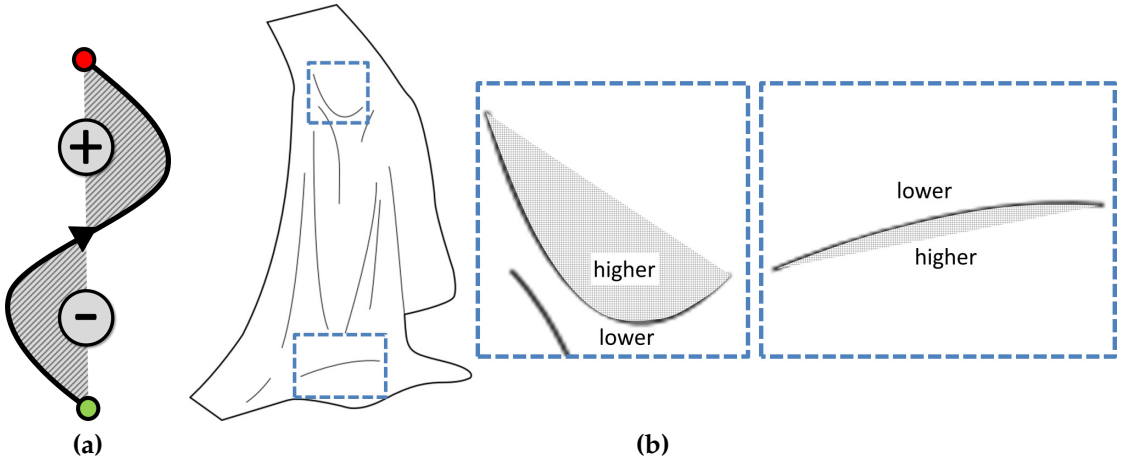


Figure 4.7: *(a) Inflections give rise to ambiguity in perceiving the depth across a stroke. (b) Perceptual Cue: Convexity.*

separate figure and ground [121] and to estimate local layering [80, 60]. For the case of wrinkles, see Figure 4.7, when a stroke bends towards one of its sides, this suggests the presence of a local bulge in the associated convex region. Hence, the bulge side of the stroke would be higher than the other. However, if the stroke contains an inflection (see Fig. 4.7(a)), ambiguity could arise in the result.

From these observations, we quantify the convexity cue based on the deviation (signed area) of a stroke (s_i) from the straight line (L_i) that joins s_i's endpoints (see again the above inset figure):

$$V_i = \texttt{clamp}(\ A_i/(\alpha A_0)\ ,\ [-1,+1]\ )\ , \tag{4.2}$$

where A_i is the signed area of s_i deviated from L_i, A_0 is the area of a semicircle with L_i as diameter, α is a parameter set to be 0.5 in all our experiments, and $\texttt{clamp}(\texttt{value}, [\texttt{min}, \texttt{max}])$ is a function that truncates the given value in the range $[\texttt{min}, \texttt{max}]$. Using this formulation, strokes with inflections receive small $|V_i|$, while convex strokes with single-sided bending receive large $|V_i|$. Again, positive V_i indicates that the left side of the stroke is higher, and vice versa.

Continuity

The Gestalt principle of good continuation [122, 58] suggests that human tends to perceive a curve as continuing along its established direction. Hence,

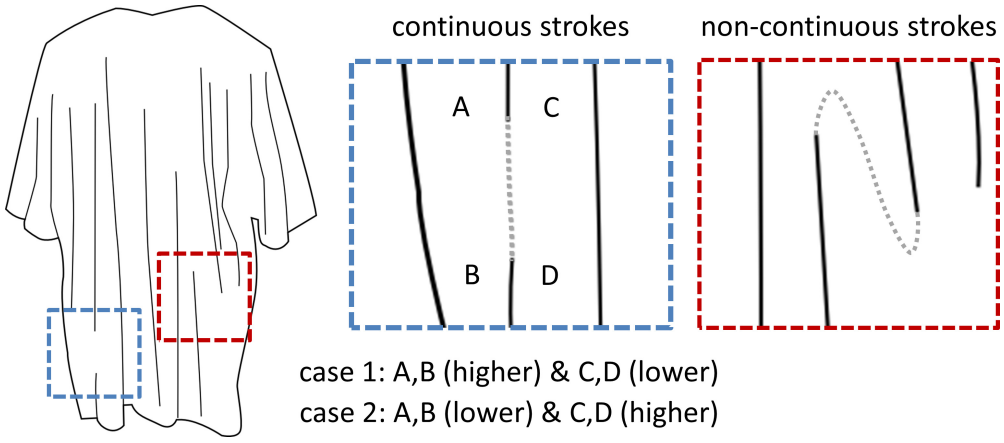


Figure 4.8: *Perceptual Cue: Continuity.*

strokes that appear to continue from each other are often perceptually seen as a single stroke, see Figure 4.8; in other words, they would appear to have similar depth profiles or gradients.

To estimate the continuity for a pair of strokes s_i and s_j, we first find a pair of stroke endpoints, one from each stroke, such that the distance between the two endpoints is the shortest among the four possible endpoint pairs. We then fit a cubic spline to connect them following the associated stroke tangent at each endpoint, see the gray dashed lines in Figure 4.8 (right). Next, we sum up the total absolute curvature $K_{i,j} = \int |k(s)|ds$ along the spline, where k is the curvature and s is the arclength parameter, and quantify the continuity term:

$$C_{i,j} = G(0; \frac{\pi}{4})(K_{i,j}) \ ,$$

where $G(\mu; \sigma)(\cdot)$ is a Gaussian kernel with mean μ and standard deviation σ for controlling the fall-off; hence, G helps to normalize $C_{i,j}$ to $[0, 1]$, such that when $K_{i,j}$ is zero, $C_{i,j}$ is one, and when $K_{i,j}$ increases, $C_{i,j}$ will gradually drop towards zero accordingly. Note also that we set $C_{i,j}$ to zero if the length of the spline is longer than the sum of the length of s_i and s_j, or the spline crosses (intersects) some other strokes in the line drawing.

Proximity

For wrinkles illustrated in line drawings, it is common to see spatially close and nearly parallel strokes that are perceived to press on one another in the drawing. Hence, the region in-between them would perceptually appear to

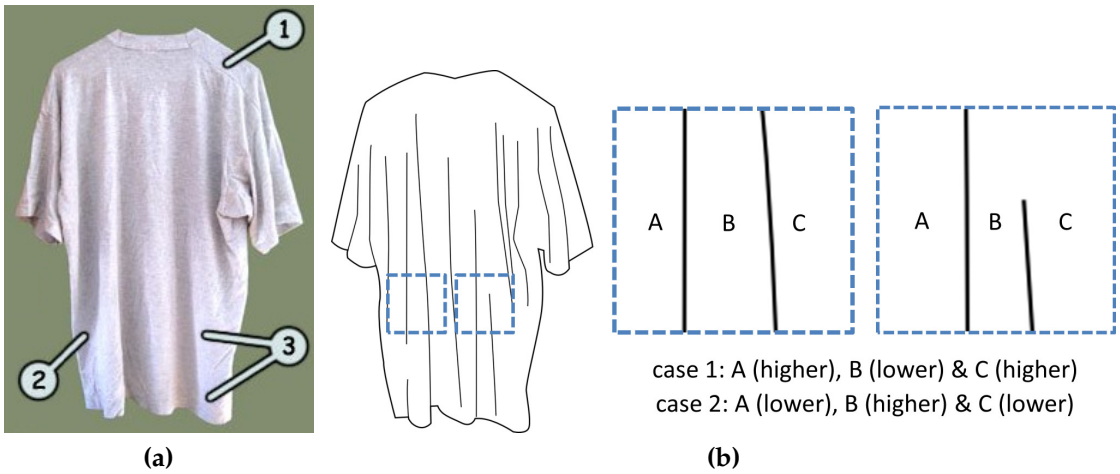
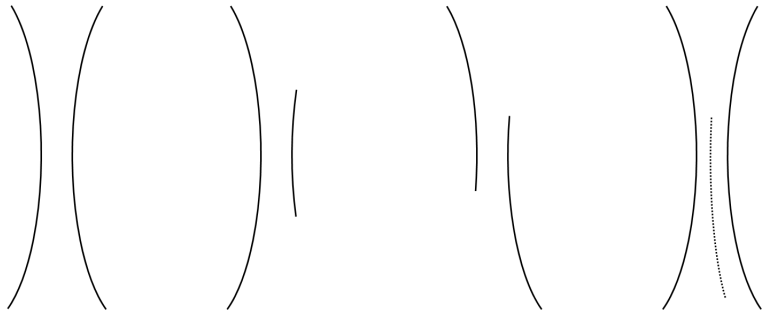


Figure 4.9: *(a) Bulges and dents appearing alternatingly due to proximal strokes. (b) Perceptual Cue: Proximity.*

be a bulge or dent resulted from a local deformation, see the T-shirt example in Figure 4.9a. In this way, these bulges and dents would occur alternatively across the regions around the two strokes, see cases 1 and 2 illustrated in Figure 4.9b. See also the inset figure (markers 1 and 2 indicate dents, while marker 3 indicates a bulge), which is the reference photo that the artist employed for sketching Figure 4.9.

From observations, the strength of the proximity cue (or the amount of local deformation or gradient) is affected by: i) spatial proximity between strokes, i.e., the closer the stronger; ii) parallelity of the stroke pair; iii) the perception would be reduced (or even nullified) if there is another stroke in-between; and iv) the perception is unaffected even though the two strokes do not have exactly the same length (see box in Figure 4.9 (right)). To model the proximity cue according to these characteristics, commonly-used metrics such as Hausdorff distance [123] and Fréchet distance [124] would perform poorly, especially for (ii)-(iv), see Figure 4.10 for a quantitative comparison.

Our metric is computed as follows. Given strokes s_i and s_j, without loss of generality, we assume that s_i is shorter than or has the same length as s_j. Then, we sample N uniformly-spaced points along s_i, say p_k with $k \in [1, N]$, where p_1 and p_N are endpoints of s_i. For each p_k, we determine a point q_k on s_j, such that the distance between p_k and q_k, i.e., $||p_k - q_k||$, is minimized. Next, we define a binary function δ_k: δ_k is zero if the straight line segment joining p_k and q_k intersects some other strokes; otherwise, δ_k is one. From



H:	0.533	0.457	0.583	0.533
F:	0.653	0.616	0.601	0.653
1-P:	0.428	0.308	0.425	0.810

Figure 4.10: *Comparison of metrics for various forms of stroke pairs commonly found in wrinkle line drawings: Hausdorff distance (H), Fréchet distance (F), and (1 - P) where P is our proximity metric. Note that proximity is inversely proportional to the distance; in fact, the left three cases should be a strong proximity (low values) while the rightmost case should be weak (high values) due to occlusion.*

these, we compute the proximity cue $P_{i,j}$ (s_i w.r.t. s_j) by averaging the distance values modulated by δ_k and a Gaussian kernel:

$$P_{i,j} = \frac{1}{N} \sum_{p_k \in s_i} \delta_k \cdot G(0; 0.2) \left(\min_{q_k \in s_j} \|p_k - q_k\| \right), \tag{4.3}$$

where G is a Gaussian kernel with μ=0 and σ=0.2 for mapping distance values to $[0, 1]$. Note that if p_k and q_k are near and unoccluded, the mapped value will be close to one, and vice versa. For normalization purpose, we scale the input drawing, so that its larger side has one-unit length while the aspect ratio is preserved.

Regularity

When three or more similarly-shaped strokes align roughly in the same orientation with similar spacing in-between, see Figure 4.11, we tend to perceive them as a group with similar property, see the regularity gestalt [58]. In this situation, the strokes in the group would appear to portray similar depth profiles rather than alternating bulges and dents.

We consider the following criteria in modeling the regularity cue: i) the spacing in-between neighboring strokes, ii) the local depth profile suggested by the convexity cue (V_i) and T-junction cue (T_i), and iii) whether the strokes

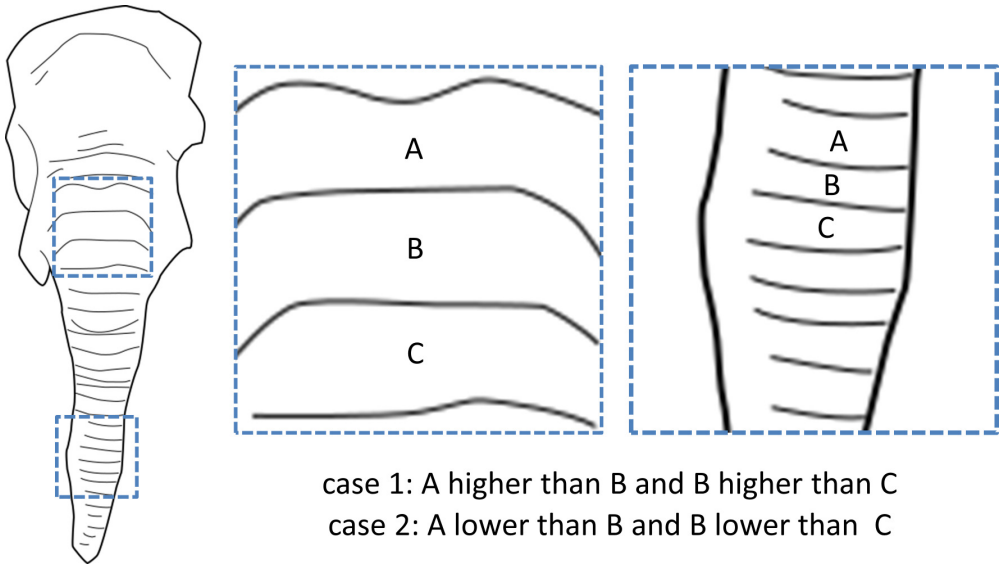


Figure 4.11: *Perceptual Cue: Regularity.*

align roughly in the same orientation. Before we formulate the cue, we first construct a stroke graph **G** to model the spatial proximity among strokes in the line drawing: each stroke s_i is represented as a node in **G** and an edge is added between nodes of s_i and s_j if P_{ij} is larger than a threshold (which is set to be 0.5) and the two strokes have similar depth profiles suggested by local cues V_i and T_i. For each edge, we compute l_{ij}, which is the size of the (average) spacing in-between s_i and s_j.

Next, we aim to find subgraphs with three or more nodes in **G** that have similar l_{ij} values. This can be done by a simple breadth-first traversal from each node in **G**. For each subgraph obtained, we compute the regularity cue for each of its connecting edges:

$$R_{i,j} = G(0; 0.2)(\mid l_{i,j} - \bar{l} \mid) \,, \tag{4.4}$$

where $\bar{l}$ is the median of $l_{i,j}$ among the edges in the subgraph, and G is a Gaussian kernel to put the regularity cue into range $[0, 1]$. In other words, $R_{i,j}$ measures how well each pair of strokes fits with other strokes within the same regularity group. For stroke pairs not in any regularity group, we set $R_{i,j} = 0$.

4.3.3 Optimization Model

Energy Terms

Next, we formulate an energy term for each perceptual cue to constrain the unknown variables κ_i accordingly:

T-junction Energy E_T measures how much the κ_i variables deviate from the corresponding T-junction cue estimations (T_i):

$$E_T = \sum_{i=0}^{N} |T_i| \cdot \|\kappa_i - T_i\|^2 , \tag{4.5}$$

where N is the number of strokes in the input. In Eq. 4.5, by multiplying $|T_i|$ inside, we can achieve the followings. If a stroke associates with a strong T-junction, $|T_i|$ will be large, so it will be harder for κ_i to deviate from T_i since our optimization model (to be presented later) minimizes E_T. On the other hand, if a stroke associates with a weak/ambiguous T-junction or simply does not associate with any T-junction, $|T_i|$ is small or zero, so κ_i can be more freely adjusted and may take a value far from T_i.

Convexity Energy E_V is formulated in a way similar to E_T, but it encourages κ_i to be close to the estimated convexity cue (V_i):

$$E_V = \sum_{i=0}^{N} |V_i| \cdot \|\kappa_i - V_i\|^2 . \tag{4.6}$$

Continuity Energy E_C encourages a stroke pair (say s_i and s_j) with high continuity value ($C_{i,j}$) to have similar depth profiles (κ_i), since they would perceptually appear to be the same stroke:

$$E_C = \sum_{i,j \text{ s.t. } i \neq j} C_{i,j} \cdot \phi(s_i, s_j) \cdot (\kappa_i \cdot \kappa_j) . \tag{4.7}$$

where ϕ has a value of either -1 or $+1$ to rectify the sign of κ_i and κ_j since the orientation of s_i and s_j may not match (see the inset figure in Section 4.3.1); it is set to $+1$ if the orientation of s_i and s_j do not match at the nearby endpoints and -1 otherwise. Note also that we multiply ϕ, κ_i and κ_j, so that when we minimize E_C in the optimization, κ_i and κ_j are encouraged to have the same

sign (both positive or both negative) if s_i and s_j have the same orientation, or opposite signs, if s_i and s_j have different orientations.

Proximity Energy, E_P encourages a stroke pair s_i and s_j with high proximity $P_{i,j}$ to have different signs, so that they can have different depth profiles with alternating bulges and dents:

$$E_P = \sum_{i,j \text{ s.t. } i \neq j} P_{i,j} \cdot (-\phi(s_i, s_j)) \cdot (\kappa_i \cdot \kappa_j) \ . \tag{4.8}$$

Note that E_P is formulated similar to E_C, but we use $-\phi$, so that we can encourage κ_i and κ_j to have opposite signs, and produce alternating depth profiles, when $|P_{i,j}|$ is large.

Regularity Energy, E_R encourages strokes in a regularity group to have similar depth profiles (κ_i) based on their affinity ($R_{i,j}$) to the group. We formulate this as a pairwise energy term similar to E_C:

$$E_R = \sum_{i,j \text{ s.t. } i \neq j} R_{i,j} \cdot \phi(s_i, s_j) \cdot (\kappa_i \cdot \kappa_j) \ . \tag{4.9}$$

Putting the terms together

To obtain a globally-consistent interpretation of the strokes over the line drawing, we need to combine and balance the estimations from individual cues (see Section 4.5 for comparison experiment) and seek a solution $\{ \kappa_i : \kappa_i \in [-1, 1]; \forall i \}$ that minimizes the combined energy terms:

$$\min_{\kappa_i \in [-1,+1]} \Big\{ \lambda_T E_T + \lambda_V E_V + \lambda_P E_P + \lambda_C E_C + \lambda_R E_R \Big\}, \tag{4.10}$$

where λ_T, λ_V, λ_P, λ_C, and λ_R are weights for balancing the influence of the energy terms. Since T-junction and regularity cues are empirically found to be stronger than the other cues, we set λ_T=λ_R=8, whereas λ_V=2 and λ_P=λ_C=5. In addition, since the objective function is quadratic but non-convex, we choose to solve it by sequential quadratic programming [125], which iteratively approximates and solves the objective function with a quadratic programming subproblem. In our experiments, the optimization usually converges in around 5 to 20 iterations.

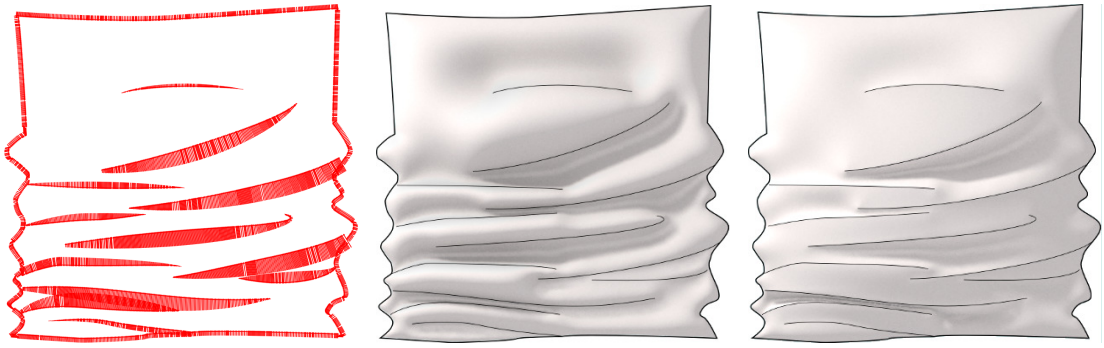


Figure 4.12: *Inflation with wrinkles. We set up a sparse gradient field based on the estimated depth profiles across strokes (a), and employ it to further estimate the 3D surface by propagating the gradients. The surface is designed to be smooth for soft shading (b), and flat on the upper side of strokes for 2D shading (c).*

4.4 Inflation with Wrinkles

After we estimate the depth profile (κ_i) of the strokes, we next need to recover partial 3D information to support the shading computation. For this purpose, we develop a wrinkle-aware inflation method to lift up the 2D drawing to 3D in three steps:

4.4.1 Sparse Stroke Gradients

Our first step is to define the gradient of the surface along each stroke using the estimated κ_i values. Here, the initial gradient defined along the strokes influences the style of shading that is achieved from the reconstructed mesh.

For soft/3D-like shading, the goal is the reconstruct a smooth and fair surface that will imbue a 3D-look to the line drawing when shaded. For this case, we set up the gradient along the stroke as follows:

1. compute 2D unit normal vectors $\mathbf{n}(s)=[n_x, n_y]^T$ of each stroke s_i (to the left side);
2. multiply $\mathbf{n}(s)$ with κ_i, so that $\kappa_i \cdot \mathbf{n}(s)$ aligns with the estimated gradient; and
3. compute the gradient along the strokes as $F^*(x,y) = w(s) \cdot \kappa_i \cdot \mathbf{n}(s)$, for each stroke s_i, where w_i is a weight function defined along a stroke to

smoothly attenuate the gradient vectors towards the stroke endpoints (excluding T-junctions), see the length of the (red) gradient vectors in Figure 4.12(a).

For 2D shading, e.g., manga style, recall that the shading region is defined by hard boundaries and seldom crosses over the strokes. While hard boundaries can be achieved by thresholding, ensuring that the shading region does not cross over the strokes is more challenging. Our key observation in achieving this effect is that, if the inflated surface is flat on the upper side of the stroke, but smooth on its lower side, then the shading/shadows will always be cast on the lower side without crossing over the stroke. To model this, we do the following:

1. split each stroke s_i into two strokes s_i^{up} and s_i^{low} which run along the original stroke in the image space, but are associated to the upper and lower side of the local region, respectively (based on κ_i). As a result, we can constrain the gradient field separately on different sides of the strokes.

2. compute the gradient along each stroke using steps 1 & 2 above for soft shading, and assign this to the lower side s_i^{low};

3. define gradient on s_i^{up} as $F^*(x,y) = [0,0]^T$ so that the upper side across each stroke is flat.

For boundary $\partial\Omega$, we define the gradient using inward-pointing normals: $F^*(x,y) = (-b(s) \cdot \mathbf{n}(s))$, where b modulates the length of the normal vectors, so that we can control the local shape near the boundary when estimating the surface geometry. For example, if $b \approx 0$, the surface is locally flat. For each of our results, b is set as a constant value based on the perceived local shape near the contour.

4.4.2 Dense Gradient Field

Our next step is to obtain a dense gradient field $F(x,y)$ over the 2D object region Ω using the sparse gradients defined above. We propagate these sparse gradient vectors by minimizing the following energy functional:

$$\min_F \int_\Omega |\nabla F(x,y)|^2 + |\nabla \times F(x,y)|^2 \, dA \tag{4.11}$$

subject to:

$$F(x,y) = F^*(x,y), \quad \forall (x,y) \in \{s_i\} \cup \partial\Omega \, .$$

Here ∇ is the gradient operator, $\nabla\times$ is the curl operator, and dA is an infinitesimal area in Ω. The first term, seeks a dense gradient field $F(x,y)$ that smoothly interpolates the sparse gradient vectors defined along the strokes and the boundary. Note that this is equivalent to seeking a zero Laplacian i.e. $\Delta F(x,y) = 0$. The second term minimizes the curl so that F is integrable. Visually, minimizing the curl ensures that the constrained gradients are retained as such in F, and spatially propagate farther throughout Ω.

4.4.3 Surface from Gradient Field

Next, we aim to estimate an approximate height field f, whose gradient matches the dense gradient field computed above. Ideally, our goal is to find the surface f that solves $\nabla f = F$. However, such a surface might not exist, since the gradient field F is often not perfectly integrable. Common approaches to alleviate this problem include projecting the gradient field into an integrable space of basis functions, e.g., Fourier basis [126], enforcing zero curl constraints [127], etc. Note that in our case, we had already enforced a soft constraint to minimize the curl in a least square sense in Equation 4.11. Now, we apply the divergence operator as in the work of Kazhdan et al. [128], and seek a surface f whose divergence of gradient i.e., the Laplacian, matches the divergence of the vector field F: $\Delta f = \nabla \cdot F$. In detail, we minimize:

$$\min_f \int_\Omega |\Delta f - \nabla \cdot F|^2 + \lambda |f - f^*|^2 dA \, . \tag{4.12}$$

subject to:

$$f = f^*, \quad \forall f \in \{s_i\} \cup \partial\Omega$$

Here, the first term seeks surface f whose gradient approximates F, whereas the second term constrains f to remain close to f^* (a height function) along strokes and boundary; this can be set to 0 for objects that are flat overall. Otherwise, f^* is estimated by solving the Poisson equation: $-\Delta f^*(x,y) = c, \ \forall (x,y) \in \Omega \setminus \partial\Omega$ subject to Dirichlet, i.e., fix the height function to 0 to simulate a bulge near the boundary, and/or Neumann boundary conditions, i.e., fix the normal derivative of the height function to 0 to simulate thin de-

formable material such as cloth. In this equation, c controls f^* similar to the inflation model in [78]. The parameter λ controls how to close to $f*$ would we like f to be and is set in the range $0.0 - 0.001$. The user may opt for Neumann boundary condition along the deformable portions of the boundary, e.g., see the bottom boundary of the cloth in Figure 4.1 (left). By this method, we can estimate a wrinkled surface f as a height field to support the shading computation, see a result in Figure 4.12(b).

Discretization Details

We solve Eq. 4.11 & Eq. 4.12 numerically by triangulating Ω into a mesh [129] with the following constraints: mesh edges pass through the strokes in the drawing and mesh faces have similar areas and angles. We discretize the differential operators at each vertex of the mesh v_i by standard linear finite element formulations that depend on the one-ring neighbors $N(i)$ to reduce Eqs. 4.11 & 4.12 to systems of linear equations. In detail, the Laplacian at each vertex v_i is defined as [130, 131]:

$$\Delta(v_i) \approx \frac{1}{A_i} \cdot \frac{1}{2} \sum_{ij \in N(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(v_j - v_i)$$

where ij refers to one-ring neighbor edges of v_i, α_{ij} and β_{ij} are angles opposite to the edge ij on either side, and A_i is the voronoi area at v_i that accounts for differences in mesh sampling.

The discrete curl used in Equation 4.11 is defined as [132]:

$$\nabla \times F(v_i) \approx \sum_{ijk \in N(i)} F_{ijk} \cdot e_{jk}$$

where the sum is taken over triangles ijk incident on v_i, e_{jk} refers to edge opposite v_i, and F_{ijk} is the per-face value of the vector field F.

The discrete divergence in Equation 4.12 is defined as [133]:

$$\nabla \cdot F(v_i) \approx \frac{1}{2} \sum_{ijk \in N(i)} \cot \theta_{ij}(e_{ij} \cdot F_{ijk}) + \cot \theta_{ki}(e_{ki} \cdot F_{ijk})$$

where the sum is taken over triangles ijk incident on v_i, θ_* refers to angles opposite to associated edge e_*.

(a) (b) (c)

Figure 4.13: *Soft shading style. We first generate an initial grayscale image with Blinn-Phong shading (a), followed by an ambient occlusion map (b). These are combined and the original line drawing overlaid to generate the final soft shading result (c).*

4.4.4 Shading

The height field f constructed in the preceding subsection provides 3D information that can be used to determine the shading regions. Our inflation method can reconstruct a 3D mesh that can adhere to the characteristics of soft shading as well as 2D flat shading. This enables us to generate these shading styles in a straightforward fashion. Since the shading region depends on geometry of the depicted 3D object as well as the direction where the light comes from, we set up a directional light and aim to generate the shading as a grayscale shaded image, which can be multiplied with or overlaid on the original line drawing image to obtain the shaded result. Since our reconstructed mesh modifies the $z-$coordinates of the original line drawing, it is necessary to use a camera with orthographic projection with the viewing direction perpendicular to the plane of the input 2D line drawing so that that the lines correctly appear on their respective geometric features. Note also that textures that are painted on the input drawing can be directly mapped using the $x-$ and $y-$ coordinates of the mesh vertices since we do not modify them.

To obtain a soft shading style, the inflated mesh can be directly rendered using any available global illumination engine. We generate such a result (Figure 4.13c), by computing the ambient occlusion (Figure 4.13a), and Blinn-Phong shading and soft shadows (Figure 4.13b) using 3dsMax 2016. For some results, we optionally map some simple textures using planar projection along the z-direction.

Figure 4.14: *2D manga-style shading. We begin with an initial shading (a), and generate a binary mask (b), where we apply a screentone texture to obtain the final manga-style shading (c).*

To generate flat 2D style shading, we first generate a grayscale shading image as above, but with hard shadows (Figure 4.14a). Next, we convert it to a binary image by Otsu's thresholding [134] to obtain shading regions with hard boundaries (Figure 4.14b). Finally, we apply a screentone texture on the shading regions to generate a manga style shading effect (Figure 4.14c).

4.5 Results & Discussion

4.5.1 Implementation

We implemented and evaluated our method on a desktop computer with a 3.2GHz Intel Core i7 processor and 12 GB memory running Windows 7. Our method takes a single clean line drawing as input. The user could set up the values for a few global parameters such as the balancing weights in the optimization, boundary conditions, c in the Poisson equation for f^*, the direction of the light source, and the shading style, or else just use the default values. Then, our method automatically performs stroke analysis, global optimization, inflation and shading. Table 4.1 shows the timing statistics for various line drawings.

Table 4.1: *Time taken to generate shading from the line drawings.*

Input	Stroke Analysis	Optimization	Inflation	
			Triangles	Time
Cloth	11.6	0.012s	38,793	156.61s
T-shirt	35.8s	0.024s	32,543	78s
Tablecloth	39.5s	0.031s	30,326	80.98s
Jean	2m51s	0.106s	29,438	75.66s
Gown	4m24s	0.118s	33,806	97.15s

4.5.2 Shading Results

We present our shading results for several input 2D line drawings with wrinkle strokes in Figure 4.15. Here, the third & fifth and fourth & sixth columns show our shading results with soft shading and 2D manga shading style, respectively, generated with a directional light source placed at the top left.

4.5.3 Comparison

We compare our method with two closely-related recent works [78] and [83], which focus on generating shading with a single line drawing as inputs, see also Figure 4.16. In particular, we compare the visual details in the shading results for illustrating wrinkles, as well as the user interaction required by the method to generate the shading.

[78] proposed a largely automatic framework to reconstruct bas-relief meshes from images of line drawings and to support the generation of global illumination effect. While their method works well for simple cartoon images, they do not incorporate information from interior strokes in their inflation formulation, see top row in Figure 4.16. It is because their Poisson-equation-based boundary inflation method only generates convex shapes without considering the wrinkle geometry. Consequently, their geometry captures the global shading of the reconstructed convex shape, but not the local shading generated from wrinkles. While their method can support certain interior strokes with T-junctions by region subdivision, stitching and grafting (see the `BUNNY` result in Figure 6 in their paper), such an approach is not feasible for most of drawings with crowded wrinkle strokes.

[83] designed a user-driven method to generate shading on line drawings us-

Figure 4.15: *Shading results generated by our method. Column 1: input line drawings, Column 2 & 3: soft shading results, Columns 4 & 5: 2D manga shading results. Note that the light source is fixed at the top-left for Columns 2 & 4 and top-right for Columns 3 & 5, respectively.*

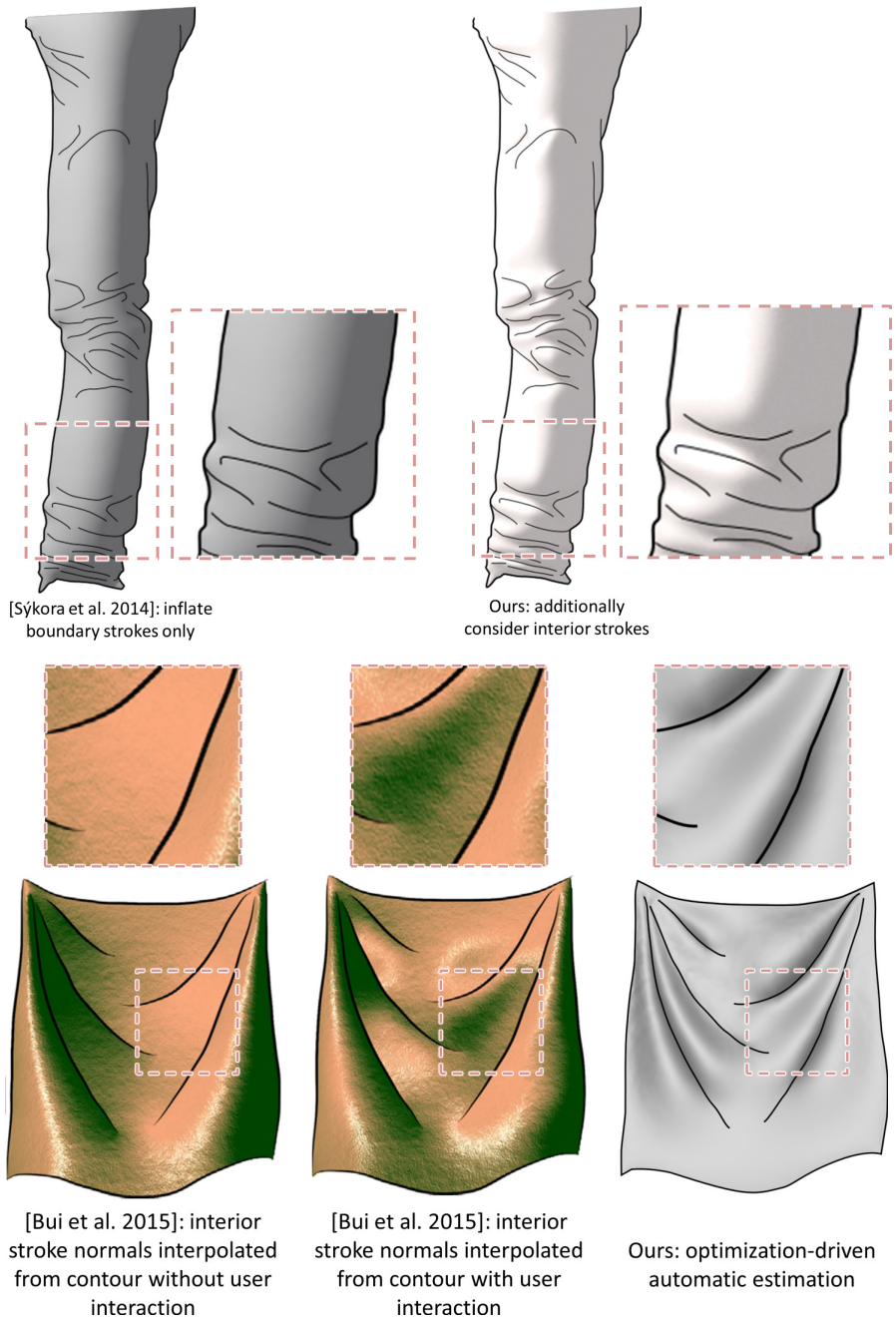


Figure 4.16: *Comparison of our results with [78] and [83].*

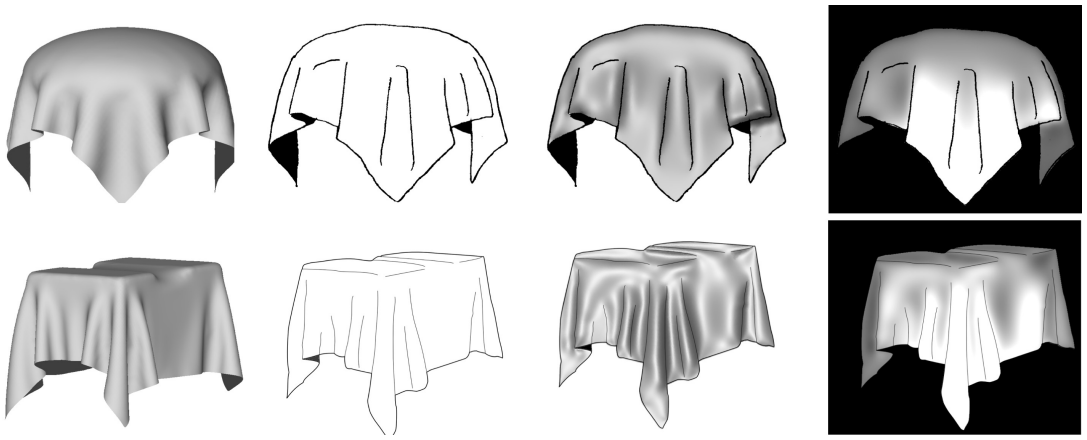


Figure 4.17: *Comparing results generated by our method (Column 3) from input line drawings (Column 2), created based on ground truth geometry (Column 1). Depth maps are visualized in Column 4.*

ing a hatching metaphor: users would sketch hatching patterns on the drawing that provide cues about the local normal orientation. In the absence of such information, their method smoothly interpolates the normals from the contour without considering the local interaction of neighboring strokes. The left and middle images in the bottom row of Figure 4.16 are taken from [83], showing the results without and with user interaction, respectively. In contrast, our method can automatically estimate a consistent geometry from the line drawing without such additional input, by analyzing the spatial interaction among the strokes in the drawing, see Figure 4.16 (bottom-right). Additionally, since our method reconstructs a proxy 3D mesh rather than just a normal field, we can generate better visual effects, e.g., ambient occlusion, ray cast shadows etc. that improve the richness of the shading.

Next, we perform an informal qualitative comparison with ground truth geometry, even though physical correctness is not our focus in this work. Two line drawings in Fig. 4.17(middle), were created based on the complex ground truth geometry shown in Fig. 4.17(left). Using these drawings as input, we generated height fields using our method, which are shown in Fig. 4.17(right). From this, we can see that our geometry roughly matches with the ground truth, and is especially consistent around the wrinkle strokes.

4.6 Summary

This chapter presents a novel method to analyze wrinkle strokes in clean line drawings and to produce manga-style shading over the drawing. Our major contributions in this work include: i) the computational models of five perceptual cues (T-junction, shape convexity, proximity, continuity, and regularity) for wrinkle strokes, where we design the models by exploring the local shape, spatial relation, and interaction of strokes by consulting relevant psychological principles and manga drawing style; ii) an optimization formulation for producing globally consistent wrinkles by combining and balancing the local estimation from the cues, and iii) a wrinkle-aware inflation method for generating proxy geometry to support two commonly used shading styles. Compared to the prior work, our method has several unique features such as the ability of automatically generating rich wrinkle geometry conveyed in the interior strokes, the ability of producing globally consistent shape inference, and supporting shading characteristics employed by artists, which have been demonstrated by various experimental examples. These features facilitate the generation of plausible shading on such complex line drawings which is difficult with previous work.

Limitations

Our method shares some common limitations with other works that perform 3D reconstruction from a single 2D input. First, our method cannot reconstruct a plausible surface when the actual (perceived) depth of the strokes in the drawing vary drastically. For example, the z-coordinates of strokes (w.r.t the viewpoint) in the `TABLE` drawing vary significantly, and hence our result will not match closely with the ground truth. Second, our inflation method assumes that the geometry is smooth and cannot generate polyhedral shapes such as sharp edges or tilted planes. Lastly, our method assumes that lighting and geometry are the main influencing factors for shading, but several other artistic criteria are often used, such as the mood of the scene, personal style of the artist, etc. We discuss potential avenues to resolve these limitations in Chapter 7.

CHAPTER 5

2.5D Cartoon Hair Modeling and Manipulation*

With the huge availability of 2D digital cartoon images and video clips available today, there is significant artistic value if we can somehow enable end-users to reuse this content and create intriguing results by animation or manipulation. Such a problem requires at least a 2.5D understanding of the image or video i.e. we need to identify the various foreground elements from the background, infer the depth ordering among the neighboring regions and complete the occluded regions so that holes do not become visible when the layers are manipulated. Particularly, our automatic layering method is motivated by Gestalt psychology, like in Chapter 4, and is based on analysis of T-junctions and convex overlapping regions. To this end, in this chapter, we specifically consider the problem of animating the hair of cartoon characters in still images by 2.5D modeling.

5.1 Introduction

Single-view modeling and animation is a challenging but valuable computer graphics problem. From just a single image, it aims to reconstruct the image contents for producing appealing visual effects with very little user input. Since there is very limited information in a single image, the problem is highly challenging, but at the same time, it is also highly valuable because it requires very little data preparation effort, and yet can deliver visually-pleasing results.

Several pioneering works have been proposed to deal with this research problem on different kinds of input images. For example, Lin et al. [135] generated animated videos from a small collection of high resolution stills by temporal ordering and a second-order Markov chain model. Xu et al. [136] created

*This author is the second contributor, and mainly helped in implementing, and data collection and processing with, the techniques mentioned in Section 5.2 (segmentation), Section 5.3, and Section 5.6 (evaluation). This work [62] was in collaboration with Mr. Chih-Kuo Yeh from National Cheng-Kung University, who is the first author.

Figure 5.1: *Overview of our 2.5D approach. Note that in subfigure (c) above, the orange arrows show the depth ordering, i.e., pointing from occludees (lower layer) to occluders (upper layer), whereas the = signs denote same depth ordering; in subfigure (d) the light blue color represents the completed hair region.*

believable animations of animals by analyzing the motion of animal groups in a single image, while Okabe et al. [137] animated fluids in images by extracting their motion patterns from video examples. More recently, Chai et al. [138] animated hairs in a real photo by estimating hair direction field and reconstructing the hair volume.

Inspired by the above works, this work takes cartoon/manga images as input instead of real world photos, and aims to analyze the image contents for animating and editing the cartoon hair in an input image. This work is driven by the fact that hair animation is very common in cartoon anime. If we are able to produce a 2.5D hair model from a cartoon image, we can effectively produce a wide variety of intriguing visual effects such as hair animation and editing. For example, we can present a cartoon character with hair blown by the wind, and manipulate his/her hair with proper layering.

However, such a modeling problem is technically very challenging: since the given cartoon image is just 2D, while animating/editing the hairs requires at least a 2.5D understanding of the image contents. In particular, we need to layer the hair strands and simulate their movement to create visually-plausible animations, while attempting to retain the artist's expressive style in the original drawing. This is very different from traditional methods, where artists manually draw the keyframes.

In this work, we take a 2.5D approach since 2.5D methods are often employed in cartoon modeling and animation. For example, Rivers et al. [86] and Yeh et al. [88] showed the simplicity and effectiveness of taking 2.5D approaches to rapidly model and manipulate cartoon characters. Moreover, the general public is used to the 2D visual style of cartoons, and 2.5D approaches can help retain the artist's drawing style. Furthermore, 2.5D hair models are sufficient

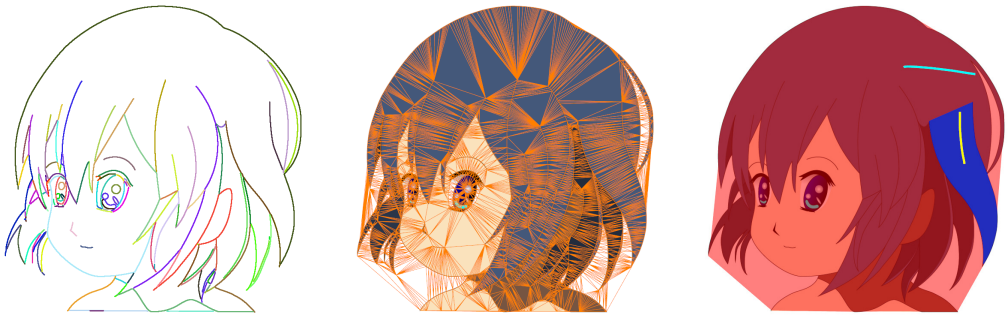


Figure 5.2: *Left: edge extraction; middle: constrained Delaunay triangulation; right: markups for segmenting hair strands.*

for our problem since we take 2D cartoon images as inputs, and by reconstructing a 2.5D hair model, various cartoon hair animation and manipulation effects can be achieved without tediously reconstructing a 3D hair model.

In summary, we propose a novel three-fold solution to handle the problem: (i&ii) *new layering* and *completion methods* to construct a 2.5D hair model from a still cartoon image; and (iii) a *deformation strategy* to animate 2.5D hairs based on fluid simulations. Our layering method relies on a simple and effective metric derived from the Gestalt psychology to automatically identify and optimize the layering among the hair strands. Moreover, hair layers from 2D segmentations could be incomplete due to occlusion; our layer completion method can automatically resolve this issue, and avoids holes in the hair animation. Then, we generate skeletons of hair strands, and devise a simplified 2D fluid simulation model to produce wind-blown hair animations. To evaluate the quality of our results, we conduct a comparative study to examine hair animations produced by our methods and conventional cartoon anime. Lastly, to show the applicability of our method, we demonstrate a wide variety of hair animation and manipulation effects with our 2.5D hair models.

5.2 Overview

Fig. 5.1 overviews the major steps in our 2.5D modeling approach with a running example.

Segmentation. For the segmentation step, we first construct a 2D mesh as a coarser representation of the input image because it is computationally more

efficient. To do so, we detect contours by the curve extraction method in [25] (see Fig. 5.2 (left)) as it produces fewer and nicer connected contours as compared to conventional edge detection methods. First, we find all endpoints and junctions from these curves and keep them fixed. Then, we smooth the curves by Gaussian filtering, and iteratively simplify them using the Douglas-Peucker algorithm [139]. Finally, constrained Delaunay triangulation [129] is used to partition the image (see Fig. 5.2 (middle)).

Next, we build a graph $G = (V, E)$, where the nodes in V and edges E correspond to faces and edges of the mesh, respectively. We adopt a mesh-based graph-cut segmentation method [140] to segment out each hair strand with markups (see Fig. 5.2 (right)). The weights of the edges $e \in E$ are set as $w(e_{ij}) = \varphi_{ij}\|e_{ij}\|$, where $(i, j) \in V$; $\|\cdot\|$ is the norm operator for computing the edge length; φ_{ij} is used to penalize the cost of cutting through edges corresponding to homogeneous regions in the image, and is empirically set to be 0 (lowest possible non-negative weight) for extracted curves, and 30 otherwise. This encourages the segmentation boundaries to pass through the contours of hair strands in the input image, while avoiding homogeneous image regions. Alternatively, φ_{ij} can also be set using the Laplacian or gradient of the input image. Moreover, we employ terminal weights in the graph-cut process, and model the foreground and background by a Gaussian mixture model; our background model B is $P(C, D|B) = 1 - P(C, D|F)$, where $P(C, D|F)$ is the joint probability that measures color similarity C and point distance D to the foreground stroke F. By this, we can reduce the amount of background strokes needed for the segmentation. Note that our method can work with gradients/shading if the hair strands have clear boundaries, but still, other curve extraction or segmentation methods, e.g., [93], may also be used.

Hair Modeling. After segmenting out each hair strand (see Fig. 5.1(a&b)), we automatically estimate the layering of hair strands from the segmented regions (see Fig. 5.1(c)) based on an analytical method we derived from psychological and experimental observations. Then, we devise a new geometric completion method based on the active contour model to fill the occluded parts in each hair strand (see Fig. 5.1(d)). Lastly, we generate the skeletons (see Fig. 5.1(e)) to enable hair animations.

Hair Animation and Manipulation. We may use keyframe-based methods to animate the hair skeletons, but they lack visual realism and are tedious.

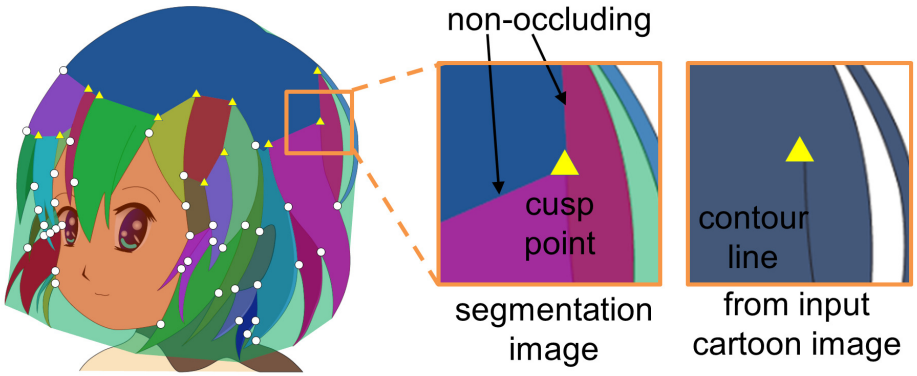


Figure 5.3: *Junctions (both yellow triangles and white circles) and cusp points (only yellow triangles).*

Hence, we devise a simplified 2D elastic model and perform a fluid simulation to animate the hair mesh, (see Fig. 5.1(f)). Other than animation, we can also perform various hair manipulation effects, e.g., hairstyle editing (see Section 5.6).

5.3 Cartoon Hair Layering

To ensure proper occlusion among hair strands when we animate and manipulate them, we next determine their depth ordering (henceforth called as layering). Previous methods mainly focus on general layering problems, requiring either tedious manual specification [51] or extensive datasets for machine learning [72]. A recent method by Palou and Salembier [60] also employed T-junctions to resolve the layering, but their results show only a few number of segmented regions, and the method deals with basic T-junction cases only. In contrast, our layering method is fully automatic; it does not require extensive datasets for learning, and can handle rather complex hair layering cases with arbitrarily shaped junctions.

5.3.1 Junctions and Cusp Points

Junctions are corner points shared among three or more image regions. See the junctions shown in Fig. 5.3. They are effective visual cues [58] for determining the layering. However, not all junctions are meaningful for layering. If a

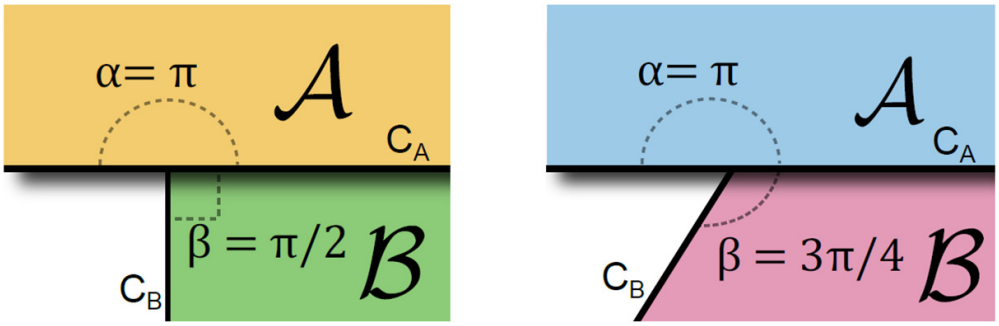


Figure 5.4: *Estimating the layering from junction angles: Left: an ideal case, and Right: a general case.*

junction is located on a contour edge without depth occlusion, see the zoom-in images in Fig. 5.3, we call it a *cusp point*, and will not use it when determining depth ordering between related image regions. Hence, before we compute the layering later in our pipeline, we first have to identify cusp points among the junctions.

The procedure to identify cusp points among junctions is as follow: If a boundary line around a hair strand is found to be on a contour line in the input cartoon image (by summing up the local Laplacian along it), we regard it as an *occluding boundary*, see again Fig. 5.3. Otherwise, it is *non-occluding*, and we label its end-point junctions as cusp points. Note that we use the equal sign ($=$), arrow ($\rightarrow$), and bidirectional arrow ($\leftrightarrow$) to denote three possible layering cases between neighboring regions, see again Fig. 5.1(c), say $\mathcal{A}$ and $\mathcal{B}$: $\mathcal{A} = \mathcal{B}$ means the same depth; $\mathcal{A} \rightarrow \mathcal{B}$ means $\mathcal{B}$ above $\mathcal{A}$; and $\mathcal{A} \leftrightarrow \mathcal{B}$ means an ambiguous order.

5.3.2 The Junction Metric Φ_J

To employ junctions for layering, we first look at the *amodal completion law* in the Gestalt psychology [58]:

> "When a curve, say C_B, stops on another curve, say C_A, thereby producing a T-junction, our perception tends to interpret C_B as the boundary of an object being occluded by the object associated with C_A."

Fig. 5.4 illustrates this law. As a notation, we name the regions associated with C_A and C_B as $\mathcal{A}$ and $\mathcal{B}$, respectively, and denote the angles subtended by $\mathcal{A}$

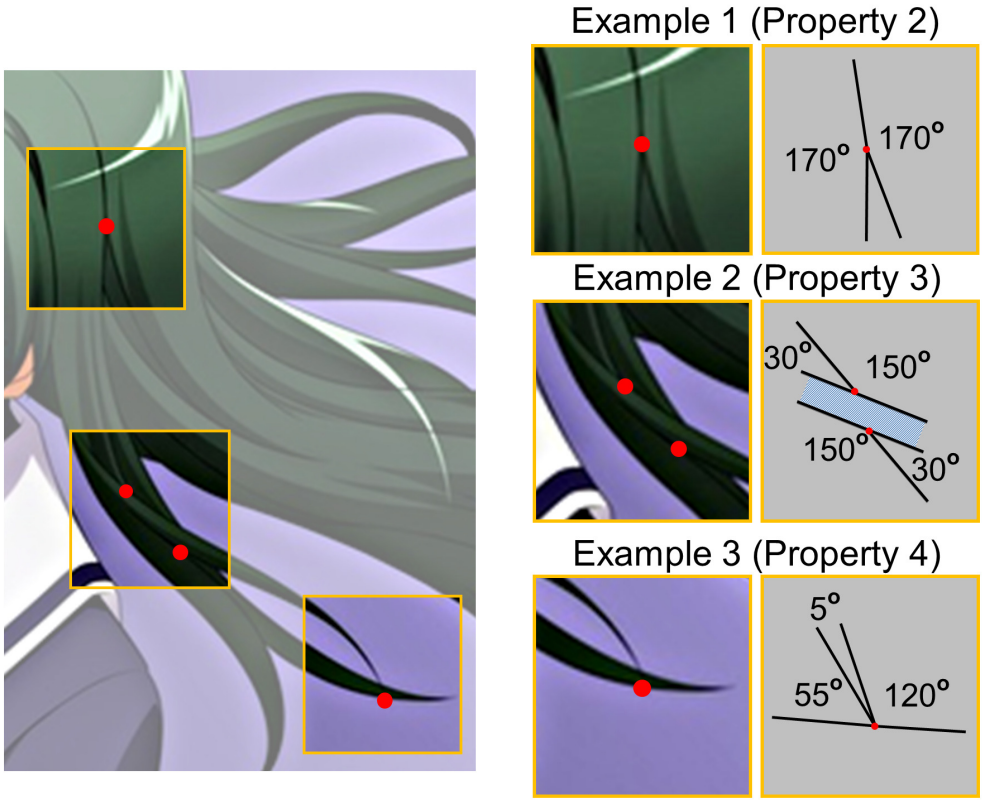


Figure 5.5: *Example Cartoon Hairs for Properties 2, 3 and 4.*

and $\mathcal{B}$ at the junction as α and β, respectively. In the ideal case, $\alpha = \pi$ and $\beta = \pi/2$; and we tend to interpret $\mathcal{B}$ as the *occludee* and $\mathcal{A}$ the *occluder*.

In the general case, C_A and C_B are arbitrary curves, see Fig. 5.4 (right), so they may not intersect at a right angle to give a layering perception [58], see Property 1:

> ***Property 1***: *If α is close to π and β is close to $\pi/2$, we regard $\mathcal{A}$ as on top of $\mathcal{B}$, i.e., $\mathcal{B} \rightarrow \mathcal{A}$.*

Moreover, based on observation with cartoon hairs, see Fig. 5.5, we develop three additional properties for layering. To begin, we first define the concept of *angles domain* for junction angles α and β. Since they are located at a common 2D junction, their sum should not exceed 2π. Moreover, without loss of generality, we assume α to be a larger angle, i.e., $\alpha \geq \beta$. Hence, the domain of α and β is basically the lower quarter of $[0, 2\pi] \times [0, 2\pi]$, which is the green area in Fig. 5.6(a).

> ***Property 2***: *If $\alpha \approx \beta$, the junction metric should not suggest any order-*

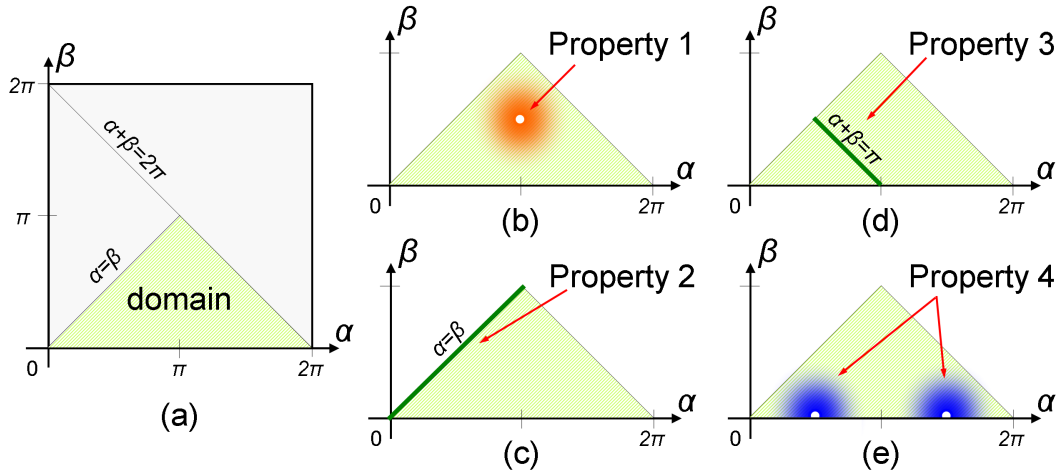


Figure 5.6: *(a) Domain of junction angles α and β; (b-e) Properties that define the junction metric for layering.*

ing preference, i.e., $\mathcal{A} \leftrightarrow \mathcal{B}$.

This is based on the fact that when α and β have similar sizes, see Fig. 5.5 (example 1), we should not arbitrarily decide a layering between their related image regions solely by α and β, see Fig. 5.6(c).

> ***Property 3**: If α+$\beta \approx \pi$, the junction metric should not suggest any ordering preference, i.e., $\mathcal{A} \leftrightarrow \mathcal{B}$.*

This relates to a cartoon-hair situation, where multiple hair strands go below another hair strand altogether, see Fig. 5.5 (example 2). In this case, α+$\beta \approx \pi$, and we again should not arbitrarily decide a layering between the two related regions solely by α and β, see Fig. 5.6(d).

> ***Property 4**: If β is close to 0 while α is not close to 0 or π, we regard $\mathcal{B}$ as on top of $\mathcal{A}$, i.e., $\mathcal{A} \rightarrow \mathcal{B}$.*

This property is related to another common situation with cartoon hairs (particularly due to image rasterization), where a hair tip lands on the edge of another hair strand, see Fig. 5.5 (example 3). Concerning this, when we compare α and β, we regard the region with a tiny sharp angle to be on the top because hair regions with sharp angles are very likely to be hair tips, i.e., β, which is the smaller angle, is close to zero. However, since we have to avoid the situations related to Properties 2 and 3, α should not be close to zero or π at the same time, see the blue areas in Fig. 5.6(e).

To capture the above layering properties, we design and formulate the follow-

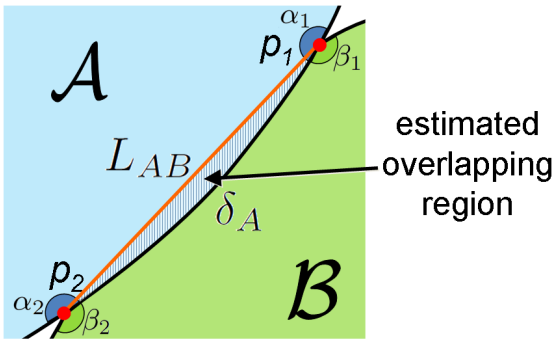


Figure 5.7: *Region overlap metric. By using Green's theorem, we estimate the signed area of the overlapping region.*

ing junction metric Φ_J:

$$\Phi_J(\alpha, \beta) = \big|(\alpha \bmod \pi) - \pi/2\big| - \big|(\beta \bmod \pi) - \pi/2\big|,$$

where mod is the modulus operator; the sign of Φ_J indicates the layering order between $\mathcal{A}$ and $\mathcal{B}$ while its magnitude indicates the tendency: if $\Phi_J \approx 0$, $\mathcal{A} \leftrightarrow \mathcal{B}$; if $\Phi_J > 0$, $\mathcal{B} \rightarrow \mathcal{A}$, and vice versa. Note that when $\alpha \approx \pi$ and $\beta \approx \pi/2$, Φ_J is relatively large and positive, indicating $\mathcal{A}$ as on top, and when $\alpha \approx \pi/2$ and $\beta \approx \pi$, Φ_J is relatively large and negative, indicating the opposite situation. Moreover, Φ_J also fulfills properties 3 and 4, which are related to cartoon hairs. Note also that to avoid rasterization noise when computing junction angles, we fit a short boundary curve from each junction point around the segmented region.

5.3.3 The Region Overlap Metric Φ_R

Another local feature we employed for computing the layering is *region overlap*, which is also motivated by the amodal completion law. Given regions $\mathcal{A}$ and $\mathcal{B}$ with junction points p_1 and p_2 (see Figure 5.7), we first construct a straight line from p_1 to p_2, say $L_{\mathcal{AB}}$, and then, define the region overlap metric as

$$\Phi_R(\mathcal{A}, \mathcal{B}) \;=\; \delta_{\mathcal{A}} - \delta_{\mathcal{B}}\;,$$

where $\delta_{\mathcal{A}}$ and $\delta_{\mathcal{B}}$ are the area of $\mathcal{A}$'s and $\mathcal{B}$'s portion on the right and left of $L_{\mathcal{AB}}$, respectively. For the example shown in the inset figure above, $\delta_{\mathcal{A}}$ is the shaded portion of $\mathcal{A}$ to the right of $L_{\mathcal{AB}}$ while $\delta_{\mathcal{B}}$ is zero. Hence, Φ_R is

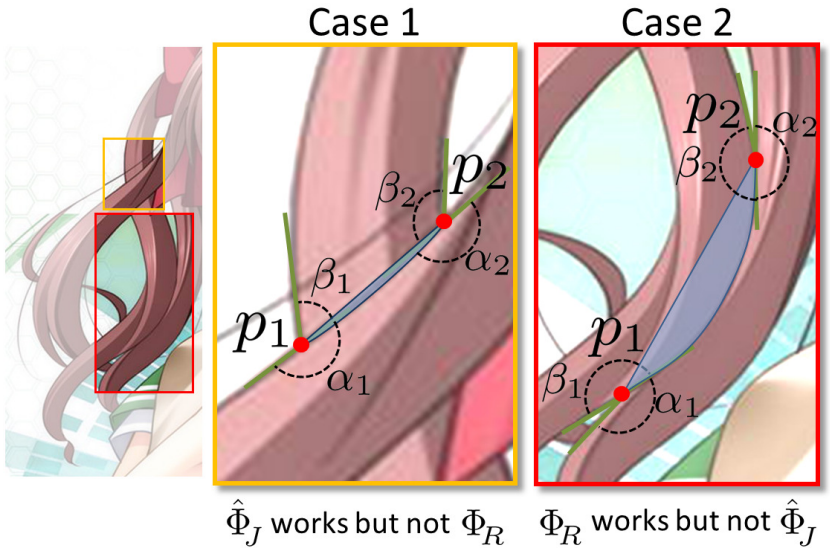


Figure 5.8: *Case 1: Φ_R does not work because the estimated overlap region is too small. Case 2: Φ_J does not work due to layering Property 2. By combining Φ_J and Φ_R, they complement each other, and help solve the layering for both cases.*

positive, suggesting $\mathcal{A}$ as on top of $\mathcal{B}$. In detail, we implement the computation by using half edge data structure to represent the segmented hair regions and arranging the edges in anticlockwise order. Hence, we can use the Green's theorem to compute the signed area of $\delta_{\mathcal{A}}$ or $\delta_{\mathcal{B}}$, which is Φ_R.

5.3.4 The Layering Metric Φ

Given neighboring regions $\mathcal{A}$ and $\mathcal{B}$, our layering metric Φ combines junction metric Φ_J and region overlap metric Φ_R to determine the layering order between them:

$$\Phi(\mathcal{A},\mathcal{B}) \;=\; w\cdot\hat{\Phi}_J(\mathcal{A},\mathcal{B}) \;+\; \Phi_R(\mathcal{A},\mathcal{B})/|L_{AB}|^2\,, \tag{5.1}$$

where $\hat{\Phi}_J(\mathcal{A},\mathcal{B}) = s(p_1)\Phi_J(\alpha_1,\beta_1) + s(p_2)\Phi_J(\alpha_2,\beta_2)$ denotes the sum of Φ_Js at the two junction points between $\mathcal{A}$ and $\mathcal{B}$, and $s(p)$ is 0 if p is a cusp point, otherwise 1. This $s(p)$ helps to avoid Φ_J at cusp points. To normalize the area in Φ_R, we divide Φ_R by $|L_{AB}|^2$, and use a weight term w (set to be 0.25) to balance the two metrics. Again, a positive Φ indicates $\mathcal{A}$ on top of $\mathcal{B}$ similar to Φ_J and Φ_R, and vice versa. Fig. 5.8 shows two examples, where we can successfully determine the local layering order by combining the strengths of Φ_J and Φ_R.

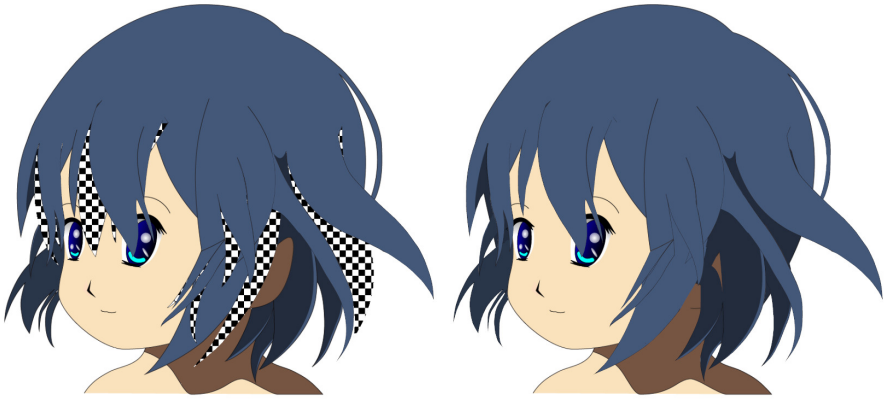


Figure 5.9: *2.5D hair completion for animation and manipulation. Left: without completion; right: with completion.*

To determine layering among all hair regions in a given image segmentation, we construct a graph data structure with nodes to denote the segmented image regions and edges to denote the connections between neighboring regions, with edge weight equal to the value of the layering metric Φ. We then apply topological sorting to further compute the rendering order among the regions, by iteratively removing nodes with zero in-degrees from the graph. Note also that deadlock may sometimes happen during the topological sort, i.e., no graph nodes with zero in-degrees. In this case, we search for the related deadlock cycle in the directed graph, and break the cycle by removing the weakest directed edge based on Φs magnitude.

5.4 Cartoon Hair Completion

To avoid holes in hair animation and manipulation, we next have to complete the hair strands, see Fig. 5.9. Single-view image completion has always been a challenging problem because the hidden parts are unknown and could have arbitrary shapes. Since we focus on hair animation and manipulation, we do not need a general completion method to fully reconstruct the hair strands, i.e., until they reach the cartoon character's head. Rather, we develop an efficient and practical solution for completing 2.5D cartoon hair, capable of delivering various hair animation and manipulation effects.

Our method considers three cases of cartoon hair completion, see Fig. 5.10(a-c): The first case happens when two hair strands intersect each other, so the

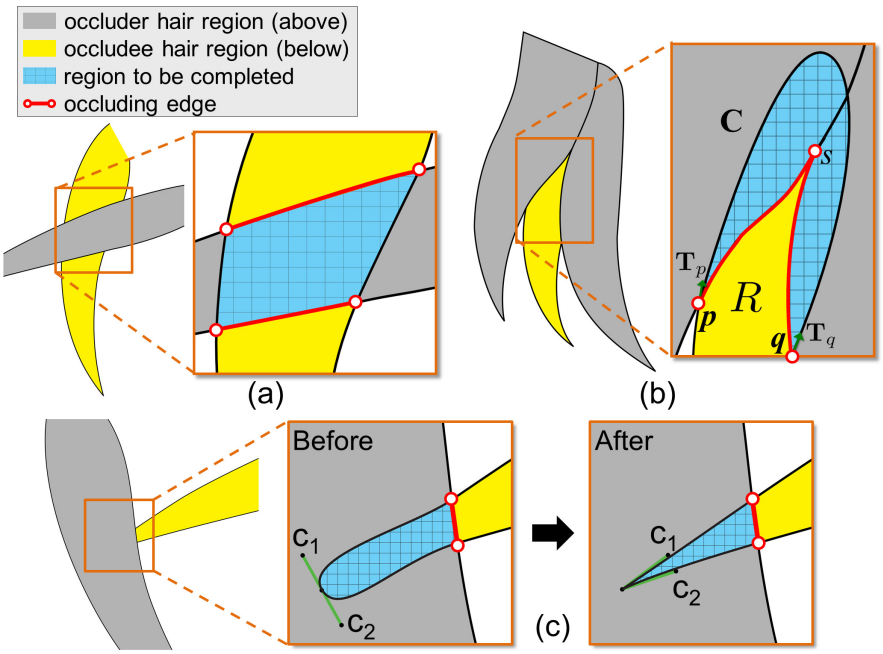


Figure 5.10: *Hair completion cases: i) intersecting hairs, ii) occluded root, and iii) occluded tip before and after refinement.*

one behind is divided into two disconnected regions. The second case happens when the root part of a hair strand (closer to the head) is occluded by other hair strands, whereas the third case happens when the tip part of a hair strand (far away from the head) is occluded by others.

Case 1: Intersecting Hairs

To identify intersecting hairs, our method starts by matching pairs of occluding edges from occludee hair regions around a common occluder, see again Fig. 5.10(a). We consider three matching criteria: 1) color difference between of the two occludee regions around their corresponding occluding edges; 2) sum of distances between the corresponding endpoints of the two occluding edges; and 3) tangent vectors at the occluding edge endpoints.

If a match is found within a threshold, we complete the occluded area by constructing two Bézier curves with C_1 continuity to connect the corresponding endpoints of the occluding edges. Then, we merge the two divided hair regions, together with the completed area, into a single hair strand in our 2.5D model.

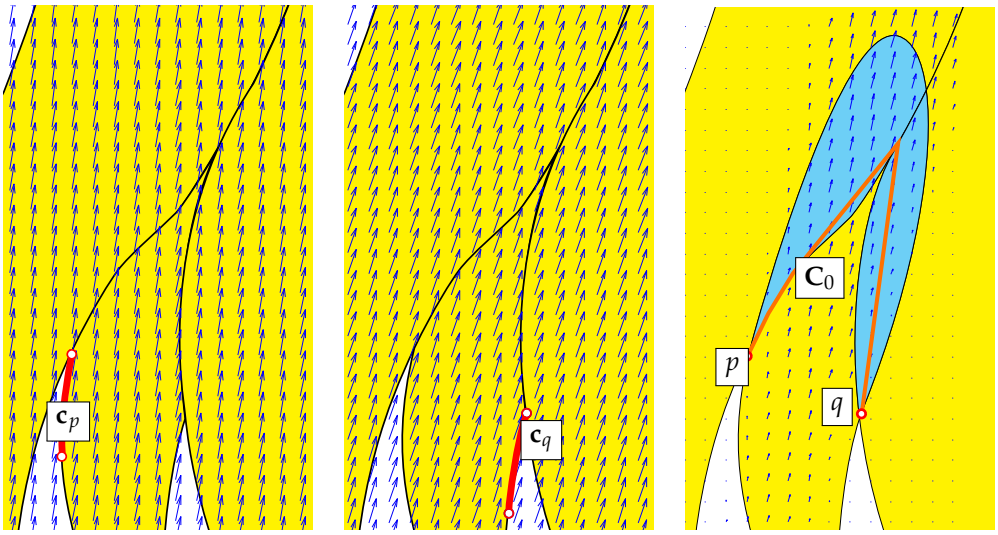


Figure 5.11: *Left & Middle: Vector fields (F_p & F_q) constructed from red short curve segments at p and q, respectively; Right: Combined vector field F_{ext} for pushing $\mathbf{C}_0$ (orange).*

Case 2: Occluded Root

The second and third cases account for occluding edges that are not paired with others. Since we do not have a clear shape in the completion, the second and third cases are more complicated than the first one. Here we first try to merge (or connect) neighboring occluding edges, e.g., *ps* and *sq* in Fig. 5.10(b), and then grow the related occludee hair strands to form the completion area.

Notations. We denote R as the occludee region to be completed, $O(R)$ as the set of occluder regions above R, p and q as the two outermost endpoints on R's (merged) occluding edge, and $\mathbf{T}_p$ and $\mathbf{T}_q$ as the related tangents at p and q, respectively. See Fig. 5.10(b).

Our Method. We devise an active-contour method based on a novel Hamiltonian function, and adopt it in a curve deformation model [141]. In short, our method iteratively refines a curve, say $\mathbf{C}_i(t)$, to form the completion area with the following constraints: $\mathbf{C}_i(0)=p$; $\mathbf{C}_i(1)=q$; $\mathbf{C}_i'(0)=\mathbf{T}_p$; $\mathbf{C}_i'(1)=-\mathbf{T}_q$; and $\mathbf{C}_i$ should be smooth and lie under $O(R)$.

To adopt the deformable model [141] for our problem, we first construct an external force field F_{ext} for pushing $\mathbf{C}_i$. To do so, we extract two short curve segments at p and q, say $\mathbf{c}_p$ and $\mathbf{c}_q$, see Fig. 5.11 (left & middle), and solve the following Hamiltonian function for A_p and B_p given $\mathbf{c}_p$: $H_p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A_p \mathbf{x} +$

$\mathbf{x}^T B_p$, which should satisfy

$$\dot{\mathbf{c}_p}(t) = \left(\frac{\partial H_p(\mathbf{c}_p(t))}{\partial y}, -\frac{\partial H_p(\mathbf{c}_p(t))}{\partial x}\right) \text{ and } \frac{\partial H_p(\mathbf{c}_p(t))}{\partial t} = 0 \ ,$$

where $\mathbf{x}$ is the 2D image space; A_p and B_p are unknowns solved by the above constraints. Then, we compute

$$F_p(\mathbf{x}) = \mathbf{R}(-\pi/2)\nabla H_p = \mathbf{R}(-\pi/2)(A_p\mathbf{x} + B_p) \ ,$$

which is the vector field derived from $\mathbf{c}_p$, see Fig. 5.11 (left). Note that we rotate ∇H_p by $\pi/2$ clockwise because ∇H_p faces outward relative to R. Similarly, we solve H_q for A_q and B_q given $\mathbf{c}_q$, and compute $F_q(\mathbf{x})$. Finally, we combine F_p and F_q to form

$$F_{ext}(\mathbf{x}) = \Omega(\mathbf{x})(tF_p(\mathbf{x}) + (1-t)F_q(\mathbf{x})) \ ,$$

where $\Omega(\mathbf{x}) = Sigmoid(\mathrm{Y}_p(\mathbf{x}) \cdot \mathrm{Y}_q(\mathbf{x}))$; $\mathrm{Y}_p(\mathbf{x}) = max(H_p(p) - H_p(\mathbf{x}), 0)$, $\mathrm{Y}_q(\mathbf{x}) = max(H_q(\mathbf{x}) - H_q(q), 0)$ are used to clamp negative values to avoid unnatural hair growth; and $t \in [0, 1]$ is an interpolation factor based on distances from p and q.

After obtaining F_{ext}, see Fig. 5.11 (right), we compute the convex hull of the occluding boundary of R from p to q, say $\mathbf{C}_0$, see the orange curve in Fig. 5.11 (right). Then, we employ Algorithm 1 with $\mathbf{C}_0$ (as an initial curve) and F_{ext} as

Algorithm 1 ITERATIVE_REFINE ($\mathbf{C}_0, F_{ext}$)

```
i ⇐ 0
while true do
    i ⇐ i + 1
    for each sample point C_{i-1}(t) do
        V_i(t) ⇐ ||F_ext(C_{i-1}(t)) · N(C_{i-1}(t))||N(C_{i-1}(t))
        C_i(t) ⇐ (I − A)^{-1}[C_{i-1}(t) + V_i(t)]
        if C_i(t) outside O(R) then
            return C_{i-1}
        end if
    end for
    if ||V_i|| < ε then
        return C_i
    end if
end while
```

inputs to iteratively refine $\mathbf{C}_0$ to form the completion area. Note that N is the normal of the curve; V_i is the velocity to push the curve outward; and $\mathbb{A}$ is a pentadiagonal matrix, see [141] for its detail.

Case 3: Occluded Tip

The third case is similar to the second case, except that we need to achieve a sharp instead of round tip. After we form a round tip by the method in case 2, see Fig. 5.10(c) (middle), we determine the farthest point on the extrapolated contour, approximate it with a cubic Bézier curve, and then adjust the tangents at the farthest point to produce a sharp tip, see Fig. 5.10(c) (right).

Overall Procedure

To differentiate among the above three cases for each occluding edge, we start with the matching process described in case 1. If an occluding edge can be paired with others, we perform the completion procedure described in case 1. For each unpaired occluding edge, we differentiate between cases 2 and 3 by computing the average distance of the edge to the center of the head. If the edge is closer to the head as compared to the non-occluded region, we go for case 2, else case 3.

After obtaining the shape of the completion area, we employ an existing inpainting method [98] to synthesize the texture and color on the occluded region using the textures sampled from the remainder of the hair image. Some artifacts may appear if the hair occludes nontrivial image features such as eyes and mouth (see Fig 5.13), where manual inpainting is required to complete the features. This could be potentially automated atleast partially, by exploiting symmetry of the character's face for simple cases, or by developing a data-driven learning approach for face detection similar to those used for human faces.

Figure 5.12: *Hair animation results: snapshots of 2.5D cartoon hair animations produced from our method.*

Figure 5.13: *Nontrivial image features (e.g., eyes) are difficult to inpaint automatically, and require manual intervention.*

5.5 Cartoon Hair Animation and Manipulation

To support hair animation and manipulation, we construct a skeleton for each completed hair strand as follows: First, we determine the tip of each hair strand by identifying the sharp corner(s) in each completed hair strand. Then, we look for corners that are locally above the neighboring regions by using the layering results we obtained earlier. Next, we construct a medial axis from the hair tip along the hair strand as the hair skeleton. Sometimes, a hair strand may have branches, so when we grow multiple medial axes from different tips, we join the medial axes in a hierarchical skeleton.

5.5.1 Cartoon Hair Animation

In general, hair simulations require modeling both the blowing wind and hair strands in 3D, and coupling 3D fluid dynamics with material deformation. This, however, could be too complicated for conventional cartoons. Hence, we propose a simplified simulation, which is efficient for generating wind-blown hairs.

Our method is based on the following assumptions. First, the movement of hair strands is approximated by the corresponding skeletons. Second, the hair dynamics is approximated by a linear elastic model with piecewise rigid deformation. Third, we ignore hair-hair interactions, and only employ 2D fluid simulations.

In detail, we move the hair skeleton by the aerodynamic forces from the wind, and employ the lattice Boltzmann model to solve the incompressible Navier-

Stokes equation [142] with the hair skeleton as the boundary. After the simulation, we obtain pressure p, which is normal to the skeleton curve. Moreover, to prevent the skeleton from excessive bending, we model bending force $\mathbf{f} = \gamma\kappa\mathbf{n}$, where γ is the stiffness; κ is local curvature; and $\mathbf{n}$ is the normal vector along the skeleton curve. Together with the gravity force $\mathbf{g}$, the total force acting on the skeleton is: $\mathbf{F} = -p\mathbf{n} + \mathbf{f} + \mathbf{g}$.

Next, we uniformly sample the skeleton curve, and compute $\mathbf{F}$ at each sample node. Then, we perform time integration to deform and animate the skeleton according to the following dynamic equation: $\ddot{\theta} = \tau\mathbf{F} \cdot \mathbf{n}$, where θ is the change in angle between successive edges at each sample node; τ is the rate of rotation, which allows for small or large amount of hair movement. Note that we deform the skeleton by considering only rotation at sample nodes, and compute the rotations to bend the hair strand starting from its root till the tip. To improve the visual realism, we also vary the rotation rate along the hair by $\tau = \tau_0 d^{\lambda}$, where τ_0 is the base rotation rate and d is the distance from the hair root along the skeleton; λ is the user-controllable parameter for tuning the amount of hair bending.

Once we obtain the skeletons and their motion trajectories, we deform the mesh to follow the skeleton motion by binding each mesh vertex to one or more bones of the skeleton with appropriate blending weights per vertex. In our model, we set the weights as the inverse distance from the skeleton curve. Other standard methods, such as [143], could also be used here.

Fig. 5.12 presents our hair animation results, while the video can be accessed at `https://youtu.be/IVyYACqgh3k`. A single wind source is used to simulate the wind blowing on the cartoon hairs. From these results, we can see that our method can properly animate the hair strands with appropriate 2.5D layering while retaining the original drawing style of the artists in the given cartoon images.

5.5.2 Cartoon Hair Manipulation

Hair Editing. Our 2.5D hair model enables flexible editing of hair strands in cartoon images, e.g., layering order, hair length, braiding, see Fig. 5.14. We achieve these effects by manipulating the hair skeleton and adopting the as-rigid-as-possible method [144] for the deformation. In detail, we formulate

Figure 5.14: *Hair manipulation results: our tool can scale the hair strands to change the hairstyle (top row); and perform hair braiding by twisting the hair strands (bottom row) from single input images (leftmost).*

it as a minimization problem with a deformation term, a smoothness term, and a constraint term, and the objective is defined as $\Omega = w_R\Omega_R + w_H\Omega_H + w_C\Omega_C$, where w_R, w_H and w_C are the weighting coefficients, which are set to $20, 1, 1000$, respectively in all our experiments. Ω_R is a deformation term defined as $\Omega_R = \sum_{i\in\mathbf{V},k\in\mathbf{S}} w_i^k \|(\mathbf{v}_i - \mathbf{s}_k) - \mathbf{T}_k(\mathbf{v}_i' - \mathbf{s}_k')\|^2$, where w_i^k is the weight of vertex v_i for skeleton $s_k \in \mathbf{S}$ ($\mathbf{S}$ is a set of bones) and is set to be the inverse of the distance from the skeleton; and symbol $'$ indicates variables of the deformed skeleton. $\mathbf{T}_k$ is the transformation matrix that constraints the resizing of hair along the tangential direction of the skeleton to maintain the overall shape; it is defined as a rigid transformation:

$$\mathbf{T}_k = \mathbf{R}_{(1,0)}^{e_{k'}} \begin{bmatrix} s & 0 \\ 0 & 1 \end{bmatrix} \mathbf{R}_{e_k}^{(1,0)},$$

where $e_k = s_{k2} - s_{k1}$ and e_k' denote the deformed edge vector, $\mathbf{R}_v^{v'}$ is the rotation matrix that transforms vertex v to v'; and $s = \|e_{k'}\|/\|e_k\|$ is a scale factor. This transformation allows (and prevents) scaling along tangential (and normal) direction of the bones. Ω_H is a smoothness term defined as $\Omega_H = \sum_{(i,j)\in\mathbf{E}} w_{ij}\|\mathbf{v}_j' - \mathbf{v}_i'\|^2$, where $w_{ij} = cot\theta_{ij} + cot\theta_{ij}'$ are discrete harmonic weights, θ_{ij} and θ_{ij}' are angles opposite to the edge (i,j) in the original mesh. Ω_C is a constraint term defined as $\Omega_C = \sum_{i\in\mathbf{C}} \|\mathbf{v}_i - \mathbf{v}_i'\|^2$, where $\mathbf{C}$ is a set of constraint vertices. We set boundary constraints at the root of the hair strand.

Hair Braiding. We adopt the twist operation in [88] for producing the hair braiding effect, see Fig. 5.14. Here we darken the texture of the hair strand as its back texture, and twist the hair strand geometry while mixing its front

and back textures. Since we can only present static images in the text, readers can refer to the supplementary video (`https://youtu.be/IVyYACqgh3k`) for the related animation results.

5.6 Discussion

Implementation and Performance. The proposed system is implemented in C++ and evaluated on a desktop computer with a 3.4 GHz CPU and 4GB memory. The time taken to manually segment the hair in the four cartoon images shown in Fig. 5.12 (from top to bottom) are 8, 12, 9, and 15 minutes, respectively. These cartoon characters have face features, but we avoid them in the figure for clarity in the results presentation. The average processing time (over four segmented cartoon images) for layering computation, hair completion (excluding color and texture inpainting), skeleton generation, and deformation are around 0.031, 1.608, 0.415, and 0.015 seconds, respectively. After the 2.5D modeling, the hair animation process, which involves a fluid simulation, takes around 1.5 seconds to compute each frame.

Evaluating the Layering Metric. We conducted an experiment to evaluate the layering metric presented in Section 5.3 with 7 subjects: ages from 23 to 27 (mean 24.43); 4 males and 3 females; and volunteer-based. Fig. 5.12 shows the four cartoon images we employed with a total of 475 layering cases, each corresponding to a pair of neighboring image regions around a cartoon hair.

In detail, we implemented a simple interface in Python to present the cartoon image to the subjects with a zoom-in view on each layering case. The subjects can enter their layering judgment by a mouse click, i.e., which region is on the top, or whether the two regions have equal depth. Then, the subject can hit Spacebar to finalize a decision, and proceed to the next layering case. Our interface records the layering judgment and the time taken per case. The subjects have no time constraint in making a decision, but they mostly took a few seconds per case, except for some more complicated ones, which may require more than 10 seconds.

After collecting the data, we first compute the ground truth (G.T.) of each layering case by voting over the subject's input because different subjects may perceive different layering orders. In detail, we define layering sites along the

Table 5.1: *Comparison: human subject vs our layering metric.*

cartoon images (see Fig. 5.12)	accuracy (vs G.T.)		average time (sec.)	
	subjects	our metric	subjects	our metric
First row	90.1%	91.4%	192.0	0.031
Second row	92.3%	93.5%	382.1	0.042
Third row	95.9%	97.0%	216.7	0.032
Fourth row	90.7%	92.3%	231.2	0.020
Average	92.3%	93.6%	255.5	0.031

shared boundary between every pair of regions, where the user can markup the perceived layering. For example, given a shared boundary between two regions A and B, the user can specify the layering using equality i.e., $A = B$, or inequalities, i.e., $A \leftarrow B$ or $A \rightarrow B$. We compute the maximum vote among all the user annotations for each site, and consider it as ground truth. Then, we compare the subject's inputs against the ground truth to compute the average accuracy of human subjects. Similarly, we compute the average accuracy of our layering metric by comparing our results against the ground truth. From the second and third columns in Table 5.1, which summarize the accuracy results, we can see that our layering metric can achieve similar or even better accuracy compared to human subjects. Moreover, our layering is automatic and simple to compute; it only takes around 0.031 seconds to compute with, see the last column in Table 5.1.

The few failure cases, where our layering result does not match the human judgment result, are mainly due to two reasons: 1) some cases are truly ambiguous even for human judgment, see the two examples shown in Fig. 5.15(a); and 2) our method predicts equal layering depth based on computing the Laplacian; hence, for unclear boundaries between hair strands, Cheng's curve extraction algorithm may miss the related contour line, see Fig. 5.15(b), and so, our layering metric may mis-classify it as non-occluding boundary with equal depth.

Comparative Study: Cartoon Videos. We compared our hair animation results with three real cartoon videos. First, we extract a single image frame from each of these videos, and apply our method to produce a 2.5D cartoon model on each extracted image. Then, we adjust the fluid simulator to create a similar wind flow, and generate hair animations. After that, we can compare our hair animation results with the original cartoon videos. Readers can refer

to the supplementary video (`https://youtu.be/IVyYACqgh3k`) for the related animations. The results show that our hair animations are visually comparable to the original videos even though our method takes only a single image as input.

5.7 Summary

This chapter presented a novel 2.5D approach to modeling, animating, and manipulating hairs in a single cartoon image. In summary, we have three key contributions: First, we derive an effective layering metric from the Gestalt psychology and our observation on cartoon images; it can automatically optimize the layering order among hair strands in a single cartoon image. Second, we develop a novel layer completion method that can automatically fill the occluding parts of hair strands; by this, we can construct a 2.5D hair model to support hair animation and manipulation with appropriate layering. Finally, we devise a simplified simulation model to animate the skeletons in hair strands, and also develop a wide variety of hair manipulation operations, including hair editing and hair braiding. To demonstrate the capability of our approach, we also compare our results with conventional cartoon videos, and apply our method to produce hair animation and manipulation results on assorted cartoon characters.

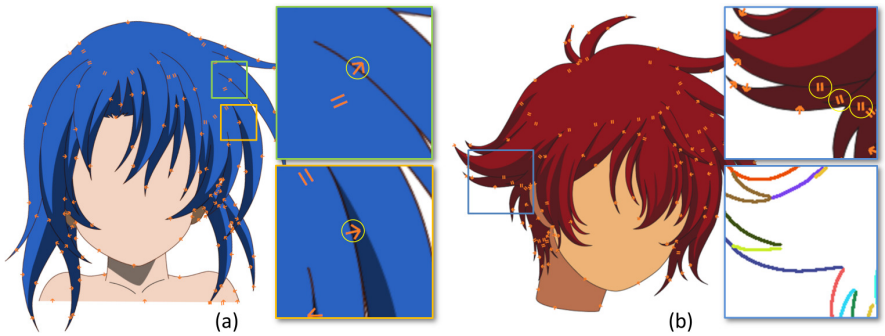


Figure 5.15: *Examples where our layering metric does not match the human judgment results: (a) layering ambiguity and (b) inaccurate edge detection.*

Limitations

There are some limitations in our method, that we plan to address in future work. First, we cannot handle messy hairstyles, because the hairs in this case cannot be properly segmented into strands for 2.5D modeling. Also, thin hair strands like real human hairs may not be suitable for use in our system. Second, our layering method currently cannot handle intertwined hair strands because it does not consider local layering [87], e.g., two graphical elements that overlay each other with different layering in different parts of the image. Third, our hair simulation model assumes no hair-hair interaction, i.e., the hair strands are animated independently. Lastly, we do not consider lighting and shading in our framework, resulting in shadows that are pinned to the same location when animated.

CHAPTER 6

Interactive High-relief Reconstruction of Organic & Double-sided Objects from a Photo*

In this chapter, we present an extension of some methods developed in the thesis for a related problem dealing with reconstructing objects present in natural images. While this method can be easily applied to cartoon images in a straightforward way as demonstrated later, we focus mainly on reconstructing smooth, organic natural objects from single photos such that they can be plausibly rotated away from the viewing angle. Recall that in Chapter 4, we presented an inflation method for lifting 2D line drawings to 3D, and in Chapter 5, we proposed a technique for 2.5D modeling of hair cartoon images with planar layers. In this chapter, a combination of these techniques is employed to interactively inflate the planar layers in the 2.5D model to 3D, so that richer depth variations can be achieved within each layer.

6.1 Introduction

Given the vast availability of 2D images that can be easily obtained from the Internet or photo taking, say, with a cell phone, we aim to develop in this work a novel interactive solution to support the reconstruction of high-relief geometry from a single photo. In particular, we focus on high-relief geometric reconstruction of non-polyhedral shapes (see Fig. 6.1 & 6.2 for various examples). High-relief conveys much more depth information as compared to bas-relief; it requires a significant mass of the figure object to be elevated from

*This author was the third contributor who helped in conceiving the identification technique in Section 6.2, assisted in implementing Section 6.3.2, and Section 6.4, and evaluated the method in Section 6.5. This work [145] was in collaboration with Mr. Chih-Kuo Yeh from National Cheng-Kung University, Taiwan who is the primary author.

the background plane [146], while some parts may be completely detached from the background [147].

This would be useful for supporting applications such as stereoscopy and rotating lenticular posters. Note that rotating lenticular posters (used as advertisements in "The Hobbit" and "The Amazing Spiderman") feature objects that appear to be raised from the background like a relief sculpture and rotate when we look around it, thereby delivering an interesting out-of-the-poster 3D perception.

Compared to conventional methods for single-photo 3D reconstruction, this work has the following distinctive aspects. First, we aim to support high-relief rather than bas-relief 3D reconstruction, so that the geometry can appear to pop out of the image plane when we look around it. Existing methods that hallucinate stereoscopic viewing [148, 149] or reconstruct shallow geometries [78] may not be able to achieve this. Second, we aim to support the reconstruction of common non-rigid objects such as human, animals, flowers and teapots. Here, the shapes of the objects can be organic rather than polyhedral or symmetric, and different sides (front and back) of some curvy parts may be revealed simultaneously in the image. Conventional 3D reconstruction methods are insufficient to handle these scenarios.

Sketch-based modeling [117, 150] is another research topic relevant to this work. Their methods provide fine-grained controls for users to author a full 3D model. However, they often require the users to specify tedious amount of constraints for creating more complex shapes. In this work, we also take an interactive user-driven approach, but we aim for reconstructing geometry from photos with a few markup cues from the users, where the user provided information is sufficient for supporting applications such as stereoscopy and

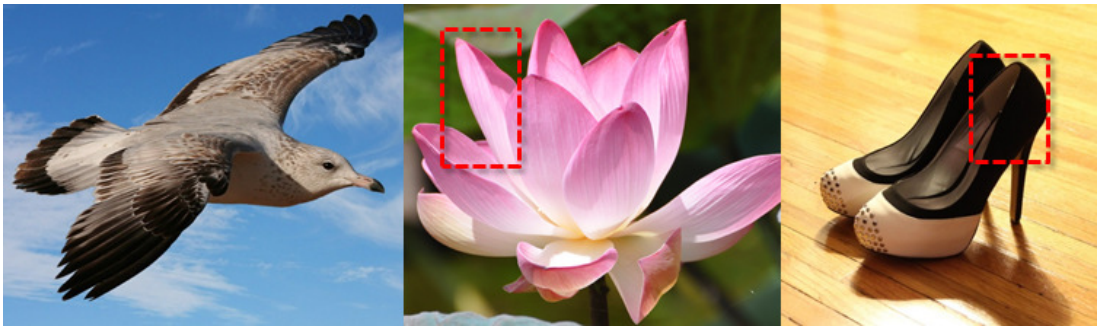


Figure 6.1: *Examples of organic objects and double-sided structures (flower petal and shoe opening) that can be handled by our method.*

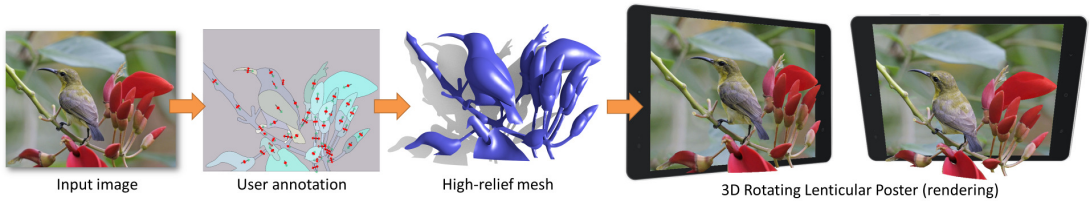


Figure 6.2: *From a single input image, our method constructs a high-relief 3D model by 2.5D layering, layer completion, inflation, and stitching with only simple user-provided cues. Our result can be rotated to produce plausible renderings from large viewing angles.*

rotating lenticular posters.

To achieve the research goal is highly challenging since we only have a single view of the objects in the input photo. More specifically, the object to be reconstructed could be organic in shape (see Fig. 6.1(left)). Moreover, it may involve multiple depth layers (hereafter referred to as regions) (see Fig. 6.1(middle)); each region could have varying geometric characteristics: convex, concave, or a mixture of both, and some regions could be partially occluded by others. Previous methods [148, 149] consider individual objects in the image as separated regions, and do not explicitly model their surface characteristics. Furthermore, we may encounter objects with front and back sides in the given photo view, e.g., flower petals, shoes, etc. (see Fig. 6.1(middle & right)), resulting in view-dependent self-occlusions; no previous work has considered this double-sided reconstruction scenario.

We address the above issues by a user-driven approach. Here, we start with a user-driven segmentation of the input image. Then, we automatically infer local layering information and construct a 2.5D model by analyzing T-junctions and overlap among the adjacent regions. After that, we complete the missing portions of each region with the help of the local layering information, where we consider three types of completion cases corresponding to common occlusion situations in natural images. Next, users can interactively mark up slope and curvature cues in our interface to guide our constrained optimization model to inflate the surface geometry; real-time preview of the inflation results is provided. Compared to previous inflation methods, which mainly produce simple convex surfaces, our method can handle much more complex shapes. Lastly, we arrange the regions in 3D and stitch them together to avoid visible gaps and discontinuities; hence, we can produce plausible high-relief results and allow much larger viewing angles.

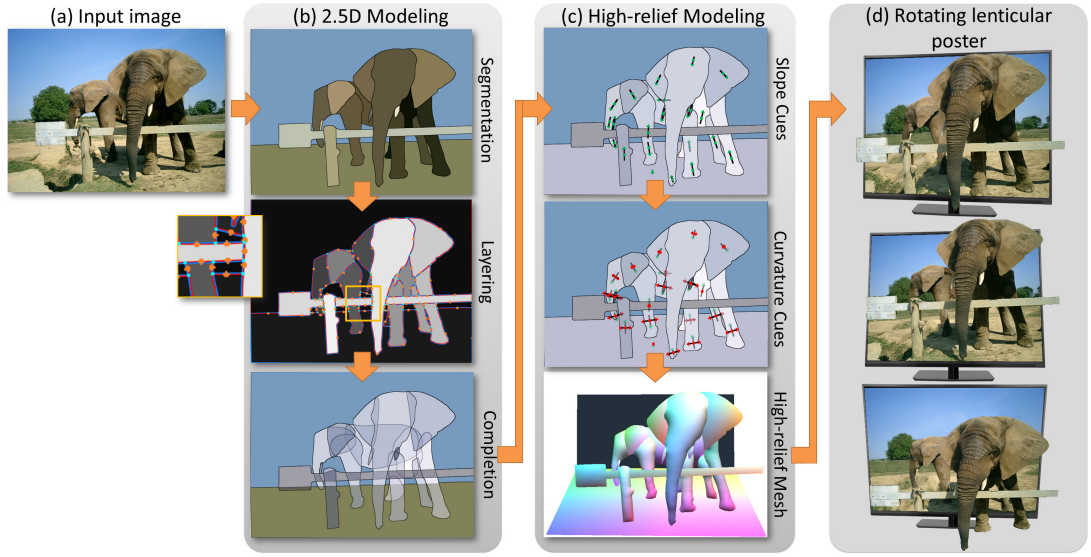


Figure 6.3: *Overview of our approach. Given an input image (a), we first generate a 2.5D model (b) by user-driven segmentation, automatic layering, and region completion. Then, using the slope and curvature cues marked up by users, we can lift up the planar regions to 3D by inflation and stitching, and produce a high-relief 3D model (c). Such a result can be plausibly rotated to produce the example rotating lenticular poster (renderings) (d).*

Our method has several advantages over existing 3D reconstruction methods. First, it can deal with common everyday objects without specific assumption on the object shape, e.g., similarity [151], planarity [152], and geometry [153]. Second, compared to [78], which produces only shallow geometry, our method can produce high-relief models, allowing the viewing of the reconstructed model at angles as large as 45 degrees from the image normal. Third, it does not require any external database for model fitting [154]. Lastly, our inflation method, which takes only simple user inputs, can support the reconstruction of complex 3D shapes with parts that are convex, concave, or even a mixture of both. These are demonstrated with wide variety of results produced from different kinds of input images.

6.2 Overview

Fig. 6.3 illustrates the procedure of our method with a running example. Overall, there are two major components:

i) 2.5D Modeling

Our 2.5D modeling method is largely the same technique that was employed in Chapter 5 for modeling the cartoon hair. However, we do not construct skeletons for each layer, since our goal is not animation or manipulation in this work.

Segmentation. This step is an important prerequisite to partition an image into semantic regions, and we utilize the same segmentation technique from Section 5.2, Chapter 5. An example is shown in Fig. 6.3(b)(top)), where the image has been segmented into semantic *regions*, where each region consists of pixels that are perceived to be depth continuous.

Local layering. Next, we determine and categorize the local layering, or depth ordering, between every pair of neighboring segmented regions: *above*, *below*, or *equal depth* (see the orange arrows in Fig. 6.3(b)(middle) that reveal the order). Again, we employ the technique from Section 5.3, Chapter 5, by analyzing T-junctions and overlap area, to construct the local layering information.

Completion. To seamlessly rotate a reconstructed result (high-relief model), we not only have to reconstruct the local geometry of each layer (or region), but also have to complete the occluded parts in each region, so that holes will not appear around the occlusion when we rotate the resulting models. Here, we consider three different completion cases for regions, by extending the completion method from Chapter 5 (see Section 6.3): i) expand a single region from the occluding boundary (same as Case 1 in Section 5.4); ii) connect non-neighboring regions under a common occlusion (same as Case 2 in Section 5.4); and iii) *double-sided* structure, where portions of an object's front and back sides are revealed simultaneously in the given image. Here, we must identify the connections between the front and back sides, and join them along the shared boundary, so that the region can be inflated as a single folded shape. This is a novel completion scenario that we consider in this work. Note that this step has to complete both geometry and texture, while considering the depth ordering among the regions (Fig. 6.3(b)(bottom)).

ii) High-relief Modeling

Inflation. As mentioned earlier, our key idea is to deform each planar layer in the 2.5D model, so that richer depth information can be conveyed. To reconstruct the depth information on each completed region, we take only simple user cues as hints (see Section 6.4.1): slope cues (Fig. 6.3(c)(top)) and curvature cues (Fig. 6.3(c)(middle)). Further, we formulate an optimization model to efficiently inflate each region while respecting the constraints (see Section 6.4.2). In particular, our method has two main advantages compared to previous user-driven methods. First, it does not require tedious specification of many different constraints while rotating the model during the reconstruction (e.g., [117]). Second, compared to state-of-the-art inflation methods such as [78], our method enables slanting of layers (by the slope cue) and control over the shape profile (by the curvature cue). By this, we can interactively inflate a wide range of natural objects with nontrivial organic shapes.

Stitching. Lastly, we sort and arrange the inflated regions based on the layering information and stitch adjacent inflated regions accordingly, so that gaps between regions do not appear when we rotate the model (Fig. 6.3(c)(bottom)). In particular, we formulate another optimization model to minimize the gaps in-between regions while maintaining proper depth ordering and connecting adjacent regions along their shared boundary with minimum deformation.

The forthcoming sections present key techniques we developed in this work: Section 6.3 for the completion step, Section 6.4 for the inflation step, and Section 6.5 for stitching.

6.3 Completion

The goal of completion is to appropriately fill the holes related to the occluded regions, so that we can produce plausible high-relief models that can be rotated, see Fig. 6.4. However, this is especially challenging with a single image as input, since we have to reconstruct the hidden occluded parts without any prior knowledge. In detail, we identify and consider three types of completion cases:

Figure 6.4: *Left: input image. Middle: rotate the high-relief result reconstructed with inflation and stitching but not completion; undesired holes will appear. Right: same result but reconstructed also with completion.*

Case (i): Expand a single region from the occluding boundary. This case refers to occluding boundaries that are not classified as the next two cases, i.e., an individual region is simply partially occluded by its neighboring regions. E.g., see the top occluding boundary of the right elephant's rightmost leg in Fig. 6.3(b) (mid), it is partially occluded by the elephant's right ear, so we extend the leg region upward from the boundary by a level-set method (adopted from Section 5.4 Case 2, and Osher et al.[155]). This method iteratively pushes the boundary curve from the boundary to expand the related region until the expanded boundary is smooth at the occluding endpoints (see Fig. 6.3(b) (bottom) for the result), while ensuring that the boundary curve is always below the associated neighboring regions.

Case (ii): Connect non-neighboring regions under a common occlusion. Here, an occlusion breaks an object part into multiple non-neighboring regions as in Section 5.4 Case 1, e.g., see Fig. 6.3(b) (top): the long horizontal bar of the fence is occluded by the right elephant's trunk. To resolve this case, we examine every pair of occluding boundaries among the segmented regions and look for a matching pair (under a common occluder) by the following criteria: i) RGB color histogram distance (d_c) by computing normalized cross-correlation between histograms with 256 bins, ii) distance between corresponding endpoints (d_e), where we normalize the image width to be 1.0 with fixed aspect ratio, and iii) difference in tangent-vector angles (in radian) at the T-junction endpoints on the occluding boundaries (d_t). If the joint probability $e^{-\sigma_c d_c^2} \cdot e^{-\sigma_e d_e^2} \cdot e^{-\sigma_t d_t^2} > 0.8$, we regard this as a match, e.g., the fence case in Fig. 6.3(b). Here, $\sigma_c = 1.0$, $\sigma_e = 2.0$, and $\sigma_t = 0.125$ for all results

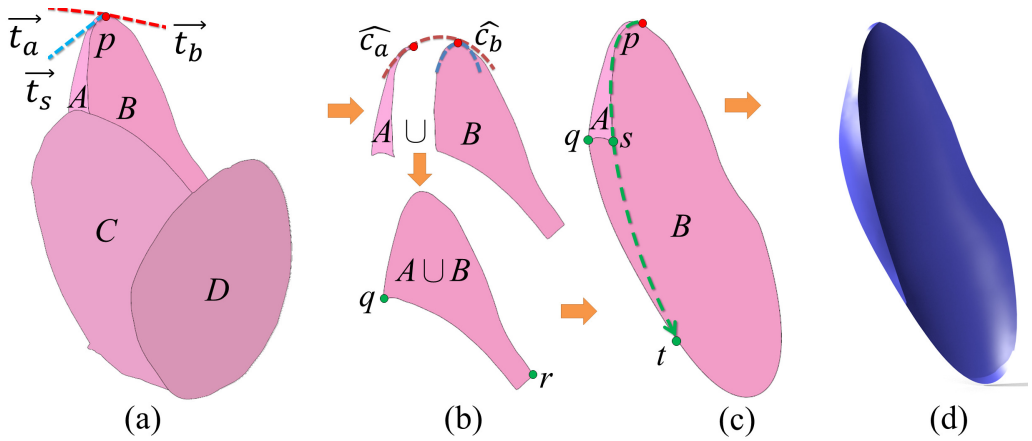


Figure 6.5: *Completion procedure in case (iii-a) for the top two flower petals in Fig. 6.4. (a) two neighboring regions A and B are identified as Case (iii-a) by analyzing the associated tangents vectors and turning angles at boundary end-points, e.g., p; (b) we extend B to be A∪B, (c) and use a co-completion strategy to complete the occluded portion of A and new B under C and D. (d) Preview: the reconstructed petal (A and B) in 3D.*

produced in the paper. After we find a match, we construct two Bézier curves (with tangent magnitude set to $0.3d_e$) to connect the related endpoints with C^1 continuity, and combine the two regions into one, see Fig. 6.3(b) (bottom) for the result of the fence. Additionally, the user may manually connect regions together.

Case (iii): Double-sided structures. This case commonly arises in objects with thin and curved structures, where both front (A) and back (B) sides of the same surface are exposed. Here, we consider two sub-cases: (a) fully-covered, where B can be extended fully to be $B \cup A$, e.g., see the top two flower petals in Fig. 6.5 and the teapot opening in Fig. 6.6; and (b) partially-covered, where B can only be extended to be $B \cup$ partial A, e.g., pinwheel and ribbon cables (see Fig. 6.6 and 6.7). Overall, we aim to identify the front-and-back association and complete the related regions plausibly into a double-sided geometry with the help of the inflation step.

To identify case (iii), we examine every 3-valence junction formed between a pair of neighboring regions, e.g., point p in Fig. 6.5(a). Here, we denote A and B as two related neighboring regions, where A is above B (without loss of generality); note that this information was produced earlier from the layering step. Next, we compute tangent vectors $\vec{t_a}$, $\vec{t_b}$, and $\vec{t_s}$ at point p along A's outer boundary, B's outer boundary, and their shared boundary, respectively (see

Fig. 6.5(a)), and the angles in-between $\vec{t_a}$ and $\vec{t_s}$ (denoted as θ_a), and $\vec{t_s}$ and $\vec{t_b}$ (denoted as θ_b). Around p, we also determine two boundary curves $\widehat{c_a}$ and $\widehat{c_b}$ (see Fig. 6.5(b)).

If $\theta_a + \theta_b \leq \pi$ and $\widehat{c_a}$ and $\widehat{c_b}$ are C^1 continuous, we regard A and B as a candidate pair for Case (iii), since by observation, their combined region locally forms a convex shape, e.g., $A \cup B$ as the back and A as the front (see Fig. 6.5(b)). Moreover, these conditions are satisfied only when the shared boundary of A and B is completely enclosed within $A \cup B$; this helps to filter out cases of non-double-sided occluding regions. We repeat the above procedure for all pairs of neighboring (occluding) regions and present the resulting candidate pairs to user for selection.

For neighboring regions identified as Case (iii), we complete their occluded parts while respecting their local layering by the following co-completion strategy:

(iii-a) Fully-covered: we first extend B to be $A \cup B$ (see Fig. 6.5(b)) since A (front) intuitively occludes part of B (back). Then, we use the procedure in Case (i) to expand B towards C and D from the boundary curve $\widehat{qr}$ (see Fig. 6.5(b&c)), and then complete A by smoothly extrapolating its non-shared boundary $\widehat{ps}$ to find $\widehat{st}$ (see Fig. 6.5(c)), where t is on the boundary of completed B, so the occluded part of A under C is bounded by $\widehat{qt}$ and $\widehat{st}$. Lastly, we connect A and B along the outer boundary given by $\widehat{pqt}$.

(iii-b) Partially-covered: we complete the occluded portion of B (bounded by p, s, and t) by smoothly extrapolating $\widehat{qs}$ until we find t, which is on A's boundary (see Fig. 6.7(c)). We choose q to be 10 pixels away from s along B's contour, excluding the non-shared boundary $\widehat{ps}$. Then, we connect A and B along the outer boundary $\widehat{tp}$ (see Fig. 6.7(d)).

By this co-completion strategy together with the upcoming inflation and stitching steps, we can reconstruct nontrivial double-sided structures, see Fig. 6.5(d) and Fig. 6.7(d). Note that [156] proposed a method to infer hidden contours from cusps and T-junctions in visible contours, but their method only deals with small suggestive contours and cannot work with the folded structures (our case), which have not been explored in any previous work.

Complete the texture. Lastly, we employ [157] to generate the textures on the

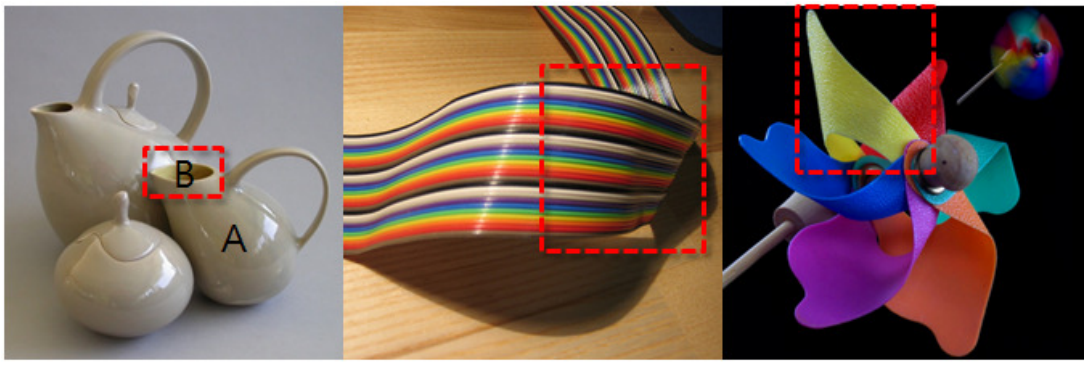


Figure 6.6: *More examples of double-sided structures (see also Fig. 6.1), where we see different sides of the same surface: the teapot opening (case iii-a: fully-covered, where B can be extended to B∪A) and cable and pinwheel structures (case iii-b: partially-covered), highlighted in red.*

Table 6.1: *Icons for slope and curvature markup cues in our GUI.*

Icon	Meaning
	Local slope along the out-of-image direction (+Z-axis)
	Positive mean curvature of surface (convexity)
	Negative mean curvature of surface (concavity)

completed portions by using the textures sampled from the remainder of the region. These texture samples are chosen to ensure the synthesized results to have minimum variations of luminance, pattern, and structure. Moreover, we fill and inpaint the holes left over on the background image by using [158].

6.4 Inflation

Automatic inference of depth and geometry from a single image is very challenging, due to ambiguities arising in the projection of a 3D scene onto a 2D image. A surface can be planar, convex, concave, or even a mixture of them; even human may fail to interpret its convexity and concavity as in the famous hollow-face dragon illusion. Hence, we take a user-driven approach, aiming at using very little user inputs to quickly reconstruct a plausible high-relief geometry. This method is similar to the inflation technique developed in Section 4.4, Chapter 4, as will be clear in the forthcoming text.

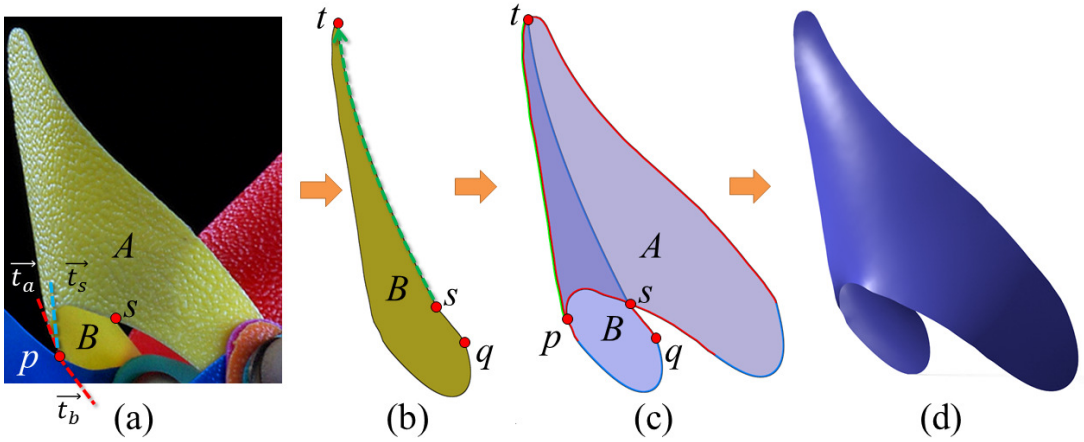


Figure 6.7: *Completion procedure in case (iii-b) with the pinwheel shown in Fig. 6.6. (a) two neighboring regions A and B that are identified as Case (iii-b); (b) complete B by the procedure described below in case (iii)(b). (c) the completed result, and (d) Preview: the reconstructed 3D pinwheel region (A and B) after the upcoming inflation and stitching steps.*

6.4.1 Markup Cues and User Interface

Markup Cues. Before the inflation step, we only have a 2.5D model of the scene, where each (completed) region is planar, i.e., parallel to the image space. To construct 3D information for each of these regions, we consider two very simple user-markup cues as hints for guiding the inflation:

(i) a *slope cue* constrains the local slope on a given region at the marked location, see Table 6.1(top): the green icon in the GUI and Fig. 6.8(left). Similar to the out-of-plane tilt handles in [159], our slope cues allow the user to slant the object to pop it out of the image plane. Mathematically, the slope cue has a *position* (the middle dot in the icon), where the cue takes effect, a *direction* (the icon arrow) in 2D, towards which the surface increases in +Z (out-of-image direction), and *length*, which corresponds to the slope magnitude.

ii) a *curvature cue* constrains the local mean curvature on a given region at the marked position, allowing us to manipulate the local shape profile. We offer two forms of curvature cues: positive and negative, which indicate convexity and concavity, respectively, see Table 6.1(middle & bottom) for the related icons. Mathematically, this cue consists of a *position* (the middle dot in the icon), where the cue takes effect, and *length*, which corresponds the amount of curvature. Note that we may also flexibly mark slope and curvature cues

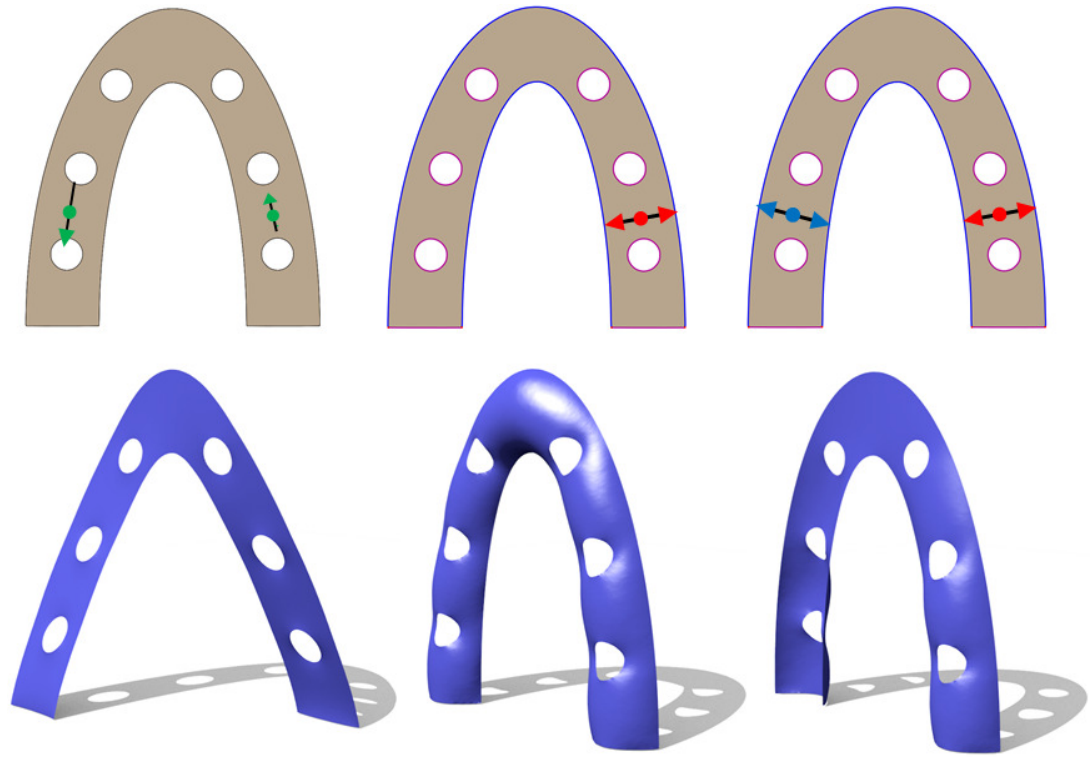


Figure 6.8: *Top: input images marked up with only slope cues (left) and only curvature cues (middle & right). Bottom: effect of slope cues on boundary vertices (left) and of curvature cues on interior vertices (middle & right) after the inflation. Note that we apply the Dirichlet and Neumann boundary conditions along the blue and purple (bottom of middle&right horseshoes in top row) boundaries, respectively.*

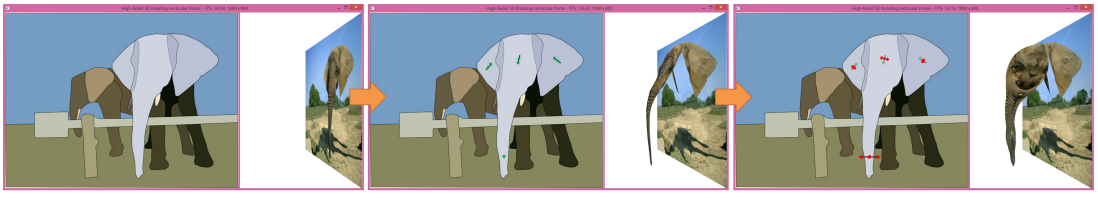


Figure 6.9: *User workflow in the inflation step. Left: the left sub-window in the GUI shows the completed regions for markup cue manipulation, while the right sub-window previews the inflation result; Middle: inflation result with only slope cues; and Right: result with both slope and curvature cues.*

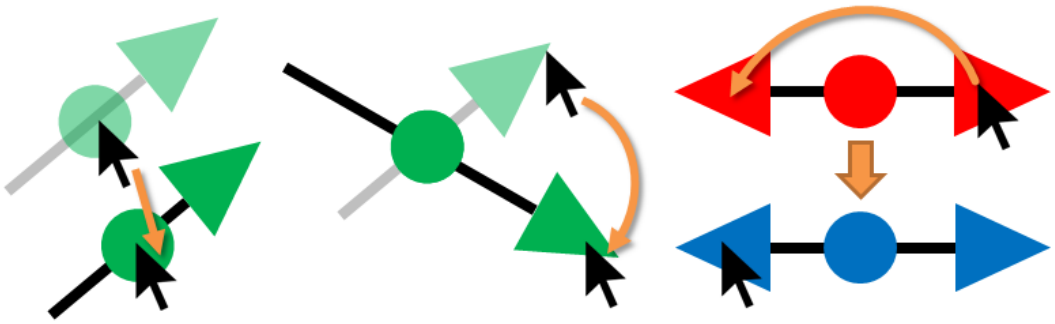


Figure 6.10: *User interaction with the markup cues. Note: black arrows indicate the mouse cursor. Left: relocate a cue by click & drag its center dot. Middle: modify a cue's magnitude/direction by click & drag its arrowhead. Right: to swap the sign of a curvature cue, click & drag its arrowhead across the center dot all the way to the opposite side.*

on different parts of the same region, see Fig. 6.8(left & right), for describing more complex shape profiles.

Interaction. We offer three interactions for manipulating the cues, see Fig. 6.10 (from left to right). First, we may drag the center dot of a cue to relocate it. Second, we may drag the cue's arrowhead to adjust its magnitude (slope or curvature, depending on which cue) and/or direction; note that dragging the arrowhead to the center dot nullifies the effect of the cue. Lastly, dragging the curvature cue's arrowhead across the center dot to the opposite side swaps between positive and negative curvature. Compared to previous works, our method neither requires painting a dense gradient map [160] nor careful and dexterous inputs [161].

User Interface. After the completion step, our interface (GUI) presents two sub-windows, see Fig. 6.9 (left): the left sub-window shows the completed regions with transparency in flat-shaded rendering style, while the right sub-window shows the inflated high-relief model, which is an interactive feedback to the user. On the left sub-window, the user can select a region and markup the slope and curvature cues. The right sub-window then updates the inflated results, when we add/modify the two cues, see Fig. 6.9 (middle and right). Note that we sort the regions based on layering order to prevent cluttering of regions caused by intersections. This is a heuristic used for preview only, and Section 6.5 presents an optimization model to ensure that the high-relief model is free of intersections and gaps.

At the beginning of the inflation step, our interface automatically initializes some slope and curvature cues with certain default values for immediate usage by the user:

- At the centroid of each region, we place one slope and one curvature cue; if the centroid lies outside the region, we randomly distribute ten points inside the region and compute the shortest distance from each point to the region boundary. We put the cue at the point with the largest distance among all.
- For slope cue, its default direction is along the major axis in the principal component analysis, and its default magnitude is flat, i.e., planar.
- For curvature cue, its default direction is perpendicular to that of the corresponding slope cue.

On the left sub-window, the user can interactively manipulate the magnitude and direction of each cue, relocate it, or add more cues to the same region to refine the inflation results. Moreover, the user may specify the boundary condition of a region for the inflation to be Dirichlet (fixed position for boundary vertices) or Neumann (zero gradient at boundary vertices along exterior normal), see Fig. 6.8. Lastly, the user may also specify whether the given region needs a back side. If so, we create and inflate a mirrored version of the given region, and sew it with the inflated geometry of the given region along the region boundary. By this, the reconstructed geometry could become complete when we rotate it at large angles.

6.4.2 Optimization

Given the user-specified slope and curvature cues, we formulate an optimization model to solve for the geometry of each region with the following goals: i) compute the z-coordinate of the boundary vertices from the user-specified slope cues, and ii) compute the z-coordinate of the interior vertices of each region from the curvature cues (see Fig. 6.8). Similar to the inflation model in Chapter 4, we first rely on differential operators to smoothly propagate the user specified constraints throughout each region to obtain the gradient and curvature fields, and reconstruct a surface that matches these fields.

i) Compute Z-coordinate of boundary vertices. Given an image region

$\phi(x,y)$, its gradient can be expressed as $\nabla\phi(x,y) = \vec{\Phi}$, which is to be reconstructed from the slope cues. The desirable slope field should be smooth over the region Ω and curl-free, so that the surface is integrable. Hence, we reconstruct $\vec{\Phi}$ by minimizing the following energy:

$$\min_{\vec{\Phi}} \iint_{\Omega} \left|\nabla\vec{\Phi}\right|^2 + \left|\nabla \times \vec{\Phi}\right|^2 dx\, dy,$$

subject to $\vec{\Phi}(x_i, y_i) = \vec{s_i}$, where $\{(x_i, y_i, \vec{s_i})\}$ denotes the set of slope cues in $\phi(x,y)$ and $\times$ denotes cross product. The first term represents the Laplacian, while the second term represents the curl. Obtaining ϕ from $\vec{\Phi}$ is straightforward through integration. We use the obtained slope field $\vec{\Phi}$ to compute the Z-coordinates of the boundary vertices by fixing one of them as 0. These boundary coordinates will be used in the next step to set the Dirichlet boundary conditions while computing the interior vertices.

ii) Compute Z-coordinate of interior vertices. Given a set of regions $\{\phi_i\}$ that form a double-sided structure, we first compute $f^* = \bigoplus_{i=1}^{m} \phi_i$, where $\oplus$ is an operation that combines the associated regions into a single 2-manifold surface by connecting the shared boundaries (from Case (iii) in Section 6.3). This allows the inflated double-sided surface to be continuous and smooth across the shared boundary. In case of single-sided structures, we simply set $f^* = \phi_i$.

The mean curvature normal of a surface is $2\kappa\vec{n} = \Delta f$, where Δ is the Laplace-Beltrami operator and f is a 3D surface. The discrete approximation using cotangent weights [162] at each vertex v_i is:

$$\kappa\vec{n} = \frac{1}{4A} \sum_{v_j \in N(v_i)} (\cot\alpha_{ij} + \cot\beta_{ij})(v_i - v_j)\ , \tag{6.1}$$

where A is the sum of the 1-ring triangle areas around v_i, $N(v_i)$ is the set of 1-ring neighboring vertices of v_i, and α_{ij} and β_{ij} are the angles opposite to edge (v_i, v_j) on either side in the original mesh, see [162].

Similar to [78], we only inflate the surface along z, so we only consider the z-component of the mean curvature normal $\kappa\vec{n}$, which we denote as $\widehat{\kappa}$. To achieve a C^1 continuous surface, particularly across the shared boundary for double-sided structures, we utilize biharmonic equation $\Delta^2 f = \nabla^4 f = 0$. The

order of this PDE can be reduced by using two second-order coupled PDEs, i.e., $\Delta f = 2\kappa\vec{n}$ and $\Delta\widehat{\kappa} = 0$. Notably, a similar reduction for the biharmonic equation has been explored in [163, 117]. Our approach is similar to [163], except we employ it to only solve for the z-coordinates of the vertices. There are two advantages: first, solving for only z-coordinates reduces the computation time, and second, this preserves the original 2D projection of the surface. Allowing the vertices to move arbitrarily in 3D space may result in inner vertices to cross the region boundary if the curvature is large. [117] used a different strategy to reduce computation time; they replaced the geometry dependent Laplace-Beltrami operator with the uniform Laplacian operator and included additional constraints to preserve the edge lengths. This eliminated the need to compute weights for the discrete Laplacian in each iteration when the geometry changes. In our case, we simply compute the cotangent weights once from the rest mesh and retain these weights in the inflation step.

Here, we first solve for $\Delta\widehat{\kappa} = 0$ by discretizing the equation at each vertex v_i as

$$\sum_{\widehat{\kappa}_j \in N(\widehat{\kappa}_i)} (\cot\alpha_{ij} + \cot\beta_{ij})(\widehat{\kappa}_i - \widehat{\kappa}_j) = 0 \ ,$$

and the region boundary $\partial\Omega$ subject to the Neumann boundary conditions:

$$\nabla\widehat{\kappa}(\mathbf{x}) \cdot \mathbf{n} \ = \ 0 \qquad \forall \mathbf{x} \in \partial\Omega \ ,$$

where $\mathbf{n}$ is the exterior normal to the boundary. The target curvature field $\widehat{\kappa}$ smoothly interpolates the user-specified curvature cues $\{(x_i, y_i, \widehat{\kappa}_i)\}$ over Ω. Next, we recover the inflated 3D surface f by solving the Poisson equation $\Delta f = 2\kappa\vec{n}$ subject to Dirichlet boundary conditions B_D or Neumann boundary conditions B_N, where $B_D \cup B_N = \partial\Omega$:

$$\begin{aligned} f(\mathbf{x}) \ &= \ f^* \qquad \forall \mathbf{x} \in B_D \quad \text{and} \\ \nabla f(\mathbf{x}) \cdot \mathbf{n} \ &= \ 0 \qquad \forall \mathbf{x} \in B_N \ . \end{aligned}$$

6.5 Stitching

After inflating individual regions, we need to arrange the regions along the image normal direction before forming a high-relief model. If this is done

Table 6.2: *Time taken to generate the high-relief models shown in this chapter, including user interaction.*

Inputs	# Triangles	Segmentation		Layering	Completion	Inflation				Stitching	Total time
						# Slope & Curvature Cues	Average Compute Time per Region		User annotation		
		# Regions	Time				Slope	Curvature			
Bird (Fig. 2)	36846	31	9m 51s	0.029s	2.051s	34	0.0258s	0.2133s	4m 40s	1m 8s	15m 41s
Hockey (Fig. 13, row 1)	52396	53	10m 20s	0.050s	3.682 s	53	0.0248s	0.1615s	6m 1s	2m 5s	18m 30s
Knot (Fig. 13, row 2)	37178	4	5m 45s	0.014s	1.007s	3	0.0787s	0.4889s	0m 23s	0m 45.3s	6m 54s
Elephant (Fig. 13, row 3)	32948	18	12m 31s	0.018s	1.391s	21	0.0299s	0.2565s	2m 45s	1m 2s	16m 19s
Ballet (Fig. 13, row 4)	33180	23	5m 12s	0.024s	1.503s	30	0.0507s	0.2213s	3m 48s	0m 40.7s	9m 42s
Seacow (Fig. 13, row 5)	24842	5	2m 33s	0.009s	0.308s	7	0.0612s	0.4264s	0m 45s	0m 35.8s	3m 54s
Flower (Fig. 14, row 1)	36597	26	4m 59s	0.021s	1.659s	26	0.1279s	0.4830s	3m 10s	0m 39.2s	8m 50s
Shoes (Fig. 14, row 2)	10974	8	2m 24s	0.008s	0.325s	10	0.0777s	0.5301s	1m 15s	0m 10.3s	3m 50s
Pinwheel (Fig. 14, row 3)	23983	18	6m 17s	0.014s	1.473s	18	0.1403s	0.4465s	2m 53s	0m 10.1s	9m 22s

inappropriately, we could see undesirable artifacts such as gaps and intersections, and the problem could be exaggerated when different parts of the model overlap (see Fig. 6.11 (left column)). Challenging examples in our work include double-sided structures, i.e., regions such as flower petals that are completed through Case (iii) in Section 6.2, which are often inflated in both directions since they are comprised of convex and concave portions.

To resolve this issue, we optimize the geometry of each region by minimizing the following energy function with respect to hard constraints derived from the relative depth information estimated during the layering step:

$$\min_{f'} \sum_i \left\{ \int_\Omega \left|\nabla f_i - \nabla f_i'\right|^2 + \left|f_i' - Above(f_i')\right|^2 dA \right\},$$

subject to:

$$\begin{aligned} f_i'(v_i) &\leq f_j'(v_j) \quad \forall (v_i, v_j) \in \{Front(f_i') \leq Back(f_j')\}, \\ f_i'(v_i) &= f_j'(v_j) \quad \forall (v_i, v_j) \in Stitch(i,j), \end{aligned}$$

where f_i and f_i' are the original (inflated) and deformed height fields of region i, respectively, v_i and v_j are the vertices (image locations) of region i (f_i) and j (f_j), respectively, $Above(\cdot)$ refers to the neighboring region(s) immediately above the current region, $Front(\cdot)$ and $Back(\cdot)$ are the front and back sides of a region, respectively, and $Stitch(i,j)$ is the set of vertices along the non-occluding shared boundary (equal depth from the layering step) of i and j (if any).

$Front(\cdot)$ and $Back(\cdot)$ are mainly used for double-sided structures. For reg-

Figure 6.11: *Stitching. Left: intersection/gaps between regions in the high-relief model before stitching. Middle: stitching result without the second term in the objective function can still contain gaps (highlighted in red). Right: stitching result using the proposed equation.*

ular single-sided geometries, we set $Front(\cdot)=Back(\cdot)$. The first term in the energy function is a fidelity term for minimizing the deformation on each of the height fields, while the second term is for minimizing the size of gaps between regions (see Fig. 6.11(right column)). The first hard constraint enforces the original 2.5D layering among the regions (see Fig. 6.11(middle column)) and the second hard constraint glues the neighboring regions along the non-occluding shared boundary between them (if any). Note that the deformation caused by this method is minimum and preserves the shape profile created by the user as much as possible. Since the stitched mesh may have discontinuities along the non-occluding shared boundary between the stitched regions, we use a local smoothing operation around these contours to reduce these artifacts.

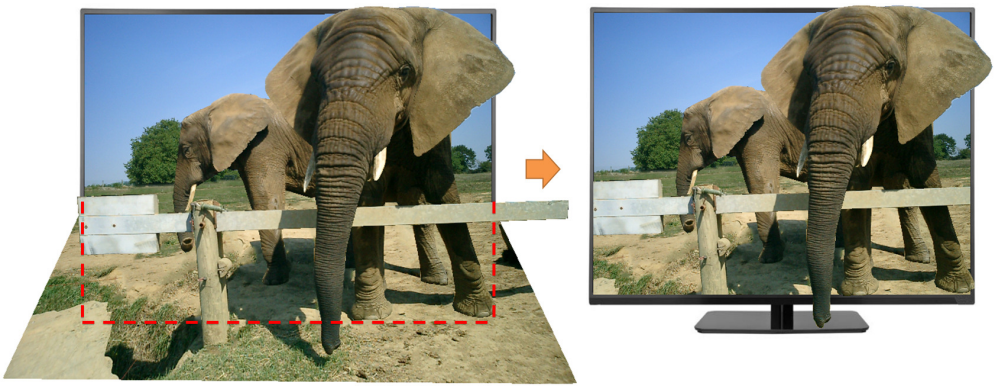


Figure 6.12: *Post-processing. Left: the high-relief model after inflation and stitching (before cropping). Right: after cropping the ground plane by the extended red box, the result could resemble a lenticular poster.*

6.6 Results & Discussion

Implementation. To evaluate our method, we implemented it in C++ and evaluated it on a desktop computer with a 3.4GHz CPU and 4GB memory.

Experiment. We recruited six participants (three female and three male; aged 24 to 32) on a volunteering basis to try out our software. Each of them was first given 10 minutes of learning time to get familiar with the software, e.g., how to define and adjust the cues to produce the 3D reconstructions. After that, they can make use of our system to edit the local shape profile and produce various inflation results with some input photos, e.g., Fig. 6.9. Overall, they took around 3-18 minutes to reconstruct a high-relief model, depending on the amount of user interactions and the image complexity. The average time taken in the whole process, including the user interactions, is shown in Table 6.2.

From the experiment, we observe that the interactive feedback mechanism is very important in both the learning process and the editing process. All participants mentioned that they could easily figure out how the arrow direction and length affect the 3D reconstruction by interactively modifying the cues and observing the changes in the reconstruction results. Therefore, all the users were able to achieve a desired local shape profile in around two to three iterations (on average), thanks to the interactive feedback mechanism. A video of one of the participants in action can be found in `https://youtu.be/8_qkHGfEnag`.

Post-processing. To make the high-relief result resemble a lenticular poster, we further extend the bottom side of the image to form a rectangular region

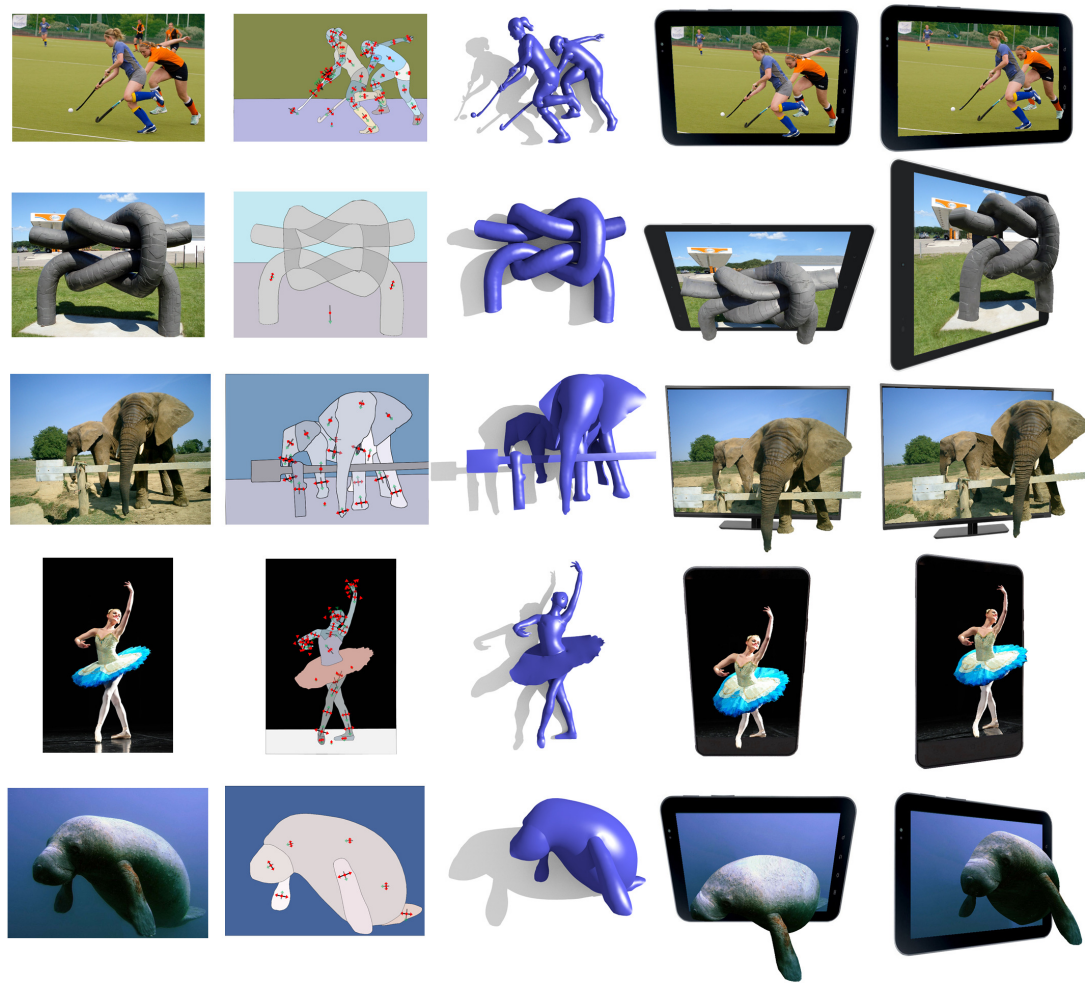


Figure 6.13: *Results for single-sided structures: Hockey, Knot, Elephant, Ballet, and Seacow (rotated* 12° *to* 45° *from the original viewing direction)*

(see the red box with dash lines in Fig. 6.12(left)), and apply this box to crop the ground plane (see Fig. 6.12(right)). In addition, we could present the resulting high-relief model optionally with a TV or smart phone frame to enhance the 3D perception.

Results. Fig. 6.13 shows our high-relief results generated for various organic objects that are without double-sided structures. Here, the first column shows the input (single) image, the second column shows the user-defined cues, the third column shows the reconstructed high-relief models, while the last two columns show novel views of the models with large rotation from the original view. Fig. 6.15 presents another example, demonstrating that our method can also handle foreground objects with holes.

Fig. 6.16 shows our high-relief results generated for objects with double-sided

Figure 6.14: *An example of our method applied to an input cartoon image (left), and rendered from four new viewpoints (right).*

structures. These results are completed by our method in Section 6.3 and later inflated and stitched by our methods in Sections 6.4 and 6.5. Since we can only show static images in the paper, we present animated versions of these high-relief results in the supplementary video (`https://youtu.be/vIkQqFIEUh4`). Note that it is straightforward to apply to cartoon images, as shown in Fig. 6.14. However, care needs to be taken in case of textured cartoon images, since cartoons usually have line based textures, which are difficult to inpaint. Additionally, anomalies in the texture mapping close to the region boundaries due to inflation are usually apparent in cartoon images. We discuss some more extensions of the current approach to cartoon applications in Chapter 7.

Comparison. Next, we compare our results with two closely-related recent works: [149] and [78]. In particular, we compare the visual plausibility of the results, with large rotations from the original viewing direction. Please refer to the supplementary video: `https://youtu.be/vIkQqFIEUh4`, for animated versions of the comparison.

Zeng et al. [149] presented a method to generate stereoscopic effects from a single segmented image. However, their method can only allow small rotation angles (7°) since their 3D mesh is flat. Our results still have plausible 3D appearance even with large rotations (see Fig. 6.17).

Figure 6.15: *Our method can handle objects with holes (within a segmented image region). Left: input image. Right: inflated result.*

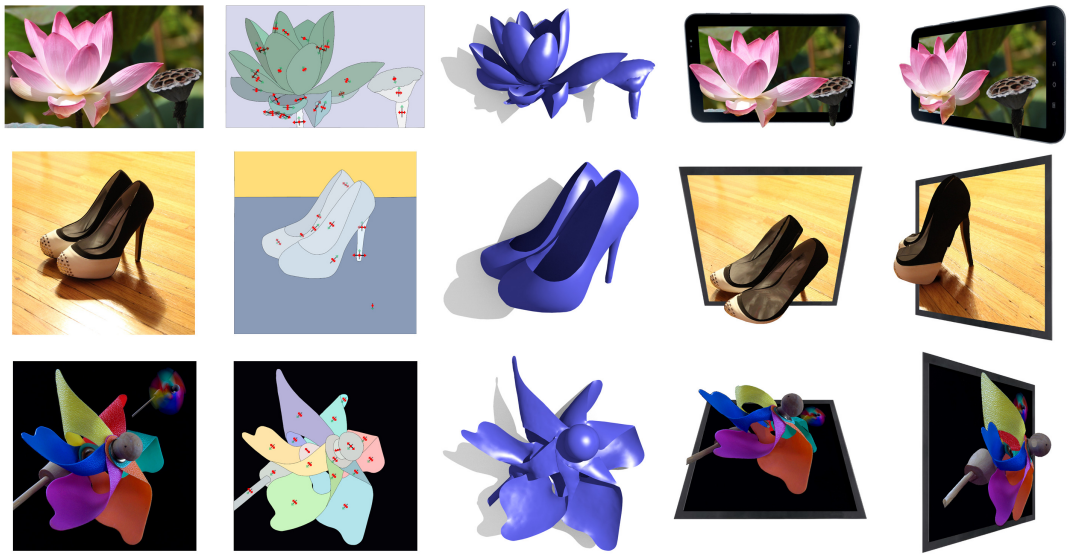


Figure 6.16: *Results for double-sided structures. Flower, Shoe, and Pinwheel (rotated* 30° *to* 45° *from the original viewing direction).*

Figure 6.17: *Comparison with [149]. Columns 1 & 2: results from [149] with ∼7° rotations downloaded from their website, and Columns 3 & 4: our results with much larger rotations. Please refer to the supplementary video for animated version of this comparison.*

The method proposed by Sýkora et al. [78] was mainly designed for creating a proxy mesh of the object in the input image and then enhancing it with global illumination. Hence, while the resulting bas-relief meshes appear visually plausible in the original view from the front, it is revealed to be flat after rotations (see Fig. 6.18). As a comparison, our high-relief mesh results can achieve a pop-out visual effect in the side views.

6.7 Summary

This chapter presented a novel approach to reconstruct high-relief 3D models from a single input image, aiming at achieving large view rotation on the reconstructed results. We particularly consider common organic objects with nontrivial shape profile and the reconstruction of objects composed of double-sided structures. In summary, this work has the following technical contributions. First, we develop a completion method that considers three common cases of occlusion found in natural images; particularly, we are able to complete the geometry of double-sided structures in different cases. Second, we

Figure 6.18: *Comparison with [78]. Columns 1 & 2: results kindly generated by Daniel Sýkora using his method in [78], and Columns 3 & 4: our results with more plausible geometry under same amount of 3D rotations from the original view. Please also refer to the supplementary video.*

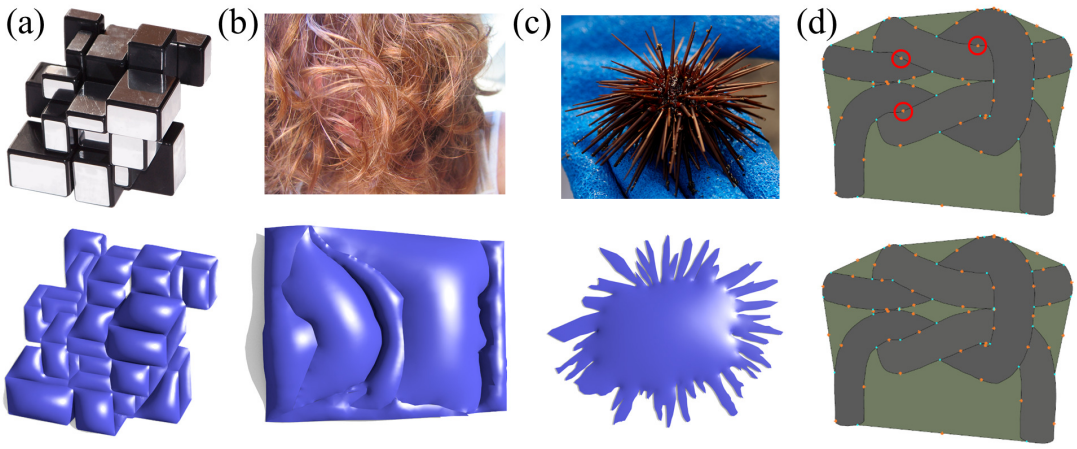


Figure 6.19: *Limitations. (a) polyhedral shape with sharp edges, (b & c) objects with very fine details are hard to be segmented, (d) intertwined regions cannot be consistently estimated by our local layering step.*

devise an interactive inflation model; taking little amount of user inputs (slope and curvature cues), we can perform a constrained optimization to quickly construct an inflated shape profile with large rotation for interactive editing with preview. We also develop a stitching method that respects the layering and double-sided information to aid the high-relief reconstruction. Lastly, we show the superiority of our results by comparing it with two very recent work and present lenticular poster renderings of our results for a wide variety of common organic objects.

Limitations

We discuss the limitations of this work as related to different steps in the method:

- *Segmentation:* some images can be too complex for proper segmentation, e.g., thin and fine structures (see Fig. 6.19(b&c)); hence, we cannot form meaningful regions and compute layering.
- *Local layering:* since this chapter makes use of the 2.5D modeling approach from Chapter 5, our method shares the same limitation in handling the layering of intertwined structures (see red circles in Fig. 6.19(d)); users can manually divide the edge and correct the layering.
- *Completion:* our image completion is a best-effort step, which tries to estimate the occluded part of a region based on what is visible. Since it does not involve semantics, it cannot connect hidden regions arbitrarily.
- *Inflation:* our method is mainly designed for reconstructing smooth and organic shapes similar to the approach in Chapter 4, where the inflation model assumes smooth surfaces (see Fig. 6.19(a)), so we need extra constraints to properly handle objects with sharp edges. Moreover, our current method supports only a parabolic profile, perfect spherical shapes cannot be generated.
- *Stitching:* our method generally works well in stitching, except for ill-completed meshes, where we lack information to correctly compute the stitching.

CHAPTER 7

Conclusions & Future Work

In this thesis, we have presented several new computational methods to help with various stages in the cartoon production pipeline. With these methods, we are able to support several applications which have been discussed in detail.

First, we presented a novel interactive camera-based method to capture and vectorize freehand line drawings on paper. By using a robust spatio-temporal tracking technique that can efficiently localize and track the pen tip in video frames, we are able to capture the trajectory of the pen tip over paper while the drawing process is being carried out. This technique combines the strength of a fast-cascade classifier with discriminable features and optical-flow tracking (which is less accurate) to achieve high performance and accuracy. Second, we extract stroke points in the video frames by considering not only the edge profiles around the estimated pen-tip trajectory but also the hand and pen occlusion to improve the stroke recognition accuracy. Lastly, by using the spatio temporal information that was captured earlier, we employ clustering to produce meaningful strokes that are spatio-temporally coherent with the actual strokes drawn by the users. We perform a number of experiments to evaluate and examine different aspects of our method: computation performance, accuracy against actual drawings and an offline commercial tool for vectorization, as well as pen and hand occlusion. We demonstrated our method on assorted clean line drawings, as well as applied it to produce the video-scribing animation effect with some of our results. The current camera based vectorization approach is was primarily designed to test the feasibility of such an approach to freehand line drawings. In future, we plan to extend our method in various directions as summarized below.

- First, we employed a simple RGB webcam in our baseline setup, which has certain limitations when it comes to handling crowded, sketchy strokes which are commonly drawn by artists in initial stages of the cartooning pipeline. This is because our current method relies on tracking the pen tip and analyzing the ink trail on paper along its trajectory to estimate

the pen-paper contact. Such an approach becomes unreliable in the presence of crowded sketchy strokes in close proximity. Therefore, we are interested in employing modern RGB-D webcams, e.g., those powered by Intel RealSense technology [164, 165], that can help to simplify these issues if we leverage the depth information.

- Second, since our approach is primarily camera based, we assume that the pen tip is not occluded in any of the video frames. While this is a reasonable assumption, it is not uncommon to see such occlusions in real world scenarios. In future, we plan to develop hybrid techniques that can combine the advantages of offline and online vectorization approaches. In detail, a technique which jointly analyzes both a scanned image of the completed drawing by employing offline techniques, as well as the entire video of the drawing process using online techniques, can significantly improve the accuracy of the vectorization. Online analysis can provide temporal information to resolve ambiguities in vectorization, while offline analysis can improve accuracy of reconstruction and help deal with corner cases such as occlusions of pen tips, where relying solely on an online approach might fail.

- Third, our current framework assumes a fixed paper setup for simplicity. However, artists often rotate the canvas for convenience when drawing. We plan to extend our method to support such a feature by tracking the orientation of the paper in each frame. This is could be carried out using markers in the corners of the paper, or a combined color (assuming the paper is white) and shape (quadrilateral shape prior) based tracking. From this recovering the homography transformation should be straightforward to tackle the effects of a paper that is not fixed.

- Lastly, we would like to further relax the constraints by assuming a free camera, so that our framework can be extended to RGB-D smart glasses, e.g. [166]. We envision a smart glass based drawing system that would allow users to wear glasses and draw on paper to easily obtain a digital line drawing.

Second, we proposed a novel method to analyze wrinkle strokes in clean line drawings and to automatically produce shading over the drawing. The major contributions of the paper include: i) the computational models of five perceptual cues from gestalt psychology (T-junction, shape convexity, proximity,

continuity, and regularity) for wrinkle strokes, where we design the models by exploring the local shape, spatial relation, and interaction of strokes by consulting relevant psychological principles; ii) an optimization formulation for producing globally consistent wrinkles by combining and balancing the local estimation from the cues, and iii) a wrinkle-aware inflation method that can support the generation of two commonly used shading styles seen in cartoons. Compared to the prior work, our method has several unique features such as the ability of automatically generating rich wrinkle geometry conveyed in the interior strokes and the ability of producing globally consistent shape inference, which have been demonstrated by various experimental examples. As the first attempt to automating shading process from a clean line drawing, our work just considers the major aspects of shading with regards to the perceived 3D shape and light source. There are several potential avenues for future work, as follows.

- First, it is worth pointing out that in real cartoon production, there are also other factors that affect the shading style, for example, the semantics of the scene and artists' personal preference. Therefore, in future we plan to extend our work by exploring more shading styles and characteristics, developing data-driven methods to learn artists' shading styles.
- Second, we are interested in refining our inflation model to support polyhedral shapes with planar facets and sharp edges, which are quite common in cartoons. This could be achieved by a user-driven approach, and modifying our inflation model to allow discontinuous interpolation of the gradient field.
- Third, we aimed for a pseudo 3D model in this work since we mainly focused on shading. It would be interesting to investigating how to extend our framework to applications requiring physical correctness, possibly by utilizing multi-view line drawings.
- Fourth, we would like to introduce significant interactivity in our framework to support drawing and editing of line strokes with real-time shading preview, and tools for interactive re-shading of existing line drawings.
- Lastly, we would like to extend our method to support shading of animated line drawings. This could be currently achieved by interpolating

between the inflated meshes generated for each keyframe. However, this may not be ideal since we need to ensure temporal consistency in our stroke interpretation framework so that we do not see sudden changes in the shading. This will be particularly exaggerated with topological changes and missing correspondence between strokes in subsequent keyframes.

Third, we presented a novel 2.5D approach to modeling, animating, and manipulating hairs in a single cartoon image. We made three key contributions in this work. First, we derived an effective layering metric from the Gestalt psychology; it can automatically compute the layering order among hair strands in a single cartoon image. Second, we developed a novel layer completion method that can automatically fill the occluding parts of hair strands; by this, we can construct a 2.5D hair model with skeletons to support hair animation and manipulation with appropriate layering. Lastly, we devised a simplified simulation model to animate the skeletons in hair strands with minimum effort. To demonstrate the capability of our approach, we compared our results with conventional cartoon videos, and applied our method to produce hair animation and manipulation results on assorted cartoon characters. We plan to explore several future extensions for this work.

- In this work, we mainly focused on modeling the hair in cartoon images. A natural extension of this work that we are interested in, is generalizing this framework to arbitrary cartoon images.
- Our layering model assumes a fixed layering between adjacent regions. However, there are cases of intertwined structures [87] in practice, where the layering order changes for different parts of the regions. We are interested in extending our layering model to automatically infer and represent such cases.
- Lastly, we would like to incorporate lighting and shadows in our framework. Currently, the shadows are unnaturally pinned to the hairs during animations or manipulations, resulting in some noticeable artifacts.

Lastly, we extended our 2.5D modeling method by interactively deforming the planar layers into 3D, and presented a novel approach to reconstruct high-relief 3D models from a single input image, aiming at achieving large view rotation on the reconstructed results. We particularly consider common or-

ganic objects with non-trivial shape profile and the reconstruction of objects composed of double-sided structures. In summary, this work has the following technical contributions. First, we develop a completion method that considers three common cases of occlusion found in natural images; particularly, we are able to complete the geometry of double-sided structures. Second, we devise an interactive inflation model. Taking little amount of user inputs (slope and curvature cues), we can perform a constrained optimization to quickly construct an inflated shape profile with large rotation for interactive editing with preview. We also develop a stitching method that respects the layering and double-sided information to aid the high-relief reconstruction. Lastly, we show the superiority of our results by comparing it with two very recent work and present lenticular poster renderings of our results for a wide variety of common organic objects. In future, we plan to explore the following research directions.

- First, we would like to identify more solid applications for our framework in the cartooning pipeline. While the current method can be directly applied to cartoon images, we are interested in finding ways to incorporate the detailed high-relief information in specific applications. For example, applications such as stereoscopy and re-lining (rendering the input drawing from different views), can be achieved by projecting the depth map from a different viewpoint, and deforming the input correspondingly while preserving the original appearance as much as possible. Another application is inbetweening, where the 3D information can aid in challenging scenarios such as out-of-plane rotations. Generally, a vector based input can better help in these scenarios.
- Second, we plan to investigate methods to automate the estimation of slope and curvature cues from the image contents by analyzing the local image information such as shading, to partially automate the reconstruction process. This is a challenging ill-posed problem, and data-driven approaches are worth exploring.
- Finally, it is also interesting to combine our inflation techniques from Chapters 4 and 6, for practical applications involving complex natural images with wrinkled surfaces.

Bibliography

[1] Wilson. Inking. mangakasjournal.blogspot.sg/2009/02/manga-drawing-process-inking-part-4.html, 2009. [Online; accessed 03-February-2017].

[2] CreativeTiffany. http://instructables.com/id/How-to-Clean-and-Tone-a-Paper-Drawing-in-Gimp-no-/, 2012. [Online; accessed 03-February-2017].

[3] Wilson. Inking. mangakasjournal.blogspot.sg/2009/01/manga-drawing-process-inking-part-1.html, 2009. [Online; accessed 03-February-2017].

[4] Nielson Sam. Painting Process. theartcenter.blogspot.sg/2010/03/sam-nielson-painting-process.html, 2003. [Online; accessed 03-February-2017].

[5] Edwin Catmull. The problems of computer-assisted animation. *Proc. SIGGRAPH*, 12(3):348–353, 1978.

[6] Jessica Abel and Matt Madden. *Mastering Comics*. Turtleback Books, 2012.

[7] T. White. *Animation from Pencils to Pixels: Classical Techniques for Digital Animators*. NetLibrary Inc. Focal, 2006.

[8] N. Burtnyk and M. Wein. Computer-generated key-frame animation. *J. of the SMPTE*, 80(3):149–153, 1971.

[9] N. Burtnyk and M. Wein. Computer animation of free form images. *Proc. SIGGRAPH*, 9(1):78–80, 1975.

[10] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Commun. ACM*, 19(10):564–569, 1976.

[11] Walt Disney Animation Studios and Pixar. CAPS: Computer animation production system, 1989.

[12] Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus Gross. Topology-driven Vectorization of Clean Line Drawings. *ACM Trans. Graph.*, 32(1):4:1–4:11, 2013.

[13] Bin Bao and Hongbo Fu. Vectorizing line drawings with near-constant line width. In *IEEE Intl. Conf. Image Processing*, pages 805–808, 2012.

[14] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. *ACM Trans. Graph.*, 27(3):92:1–92:8, 2008.

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

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

[17] Liu Wenyin and Dov Dori. A survey of non-thinning based vectorization methods. In *Advances in Pattern Recognition*, volume 1451 of *Lecture Notes in Computer Science*, pages 230–241. 1998.

[18] Ju Jia Zou and Hong Yan. Cartoon Image Vectorization Based on Shape Subdivision. In *Proc. Intl. Conf. Computer Graphics*, CGI '01, pages 225–231, 2001.

[19] Xavier Hilaire and Karl Tombre. Robust and Accurate Vectorization of Line Drawings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(6):890–904, 2006.

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

[21] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Fidelity vs. simplicity: A global approach to line drawing vectorization. *ACM Trans. Graph.*, 35(4):120:1–120:10, 2016.

[22] David S. Doermann and Azriel Rosenfeld. Recovery of temporal information from static images of handwriting. *Intl. J. Comput. Vis.*, 15(1-2):143–164, 1995.

[23] Y. Kato and Makoto Yasuhara. Recovery of drawing order from scanned images of multi-stroke handwriting. In *Proc. Intl. Conf. Document Analysis and Recognition.*, pages 261–264, 1999.

[24] Carsten Steger. Extracting curvilinear structures: A differential geometric approach. In *Proc. Euro. Conf. Comput. Vis. (ECCV)*, volume 1064 of *Lecture Notes in Computer Science*, pages 630–641. 1996.

[25] Ming-Ming Cheng. Curve structure extraction for cartoon images. In *Joint Conf. Harmonious Human Machine Environment*, pages 13–25, 2009.

[26] Adobe Systems. Adobe Illustrator. `http://www.adobe.com/products/illustrator.html`. [Online; accessed on 03-Feb-2017].

[27] SoftSoft. WinTopo. `http://wintopo.com/`, 2012. [Online; accessed on 03-Feb-2017].

[28] Jrgen Steimle. Survey of pen-and-paper computing. In *Pen-and-Paper User Interfaces*, Human-Computer Interaction Series, pages 19–65. 2012.

[29] M. Sperber, M. Klinkigt, K. Kise, M. Iwamura, B. Adrian, and A. Dengel. Handwriting reconstruction for a camera pen using random dot patterns. In *Intl. Conf. Frontiers in Handwriting Recognition*, pages 160–165, 2010.

[30] Megumi Chikano, Koichi Kise, Masakazu Iwamura, Seiichi Uchida, and Shinichiro Omachi. Recovery and localization of handwritings by a camera-pen based on tracking and document image retrieval. *Pattern Recognition Letters*, 35:214–224, 2014.

[31] Anoto. Anoto Digital Pens. `http://www.anoto.com`. [Online; accessed 03-February-2017].

[32] Wacom. Inkling. `http://inkling.wacom.com/`. [Online; accessed 03-February-2017].

[33] Feng Lin and Xiaoou Tang. Dynamic stroke information analysis for video-based handwritten chinese character recognition. In *Proc. Intl. Conf. Comp. Vis. (ICCV)*, volume 1, pages 695–700, 2003.

[34] Xiaoou Tang, Feng Lin, and Jianzhuang Liu. Video-based handwritten Chinese character recognition. *IEEE Trans. Circuits and Sys. for Video Tech.*, 15(1):167–174, 2005.

[35] M.E. Munich and P. Perona. Visual input for pen-based computers. In *Proc. Intl. Conf. Pattern Recognition*, volume 3, pages 33–37, 1996.

[36] M.E. Munich and P. Perona. Visual Input for Pen-based Computers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(3):313–328, 2002.

[37] Markus Wienecke, Gernot A. Fink, and Gerhard Sagerer. A handwriting recognition system based on visual input. 2nd Intl. Workshop on Computer Vision Systems, pages 63–72, 2001.

[38] Jae-Hyun Seok, Simon Levasseur, Kee-Eung Kim, and Jin Hyung Kim. Tracing handwriting on paper document under video camera. In *Proc. Intl. Conf. Frontiers in Handwriting Recognition*, pages 109–110, 2008.

[39] Y. Cao, A. B. Chan, and R. Lau. Automatic stylistic manga layout. *ACM Trans. Graph.*, 31, 2012.

[40] Ying Cao, Rynson W. H. Lau, and Antoni B. Chan. Look over here: Attention-directing composition of manga elements. *ACM Trans. Graph.*, 33(4):94:1–94:11, 2014.

[41] Alexander Kort. Computer aided inbetweening. In *Proc. Intl. Symp. Non-photorealistic Animation and Rendering*, NPAR '02, pages 125–132, New York, NY, USA, 2002. ACM.

[42] Jun Xing, Li-Yi Wei, Takaaki Shiratori, and Koji Yatani. Autocomplete hand-drawn animations. *ACM Trans. Graph.*, 34(6):169:1–169:11, 2015.

[43] Daniel Sýkora, John Dingliana, and Steven Collins. LazyBrush: Flexible painting tool for hand-drawn cartoons. *Comput. Graph. Forum*, 28(2):599–608, 2009.

[44] Yingge Qu, Tien-Tsin Wong, and Pheng-Ann Heng. Manga colorization. *ACM Trans. Graph.*, 25(3):1214–1220, 2006.

[45] Soon Hock Seah and Tian Feng. Computer-assisted coloring by matching line drawings. *The Visual Computer*, 16(5):289–304, 2000.

[46] Jie Qiu, Hock Soon Seah, Feng Tian, Quan Chen, and K. Melikhov. Computer-assisted auto coloring by region matching. In *Proc. Pacific Conf. Computer Graphics and Applications*, pages 175–184, 2003.

[47] O. Wang, M. Lang, M. Frei, A. Hornung, A. Smolic, and M. Gross. Stereobrush: Interactive 2d to 3d conversion using discontinuous warps. In

Proc. Eurographics Symp. Sketch-Based Interfaces and Modeling, SBIM '11, pages 47–54, New York, NY, USA, 2011. ACM.

[48] S. Schar, H. Bieri, T. Killer, and X. Jiang. Introducing stereo effects into cel animations. In *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*, pages 353–356, 2008.

[49] J. Assa and L. Wolf. Diorama construction from a single image. *Comput. Graph. Forum*, 26(3):599–608, 2007.

[50] Jonathan Ventura, S. DiVerdi, and T. Höllerer. A sketch-based interface for photo pop-up. In *Proc. Eurographics Symp. Sketch-Based Interfaces and Modeling*, SBIM '09, pages 21–28, New York, NY, USA, 2009. ACM.

[51] Daniel Sýkora, David Sedlacek, Sun Jinchao, John Dingliana, and Steven Collins. Adding depth to cartoons using sparse depth (in)equalities. *Comput. Graph. Forum*, 29(2):615–623, 2010.

[52] Kent A. Stevens. The visual interpretation of surface contours. *Artificial Intelligence*, 17(1):47 – 73, 1981.

[53] J. J. Koenderink. What does the occluding contour tell us about solid shape? *Perception*, 13(3):321–330, 1984.

[54] D Knill. Perception of surface contours and surface shape: from computation to psychophysics. *J. Optical Society of America*, 9(9):1449–1464, 1992.

[55] Jitendra Malik. Interpreting line drawings of curved objects. *Intl. J. Comput. Vis.*, 1(1):73–103, 1987.

[56] Indranil Chakravarty. A generalized line and junction labeling scheme with application to scene analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1(2):202–205, 1979.

[57] Rudolf Arnheim. *Art and Visual Perception: A Psychology of the Creative Eye*. University of California Press, 1974.

[58] Agns Desolneux, Lionel Moisan, and Jean-Michel Morel. *From Gestalt Theory to Image Analysis: A Probabilistic Approach*. Springer, 1st edition, 2007.

[59] Yansheng Ming, Hongdong Li, and Xuming He. Connected contours: A new contour completion model that respects the closure effect. In *Proc. IEEE Conf. Comput. Vis. Pattern Rec. (CVPR)*, pages 829–836, 2012.

[60] G. Palou and P. Salembier. Monocular depth ordering using t-junctions and convexity occlusion cues. *IEEE Trans. Image Process.*, 22(5):1926–1939, 2013.

[61] Xueting Liu, Xiangyu Mao, Xuan Yang, Linling Zhang, and Tien-Tsin Wong. Stereoscopizing cel animations. *ACM Trans. Graph.*, 32(6):223:1–223:10, 2013.

[62] Chih-Kuo Yeh, P.K. Jayaraman, Xiaopei Liu, Chi-Wing Fu, and Tong-Yee Lee. 2.5D cartoon hair modeling and manipulation. *IEEE Trans. Vis. Comput. Graphics*, 21(3):304–314, 2015. ©2015 IEEE. Reprinted, with permission.

[63] Liangliang Nan, Andrei Sharf, Ke Xie, Tien-Tsin Wong, Oliver Deussen, Daniel Cohen-Or, and Baoquan Chen. Conjoining gestalt rules for abstraction of architectural drawings. *ACM Trans. Graph.*, 30(6):185:1–185:10, 2011.

[64] Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. Closure-aware sketch simplification. *ACM Trans. Graph.*, 34(6):168:1–168:10, 2015.

[65] Boaz J. Super and Alan C. Bovik. Shape from texture using local spectral moments. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(4):333–343, 1995.

[66] BertholdK.P. Horn. Height and gradient from shading. *Intl. J. Comput. Vis.*, 5(1):37–75, 1990.

[67] Tai-Pang Wu, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Interactive normal reconstruction from a single image. *ACM Trans. Graph.*, 27(5):119:1–119:9, 2008.

[68] S.K. Nayar and Y. Nakagawa. Shape from Focus. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(8):824–831, 1994.

[69] N. Apostoloff and A. Fitzgibbon. Learning spatiotemporal t-junctions for occlusion detection. In *Proc. IEEE Conf. Comput. Vis. Pattern Rec. (CVPR)*, volume 2, pages 553–559, 2005.

[70] Mariella Dimiccoli and Philippe Salembier. Exploiting t-junctions for depth segregation in single images. In *IEEE Intl. Conf. Acoustics, Speech, and Signal Processing*, 2009.

[71] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. Learning depth from single monocular images. In Y. Weiss, B. Schölkopf, and J.C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1161–1168. MIT Press, 2006.

[72] Zhaoyin Jia, A. Gallagher, Yao-Jen Chang, and Tsuhan Chen. A learning-based framework for depth ordering. In *IEEE Conf. on Comp. Vision and Pattern Recognition*, pages 294–301, 2012.

[73] Yingze Wang, Yu Chen, Jianzhuang Liu, and Xiaoou Tang. 3D reconstruction of curved objects from single 2D line drawings. In *Proc. IEEE Conf. Comput. Vis. Pattern Rec. (CVPR)*, pages 1834–1841, 2009.

[74] Tianfan Xue, Jianzhuang Liu, and Xiaoou Tang. Example-based 3d object reconstruction from line drawings. In *Proc. IEEE Conf. Comput. Vis. Pattern Rec. (CVPR)*, pages 302–309, 2012.

[75] Pushkar Joshi and Nathan A. Carr. Repoussé: Automatic Inflation of 2D Artwork. In Christine Alvarado and Marie-Paule Cani, editors, *Eurographics Workshop on Sketch-Based Interfaces and Modeling*. The Eurographics Association, 2008.

[76] James Andrews, Pushkar Joshi, and Nathan Carr. A linear variational system for modelling from curves. *Comput. Graph. Forum*, 30(6):1850–1861, 2011.

[77] Jooyoung Hahn, Jie Qiu, Eiji Sugisaki, Lei Jia, Xue-Cheng Tai, and Hock Soon Seah. Stroke-based surface reconstruction. *Numerical Mathematics: Theory, Methods and Applications*, 6(01):297–324, 2013.

[78] Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. *ACM Trans. Graph.*, 33(2):16, 2014.

[79] Olga A. Karpenko and John F. Hughes. Smoothsketch: 3D free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, 2006.

[80] Scott F. Johnston. Lumo: Illumination for cel animation. In *Proc. Intl. Symp. Non-photorealistic Animation and Rendering*, NPAR '02, pages 45–ff, New York, NY, USA, 2002. ACM.

[81] Tai-Pang Wu, Chi-Keung Tang, Michael S. Brown, and Heung-Yeung Shum. ShapePalettes: Interactive normal transfer via sketching. *ACM Trans. Graph.*, 26(3), 2007.

[82] Q. Xu, Y. Gingold, and K. Singh. Inverse toon shading: Interactive normal field modeling with isophotes. In *Proc. Workshop on Sketch-Based Interfaces and Modeling*, SBIM '15, pages 15–25, Aire-la-Ville, Switzerland, Switzerland, 2015. Eurographics Association.

[83] Minh Tuan Bui, Junho Kim, and Yunjin Lee. 3D-look shading from contours and hatching strokes. *Computers & Graph.*, 51(C):167–176, 2015.

[84] Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. CrossShade: Shading concept sketches using cross-section curves. *ACM Trans. Graph.*, 31(4):45:1–45:11, 2012.

[85] Emmanuel Iarussi, David Bommes, and Adrien Bousseau. BendFields: Regularized curvature fields from rough concept sketches. *ACM Trans. Graph.*, 34(3):24:1–24:16, 2015.

[86] Alec Rivers, Takeo Igarashi, and Frédo Durand. 2.5D cartoon models. *ACM Trans. Graph.*, 29:59:1–59:7, 2010.

[87] James McCann and Nancy Pollard. Local layering. *ACM Trans. Graph.*, 28(3):84:1–84:7, 2009.

[88] Chih-Kuo Yeh, Song Peng, Lin Peng-Yen, Chi-Wing Fu, Lin Chao-Hung, and Lee Tong-Yee. Double-sided 2.5D graphics. *IEEE Trans. Vis. Comput. Graphics*, 19(2):225–235, 2013.

[89] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):583–598, 1991.

[90] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2274–2282, 2012.

[91] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Intl. J. Comput. Vis.*, 59(2):167–181, 2004.

[92] Daniel Sýkora, Jan Buriánek, and Jiří Žára. Segmentation of black-and-white cartoons. In *Proc. Spring Conf. Computer Graphics*, pages 233–230, 2003.

[93] Song-Hai Zhang, Tao Chen, Yi-Fei Zhang, Shi-Min Hu, and Ralph R. Martin. Vectorizing cartoon animations. *IEEE Trans. Vis. Comput. Graphics*, 15(4):618–629, 2009.

[94] Jian Sun, Lu Yuan, Jiaya Jia, and Heung-Yeung Shum. Image completion with structure propagation. *ACM Trans. Graph.*, 24(3):861–868, 2005.

[95] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Proc. Intl. Conf. Comp. Vis. (ICCV)*, volume 2, pages 1033–1038, 1999.

[96] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proc. SIGGRAPH*, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[97] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher. Simultaneous structure and texture image inpainting. In *Proc. IEEE Conf. Comput. Vis. Pattern Rec. (CVPR)*, volume 2, pages II–707–12, 2003.

[98] Antonio Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Trans. Image Processing*, 13(9):1200–1212, 2004.

[99] Lei Zhang, Hua Huang, and Hongbo Fu. EXCOL: An EXtract-and-COmplete Layering Approach to Cartoon Animation Reusing. *IEEE Trans. Vis. Comput. Graphics*, 18(7):1156–1169, 2012.

[100] D. Geiger, Hsingkuo Pao, and N. Rubin. Salient and multiple illusory surfaces. In *Proc. IEEE Conf. Comput. Vis. Pattern Rec. (CVPR)*, pages 118–124, 1998.

[101] Pradeep Kumar Jayaraman and Chi-Wing Fu. Interactive line drawing recognition and vectorization with commodity camera. In *Proc. 22nd ACM Intl. Conf. Multimedia*, MM '14, pages 447–456. ACM, 2014.

[102] Google. Google Glass. `http://www.google.com/glass/start/`, 2013. [Online; accessed on 03-Feb-2017].

[103] Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27(7):773 – 780, 2006.

[104] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Proc. Euro. Conf. Comput. Vis. (ECCV)*, volume 1, pages 430–443, 2006.

[105] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. Intl. Joint Conf. Artificial Intelligence*, volume 2, pages 674–679, 1981.

[106] David Marr and E. Hildreth. Theory of edge detection. *Proc. Royal Society of London Series B*, 207(1167):187–217, 1980.

[107] S.L. Phung, A. Bouzerdoum, and Sr. Chai, D. Skin segmentation using color pixel classification: analysis and comparison. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(1):148–154, 2005.

[108] C. Li and K. M. Kitani. Pixel-level hand detection in ego-centric videos. In *Proc. IEEE Conf. Comput. Vis. Pattern Rec. (CVPR)*, pages 3570–3577, 2013.

[109] Gary Bradski. OpenCV. *Dr. Dobb's Journal of Software Tools*, 2000.

[110] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

[111] Jianchao Tan, Marek Dvorožňák, Daniel Sýkora, and Yotam Gingold. Decomposing time-lapse paintings into layers. *ACM Trans. Graph.*, 34(4):61:1–61:10, 2015.

[112] A.K. Macworth. Interpreting pictures of polyhedral scenes. *Artificial Intelligence*, 4(2):121 – 137, 1973.

[113] Kokichi Sugihara. Mathematical structures of line drawings of polyhedrons-toward man-machine communication by means of line drawings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4(5):458–469, 1982.

[114] H Lipson and M Shpitalni. Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Computer-Aided Design*, 28(8):651 – 663, 1996.

[115] P. A. C. Varley and R. R. Martin. Estimating depth from line drawing. In *Proc. ACM Symp. Solid Modeling and Applications*, SMA '02, pages 180–191, New York, NY, USA, 2002. ACM.

[116] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. In *Proc. SIGGRAPH*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[117] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. FiberMesh: Designing freeform surfaces with 3D curves. *ACM Trans. Graph.*, 26(3), 2007.

[118] Yotam Gingold, Takeo Igarashi, and Denis Zorin. Structured annotations for 2D-to-3D modeling. *ACM Trans. Graph.*, 28(5):148:1–148:9, 2009.

[119] getty. `getty.deviantart.com/art/Where-are-we-25302128`. [Online; accessed 03-February-2017].

[120] G. Kanizsa, W. Gerbino, and M. Henle. Convexity and symmetry in figure-ground organization. vision and artifact. *Vision and Artifact, Springer*, page 2532, 1976.

[121] Hsing-Kuo Pao, Davi Geiger, and Nava Rubin. Measuring convexity for figure/ground separation. In *Proc. Intl. Conf. Comp. Vis. (ICCV)*, volume 2, pages 948–955, 1999.

[122] S. Ullman. Filling-in the gaps: The shape of subjective contours and a model for their generation. *Biological Cybernetics*, 25(1):1–6, 1976.

[123] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):850–863, 1993.

[124] Thomas Eiter and Heikki Mannila. Computing Discrete Fréchet Distance. Technical report, Technische Universität Wien, 1994.

[125] Paul T. Boggs and Jon W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.

[126] Robert T. Frankot and Rama Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(4):439–451, 1988.

[127] Amit Agrawal, Rama Chellappa, and Ramesh Raskar. An algebraic approach to surface reconstruction from gradient fields. In *Proc. Intl. Conf. Comp. Vis. (ICCV)*, volume 1, pages 174–181. IEEE, 2005.

[128] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proc. Symp. Geometry Processing*, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[129] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry*, volume 1148 of *Lecture Notes in Comp. Sci.*, pages 203–222. Springer-Verlag, 1996.

[130] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. SIGGRAPH*, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[131] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. *Visualization and Mathematics III*, chapter Discrete Differential-Geometry Operators for Triangulated 2-Manifolds, pages 35–57. 2003.

[132] Fernando do Goes, Mathieu Desbrun, and Yiying Tong. Vector field processing on triangle meshes. In *SIGGRAPH Asia 2015 Courses*, SA '15, pages 17:1–17:48. ACM, 2015.

[133] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph.*, 32(5):152:1–152:11, 2013.

[134] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Trans. Sys. Man, and Cybernetics*, 9(1):62–66, 1979.

[135] Zhouchen Lin, Lifeng Wang, Yunbo Wang, S.B. Kang, and Tian Fang. High resolution animated scenes from stills. *IEEE Trans. Vis. Comput. Graphics*, 13(3):562–568, 2007.

[136] Xuemiao Xu, Liang Wan, Xiaopei Liu, Tien-Tsin Wong, Liansheng Wang, and Chi-Sing Leung. Animating animal motion from still. *ACM Trans. Graphics (SIGGRAPH Asia 2008)*, 27(5):117:1–117:8, 2008.

[137] Makoto Okabe, Ken Anjyo, Takeo Igarashi, and Hans-Peter Seidel. Animating pictures of fluid using video examples. *Comput. Graph. Forum*, 28(2):677–686, 2009.

[138] Menglei Chai, Lvdi Wang, Yanlin Weng, Xiaogang Jin, and Kun Zhou. Dynamic hair manipulation in images and videos. *ACM Trans. Graph.*, 32(4):75:1–75:8, 2013.

[139] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: Intl. J. Geographic Information and Geovisualization*, 10(2):112–122, 1973.

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

[141] Chenyang Xu, Dzung L Pham, and Jerry L Prince. Image segmentation using deformable models. *Handbook of medical imaging*, 2:129–174, 2000.

[142] Dazhi Yu, Renwei Mei, Li-Shi Luo, and Wei Shyy. Viscous flow computations with the method of lattice boltzmann equation. *Progress in Aerospace Sciences*, 39(5):329 – 367, 2003.

[143] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3D characters. *ACM Trans. Graph.*, 26(3):72:1–72:8, 2007.

[144] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141, 2005.

[145] C. K. Yeh, S. Y. Huang, P. K. Jayaraman, C. W. Fu, and T. Y. Lee. Interactive high-relief reconstruction for organic and double-sided objects from a photo. *IEEE Trans. Vis. Comput. Graphics*, PP(99):1–1, 2016. ©2016 IEEE. Reprinted, with permission.

[146] Wikipedia. Relief. `http://en.wikipedia.org/wiki/Relief#High_relief`, 2013. [Online; accessed 17-March-2015].

[147] P. Cignoni, C. Montani, and R. Scopigno. Computer-assisted generation of bas- and high-reliefs. *J. Graph. Tools*, 2(3):15–28, 1997.

[148] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Automatic photo pop-up. *ACM Trans. Graph.*, 24(3):577–584, 2005.

[149] Qiong Zeng, Wenzheng Chen, Huan Wang, Changhe Tu, Daniel Cohen-Or, Dani Lischinski, and Baoquan Chen. Hallucinating stereoscopy from a single image. *Comput. Graph. Forum*, 34(2):1–12, 2015.

[150] Pushkar Joshi and Nathan A. Carr. Repoussé: Automatic inflation of 2D artwork. In *Eurographics Conf. Sketch-Based Interfaces and Modeling*, pages 49–55, 2008.

[151] Feilong Yan, Minglun Gong, Daniel Cohen-Or, Oliver Deussen, and Baoquan Chen. Flower reconstruction from a single photo. *Comput. Graph. Forum*, 33(2):439–447, 2014.

[152] A. Saxena, Min Sun, and A.Y. Ng. Make3D: Learning 3D scene structure from a single still image. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(5):824–840, 2009.

[153] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-sweep: extracting editable objects from a single photo. *ACM Trans. Graph.*, 32(6):195:1–10, 2013.

[154] Natasha Kholgade, Tomas Simon, Alexei Efros, and Yaser Sheikh. 3D object manipulation in a single photograph using stock 3D models. *ACM Trans. Graph.*, 33(4):127:1–12, 2014.

[155] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*. Springer Science & Business Media, 2003.

[156] Olga A. Karpenko and John F. Hughes. SmoothSketch: 3D free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, 2006.

[157] Antonio Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Trans. Image Processing*, 13(9):1200–1212, 2004.

[158] Maxime Daisy, David Tschumperlé, and Olivier Lézoray. A fast spatial patch blending algorithm for artefact reduction in pattern-based image inpainting. In *SIGGRAPH Asia 2013 Tech. Briefs*, pages 8:1–4, 2013.

[159] Yotam Gingold, Takeo Igarashi, and Denis Zorin. Structured annotations for 2D-to-3D modeling. *ACM Trans. Graph.*, 28(5):148:1–9, 2009.

[160] CWAM Van Overveld. Painting gradients: Free-form surface design using shading patterns. In *Graphics Interface*, pages 151–158, 1996.

[161] Tai-Pang Wu, Chi-Keung Tang, Michael S. Brown, and Heung-Yeung Shum. ShapePalettes: Interactive normal transfer via sketching. *ACM Trans. Graph.*, 26(3), 2007.

[162] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH*, pages 317–324, 1999.

[163] Robert Schneider and Leif Kobbelt. Geometric fairing of irregular meshes for free-form surface design. *Comput. Aided Geom. Des.*, 18(4):359–379, 2001.

[164] Intel RealSense Technology. `intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html`. [Online; accessed 03-February-2017].

[165] Intel RealSense SR300. `software.intel.com/en-us/RealSense/SR300Camera`. [Online; accessed 03-February-2017].

[166] Atheer AiR Glasses. `atheerair.com/air-glasses`. [Online; accessed 03-February-2017].

Author's Publications

- *Pradeep Kumar Jayaraman*, Chi-Wing Fu, Jianmin Zheng, Xueting Liu and Tien-Tsin Wong. "Globally Consistent Wrinkle-Aware Shading of Line Drawings," *under review* in IEEE Transactions on Visualization and Computer Graphics, 2017.
- Chih-Kuo Yeh, Shih-Yang Huang, *Pradeep Kumar Jayaraman*, Chi-Wing Fu, and Tong-Yee Lee. "Interactive High-relief Reconstruction of Organic and Double-sided Structures from a Photo," IEEE Transactions on Visualization and Computer Graphics, 2016.
- Chi-Wing Fu & Peng Song, Xiaoqi Yan, Lee Wei Yang, *Pradeep Kumar Jayaraman*, and Daniel Cohen-Or. "Computational Interlocking Furniture Assembly," ACM Transactions on Graphics (SIGGRAPH), 2015.
- Chih-Kuo Yeh, *Pradeep Kumar Jayaraman*, Xiaopei Liu, Chi-Wing Fu, and Tong-Yee Lee. "2.5D Cartoon Hair Modeling and Manipulation," IEEE Transactions on Visualization and Computer Graphics, 2015.
- *Pradeep Kumar Jayaraman* and Chi-Wing Fu. "Interactive Line Drawing Recognition and Vectorization with Commodity Camera," ACM Multimedia, 2014. *Awarded ACM Student Travel Grant.*