Digital Micrography

by

Ron Israel Maharik

B.Sc., Technion – Israel Institute of Technology, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

November 2011

© Ron Israel Maharik 2011
Abstract

This work is concerned with computer-assisted generation of text art. Specifically, we consider an artistic style known as micrography, in which complex images are constructed from lines of tiny, readable text. Traditionally created by trained scribes and requiring immense amounts of effort, micrography has in recent years crossed over into the digital domain, with modern artists adapting the style in various ways and using it to convey their own message. Unfortunately, creating micrography with standard digital graphic design techniques remains a tedious and frustrating exercise.

The method proposed here is capable of automating the work-intensive aspects of micrography, while still keeping the designer in control of the overall look and feel of the result. For artists, this means they can now rapidly experiment with different designs in a way that would be infeasible using traditional digital techniques.

We begin with an analysis of the aesthetic properties of text flow that make for a visually appealing micrography. Observing that some of these requirements are shared with well-studied problems in computer graphics such as parameterization and quadrangulation, we base our solution on the popular framework of smooth fields, augmenting it with a novel approach to boundary constraint design for planar shapes. The result is a system that lets artists generate beautiful micrography in a matter of minutes, automatically optimizing for aesthetic appeal and readability while also allowing for intuitive manual intervention.
Preface

The method presented in this thesis was described in the following research paper:


Figures and text from the publication are copyrighted by ACM (2011) and reused in this thesis by permission.

Chapter 8 describes an algorithm developed by Mikhail Bessmeltsev and Prof. Alla Sheffer. The text of the chapter appears in the publication above, and is reproduced here for the sake of completeness.

The algorithm in Section 6.3 was developed by Mikhail Bessmeltsev, Alla Sheffer and myself in collaboration.

Other ideas in this thesis were fleshed out in discussions with Alla Sheffer, Ariel Shamir at the Interdisciplinary Center Herzliya, and Nathan Carr at Adobe Systems. I carried out most of the background literature survey and all experiments and development.
# Table of Contents

Abstract ................................................................. ii
Preface ................................................................. iii
Table of Contents ................................................... iv
List of Tables ........................................................ vii
List of Figures ........................................................ viii
Acknowledgements ................................................... x

1 Introduction ......................................................... 1

2 Related Work ....................................................... 6
  2.1 Text Art .......................................................... 6
  2.2 Layout Tasks .................................................... 10
  2.3 Smooth Fields .................................................. 12

3 Text Layout Design ................................................ 15
  3.1 Aesthetics ......................................................... 15
     3.1.1 Alignment ................................................. 15
     3.1.2 Shading and Spacing .................................... 16
     3.1.3 Bounded Curvature ...................................... 17
     3.1.4 Distinction ................................................ 17
  3.2 Readability ...................................................... 17
     3.2.1 Curvature .................................................. 17
     3.2.2 Line Length and Word Breakup ......................... 20
     3.2.3 Coherence and Line Ordering ......................... 20
     3.2.4 Line Spacing ............................................. 21
## Table of Contents

4 System Overview ............................................ 22  
4.1 Algorithm ............................................. 22  
4.1.1 Input ............................................. 22  
4.1.2 Preprocessing ..................................... 22  
4.1.3 Smooth Field Design ................................. 22  
4.1.4 Streamline Tracing and Ordering .................. 24  
4.1.5 Text Placement .................................... 24  
4.2 Implementation ........................................ 24  
4.2.1 Configuration File ................................ 25  
4.2.2 Dependencies ..................................... 26  
4.2.3 Performance ....................................... 26  

5 Preprocessing ............................................. 27  
5.1 Image Preprocessing ................................... 27  
5.2 Text Preprocessing .................................... 30  

6 Smooth Field Design ..................................... 32  
6.1 Field Design Primer ................................... 32  
6.2 Field Design for Text Layout ......................... 35  
6.3 Boundary Condition Design ......................... 39  
6.3.1 Pairwise Preference Weights ....................... 42  
6.3.2 Individual Preference Weights ..................... 43  
6.3.3 Completing the Formulation ....................... 44  
6.3.4 Solving ........................................... 45  
6.4 Computing the Field ................................... 47  
6.5 User Control .......................................... 48  
6.6 Symmetry ............................................. 50  

7 Line Tracing ............................................... 52  
7.1 Goals and Requirements ............................... 52  
7.2 Tracing Algorithm ..................................... 53  
7.3 Deviations from the Field ............................. 55  
7.4 Spacing and Enrichment ............................... 56  

8 Line Orientation and Ordering ......................... 59  
8.1 Orientation .......................................... 59  
8.2 Ordering ............................................. 61
## Table of Contents

9 Rendering Text .................................................. 63  
  9.1 Typesetting ................................................. 63  
  9.2 Text Warping .............................................. 64  

10 Results .............................................................. 67  
  10.1 Running Times ........................................... 75  

11 Summary .............................................................. 81  

12 Future Work ....................................................... 82  

Bibliography .......................................................... 84
List of Tables

10.1 Run time breakdown . . . . . . . . . . . . . . . . . . . . . . . . 80
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Hand-drawn micrography (1)</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Hand-drawn micrography (2)</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Path-based text art</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Grid-aligned digital text art</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Other digital text art</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Flow visualization using streamlines</td>
<td>11</td>
</tr>
<tr>
<td>2.5</td>
<td>Field-guided mosaic pattern</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Boundary-aligned vs. unaligned layout</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Region distinction</td>
<td>18</td>
</tr>
<tr>
<td>3.3</td>
<td>Region distinction: hand-drawn example</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>Coherent vs. incoherent layout</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Algorithm overview</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>User interface</td>
<td>25</td>
</tr>
<tr>
<td>5.1</td>
<td>Planar map</td>
<td>27</td>
</tr>
<tr>
<td>5.2</td>
<td>Image preprocessing</td>
<td>29</td>
</tr>
<tr>
<td>6.1</td>
<td>Smooth field types</td>
<td>33</td>
</tr>
<tr>
<td>6.2</td>
<td>Singularities and text layout</td>
<td>34</td>
</tr>
<tr>
<td>6.3</td>
<td>Full boundary alignment</td>
<td>36</td>
</tr>
<tr>
<td>6.4</td>
<td>Narrow regions</td>
<td>37</td>
</tr>
<tr>
<td>6.5</td>
<td>Partial boundary alignment</td>
<td>38</td>
</tr>
<tr>
<td>6.6</td>
<td>Pairwise alignment preferences</td>
<td>40</td>
</tr>
<tr>
<td>6.7</td>
<td>Narrow features</td>
<td>40</td>
</tr>
<tr>
<td>6.8</td>
<td>Boundary condition formulation</td>
<td>41</td>
</tr>
<tr>
<td>6.9</td>
<td>Feature size estimation</td>
<td>43</td>
</tr>
<tr>
<td>6.10</td>
<td>Boundary condition solution</td>
<td>45</td>
</tr>
<tr>
<td>6.11</td>
<td>Boundary solver procedure</td>
<td>46</td>
</tr>
<tr>
<td>6.12</td>
<td>Alignment/coherence trade-off</td>
<td>49</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.13</td>
<td>Forcing symmetry</td>
<td>51</td>
</tr>
<tr>
<td>7.1</td>
<td>Greedy tracing algorithm</td>
<td>54</td>
</tr>
<tr>
<td>7.2</td>
<td>Elements of our tracing method</td>
<td>57</td>
</tr>
<tr>
<td>8.1</td>
<td>Orientation and ordering formulation</td>
<td>60</td>
</tr>
<tr>
<td>9.1</td>
<td>Text warping example</td>
<td>64</td>
</tr>
<tr>
<td>9.2</td>
<td>Line end warping</td>
<td>66</td>
</tr>
<tr>
<td>10.1</td>
<td>Microgram: “Motif”</td>
<td>68</td>
</tr>
<tr>
<td>10.2</td>
<td>Microgram: “Summer Girl”</td>
<td>70</td>
</tr>
<tr>
<td>10.3</td>
<td>Microgram: “In Flanders Fields”</td>
<td>71</td>
</tr>
<tr>
<td>10.4</td>
<td>Microgram: “Mona Lisa”</td>
<td>72</td>
</tr>
<tr>
<td>10.5</td>
<td>Microgram: “Light and Shadow”</td>
<td>73</td>
</tr>
<tr>
<td>10.6</td>
<td>Microgram: “Fedora”</td>
<td>74</td>
</tr>
<tr>
<td>10.7</td>
<td>Microgram: “Dark Eyes”</td>
<td>75</td>
</tr>
<tr>
<td>10.8</td>
<td>Two cats: comparison with Li et al.</td>
<td>76</td>
</tr>
<tr>
<td>10.9</td>
<td>Variations: text length</td>
<td>77</td>
</tr>
<tr>
<td>10.10</td>
<td>Variations: typeface and colour</td>
<td>78</td>
</tr>
<tr>
<td>10.11</td>
<td>Field design with user constraints</td>
<td>79</td>
</tr>
</tbody>
</table>
Acknowledgements

I owe a debt of gratitude to my supervisor, Prof. Alla Sheffer, for teaching me how to look critically at images and how to undertake research work in computer graphics. Her drive, and her endless flow of ideas, helped shape this work at every turn. I am further grateful to the rest of the project team: Prof. Ariel Shamir, Nathan Carr, and Mikhail Bessmeltsev, all of whom were deeply involved and invested in this work and helped make it what it is. Many thanks also to my second reader Prof. Tamara Munzner for her valuable feedback on this manuscript.

I am indebted to everyone at Adobe’s Advanced Technology Labs for hosting me in San Jose, for their enthusiasm about this work and for providing the resources and know-how that made it possible. It was an honor to be part of your team. Thanks also to the Canadian agencies that funded my work: the Natural Sciences and Engineering Research Council of Canada, GRAND NCE, and MITACS/Mprime Network. Many thanks to all the artists whose creations are used here, either as source material or as a source of inspiration.

I am ever grateful to my parents and grandparents, my siblings, and the extended family in Israel for their love and support, and to Dana, my partner in this adventure and all others.

Finally, I am thankful to everyone who shared their wisdom and experience with me over the past two years, offered words of encouragement and kindness, and provided some much-needed balance: my colleagues and friends in Imager, CS department faculty and staff, the CSGSA, and all my friends in Vancouver. Thank you for being part of this amazing experience.

Ron Maharik
The University of British Columbia
November 2011
Chapter 1

Introduction

This work is concerned with text art. The idea of using the form and arrangement of letters and words to convey an artistic message is perhaps as old as written language itself. Calligraphy, literally “beautiful writing”, has been in practice in Asia for thousands of years [52]. According to British book designer Thomas James Cobden-Sanderson, “The whole duty of Typography, as of Calligraphy, is to communicate to the imagination, without loss by the way, the thought or image intended to be communicated by the Author”.

A calligram is a specific form of text art whereby words or letters are arranged so as to create a visual image. Calligrams enjoy a rich tradition and come in a wide variety of styles, limited only by the artist’s imagination. In this work we focus on a genre of calligram known alternatively as micrography or microcalligraphy. Practised by Jewish artists since the Middle Ages [7], micrography utilizes minute letters to provide a unique interplay between textual content and image – presenting a story or poem at the small scale, while forming an image when viewed as a whole. This gap in scale between lettering and image is a defining characteristic of micrographic works, or micrograms, distinguishing them from other types of calligram.

Despite the small scale of the lettering, micrography places much emphasis on readability. Not only should individual letters be neat and legible, but the text as a whole must be arranged in a way that allows readers to easily follow its progression. Thus a close correspondence is established between the microgram and its source text – a correspondence that plays a crucial part in the artistic appeal of the work, even for observers who would not follow the text from start to finish. This is especially true when the text is a recognizable piece such as scripture or a popular poem.

Different styles exist in traditional micrography, ranging from simple ornamentation to fully painted scenes with colour and shading effects. A common theme in all styles is the treatment of individual text lines as brush or pencil strokes. Hence, classic illustration techniques such as line-art or hatching can be applied in the creation of micrography, and aesthetic conventions that govern illustration apply to micrograms as well. First and
Figure 1.1: Examples of hand-drawn micrography, by (clockwise from top left) G. Apollinaire, G. Klopstock (gilklop.co.il/apage/35286.php), and J. Sibal (jonsibal.com). Used with permission.
foremost among these is the practice of aligning strokes with boundaries and other salient features. The three hand-drawn examples in Figure 1.1 demonstrate effective use of text lines as strokes to convey shape and structure.

Wonderful examples of micrography have been created by artists over the centuries ([3, 7]), and contemporary artists are continuing this tradition today ([50]). However, the popularity of micrography has always been limited by the sheer amount of effort required to create a single work. Traditional artists, working without the benefit of digital tools, must legibly scribe every letter at the desired minuscule resolution – a process that can take weeks to complete and leaves little room for error or for experimentation with different designs. An artist will generally use a combination of rough underlying sketch and guiding lines to design the layout, then fill in the lettering.

With the advent of digital creative software, the barriers for practitioners of micrography are lower than in the past. Digital artists have in recent years adopted the form and are rapidly expanding its frontier – developing new
styles, importing nontraditional source material, and using the medium to promote commercial products or political movements [50]. However, even with digital graphic design tools, microgram creation remains a very tedious process. To produce an appealing design the artist must manually lay out and stylize individual sentences, words, even specific letters. As a result, many designs are either limited in size or rely on very simple text flow patterns, such as all-horizontal layout (Figure 3.1 left).

In this work we present a method for creating digital micrograms starting from a given text and an appropriate vector-graphics image. Our goal is to automate the highly technical and time-consuming aspects of microgram creation, while providing high level tools for user control. This frees the artist to efficiently experiment with multiple designs and settings to achieve the intended result. The micrograms created using our system are easy to read, effectively convey the input image, and are visually appealing.

Not every image is suitable for creating micrography. Micrograms usually consist of several contiguous regions of text conveying large image components, as well as individual text lines representing isolated line art. Some works also contain non-textual elements (Figure 1.1, top right) enhancing the visual appeal of the image. Hence, the natural input to our system is vector graphics images, where such regions and lines are well-defined. More complex images can be simplified and converted to vector representation using standard tools, e.g. Adobe Illustrator [1]. In this work we deal exclusively with vector representations, which are practically resolution independent. Reproducing such vector art on physical media is a separate topic with its own literature [25, 33, 41].

In approaching the problem, we begin by observing that the primary challenge of micrography lies in designing a suitable text layout – an arrangement of paths along which the text will be placed. We examine the criteria for a good layout, taking into account aesthetic and readability requirements. Next, we show how to construct text layouts over a set of closed regions using smooth fields, a mathematical tool that has been applied extensively and successfully in recent years to a variety of related problems.

Given a suitable field per region, we use a custom algorithm to trace streamlines through the regions and obtain a set of paths for text placement. We then compute an ordering of these streamlines, optimizing for natural line-to-line progression. Finally, we render text along the ordered set of streamlines, adaptively scaling and warping as necessary to form clearly delineated, evenly shaded regions.

Our system can provide a fully automatic solution for a given input text and vector image. For greater expressiveness, it also provides a user interface
Chapter 1. Introduction

giving artists more control over the layout. This includes local directional preferences, font face and size customization, and additional controls over various artistic effects. By implementing the system as a plug-in for Adobe Illustrator we provide artists with a full-featured graphic design tool for creating intricate, appealing digital micrography.
Chapter 2

Related Work

2.1 Text Art

The visual appeal of artistic expression using text, as well as its labour-intensive nature, are widely recognized. Technologies for placing text along user-specified trajectories have been well explored and are a key component of several commercial design tools such as Adobe Illustrator [1]. These packages usually deal with text on a line-by-line basis: The user specifies the text to be placed, designs a path for the text to follow, and may include additional deformation attributes.

Illustrator implements a modified version of the Skeletal Strokes approach proposed by Hsu et al. [21]. A skeletal stroke is an image defined in relation to a reference line (the skeleton) and a reference width. When the skeleton is deformed, the image deforms accordingly in an intuitive manner. The original method supports a variety of deformation options, which Illustrator augments with locally-defined width profiles and folding avoidance [6].

Text Sculpting [44, 47] provides a further degree of freedom in deformation by replacing the skeleton with a parametric surface $S(u,v)$, constructing the layout as a symbolic combination of this surface and the given text string, which is represented as a set of Bézier curves. Text sculpting also adds explicit support for animation.

Example designs using skeletal strokes and text sculpting are shown in Figure 2.1. While our method uses skeletal strokes as a building block, single-line text placement alone is not sufficient for creating sophisticated micrography. Specifying dozens and often hundreds of lines to convey a complex image would still be a daunting task, and exploration of the design space essentially requires repeating the entire exercise from scratch.

Another group of text art methods focuses on generating specific forms of text art on a larger scale. Xu et al. [59] show how to create structure-based ASCII art by optimizing a specially designed shape similarity metric. ASCII art aims to approximate a given shape using a limited set of symbols under rigid rules for alignment and position, but it does not attempt to
2.1. Text Art

Figure 2.1: Path-based text art. Top: adaptive width profiles using Adobe Illustrator CS5. Bottom: adaptive deformation with SculpText.

place meaningful text in a legible form. The calligraphic packing work of Xu and Kaplan [56] packs and warps a word or short phrase to fit a given shape. The resulting macroscopic calligrams are certainly engaging, but the technique relies heavily on individual letter arrangement by the user, and therefore does not scale well to the case of micrography.

A number of software packages such as Textorizer [16] and Textaizer [4] produce readable text in the form of grid-aligned text-mosaics (Figure 2.2). This approach lacks an explicit notion of shape, relying only on local colour and shading to depict the given image. In contrast, micrography draws from a much broader artistic palette, using shape at both the macro image-level and the micro font-level. Our system aims at delivering this greater degree of freedom. Textorizer can also generate images using a stippling technique with a given set of words as brush strokes (Figure 2.3), but this method does not produce readable text passages.

Finally, tag clouds such as those generated by Wordle ([14], Figure 2.3)
2.1. Text Art

Figure 2.2: Examples of grid-aligned digital text art tools. Top: Textorizer mode 2. Bottom: Textaizer.
2.1. Text Art

Figure 2.3: Examples of digital text art tools. Top: Textorizer mode 1. Bottom: Wordle.
2.2 Layout Tasks

Our goal of placing text inside a set of regions is related to a number of well-studied problems that can be loosely grouped as layout tasks. These include space-filling curves, field visualization using streamlines, maze generation, ornamental patterns, and hatching.

The criteria for designing ornamental patterns, as described by Wong et al. [55], are quite similar to those for laying out text. A given shape boundary must be adhered to, and there is often the expectation of a deeper interplay between the fill and the boundary, such as tracking a medial representation of the shape. The given space should be filled in a balanced manner with uniform density, while also preserving the aesthetic qualities and topological characteristics of the pattern used for the fill. Wong et al. propose a greedy layout method that finds large regions of empty space and fills them first, then gradually expands to fill gaps of diminishing size. Although this approach can successfully mimic natural plant growth patterns in certain cases, its applicability to other pattern types is limited. Their work also lacks support for a global layout strategy, as they point out in their future work section. Our work introduces such a global strategy in the context of text layout.

Computer generated mazes and labyrinths pack evenly-spaced paths into a given region. Pedersen and Singh [35] evolve their “organic” mazes from basic user-defined curves using a biologically inspired iterative algorithm. Xu and Kaplan [57] combine the organic approach with other maze types such as spiral and directional mazes. Both methods segment the underlying input into regions and use differences in pattern to make these regions distinct. Our system achieves a similar distinction effect through the use of text directions and font characteristics. A key difference in design goal between maze art and micrography is that maze generators actually encourage the resulting paths to wind and bend to increase the complexity of the maze. In contrast we are interested in generating coherent, readable flows, and work to minimize path curvature where possible.

Hatching and other non-photorealistic rendering methods use strokes to convey the shape of an input model, typically a manifold surface in 3D.
2.2. Layout Tasks

Figure 2.4: Example of flow visualization with evenly-spaced streamlines. Original image by Crowsnest at en.wikipedia.

Praun et al. [36] support arbitrary hatching styles in real time by pre-rendering a set of textures corresponding to different shades. To orient the strokes they rely on either a surface parameterization, a user-guided direction field, or principal curvature directions. Surazhsky and Elber [45] take this idea a step closer to our goal by replacing hatching strokes with lines of text. Specifically, they generalize the Text Sculpting method (Section 2.1) to B-spline surfaces in 3D space and multiple text lines. The result, while quite appealing, is not equivalent to 2D micrography and does not fulfill the readability requirement. From any single point of view some areas will contain easily readable text, but other portions are either flipped, excessively sheared, or occluded entirely. Their result is also highly dependent on surface parameterization, whose definition is external to their layout algorithm.

A final technique in this discussion is flow visualization with evenly-spaced streamlines. A streamline is a curve that is tangential to an underlying flow field at every point. Streamlines provide a natural way to visualize flow patterns in two or three dimensions. As an example, Figure 2.4 demonstrates the use of streamlines to visualize air flow around the wing of an aircraft. Two primary considerations in 2D streamline visualization are: avoidance of short lines where possible, and maintaining relatively uniform line density throughout the region of interest (equivalently, maintain roughly even spacing between adjacent lines) [30]. Note the similarities to our goal of filling a region with lines of text – this motivates our use of streamline placement as a proxy during the text layout process. Turk and Banks [49] proposed an image-guided technique that produces pleasing streamline patterns, but their iterative optimization approach was not fast
2.3 Smooth Fields

The use of smooth fields as a design tool has exploded in recent years, thanks to improved computational methods and a number of novel applications. A brief introduction to fields in the context of geometry processing is given in Section 6.1. Xu et al. [58] use a vector field to synthesize anisotropic textures on manifolds in a manner that respects feature lines and user constraints. They argue that aligning with salient features causes the texture to reinforce the underlying structure, rather than being at odds with it. The same argument motivates our approach to field-guided layout of text. Li et al. [26] successfully combine vector or tensor fields with shape grammars to synthesize geometry on manifolds. They also demonstrate anisotropic mosaic generation in 2D, guided by a tensor field which they align with manually selected directions approximating an image gradient (Figure 2.5).

Two closely related applications that often rely on smooth fields are uv parameterization and quadrangulation. Ray et al. [37] use two orthogonal vector fields, typically derived from principal curvature directions, to construct globally smooth parameterizations of triangle meshes. Bommes et al. [8] compute a parameterization based on a smooth feature-aligned cross-field. They employ a mixed-integer solver to constrain all singular points to lie at integer locations, which allows for straightforward conversion of the resulting parameterization into a coarse all-quad mesh.

The maze-art method of Xu and Kaplan [57] and hatching algorithm of Praun et al. [36], both of which were discussed in Section 2.2, also use smooth fields to guide flow and improve alignment with region boundaries or feature lines.

Field design is the process of constructing a smooth field suitable for a particular application. A number of design schemes have been proposed, most of which fall into one of two categories: topological or geometric [38]. Topological methods are concerned primarily with singular points in the field – where to place them, what types are allowed, and how to reduce their occurrence, since singularities almost always give rise to undesirable
2.3. Smooth Fields

Figure 2.5: Field-guided mosaic pattern by Li et al. [26]. The field is manually aligned with salient edges in the image. Reprinted with permission of IEEE.

artifacts. Geometric methods, on the other hand, focus on relating the field to the shape over which it is defined.

Ray et al. [39] and Crane et al. [12] take the topological route, constructing smooth direction fields given a user-defined set of singularities. This approach can generate fields with the absolute minimum number of singularities, but often at the cost of increased field curvature.

In contrast, Fisher et al. [15] and Bommes et al. [8] start with a set of user-defined local directional constraints, and let the singularities arise automatically. Fisher et al. use weighted least squares to minimize a quadratic energy function that accounts for smoothness as well as user constraints, while Bommes et al. develop a mixed integer problem that interpolates user-defined directions while maximizing smoothness.

Finally, Ray et al. [38] propose a geometric method based on a novel direction field formulation that lets the user adjust the trade-off between curvature (smoothness) and singularity count. The fact that curvature, singularities, and feature alignment are at odds with each other, and must be balanced according to the needs of the application, will be at the heart of our discussion of field design in Chapter 6.
An aspect of these works that is of particular interest to us is their treatment of boundaries. Three popular approaches for incorporating boundaries into the field design process are to leave them unconstrained (free), treat them as feature lines, or manually assign directions along the boundary according to some external criterion.

In applications that focus on manifold surfaces in 3D, boundaries in non-closed manifolds have both topological and geometric implications. The topological method of Crane et al. accounts for boundaries by modifying the algorithm to produce a free-boundary effect, which they refer to as “natural boundary behaviour”. An alternative solution is mentioned which increases the turning number of the field along the boundary, but that approach is not motivated or demonstrated. Ray et al. [39] rely heavily on boundaries in their topological analysis of N-symmetry direction fields: they analyze the behaviour at singular points by modelling them as infinitesimal “holes” surrounded by boundaries. Actual boundaries in open manifolds are handled by temporarily patching the holes and constraining the turning number along the boundary similarly to Crane, with similar results. The method of Fisher et al. supports either free boundaries or arbitrary user-defined directions.

All of the above also applies in the special case of planar shapes in 2D. Yet here boundaries play a key role, whereas in many 3D applications they are almost an afterthought. This is because boundaries in 2D are often the most prominent or even sole existing type of salient feature. One consequence is that default approaches to dealing with boundaries – treat all as feature lines or leave all unconstrained – often fail to achieve the desired result, forcing users to intervene manually in boundary condition design. In this thesis we present a principled approach to boundary design for smooth fields in 2D. While our method is carefully optimized for the purpose of text layout, the underlying principles can be generalized to other anisotropic layout scenarios, enabling automatic generation of results that would otherwise require significant manual effort.
Chapter 3

Text Layout Design

In this chapter we establish the characteristics of effective text layout for micrography.

Layout is the primary device for conveying the structure and aesthetics of the source image in works of micrography [7]. This is especially true for works rendered in black-and-white – a common practice in traditional as well as modern pieces – but holds true in the presence of colour or tone as well. In micrographic depiction of a given image, text lines are used in the manner of brush strokes, hatching strokes, or lines in sketching or painting. In Section 2.1 we discussed text-art styles in which layout plays no part in image depiction, but such works are normally not considered examples of micrography.

In addition to its contribution to image depiction, layout plays a key role in making a microgram readable. Although some readability choices are independent of layout (e.g. font face), a readable microgram cannot be produced given a bad layout.

We therefore see text layout as the main challenge in computer-generated micrography. The dual nature of micrograms dictates that our layout must satisfy aesthetic and image recognition requirements on the one hand, and text readability on the other. This set of requirements can often be contradictory, and consists of global as well as local considerations. The remainder of this chapter investigates each requirement in more detail.

3.1 Aesthetics

3.1.1 Alignment

To properly depict boundaries and other salient curves in the given image, text flow should align with these curves when possible. That is, the text line immediately adjacent to the curve should ideally run parallel to the curve at every point. This is an almost universal convention in sketching and painting, and is consistent with the manner in which the human visual system perceives edges in a scene [10]. Figure 3.1 compares a simple all-horizontal
3.1. Aesthetics

Figure 3.1: The all-horizontal configuration on the left does not depict the shape well, suffers from jagged aliasing artifacts, and creates very short lines in thin features. The layout on the left is aligned with shape boundaries and better conveys the shape, also avoiding short lines and aliasing. Reprinted with permission of ACM.

layout, which does not follow the convention (left), with a layout that does (right). The former suffers from jagged aliasing artifacts, and is at odds with the natural flow directions of the shape. The latter produces a more appealing, natural-looking microgram. If we focus on the all-horizontal result on the left, we can make another observation: from a shape depiction point of view, text flow perpendicular to the curve is an adequate substitute for aligned flow. Aliasing artifacts are most evident when the angle between boundary and text is small, and they gradually diminish as the angle approaches 90 degrees.

3.1.2 Shading and Spacing

Evenly shaded regions are common in source images used in micrography (Chapter 1). Text layout influences how such regions will be perceived by an observer. The simplest and most common method for even shading with text is to generate text lines at roughly even baseline-to-baseline spacing, then use a constant font size throughout the region (perhaps with minor local adaptations) to achieve a uniform look. The font size is generally set so that
3.2 Readability

leftover space between lines is kept to a minimum, although retaining a bit of space can improve readability [54]. In modern works artists sometimes add an element of texture by varying the text size within a region. This approach was used to great effect for rendering the text in Figure 1.1 (bottom).

3.1.3 Bounded Curvature

Aesthetic as well as readability considerations dictate that excessive bending along any given text line should be avoided. On the aesthetic side, humans are not accustomed to viewing sharply curving text lines, and may find such designs uncomfortable or distracting. Another consideration is visual compatibility with brush strokes and hatching strokes, both of which rarely exhibit high curvature.

3.1.4 Distinction

A boundary between adjacent regions presents a potential problem when constructing images from lines of text. Proper image depiction requires that the boundary be easily discernible in the resulting microgram. However, if we follow the alignment convention discussed in Section 3.1.1 then the text would flow in the same direction on either side of the boundary, obscuring the boundary (Figure 3.2 middle). Variations in colour or tone between the two regions will solve the problem, as will differences in font face, size or density. Another approach, popular in micrography, is to create distinction using text directions: rather than aligning with the boundary, the text flow on one side is designed to approach the boundary at an angle (Figure 3.2 bottom). This layout-based solution for region distinction can be seen in the handcrafted image in Figure 3.3, which also demonstrates another similar technique: trace one or two text lines along the boundary, and have the text flow toward the boundary at an angle on either side.

3.2 Readability

3.2.1 Curvature

Human observers are accustomed to reading text laid out in straight lines – be they left-to-right, right-to-left, or top-to-bottom depending on language. Although creators of text art are allowed to deviate from this principle, excessive bending (high curvature) makes the text difficult to follow, and the problem is exacerbated in extreme cases by deformation of the individual
Figure 3.2: Two regions separated by a thin boundary (top). If the text is aligned on both sides, the boundary may be difficult to distinguish (middle). Distinction is improved by enforcing differing text directions across the boundary (bottom).
3.2. Readability

Figure 3.3: In this work, region distinction is achieved via a combination of techniques: differences in orientation, varying text density, and explicit demarcation using isolated lines. Portrait by L. Rotblat, 1897
3.2. Readability

Letters [6, 44]. Lines that wind and bend steeply are also difficult to assemble into coherent text passages (Section 3.2.3).

Note that when discussing the curvature of text lines, we should scale our measurements by the font size in use. A path suitable for rendering tiny 6-point text may be completely inadequate for 36-point headlines.

3.2.2 Line Length and Word Breakup

Passages constructed from short fragments of text are difficult to read, since the reader is forced to skip frequently between different parts of the page. We therefore seek to eliminate short fragments. Beyond that, there is no requirement to maximize line lengths.

As in general typography, breaking up a word across multiple lines is undesirable but not necessarily forbidden. Different artists take different approaches to this matter. Note that extremely short lines might not be able to contain even a single word, which is another reason to avoid them in our layout.

3.2.3 Coherence and Line Ordering

While curvature and line length characterize individual text lines, the concept of coherence applies to the arrangement of multiple lines that forms the complete image. Readers must be able to follow text progression in a smooth and natural manner, and to facilitate this an ideal layout would again adhere to standard typographic convention: regions of evenly-spaced lines, with roughly aligned start and end points within each region. This ideal is often not achievable in micrography, but some layouts approximate it better than others. In figure 3.4, the layout on the left is highly incoherent, while the one on the right is not far removed from typical creative graphic designs (e.g. printed advertisements).

The other piece needed for natural text progression is a full ordering of lines within a given region, and also between different regions, in a manner that matches reader expectations. The reader should be able to accurately predict, given the location of one text line, where to search for the next line in the chain. In the absence of coherent layout this is generally impossible to achieve (Figure 3.4 left), but even with coherent layout the ordering may be ambiguous (Figure 3.4 right).
3.2. Readability

Figure 3.4: Given an incoherent layout (left), it is usually impossible to follow the text in a natural progression. Even with coherent layout (right), determining a natural full ordering among text lines can be far from trivial. Our layout (right) selectively relaxes the alignment requirement to provide both a readable text and a visually appealing representation of the input image. Reprinted with permission of ACM.

3.2.4 Line Spacing

As mentioned in Section 3.1.2, interline spacing in micrography is generally kept to a minimum, but some spacing often remains in the interest of readability. At the very least, overlaps between adjacent lines must be carefully avoided. When using Latin script this task is complicated by variability in the vertical span of letters due to the presence of ascenders and descenders. We have observed that many artists circumvent this issue by designing their art using all caps (Figure 1.1 bottom).

One could argue that line spacing is unrelated to layout, since letters could always be scaled uniformly and/or vertically to alter the amount of leftover space. However, vertical scaling introduces distortion, and uniform scaling changes the amount of text that will fit in a given region and in the image as a whole. Line spacing therefore becomes a factor in the complex planning task that is text layout for micrography.
Chapter 4

System Overview

4.1 Algorithm

4.1.1 Input

The input to the algorithm consists of a vector graphics image, the text from which the microgram is to be constructed, and optional configuration information. The input image contains some combination of three component types: regions to be filled with text, individual strokes to be used directly as text paths (see Figure 10.6), and additional graphical elements which are left untouched by the system (Figure 10.2). Configuration data consists of global settings, adjustments to region-specific properties such as font face and size, and/or local alignment preferences enabling the user to influence the resulting text layout.

4.1.2 Preprocessing

Since the input is a vector image, very limited processing is needed to extract regions and curves. Region boundaries are uniformly sampled, approximated by a polygon, and then triangulated, in preparation for smooth field computation in the following stage. An estimated font size per region is also computed at this time based on region areas, length of the input text, and user preferences. See Chapter 5 for details.

4.1.3 Smooth Field Design

Our primary machinery for designing text layouts is a smooth field computed per region. The field is constrained to align locally with parts of the region boundary. This partial alignment is carefully designed to balance alignment requirements against other aesthetic and readability considerations including smoothness, coherence, and text line length. To determine which parts of the boundary should be aligned we could make use of outlier-robust optimization techniques, which allow one to balance a set of constraints, selectively relaxing ones that are hard to satisfy. However, such methods are
4.1. Algorithm

Figure 4.1: Overview of our text layout method. Given the input image (a), we first compute suitable boundary conditions for a desired smooth field, expressed via region boundary alignment constraints (b), where green denotes a requirement for alignment and red denotes a free boundary. We then calculate a smooth field (c) satisfying these boundary conditions. Next, we trace a set of streamlines, orient and order them, with blue-to-red colouring visualizing line order inside each region (d). Finally, we render a line of text along each streamline (e). Reprinted with permission of ACM.
typically numerically expensive and would require solving a large non-linear system throughout each region. Instead, we separate the problem into two phases: first, we partition the sampled boundary into aligned and unaligned portions. This is done by modelling the problem as a graph-cut and solving using a stochastic quadratic programming search algorithm (Section 6.3). We then construct a smooth 2-rotational symmetry field (Section 6.4) per region given these boundary conditions. The fields are defined over the vertices of the triangulation computed in the preprocessing stage.

4.1.4 Streamline Tracing and Ordering

To construct the layout, we trace a set of streamlines through the smooth field using a custom tracing mechanism which aims to avoid short lines, optimize interline spacing, and place endpoints coherently (Chapter 7). The streamlines are smoothed in a post-processing step to further improve spacing and coherence. In order for readers to easily follow the text’s progression, we determine a consistent orientation among the set of streamlines and compute a traversal order conforming to typographic conventions (Chapter 8). Finding the optimal ordering is equivalent to solving a shortest Hamiltonian path problem, which is known to be NP-Hard, but by relying on coherence of the traced streamlines we can obtain a suitable ordering efficiently.

4.1.5 Text Placement

Given the set of oriented and ordered streamlines, the text is partitioned into fragments of appropriate size. To a large extent we are able to rely in this step on the Adobe Text Engine, which is designed for text handling in traditional publishing tasks. However, some challenges specific to micrography must be addressed by our method. These include choosing the optimal font size per region, and ensuring sufficient text fill for shorter streamlines.

Finally, each text fragment is rendered on its corresponding streamline, using font and colour information supplied by the user. The text may be subtly warped to ensure a good fit, both at the individual streamline level and over the image as a whole.

4.2 Implementation

Our system is implemented in C++ as a plug-in for Adobe Illustrator CS5 [1]. An industry standard vector-based graphic design suite, Illustrator
4.2. Implementation

Figure 4.2: Adobe Illustrator CS5 user interface

provides services such as loading and storing of vector images, a comprehensive library for modelling curves and regions, advanced text processing, a user interface for editing and compositing, etc. Figure 4.2 presents a snapshot of Illustrator’s user interface.

4.2.1 Configuration File

We found that associating custom properties (e.g. font face and size) with specific regions using the Illustrator interface would represent a significant engineering effort. We therefore opt to use a human-readable configuration file instead. An example configuration file is shown in Program 4.1.
4.2. Implementation

Program 4.1 Typical configuration file from our system. The first four lines contain font settings for specific regions. The remaining lines adjust global parameter values.

```
0 1.15 AGaramondPro-Regular 0.1 0 1 1
1 1.5 HamburgerHeavenNF 0.3 0 1 1
2 1.0 AGaramondPro-Regular 0.1 0 1 1
3 0.36 Greetings 0.05 0 1 1
config SAMPLE_DIST_FACTOR 100.0
config LINE_HEIGHT_STRETCH_FACTOR 0.8
config MAX_WARP 0.6
config MIN_STRAIGHT_PATH_LENGTH 3.7
```

4.2.2 Dependencies

Like many geometry processing applications, we rely on standard software components for services such as meshing, numerics and optimization. The system’s dependency list includes:

- Triangle [42] and CML [46] for discrete geometry
- TAUCS [48] linear solver
- Ipopt [51] and MATLAB [29] for quadratic programming

4.2.3 Performance

The system described here is a full-featured proof of concept. It was not engineered for optimal performance. Although the dependencies we use are in general mature software packages, with performance comparable to industry standards, there is room for improvement in our own implementation, integration, and data structures. Section 10.1 presents performance figures for the system as currently implemented.
Chapter 5

Preprocessing

The input to our system consists of a vector graphics image, a given text, and optional configuration. Preliminary processing of the vector image is described in Section 5.1. An estimated font size per region is also computed at this time, explained in Section 5.2.

5.1 Image Preprocessing

The given vector image is first converted to a planar map [5]. This representation, which consists of disjoint regions and non-intersecting curves, is well suited to micrography since it resolves ambiguities in region boundaries, removes overlaps or layers, and establishes clear adjacency relationships between regions. The user can further edit the planar map by splitting/merging regions and by adding new strokes (Figure 5.1). The representation is constructed using Illustrator’s built-in support for planar map conversion and editing. The input image is also scaled uniformly to a standard size, allowing us to process all inputs in a scale-invariant manner. Because of the resolution-independent nature of vector graphics, this is a safe, information-preserving transformation, with none of the adverse effects associated with scaling in raster images.

![Figure 5.1: Planar map editing. Left: a vector image consisting of two overlapping elliptical regions. Centre: the input is converted to a planar map with five non-overlapping regions. Right: the user can further edit the map, e.g. by merging adjacent regions as done in this case.](image)

To maximize the richness and expressiveness of our results, each element in the planar map can be handled in one of three ways:
As a region to be filled with text. A region is described using one closed, piecewise-smooth boundary path and a possibly empty set of closed paths representing holes (Figure 5.2).

- As an isolated stroke used directly for placing a line of text (e.g. the contours of the face in Figure 10.6). Each stroke is specified by an open, piecewise-smooth path.

- As an additional graphical element to be left untouched by the method (e.g. the eyes and eyebrows in Figure 10.2)

Elements of the latter type are copied to the output and removed from consideration. The paths representing the remaining elements are sampled uniformly using arc-length parameterization, and each path is approximated by a polyline. Closed regions are then tessellated using constrained Delaunay triangulation [42]. The vertices generated by the triangulation process will be used to define a smooth field over the region (Chapter 6).

There is a close correspondence between path sampling rates, path feature size, font size, and complexity of the smooth field. The principal guideline here is that image details in micrography cannot be smaller than individual letters, therefore we can use an estimate of the font size in each region (Section 5.2) to derive the minimum feature size and sufficient sampling density. Our smooth fields will also be designed in a way that avoids fine details, so they too do not require very fine sampling. We found that setting the sampling distance to equal the estimated font size is a suitable choice in every case.

Note that this approach may well lead to loss of image details that are deemed necessary by the user. Such issues can be resolved by increasing the amount of available text, either in the microgram as a whole or within the specific region, as described in the next section.

Fill-colour information from the input image is stored in a database and eventually applied to the resulting microgram. This provides a simple and intuitive way for the user to handle colouring. Although only uniform colour per region is supported directly, integration with Illustrator makes it simple matter to use our output as the starting point for further processing, such as applying colour gradients or bitmap image data (Figure 10.8).

Figure 5.2 presents an overview of the system’s image preprocessing workflow.
5.1. Image Preprocessing

Figure 5.2: Image preprocessing workflow. Top: input image. Middle: paths are sampled and approximated by polylines. Bottom: triangulation of closed regions.
5.2 Text Preprocessing

The method is designed to fit the given text into the image exactly once. Clearly this implies a strong link between the amount of text available, the area covered by the input image, and the font size to use. In our setup the former two are given and the latter is determined by the system as explained below. As an additional degree of freedom, we would like to provide user control over font sizing in different regions within the image. The approach we adopt is to let the user specify font sizes in relative terms, as multipliers of some common base size which is not known a priori.

Unfortunately, it is not possible to compute exact font sizes at this point in the algorithm, due to variability in the amount of text that can fit in a given region using a given font size. One source of variability is the fact that text lines can seldom be packed to fill a region with perfectly even spacing. Another is the need to fit complete words into each line (see Chapter 9). The base font size computed at this time is therefore a working estimate, to be adjusted later as part of the text placement stage.

Why do we need font size estimates at this point in the process? One reason is path sampling rates, as discussed in the previous section. Another important reason is the close link between font size and inter-streamline spacing. Typographers measure the size of a font in western languages as the distance from the top of an ascender to the bottom of a descender (e.g. the vertical dimension of the bounding box of the word “boy”). This means that two text lines whose baselines are spaced font-size apart are guaranteed not to overlap.

The base font size $F$ is estimated using a simple quadratic model. Let $A_r$ be the area of region $r$, and let $M_r$ be the user-specified font size multiplier for that region. Then the number of characters used to fill the region is roughly proportional to $\frac{A_r}{(FM_r)^2}$. Recall that in addition to filled regions, the given image may contain isolated strokes representing individual text lines. Assuming a common font size for all such strokes, the total number of characters to be placed along them is proportional to $\frac{L_{la} F}{M_{la}}$, where $L_{la}$ is the total length of isolated strokes and $M_{la}$ is the font size multiplier for isolated strokes. Incorporating the normalization constants, and denoting the number of characters in the given text by $N$, we get the following condition for exact fitting of the text to the image:

$$NF^2 = \left( \frac{1}{K_1} \sum_r A_r \frac{M_r^2}{F^2} \right) + \left( \frac{1}{K_2} \frac{L_{la} F}{M_{la}} \right)$$

The constants $K_1$ and $K_2$ were set by experimentation to 0.2 and 0.4 re-
spectively. We solve this quadratic equation in $F$ and apply the multipliers to obtain an estimate of the font size for each region.
Chapter 6

Smooth Field Design

Smooth fields are at the core of many recent geometry processing methods. The mathematical theory behind them as well as their application domain are rapidly evolving. Since field-guided methods have already been applied successfully to a host of layout problems, they present a natural choice for tackling text layout design. We begin this chapter with a brief overview of field terminology and characterization, then describe a method for field-guided text layout.

6.1 Field Design Primer

A field is an assignment of a value to each point in a subset of Euclidean space. If the assigned value is a scalar, this is called a scalar field. If it is a vector, a vector field. In discrete contexts a field is typically defined at a finite set of points in space (e.g. on a regular grid, at the vertices of a polygon mesh, or on the faces of a triangle mesh), and an interpolation scheme is used to evaluate its values elsewhere.

Most fields of practical interest are smooth in the sense that their value changes gradually as one moves in space, i.e. if the field is defined at two points whose spatial distance is small then the two field values are highly correlated. This also motivates the use of interpolation to determine field values at intermediate locations.

In what follows we focus on the two-dimensional case, which is of interest in the context of this work. The general theory is of course richer and more complex.

A number of different field formulations (Figure 6.1) can be used to guide layout. General 2D vector fields define both a magnitude and a direction at every point. A direction field is a scalar field whose value is interpreted as a rotation angle with respect to some reference frame. The angle is not necessarily constrained to the range \([0, 2\pi]\). An orientation field differs from a direction field in that the angles are defined up to an integer multiple of \(\pi\), inducing a two-way symmetry. To complete the picture, \(N\)-rotational symmetry (N-RoSy) fields have both magnitude and direction, where the
Figure 6.1: Different types of smooth field. Clockwise from top left: a general vector field, direction field, orientation field, 2-rotational symmetry field. Highlighted regions indicate the location of singularities.
6.1. Field Design Primer

Figure 6.2: Effect of singularities on text layout. The presence of a trisector singularity in the middle of this image precludes the definition of a coherent ordering or orientation of text lines in its vicinity.

direction is defined up to an integer multiple of \( \frac{2\pi}{N} \). An orientation field can therefore be thought of as a 2-RoSy field with unit magnitude everywhere.

A singularity (also known as singular point or isolated zero) is a point at which the field vanishes (Figure 6.1 left) or becomes degenerate (Figure 6.1 right). Singularities are interesting from a theoretical point of view since they define the topological nature of a smooth field. They are also important in the context of field-guided layout since they are almost always associated with undesirable artifacts – distortion, irregular vertices, increased computational complexity, and so forth depending on the application. In our method, singularities correspond to text line configurations that cannot be ordered coherently, and – depending on the type of singularity – may be impossible to orient coherently either (Figure 6.2). We would therefore like to identify the conditions under which singularities occur and avoid them where possible.

Theoretical results from differential topology show that singularities cannot always be avoided. The well-known hairy ball theorem guarantees that a smooth field defined over a topological sphere will have at least one singularity. The more general Poincaré-Hopf theorem [32] extends this result to other surfaces, including the one most pertinent to our work – 2D topological disks. According to the theorem, if the field points in the outward direction everywhere along the boundary, then at least one singular point must occur inside the disk. It is easy to see that forcing alignment of the
field everywhere along the boundary would lead to the same result. This is a first indication that field-guided layout methods may want to avoid full alignment along boundaries.

Beyond these topological considerations, singularities also arise naturally in methods that minimize curvature. Field curvature is estimated by the angle deviation of the field between adjacent vertices or faces. Curvature is the opposite of smoothness – a smooth field is one whose curvature is low. Ray et al. [38] show that while it is always possible to reduce the number of singularities to the theoretical minimum, the curvature of the resulting field may be quite high when compared to a field with additional singularities strategically placed at geometric feature points. Their work then proceeds to analyze this trade-off in manifold meshes and proposes a method for controlling it.

We conclude this introduction to field design with a discussion of user constraints. By far the most commonplace and practical type of user intervention in field design is local directional constraints, i.e. having the user specify the direction of a field at a single point, along a path, or within a region. Although such constraints are geometric by nature, they can be supported in topological design frameworks as well [12]. The exact definition of a directional constraint corresponds to the field formulation: in a general vector field directional constraints would be vectors, in a direction field they are angles, and so on.

When applied in the context of curvature-minimizing methods, directional constraints will generally cause an increase in curvature. This takes us back to boundary alignment, presenting another reason why full alignment should be avoided: a field constrained to align with a high-curvature boundary will by necessity have areas of high curvature (Figure 6.3).

## 6.2 Field Design for Text Layout

Since we aim to use the streamlines of a smooth field for placing lines of text, the field should facilitate the formation of streamlines that a priori satisfy the aesthetic and readability requirements discussed in Chapter 3. While the line spacing criterion (Section 3.1.2) is only relevant for streamline tracing and subsequent steps, all others (boundary alignment, curvature, coherence, minimal length and distinction) can be taken into account during field design. Here is how to reformulate these requirements using smooth field terminology:
6.2. Field Design for Text Layout

Figure 6.3: Forcing the field to fully align with region boundaries often leads to high field curvature, which in turn produces sharply bending streamline patterns (highlighted in orange) with multiple singularities (shown as purple dots).

- **Boundary alignment** is a directional constraint enforced along boundaries, specifying that the field’s orientation should be locally tangent to the boundary.

- Similarly, **distinction** is a directional constraint applied along the mutual boundary of two regions, enforcing distinct flow directions on either side.

- **Low curvature** of text lines translates directly as low curvature of the field.

- **Coherence** in field design is largely a matter of avoiding singularities, since there is no way to coherently order lines of text around a singular point. Other coherence considerations are discussed in later chapters.

- Finally, avoidance of **short lines** is a directional constraint enforced at narrow parts of the region, specifying that the text should flow through the narrow part rather than across it (Figure 6.4).

Lines of text have a flow direction rather than just an orientation, i.e. each line has a well-defined starting point. One might therefore presume that the fields we design should also be directional. However, this poses
6.2. Field Design for Text Layout

Figure 6.4: Orienting the field to flow across narrow regions (top) leads to short text lines. These can be avoided by routing the field through narrow regions instead (bottom).

a problem: how to initialize the flow direction along boundaries? Various criteria could be used (e.g. always choose a left-to-right flow), but ultimately these are better applied to a resulting set of streamlines than as an \textit{a priori} constraint on the field – since any added constraint only serves to degrade the smoothness of the generated field. We therefore use an undirected formulation for the field (Section 6.4) and resolve text flow directions in a separate step (Section 8.1).

Various techniques exist for computing fields that satisfy conditions such as alignment or distinction. However, such fields are not necessarily coherent (singularity-free) or have low curvature. As we saw in Section 6.1, fully boundary-aligned fields will in general contain high-curvature regions and multiple singularities. Fortunately, full alignment is not necessary for proper shape depiction – as Figure 3.4 shows, equally good or even better shape depiction can be achieved by \textit{selectively relaxing} the alignment constraint, or boundary conditions, so that the field is no longer required to aligned with certain parts of the boundary. Using a suitable set of boundary conditions we can avoid singularities, eliminate high curvature, and still maintain a pleasing text-art representation of the input shape. By designing these
6.2. Field Design for Text Layout

Figure 6.5: Partial alignment with boundaries can provide the correct balance between shape depiction, smoothness and coherence. Reprinted with permission of ACM.

boundary conditions for an ensemble of adjoining regions simultaneously, distinction requirements can be included in the formulation explicitly.

Figure 6.5 demonstrates the benefits of partial alignment. In the puzzle piece on the right, alignment is only enforced along the more horizontal boundaries of the piece. On the left, the alignment constraint is relaxed at the two rounded tips.

The question we must now answer is how to control the boundary conditions to achieve the right balance between alignment and smoothness. One appealing approach is to integrate the decision into field computation by using a robust error term such as Cauchy’s function [22] or Geman-McClure [17]. Such a method would automatically treat boundary constraints that cannot be met as outliers. There are two problems though: first, the discretization of the problem in complex regions may be quite large, making these methods computationally expensive. Second, incorporating user control into this framework is a difficult challenge.

Another approach involves an iterative reweighting scheme, somewhat similar to the curve orientation method proposed by Xu et al. [58], whereby pieces of the boundary are iteratively relaxed (or constrained) based on an optimality measure. While this method is better able to accommodate user constraints, it requires successive computations of the field, and in most cases will not find a globally optimal solution without extensive backtracking.

Instead, we observe that in our setup the behaviour of the field can be predicted based on the shape of the boundary alone, without computing the
field at all. We can therefore break the process into two steps: first compute a suitable set of boundary conditions, then use a simple linear solver to obtain the field throughout. Each step provides a global, one-step solution to the problem it addresses.

6.3 Boundary Condition Design

The only type of boundary constraint in our field design task is alignment with region boundaries. To design the conditions we therefore only need to decide for each point on the boundary: should the text be aligned to the boundary at that point, or not? We use the sampled boundary from Section 5.1 to discretize the problem, and aim to assign a boolean value per sample: 1 for alignment, 0 for relaxation (free boundary). We formulate this assignment as a global labelling problem for all points along region boundaries.

The resulting labelling should satisfy two imperatives. First, the boundary conditions it induces should cause the field’s direction to change slowly as we move in space. Slow change takes care of the low-curvature requirement, and also precludes singularities, since at singular points the direction changes in a discontinuous manner. Applied to boundary points, the slow-change requirement defines a preference relation between pairs of nearby points, based on the similarity or dissimilarity of their tangent directions. Figure 6.6 illustrates this idea: if two nearby points have similar tangents, we want to avoid a situation where the field is forced to align with one of them but not the other, since having the field follow the boundary and then suddenly veer away would be damaging from a shape-depiction point of view. Conversely, if two nearby points have dissimilar tangents, we want to avoid aligning the field with both of them since high curvature would result. As Figure 6.6 shows, dissimilarity is most evident near sharp corners but can also occur in smooth high-curvature sections of the boundary.

The second requirement that the labelling must fulfill is to maximize the proportion of the boundary that remains aligned, in the interest of shape depiction. This is an individual, pointwise preference. By default, every point on the boundary prefers to be assigned the “aligned” label, with the exception of points at sharp corners where no meaningful tangent direction is available. However, the strength (or weight) of this preference can vary based on global considerations and specifically the impact of alignment choice on the length of traced streamlines, and consequently text lines (see Section 3.2.2). While it is hard to predict the length of a streamline aligned
6.3. Boundary Condition Design

Figure 6.6: To ensure that the field’s direction changes slowly, pairs of boundary point should have similar alignment labels if their tangent directions are similar (bottom and right), and differing labels if their tangents are dissimilar (top and left).

Figure 6.7: By forcing boundary alignment at points where the feature diameter is small (marked red), the field is constrained to flow through the narrow feature rather than across it.
with a boundary at any given point, a good estimate of the length of a non-aligned, or orthogonal, streamline is the feature diameter at that point. Hence, since we aim to reduce the number of short streamlines, the preference for alignment should have an inverse relationship to the diameter. This ensures that at very narrow features the field would flow through the feature rather than across it (Figure 6.7).

Labelling problems such as this one are often formulated as a graph-cut. Each point is represented as a node in the graph, and the pairwise preferences are weighted edges between nodes. We construct such a graph, adding an edge from each point to all points that are visible from it, i.e. the line between the two points does not intersect region boundaries (see Figure 6.8 right).

Figure 6.8: The graph constructed for the jigsaw example in Figure 4.1. Left: strength of individual preference for alignment for each boundary point, visualized by a red to green range (green represents maximal preference). Right: visualization of the weighted-edges graph from one boundary point (shown in yellow) to its neighbourhood; node and edge colour represent pairwise weights, with negative weights in red, positive in green. The full graph consists of such edges from all boundary points. Reprinted with permission of ACM.
6.3. Boundary Condition Design

6.3.1 Pairwise Preference Weights

Edge weights in our labelling problem reflect the expected degree of similarity or dissimilarity between node labels. This requirement can be encoded as a positive or negative weight assignment $w_{ij}$ to the edge between nodes $i$ and $j$, with positive weights indicating a preference for similar labels and negative weights signifying a preference for dissimilar labels. The angle $a_{ij}$ between the two tangent directions controls the sign as well as the strength of the preference.

Since our rationale for pairwise preferences applies to boundary points that are near one another, the distance $d_{ij}$ between the two points also figures into the weight assignment. The exact definition of distance for this purpose requires some care: we distinguish between point pairs that are mutually visible inside the region (i.e. the line segment connecting them is contained within the region) and points that are visible outside. Euclidean distance can be used as an indicator of mutual influence in the former case, but is often misleading in the latter – for example, the two “tips” at the bottom of the jigsaw piece in Figure 6.5 are close to each other in Euclidean space, but their local field orientations are largely independent. On the other hand, if we exclude “outside-pairs” from the formulation entirely we lose valuable information. Case in point: without them, no dissimilarity weights would be generated at a concave corner. Our approach to outside-pairs is to measure their distance $d_{ij}$ by the length of the boundary section connecting them. Geodesic distance would be another suitable alternative for this purpose. The boundary distance is then multiplied by $c = 3$ to further attenuate the influence of outside-pairs compared to the more critical inside-pairs.

The final weighting formula for a single edge is

$$w_{ij} = F_a(a_{ij})F_d(d_{ij})$$  \hspace{1cm} (6.1)

where the angle factor $F_a$ is a sigmoid function that varies smoothly from -1 to 1. Specifically, we use the hyperbolic tangent function $tanh$ as follows:

$$F_a(a_{ij}) = tanh\left(\frac{\pi}{2} - a_{ij} - \frac{\pi}{4}\right)$$  \hspace{1cm} (6.2)

With appropriate scaling this results in a smooth transition from a weight of 1 for perfectly aligned tangents to -1 for orthogonal ones. For the distance factor $F_d$ we use Gaussian falloff, causing the preference to decay rapidly as distance increases:

$$F_d(d_{ij}) = e^{-d_{ij}^2/\sigma_d^2}$$  \hspace{1cm} (6.3)
6.3. Boundary Condition Design

with $\sigma_d$ set to 20% of the diagonal length of the region’s bounding box. This scale-invariant falloff achieves good balance between coherence on the one hand and individual alignment preferences on the other.

If the input contains multiple adjoining regions, we may want to distinguish them visually using text directions (see Section 3.1.4). To do this the user can invoke an algorithm variant that incorporates this constraint into the labelling problem: we duplicate the nodes of boundary points located on shared boundaries and associate a copy of the node with each adjacent region. We then introduce an edge with the maximal negative weight $w_{ij} = -1$ between the two copies. This approach was used in producing the jigsaw image in Figure 4.1. The shared-boundary distinction constraint is shown as a red line connecting the two jigsaw pieces in Figure 6.8 (right).

Note that in the absence of distinction constraints our labelling problem can be solved per region in isolation. Our system takes advantage of this, solving per region when possible, and only constructing and solving a unified problem if the user has indeed requested distinction constraints.

6.3.2 Individual Preference Weights

We want to relate the individual preference strength at each boundary point to its feature diameter. One way to measure the diameter is to use the local feature size [2] and the degree to which the boundary tangent is aligned with

![Figure 6.9: Local feature size estimation. A bundle of rays around the inward normal direction is used to obtain a robust estimate for the distance and orientation of the “opposite” boundary.](image)

43
6.3. Boundary Condition Design

the local feature skeleton. However, robustly computing the local feature size and skeleton and matching the boundary to the skeleton are nontrivial tasks. Instead, similar to Shapira et al. [40], we use an approximate “diameter” measure by evaluating the distance and degree of alignment between the normal at each point and the normal at the “opposite” boundary of the region (Figure 6.9).

For node $i$ representing boundary point $v_i$, we define the opposite boundary point as the closest intersection of the boundary with a ray emanating from $v_i$ in the inward normal direction. The distance to the intersection approximates twice the local feature size, while the angle between the normals (or tangents) at $v_i$ and its opposite point reflects the degree of alignment with the local skeleton. To avoid inaccuracies due to slight variations in a normal we consider a cone of rays around the inward normal of $v_i$ rather than just a single ray. We trim the resulting set of opposite points to remove outliers in terms of distance and normals, and use an averaged distance $d_i$ and average normal direction $a_i$ to compute the individual preference weight $\alpha_i$ for node $i$ (Figure 6.8 left):

$$\alpha_i = F_a^+(a_i)F_d(d_i)$$  \hspace{1cm} (6.4)

This formulation is similar to the pairwise weight function in Equation 6.1 except that we clamp negative values of the angle factor $F_a$ to zero to produce $F_a^+$, ensuring that the individual preference weights obtained are non-negative.

6.3.3 Completing the Formulation

We now have a classical labelling problem based on graph cut, where the optimal solution is found by minimizing a quadratic functional:

$$\min \sum_{ij} w_{ij}(l_i - l_j)^2 + \omega \sum_i \alpha_i (1 - l_i)$$  \hspace{1cm} (6.5)

subj. to $0 \leq l_i \leq 1$

The left-hand term is the aggregate of all pairwise preferences. The right-hand term corresponds to individual preferences. The coefficient $\omega$ controls the trade-off between the two, and has a default setting of $\omega = 8$. There is however a scaling issue to resolve: there are as many individual-preference terms as there are boundary points, but the number of pairwise preferences is quadratic in the number of boundary points. As a result, the relative influence of the two terms is dependent on sampling density along the boundaries.
6.3. Boundary Condition Design

To remove this dependence, we scale the individual preference weights $\alpha_i$ by $\|W\|/\|\alpha\|$ where $W$ is the symmetric weight matrix and $\alpha$ the vector of individual preference weights.

6.3.4 Solving

If all weights $w_{ij}$ were positive, Equation 6.5 would be a standard convex problem. However, with negative weights the problem becomes much harder. In fact, if we ignore the individual preference component, then by flipping the negative and positive weights the graph-cut problem becomes one of finding a maximal cut – one of the 21 classical NP-complete problems [24]. A combinatorial approach would therefore be prohibitively expensive.

Another approach that may seem appealing is graph partitioning. Approximation methods such as normalized cut [43] and isoperimetric partitioning [19] can handle negative edge weights efficiently. However, these methods are designed to partition a graph into subsets of roughly equal size (or “volume” according to various metrics). They can be adapted to partition a graph into unequal size sets, but the sizes must be specified ahead of time. In our case, there is no reason to aim for balanced partitioning of the boundary into aligned and unaligned points, or to aim for any other size ratio defined in advance. Instead, the solution should be driven only by

Figure 6.10: Two candidate solutions encountered during the stochastic search. One of the local minima appears on the left. The best labelling found is shown on the right. Reprinted with permission of ACM.

45
the pairwise dissimilarity preferences, with the number of unaligned points minimized subject to these preferences.

In lieu of these general graph-cut techniques we develop a search procedure tailored to this case. The weight settings in our formulation impose a special structure on the problem: optimal solutions tend to have chains of consecutive boundary points with similar labels, with only a small number of discontinuities where the label changes (see Figure 6.10). Consequently, we can to use a custom stochastic search approach to compute a satisfactory solution.

We generate a series of randomized initial guesses, then search locally for the best solution near each guess. For the local search we use a quadratic programming approach [18], which is well suited for this type of problem. The first initial guess is a neutral setting of \( l_i = 0.5 \) for all \( i \). Intuitively, given this setting the method is most likely to move the labels according to individual preference weights, keeping chains of boundary points together when they are similar and separating them when they are not. We proceed with a local randomization strategy whereby each new initial guess is generated either from the best global solution so far, or from the most recent solution. To generate a new initial guess from a given solution, we randomly select consecutive portions of the region boundary and invert their labels.

![Figure 6.11: A single iteration of the boundary solver. Aligned points are shown in red, unaligned in blue. Left: labelling from a previous iteration. Centre: new initial guess produced by flipping the labels along a portion of the boundary. Right: new optimized labelling computed by the quadratic programming solver.](image-url)
6.4 Computing the Field

The location of the inverted portion and its length are selected randomly with uniform probability, with at least one third of boundary points inverted in each iteration (Figure 6.11). Ten to twenty iterations are typically sufficient to arrive at the desired result.

Since the procedure above computes real-valued labels, a rounding mechanism is required to define the final partition. Thanks to the nature of our pairwise preference weighting, the number of labels whose value is neither 0 nor 1 is typically very small. Our implementation uses a permissive rounding threshold of $T_r = 0.9$, labelling nodes as unaligned if their real-valued label is less than the threshold. In practice, the influence of the threshold on the final result is minimal.

6.4 Computing the Field

Once the boundary conditions are computed, we solve for a smooth field within each region. As previously discussed, many methods have been proposed for tackling this problem. However, since the boundary conditions were carefully designed to produce simple, slowly changing fields with no singularities, the choice of field formulation is less crucial—any reasonable approach will produce the field we want. In order to decouple the question of text-line orientation from field computation (see Section 6.2) we use a two-way symmetric formulation. Specifically, we follow the method of Palacios and Zhang [34] by solving for 2-rotational symmetry (2-RoSy) fields. This formulation offers a high degree of smoothness and is very efficient to compute.

The field is defined over the vertices of the triangulation from Section 5.1. It is computed individually inside each region. As in the work of Palacios and Zhang, the 2-RoSy field is derived by solving for its corresponding representation vector field $R$:

$$R_i = \begin{pmatrix} \cos 2\theta_i \\ \sin 2\theta_i \end{pmatrix}$$

where $\theta_i \in [0, \pi)$ is the angle of the 2-RoSy field at vertex $i$ of the triangulation. The solution itself is a standard least squares minimization problem. For each boundary point $l$ labelled “aligned” by the boundary constraint design algorithm, the representation $R'_l$ of the tangent at that point is added to the system as a soft constraint on the corresponding variable $R_l$. We thus have:

$$\min \sum_{ij} \| R_i - R_j \|^2 + \sum_l \| R_l - R'_l \|^2$$
The solution is computed using the TAUCS linear solver [48].

We found that the fields generated using our two-step approach are in general singularity free and low-curvature whenever a solution exists that does not violate other critical requirements such as user constraints or very thin features. If no such solution is available, the resulting field may still contain a few singularities or areas of higher curvature. Advanced field design schemes such as those of Ray et al. [38] or Fisher et al. [15] may handle such cases better than the 2-Rosy approach. On the other hand, methods that avoid singularities at all costs [12] are less suitable for our needs as they tend to compensate with an undesirable increase in curvature.

An example of a singularity generated due to user-imposed alignment constraints is shown in Figure 10.11 (bottom), where the artist required that the text be aligned with both the top and the sides of the head. Figure 10.1 contains a few regions where coherence and low curvature could not be satisfied without including some very short lines. Consequently, higher curvature or one or two singularities were introduced automatically to allow for longer streamlines.

6.5 User Control

A key advantage of field-guided methods is that users can interact with the process in an efficient and intuitive manner, imposing their requirements in the form of sparse constraints. This is helpful since layout tasks in general do not have a single “correct” solution, and artists may have additional considerations that are not part of our framework.

We considered a number of mechanisms for users to control the design of boundary conditions and consequently the smooth field. These included a direct editing interface for algorithm parameters, a post-process field editor as described by Palacios and Zhang [34], and selection among different local minima of the quadratic programming solver. We found that the most intuitive approach is a direct visual interface for specifying preferred (or undesirable) alignment constraints. In our system, users are able to control the field in two different ways:

- Directly sketching preferred directions as strokes inside the relevant region. These strokes take precedence over boundary alignment considerations and are used in rare circumstances, e.g. the background of the poppy in Figure 10.3.

- Marking portions of the boundary as ones that should (or should not)
be aligned with the field. These preferences are added as strong individual preference weights to the boundary formulation.

User interaction is most often an interactive process. In the first phase the system is invoked without constraints, generating boundary conditions, smooth fields and a visualization using the default settings. The user then applies a set of preferences and runs the algorithm again, producing a new layout design and visualization. The process repeats until the desired result is obtained.

Figure 10.11 presents two micrograms generated from the same image and text – one produced automatically, the other with the inclusion of user-defined boundary constraints.

Another interaction paradigm involves the trade-off weight \( \omega \) in Equation 6.5, which controls the balance between alignment and coherence. By modifying its value and regenerating the boundary conditions, the user can raise the proportion of boundary points that are aligned at the cost of increased curvature, or vice versa. See Figure 6.12 for an example of the effect of this parameter.

In addition, variants of the boundary labelling formulation supporting

![Figure 6.12: Effect of the trade-off parameter \( \omega \). When set very low (left), boundary condition design is dominated by coherence, resulting in a very smooth field but also producing large unaligned sections and many short lines. A balanced setting (top centre) improves alignment at the cost of some increase in curvature. Very high settings (left) result in a field that is almost fully aligned.](image-url)
distinction constraints (Section 6.3.1) and symmetry (Section 6.6) can be enabled or disabled by the user as global switches.

6.6 Symmetry

Figure 6.13 presents a shape with a high degree of symmetry, which raises the question of whether we can (or should) design text layouts that reflect symmetry in the input. By default, our boundary condition formulation prefers to maximize smoothness and coherence even at the cost of broken symmetry (Figure 6.13 a,b). The user can however choose to override this behaviour.

To account for symmetry we use a modification of the boundary labelling formulation (Equation 6.5) similar to the one used to create region distinction in Section 6.3.1. Specifically, we add strong pairwise similarity constraints between points that are equivalent under symmetry. The equivalence relationships, computed by hand or by a symmetry analysis algorithm [27], must be provided as auxiliary input. The resulting layout, shown in Figure 6.13 c, respects the shape’s symmetry, but the added constraints produced a few singularities and areas of high curvature.
6.6. Symmetry

Figure 6.13: Forcing symmetry: (a) default asymmetric field generated by our method. (b) corresponding text layout (c) alternative text layout generated by adding pairwise symmetry weights to the boundary formulation. Text: *Lorem ipsum*. Reprinted with permission of ACM.
Chapter 7

Line Tracing

The next task after field design is to trace a set of streamlines through each region. These lines will then serve as skeletons for individual lines of text in the final microgram. Since our fields were carefully designed to satisfy shape depiction and coherence requirements a priori, we can use a fairly simple approach when tracing streamlines through them. Still, there are gains to be made by analyzing and optimizing the tracing process for the purpose of micrography.

7.1 Goals and Requirements

By default, the set of lines we trace should be as evenly spaced as possible, so that text can be rendered on them without overlaps or gaps with only minimal deformation of individual letters. An exception to this rule is when the artist chooses to vary the font size within a given region to increase visual interest. The system supports this feature, with certain restrictions – see Section 7.4.

Traced lines should of course avoid abrupt curves to prevent excessive deformation of text placed along them. Much of our focus in field design has been on ensuring that the smooth field does not exhibit high curvature, but we also designed the field to allow for the existence of singularities in certain situations. The curvature of a traced streamline as it approaches a singular point is unbounded. The tracing method must therefore take special measures near singular points.

Short lines are another artifact that we would like to avoid. The field design process eliminates the “inevitable” short lines that would result from field flow across narrow features, so the tracing step need only avoid spurious short lines in other places. As it turns out, this goal is shared with standard techniques for field visualization. We can therefore adopt existing solutions for our purpose.

A requirement that sets us apart from other streamline-tracing methods is that we aim to mimic, as much as possible, the reading flow of standard rectangular layouts that readers are accustomed to. In a rectangular layout,
7.2. Tracing Algorithm

Our method is based on the greedy tracing algorithm of Jobard and Lefer [23]. Their approach is appealing because of its simplicity and amenability to modifications that stress various design goals, as exemplified in the street modelling method of Chen et al. [11]. Starting from an initial seed point, a streamline is grown by tracing the field in both directions using numerical integration. At each integration step, new seed candidates are produced near the current position at a fixed distance in the direction orthogonal to the field (i.e. “above” and “below” the line currently being traced), and inserted into a queue. The fixed distance reflects ideal spacing between adjacent streamlines, which in our case depends on the font size. Once the current line has been traced to its termination on both ends, a new candidate is selected from the queue and used as the seed for the next streamline. The process ends when the candidate queue becomes empty. Figure 7.1 illustrates the tracing method.

The original Jobard and Lefer algorithm used a simple FIFO queue to store candidate seeds. Chen et al. introduced a priority queue for smarter selection of the next seed point, and our system does the same. Our priority function is designed to encourage streamlines that extend to the boundary of the region and have their endpoints aligned with adjacent streamlines. A good heuristic for both objectives is to begin tracing new lines close to the endpoint of an existing line. Hence, seeds are given a higher priority the closer they are to an existing endpoint.

To improve approximation of the region’s boundary, the process is initialized by filling the queue with points near the region’s boundaries, specifically points at an offset of half the text height inward from the boundary. Higher priority is given to points where the field direction is well aligned with the
7.2. Tracing Algorithm

Figure 7.1: Illustration of the greedy streamline tracing algorithm. Top: the queue of seed candidates is initialized from the boundaries. Here seeds are coloured according to how well the field aligns with the boundary at that location (red for perfectly aligned, blue for orthogonal). Centre: An initial seed (in green) is selected and a streamline traced from it in both directions. Bottom: A new seed is selected near the previous streamline, and a new streamline is traced. Note that the new seed is not one of the initial seeds shown at the top.
7.3 Deviations from the Field

boundary. To prevent these initial seeds from interfering with the rest of the tracing logic, their priorities are set lower than any of the candidates generated during tracing. This means that in most circumstances, only one initial seed will ever be used in any given region. The exception is regions containing very narrow features, for example the two “hooks” of the large orange-coloured region in Figure 10.1. Streamlines generated elsewhere in the region usually cannot penetrate into such narrow spots. In that case, initial seeds placed inside the feature will ensure that at least one streamline is generated within it.

When it is time to start tracing a new streamline, the highest-priority candidate is removed from the queue and tested to see if it is sufficiently distant from the boundaries and from all existing streamlines. If so, it is used to trace a new streamline. Otherwise it is discarded and the next candidate is extracted from the queue.

In our implementation the actual tracing is performed using the midpoint integration rule [20], which balances out errors and provides an excellent degree of accuracy (given that abrupt direction changes are largely absent from the field). Lines are terminated when they reach the boundary, come too close to another streamline, or make a sharp turn. The latter can only indicate the presence of a nearby singularity. The strategy of terminating streamlines that approach a singularity has performed well in streamline visualization methods, and is a workable solution in our case as well (see Figure 10.11 bottom).

7.3 Deviations from the Field

Since exact visualization of the underlying field is not our goal, we may deviate from the strict “streamline” definition in the interest of coherence and shape depiction. This comes into play when a streamline follows a trajectory almost parallel to an existing feature – either another streamline or a boundary – with the distance to the feature gradually decreasing. At some point the distance will drop below a preset threshold, causing the new streamline to be terminated. If the streamline was following a boundary, terminating it at this point produces an aliasing artifact. If it was following another streamline, this represents a streamline that stopped midway through the region. Both effects are undesirable, and can often be avoided by letting the streamline continue to run in parallel until it hits a boundary “head-on”. To support this behaviour robustly we augment the primary smooth field with a weak field around each feature that repels new streamlines away
from it. This repelling field decays exponentially with distance, and is weak enough to only affect the trajectory of streamlines that are almost parallel. To witness the effect of the repelling field, compare images (b) and (c) in Figure 7.2. Several streamlines that were cut short in the version without repelling fields (b) extend much further when these fields are introduced (c), in many cases all the way to the opposite boundary.

Once tracing has completed we abandon the field-guided paradigm, working directly with the set of generated streamline curves. A number of cleanup operations are performed at this time: first, lines that are near-continuations of one another are stitched together. The tracing method is designed to minimize such cases, but if they do occur we have more freedom in our scenario to patch them, whereas in pure streamline-based visualization this would not be allowed. Very short lines, if generated, are simply discarded. Lines that were terminated before reaching the boundary can sometimes be extended to the boundary. The idea behind these transformations is to remove visual artifacts that are damaging in the context of micrography, trading them for increased variability in interline spacing, which is more easily handled.

The progression from basic Jobard and Lefer tracing to our final algorithm is presented in Figure 7.2.

### 7.4 Spacing and Enrichment

The final step in our layout design is a readjustment of the spacing between adjacent lines. The user may prefer the spacing between streamlines in a specific region be uniform, or not: uniform spacing produces even shading, as seen for example in Figures 10.9 and 10.11, while non-uniform spacing can add richness to the output, as in Figures 10.2 and 10.10.

In the uniform case, although the tracing algorithm aims to produce evenly-spaced streamlines, we can still improve on its results after the fact. This is due to a number of reasons: first, the greedy algorithm makes no guarantee of global optimality. Second, during tracing we constrained ourselves to follow an underlying smooth field, but small deviations from the field are perfectly acceptable in our setup. Compare for example images (c) and (d) in Figure 7.2, which show the result before and after post-process adjustment. Notice that in (d), whenever a line terminates before reaching the boundary, the lines on either side of it bend toward each other to fill the resulting gap. Such behaviour would be difficult to integrate into the tracing process, but is easy to implement once tracing has completed. A final reason for post-process adjustment is compensation for the effect of
Figure 7.2: Elements of our tracing algorithm: (a) The basic tracing mechanism of Jobard and Lefer creates an unorganized layout with numerous short lines. (b) Our seed selection strategy encourages streamlines that are flush with the boundary. (c) Repelling fields help prevent lines breaks inside the shape. (d) After post-process smoothing, the layout is ready for text rendering. Reprinted with permission of ACM.
7.4. Spacing and Enrichment

cleanup operations such as short line removal.

Spacing adjustment is performed by iteratively moving streamline vertices (i.e. the points computed via numerical integration) along the orthogonal axis to the local tangent direction. For each vertex, the intersection of this axis with the two adjacent streamlines (or borders, as the case may be) is computed. If uniform spacing is desired, the vertex is moved to the average of the two intersections. For non-uniform spacing we use a weighted average, with weights per vertex determined at random ahead of time.

The random weighting approach has a number of limitations, but it still illustrates the added richness that non-uniform spacing can provide. Some tracing methods support non-uniform streamline tracing by introducing a spatial density function [31]. In micrography, it is often desirable to use non-uniform sizing to emphasize a specific word or phrase, regardless of where exactly it is located. Our system does not support these options however.
Chapter 8

Line Orientation and Ordering

Laying out text along a set of arbitrarily oriented streamlines requires selecting a text orientation along each line and computing a line ordering\(^1\).

8.1 Orientation

A basic requirement for intuitive line orientation within each region is that adjacent parallel lines have the same orientation. Depending on the curvature of the lines, this requirement may not always be satisfiable. For instance, consider one line that bends around another as on the skull forehead in Figure 10.11 (bottom). Our method aims to find a consistent global orientation that is similar on adjacent lines, when possible.

We define a weighted graph where each node corresponds to a streamline and directed edges connect each pair of adjacent parallel streamlines. The weight associated with each edge from one streamline to another is set to be the percentage of the length of the first streamline where the two lines are adjacent. Note that two adjacent streamlines will have two directed edges, but these edges may have different weights (Figure 8.1a).

The orientation within each region is defined using a prioritized breadth-first traversal on this graph. To initialize the traversal we select a line that has a clear orientation preference, i.e. it is largely horizontal, and has sufficient length (at or above the average line length in the region). The initial line is oriented left-to-right and this orientation is propagated through the graph using a breadth traversal order based on the edge weights. If no line has a strong left-right preference, e.g. if all streamlines are vertical, we use a bottom-up orientation instead, consistent with a left-right line order.

\(^1\)This chapter was written by Mikhail Bessmeltsev and Prof. Alla Sheffer, and describes an algorithm developed by them. The text of this chapter was published previously as detailed in the preface, and is reproduced here for the sake of completeness.
8.1. Orientation

Figure 8.1: Orientation and ordering demonstrated on a subset of streamlines: (a) initial adjacency graph, with edge colour representing weights (b) spanning forest of oriented lines, with connection edges highlighted in black (c) final order of lines (blue to red). Reprinted with permission of ACM.
8.2 Ordering

The requirement for intuitive ordering involves both region ordering and line ordering within each region. Inside a region, for any two adjacent parallel lines, the lower one with respect to the determined orientation should, whenever possible, come after the upper one. This requirement defines a partial, rather than full, order inside the region, as a streamline can have more than one adjacent line above or below it. Moreover, this partial order can contain loops when lines have high curvature, potentially wrapping around other lines, as is the case in the previously mentioned skull example. To define ordering we must convert this partial order to a full order. In the general case, solving this problem with respect to some metric is equivalent to finding a shortest Hamiltonian path, and is NP-Hard. The algorithm we use is targeted at finding an intuitive, readable order when the partial order is consistent, and a plausible one when loops are present.

To find the ordering we use a subset of the streamline graph defined for orientation, discarding edges from lower streamlines to the streamlines above them. Next, we compute an optimum branching [13], which defines a spanning forest of the graph (Figure 8.1b). To create a full order we need to connect the forest into a single graph. We traverse the nodes of all trees to locate the shortest edge connecting two trees and add it to the graph. This process is repeated until all the trees are joined together or no edges linking trees are found. In the latter case we treat each sub-graph as a separate region for text layout.

To traverse and order the combined graph, we pick as a root the left-most orphan node (i.e. one with no incoming edges) with respect to the frame defined by the line orientation. We traverse the graph starting from the root. Each time we encounter a node with more than one parent, we traverse all branches above this node in a left-to-right order and only then traverse the node itself. At each branching point with multiple children we use the same left-to-right order to traverse those (Figure 8.1c).

To order the regions themselves, we use a similar process, but on a region graph instead of a lines graph. The order here is more vague as we have no clear notion of adjacency or parallelism. Instead region nodes are connected by edges based on global adjacency computed using a Voronoi diagram, and edge weights are set to be inversely proportional to the distance between the regions, and directly proportional to the dot product of the transition direction between regions and the preference direction. The preference can follow a lexicographic left-right, top-bottom order, or some mix of the two directions set by specifying an arbitrary direction vector such as (1, 1). We
then use a similar graph traversal mechanism where the local frame is set based on the average of the line orientations in adjacent regions.

To order individual strokes (text lines not included in a region) we can treat each as a stand-alone region, or we can treat them all as a single region. While we cannot claim that the ordering computed is necessarily globally optimal, we found that our results agree, in general, with manual ordering both inside and in between regions.
Chapter 9

Rendering Text

We now have a set of ordered curves on which to render the text of the microgram. Placing of text is a typesetting problem, albeit a highly unusual one: lines in general are not straight, their lengths can vary dramatically, and there are good reasons to vary the font size locally, as we shall see.

9.1 Typesetting

A dedicated typesetting engine for micrography, while useful, is beyond the scope of this work. Instead we use the Adobe Text Engine (ATE) package included with Adobe Illustrator. The ATE takes care of two major tasks: partitioning the given text into fragments that can fit onto the curves, and rendering fragments with user-specified font face, size and colour. These fragments are then transferred onto the target curves using the skeletal strokes method (Section 2.1).

Remember that our goal is to fit the given text onto the microgram exactly once. An approximate base font size for achieving this was derived in Section 5.2, but the precise size may differ somewhat. We therefore perform a simple search procedure to find the optimal base size, repeatedly rendering the text at different sizes to determine the largest font that does not cause any overflow.

We configure ATE to prevent breaking up individual words over multiple lines. As with any typesetting system, the text to be rendered on a line may not cover the available space exactly – this is normally handled either by “flush left”, i.e. aligning the text with the start of the line while leaving some white-space at the end, or by “full justification”, which increases the spacing between words so that the text aligns with both start and end. In our case, flush left is clearly unsuitable due to shape-depiction requirements. Full justification is better, but large chunks of white-space within a line can become a noticeable shading artifact. Instead, we render the text in flush-left mode and stretch the result to the desired length. As long as the stretch factor is limited, the effect remains largely invisible and produces more uniform shading than full justification.
9.2. Text Warping

Short lines present a special challenge. Since word breakups are avoided, the fitting of text to a short line in the presence of long words may fail, leaving the line partially or even entirely blank. We found that blank lines in the microgram – even short ones – immediately draw the observer’s eye and greatly diminish the work’s aesthetic appeal. To prevent this we use a version of the stretching idea above: when fragmenting the text, lines shorter than some minimal threshold (e.g. 4 times the font size) are clamped up to the threshold. Once the fragments are determined, if the size of a fragment exceeds the actual length of its curve, the text is rescaled to fit in the available space. This introduces some distortion but is less damaging than blank space.

The bear images in Figure 10.9 demonstrate the effect of our typesetting approach. Most of the bear’s body, constructed of long text lines, is very uniform in both shading and text size. More variation in size and shading is found in those legs of the bear that, in the interest of coherence, were laid out using shorter lines. This variation is due to stretching of shorter words and contraction of longer phrases to fit a particular line length.

9.2 Text Warping

At this point the microgram is essentially complete, but a more aesthetically pleasing result is obtained by subtly warping the text in two ways. First,
we account for local variations in spacing between neighbouring lines. Although our field design and tracing algorithm reduce such variations to a bare minimum, perfect spacing is impossible in any nontrivial layout task. To avoid visual gaps in the image we adaptively warp the height of the text to fill the space more uniformly. We measure the distance from each curve to its neighbours at regular intervals along the curve and construct smoothly varying height profiles, which are then applied to the rendered text.

Next, the beginning and end of each text line are adjusted to better match the region boundary. Whenever a text line is neither aligned nor orthogonal to the boundary, some jagged aliasing artifacts may result. Here too our careful field design and streamline tracing methods minimize this effect, but it can still occur as we balance other considerations as well. To reduce the effect further we apply a cage deformation [53] as illustrated in Figure 9.2. We first render the text horizontally and define an initial cage coinciding with the text’s bounding box. This rectangular cage is then deformed into a trapezoid by rotating its vertical edges, with the amount of rotation defined by the angle between streamline and boundary at the corresponding endpoint of the streamline. For endpoints that do not occur at a boundary, no deformation is applied. The resulting warped text is then rendered along the streamline.

Figure 9.1 demonstrates the effect of text warping on the final result.
9.2. Text Warping

Figure 9.2: Warping the endpoints of text lines to improve boundary alignment. Top: results with no warping. Bottom: with warping. The cage-warped horizontal text is shown in the bottom left corner. The cage itself is visualized for the last four text lines.
Chapter 10

Results

This chapter presents a diverse selection of micrograms created with our system, from graphics and text available freely online. Part of the artistry of micrography lies in selecting source graphics that have the right amount of detail, along with the correct amount of text. If the image is very detailed or the font too large then individual letters begin to play a crucial role; we thus leave the realm of micrography for that of general calligraphy, which is outside the scope of this work. In our system, the choice of sources remains entirely up to the user.

Except where stated otherwise, the layouts for all results were computed automatically based on the input graphics and text. This allowed us to rapidly experiment with different fonts and texts to achieve a pleasing result. We consider this capacity for iterative microgram design a key advantage of our system over established digital art software for artists working in this field.

The microgram “Motif” (Figure 10.1) was created using text from Lewis Carroll’s *Alice in Wonderland*. Font size multipliers were used to vary the size of the text in the various regions, adding to the visual complexity of the result. By traditional standards this is only a medium-sized microgram, but even so determining the hundreds of lines it is comprised of would be difficult as well as tedious if done by hand. This example also demonstrates the challenge of balancing readability and image depiction requirements. Given the complicated nature of the abstract input shapes, maintaining coherence in some of them is far from trivial. For most regions a singularity-free solution exists which naturally aligns with narrow features. However, the orange-coloured region at top-right has six narrow areas in various orientations which are very hard to reconcile. Consequently, a few high-curvature areas are produced within the region to obtain what is deemed an acceptable trade-off between coherence and alignment.

The “Summer Girl” image is combined with words from the *Song of Songs*. The hair consists of two highly intricate regions, with smooth random variation in size used to create the visual effect of hair strands. The large size and high complexity of these regions make for a particularly challenging
Figure 10.1: “Motif”. Image source: allfreevectors.com. Text: excerpt from Alice in Wonderland. Input image inset. Reprinted with permission of ACM. Please zoom into the figures in this chapter using the digital version to read the fine text.
boundary condition problem. Here we are able to produce a pleasing result at reduced computational cost by using a local reweighting approach, which is based on individual alignment preferences (Section 6.3.2). This approach enforces boundary alignment at narrow areas while allowing some relaxation in places where the feature diameter is large.

Figure 10.3 adopts the text of the poem *In Flanders Fields*, placing one full copy of the poem on each petal, as well as in the centre and in the background. The size of the text is automatically adjusted according to the area of each region. Randomized line thickness is applied inside the petals for enhanced visual effect.

The Mona Lisa image in Figure 10.4 and the portrait in Figure 10.5 exemplify black and white classic style micrography. Both of them use text from T. S. Eliot’s *Portrait of a Lady*. Note the use of text flow directions to achieve visual distinction between different regions.

In Figure 10.6, lines of text are used to fill regions as well as depict outlines, creating an interesting visual interplay between the two.

Figure 10.7 combines graphical content with a translation of the well-known Russian poem *Dark Eyes*, generating a compelling black and white image where font properties are used to visually separate image regions.

In Figure 10.8 we compare the results of using two different smooth fields to guide text layout over the same input image, specifically the cat picture from Figure 2.5. On the left is the original field used by Li et al. [26] to generate their mosaic pattern – a single field computed over the entire region of interest and fully aligned with the feature lines (in this case, the cat’s eyes and the outline of its face). On the right is our field, computed over a segmented version of the same input. Mosaic and text layouts share some similarities – both aim to fill a given region using anisotropic texture elements while conforming to feature lines. But while the approach of Li et al. was successful in the case of the mosaic, it generates an incoherent layout when used for text. In contrast, our result is more coherent. It also allows for clearer separation of the elements of the image, by varying the font size and text direction between regions. The differences become more pronounced in the black-and-white version (bottom).

Figure 10.9 shows two results created from the same input using different amounts of text, highlighting the flexibility offered to artists using our system. In this example, a very simple input is converted into a visually appealing, intricate design by overlaying the microgram with a colour layer specified by the user. Such effects are very easy to apply since our outputs are naturally presented as vector graphics.

Figure 10.10 presents variations on a theme in colour and typeface, as
Figure 10.2: “Summer Girl”. Image source: vector.net. Text: Song of Songs. Input image inset. Reprinted with permission of ACM.
Figure 10.3: “In Flanders Fields”. Image source: Wikimedia Commons. Text: *In Flanders Fields* by John McCrae. Input image inset. Reprinted with permission of ACM.
Figure 10.5: “Light and Shadow”. Image source by Ruby Mawira. Text: Portrait of a Lady by T. S. Eliot. Input image inset.
Chapter 10. Results

Figure 10.6: “Fedora”. Text and image source: Wikipedia. Input image inset. Reprinted with permission of ACM.
10.1 Running Times

The primary cost in our implementation involves the boundary condition computation, which requires solving a quadratic programming (QP) optimization problem. This step was initially implemented in MATLAB using the built-in active-set based QP solver [9]. A more efficient C++ implementation was later developed based on the Interior-Point Optimization (Ipopt) package [51], resulting in a sixfold performance improvement on average.

The two other computationally demanding steps are text rendering using the Adobe Text Engine, and the spacing adjustment step performed after streamline tracing. The latter relies on a large number of ray-intersection queries, and its cost can be reduced significantly by using techniques from
Figure 10.8: Top left: layout guided by the fully-aligned field of Li et al. Top right: Layout created by our method. Bottom: the differences between the two layouts become more evident in the absence of colour. Original image courtesy of Li et al. Text: *The Cat Came Back*, adapted from Harry S. Miller. Reprinted with permission of ACM.
Figure 10.9: In this example, different lengths of input text produce different results from an otherwise identical input. Text: Wikipedia. Reprinted with permission of ACM.
Figure 10.10: Endless variations on a topic can be produced by modifying the typeface, colour scheme, etc. Text: Genesis 1 (King James Version) and On the Origin of Species by Charles Darwin. Reprinted with permission of ACM.
Figure 10.11: Field design with user-defined constraints: the *Jolly Roger* flag at the top was created with automatic boundary constraints. It has long text lines and no singularities. With user preference for text alignment around the top and sides of the skull (shown in green), the field becomes less coherent (containing a singularity on the forehead), but the overall effect is arguably more appealing. Image source: Wikipedia. Text: *Derelict* by Young E. Allison. Reprinted with permission of ACM.
10.1. Running Times

<table>
<thead>
<tr>
<th>Microgram</th>
<th>Boundary</th>
<th>Spacing</th>
<th>Text</th>
<th>Other</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MATLAB</td>
<td>C++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fedora</td>
<td>102</td>
<td>27</td>
<td>19</td>
<td>13</td>
<td>61</td>
</tr>
<tr>
<td>Dark Eyes</td>
<td>284</td>
<td>38</td>
<td>11</td>
<td>18</td>
<td>79</td>
</tr>
<tr>
<td>Jolly Roger</td>
<td>215</td>
<td>56</td>
<td>12</td>
<td>28</td>
<td>109</td>
</tr>
<tr>
<td>Mona Lisa</td>
<td>189</td>
<td>37</td>
<td>51</td>
<td>23</td>
<td>122</td>
</tr>
<tr>
<td>In Flanders Fields</td>
<td>418</td>
<td>45</td>
<td>21</td>
<td>30</td>
<td>105</td>
</tr>
<tr>
<td>Beetle</td>
<td>340</td>
<td>83</td>
<td>36</td>
<td>13</td>
<td>142</td>
</tr>
<tr>
<td>Light and Shadow</td>
<td>706</td>
<td>104</td>
<td>65</td>
<td>25</td>
<td>209</td>
</tr>
<tr>
<td>Cat</td>
<td>403</td>
<td>68</td>
<td>36</td>
<td>83</td>
<td>231</td>
</tr>
<tr>
<td>California Republic</td>
<td>568</td>
<td>82</td>
<td>146</td>
<td>50</td>
<td>314</td>
</tr>
<tr>
<td>Summer Girl</td>
<td>130</td>
<td>23</td>
<td>271</td>
<td>93</td>
<td>481</td>
</tr>
<tr>
<td>Motif</td>
<td>4619</td>
<td>738</td>
<td>25</td>
<td>92</td>
<td>916</td>
</tr>
</tbody>
</table>

Table 10.1: Run time breakdown: boundary condition solvers; spacing adjustment; text rendering; rest of the algorithm; total running time when using the C++ boundary solver. All times are given in seconds. Note: in the *Summer Girl* image, the boundary solver was only invoked for the smaller regions.

real-time rendering and global illumination.

The rest of the algorithm requires a few seconds for simpler images and up to about a minute for the largest examples. Table 10.1 lists running times for the results presented in this chapter. All runs were executed on a dual-core Intel workstation equipped with the E6400 CPU running at 2.4 GHz.
Chapter 11

Summary

This thesis described an automated method for generating micrography, an appealing and intricate form of text art. Traditional hand-drawn micrography has been practised by artists for many centuries, but designing micrograms remains a highly technical, time-consuming task requiring significant expertise. In contrast, by using the digital design tool implemented as part of this work, such designs can be created from standard vector art in a matter of minutes. To the best of our knowledge this is the first attempt by the computer graphics community to tackle micrographic design.

The proposed method takes in a vector graphics image and text and produces a richly detailed, visually appealing and easily readable microgram. A range of artistic controls are provided to the user, including font styling and local directional constraints. The flexibility and applicability of the system are demonstrated in Chapter 10 using a wide variety of finished designs, ranging from classic style black and white imagery to composite works that take full advantage of the digital platform.

Our main technical contribution is an algorithm that uses smooth fields to generate text layouts inside a closed planar shape. Field-guided methods are by now well-established in related applications, e.g. parameterization, quadrangulation and texture synthesis, and we extend the paradigm to cover text line patterns as well. However, this work goes beyond standard field-guided techniques by presenting a principled approach to boundary condition design for smooth fields in 2D. Through careful analysis and specification of boundary conditions, we are able to shift much of the complexity away from field computation and into a separate preprocessing step, which is smaller in scale and better suited to the kind of reasoning we want to apply. In the case of micrography this approach helps produce fields that balance aesthetic and readability requirements, aligning with the boundary where possible while remaining largely free of singularities and high curvature. The same ideas can potentially transfer to other applications that use smooth fields as a design tool, such as anisotropic texturing and meshing.
Chapter 12

Future Work

This work presents opportunities for further development in two primary areas: text art, and boundary condition design.

On the text art front, it would be interesting to map and expand the range of styles supported by the proposed framework. While most micrograms can be generated using piecewise smooth fields as done in our system, there are examples where this approach requires an unintuitive partition of the image into regions. Modern creations in particular sometimes include elements such as partial text overlap, pseudo-3D effects, or interleaving of different text flow directions, none of which fit well into the existing algorithm. It would also be instructive to work closely with designers of micrography – hand-drawn as well as digital – to observe what kinds of control mechanisms are most helpful to them in expressing and producing the result they are after.

There is clearly room for improvement in the text placement and rendering mechanism of our system. In the method presented, local properties of the text only come into play during the very last step – an approach that has benefits as well as disadvantages. On the one hand, a key goal of our system is to let artists focus on high-level design rather than devote their attention to individual words and letters. On the other hand, ignoring text details until later in the process means that these details cannot inform the layout process. Better text placement – leveraging knowledge about the shape of individual words and letters – could help reduce artifacts, achieve more uniform density, and improve readability.

A performance boost is needed if our implementation is to mature into a practical, user-friendly tool. Ideally we would like to provide artists with real-time responses to changes in their design. This requires a combination of engineering efforts and a revisit of some of the core algorithmics. In particular, the boundary formulation is a non-convex search problem. Although our method performs the search quite efficiently, an alternative convex formulation for the boundary condition problem would be an important contribution.

Beyond text art, a number of applications could benefit from the bound-
ary condition design paradigm proposed in this work. Quadrangulation and uv-parameterization methods typically aim to follow feature lines, minimize curvature, and avoid singularities. In the 2D case our boundary formulation could probably be applied to these problems as it is. The broader concept, whereby parts of the boundary are aligned and others relaxed according to a set of custom criteria, is applicable to a host of other planar layout tasks, and could perhaps be extended to feature-aligned designs in 3D as well.
Bibliography


[29] The MathWorks Inc. MATLAB [computer software], 2010. version 7.11.0 (R2010b).


