Tracing on the Right Side of the Brain: Unsupervised Image Simplification and Vectorization

by

Sven Crandall Olsen

Bachelor of Arts, Swarthmore College, 2003
Master of Science, Northwestern University, 2006

A Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of

Doctor of Philosophy

in the Department of Computer Science.

Supervisory Committee

Tracing on the Right Side of the Brain:
Unsupervised Image Simplification and Vectorization

by

Sven Crandall Olsen

Bachelor of Arts, Swarthmore College, 2003
Master of Science, Northwestern University, 2006

Supervisory Committee

Bruce Gooch
Department of Computer Science
Supervisor

Amy Gooch
Department of Computer Science
Departmental Member

Brian Wyvill
Department of Computer Science
Departmental Member

Wu-Sheng Lu
Department of Electrical and Computer Engineering
Outside Member

Abstract

Supervisory Committee

Bruce Gooch, Supervisor
Department of Computer Science

Amy Gooch, Departmental Member
Department of Computer Science

Brian Wyvill, Departmental Member
Department of Computer Science

Wu-Sheng Lu, Outside Member
Department of Electrical and Computer Engineering

I present an unsupervised system that takes digital photographs as input, and generates simplified, stylized vector data as output. The three component parts of the system are image-space stylization, edge tracing, and edge-based image reconstruction. The design of each of these components is specialized, relative to their state of the art equivalents, in order to improve their effectiveness when used in such a combined stylization / vectorization pipeline. I demonstrate that the vector data generated by this system is often both an effective visual simplification of the input photographs, and an effective simplification in the sense of memory efficiency, as judged relative to state of the art lossy image compression formats.

Many recent image-based stylization algorithms are designed to simplify or abstract the contents of source images; creating cartoon-like results. An ideal cartoon simplification preserves the important semantics of the image, while de-emphasizing unimportant visual details.

In order to fully exploit image simplification in a software engineering context, an abstracted image must be "simpler" not just in terms of its apparent visual complexity, but also in terms of the number of bits needed to represent it. At present, the most robust image abstraction algorithms produce results that are merely visually simpler than their source data; the storage requirements of the "simplified" results are unchanged.

In contrast to computationally stylized images are vector-graphic cartoons, created by a human artist from a reference image. Vector art is more easily modified than bitmap images, and it can be a more memory efficient image representation. However, the only reliable way to generate vector cartoons from source images is to employ

a human artist, and thus the advantages of vector art cannot be exploited in fully automatic systems.

In this work, I approach image-based stylization, edge tracing, and edge-based image reconstruction with the assumption that the three tasks are synergistic. I describe an unsupervised system that takes digital photographs as input and uses them to create stylized vector art, resulting in a simplification of the source data in terms of bit encoding costs, as well as visual complexity. The specific algorithms that comprise this system are modified relative to the current state of the art in order to take better advantage of the complementary nature of the component tasks. My primary technical contributions are:

1) I show that the *edge modeling problem*, previously identified as one of the fundamental challenges facing edge-only image representations, has a relatively simple and robust solution, in the special case of images that have been stylized using aggressive smoothing followed by soft quantization. (See Section 2.4.2.)

2) In Chapter 4 I introduce a novel edge-based image reconstruction method, which differs from prior work in that anisotropic regularization is used in place of a varying width Gaussian blur. While previous vector formats have successfully used variable width blurring to model soft edges, the technique leads to artifacts given the unusually large widths required by the traced vector data. My anisotropic regularization approach avoids these artifacts, while maintaining a high degree of reconstruction accuracy. (See Sections 4.2 and Figure 4.18.)

3) I demonstrate that the vector data generated by my system is, in the sense of memory efficiency, significantly simpler than the input photographs. Specifically, I compare my vector output with state of the art lossy image compression results. While my vector encodings are in no sense accurate reproductions of the input photographs, they do maintain a sharp, stylized look, while preserving most visually important elements. The results of general purpose compression codecs suffer from significant visual artifacts at similar file sizes. (See Sections 3.2, and 5.2.)

Contents

List of Figures

Notation

I have attempted to use notational conventions that are as consistent as possible with related publications in computer vision, graphics, and applied math. With few exceptions, italic lowercase letters denote scalars while bold lower case letters denote vectors. Capital italic letters are used to denote a relatively wide range of objects, though most often they refer either to matrices or image data. Thus, $\mathbf{x}$ is a vector, x_i is an element of $\mathbf{x}$, and $A\mathbf{x}$ is a matrix vector product.

Additionally, I use the following mathematical shorthand:
$\mathbb{R}$: the set of real numbers.
$\mathbb{R}^+$: the set of positive real numbers.
$\mathbb{Z}$: the set of integers.
$\mathbb{Z}^+$: the set of positive integers.
C^n: the set of functions that are continuous in derivatives 0 through n.
$\forall i > 1$: for all $i > 1$.
$\exists x > 1$: there exists x such that $x > 1$.
$i \in \mathbb{Z}^+$: i is an element of the set of positive integers.
$i \ni \mathbb{Z}^+$: i is not an element of the set of positive integers.
$\{x \mid x < 2\}$: the set of all x such that $x < 2$.
$\mathbb{R} \times \mathbb{R}$: the set of all ordered pairs of real numbers, i.e. $\{(x, y) \mid x, y \in \mathbb{R}\}$.
$\mathbb{R}^n$: the set of all n-tuples of real numbers, i.e. $\mathbb{R} \times \mathbb{R} \times ... \times \mathbb{R}$.
$\mathbf{f} : \mathbb{R} \to \mathbb{R}^2$: $\mathbf{f}$ is a function that maps $\mathbb{R}$ to $\mathbb{R}^2$.
$\lfloor x \rfloor$: the largest integer which is less than or equal to x.
$\lceil x \rceil$: the smallest integer which is greater than or equal to x.

Square brackets and parentheses are used to define intervals of $\mathbb{R}$, for example,

$$[0, 1) = \{x \mid x \geq 0 \quad \text{and} \quad x < 1\}.$$

When they enclose a predicate, square brackets define indicator functions, thus,

$$[x \geq y] = \begin{cases} 1 & \text{if } x \geq y, \\ 0 & \text{otherwise.} \end{cases}$$

(The use of square brackets to define indicator functions is known as Iverson Notation, and was popularized by the authors of *Concrete Mathematics* [29].)

1 Introduction

You should call it 'entropy', for two reasons.
In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage.
– John von Neumann to Claude Shannon, on the topic of information theory.

This dissertation describes an image simplification system, inspired by the artistic traditions of cartoons and pencil sketches. The stored image data is designed to be reproduced at arbitrary resolution, and is composed exclusively of parametric curve data.

In the domain of digital art and design, artists will often create cartoon-like drawings from a source photograph by tracing the outlines of all the shapes in the scene. Image editing programs such as Adobe Illustrator include several tools that can aid in tracing tasks. Using these tools, it is possible for a user to convert an input bitmap image into a vector format cartoon drawing. For example, a PostScript cartoon might be created from a digital photograph stored in JPEG format. In the graphic arts community, the process of converting a bitmap image to a vector cartoon is referred to as *vectorization*.

The art of vectorization is related to that of drawing, as, given an input photograph, an artist must decide how to best decompose the photo into a collection of closed regions. As Edwards advises in *Drawing on the Right Side of the Brain*, one of the fundamental components of learning to draw is mastering the related task of decomposing a scene into areas of shadow and highlight [19].

In computer vision and computer graphics, there is a long tradition of using the precedent of visual art, most commonly line drawings or cartoons, to support the argument that only a very small portion of the information contained in most photographs is visually important. Thus, algorithms for performing low level vision tasks like edge detection or image segmentation, or algorithms for applying artistic effects such as cartooning, are frequently motivated in terms of the need to discard the visually unim-

portant information that exists in any input photograph.

However, while the precedent of art has played a major role in influencing the development of algorithms in computer graphics and computer vision, the results of those algorithms remain inferior to the level of visual simplification and data efficiency that can be achieved by a skilled artist.

Automatic object tracing algorithms exist, and some have proven to be relatively robust solutions to the problem of converting input bitmap data into vector format curves. Such automatic tracing algorithms are not used to simplify input images, but, rather, to convert them to a resolution independent format, and allow them to be more easily manipulated as components of vector graphics editing programs.

To imitate the vector cartoons that can be created by a human artist, an algorithm would need to first perform a visual simplification of the input photograph, and then convert that visual simplification into a vector format. Such algorithmic imitations are possible, using state of the art tools. The algorithm can begin by running an image space stylization filter on an input photograph. This stylization may be one of the cartoon or pencil sketch filters included with Adobe Photoshop [71], or any of the more recent image space stylization filters that have appeared in the computer graphics literature. The result can then be input to an automatic tracing system such as *Vector Magic*[1]. However, the curve data returned by high quality automatic tracing systems tends to be information dense, and thus, the file size size of the vector output will often exceed that of the bitmap input.

The image simplification system I present is composed of both an image space stylization algorithm and an automatic curve tracing algorithm. These two component algorithms are specialized, relative to their state of the art equivalents, in that they are treated as pieces of a single simplification pipeline, rather than independent algorithms meant to be applied to arbitrary inputs. These specializations greatly improve the memory efficiency of the resulting vector data. The results have a recognizable artistic look—somewhere between a cartoon and a pencil tone study.

The very low encoding cost of my vector format is interesting from an academic perspective, as it provides a concrete example of the link between art and information efficiency. While this link has often been hypothesized in both computer vision and computer graphics, rarely has it been so convincingly demonstrated [51, 41, 18, 48, 34]. On some images, my system generates attractive vector art that has significantly lower bit costs than those that can be achieved with state of the art lossy compression techniques. As such, it improves by several orders of magnitude the compression results reported for ARDECO, the most closely related joint vectorization / stylization sys-

[1] http://vectormagic.com

tem [68].

The image space stylization algorithms described in Chapter 2 introduce some novelties relative to the state of the art, and I suspect that many digital artists may find it an improvement, relative to current image stylization filters. However, while my stylization algorithm anticipates its use in a vectorization pipeline, it does not require it, and the results of vectorization are rarely more attractive than the intermediate stylized images.

Given the efficiency of modern lossy image compression formats, and the tremendous amounts of storage space available on most computers, there is little benefit to using my vector format to store local image data. In today's computing environment, there is little practical difference between a 2kB image file and a 20kB image file. Even so, my format could prove useful in web applications. Converting the vector data to Flash or SVG would be possible, though doing so would both reduce the quality of the image data and its memory efficiency, as those formats do include a means of representing the edge shading data generated by my algorithm. With some adjustment, the core vector content could prove useful for web designers creating pages meant to load under bandwidth limited conditions. In the near term, an algorithm similar to Sun et al.'s could likely be used to convert my vector content to the more widely supported gradient mesh format [70]. And, in the more distant future, it may be possible to expand the vector formats supported on most web browsers to allow smooth shaded images more like those generated by my system.

I am also hopeful that the image vectorization system presented here could be expanded to the case of video. Most of the component algorithms have equivalents designed to work on video data, and the ability to quickly transmit stylized video data over low bandwidth connections would likely have even more applications than the vector image case.

1.1 Background

The idea that image stylization, simplification, and edge tracing should be approached as complementary tasks has a long history in both computer graphics and computer vision.

In their seminal work on the theory of image segmentation, Mumford and Shah cited the ability of artists to capture most important image information in simple cartoon drawings as evidence that it should be possible to create image segmentations that contain most of the semantic content present in natural images [51].

Leclerc went further than most other early computer vision researchers, and hy-

pothesized that the most efficient possible representation of natural images would be as the sum of a piecewise smooth image and noise data [41]. Using this assumption, he was able to leverage Shannon information theory to derive a set of Bayesian priors for images, and demonstrated that those in turn could be used to improve the performance of edge detectors and segmentation algorithms.

Much of the continuing work on the topic of image segmentation has been heavily influenced by either Mumford and Shah or Leclerc, and a number of the most relevant developments are discussed in Section 1.2. I have adopted several concepts from image segmentation for use in my own system. The energy functionals described by Mumford and Shah are one of the main inspirations for my own approach to image reconstruction, and forms of anisotropic diffusion are used when creating both the initial image stylization, and when rendering the vectorized result.

Inspired by the segmentation work that preceded him, Elder hypothesized that a sparse, edge-only image representation could be used to store all the visually important content of most natural images [20]. Elder developed an image format that contained only edge locations and edge gradient samples, and demonstrated that it was possible to reconstruct high quality grayscale images from that data. However, this format did not show competitive memory efficiency, when compared with more conventional lossy image encodings.

An interesting variation on Elder's edge only format was developed by Orzan et al., who introduced *diffusion curves* [52]. The primary purpose of the diffusion curve format is to enable artists to more easily construct soft-shaded vector art. The underlying edge data is thus parameterized by splines, in contrast to Elder's use of point samples. The image reconstruction method remains closely related to Elder's, although modifications have been made to support the presence of color. Diffusion curve vector data can also be generated automatically from source photographs, though the consequences of this process for either memory efficiency or visual fidelity relative to the source photograph have not been studied.

In 2002, DeCarlo and Santella described a system for converting input photographs to cartoon-like images [18]. The goal of this system was to simultaneously simplify and clarify the contents of an image. It operated by combining eye tracking data with mean shift segmentations and a b-spline wavelet analysis of edge lines. The resulting images were qualitatively simpler than the source data, but also appealing when considered as works of digital art.

In 2006, Lecot and Lévy developed ARDECO, a combined image stylization / vectorization system [42]. Ardeco operates by combining a Mumford-Shah energy minimization with a sequence of increasingly simplified triangle meshes, which are converted

to spline boundary curves at the end of the process [42]. In an approach similar to Elder's edge image reconstruction, adaptive blurring is used to model soft edges between regions. A study of the memory efficiency of Ardeco's vector output showed that the system could, under some circumstances, outperform JPEG encoding, but the compression results were less competitive relative to the more modern JPEG2000 standard [68].

The image-space stylization filter presented in Chapter 2 benefits from the many recent computer graphics papers that have advanced the art and science of image space stylization. In particular, I make use of the ability of difference-of-Gaussians filtering to effectively simplify and abstract facial features, something first noted by Gooch et al. [27]. Variations on difference of Gaussian filtering that allow a wider range of artistic effects and higher quality results have since been developed by Winnemöller et al., Kang et al., and Kyprianidis and Döllner [76, 35, 39]. The flow-guided filters introduced by Kang et al. have proven very useful in creating high quality stylizations for use as input to the vector tracing and reconstruction algorithms.

1.2 Segmentation

Historically, image and video segmentation algorithms have been grouped into one of several competing paradigms. Three of the most popular categories have been Mumford Shah region energy minimizations [51], active contour spline fitting [37], and global optimizations derived from Bayesian or minimal description length (MDL) criteria [25, 41].

Image segmentation is closely related to the task of edge detection, as the most significant edges in an image are typically those that occur on the boundaries between regions, while the most desirable segmentations are those in which most of the region boundary lines are also edges. Christoudias et al. argued that the two tasks of edge detection and image segmentation are naturally synergistic—and that the best approach to both problems was to integrate local image information obtained from edge detection into global methods for image segmentation [11]. Many modern approaches to computer vision problems are arguably examples of such a "synergistic" approach. For example, anisotropic diffusion was originally proposed as an edge finding method but is now often used as a component in a variety of low level vision tasks, including image segmentation, texture classification, and optical flow approximation [53, 5].

In 1996, Zhu and Yuille argued that most approaches to image segmentation could be encompassed by a single unified optimization framework, and proposed an efficient strategy for solving any segmentation problems in this broad category, which they

called *region competition* [79]. Since then, several new video segmentation algorithms have been defined by extending the region competition framework to additionally account for optical flow constraints [16, 7].

1.2.1 Parametric Models

In 1988, Kass et al. introduced *active contours* as a low-level computer vision tool [37]. Also known as *snakes*, active contours are parametric curves that locally adjust their control points in order to seek a minimum energy state. By convention, the energy functional used by a snake is divided into two parts: the internal energy, which is typically defined to favor straight, smooth curves, and the external energy, which is defined so as to attract the curve to certain features in the image, most typically edges. For a parametric curve $C(t)$, and a greyscale image defined by the intensity function $I((x))$, the standard snake energy functional is,

$$E(C) = (\mu_1 \int_0^1 |C(t)'|^2 dt + \mu_2 \int_0^1 |C(t)''|^2 dt) - \int_0^1 ||\nabla I(C(t))||^2 dt.$$

If the curve $C(t)$ is open, the snake will match an edge in the image, if C is closed, the snake will form the border of a closed region. The μ terms may be used to control the relative weightings of the curve's first and second degree smoothness.

An additional external term may also be added to take into account input supplied by either a user or a high level computer vision algorithm. The external energy term may also be adjusted to react to intensity gradients measured at multiple levels of scale space, in order to better match blurry region boundaries. Because the snake performs only local optimizations, and fits itself to the closest minimal energy state, active contour edge finding is very fast. However, for the same reason, traditional active contours cannot be considered an automatic segmentation technique. They require the output of other algorithms or user input to supply them with useful initial conditions. In the case of closed snakes, one simple method for reducing the high dependence on initial conditions is to add an energy term penalizing the snake for enclosing a small area. Thus, closed curves will tend to expand, and seek the boundaries of large regions. Such methods are known as balloon models [13].

Snakes in Video Segmentation

The sensitivity of snakes to their initial placement has made them attractive to researchers interested in video segmentation. Once a snake has been matched to an object in one frame of video, the minimal energy state that it finds can be used as an

initial condition for the next frame. Thus, a naive application of snakes to video segmentation can provide a very fast and simple region tracking algorithm. Unfortunately, the approach has proven to be quite brittle [50].

One significant source of problems are region topology changes. If the segmentation of an image into regions is mapped to a graph, with each closed region implying a node, and edges inserted between all regions adjacent to one another, then that graph will often remain constant throughout several frames of a video sequence. However, some common events will cause the topology of the region adjacency graph to change. If one object moves to partially occlude another, the occluded region may be split into two distinct closed regions. A traditional active contour model has no way of dealing with such a topology change—the enclosing curve will typically be forced to choose to track one closed region or the other.

Two different strategies have been suggested for adapting active contour models to better handle topology changes. In 1995, McInerney and Terzopoulos described a segmentation system in which the active contour curve evolution step was alternated with a curve reparameterization step [49]. After each update of the curve's control points, the active contour was projected onto a grid. This grid was used to re-parameterize the curve, and, if necessary, split or join regions in the case of topology changes. The authors named this method topologically adaptive snakes, or t-snakes, and it has since been successfully applied to the case of tracing volume data in medical datasets [50].

A very different strategy was proposed by Caselles et al. in 1997 [9]. A potential energy field was defined such that its zero level set would roughly match the behavior of a classical snake energy. As the method did not require an explicit parametrization of the boundary curve, it had no difficulty handling region topology changes. The authors named the method *geodesic active contours*, because similarities existed between their energy functional and the laws of relativistic dynamics. Unfortunately, optimizing the potential function proved much more computationally intensive than optimizing the control points of a parametric curve, and geodesic active contour methods can require several minutes to process a single image.

1.2.2 Region Energy Minimization

While active contour methods optimize an energy defined only for the boundary points of a region, other methods have been proposed which take into account the character of the pixels inside each region. The Mumford-Shah functional, proposed 1989, is a popular means of defining such an energy [51].

The Mumford-Shah energy functional is defined in terms of a source image g, and output image f [51]. Both f and g are considered to be scalar functions defined in the

domain $\Omega \subset \mathbb{R}^2$. In addition to creating an output image f, which may not be identical to the input image, g, the Mumford Shah energy functional is also defined in terms of a segmentation of Ω into N closed regions R_i. The boundary curves between those regions are defined as Γ, and the combined length of all boundary curves denoted by $|\Gamma|$. The Mumford Shah energy functional is defined as,

$$E(f,\Gamma) = \mu^2 \iint_{\Omega} (f-g)^2 dxdy + \iint_{\Omega-\Gamma} ||(\nabla f)||^2 dxdy + \nu|\Gamma| \tag{1.1}$$

The energy functional can be interpreted as a formalization of the following two goals:

- The values in the output image should be close to the values in the source image. However, the output values should not vary much inside a given region.
- The region boundaries should be as simple as possible.

The boundary length $|\Gamma|$ can be understood as a measure of the complexity of the region boundaries, thus, the scalar ν can be used to control the relative weighting of the boundary simplicity goal. The scalar μ controls the relative weight of source image matching.

When the functional was first introduced, Mumford and Shah proved two important theoretical results. The first is that, for any fixed Γ, the output image f is always uniquely determined by the following Poisson equation:

$$\text{Inside } R_i,\ \nabla^2 f = \mu^2(f-g), \text{ and on } \partial R_i,\ \frac{\partial f}{\partial n} = 0.$$

Based on this result, Mumford and Shah introduced the concept of the *cartoon limit*. As μ^2 goes to 0, the $(f-g)^2$ term becomes vanishingly small relative to the $||(\nabla f)||^2$ term[2]. Thus, $f(x,y)$ will be forced to take on a constant value inside each R_i. It can be shown that the optimal constant color value inside each R_i will be the mean color of g in R_i. This piecewise constant special case is called the *cartoon limit*.

Mumford and Shah's second important theoretical result is that it is possible to can characterize the extrema of Γ well enough to refine any Γ using standard nonlinear optimization techniques. Specifically, small perturbations in Γ imply that the total energy of the segmentation will change relative to the changes in curvature measured over a parametrization of Γ.

[2]One reason to consider the limit st $\mu^2 \to 0$, rather than simply setting μ to 0, is that, strictly speaking, f is not "uniquely determined" by the Poisson equation given above, in the case that $\mu = 0$. However, there is always a unique solution to as long as $\mu > 0$, and that solution converges to the constant color case as $\mu \to 0$.

Despite the two significant theoretical results listed above, in practice, finding optima of the Mumford-Shah energy functional tends to be difficult. If it is assumed that there are only two distinct regions in the image, then a potential-based level set method, like that used in geodesic active contours, can be used to optimize the functional. Handling the case of multiple regions is more difficult. Vese and Chan have proposed a method whereby $\log(n)$ potential functions can be used to implicitly encode n different regions [73]. Methods such as this, in which multiple regions are found using a collection of connected level set optimizations, are known as *multiphase level set* optimization frameworks.

1.2.3 Bayesian Segmentation

In its simplest form, Bayesian segmentation considers the input image D to be a corrupted version of some model image M_i, where the model image can be any piecewise smooth image corresponding to a segmentation of the scene pixels [25]. From Bayes' rule, it follows that the most likely model image is the one that maximizes the product of the product of the two probability functions, $P(D|M)$ and $P(M)$. Defining the probability of the source image given a particular model image is straightforward, it requires only that we define a model for the corrupting process that generated D from M. Typically, a Gaussian error distribution is used for this purpose. However, defining the prior probability of a piecewise smooth image, $P(M)$, presents a problem. A priori arguments to the effect that one type of model image should be considered more likely than another are hard to justify. Generally, the prior probability of a model is chosen to reflect a particular assumption of about what a good segmentation ought to look like, for example, if we wish to avoid segmentations having many small regions, $P(M)$ may be defined as a function that decreases with the number distinct regions in the image.

MDL Segmentation

The minimum description length segmentation method, introduced by Leclerc in 1989, provides an alternative means of justifying the Bayesian approach, one that allows the prior probability of a given model to be defined in a less arbitrary manner [41]. Leclerc makes use of a result from information theory, which states that if probability functions governing the likelihood of model image corruption and model image prior probability exist, then there must also exist optimal languages for encoding those images, such that the number of bits needed to encode the model image M is $-\log_2(P(M))$, and the number of bits needed to encode the pattern of corruption is $-\log_2(P(D|M))$. Assuming that an image is most efficiently described by first specifying the piecewise smooth

segmentation M, and then describing the pattern of corruption $D - M$, it follows that the most likely model image for a given input source image is the segmentation that can be used to create a minimal bit count description of that image. Thus, the prior probability of a given model image can be derived from the number of bits needed to describe it, assuming the use of an optimal language.

Leclerc thus recommends building segmentation algorithms by combining some definition of $P(D|M)$, which can be thought of as a general specification of the behavior expected inside each region, with a "language" for representing a given piecewise smooth segmentation. While in principle, such a language would be equivalent to an encoding scheme for the image data, in practice, it need only be a way of estimating the number of bits needed to store the model segmentation under ideal conditions.

1.2.4 Unified Models

In 1996, Zhu and Yuille argued that, if Leclerc's approach were extended to consider an images as a continuous field of intensity values, rather than discrete data points, then most other popular segmentation algorithms could be considered as special cases of a generalized MDL method [79]. For example, in the Mumford-Shah energy functional, equation (1.1), the segmentation encoding cost becomes the $\nu|\Gamma|$ term, while the smoothness terms define the expected internal region behavior. The notion of a closed active contour was argued to be equivalent to the special case in which corruption inside each region is assumed to uniformly distributed, and the cost of encoding segmentation edges varies as a function of the underlying image gradients.[3] Zhu and Yuille then went on to introduce a novel algorithm for solving the difficult optimization problem implied by their generalized MDL criterion. This algorithm worked by alternating between updating region boundries and updating the internal parameters that described each region. The algorithm also included provision for performing region merges, and creating new regions.

At least two different joint segmentation and optical flow finding algorithms have been proposed as extensions to Zhu and Yuille's general framework. The first was presented by Cremers et al. in 2005, and provided both a parametric binary segmentation implementation and an multiregion segmentation, based on Chan and Vesse's multiphase level set representation [16]. The second was proposed by Brox et al. in 2006 [7]. The Brox et al. system did not include the option of using parametric curves,

[3] At first glance, specifying the prior probability of the model image M in terms of the gradients in the source image D seems like a blatant violation of what it means to be a 'prior' probability. However, it is not necessarily as bad as it seems, as there might well be an extent to which the distribution of gradients in the source image is representative of the distribution of edges in all ideal image segmentations.

and it made use of a different multiphase level set framework. The 2006 system also produced significantly better results than the 2005 system.

In both these systems, the segmentation's energy functional was modified to include optical flow energy terms, such as those defined by a Horn/Schunck solver [32]. A minimum of the functional thus defines not only a segmentation of each frame, but also flow fields that specify the apparent motion between frames. In Brox et al., a multiscale optimization process was used, in which region competition was performed at successively fine levels. This multiscale optimization was similar to that used by the same authors in their 2004 paper on calculating optical flow [6]. The different goals provided by the combined optical flow and segmentation energy functional proved to be synergistic; the region definitions found appeared better than those that could be found by a per-image segmentation algorithm, while the optical flow fields reported by Brox et al. are higher quality than those that could be found using any prior optical flow algorithms [7]. In addition to breaking many previous records for accuracy in computed flow, the Brox et. al system also proved so accurate that the authors uncovered errors in the ground truth data used to evaluate performance on the Yosemite test dataset [7].

1.2.5 Graph Optimization

All of the segmentation methods that can be encompassed by Zhu and Yuille's unified framework fall under the broad heading of continuous energy minimization approaches. This is to say, they require that the discrete image data be treated as a continuous field of intensity or color values, upon which the tools of multivariable calculus can be brought to bear. In contrast to this, are the original, non-continuous formulations of Bayesian and MDL segmentations, which consider image data as a discrete grid of scalar values. There can be a computational advantage in leaving the image data in a discrete form, as segmentation algorithms can then be defined in terms of optimal paths through the pixel connectivity graph, rather than as the minimizers of integral expressions. Such natively discrete, graph based methods have proven very successful in some application domains. In 2003, Kwatra et al. used graph optimizations to find regions in a source image for the purposes of texture generation [38]. In 2004, Rother et al. showed that a mincut graph algorithm could prove useful in the the case of user guided binary segmentation [62]. In 2006, Schoenemann et al. used a similar graph cut algorithm to perform fast per-frame binary segmentations of video data [64].

1.3 Multiresolution Curves

Wavelets have proven a very useful tool for image simplification and compression tasks. The application of wavelet analysis to images was pioneered by Mallat [46] and Daubachies [17], leading to an explosion of research in multiresolution imaging applications [10]. Wavelet decompositions have become an important component of modern image compression standards, such as JPEG2000 [1].

Wavelets have also been applied to the problem of simplifying vector curve data [69]. In 1994, Finkelstein and Salesin introduced the graphics community to the use of B-spline wavelets for curve analysis [23]. Also known as Chui wavelets or B-wavelets, B-spline wavelets are constructed from a nested space of B-spline scaling functions, and had recently become popular among mathematicians studying wavelet theory [12]. Finkelstein and Salesin demonstrated that B-wavelets had many potential applications in computer graphics. Decomposing a hand drawn curve into detail and low-resolution components could allow an artist to separate the line style from the shape of the curve. Thus, it was possible to replace the line style from one drawing with that of a different drawing. Postscript curve data could also be automatically simplified before being spooled to a printer, minimizing the transmission cost required to print detailed vector art.

As the b-wavelet basis functions are only semi-orthogonal, separating the detail and low-resolution components of a b-spline can be computational intensive. A naive decomposition algorithm, which computes the necessary analysis matrices for each resolution, will require two $n \times n$ dense matrix multiplications per decomposition. Given input curves with thousands of initial control points, storing and applying these large matrices can become a significant computational burden. However, Finkelstein and Salesin recommended making use of the linear time decomposition algorithm proposed by Quak and Weyrich [57], in which the properties of the b-wavelet's dual space were used to derive decomposition matrices defined in terms of the inverse of sparse banded matrices. As those sparse matrix inversions can be performed in linear time, the entire B-wavelet decomposition may be computed relatively quickly. This fast decomposition method is limited, however, to the case of end point interpolating b-spline wavelets. Periodic B-splines are a more natural tool for representing closed curves. An accelerated decomposition method for periodic b-wavelets, based on fast Fourier transforms, has been proposed by Plonka and Tasche [54].

In 1995, Gortler and Cohen demonstrated that using a B-wavelet basis could improve on the performance of finite element solvers applied to the task of finding minimal energy curves, such as those used in active contour models [28]. They also de-

scribed how the curve could be adaptively refined or simplified during the course of the optimization, by altering the resolution of different areas using an appropriately designed oracle. Such an oracle will tend to produce a minimal energy spline curve represented using the smallest possible set of control points.

1.4 Tracing Binary Images

In the computer vision literature, algorithms that identify regions in an image either by assigning each pixel a region ID, or by finding parametric curves that describe region boundaries, are both referred to as *segmentation* techniques. However, from an artist's perspective, the difference between blocky pixel labeling information and spline curves is significant.

Inferring a set of boundary curves from a binary image may be seen as a trivial interface extraction problem. However, given the end goal of generating vector data, the nature of the extracted boundary curves, both in terms of memory efficiency and less easily measured aesthetic qualities, is important. To the best of my knowledge, the computer graphics literature contains no references on the topic of extracting visually attractive boundary curves from segmented bitmap data, though Finkelstein and Salesin have addressed the issue of improving the memory efficiency of spline curve drawings [23]. However, sophisticated algorithms designed for the express purpose of generating attractive boundary curves from binary image data do exist. Currently, the most effective of these is considered to be the proprietary algorithm used by the website *Vector Magic*. The most popular documented tracing algorithm is curve extraction program *Potrace*, an open source project developed by the mathematician Peter Selinger [65].

The Potrace algorithm generates PostScript curve data from a binary input image. The algorithm proceeds as follows: First, the marching squares algorithm is used to generate lists of boundary pixels. Next, a graph based minimization finds a minimal vertex polygon having edges contained inside an area defined by the boundary pixels. A second minimization is then used to generate curves from the polygon data. The output curves are piecewise cubic splines, and special attention is paid to detecting and preserving sharp corners. The algorithm is relatively fast, with typical runtimes of less than a second. The bottleneck is the polygon finding energy minimization, which uses a relatively expensive method to find a global minimum of a graph optimization problem. If a faster version of the algorithm were required, this global optimization step could be replaced by a cheaper heuristic.

The vector editing program *Inkscape* includes an automatic image vectorization tool

based on Potrace. In order to use Potrace for vectorization, it is necessary to first create a segmentation of the input image, and then use Potrace to generate curves data for each of the resulting regions. Typically, there will be areas of overlap as well as gaps between the output curves for the different segmentation regions. To avoid this problem, Inkscape generates a hierarchy of increasingly coarse segmentations, and renders the curves found for the finer segmentation on top of those found for the coarser levels.

1.5 Image Morphology

While not directly related to the task of generating vector cartoons, morphological skeletonization is an important precedent for the more general problem of developing highly memory efficient simplifications of image data. Morphological skeletonization is a technique in which a sequence of morphological operations are applied to a binary image in order to reduce it to handful of points or lines in the centers the image regions. These operations may then be reversed to create an output image. In 1987, Maragos and Schafer [47] proposed using morphological skeletonization to compress binary video data. Binary image skeletonization can be performed very quickly, and the resulting skeletons can be easily compressed, thus Maragos and Schafer argued that the algorithm would be suitable for low-bandwidth telephony for the deaf. However, their high sensitivity to noise has thus far prevented skeletonization techniques from being widely applied.

1.6 Scattered Data Interpolation

Scattered data interpolation is the task of interpolating scalar or vector valued data for all points in a domain, given a set of irregularly distributed, unordered control points. Reconstruction of smooth images from edge only image data is a scattered data interpolation problem.

Common scattered data interpolation techniques include radial basis functions, membrane energy minimizations, and mesh based techniques [24]. Thin plate splines are an interesting special case, as they can be expressed either as a radial basis technique, or a tension-based energy minimization [72]. Depending on the method, the computational complexity of a scattered data technique may be determined by either the size of the domain over which it is applied, or the number of control points present. In 1984, Terzopoulos demonstrated that the potential to apply multigrid solvers to the partial differential equations implied by the tension-energy form of the thin plate

spline equations made very fast surface reconstructions from scattered depth samples possible [4, 72]. In 1997, this same technique was adapted to the domain of stylized video filtering by Litwinowicz, who used it to generate smooth brush stoke alignment fields [44]. However, the generated alignment fields had a tendency to change dramatically from one frame of video to the next. To overcome that problem, Hays and Essa suggested switching to a radial basis method, defined in video volume space rather than image space [30].

1.7 Video Vectorization

Many different tools exist for creating cartoon animations from captured video. The most commercially successful of these is Rotoshop, a software system developed by Robert Sabiston, and marketed by Flat Black Films. Rotoscoping is a traditional animation technique in which an artist traces the outline of an object in each frame of a source film. As most video postprocessing is now performed digitally, the terms 'rotoscoping' and 'vectorization' are sometimes used interchangeably, though it is still typically the case that rotoscoping implies tracing one or more objects of interest, while vectorization implies generating curve data for all elements of the input. Rotoshop provides artists with tools that simplify the rotoscoping process. It has been used to generate cartoon-like visual effects in several Hollywood films, including *Waking Life* and *A Scanner Darkly*.

In the academic literature, Agarwala et al. have described a software tool for simplifying the rotoscoping process [2]. It includes two components: a per-frame tool that allows artists to specify object outlines by adjusting a small set of control points, and a temporal optimization capable of generating intermediate outlines given outlines defined at a sparse set of keyframes.

In the field of computer graphics, there are two published systems that are arguably capable of automatically generating vector cartoons given input video; though both systems typically require user input to produce good results, in addition to being quite computationally expensive. Wang et al. described a *Video Tooning* system in which mean shift clustering was applied to a three dimensional video volume [74]. Marching cubes was then used to generate a set of boundary pixels from the clustered data. By sketching on the video frames prior to segmentation, users could introduce biases into the mean shift kernel, giving them a degree of control over the resulting volume segmentations. Once the segmentation was complete, the volumes could be adjusted by hand, to deal with occasional over- or under-segmentations. After the volume data had been defined, several video stylization options where made available to the user, in-

cluding a flat shaded cartoon style. Because the system relied on a relatively expensive three dimensional clustering algorithm, several hours of processing time were required to generate results for a 10 second, 300 frame clip of input video.

Collomosse et al. described a similar video stylization system, which improved on many of the weakness in Wang et al. [14]. Rather than applying a single clustering operation to the entire 3D video volume, an off the shelf image segmentation algorithm was used to quickly perform per-frame segmentations. A second, active contour-like optimization was then used to fit a 3D spacio-temporal Catmull-Rom surface to the sequence of segmented video frames. A graph defining region topology changes and adjacency information was generated based on these surfaces. Users could edit the graphs in order to correct for errors made by the surface fitting algorithm. After the 3D surfaces had been defined, users were able to create animations by applying several different stylization effects. In order to improve the temporal coherence of any texturing effects, a moving reference frame was maintained for each region. Temporally averaged colors were also calculated using the stored region topology data, which allowed a flat shaded cartoon style to be implemented without noticeable flickering. Collomosse et al. also presented results showing that a file containing the volume surface information and the user supplied parameters necessary to generate a stylized output video could be significantly more memory efficient than an MPEG-4 compression of the implied output.

In 2002, DeCarlo and Santella described a system for converting input photographs to simplified cartoon-like line drawings [18]. To the best of my knowledge, this is the only published system capable of generating vector format cartoon drawings from input still images without human supervision (though Inkscape's Potrace-based vectorization method is arguably an unpublished algorithm capable of achieving a similar effect). The system operates by using either eye tracking data or image saliency estimates to inform a series of increasingly fine mean shift segmentations, which are then merged to create an initial segmentation of the scene. Endpoint interpolating spline wavelets were used to simplify the initial segmentation interfaces [23]. Finally, edge emphasis lines were added based on a visual acuity model.

Image editing programs such as Adobe Photoshop have long included image filters designed to mimic certain artistic styles. For example, a cartoon filter might be implemented by multiplying the result of an edge finding filter with that of a color quantization. However, such stylizations are difficult to apply in the case of video data, as small differences between the image data in successive frames can drastically alter the output of the filters, leading to distracting flickering effects when the stylized frames are viewed in sequence.

In 2006, Winnemöller et al. presented a cartoon filter that avoided these temporal problems [76]. The filter was applied to each video frame in sequence. It did not make use of any temporal data, such as optical flow or comparisons with neighboring frames. The authors observed that the flickering effects that resulted from most stylization filters were typically the result of threshold functions in a component filter, which would send values on either side of the threshhold to radically different colors. As a value close to one of these thresholds evolved through time, it tended to move in and out of the threshold condition, which in turn lead to flickering. Therefore, any component of the cartoon filter that included such a thresholding step was replaced with a nonlinear function having smoother behavior. In order to avoid over-smoothing in regions where sharp thresholds were desired, a spatially varying sharpness field was also defined, based on filtered image gradients. Using these field values to determine the behavior of the nonlinear scaling functions tended to increase contrasts in the foreground objects while leaving the background relatively blurry and abstract.

1.8 Gradient Meshes

Gradient meshes are a vector format capable of representing smooth shaded images. A gradient mesh uses a collection of spline patches to parameterize smoothly varying colors over an image. Gradient meshes are supported by vector editing programs such as Adobe Illustrator, but have traditionally required a large amount of user guidance to create. However, in 2007, Sun et al. showed that the task of finding an optimal gradient mesh representation of an input image could be productively approached using a nonlinear least squares solver [70]. Subsequent work by Xia et al. demonstrated that arbitrary input images could be represented by relatively simple gradient meshes at a very high level of accuracy [78].

The primary motivation of gradient mesh generation algorithms has been to simplify the graphic design tasks [56]; however, Sun et al. was able to show that for simple images, their optimized gradient mesh results led to more compact files than JPEG compression [70].

2 Image Space Simplification

Art is the lie that tells the truth.

– Pablo Picasso

My vectorization system begins by applying a number of image space operations to the input photograph. The purpose of these operations is to abstract and simplify the content of the photograph, thus preparing it for subsequent vectorization.

The image space signification step benefits from the example of many previously published image stylization filters. The only relative novelties are my use of a nonuniform soft quantization function, as detailed in Section 2.4, and the p-value reparameterization discussed in Section 2.2.4. The structure tensor guided image abstraction developed by Kyprianidis and Döllner [39] includes most of the important features of my own system. The work of Kyprianidis and Döllner, meanwhile, drew heavily on the prior image stylization methods of Winnemöller et al. [76], integrating it with the structure tensor guided adaptive smoothers previously studied by computer vision researchers such as Weickert, and Kass and Witkin [75, 36].

The image space simplification is composed of three steps. First, the photograph is converted to grayscale, and a combination of blurring and unsharp masking used to exaggerate the edges (see Section 2.2). Next, an edge orientation field is generated, and used to guide a line integral convolution operation, as described in Sections 2.3.1 and 2.3.2. The operation finishes by applying the non-uniform soft quantization filter, described in Section 2.4.

In Section 2.4.2, I argue that, as a result of the image space simplification step, the behavior of the image across quantization boundaries becomes predictable. This is important, as it implies that in the case of the simplified images, Elder's *edge model* problem has a simple solution. That solution allows my system to reconstruct soft edges more accurately than prior methods, which can lead to vector results that better preserve shape and shading information, as shown in Figure 4.19.

2.1 Discrete and Continuous Image Operations

Some image space operations are most naturally described in terms of continuous mathematics, while others are most simply described in terms of discrete math. While digital logic requires that any continuous definition be approximated using discrete data, there are still many cases in which continuous definitions are more clear. Operations such as "line integral convolution" or "anisotropic diffusion" are most naturally defined in terms of continuous math.

When defining an image operation using continuous math, an image I is considered to be a function with domain Ω that returns real number luminance values, where Ω is a subset of $\mathbb{R}^2$ defined as $\Omega = [0, w] \times [0, h]$. In cases where a definition requires evaluating I for a point $\mathbf{x}$ outside its domain, the domain of I may be extended to all of $\mathbb{R}^2$ by mapping any $\mathbf{x} \ni \Omega$ to the closest $\mathbf{x}' \in \Omega$ before evaluating I.

When defining an image operation using discrete mathematics, an image I is considered to be a matrix of pixel values.[1] While I is a $w \times h$ matrix, it is often convenient to refer to image pixels using a single index i. The image space position of pixel i is the vector formed from the row/collumn indices of that pixel, and is given by $\mathbf{p}_i$.

Occasionally, it is useful to define image operations by mixing both continuous and discrete conventions, in which case, terms that assume continuous data, such as $I(\mathbf{x})$, can be assumed to be interpolations of the nearest discrete data stored in the matrix I.

2.2 Blurring and Unsharp Masking

The goal of the image simplification step is to both simplify and clarify the contents of the input photograph. Unsharp masking is a useful tool for clarifying image features, while Gaussian blurring is an effective means of simplifying visual content. In the first phase of the image simplification step, the two operations are combined by applying an unsharp mask to a blurred image.

2.2.1 Gaussian Blurring

Let $G(I, \sigma)$ denote the Gaussian blur of image I using a kernel of standard deviation σ. I will use the shorthand notation I_σ for the same result.

$$I_\sigma(\mathbf{x}) := G(I, \sigma)(\mathbf{x}) := \iint_\Omega I(\mathbf{x} - \mathbf{y}) \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{||\mathbf{x}-\mathbf{y}||^2}{2\sigma^2}} d\mathbf{y}.$$

[1] Properly, this is only a semi-discrete image representation, as the value at each pixel is still allowed to be any real number.

Recall that repeated Gaussian blurring with two kernels of width σ_1 and σ_2 is equivalent to a single blur of width $(\sigma_1 + \sigma_2)$, i.e., $G(I_{\sigma_1}, \sigma_2) = I_{\sigma_1+\sigma_2}$.

2.2.2 Unsharp Masking

Unsharp masking is a technique first used by darkroom photographers. To perform an unsharp mask, a photographer uses a negative duplication technique to create a low-detail version of an original negative. Using the low-detail negative as a mask when creating a print from the original has the effect of increasing the contrast of the result [40].

An unsharp mask can be implemented digitally by subtracting a small multiple of a blurred image from the unblurred source. For example, given a source image I, and blurred image I_σ, an unsharp mask result I_m can be defined as,

$$I_m(p) := I - pI_\sigma + pI. \tag{2.1}$$

Here p is a scalar that defines the strength of the unsharp masking effect. In equation (2.1), a small multiple of the source image, pI has been added to the result in order to compensate for the darkening implied by subtracting pI_σ.

2.2.3 Combination of Gaussian Blurring and Unsharp Masking

The result of applying a Gaussian blur followed by an unsharp mask can be expressed as follows,

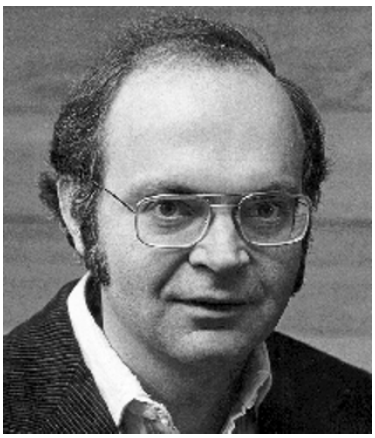
$$I_m := I_{\sigma_1} + p(I_{\sigma_1} - I_{\sigma_2}). \tag{2.2}$$

In the above, σ_1 is the width of the initial Gaussian blur, and $(\sigma_2 - \sigma_1)$ is the width of the blur used to create the unsharp mask.

2.2.4 Parameter Selection

The result of blurring followed by unsharp masking is dependent on three parameters, the blur widths σ_1, σ_2, and the unsharp mask strength, p. Of these, σ_1 is the most intuitive. It corresponds to the width of the initial blur, and thus, increasing σ_1 has the effect of removing smaller visual features and details.

The remaining two parameters, p and σ_2, however, are interdependent. Their relation can be clarified by considering the relation of equation (2.2) to difference-of-Gaussians filtering.



Input Photograph

Gaussian Blur

Unsharp Masking

Line Integral Convolution

Soft Quantization

Figure 2.1: An overview of the image simplification process. First, a Gaussian blur is applied to the input photograph. Then unsharp masking is used to reintroduce strong edges to the blur result. Next, the structure tensor is computed, and used to perform line integral convolution. Finally, the range adjustment and soft quantization operations are applied.

Origional Unsharp **Larger σ_2, constant p** **Larger σ_2, constant p'**

Figure 2.2: Changes to the blur widths σ_1 and σ_2 will significantly change the unsharp result, if the unsharp weight p is held constant. In the second two images, the unsharp result from Figure 2.1 has been changed by recomputing using $\sigma_2 = 1.6\sigma_1$ rather than $\sigma_2 = 1.1\sigma_1$. The relatively slight increase to σ_2 causes a dramatic change in the result. However, reparameterizing the operation in terms of p', as in equation (2.3), implies that the change in σ_2 has almost no impact on the result.

Difference-of-Gaussians filtering is a simple and effective means of highlighting the edges in an image. The difference-of Gaussians filter E is defined as,

$$E := I_{\sigma_1} - I_{\sigma_2}, \qquad \text{where } \sigma_2 > \sigma_1.$$

From equation (2.2), it is clear that I_m will be the sum of the edge image E and the initial blur image I_{σ_1}. The role of the parameter p is to determine the weighting of the edge image relative to the blur image. Large p will cause the edge image to dominate, while small p will cause I_m to approach I_{σ_1}.

However, the range of values in the edge image E will vary with σ_1 and σ_2. For example, on the example image shown in Figure 2.1, the variance of the edge image is $Var(E) = 1.34 \times 10^{-5}$, given $\sigma_1 = 3$ and $\sigma_2 = 1.1\sigma_1$. However, the variance of edge response values increases by more than an order of magnitude if the second blur width is changed to $\sigma_2 = 1.6\sigma_1$, in which case, $Var(E) = 3.66 \times 10^{-4}$.

The closer σ_2 is to σ_1, the larger p must be to create a noticeable edge enhancement effect. This makes experimenting with different parameter values tedious, as small changes to either σ value can dramatically change the effect of different p values.

The parameter space can be substantially simplified by compensating for any changes in the variance of E when defining p. Thus, the reparameterized blur+unsharp mask

operation is,

$$I_m := I_{\sigma_1} + p' \sqrt{\frac{Var(I_{\sigma_1})}{Var(E)}} E. \tag{2.3}$$

This reparameterization causes the original unsharp mask weight p vary as a function of the image contents. As shown in Figure 2.2, after reparameterizing I_m as in equation (2.3), the ratio $\frac{\sigma_2}{\sigma_1}$ has relatively little impact on the result. The system default is to use $\sigma_2 = 1.1\sigma_1$, mainly because that decreases the costs of computing I_{σ_2}.

2.3 Edge Aligned Line Integral Convolution

The second image simplification stage is another smoothing operation, designed to simplify object boundaries. This second smoothing stage also serves to eliminate any noise introduced by unsharp masking.

The technique of line integral convolution is used to apply a directionally biased blur. Pixel values are averaged along lines tangent to the strongest edges in the image; with the result that edges become smoother and more coherent.

2.3.1 The Structure Tensor

The structure tensor is a useful tool for defining vector fields that correspond to the direction of image edges. In 1985, Kass and Witkin proposed a variety of approaches for inferring edge alignment fields from black and white images [36]. In 1991, Rao and Schunk showed that one of Kass and Witkin's field definitions could be equivalently derived by using the concept of a *structure tensor* (which they referred to as the *moment tensor*) [59]. Structure tensors have since become commonly used tools in image analysis [75].

The structure tensor has become so pervasive that more modern authors typically cite its properties without proof or reference. This is unfortunate, as the structure tensor arises very naturally from the core problem of finding edge alignment fields over images. In order to clarify the use of structure tensors in my own system, I rederive the salient properties of the tensor here. In doing so, I follow the original line of argument given by Rao and Schunk, expanded slightly to cover the case of structure tensor blurs other than box filters.

Sobel filters, or similar gradient approximation techniques, return a vector field defined at each pixel in an image. High magnitude gradients are likely associated with edges. Specifically, at high magnitude gradient points, the edge direction is likely to be orthogonal to the gradient direction. However, low magnitude gradients will

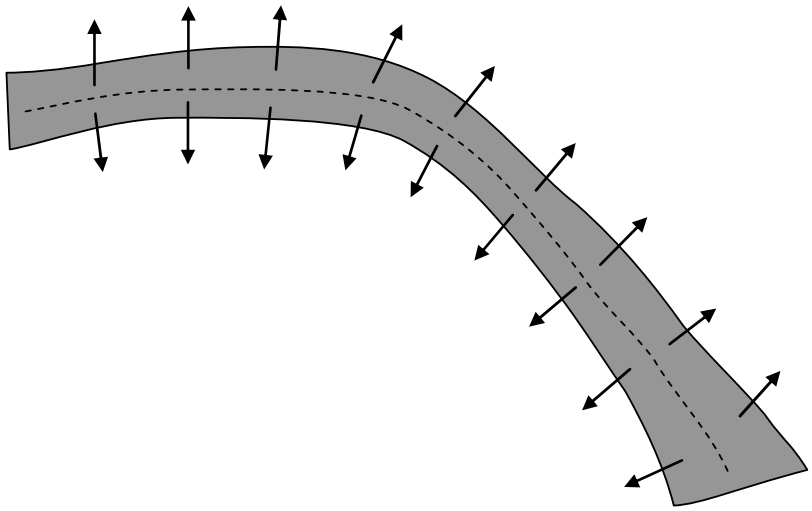


Figure 2.3: A grey ribbon on a white background. The implied image gradient vectors are shown in black. While the gradients frequently point in opposite directions, the sweep of the edges follows the center line of the ribbon, as indicated by the dotted line. Generating a vector field corresponding to the dominant edge direction requires a numerical method in which opposite direction gradients reinforce each other.

have little, if any, relation to the direction of any nearby edge lines, while even high magnitude edge data is frequently noisy.

However, it is possible to both reduce the noise in the gradient data, while simultaneously extending edge direction information stored at high magnitude points to nearby low magnitude points, by defining an edge direction field $\mathbf{u}$ as follows.

If the gradient value at pixel i is given by $\mathbf{v}_i$, and $N(j)$ is the set of pixel indices that correspond to locations near pixel j, let the edge direction vector $\mathbf{u}_j$ be a solution to the following optimization problem,

$$\min_{u} \sum_{i \in N(j)} |\mathbf{v}_i \cdot \mathbf{u}_j|^2 \quad \text{subject to } ||\mathbf{u}_j|| = 1. \tag{2.4}$$

The set of nearby pixels, $N(j)$, is most commonly defined as the set of pixels that lie inside a radius r box centered at pixel j. However, other definitions may be used without impacting the following arguments.

Despite the simplicity of equation (2.4), it accomplishes several important goals. First, it ensures that the impact of any gradient sample $\mathbf{v}_i$ will be proportional to its magnitude. The maximum possible penalty associated with a sample $\mathbf{v}_i$ is $||\mathbf{v}_i||^2$, which occurs in the case that $\mathbf{u}_j$ is parallel to $\mathbf{v}_i$. Second, because the penalty term is the square of a dot product, two gradients pointing in opposite directions will act to reinforce each other. Cases in which two nearby gradients point in opposite directions

are relatively common, as they occur any time an image contains a long thin object, see, for example, Figure 2.3. In such cases, blurring the gradient data will cause the two opposing gradients to cancel each other out. However, if the edge direction is defined as per equation (2.4), the two opposing gradients will both strongly bias the edge direction of nearby pixels towards a direction that is orthogonal to both gradients.

Finally, note that equation (2.4) does not have a unique minimizer. In fact, it is clear that if equation (2.4) is minimized by $\mathbf{u}_j$, it is also minimized by $-\mathbf{u}_j$.

However, finding all minimizers of equation (2.4) turns out to be straightforward. In the following, let the x and y components of the gradient vectors $\mathbf{v}_i$ be denoted by x_i and y_i. Thus, equation (2.4) can be rewritten as follows,

$$\min_{\mathbf{u}} \sum_{i \in N(j)} |\begin{pmatrix} x_i & y_i \end{pmatrix} \cdot \mathbf{u}_j|^2 \quad \text{subject to } ||\mathbf{u}_j|| = 1.$$

The sum can be rewritten as,

$$\sum_{i \in N(j)} \mathbf{u}_j^T \begin{pmatrix} x_i \\ y_i \end{pmatrix} \begin{pmatrix} x_i & y_i \end{pmatrix} \mathbf{u}_j \quad = \quad \mathbf{u}_j^T \left(\sum_{i \in N(j)} \begin{pmatrix} x_i^2 & x_i y_i \\ x_i y_i & y_i^2 \end{pmatrix} \right) \mathbf{u}_j.$$

It is now useful to define a matrix equal to the sum in the expression above.

$$J_{N(j)} := \sum_{i \in N(j)} \begin{pmatrix} x_i^2 & x_i y_i \\ x_i y_i & y_i^2 \end{pmatrix}.$$

The matrix $J_{N(j)}$ is the *structure tensor*. Written in terms of the structure tensor, the optimization problem simplifies to,

$$\min_{\mathbf{u}} \quad \mathbf{u}_j^T J_{N(j)} \mathbf{u}_j \qquad \text{subject to } ||\mathbf{u}_j|| = 1. \tag{2.5}$$

It is now straightforward to prove that minimizing $\mathbf{u}_j$ must be unit length eigenvectors corresponding to the smallest eigenvalue of $J_{N(j)}$. Provided that $J_{N(j)} \neq 0$, there will be exactly two such eigenvectors, which can be denoted as $\mathbf{u}_j$ and $-\mathbf{u}_j$. In the case that $J_{N(j)} = 0$, all unit length vectors will minimize equation (2.4), so the edge direction $\mathbf{u}_j$ will be undefined. Note that the case of $J_{N(j)} = 0$ will arise if and only if all the gradient vectors $\mathbf{v}_i$ equal 0, and is thus an expected outcome if and only if the image is constant valued inside the neighborhood $N(j)$.

To prove that minimizing $\mathbf{u}_j$ must be eigenvectors corresponding to the smallest eigenvalue of $J_{N(j)}$ decompose any candidate unit length $\mathbf{u}_j$ into its eigenvector components, and then expand the expression $\mathbf{u}_j^T J_{N(j)} \mathbf{u}_j$. Given $\mathbf{u} = \sum_i \lambda a_i \mathbf{e}_i$, where $\mathbf{e}_i$ are orthonormal eigenvectors of $J_{N(j)}$, $\mathbf{u}_j^T J_{N(j)} \mathbf{u}_j = \sum_i \lambda_i a_i^2$. As $\mathbf{u}_j$ must be unit length,

$\sum_i a_i^2 = 1$, thus, the minimum possible $\sum_i \lambda_i a_i^2$ is achieved when $\mathbf{u}_j = \pm\mathbf{e}_k$, and λ_k is the smallest eigenvector of $J_{N(j)}$.[2]

Notice that the structure tensor and the implied edge alignment field $\mathbf{u}$ can be calculated quickly using box filter blurs. Specifically, if I_x and I_y are images that store the x and y components of the image gradients, and $B(I, r)_j$ denotes the result of a box filter with radius r applied to image I at pixel j, then the three components of the structure tensor $J_{N(j)}$ will be,

$$J_{N(j)} = \begin{pmatrix} a & b \\ b & c \end{pmatrix}, \quad \text{where} \quad a := B(I_x^2, r)_j,\ b := B(I_x I_y, r)_j,\ c := B(I_y^2, r)_j. \tag{2.6}$$

In the above, I_x^2 denotes component-wise squaring of the elements of the image I_x, and $I_x I_y$ similarly denotes component-wise multiplication. Properly, the expressions in equation (2.6) are only correct up to a factor of scale, but, given the definition of the edge orientation vector, correctness up to a factor of scale is sufficient.

With $J_{N(j)}$ expressed as in equation (2.6), a minimum eigenvalue eigenvector will be given by,

$$\mathbf{e} \quad = \quad \begin{pmatrix} a - c - \sqrt{a^2 + 4b^2 - 2ac + c^2} \\ \\ 2b \end{pmatrix}.$$

The edge orientation vector $\mathbf{u}_j$ can then be defined by normalizing $\mathbf{e}$.

Equation (2.6) implies that, from an implementation standpoint, the collection of structure tensors defined for all pixels j is naturally represented as three blurred images, $B(I_x^2, r)$, $B(I_x I_y, r)$ and $B(I_y^2, r)$. This raises the question of whether substituting other blurring operations in place of the box filter might also lead to useful edge orientation definitions. Returning to the original optimization problem of equation (2.4), notice that it is natural to introduce weighting terms w_{ij}, which can be used to increase the influence of closer points on the calculated edge orientation vector.

$$\min_{\mathbf{u}} \sum_{i \in N(j)} w_{ij} |\mathbf{v}_i \cdot \mathbf{u}_j| \quad \text{subject to } ||\mathbf{u}_j|| = 1. \tag{2.7}$$

Provided that all w_{ij} are positive, the derivation of an optimal $\mathbf{u}_j$ can now proceed as before, with the exception that the structure tensor will now be defined as,

$$J_{N(j)} := \sum_{i \in N(j)} w_{ij}^2 \begin{pmatrix} x_i^2 & x_i y_i \\ x_i y_i & y_i^2 \end{pmatrix}.$$

[2] Similar applications of eigenvector analysis are very common in engineering and applied mathematics. For an excellent review of eigenvector analysis, see Shewchuck [66].

In the case of $w_{ij} = 1$, this simplifies to the box-filter case, but, alternate w_{ij} definitions will correspond to alternate weightings of the nearby gradient values. The form of the structure tensor that occurs most frequently in practice is J_σ, which is defined as,

$$J_\sigma := \sum_i e^{\frac{-||\mathbf{p}_i - \mathbf{p}_j||^2}{2\sigma^2}} \begin{pmatrix} x_i^2 & x_i y_i \\ x_i y_i & y_i^2 \end{pmatrix}.$$

Using J_σ is equivalent to stating that the optimization problem in equation (2.7) should be solved using Gaussian weighting terms of width $\sqrt{2}\sigma$. Specifically, the optimization problem becomes,

$$\min_{\mathbf{u}} \sum_i e^{\frac{-||\mathbf{p}_i - \mathbf{p}_j||^2}{4\sigma^2}} |\mathbf{v}_i \cdot \mathbf{u}_j| \quad \text{subject to } ||\mathbf{u}_j|| = 1.$$

The separability of Gaussian blurs makes J_σ popular in situations where minimizing computational costs is desirable. However, Rao and Schunk remarked that box filtering often seems to produce more attractive alignment fields [59], and the generality of equation (2.7) makes it clear that any smoothing operation might reasonably be used as an alternative to either Gaussian blurring or box filtering.

2.3.2 Line Integral Convolution

Line integral convolution gained popularity as a visualization technique in 1993, when Cabral showed that line integral convolutions of white noise were useful when studying the behavior of vector fields [8]. Later work by Kang et al., and Kyprianidis and Döllner, demonstrated that line integral convolution was also useful as a post-process for difference of Gaussian edge images [35, 39].

When defined for a continuous domain image, the line integral convolution L of image I with vector field $\mathbf{v}$ and weighting function $w(t)$ is defined for any point $\mathbf{x}_0 \in \Omega$ by,

$$L(\mathbf{x}_0) = \int I(C(t))w(t)dt.$$

$C : \mathbb{R} \to \Omega$ is a path with the properties that,

$$C(0) = \mathbf{x}_0, \quad \text{and} \quad \frac{dC(t)}{dt} = \frac{\mathbf{v}(C(t))}{||\mathbf{v}||}.$$

Common choices for the weighting function $w(t)$ include the box function $w(t) = [-r < x < r]$, and the Gaussian weighting, $w(t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-t^2}{2\sigma^2}}$.

Line integral convolution can be implemented efficiently in the case of discrete image data by taking a series of length 1 steps though the image. Formally, the line

integral result for the case of discrete image data can be defined as,

$$L(\mathbf{x}_0) = \sum_{i\in\mathbb{Z}} I(C(i))w(i), \quad \begin{array}{l} \text{where } C(0) = \mathbf{x}_0, \\ \quad \text{and } C(i+1) = C(i) + \mathbf{v}(C(i)), \text{ for } i \geq 0, \\ \quad \text{and } C(i-1) = C(i) - \mathbf{v}(C(i)), \text{ for } i \leq 0. \end{array} \tag{2.8}$$

In equation (2.8), assume that bilinear interpolation is used to sample image data, nearest neighbor interpolation used to sample vector data, and that the vector data $\mathbf{v}$ has been normalized.

In the second stage of image simplification, line integral convolution is applied to the unsharp mask result I_m. The system uses a Gaussian weighting function with width σ_c, and the edge alignment field $\mathbf{u}$ guides the line integration paths, where $\mathbf{u}$ is derived from the structure tensor J_{σ_d}, as described in Section 2.3.1. Thus, the line integral convolution step introduces two additional parameters, σ_c and σ_d.

However, recall that there is an ambiguity in the definition of $\mathbf{u}$. The edge orientation at any point can be given by $\pm\mathbf{u}_j$. Thus, when performing line integral convolution guided by the edge alignment field the direction of any line segment i could be either of $\pm\mathbf{u}(C(i))$.

In cases such as the thin ribbon in Figure 2.3, consistently using the positive $\mathbf{u}$ direction will frequently case the path $C(t)$ to double back on itself. Therefore, some additional logic is needed to ensure that the path is as straight as possible. Each of the two candidate line segment directions $\pm\mathbf{u}(C(i))$ is compared to the direction of the prior line segment, in order to determine which direction implies a straighter path. Formally, for each point $\mathbf{x}_0$, the sequence of direction samples $\mathbf{u}(i) := \mathbf{u}(C(i))$ is replaced by $\mathbf{u}^*(i)$, where

$$\mathbf{u}^*(i) := \begin{cases} \mathbf{u}(i) & \text{if } i = 0. \\ \mathbf{u}(i) & \text{if } i > 0 \text{ and } \mathbf{u}^*(i-1)\cdot\mathbf{u}(i) > 0. \\ \mathbf{u}(i) & \text{if } i < 0 \text{ and } \mathbf{u}^*(i+1)\cdot\mathbf{u}(i) > 0. \\ -\mathbf{u}(i) & \text{otherwise.} \end{cases}$$

2.3.3 Coherence Enhancing Anisotropic Diffusion

Coherence enhancing anisotropic diffusion is an adaptive smoothing technique introduced by Weickert [75].[3] Qualitatively, the results of edge aligned line integral convolution can be very similar to those of coherence enhancing anisotropic diffusion. This

[3] Weickert defines image "coherence" in terms of the two eigenvalues of the structure tensor, so that the coherence at any point is given by $(\lambda_1 - \lambda_2)^2$. In practice, more coherent structures are those that have sharp but smooth edge lines.

similarity is unsurprising, as both techniques are designed to smooth images along flow lines that follow the lower eigenvalue eigenvector of the structure tensor. However, the implementations of the two techniques are quite different. In practice, I have found that coherence enhancing anisotropic diffusion tends to produce higher quality results when a relatively subtle smoothing effect is desired. Line integral convolution has the advantage of being much faster to compute, and is useful for quickly producing results having a high degree of smoothing.

My system uses line integral convolution during the image simplification step, as this is a situation in which aggressive smoothing is desirable. However, coherence enhancing anisotropic diffusion is used as a post-process to the image reconstruction step; a case in which a more subtle edge smoothing effect is required. The significance of the reconstruction post-processing step is described in Section 4.5, however, I provide a description of Weickert's coherence enhancing diffusion operation here, as it is most naturally described in the context of other image space smoothing operations.

Let I be some source image, and $\phi : \Omega \times \mathbb{R}^+ \rightarrow \mathbb{R}$ be a function that returns an image as a function of time t. The Perona-Malik anisotropic diffusion equation is

$$g(||\nabla\phi||)\nabla^2\phi = \phi_t, \qquad \text{given } \phi(0) = I. \tag{2.9}$$

Under the anisotropic diffusion equation, increasing t causes the image $\phi(t)$ to become increasingly smooth.

In equation (2.9), g may be any function designed to approach 0 for large gradients.

Weickert proposed a generalization of the Perona-Malik model to allow directionally biased diffusion, which uses a diffusion tensor D as follows,

$$\nabla \cdot (D\nabla\phi) = \phi_t. \tag{2.10}$$

In order to create a directionally biased diffusion effect that will increase the coherence of the image, D is defined to be a symmetric matrix having the same eigenvectors as the structure tensor J_σ. However, the eigenvalues of D are modified in order to increase the rate of diffusion along the edge orientation direction. Specifically, if λ_1 is the larger eigenvalue of J_σ, and λ_2 the smaller eigenvalue, then the two corresponding eigenvalues of D, λ_1' and λ_2', are defined as,

$$\begin{aligned} \lambda_1' &= \alpha \\ \lambda_2' &= \begin{cases} \alpha & \text{if } \lambda_1 = \lambda_2. \\ \alpha + (1-\alpha)e^{-(\lambda_1 - \lambda_2)^{-2}} & \text{otherwise.} \end{cases} \end{aligned} \tag{2.11}$$

The parameter α controls the amount of diffusion across edge lines, and is set to a very small value, typically $\alpha = .001$.

Equation (2.10) is solved for any positive t value using finite difference methods [75].

2.4 Range Adjustment and Soft Quantization

After applying line integral convolution, the image is modified in such a way that shadows and highlights will be exaggerated. The operation begins by using a linear mapping to adjust the highlights, midtones and shadows of the image such that they fall into a standard range. Then a soft quantization operation is applied in order to emphasize the transitions between shadows, highlights, and midtones, while de-emphasize all other luminance variation.

The range adjustment step is performed by creating a C_0 continuous, piecewise linear transformation $h : \mathbb{R} \to \mathbb{R}$, consisting of two linear segments chosen such that the values $\mathbf{b}_1 = (.45, .75, .85)$ are mapped to $\mathbf{b}_2 = (.2, .61, .95)$. Applying this transform to the line integral convolution result L yields $h(L)$, which will be referred to as the *pre-quantized image*.

The final step of the stylization is non-uniform soft-quantization, using the previously defined bin values $\mathbf{b}_2$, and quantization sharpness parameter $s = 2$. The non-uniform soft-quantization is defined in the following section. The end result of the image simplification process is referred to as as the *stylized source image*.

2.4.1 Non-Uniform Soft Quantization

Given an ordered list of characteristic values, $\mathbf{b} = (b_1, ..., b_n)$, the *hard quantization* of a value v is given by,

$$q(v, \mathbf{b}) := b_i \quad \text{where } b_i \text{ is the characteristic value closest to } v.$$

To create an analogous soft quantization function, split the interval $[b_1, b_n]$ into $n-1$ regions, each of which will map to a different sigmoid curve. For $v \in [b_1, b_n]$, let b_i be the closest characteristic value to v. Now define the sigmoid curve index j as $j := i - [b_i \geq v] + [v = b_1]$. Finally, define the width of sigmoid j as $w_j := \frac{1}{2}(b_{j+1} - b_j)$, and its vertical shift as $c_j := \frac{1}{2}(b_{j+1} + b_j)$. Using these variables, create the following non-uniform, continuous soft quantization function,

$$p(v, \mathbf{b}, s) := w_j \frac{\text{sig}(\frac{s}{w_j}(v - c_j))}{\text{sig}(s)} + c_j. \tag{2.12}$$

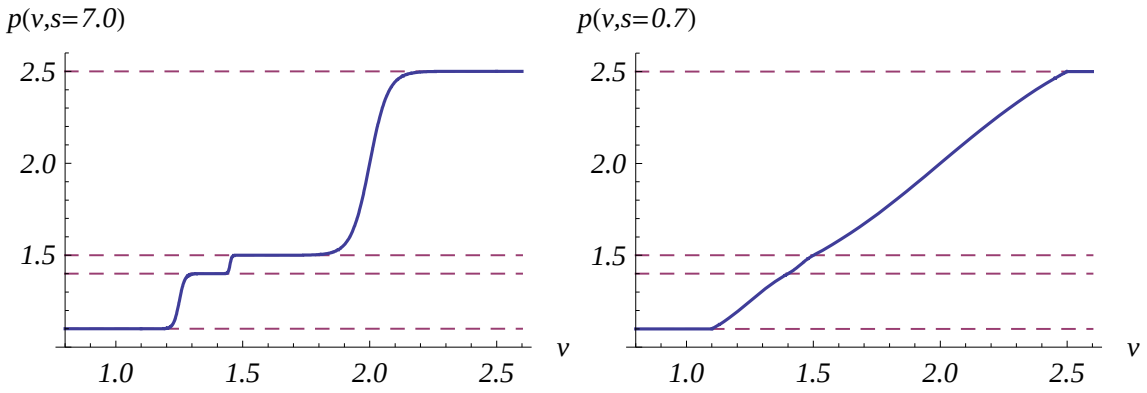


Figure 2.4: Two examples of the non-uniform soft quantization function defined in equation (2.12). Both use the same characteristic values, $\mathbf{b} = (1.1, 1.4, 1.5, 2.5)$. While the graph on the left approximates a hard quantization function, the graph on the right uses a very low sharpness value, $s = 0.7$, which causes the function to approach the identity transform. For both graphs, the domain of the soft quantization has been extended by clamping input values to the interval $[b_1, b_n]$. The sigmoid function used in both examples is the hyperbolic tangent.

Note that the the division by $\text{sig}(s)$ ensures C_0 continuity. Also note that that the derivative at the border between two quantization regions is independent of the spacing between bins. Specifically, $p'(c_j, \mathbf{b}, s) = \frac{s\,\text{sig}'(0)}{\text{sig}(s)}$. Thus the sharpness of the soft quantization is controlled exclusively by the sharpness parameter s; it is independent of the spacing of the characteristic values. For sigmoid functions with $\text{sig}'(0) = 1$, equation (2.12) converges towards $p(v, \mathbf{b}, s) = v$ as $s \to 0$. In other words, the maximally smooth soft quantization function is simply the identity transform.

Given arbitrary $v \in \mathbb{R}$, clamp v to the interval $[b_1, b_n]$, before applying equation (2.12), thus extending the domain of the soft quantization function to all $\mathbb{R}$. Examples of the non-uniform soft quantization function are shown in Figure 2.4.

I choose to perform soft quantization using $\text{sig}(x)$ defined to be the exponential sigmoid, rather than using more commonly seen sigmoidal curves such as tanh or erf. The choice sigmoid function impacts that edge model used when vectorizing and reconstructing the stylized image, and using the exponential sigmoid in the soft quantization step implies that the differential equations in Section 4.2 have simple solutions.

2.4.2 A Solution to the Edge Modeling Problem

When Elder initially proposed his edge-only image format, he identified *edge modeling* as a central challenge facing any edge-only format, and one that might limit the utility

of edge only images in practice. To solve the edge modeling problem, we must be able to characterize how image data should behave on either side of an edge. For soft edges, simply sampling the gradient at the edge crossing is not sufficient, it is necessary to have some model that specifies how luminance will vary as the distance from the edge increases. Elder's solution to the edge modeling problem was to assume that all luminance variations across edges could be modeled by the error function $a\text{erf}(bx)$. The edge model parameter k was then found by fitting this function to the source image data near the edge [20].

Elder's edge fitting method relied on a very narrow pixel sampling, but, even so, it resulted in good results in the context of his own edge-based system. However, performing a high accuracy fit of an edge model function like $a\text{erf}(bx)$ to the very smooth edges that appear in our own stylizations would be both difficult and computationally expensive.

Yet, in the context of my system, it is not necessary to perform such a model function fit, as there is additional information that can be used to derive the edge model functions.

Quantizing the image implicitly partitions it into a set of regions having shared quantization values. Some of the boundaries between regions will occur in areas having low difference of Gaussian response, and thus, they will tend to have smoothly varying, locally linear luminance values. It is such smooth region boundaries that are responsible for most of the apparent "smooth shading" in our stylized images. I now show that, by assuming local linearity near region boundaries, it is straightforward to derive a single sharpness parameter, k, which models the behavior of the stylized image across such a region edge.

Let $I : \Omega \rightarrow [0,1]$ be a grey scale image, having domain $\Omega \subset \mathbb{R}^2$. Let $S : \Omega \rightarrow \mathbb{R}^+$ be an image that defines the desired sharpness of the soft quantization at any point in the image. The soft quantization of I using sharpness image S is thus $p(I,S)$.

Consider the case in which the boundary between quantization regions occurs in an area where I is approximately linear. More specifically, let $\mathbf{x}_0$ be a point on the boundary between two adjacent regions, and Ω_0 be a circular subset of Ω for which

$$I(\mathbf{x}) \approx I(\mathbf{x}_0) + \nabla I(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0), \text{ and } \nabla I(\mathbf{x}) \approx \nabla I(\mathbf{x}_0), \ \forall \mathbf{x} \in \Omega_0.$$

Now consider a line $\mathbf{x}(\delta)$ normal to the boundary, where δ denotes the distance from the boundary point $\mathbf{x}_0$. By local linearity, the image value at a point $\mathbf{x}(\delta) \in \Omega_0$ can be approximated using,

$$I(\delta, \mathbf{x}_0) \approx I(\mathbf{x}_0) + \delta ||\nabla I||. \tag{2.13}$$

Substituting equation (2.13) into equation (2.12) yields the result that, for points inside

the region Ω_0, the soft quantization $p(I,S)$ can be approximated by

$$p(\delta,\mathbf{x},s) \approx w_j \frac{\text{sig}(\frac{s}{w_j}(\delta||\nabla I(\mathbf{x})||))}{\text{sig}(s)} + c_j. \tag{2.14}$$

Where the sigmoid index j is chosen to correspond to the sigmoid centered at the boundary value, such that $c_j = I(x_0)$.

At a high level, this is not a surprising result. When soft quantization is applied to sufficiently smooth image data the behavior near region boundaries can be described in terms of the sigmoid curve used to define the soft quantization, and the gradient magnitude of the source image.

In general, if an image has been aggressively smoothed, then luminance variations throughout the image are likely to be both small and locally linear. However, if an image has been sharpened, then luminance variations near edges are likely to be large and discontinuous.

In areas with high edge response, I thus model edges as sharp step functions, which are represented as edges with infinitely large sharpness parameter, k. For low edge response cases, however, I use the edge model function $\frac{w_j}{\text{sig}\, s}\,\text{sig}(kx)$. The k value used in my system is close to that implied by equation (2.14), though the small scaling factor $\text{sig}(s)^{-1}$ is ignored, for reasons discussed in Section 4.2.5. Removing the scaling factor yields $k = \frac{s}{h_j}||\Delta I||$.

2.5 Tracing on the Right Side of the Brain

The boundaries between quantization regions will form the starting lines input to the tracing algorithms. As the image shadows and highlights are exaggerated by the unsharp masking step, and then quantized using bin values chosen to divide the image into extremes of light and dark regions, there is a sense in which the system's approach to combined stylization and vectorization follows the advice given by Edwards in her famously effective instructional book, *Drawing on the Right Side of the Brain*. Rather than attempting to trace the the outlines of individual objects, it simply focuses on tracing patterns of light and dark tones [19].

3 Vectorization

/ A number that speaks for itself, every kissable digit. */*
#define PI *3.14159265358979323846264338327950288419716*
/ Another fave. */*
#define SQUAREROOTTWO *1.41421356237309504880168872*
/ And here's one for those of you who are intimidated by math. */*
#define ONETHIRD *0.33333333333333333333333333333333*

– Jonathan Richard Shewchuk

The vectorization algorithm converts the image space stylization result of the previous chapter to a set of curves. It begins by using the results of the image space stylization to define an initial segmentation of the image. Then small regions are eliminated. After extracting curves from the region, the curve set is simplified, and a set of edge sharpness parameters are sampled for the simplified curve set.

Finally, the vectorization algorithm records the characteristic luminance value inside each closed region, with the result that the curve data can be used to construct a piecewise constant approximation to the source image.

The complete vector format thus consists of the region boundary curves, an additional parametric curve that stores the sharpness parameter k at each point along the boundary, and a set of luminance samples, defined for each connected component implied by the boundary curve set. The vector tracing algorithms are described in Section 3.1.

After the vector data has been sampled, much of the edge sharpness data is discarded, in order to allow for more efficient encoding. This final sharpness simplification, and the encoding procedure, is described in Section 3.2.

Completing the vectorization process quickly requires a fast operation that can be used to identify the connected components implied by the curve set. The graph cutting and connected components labeling algorithms described in Sections 3.3.1 and 3.4.1 allow connected components to be calculated quickly. Both algorithms are reused as components of the image reconstruction step, described in Chapter 4.

3.1 Tracing

3.1.1 Small Region Elimination

The first step is to create a segmentation based on the the stylized image. This is done by simply setting each pixel to the characteristic value of the closest quantization bin, thus creating the hard-quantization associated with the soft-quantization generated in Chapter 2. (The luminance value that characterizes a quantization bin is referred to as the *characteristic value* of that bin.)

The resulting quantization is then simplified, by eliminating any small regions that may have resulted from the quantization. The result of this simplification will be referred to as the *simplified quantization*.

Specifically, for each region R formed by 4-connected pixels that share the same quantization bin, I define an energy over all pixel locations $\mathbf{p} \in R$. The energy contributed by each pixel is a function of b, the characteristic bin value of the region, w, b between the maximum and minimum luminance values falling into the region's bin, and $sq(\mathbf{p})$, the value in the soft-quantized source image. The region energy is defined as,

$$\sum_{\mathbf{p}\in R} \min(2 \cdot \left(\frac{w}{2} - |b - sq(\mathbf{p})|\right), \epsilon).$$

A pixel with a soft-quantized luminance value very close to the edge of the bin will thus contribute ϵ to the region's energy, while a soft-quantized value that exactly matches the region's characteristic value will contribute w. I eliminate any region with a total energy below ω. Both ω and ϵ are parameters of the region elimination step. The default values used by the vectorization system are $\epsilon = .2, \omega = 3$.

After deleting a region, all pixels inside the deleted region must be assigned to one of the neighboring regions. It is important that no new closed regions are created as a result of this operation. The necessary infilling is achieved using a simple greedy search. The search starts by considering all border points in the region to be deleted, i.e., all points $\mathbf{p} \in R$ that have neighboring points $\mathbf{p}'$ inside other regions R'. The point with $sq(\mathbf{p})$ closest to the characteristic value of a neighboring $\mathbf{p}' \in R'$ is assigned to the region R'. The set of border points is then updated, and the operation is repeated until the region has been entirely filled in.

3.1.2 Generating Curve Segments

For each of the quantization bin values, b_i, a binary image can be created, by simply setting each pixel in the image to 1 if b_i is the value in the simplified quantization, or 0

otherwise. An initial set of boundary curves is then created by applying the marching squares algorithms to each of these binary images.

Marching squares is, itself, a simplification of the commonly used marching cubes boundary tracing method [45, 77]. The algorithm operates by traversing the set of all pixel corners. When a pixel corner is encountered that lies on the edge of the region boundary, it triggers a border tracing event, in which the algorithm "marches" along the border of the region, adding pixel corners to the boundary set until it returns to the starting point. After marching past any point, it is marked as having been processed; points that have already been processed do not trigger border tracing.

Thus, when specifying the marching squares algorithm, one must specify where the next pixel center in the march will lie, given the binary values stored in the four adjacent pixels. To derive this specification, imagine that the binary image defines the boundaries of a maze, which is being explored by a hero. In order to avoid getting lost, the hero places his left hand against the side of the maze, and then walks forward. In most cases, this "left hand rule" makes it clear what the next point in any marching sequence must be. For example, in the case that the four adjacent pixels have the following values, the left hand rule implies moving up.

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

However, there are two special cases, in which the result of the left hand rule is unclear. For example, given a pixel corner for which the four adjacent pixels have either of the following values, it is unclear whether the left hand rule implies moving up or down.

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

When encountering one of the two 2x2 checkerboard patterns, the location of the prior point in the marching sequence is used to determine which of the two possible choices to make, in order to ensure that the marcher passes between the two diagonally adjacent pixels. In the special case that the first point added to the boundary is an ambiguous case, the direction is always set to either left or down, this ensures that in larger checkerboard cases, such as the 3x3 case, each pixel's boundary will be traced exactly once.

The interfaces found by marching squares take the form of closed loops around each region of the simplified quantization. These closed loops are used to represent region interfaces during the previously described small region detection and infilling steps.

However, such closed loops are not a desirable representation of the boundary information during later stages of the pipeline. Boundary points at the border between two regions will be represented twice, as they belong to the closed loops found for two of the implied binary images.

There are two problems with this data redundancy. First, the boundary curve data will form the bulk of the information stored in the vector format, and thus a more efficient representation of the boundaries is desirable. Second, simplifying or smoothing the boundary curve data is complicated when each boundary point belongs to two independent curves.

Thus, before proceeding any further, the vectorization algorithm converts the closed loops returned by marching squares into a series of curve segments, such that every boundary point appears only once.

This is done as follows. The marching squares results take the form of lists of integer pairs, which represent the indices of pixel corners on the boundaries of the regions. Any point on the boundary curve is defined to be a *boundary corner* if the adjacent pixels belong to three or four different regions.

The marching squares results are then converted into a set of curve segments, such that each segment connects two boundary corner points. This is done by iterating over the entire set of recorded boundary points, and adding each corner-to-corner sublist to the set of boundary segments, unless that segment is already included in the set. In the special case that a closed loop contains no corner points, it is converted to a single closed boundary curve.

Boundary points that match either of the 2x2 checkerboard cases are also considered to be boundary corners, for the purposes of this algorithm. Depending on the surrounding quantization data, the checkerboard cases may be adjacent to as many as four different connected components, or as few as two different components. Inserting a corner point in the case that only two components are bordered will not cause any problems later in the algorithm, other than a small potential loss of memory efficiency. Failing to insert a corner point in cases in which more than two different connected components are bordered, however, can lead to serious problems. For an example of such a problem case, consider the 3x3 image having the following values,

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

If the pixel boundaries adjacent to the center pixel are not all considered to be boundary corner points, the line segments found by the curve segment definition algorithm may not contain any boundaries around the center.

3.1.3 Boundary Curve Simplification

At this point in the process, the boundary curve data is represented as lists of pixel corner points. Those lists can be interpreted as parameterizing C^0 continuous curves, defined as a list of vertical and horizontal line segments, where each segment is exactly one pixel width long.

The curve simplification step converts these small line segments into a more efficient format.

Given an initial list of line segment endpoints $V = (\mathbf{v}_0, ..., \mathbf{v}_m)$, the simplification step extracts a sublist $Z = (\mathbf{z}_0, ..., \mathbf{z}_n)$, which contains the list of line endpoints for the simplified curve. To determine which of the $\mathbf{v}_i$ should be included as elements of Z, I use a *straightness constraint*, similar to that used in Selinger's polygon extraction phase [65]. Specifically, a sequence of points $(\mathbf{v}_i, ..., \mathbf{v}_j)$ is called δ*-straight* if and only if the line segment with endpoints at $\mathbf{v}_i$ and $\mathbf{v}_j$ intersects a square of radius δ centered at every $\mathbf{v}_k$, for $k \in [i, j)$.

Starting with v_i=v_0, the simplification algorithm walks through the remaining points in sequence. When it encounters a point $\mathbf{v}_j$ that fails the δ-straightness constraint, $\mathbf{v}_{j-1}$ is added to the simplified line segment list Z. The traversal then continues, with $\mathbf{v}_j$ taking the place of $\mathbf{v}_i$ in the next set of straightness tests. The final point in the curve, $\mathbf{v}_m$, is always added to Z, this is ensures that the endpoints of boundary line segments never change as a result of the simplification step.

The choice of the parameter δ in the above algorithm makes it possible to control how much the simplified curves can deviate from the initial boundary. However, there are some types of boundary curves that create error cases for the above simplification algorithm. For example, with $\delta > .5$, a long, single-pixel wide closed loop will reduce to a segment only one pixel-width long. In such cases, the algorithm can travel arbitrarily far from $\mathbf{v}_i$, and then travel all the way back to $\mathbf{v}_i$ without ever breaking the straightness constraint. To disallow such behavior, I add an additional constraint that must be satisfied before adding each new vertex $\mathbf{v}_j$. This constraint states that the magnitude of $||\mathbf{v}_j - \mathbf{v}_i||^2$ must be at least $||\mathbf{v}_k - \mathbf{v}_j||^2 - \frac{1}{2}\delta^2$, for all $k \in (i, j)$.

3.1.4 Edge Sharpness Sampling

While performing the boundary simplification step, I sample the gradient magnitude of the underlying stylized image data. In Section 2.4.2, I argued that, for the purposes of modeling the edge behavior in the soft-quantized result, the pre-quantized image $h(I_m)$ is actually a better source of information than the final result image. Thus, to calculate edge sharpness data corresponding to the soft-quantized result, the gradient

magnitude is calculated at every point of the pre-quantized image.

Sobel convolution filters are used to approximate the x and y components of the gradient are, they are defined as follows:

$$S_x := \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \qquad S_y := \frac{1}{8}\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

A max filter is then applied to the gradient magnitude image. This filter sets every gradient magnitude sample to the maximum value within radius r_g of the pixel's center. The application of a small radius max filter has the effect removing high frequency components of the gradient data, while additionally biasing the results of the sampling towards sharper edges.

Gradient samples are then recorded as follows. For each line segment found by the boundary curve simplification step, one gradient sample is recorded for each of the pixel wide vertical or horizontal line boundary line segments implied by the pre-simplified data. The gradient sample for each such line segment is the average of the two gradient values stored on either side of the line segment. The gradient samples are then converted to a list of scalar tuples, which allows a continuous gradient value to be defined at every point along the boundary curve.

Specifically, a list of t-value samples is generated for the gradient samples associated with each simplified line segment. If the list of gradient samples for the simplified line $(\mathbf{x}_i, \mathbf{x}_{i+1})$ is $(g_1, g_2, .., g_m)$, then the t value created for sample g_j is,

$$t_j := i + \frac{j-1}{m}.$$

These t values allow the gradient at any point on the curve to be evaluated as a linear spline, defined for knot points given by (t_j, g_j). The gradient values g_j are then converted to sharpness values k_j, according to the formula derived in Section 2.4.2.

The gradient sampling method generates data that is typically much denser than necessary, but, as explained in Section 3.2.1, this data is dramatically simplified prior to encoding.

3.1.5 Luminance Sampling

The final step of the vectorization process is to record a characteristic luminance value for each of the connected components implied by the boundary curves.

This is potentially a difficult problem, for several reasons. First, it is possible that the boundary line simplification step may have introduced self intersections into the

curve set. Occasionally, a narrow region in the simplified data will have boundary curves that reduce to identical lines, effectively causing it to disappear.

Image reconstruction will require a piecewise constant image created from the luminance samples and boundary curves. Therefore, it is necessary that the results of the luminance sampling, taken together with the curve data, be capable of reliably generating such a piecewise constant image.

The method I have chosen to use for luminance sampling is computationally efficient but it does not make any direct attempt to remove self intersections or degenerate regions from the boundary curves.

First, a 4-connected pixel adjacency graph is generated for an image of resolution $w' \times h'$, which is defined to be three times the resolution of the source image. The boundary curve data is then scaled to cover that image, and the line segment driven graph cutting algorithm, described in Section 3.3.1, is used to remove the edges of the pixel adjacency graph that intersect the boundary curves.

Labeling the connected components of the cut adjacently graph then becomes a relatively straightforward operation, as degeneracies or self intersections in the boundary curve set no longer complicate the task of identifying connected regions.

The component labeling operation is performed using a variation on the standard image-based connected component labeling algorithm, described in detail in Section 3.4.1. The result of this labeling is an image in which each pixel contains an integer ID indicating the component to which it belongs, and the range of those IDs is $[0, n-1]$, where n is the total number of connected components in the cut graph.

The simplified quantization is upscaled to the resolution of the connected component data, using nearest neighbor sampling. For each connected component, the luminance sample recorded for that component is the most frequently occurring simplified quantization value inside the connected component.

In the case that the δ parameter used during boundary simplification is 0, all connected components will contain the same simplified quantization value. However, larger δ values will cause the boundaries of the connected components to shift relative to the simplified quantization. There will often be some pixels, near the boundaries of each component, in which the simplified quantization value does not match the most frequently occurring value for that component.

Given the luminance samples, and the boundary curve data, a piecewise constant image can be calculated at arbitrary resolution as follows. First, the cut adjacency graph and implied component label image are recalculated from the boundary curve data, at the resolution $w' \times h'$. Then, a piecewise constant image I_{char} is generated at that resolution, by setting each pixel to the luminance value stored for its connected

component. Finally, nearest neighbor sampling is used to resize the $w' \times h'$ resolution piecewise constant image I_{char} to the target resolution. The value for each connected component label is set to the most frequently occurring resized label image value inside that component.

This method may seem overly complex, but, in practice, it has two important advantages. First, it is fast—performing the graph cuts, labeling the connected components in the image, and setting each pixel to the most frequent value in its component are all cheap operations. Formally, there is no guarantee that connected component labeling will complete in linear time, but, in practice, the labeling operation is significantly faster than most image filtering operations.

Secondly, by generating a cut graph at the target resolution, resolution dependant artifacts such as "jaggies" are eliminated. Thus the edges between the piecewise constant regions will always be as smooth as possible, given the resolution of the grid and the recorded boundary curve data.

3.2 Encoding

In order to generate memory efficient vector data, I further simplify the traced data, as described below.

3.2.1 Sharpness Data

First, the edge sharpness data sharpness samples (t, k) are resampled using sampling frequency f_k, where f_k is defined relative to the image space distance between two consecutive sharpness samples. This resampling requires converting the t_j values, which parameterize sharpness relative to the boundary line sample positions, to x_j values, which parameterize sharpness relative to their image space distance along the boundary line.

I then define k_{max}, a maximum bound on the edge sharpness parameter k, and threshold all samples using this upper bound. Next, I lower the resolution of the k values, by limiting each k sample to one of n_k values, which are found by applying k-means clustering to the set of sharpness samples for which $k < k_{max}$. The system defaults to $k_{max} = .6$, $f_k = 1.7$, $n_k = 5$. After these simplifications, edge samples of the form (x_i, k) can be eliminated, if both adjacent edge samples store the same k value, because such samples correspond to redundant knots in the linear spline that defines the sharpness values at any point.

After those simplifications have been applied, the k sample data is stored using

Huffman coding [33]. This requires storing frequency information for both each k_j value, and each sample position, x_j. To improve on the efficiency of this coding, the position data x_i in reexpressed in terms of a sequence of offsets, $d_i = x_{i+1} - x_i$.

3.2.2 Luminance data

Luminance information makes up a very small part of the vector format. The set of possible luminance values is stored as an uncompressed double array, and an integer index mapping into that array is stored for each of the connected components implied by the edge data. As the system defaults to three value luminance quantization, this implies that each luminance entry requires 2 bits of storage space.

3.2.3 Edge Data

After the edge model data simplifications have been applied, by far the largest portion of the vector data is the edge lines themselves. These are stored using Huffman encoding, after reexpressing each edge line in terms of its start point, and a sequence of delta values that define each successive point in the edge.

3.2.4 Format Details

The details of the encoding format are summarized in Figure 3.1. The format is most simply described in terms of the following primitives:

Integer Arrays A list of unsigned integer values, $L = (i_0, ..., i_n)$. Integer arrays are encoded by first storing the number of elements, followed by the number of bits b needed to encode any element of array, calculated as $b = \lceil \log_2(\max(L)) \rceil$. Both n and b are stored as 32-bit integers. Finally, the elements $(i_0, ..., i_n)$ themselves are stored using b bits each.

Compressed Integer Array A slightly more memory efficient variant on an integer array. For a list of unsigned integer values, $(i_0, ..., i_n)$, the number of elements n is recorded, followed by the the maximum integer $m = \max(L)$. Then a huffman encoding for the integers between 0 and m is generated by assuming that all integers in that range are equally probable. If m is a power of two, a compressed integer array encoding will be nearly identical to the uncompressed encoding. In other cases, it may be slightly more memory efficient.

Double Arrays A list of 64 bit floating point values, $(d_0, ..., d_n)$. Double arrays are encoded by first storing the number of elements, followed by the 64-bit representation of each element of the array.

Huffman Coded Data Store a list of unsigned integer values $L = (i_0, ..., i_n)$, using a matching list of probabilities $p_0, ..., p_m$. The probability data is generated by calculating the frequency of each integer value, and then setting p_i equal to the number of times i occurs in L. The probabilities are used to generate Huffman coding strings for each element of L, which are then used to store each element of L. In order to store data in this format, the probability data (i.e., the data histogram) must be stored in a separate integer array.

Typically, the Huffman encoded integer list will contain indices into a set of either floating point or integer values. Thus a list of source double values $(d_0, ..., d_n)$ might be coded by first finding the set of all unique d_j values. Represented as a double array, that set defines a mapping between the double values d and integers i. A histogram for the implied integer list $(i_0, ..., i_n)$ would then be calculated. As written to disk, the source double array $(d_0, ..., d_n)$ would represented as the combination of its mapping data, histogram data, and the Huffman coded indices $(i_0, ..., i_n)$.

The edge and sharpness data are both stored using flattened lists. Position data for all edges is concatenated into a single list, while the number of points in each edge is stored as a separate integer array.

It is often the case that all points on an edge line will be "infinitely sharp". No sharpness data is stored for such edge lines. To facilitate this, edges are arranged into a list having the property that the first n_{inf} edges fall into the "infinitely sharp" case. Thus, the number of sharpness splines stored is equal to the total number of edge lines, minus n_{inf}.

3.3 Edge Cutting

Quickly and accurately creating pixel adjacency graphs based on a sequence of line boundary segments is a core operation of the system. Such operations are required in both the luminance sampling and image reconstruction steps, described in Sections 3.1.5 and 4.3.4. In principle, creating these graphs is a fairly straightforward problem. Removing the edges of a regular grid graph is not unlike drawing a line, and minor modifications to Bresenham's line drawing algorithm would allow it to act on pixel adjacency graphs, rather than bitmaps. However, in practice, the edge cutting cutting task proves to have some special complications.

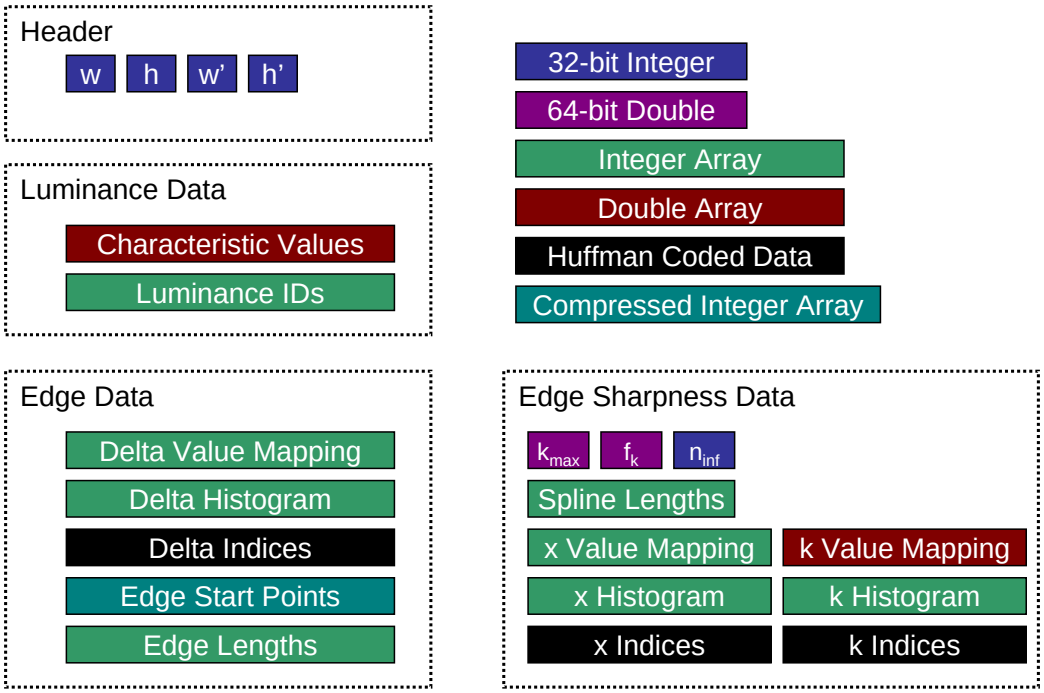


Figure 3.1: Encoding Format. The header includes the resolution of the source image, as well as the resolution of the label image used when calculating connected components (the significance of the label image resolution is discussed in Section 3.1.5). The edge sharpness data includes the sampling frequency of the position data, f_k, along with the maximum sharpness bound k_{max} (see Section 3.2.1). The integer n_{inf} records the number of edge lines for which the sharpness is uniformly infinite, n_{inf} (as discussed in Section 3.2.4). The histogram calculated for edge delta values is slightly compressed; in that the edge delta histogram is forced to be symmetric about the origin. For example, if there are three delta values equal to -4 and one value equal to 4, the recorded histogram will consider the frequency of both 4 and -4 to be two.

3.3.1 Edge Cutting Algorithm

To define the edge cutting algorithm formally, consider an image of resolution $w \times h$. Arrange a regular grid of points having the given dimensions. These points will be referred to as *pixel centers*. Let $\mathbf{p}_i$ refer to the position of the i-th pixel center, which can be represented as a pair of integers in the range $([1,w],[1,h])$.

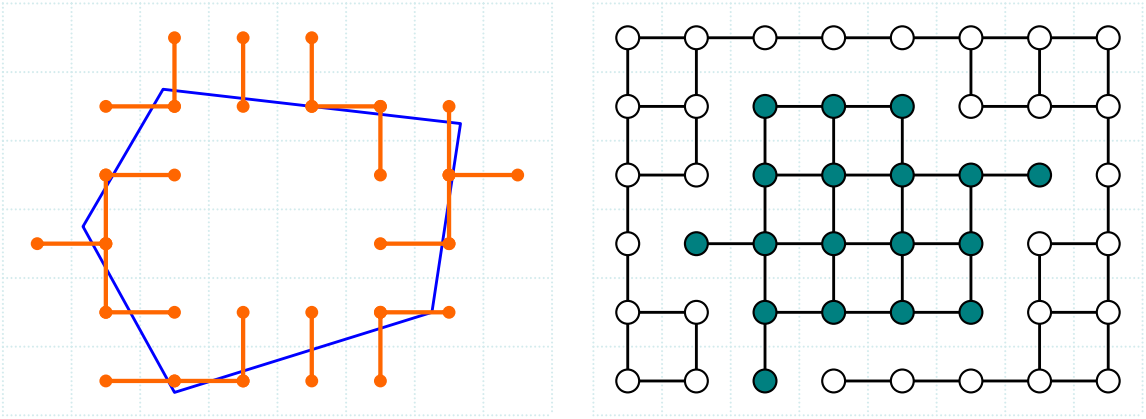
The pixel centers will form the vertex set of the pixel adjacency graph. The adjacency graph is defined by stating that there is an edge between any two pixels that are immediate horizontal or vertical neighbors of each other. Thus, each pixel is neighbors with at most four other pixels. Notice that pixels on the boundaries of the image are neighbors with at most three other pixels, while the corner pixels have only two neighbors. Despite this, graphs formed in this fashion are referred to as *4-connected* pixel adjacency graphs.

The purpose of the edge cutting operation is to create a graph in which the connected components reflect the regions of $\mathbb{R}^2$ implied by the line boundary data. However, creating an adjacency graph with the property that each pixel shares a connected component with exactly those pixels having centers that lie in the same region is an under-defined problem, as it is not clear how pixel centers that lie on the border between two regions should be handled. For example, the boundary lines shown in Figure 3.2 include one line that passes through a pixel center, thus, there are two possible adjacency graphs that provide equally good discrete approximations to the implied regions. For the purposes of my algorithms, either of these adjacency two graphs would serve equally well.

Given an edge in the pixel adjacency graph that connects the pixels i and j, the closed line segment with endpoints at $\mathbf{p}_i$ and $\mathbf{p}_j$ is referred to as the *edge line*. Given a set of boundary line segments, removing all edge lines from the adjacency graph that intersect any boundary line is a fairly straightforward problem. And, if no pixel centers lie on boundary lines, such a cut graph will partition the set of pixels into connected components that reflect the regions of $\mathbb{R}^2$ implied by the boundary curves.

The cases in which pixel centers do lie on boundary lines are resolved by first modifying the input line segments to ensure that no segment endpoint lies exactly on a pixel center. Thus, any integer valued line endpoint coordinate is incremented by 2^{-9}. However, note that after this step it is still possible for pixel centers to lie inside boundary line segments.

Strictly vertical or strictly horizontal boundary lines are then handled as special cases. The increment by 2^{-9} ensures that the boundary lines in the axis aligned cases will not contain any pixel centers, thus, computing the set of cut edges in such cases is straightforward. A strictly vertical line from (x_0, y_0) to (x_0, y_1) will cut all horizon-



Cut Edges **Connected Components**

Figure 3.2: Cut Edges and Connected Components. The boundary lines (blue) divide the image domain into two regions, which, formally, are disjoint subsets of $\mathbb{R}^2$. The cut edges, shown in orange, are a selection of the edge lines intersecting the boundary lines. The connected components of the graph formed when the cut edges are removed are discrete approximations of the regions formed by the boundary lines.

tal edge lines having endpoints $(\lfloor x_0 \rfloor, y), (\lceil x_0 \rceil, y)$, with integers valued y such that $y_0 < y < y_1$. Strictly horizontal boundary lines similarly cut sets of vertical edge lines.

For the case of diagonal edges, the edge cutting algorithm proceeds by choosing one pixel center as a starting position, then entering into a loop in which the current position may be changed to the pixel center of one of the neighboring pixels, cutting an edge in the process. As each position change is associated with exactly one edge removal, it is possible to ensure that a pixel center that lies on a boundary line will have exactly two of its edge lines removed as a result.

Given a boundary line with endpoints $\mathbf{p}_0 = (x_0, y_0)$ and $\mathbf{p}_1 = (x_1, y_1)$, consider the special case in which $x_0 \leq x_1$, and $y_0 \leq y_1$. The current position is initialized to be $(i, j) = (\lfloor x_0 \rfloor, \lfloor y_0 \rfloor)$. Two candidate edge lines are tested for intersection with the boundary. The first edge line tested is the horizontal edge line immediately above the current position, having end points $(i, j+1), (i+1, j+1)$. If that edge intersects the boundary, the current position is immediately incremented by $(0, 1)$, and the algorithm repeats. Otherwise, the vertical line immediately to the right of the current position is tested, which has endpoints $(i+1, j), (i+1, j+1)$. If that line intersects the boundary, the current position is incremented by $(1, 0)$, and the algorithm repeats. If neither line is intersected, the algorithm terminates.

In the case that $x_0 > x_1$, the algorithm proceeds as above, but, decrementing i

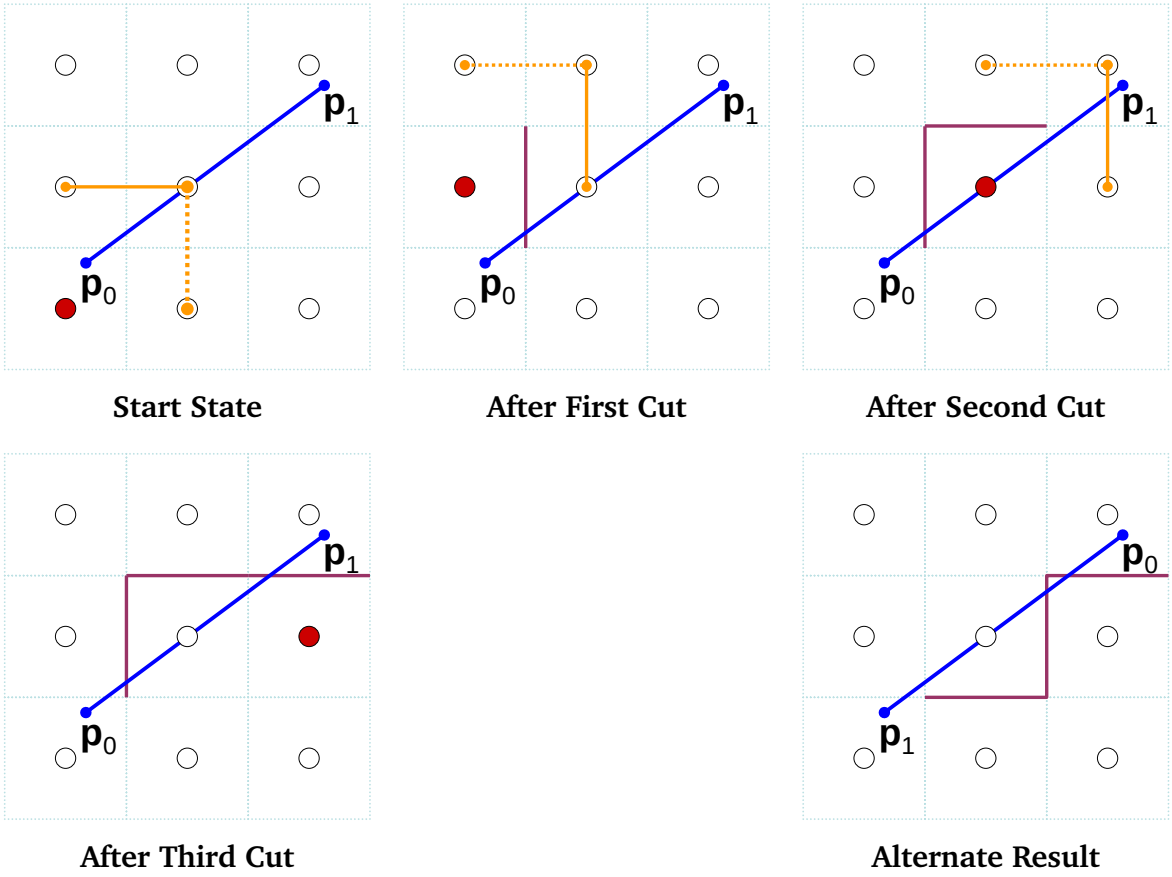
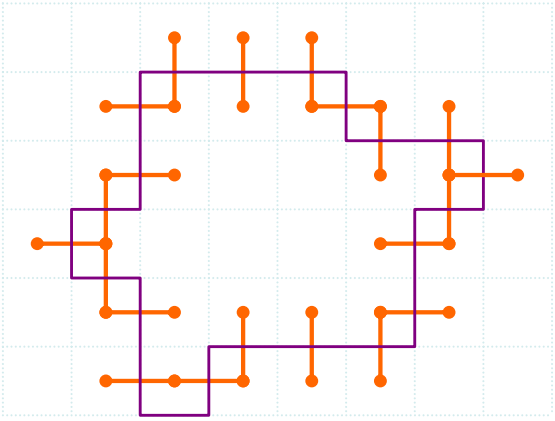
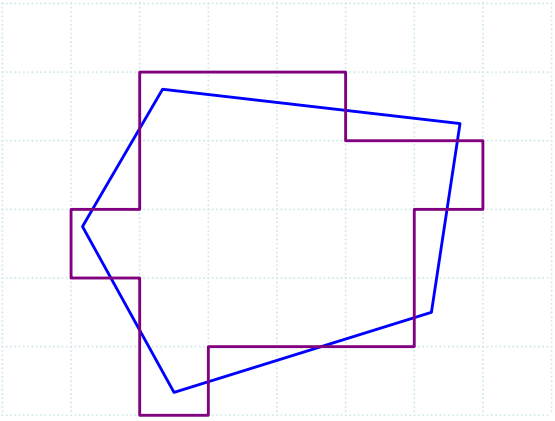


Figure 3.3: Edge Cut Example. Pixel centers are shown as circles, the current position is red. In each step, the two candidate edge lines (orange) are checked for intersection with the boundary line (blue). If one of the candidate edge lines intersects the boundary, then the edge is removed, and the current position updated. The candidate edge line that is removed in each step is shown as a solid line, while the edge left uncut is shown as a dashed line. The current position is changed by $(1, 0)$ in the case of a vertical edge line, and $(0, 1)$ in the case of a horizontal edge line. As edges are removed, the emerging boundary between connected components is shown in purple. Intersection with horizontal edge lines is checked first; thus, in the case that the boundary passes through a pixel center, the current position will move vertically, before moving horizontally. Therefore, the connected components implied by edge cutting can differ, depending on the order in which the endpoints of the boundary line are specified, as shown in the alternate result.



Cut Edges and Component Boundary



Boundary Lines and Component Boundary

Figure 3.4: Boundary Comparison. Each edge removed from the adjacency graph implies a length 1 vertical or horizontal line segment. The removed edges are shown in orange, while the boundaries of the implied connected component are shown in purple. Notice that the connected component boundaries form a fairly good approximation for the boundary line segments used to perform the edge cutting operation.

in place of incrementing it, and starting the initial position at $i = \lceil x_0 \rceil$. The case of $y_0 > y_1$ is handled similarly. An example of the edge cutting algorithm given a short boundary line segment is shown in Figure 3.3.

During the early phases of image reconstruction, described in Section 4.3.4, it is necessary to find not only the set of cut edges, but, also a sharpness sample k for each cut edge. Such a sample is found by calculating the intersection point of the boundary line and the edge line, and evaluating the sharpness spline at that point, using the linear spline data gathered in Section 3.1.4.

A formal proof that the connected components of the cut graph will be, in some sense, a "good discretization" of the regions formed by the boundary lines is beyond the scope of this dissertation. Furthermore, it is unclear how exactly such a proof should be approached, as a formalization of the "good discretization" concept is likely to be complex. For example, consider the case of a region that contains no pixel centers. Such a region need not have any matching connected components in the adjacency graph, but, such a region might, or might not, have an associated connected component in the case that at least one pixel center lies on the region's border.

Thus, rather than attempting such a formal proof, I will simply state that, in practice, this edge cutting algorithm has proven to reliably create connected components that are close matches for the simplified quantization data from which the boundary

curves are derived.

3.3.2 Ensuring Exact Intersection Tests

The edge cutting algorithm described in Section 3.3.1 requires a reliable means of determining whether any boundary line segment intersects any edge line. To avoid “leaks", cases in which the connected components implied by the graph cover more than one region, it is essential that the line intersection test be exact. Both false positives and false negatives are problematic. Either case will cause cascade errors in the edge cutting algorithm, and logic capable of identifying and correcting for such errors would be both complex and computationally expensive.

While the point sets generated during the boundary curve simplification process will be integer valued, the vector data may be rasterized at any resolution. Thus, the boundary line points input to the edge cutting algorithm will often be floating point values. It is possible that precision errors in the line intersection test computations may lead to incorrect results. To avoid such precision errors, both the input data and line intersection algorithm are modified, to ensure that the results of the line intersection test will always be correct.

The strictly vertical and strictly horizontal boundary line special cases both require no floating point calculations, beyond the initial floor and ceiling operations, thus, it is only the diagonal case that requires special treatment to avoid precision problems.

First, assume the boundary line segment is parameterized by:

$$L(t) := \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0), \quad t \in [0,1].$$

To simplify the following discussion, further assume the algorithm needs to check the intersection of $L(t)$ with the horizontal edge line $(i,j) - (i+1,j)$, and that both of $\mathbf{p}_0$'s coordinates are less than $\mathbf{p}_1's$ coordinates. In order to avoid the potential loss of precision from floating point division operations, the y-coordinate intersection check is performed as follows:

$$j = y_0 + t(y_1 - y_0) \quad \rightarrow \quad t(y_1 - y_0) = j - y_0.$$

The boundary line passes through the line $y = j$ if the above equation is satisfied for some $t \in [0,1]$, or, equivalently, if $(j - y_0) \in [0, y_1 - y_0]$. Checking the latter condition requires only floating subtraction and comparison operations. If the condition is passed, then the boundary line will intersect the edge line if $L(t)_x \in [i, i+1]$. This condition can be reexpressed as follows,

$$\begin{aligned} & x_0 + t(x_1 - x_0) \in [i, i+1] \\ & \quad \Rightarrow \quad x_0(y_1 - y_0) + j(x_1 - x_0) - y_0(x_1 - x_0) \in [i(y_1 - y_0), (i+1)(y_1 - y_0)]. \end{aligned} \tag{3.1}$$

Thus, the line intersection test can be performed without the possibility of error if the floating point multiplication, addition, and comparison operations in equation (3.1) can be evaluated without precision errors.

I now derive conditions sufficient to ensure exact evaluation of equation (3.1), assuming IEEE 64bit floating point numbers.

First consider the case of two positive numbers expressed as fixed point bitstrings I,

$$f = \sum_{i \in I \subset [i_0, i_1]} 2^i.$$

Clearly, any product of the form $f_1 f_2$ must lie in the range $[2^{2i_0}, 2^{2i_1+2})$. (The largest possible product is that when $f_1 = f_2 = 2^{i_1+1} - 2^{i_0}$, which is slightly smaller than 2^{2i_1+2}.) Therefore, all such products can be expressed as fixed point bitstrings J of the form,

$$f_1 f_2 = \sum_{j \in J \subset [2i_0, 2i_1+1]} 2^j.$$

Somewhat more simply, note that:

$$f_1 - f_2 = \sum_{i \in I \subset [i_0, i_1]} 2^i, \text{ if } f_1 >= f_2.$$

And

$$f_1 + f_2 = \sum_{i \in I \subset [i_0, i_1+1]} 2^i.$$

Thus, by guaranteeing that the smallest significant bit of any line coordinates will be i_0, and the largest significant bit will be i_1, it is possible to guarantee that the expression $(x_1 - x_0)$ will share the same significant bit bounds. The significant bit bounds of $y_0(x_1 - y_1)$ are $2i_0$, $2i_1 + 1$. And the bounds of

$$x_0(y_1 - y_0) + j(x_1 - x_0) - y_0(x_1 - x_0)$$

are $2i_0$, $2i_1 + 2$. (The maximum of any of the summed expressions is less than 2^{2i_1+1}, so a sum of three such expressions must be less than 2^{2i_1+3}, therefore, the largest possible result is 2^{2i_1+2}. As a concrete example, consider, $3 \times 7 < 32$.)

Now all that remains is to choose values of i_0, i_1 such that the resulting expressions will imply numbers that do not overflow the precision of IEEE 64bit floating point. The multiplication algorithms used by most CPUs are proprietary, so the exact point at which precision errors can result is unclear. However, given that there are 52 mantissa bits in a 64 bit float, it is safe to assume that a significant bit range of 50 or less will be sufficient to ensure that all computations are exact. This, in turn, requires choosing i_0 and i_1 such that

$$2i_1 + 2 - 2i_0 \leq 50 \Rightarrow \quad (i_1 - i_0) \leq 24.$$

Choosing $i_0 = -9$, $i_1 = 15$ provides a range of allowed line coordinate values that should be more than sufficient to handle any reasonable boundary line inputs. I trusting users to never input coordinate values larger than 2^{i_1}, while I enforce the least significant bit constraint by truncating the input line endpoint coordinates such that any bits corresponding to powers smaller than 2^{i_0} are set to 0. An even larger range of significant bits could be allowed by implementing the line intersection tests with 64 bit integers, after scaling all input points by 2^{-i_0}.

While using 64 bit floats makes it possible to choose comfortably large bounds, 32 bit floats would force some unpleasant choices. The maximum significant bit range of 32 bit floats is 24, which requires $i_1 - i_0 \leq 11$. As a very high resolution render might reasonably involve a $2^{12} \times 2^{12}$ grid, this is an unacceptable constraint. That said, if speed or hardware constraints did force the use of 32 bit floats, it could be possible to vary i_0 and i_1 depending on the grid resolution – thus allowing low resolution image reconstructions to use points specified with sub pixel accuracy, while the maximum resolution images would limit line coordinates to pixel corners. Even so, such a system could still render, at best, $2048x2048$ images. Use of the sign bit might make it possible to push the effective maximum resolution close to 4096×4096. Additionally, inserting additional control points into boundary line segments to lower the bit range of any difference terms could further increase the maximum possible resolution. Doing so would require careful engineering, however, and at present, the edge cutting algorithm is one of the fastest components of the system, even given the use of 64 bit floating point computations.

3.4 Locally Sequential Image Operations

Gathering luminance samples requires an algorithm for generating an integer label corresponding to each connected component in the pixel adjacency graph. The algorithm used to find these labels is a member of a larger family of algorithms, and several other algorithms belonging to that family are used in other parts of my system.

In 1966, Rosenfeld and Pfaltz identified several related digital image processing problems for which local sequential processing would, necessarily, be more efficient than local parallel computation [61]. Two of these operations were connected components labeling and distance function computation. In my system, both operations are used frequently during image vectorization and reconstruction, and their implementations are both similar to the original "local sequential" definitions given by Rosenfeld and Pfaltz.

Formally, the *connected components* of an undirected graph having vertices V and

edge set E are a partitioning of V into components $C \subset V$ having the property that if vertex $i \in C$, then for every $j \in C$ it is possible to form a list of adjacent vertices that includes both i and j.

The *connected components labeling* problem is defined as follows. Given a pixel adjacency graph, find an integer label $L(i)$ for each $i \in V$ that identifies the connected component to which each pixel index i belongs. Additionally, find n, the number of connected components in the image, and ensure that $L(i) \in [0, n-1]$, for all $i \in V$.

When the connected components problem occurs in an image processing context, the graph (V, E) can typically be assumed to be a subgraph of either the 4-connected or 8-connected pixel adjacency graph. In the following discussion, I assume the 4-connected case, thus edges are only allowed between pixels that are immediate horizontal or vertical neighbors.

The connected components labeling problem sometimes occurs when there is a need to assign a unique identifier to each of the regions found by a quantization operation. In such cases, the edge set E is defined as a function of the quantized image values, $q(i)$, as follows,

$$E := \{\ (i,j) \mid q(i) = q(j), \text{ and the grid position of } i \text{ is adjacent to that of } j\ \}.$$

However, the edge set E can also be defined using the result of the edge cutting algorithm given in Section 3.3.

Given $\mathbf{p}(i)$, the image space position of any pixel i, and a set of pixels $U \subset V$, the *distance function* problem requires calculating values $D(i)$ such that,

$$D(i) = \min_{k \in U} ||\mathbf{p}(i) - \mathbf{p}(k)||.$$

Closely related to the distance function problem is the *nearest point* problem, which requires calculating, for each pixel i, the set of points $S(i) \subset U$ such that,

$$S(i) = \{k \mid k \in U, \mathbf{p}(k) \text{ is minimally distant from } \mathbf{p}(i)\}.$$

Connected components labeling, distance function computation, and nearest point computation are all related, in that they can be solved by iteratively applying a local update function.

A local update operation is an operation that updates a value stored at each pixel of an image based on the values stored in the pixel's neighbors. Let $N(i)$ be the set of neighbors of i, expanded such that $i \in N(i)$. Formally, define

$$N(i) := \{j \mid (i,j) \in E \text{ or } j = i\}.$$

In the case of the connected components labeling problem, a useful local update operation is,

$$L(i) = \min_{j \in N(i)} L(j). \tag{3.2}$$

If the label values at each pixel are initialized to $L(i) = i$, and equation (3.2) is applied to all pixels repeatedly, $L(i)$ will eventually converge to a unique id for every connected component in the image. Then all that remains to solve the connected components labeling problem is to relabel the pixels such that $L(i) \in [0, n-1]$.

In the case of the nearest point problem, a useful local update operation is,

$$\begin{gathered} S(i) = \{k | k \in S', \mathbf{p}(k) \text{ is minimally distant from } \mathbf{p}(i) \}, \\ S' := \text{union of all } S(j), \text{ where } j \in N(i). \end{gathered} \tag{3.3}$$

If $S(i)$ is initialized to the empty set for all $i \neq U$, and $\{i\}$ for all $i \in U$, then applying equation (3.3) repeatedly will eventually lead to a stable state which solves the nearest point problem. Solutions to the nearest point problem also imply solutions to the distance function problem, as $D(i)$ can simply be defined as the distance between i and any element of $S(i)$.

As demonstrated by Rosenfeld and Pfaltz, sequentially applied local update operations have the potential to propagate information very quickly throughout an image. The number of times that an update operation must be applied to all pixels before convergence depends on both the complexity of the input and the order in which the image data is traversed, with row-major traversals being effective in many situations [61].

3.4.1 Connected Components Labeling

In the general case, identifying the connected components of an arbitrary graph is a common software engineering problem. And a general purpose connected component algorithm can be found in Chapter 21 of Cormen et al.'s *Introduction to Algorithms* [15].

As an alternative to using local update functions, the standard connected component algorithms given in Cormen et al. can be used to generate a connected components labeling in images. A set of equivalence classes, which can be formally expressed as a set of disjoint sets, $\mathscr{S} = S_1, ..., S_k$, is initialized by adding a single element set $S_i = i$ corresponding to each vertex of the graph. Then, for edge in the graph, the equivalence classes of the two incident vertices are merged. Finally, the label of each vertex is assigned based on the identifier of its equivalence class. As discussed in Cormen et. al, there is a necessary trade off between the time required to merge two equivalence classes, and the time required to identify the equivalence class to which a given vertex belongs; but a category of techniques called *disjoint set forests* allow both operations to

be performed quickly. It is possible to prove that by using disjoint set forests, the runtime bound of connected components labeling on 4-connected image adjacency graphs will be $O(\ |V|\alpha(|V|)\ )$, where α is a function which grows so slowly that $\alpha(|V|)$ can be relied on to be at most 4 in any practical situation. Thus, the runtime a disjoint set forest implementation of connected components labeling is effectively linear.

However, in practice, the overhead costs of a disjoint forest implementation on a megapixel sized pixel adjacency graph are significant. In a naive application of the method, representing the disjoint set forest will require as much memory as representing the source image. Given megapixel images, that implies that read or write operations into the disjoint forest data are likely to be made significantly more expensive as a result of frequent $L1$ cache misses.

The overhead costs of a disjoint forest implementation can be substantially reduced by applying one or more passes of the local label update operation, given in equation (3.2). For example, assuming a $2^{10} \times 2^{10}$ image, applying the disjoint forest algorithm directly would imply initializing storage space for 2^{20} separate equivalence classes. However, a single row-major sequential pass of equation (3.2) typically reduces the number of distinct labels by 2 or 3 orders of magnitude, which in turn implies a dramatic reduction in the memory overhead of the datastructures needed to represent the disjoint set forest, and thus a dramatic reduction in the chances of cache misses when performing the equivalence class operations.

Ali Rahimi's *Fast Connected Components on Images* webpage includes an optimized implementation of such a combined local sequential, disjoint forest connected components labeling algorithm [58]. It performs a single row-major sequential pass of the local update function, and implements the disjoint set forest operations using a variation on the *path compression* heuristic described in Cormen et al. The connected components labeling algorithm used in my system is based on Rahimi's, but modified to be compatible with the adjacency graphs created by the edge cut algorithm of Section 3.3.

3.4.2 Nearest Point Propagation

A naive, nonlocal implementation of the nearest point algorithm requires checking the distance between each pixel vertex and each element of the point set U, for a total of $O(|V| \cdot |U|)$ operations. As explained in Section 4.3.4, nearest point sets must be found for each pixel using sets U that correspond to the edges removed by the edge cutting algorithm, thus $|U|$ is typically large. For a megapixel image, generating an initial point set as discussed in Section 4.3.4 will typically result in $|U| > 2^{14}$, thus the number of distance calculations required for a naive solution to the nearest point algorithm will

be at least 2^{34}, an impractically large computational expense.

The local update operation in equation (3.3), however, is relatively cheap to apply. Specifically, the cost of updating the nearest point set $S(i)$ for each pixel in the image is $O(\ |V| \cdot \max_i(|S(i)|)\ )$. While it is possible to construct pathological cases in which a single pixel is equidistant from all points in $|U|$, in practice, $\max_i(|S(i)|)$ tends to be small – pixels equidistant from more than 4 edge points are very unusual.

As is the case in the L1 distance function algorithm discussed by Rosenfeld and Pfaltz, the convergence speed of the nearest point algorithm can be significantly improved by varying the the sequence in which the pixel set is traversed [61]. For example, in the special case in which the point set U contains only a single point k, and $\mathbf{p}(k) = (0,0)$, the algorithm will converge after a single pass through the image, given row-major traversal order. However, if $\mathbf{p}(k) = (w-1, h-1)$, the algorithm will require $\max(w-1, h-1)$ passes before convergence. Alternating row-major and reverse row major traversal orders will guarantee convergence in 2 passes, for any case in which there is only a single element of in U. For the $L1$ distance case discussed by Rosenfeld and Pfaltz, the 2 pass convergence guarantee holds for arbitrary $U \subset V$. However, for the purposes of image reconstruction, the nearest point set relative to standard Euclidean distance is required, thus, more than 2 iterations are typically necessary.

As illustrated in Figure 3.5, the order in which the pixels are updated impacts whether or not it is possible that information changed at one pixel may affect the result at any other. In the case of a row major traversal, a pixel at (i_0, j_0) may inherit information from a pixel at (i_1, j_1) if $i_0 \leq i_1$ and $j_0 \leq j_1$. Note that the results of a row major traversal of the image will always be identical to that of a column major traversal, as the two traversal orders imply identical information flow.

As shown in Figure 3.6, there are three simple variations on row major traversal which imply different patterns of information flow. Alternating between these 4 traversal orders causes the nearest point set update to converge quickly. Even on large, complex images, convergence is typically achieved after less than 10 passes. Thus, solving the nearest point set problem is typically a cheap operation, in comparison to the linear solver iterations that form the main bottleneck of the reconstruction process.

Because they depend on sequential execution, neither the connected components algorithm nor the nearest point algorithm would be natural to port to graphics hardware. Yet both algorithms are foundational tools for many image analysis tasks, thus, it is likely that nearest point and connected component algorithms that can execute quickly on graphics hardware will soon be developed. Rong has recently explored GPU algorithms for the related problem of distance function evaluation, and similar methods could be used for nearest point computation [60].

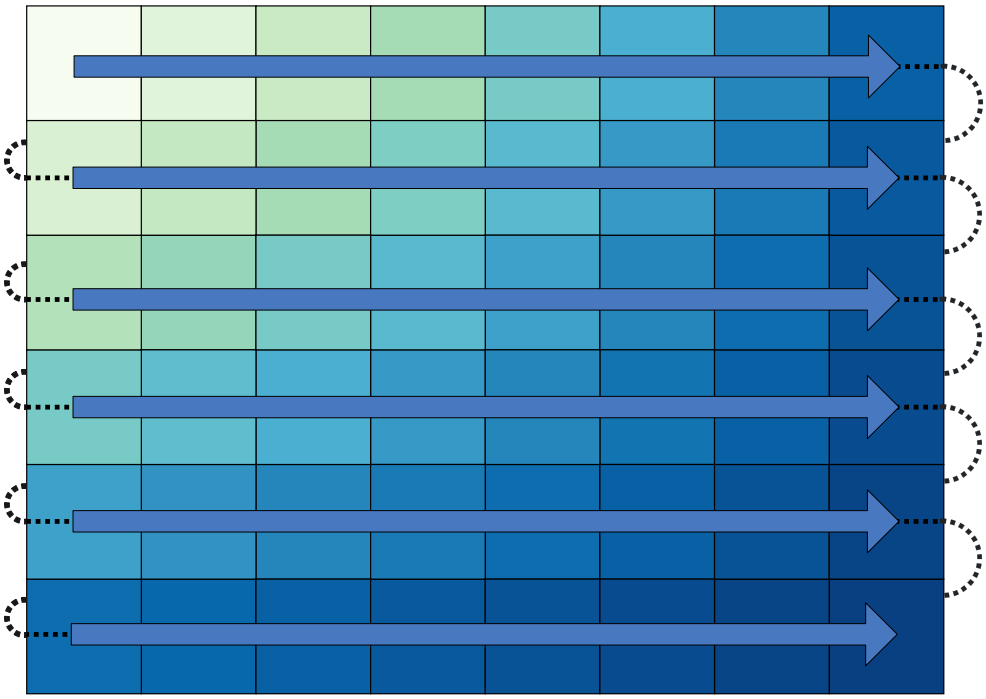


Figure 3.5: Information Flow, in the case of a local update operation applied during row major traversal of a 4-connected image. Pixels have a chance of inheriting information from neighboring pixels that were updated before the traversal reached it. The brightness of each pixel in the 8×6 image above corresponds to the number of pixels that may inherit information from it.

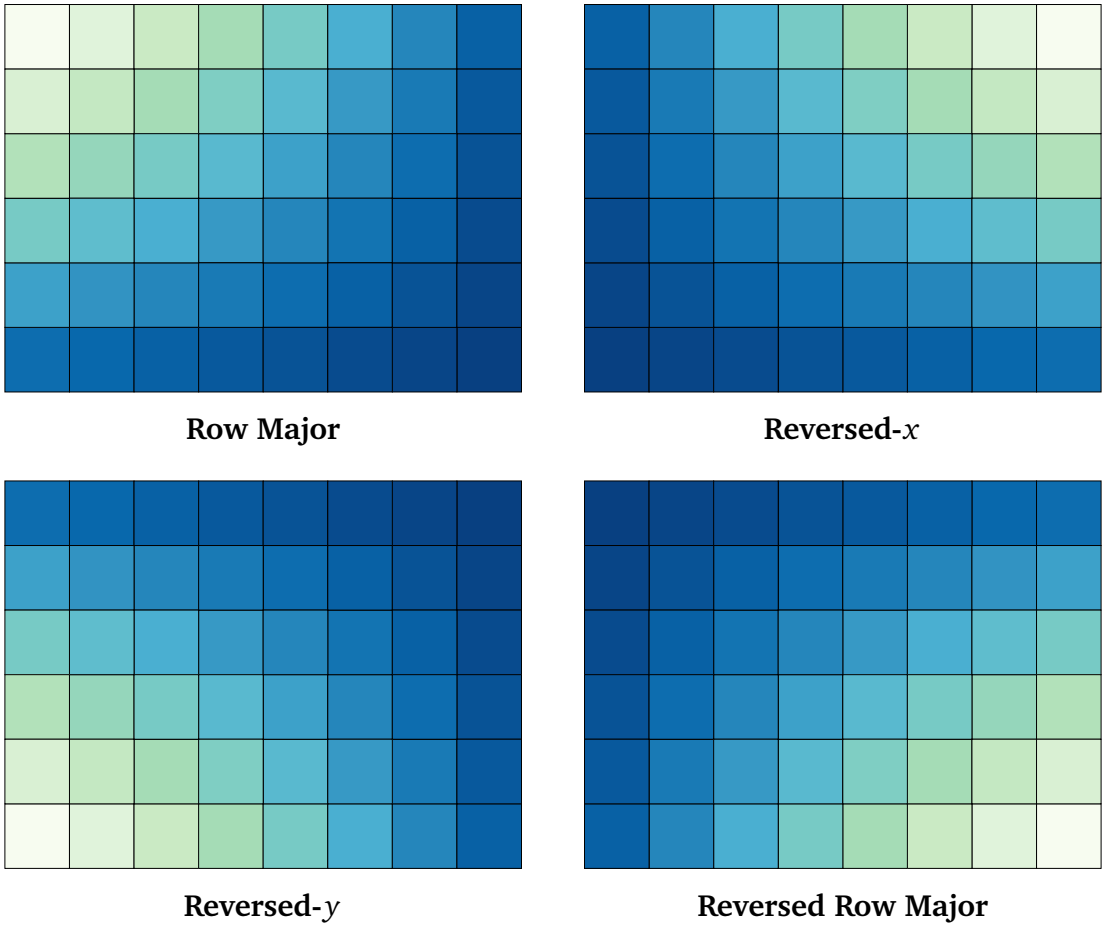


Figure 3.6: Information flow, for the four traversal orders used during a nearest point set solve. The order of traversals used by the system is row major, reversed row major, reversed-y, and reversed-x. A total of 8 traversals is typically sufficient to ensure convergence of the nearest set values.

4 Image Reconstruction

Was nicht translationsinvariant ist, ist Scheiße in jeder Dimension.

– Joachim Weickert to Thomas Brox, on the topic of discrete methods for solving anisotropic diffusion equations.

The boundary curve data gathered in Chapter 3 implies a segmentation of the image pixels into connected components. The characteristic luminance values stored for each of these components can be used to construct a piecewise constant image. That piecewise constant image is henceforth referred to as the *target image*.

Calculating the target image is straightforward, given the connected components labeling algorithm defined in Section 3.4.1. The central image reconstruction problem is thus to modify this piecewise constant image to create a piecewise smooth image that reflects, as accurately as possible, the edge sharpness values stored along the boundary lines.

This chapter describes how *anisotropic regularization* can be used to generate such a piecewise smooth image. Anisotropic regularization is a an image smoothing technique closely related to Poisson's equation [5]. I show how regularization weights may be chosen to minimize deviations from the edge model derived in Section 2.4.2, while simultaneously providing a result that avoids introducing creases or high gradients at points other than the established edge set.

4.1 Chapter Overview

Sections 4.2 and 4.3 are concerned with the problem of finding a regularization definition that will imply that edges are reconstructed as accurately as is possible. By considering the very simple case in which the image consists of only a single edge, I identify a large class of regularizations that, for the single edge case, imply regularization results that are perfect matches for the edge model curves. In Section 4.2.1, I identify a specific regularization which is related to this class, and which is suitable for generalization to the multiple edge case. Importantly, this regularization definition is designed to avoid creasing artifacts that would otherwise arise in the 2D, multi-edge

case. The full 2D regularization definition is given in equation (4.10).

The regularization definition found in Section 4.2.1 assumes that the regularization operation can be implemented using continuous functions. To obtain more accurate reconstructions given discrete image data, I convert the continuous regularization problem given in equation (4.2) to a system of linear equations, using a standard finite difference scheme. Sections 4.3.2 and 4.3.3 adapt the continuous arguments from Section 4.2.1 to the discrete 1D case, thus deriving a natively discrete regularization definition. This discrete definition is significantly more complex than the continuous definition found in Section 4.2.1. Generalizing the 1D, single edge definition to a 2D, multiple edge definition is also more complex in the discrete case. Section 4.3.4 defines the 2D regularization for the discrete case.

In Section 4.4, I show that the additional complexity of the discrete definitions is justified, as it leads to significantly more accurate edge reconstructions, while minimizing the sensitivity of the reconstruction to the resolution of the discrete grid. Section 4.4 also includes studies of the interactions of the key parameters used in defining the regularization equations. Specifically, k, K, ϵ, and h. In Section 4.4.3, I show that anisotropic regularization is both faster and produces higher quality results than the related image reconstruction algorithms of Elder [20] and Orzan et al. [52].

4.2 Regularization

Consider first the very simple case in which the piecewise constant target image can be described as a step function, defined in Iverson notation as,

$$v(x) := [x > 0] - [x < 0].$$

This requires the assumption that, though the image is two dimensional, the value at any point is determined solely by the x-position. Additionally, assume that the domain of the x-coordinates of the image is given by $\Omega_x = [-r, r]$, thus r is half the width of the image.

There is only one boundary line in this image. It occurs at $x = 0$. Assume that the edge sharpness parameter at this edge is k, so the correct smooth image is defined by the exponential sigmoid curve,

$$\text{esig}(kx) = \begin{cases} 1 - e^{-kx} & \text{if } kx \geq 0, \\ e^{kx} - 1 & \text{otherwise.} \end{cases} \tag{4.1}$$

Now assume that $f(x)$ must be reconstructed from $v(x)$ using an energy minimization process, in which $f(x)$ is defined as the minimum of the following functional,

$$\min_f \int_{-r}^{r} w_1(x)|f'(x)|^2 + w_0(x)(f(x)-v(x))^2 dx, \quad \text{subject to } f'(r) = f'(-r) = 0. \tag{4.2}$$

Here $w_0(x)$ is a weighting function that determines how strongly $f(x)$ is attracted to the target image $v(x)$, while $w_1(x)$ is a weighting that causes $f(x)$ to tend towards smooth functions.

Equation(4.2) is an *anisotropic regularization* problem [5]. Minima of the functional will be smoothed versions of the target image $v(x)$, and are referred to as *regularizations* of $v(x)$. Regularization functionals occur frequently in computer vision applications, and many different choices for the weightings functions w_1 and w_0 have been studied in that context. Perhaps the most famous weighting definition is *total variation regularization*. Repeated applications of total variation regularization can be used to implement *total variation diffusion*, an algorithm which has proven useful in the contexts of edge detection, image segmentation, texture classification, and optical flow estimation. Indeed, total variation methods have become sufficiently popular that they may be considered a significant area of study in and of themselves [10].

4.2.1 Solutions of the Variational Problem

Depending on the choice of weighting functions, equation (4.2) may be ill-posed. For example, in the case that $w_0(x) = 0, w_1(x) = 1$, the functional will be minimized by any constant valued f. Also note the discontinuity in v, which implies that applying the Euler-Lagrange equation directly to equation (4.2) will yield a differential equation that is merely a necessary condition for minimizers, rather than a sufficient condition, as would be the case if v were C^2 continuous [22].

However, assuming that w_0 and w_1 are even and strictly positive implies that any minimizing f of equation (4.2) must be odd—a short proof is given in Section 4.2.2. Strictly positive, even weighting functions simplify the task of generalizing the regularization definition to the case of more complex edge sets, so in practice little is lost by making this assumption.

To avoid the presence of undefined terms in equation (4.2), f must be C^1 continuous. Given f both C^1 continuous and odd, it is possible to re-express the initial regularization in a simpler form. Given odd, continuous f, it follows that minimizing

equation (4.2) is equivalent to minimizing,

$$\min_f \int_0^r w_1(x)|f'(x)|^2 + w_0(x)(f(x)-1)^2 dx, \quad \text{subject to } f'(r) = f(0) = 0. \tag{4.3}$$

Given the prior assumption of strictly positive weighting functions, the functional in equation (4.3) is convex, and thus the regularization is provably well-posed [21].

Upon finding a set of function definitions that minimize equation (4.3) in the domain of $x \in [0, r]$, those definitions can be extended to the range $[-r, r]$ using,

$$f(-x) := -f(x), \quad w_1(-x) := w_1(x), \quad w_0(-x) := w_0(x), \quad \forall x \in (0, r].$$

The single dimensional Euler-Lagrange equation may be written as follows:

$$f \text{ minimizing } \int L(f, f')dx \quad \text{must satisfy} \quad \frac{\partial L}{\partial f} = \frac{\partial}{\partial x}\left(\frac{\partial L}{\partial f'}\right).$$

Applying the Euler-Lagrange equation to equation (4.3) yields the result that any minimizing f must satisfy the second order ordinary differential equation,

$$w_0(x)(f(x)-1) = f'(x)w_1'(x) + w_1(x)f''(x). \tag{4.4}$$

By forcing the regularization result $f(x)$ to be a perfect reconstruction of the correct image $\text{esig}(kx)$, equation (4.4) can reinterpreted as a constraint on possible choices for w_1 and w_0. This transformation of a minimization problem in f to a differential equation in w may seem unusual, but there is ample precedent for it. Any physics problem that requires recovering the set of forces capable of implying an observed motion path may be solved using the same method. Such physics applications are known as *inverse problems*, and both the theory and the history of inverse problems are closely related to variational calculus [26]. In fact, the edge reconstruction method presented here is multiply indebted to the history of inverse problems, as before they were adapted to the fields of computer vision or computer graphics, regularization energies were used by physicists as a tool for narrowing down the solution sets of inverse problems [3].

After assuming $f(x) = \text{esig}(kx)$ for $x \in [0, r]$, equation (4.4) simplifies to the following differential equation, [1]

$$w_0(x) + kw_1'(x) - k^2 w_1(x) = 0. \tag{4.5}$$

[1] Weight functions for alternate sigmoidal model curves can be derived by substituting other sigmoids into equation (4.4). I have chosen to use the exponential sigmoid as my edge model because it leads to an unusually simple family of differential equations. Note that the exponential sigmoid is the only sigmoidal curve that can be expressed as the regularization of a step function using constant-valued weight functions.

Note, however, that forcing $f(x)$ to equal esig(kx) introduces a complication, as esig(kx) does not obey the boundary conditions set in equation (4.3), except in the limit as $r \to \infty$. Thus, given finite r, weight functions derived by solving equation (4.5) will never exactly imply $f(x) = \text{esig}(kx)$. However, it is reasonable to assume that $f(x)$ will approach esig(kx) in the case of large images. In Section 4.4, I show several experimental results that strongly support this assumption.

4.2.2 Proof that f is odd

The following is a short proof that f is odd, a result necessary to derive equation (4.3).

Recall first that any function f may be uniquely decomposed into even and odd components f_e and f_o, using,

$$f_e(x) = \frac{1}{2}(f(x) + f(-x)), \quad f_o(x) = \frac{1}{2}(f(x) - f(-x)).$$

For brevity, I will now write functions $f(x)$ using single letter variables f. Equation (4.2) may be rewritten as,

$$\int_{-r}^{r} w_1 \left((f_o')^2 + 2 f_o' f_e' + (f_e')^2 \right) + w_0 \left((f_o - v)^2 + f_e^2 + 2 f_e (f_o - v) \right) dx.$$

Recall that the sum of any two odd functions is odd, and the product of an even and odd function is odd. Also, the integral from $-r$ to r of any odd function will be 0. Finally, recall that the derivative of an even function f_e' must be odd, and that the derivative of an odd function f_o' must be even. Thus, assuming even w_1, w_0 implies,

$$\int_{-r}^{r} w_1 (2 f_e' f_o') + w_o (2 f_e (f_o - v)) dx = 0.$$

Additionally assuming $w_1, w_0 > 0$ implies,

$$\int_{-r}^{r} w_1 \left((f_o')^2 + (f_e')^2 \right) + w_0 \left((f_o - v)^2 + f_e^2 \right) dx \geq \int_{-r}^{r} w_1 (f_o')^2 + w_0 (f_o - v)^2 dx.$$

Thus proving that even, strictly positive weighting functions force any f minimizing equation (4.2) to be odd.[2]

[2]This short proof was shown to me by Reinhard Illner; to whom I am indebted not only for this proof, but also for several helpful conversations on the topic of the calculus of variations and related inverse problems.

4.2.3 Weight Function Definitions

There are many weighting functions that satisfy equation (4.5). And if the only images input to the system were very large images that contained only a single straight edge line, any of these possible definitions would work equally well. However, while it is important that the reconstructed image be correct in the single edge case, it is also important that any weighting function definitions lead to reasonably good reconstructions given much more complex edge sets. I have found that some weighting function definitions lead to better reconstructions of complex edge images than others, even if both functions produce perfect results in the single edge case.

I now provide the w definitions used in my system. These definitions have proven to yield good reconstructions even given complex edge information and gradient samples, but they are derived by enforcing the condition that the regularization produce nearly perfect results when the image to be reconstructed contains only a single edge.

In order to find solutions to equation (4.5), additional constraints must be placed on the weighting functions. The first of these constraints is that the weighting functions take on the values $w_1(x) = 1$ and $w_0(x) = K^2$, for some large value of x. Formally, this constraint is stated as follows:

$$w_1(y) = 1, w_0(y) = K^2, \text{ for } y \text{ such that } 1 - \text{esig}(ky) = \epsilon. \tag{4.6}$$

I use a different set of weighting function definitions depending on whether or not the edge sharpness parameter k is less than K. Reconstruction errors become increasingly likely when k is either much greater or much smaller than K, thus it is beneficial to set K to be a value that represents an average level of edge sharpness. I use $K = 0.3$.

For the case of a low sharpness edge, in which $k \leq K$, I use a constant data weighting function $w_0(x) := K^2$. This reduces equation (4.5) to a first order ordinary differential equation. It can be solved as an initial value problem given the constraints in equation (4.6) [55]. Solving for w_1 and then extending the definition to $\mathbb{R}$ by enforcing the assumption that w_1 is even yields,

$$w_1(x) = \frac{K^2}{k^2} + e^{k|x|}\epsilon(1 - \frac{K^2}{k^2}). \tag{4.7}$$

This weighting function definition implies that for values of k smaller than K, the gradient smoothness weight w_1 will be large close to the edge line at $x = 0$. However, w_1 decreases as the distance from the edge increases. The data weight w_0 will, meanwhile, be constant at all points in the image. Note that for the average edge sharpness case, defined by $k = K$, the equation for w_1 simplifies to $w_1(x) = 1$. The main drawback of

these weighting function definitions is that $w_1(x)$ is negative for very large x. However, clamping w_1 to have a minimum value of 1 causes minimal reconstruction errors, provided that ϵ is sufficiently small. The default setting used by my reconstruction system is $\epsilon = .005$.

A different pair of w definitions is needed to handle the high edge sharpness case, in which k is larger than K. I derive weighting functions for this case by enforcing the constraint that,

$$w_0(x)K^2 = w_1(x).$$

Solving equation (4.5) given this constraint yields,

$$w_1(x) = e^{(k-\frac{K^2}{k})|x|}\epsilon^{1-\frac{K^2}{k^2}}, \quad w_0(x) = K^2 w_1(x). \tag{4.8}$$

Like the low sharpness solutions, these weighting functions have undesirable behaviors far from $x = 0$. In this case, the problem is that the exponential increase in the two functions leads to unreasonably large weighting terms. However, as in the low sharpness case, the values can be clamped so that $w_1(x)$ is at most 1, and $w_0(x)$ at most K^2. Given a sufficiently small choice of ϵ, reconstruction errors due to clamping are minimal. Note that, after clamping is applied, these equations imply that for very large k, $w_1(x)$ approaches a constant function having a point discontinuity, specifically,

$$w_1(x) \approx \begin{cases} \epsilon & \text{if } x = 0, \\ 1 & \text{otherwise.} \end{cases} \tag{4.9}$$

This suggests that very sharp edges may be modeled using weighting functions defined as in the average $k = K$ case, but ignoring pixels in neighboring regions whenever calculating the gradient term associated with the smoothness weight, $w_1|f'|^2$. This is a concept that will prove important when defining the finite difference equations in Section 4.3.3.

Continuous Extension to 2D

In the general case, the reconstructed image $f(\mathbf{x})$, defined over image domain $\Omega \subset \mathbb{R}^2$, is found by solving the 2D regularization problem,

$$\min_f \iint_\Omega w_1(\mathbf{x})||\nabla f(\mathbf{x})||^2 + w_0(\mathbf{x})(f(\mathbf{x}) - v(\mathbf{x}))^2 d\mathbf{x}. \tag{4.10}$$

The 2D weighting functions $w_1(\mathbf{x})$ and $w_0(\mathbf{x})$ are defined by choosing the weighting function definition implied by the edge sharpness value k of the closest edge point $\mathbf{x}_e$, and evaluating those functions with $x = ||\mathbf{x} - \mathbf{x}_e||$. The partial differential equation implied by equation (4.10) may then solved using finite difference methods [63].

4.2.4 Box Function Regularization

Thus far, all my arguments for deriving regularization function weights have depended on assumptions that will never be true in practice—specifically, the assumptions that the image will contain only a single edge, and that the image will have infinite width. The gap between the image reconstruction problems that occur in practice and the idealized case is significant.

This is not an uncommon problem—Elder's model-functions for edge interfaces are derived using the same faulty assumptions [20]. However, the errors introduced by these assumptions are typically less serious than they might first appear. Additionally, there are good reasons to ignore many of the "errors" caused by the mismatch between the sigmoidal edge model functions and the results of a regularization operation. The most important reason is that the sigmoidal curves themselves are a necessarily imperfect approximation of the source image data.

For large regions one can reasonably expect the differences between the regularization results and the model sigmoid curves to be small. However, consider the case of a small narrow region—in which the underlying luminance briefly moves up into the neighboring quantization bin, before moving back down to its original bin value.

A one dimensional analog to this case can be represented as a box-function regularization. Applying the average case, $k = K$ weight functions $w_1(x) := 1, w_0(x) := k^2$ to an arbitrary target image $v(x)$ implies that the the differential equation characterizing the regularized image f will be:

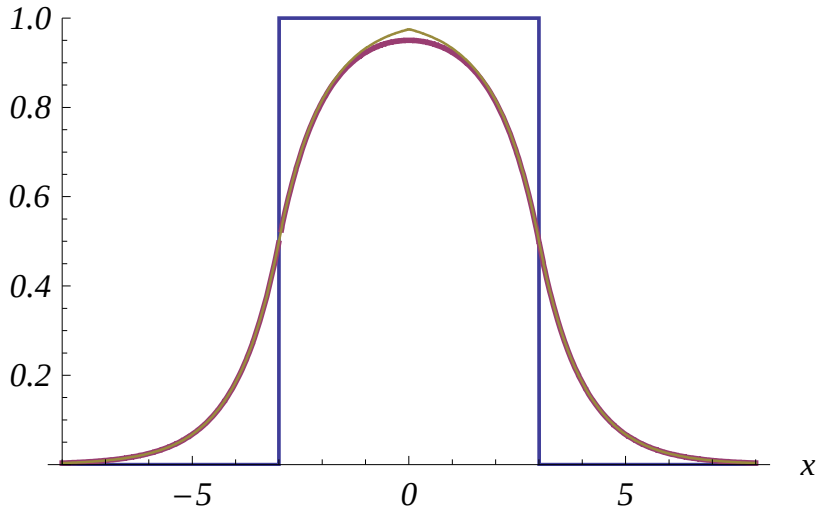
$$f''(x) - k^2 f(x) = k^2 v(x). \tag{4.11}$$

A function definition that corresponds to the problematic case of a box function regularization is $v(x) := c[x_0 < x < x_1]$. This reduces equation (4.11) to a second order nonhomogeneous linear ordinary differential equation, having constant coefficients [67]. Applying the variation of parameters method to the equation, and enforcing the boundary condition that $f(x)$ must converge towards 0 at either extreme of the real line yields:

$$f(x) = \frac{c}{2} \begin{cases} e^{kx}(e^{-kx_0} - e^{-kx_1}) & \text{if } x < x_0 \\ 2 - e^{-k(x-x_0)} - e^{k(x-x_1)} & \text{if } x_0 \le x \le x_1 \\ e^{-kx}(e^{kx_1} - e^{kx_0}) & \text{if } x > x_1 \end{cases} \tag{4.12}$$

These curves will be very similar to the exponential sigmoid, when the interval

characterizing the box function is large. For example, consider the following plot:

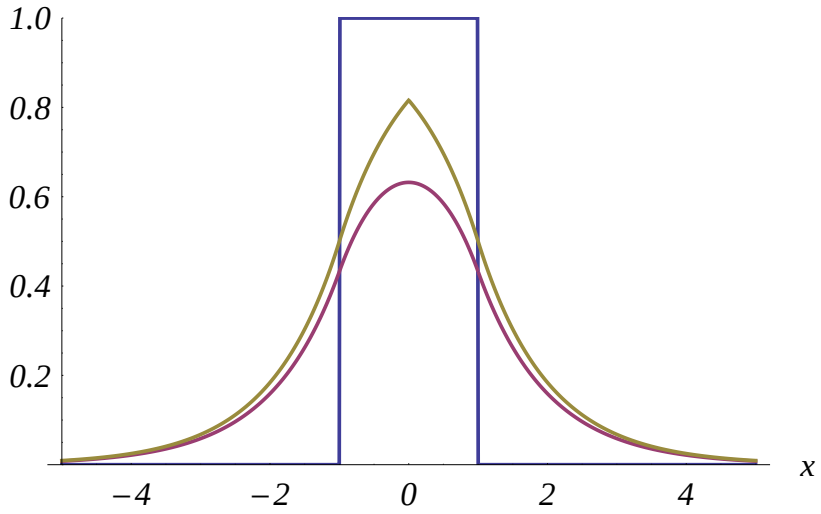


The target value function $v(x)$ is shown in blue, the regularization result as defined by equation (4.12) is shown in purple, and the piecewise combination of exponential sigmoid curves implied by the sigmoidal edge model is shown in yellow.

Notice that while the difference between the regularization result and the sigmoidal curves is very slight at most x values, there is a noticeable separation at the center of the box function. This separation is greatest at the transition point between the two edge model curves, where the piecewise-sigmoidal curve forms a visible crease. Such a crease implies that somewhere in the original image, the local linearity arguments used to justify sigmoidal edge model curves have failed to hold (see Section 2.4.2). In fact, the presence of such creasing artifacts is the main justification for the complexity of both Elder's and my own image reconstruction methods [20]. Were it not for the frequent occurrence of creasing artifacts, images could be reconstructed by simply assigning each pixel to the value implied by the model curve of the closet edge point.

Using a narrower box function yields more visible differences between the regular-

ization and the model curves:



Notice that even in this case, the behavior of the two curves is similar near region boundaries, while the most visible difference is that the regularization lacks the pronounced crease that occurs in the piecewise sigmoidal curve.

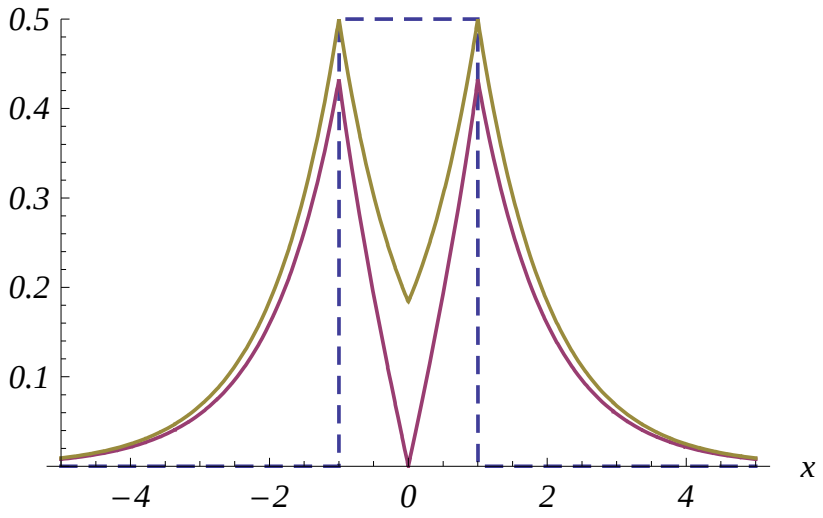
Note that the derivatives of $f(x)$ at the region boundaries are given by,

$$f'(x_0) = -f'(x_1) = \frac{c}{2}k(1 - e^{k(x_0 - x_1)}).$$

Thus the maximum gradient magnitude may stay relatively close to the expected value for a region interface, i.e., $|f'(x_0)| \approx \frac{c}{2}k$, even while the value of the function at the center of the region is much less than c. This is significant, as it is typically the deviations in luminance gradient, rather than changes to the maximum and minimum luminance values, that will be most noticeable to a human observer.

The following plot shows the absolute values of the derivatives of the exponential sigmoid curves (yellow) and the regularization result (purple) in the narrow box case.

The dashed blue lines show a scaling of the narrow box function, $\frac{1}{2}v(x)$.



Again, note that the largest differences occur near the center of the box function, which is also the region in which the sigmoidal edge model is least likely to be correct.

4.2.5 Relating the Edge Model to Target Image Regularization

All the equations for weight functions derived up until this point have assumed that the target image v will be limited to the values -1 and 1. In practice, the target image will have values defined by the quantization bin locations; for example, the bin values defined in Section 2.2 are $\mathbf{b} = (.2, .61, .95)$.

It is reasonable to expect that, if the target image v is changed to be some scaled, shifted version of itself, then the implied regularization result will be similarly scaled and shifted. For example, consider the following v_c definition, which would describe the target image for a single edge case in which the two region characteristic values are $c_2 \pm c_1$.

$$v_c(x) := c_1 v(x) + c_2.$$

Notice first that the energy term in equation (4.2) can be multiplied by any strictly positive constant c_1^2 without changing the minimizing function f.

$$\min_f \int_{-r}^{r} w_1 |f'|^2 + w_0 (f - v)^2 dx = \min_f \int_{-r}^{r} w_1 |c_1 f'|^2 + w_0 (c_1 f - c_1 v)^2 dx.$$

Adding $0 = c_2 - c_2$ to the w_0 term yields the result that minimizers f of equation (4.2) can be used to generate minimizers for any regularization problem of the following

form:

$$\min_{f_c} \int_{-r}^{r} w_1 |f_c'|^2 + w_0 (f_c - v_c)^2 dx, \quad \text{where} \quad f_c(x) = c_1 f(x) + c_2. \tag{4.13}$$

This formally proves a relation that was previously stated as intuition; if the target function can be represented as a scaled, shifted version of the base target function $v(x) := [x > 0] - [x < 0]$, then the implied regularization f_c will be a similarly shifted, scaled version of the minimizer f of equation (4.2). Importantly, proving this relation required no changes to the weighting function definitions w_0 and w_1. This has practical importance, as it implies that as long as the edge curves which we wish to reconstruct have the form of a shifted, scaled exponential sigmoid, the weighting function definitions do not need to be modified relative to the derivations in Section 4.2.3, regardless of the values of the scaling coefficients c_1 and c_2.

The edge model function found in Section (2.4.2) is such a scaled, shifted exponential sigmoid. Specifically, the behavior of the non-uniform soft quantization result across an edge j found in equation (2.14) is,

$$p_j(x,s) \approx h_j \frac{\text{esig}(\frac{s}{h_j}||\Delta I||x)}{\text{esig}(s)} + z_j.$$

In the above, x is the distance from the center of the edge, j is the index of the edge's sigmoid, h_j is the height of that sigmoid, and z_j is the edge's shift term. The parameter s holds the sharpness of the soft quantization, for which the system default is $s = 2$.

Thus, the edge model curve $p_j(x,s)$ has the form $f_c(x)$ given,

$$c_1 := \frac{h_j}{\text{esig}(s)}, \quad c_2 = z_j. \tag{4.14}$$

This implies that the edge model curve is a minimizer of equation (4.13), if and only if $\text{esig}(kx)$ is a minimizer of equation (4.2), and the sharpness term k is given by,

$$k := \frac{s}{h_j}||\Delta I||. \tag{4.15}$$

However, note that the constant definitions necessary to put the edge model curve p_j into the form of f_c also imply that the target image function v_c must be given as,

$$v_c(x) = \frac{h_j}{\text{esig}(s)} v(x) + z_j.$$

This is problematic. The piecewise constant image reconstructed from the sampled quantization bin values $\mathbf{b}$ will, near any edge j, have the form,

$$v_b(x) = h_j v(x) + z_j.$$

This is slightly different from the target image v_c. The problem is the division by esig(s). This division was introduced into the soft quantization to ensure that the function was C^0 continuous. However, it also implies that the convergent values of the model curve for any edge will be either slightly above or slightly below the characteristic bin values on either side of the edge. In practice, the difference between v_b and v_c is slight, as given the values of s used in the system, esig(s) ≈ 1.

It is possible to create a target image that is a multi-edge generalization of v_c, simply by defining j relative to the closest edge point. However, such a target image will contain discontinuities at the centers of quantization regions, and those discontinuities would lead to artifacts in the resulting reconstruction.

The purpose of including the division by esig(s) in the initial definition of the nonuniform soft quantization function was to avoid cases in which visible discontinuities would be introduced at the centers of quantization regions. Ironically, that adjustment now implies that reconstructing the edges in the soft quantization result as accurately as possible would require introducing exactly the same kinds of discontinuities into the target image.

It is thus useful to consider two different types of non-uniform soft quantization. One is the C^0 continuous definition given in Section 2.4, which includes the esig(s) division, and the other is an otherwise identical definition that omits the division by esig(s). The C^0 definition leads to image-space stylizations that are free of any artifacts at the centers of regions. However, it is the discontinuous, uncorrected definition that will most closely match the reconstruction results near image edges.

4.3 Finite Difference Implementation

Using classic finite difference approximations to solve equation (4.10) has a significant draw back. When the edge model functions esig(kx) are sufficiently sharp that they cannot be closely approximated by piecewise linear functions over the given grid discretization, then the shape of the reconstructed edges tends to vary significantly, depending on the number of points in the discretization. This is a problem, as the grid discretization will vary depending on the resolution at which the system is rendering the vector image, and, ideally, vector image renderings should be resolution independent.

This resolution dependence can be greatly reduced, however, by deriving a discrete analog to the differential equation (4.4). Using arguments that parallel those in Section 4.2.3, I derive weight function definitions that will imply $f(x) = \text{esig}(kx)$ at all points in the grid. Unlike the definitions given in equations (4.7) and (4.8), these

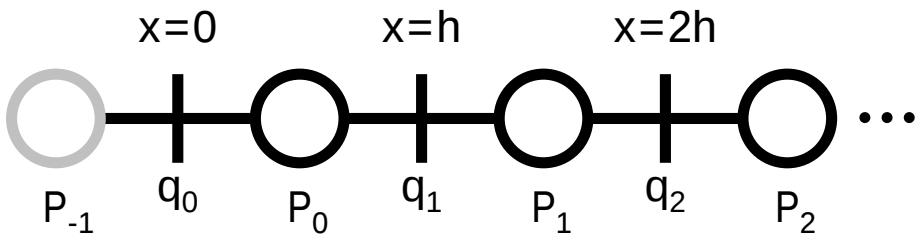


Figure 4.1: 1D Discretization coordinates. Functions that are most naturally defined at pixel centers are sampled at the circles, while functions more naturally defined on the edges between pixels are sampled on the vertical lines.

weight function definitions will depend on the discretization parameter h. As shown in Section 4.4.1, that dependence serves to correct for most of the distortions that would normally result from using coarse grids.

4.3.1 Discretization of the 1D Problem

In the discrete, one dimensional case, assume the edge weights w_1 are sampled at the positions between pixels. Therefore the sample positions are given by $q_i := ih$ for $i \in \mathbb{Z}$. The data weight values, and reconstructed image function function values, however, will be sampled at pixel positions, which are offset relative to the edge sample positions. They will be sampled at the x-values $p_i := ih + \frac{h}{2}$. Figure 4.1 illustrates the relation between the two sampling coordinates. Assume that the image is $2m$ pixels wide, so $r = mh$.

Discrete samplings of w_1, w_0 and f are then defined as follows,

$$\begin{gathered} g_i := \frac{w_1(q_i)}{h^2}, \quad d_i := w_0(p_i), \quad u_i := f(p_i), \\ \text{with} \quad q_i := ih, \quad p_i := ih + \frac{h}{2}. \end{gathered} \tag{4.16}$$

Now the variational problem in equation (4.3) can be discretized as,

$$\min_u \sum_{m>i\geq 0} d_i(1-u_i)^2 + g_i(u_i - u_{i-1})^2, \quad \text{with } u_{-1} := -u_0. \tag{4.17}$$

This discretization enforces the assumption that f is odd by defining $u_{-1} := -u_0$. The use of backward differences implies a discrete analog to the Neumann condition $f'(r) = 0$, as enforcing the boundary condition $u_m - x_{m-1} = 0$ is equivalent to defining $g_m := 0$, or, as in equation (4.17), not including any g_m term in the formula.

As equation (4.17) is a linear least squares problem, minimizing x are characterized by the following system of m linear equations,

$$d_i(1-u_i) = g_i(u_i - u_{i-1}) + g_{i+1}(u_{i+1} - u_i), \quad \forall i \in [0, m), \text{ with } g_m := 0. \tag{4.18}$$

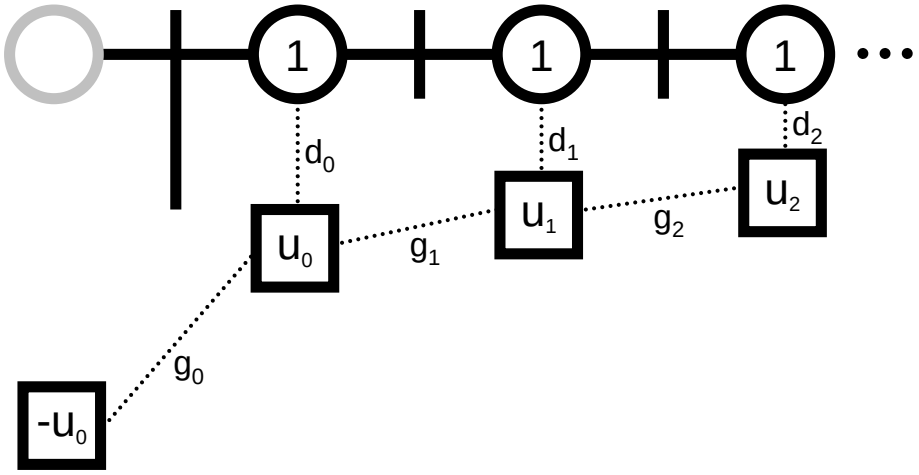


Figure 4.2: Relation between the data weights d_i, edge weights g_i, and the discrete reconstructed image u_i. The relationships are similar to those of a spring system. Each pixel value u_i is attracted to the value 1 by the dataweight d_i, but, it is also attracted to the values of its neighboring pixels, as a result of the edge weights g_i and g_{i+1}. The boundary condition $u_{-1} := -u_0$ is what causes the sigmoidal bend in $\mathbf{u}$ – without that condition, all the pixel values would snap to 1.

As in Section 4.2.1, replacing the reconstructed pixel values u_i with the model edge curve $\text{esig}(p_i)$ results in an under constrained linear system for the weight vectors g and d. Specifically, it results in a linear system of m equations having $2m$ unknowns.

$$d_i(1 - \text{esig}(p_i)) = g_i(\text{esig}(p_i) - \text{esig}(p_{i-1})) + g_{i+1}(\text{esig}(p_{i+1}) - \text{esig}(p_i)). \qquad (4.19)$$

Equation (4.19) is thus the discrete analog to the differential equation (4.4). Figure 4.2 illustrates the relationships between the variables d, g, and u.

Definitions for w_1 and w_0 are now derived by following the arguments in Section 4.2.3, with the modification that the linear system (4.19) is used in place of the differential equation (4.5).

As in the continuous case, it is helpful to consider the linear system (4.19) in the limit as m approaches infinity. There are two reasons for this. The first is that, in the multiple edge case, the choice of m that would imply the best possible reconstruction of the grey values at a given pixel is not typically half the width of the image. Rather, it is likely to be a complex function of both the size and shape of the connected component to which that pixel belongs, making an optimal m difficult to calculate in practice. Assuming $m \to \infty$ removes a potentially complex unknown from our system.

Assuming $m \to \infty$ also changes the finite linear system (4.19) into a recurrence relation in g_i. This makes it possible to find closed form expressions that satisfy the

weighting function constraints [29].

4.3.2 Corrected Definitions for the Soft Edge Case

I begin by searching for an analogue to the $w_1 = 1$, $w_0 = K^2$, case that defines the fixed points in equation (4.6). If $g_i = 1$, and $k = K$, equation (4.19) implies,

$$d_i = 2(\cosh(hK) - 1), \quad \forall i \geq 1.$$

This is a promising start, as this definition approaches the continuous case as h approaches 0. When the weight functions $w_1 = 1$, $w_0 = K^2$ are discretized, they imply $g_i := \frac{1}{h^2}$ and $d_i := K^2$. That discretization is equivalent to $d_i = K^2h^2$ and $g_i = 1$. And L'Hôpital's rule implies that,

$$\lim_{h\to 0} \frac{2(\cosh(hK) - 1)}{h^2} = K^2.$$

However, there is a complication. Setting $g_i = 1$ for all i implies,

$$d_0 = 2e^{\frac{hK}{2}} + e^{-hK} - 3.$$

This is problematic, as it implies that for some values of h and K, d_0 may be negative.

As a general rule, any discrete weighting functions that satisfy (4.19) will require that either d_0 or g_0 be defined as a special case. This is because in all but the $i = 0$ case, esig$(pi - 1)$ can be simplified to $1 - e^{ih-\frac{h}{2}}$. However, in practice, treating g_0 as a special case is generally preferable to taking d_0 as a special case. For example, in the $k = K$ case it is preferable to define $d_i := 2(-1 + \cosh(hK))$ for all i, and $g_i = 1$ for all $i \geq 1$, which implies,

$$g_0 = \frac{1}{2}(1 + e^{\frac{hK}{2}}).$$

To extend this definition to the more general case of $k <= K$, begin by setting,

$$d_i = 2(\cosh(hK) - 1). \tag{4.20}$$

Now observe that together, equations (4.19) and (4.20) imply that for $i \geq 1$,

$$2(\cosh(hK) - 1)e^{-hk(i+\frac{1}{2})} = g_i(e^{-hk(i-\frac{1}{2})} - e^{-hk(i+\frac{1}{2})}) + g_{i+1}(e^{-hk(i+\frac{3}{2})} - e^{-hk(i+\frac{1}{2})}).$$

This is a recurrence relation in g_i. It is satisfied by any g_i of the form,[3]

$$g_i = \frac{\cosh(hK) - 1}{\cosh(hk) - 1} - \epsilon e^{ihk}\frac{\cosh(hK) - \cosh(hk)}{\cosh(hk) - 1}. \tag{4.21}$$

[3]Finding solutions for these recurrence problems requires either a real passion for concrete mathematics, or a passing familiarity with Mathematica's `RSolve[]` function.

In the solution given above, the free parameter ϵ has been chosen so that $g_i = 1$ given i for which $1 - \text{esig}(ihk) = \epsilon$, mimicking the constraint in equation (4.6). Like the definition for $w_1(x)$ given in (4.7), equation (4.21) has the form $b - ae^{kx}$, and is thus negative for large i. However, by choosing a sufficiently small ϵ, and clamping g_i to be at least 1, the negative values can be avoided while causing minimal reconstruction errors. See Section 4.4.1 for further discussion of the ϵ parameter.

Note that while equation (4.21) was derived by considering the discrete edge reconstruction problem, g_i is smooth for all $i \in \mathbb{R}^+$. It can thus be converted into a continuous definition for $w_1(x)$, by replacing ih with x. Converting back to a continuous weighting definition is important, as when the definition is generalized to 2D, the distance between any grid position and the nearest edge point will not always be integer valued.

Given g_1 defined as per equation (4.21), we can now solve for g_0,

$$g_0 = \frac{\epsilon(\cosh(hk) - \cosh(hK)) + \cosh(hK) - 1}{e^{-hk} + 2\sinh(\frac{hk}{2}) - 1}. \tag{4.22}$$

Note that this g_0 definition is strictly positive, given $K \geq k > 0$, $h > 0$, and $\epsilon > 0$ and sufficiently small.

4.3.3 Corrected Definition for the Sharp Edge Case

In order to model an edge that is sharper than the average case, the ratio $\frac{d_i}{g_i}$ must become larger than K^2 as we approach the edge. Large data weights d_i tend to cause problems in the multiple edge case. When a region is bordered by both both a sharp edge and a soft edge, the large data weights implied by the sharp edge can lead to visible discontinuities in the region's interior. Such discontinuities may be avoided by forcing the ratio of the edge weights g_i and data weights d_i to remain constant at any point. Specifically, by enforcing the condition,

$$d_i = 2(\cosh(hK) - 1)\frac{g_i + g_{i+1}}{2}. \tag{4.23}$$

This ensures that the ratio $\frac{2d_i}{g_i + g_{i+1}}$ will reach its maximum in the average sharpness $k = K$ case, and remain there for all $k > K$.

Once again, equations (4.23) and (4.19) combine to imply a recurrence relation in g_i, and the recurrence has a closed form solution when considered in the case of $i \geq 1$.

$$(\cosh(hK) - 1)e^{-hk(i+\frac{1}{2})}(g_i + g_{i+1}) = g_i(e^{-hk(i-\frac{1}{2})} - e^{-hk(i+\frac{1}{2})}) + g_{i+1}(e^{-hk(i+\frac{3}{2})} - e^{-hk(i+\frac{1}{2})}).$$

In this case, the recurrence is satisfied by any g_i of the form

$$g_i = \left(\frac{e^{2hk} - 1}{e^{hk}\cosh(hK) - 1} - 1 \right)^{i-n}. \tag{4.24}$$

Like the sharp edge weight derived in (4.8), this implies that $w_1(x)$ will have the form a^x, and thus increase towards infinity for large x. Here the free parameter n has been chosen such that $g_n = 1$. As in the continuous case, I clamp the values of g_i to be at most 1. The point at which clamping will be applied is determined by ϵ, using the equation,

$$n = \lceil \left(\frac{-\ln(\epsilon)}{hk + \frac{1}{2}} \right) \rceil.$$

Once again, this n definition is a discrete form of the constraints given in equation (4.6). However, the use of the ceiling operation implies that the clamping point will always be both integer valued and greater than or equal to 1. In the case of soft edges, the point at which g_i is clamped to 1 will likely occur far from the edge, and thus allowing the clamping point to occur between discretization samples is not a practical problem. However, in the case of a hard edge, the clamping may occur very close to the edge itself. In fact, for very large k, the system will typically clamp at $n = 1$.

By combining equations (4.23), (4.24), and (4.19), it can be shown that for all $i \geq 1$,

$$d_i = g_i \frac{4\sinh(hk)\sinh(\frac{hK}{2})^2}{e^{hk} - \cosh(hK)}. \tag{4.25}$$

By requiring that equation (4.25) also hold for $i = 0$, I can now derive the special case of the boundary crossing edge weight,

$$g_0 = \frac{1}{2}\left(1 + e^{\frac{hk}{2}}\right)\left(\frac{e^{2hk} - 1}{e^{hk}\cosh(hK) - 1} \right)^{-n}. \tag{4.26}$$

Note that if $n = 1$, g_0 will converge to 0 as k goes to infinity. This forms the discrete analogue to the convergent definition of w_1 given in equation (4.9). In the discrete context it is easier to see how the limiting case can be applied in practice. As illustrated in Figure 4.2, eliminating the weight variable g_0 causes the implied image to exactly match the piecewise constant target image.

4.3.4 Extension to 2D

The generalization of the weight functions to a two dimensional regularization problem, as defined by equation (4.10), assumes that image data can be modeled as a

continuous function. In order to define a similar discrete generalization to 2D, I start by applying the graph cutting algorithm given in Section 3.3.

Consider the following graph cutting result.

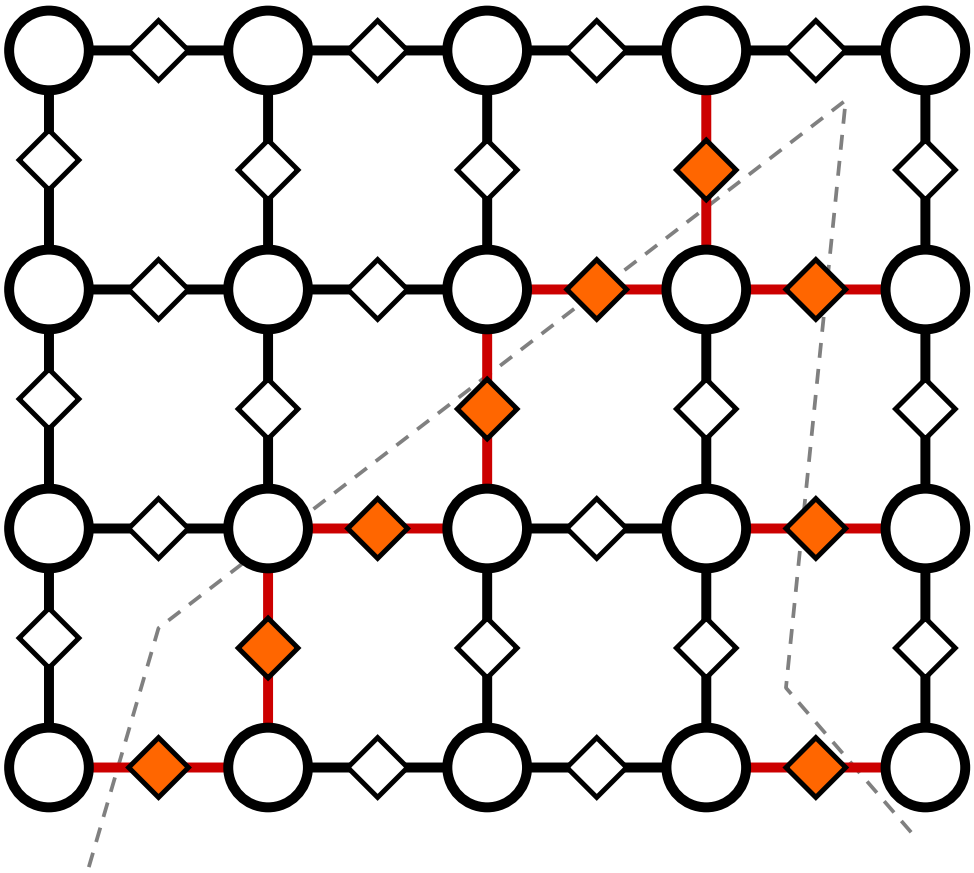


Figure 4.3: Edge Cutting Result

The edge sharpness values are stored in a linear spline $k(t)$, and thus a k value is defined at every point on the region boundary line. The edge cutting algorithm defined in Section 3.3 will sample from $k(t)$ in order to set a k value at each of the edge center points, which are marked in orange.

Generalizing the regularization problem defined in Section 4.3.1 to the two dimensional case requires defining a k value and distance x for both every graph edge and every pixel center. More specifically, if the set of pixel indices is V, and the set of edges in the 4-connected pixel adjacency graph is E, then defining the regularization requires defining the values k_i and x_i, for all $i \in V$, and k_{ij} and x_{ij}, for all $(i, j) \in E$.

Given such values, the one dimensional regularization given in equation (4.17) has

the following natural 2D generalization,

$$\min_u \sum_{i \in V} d(k_i, x_i)(v_i - u_i)^2 + \sum_{(i,j) \in E} g(k_{ij}, x_{ij})(u_i - u_j)^2. \tag{4.27}$$

As in the 1D case, equation (4.27) is a linear least squares problem, and can thus be converted into a matrix equation of the form $A\mathbf{u} = \mathbf{b}$. Notice that it is necessary to reinterpret the definitions of the weighting functions g and d as functions from $\mathbb{R}^+ \times \mathbb{R}^+$ to $\mathbb{R}^+$, which take as input both a sharpness value k and a distance value x. Also notice that the special case definitions for g_0, given in equations (4.21) and (4.24), will be used at every x_{ij} for which (i, j) is a cut edge, as those are exactly the cases in which $x_{ij} = 0$.

In image space, the positions associated with the pixels i and edges (i, j) lie on three overlapping grids. The first grid corresponds to pixel positions, the second to horizontal edges, and the third to vertical edges. In Figures 4.3 and 4.4, pixel centers are marked with circles, while vertical and horizontal edge positions are marked with diamonds.

Distance values in each of of the three grids are initialized as follows. At each cut edge the distance is set to $x = 0$. Any pixel center adjacent to a cut edge is set to $x = \frac{1}{2}$. Finally, for any vertical edge which is not cut, but which shares a vertex with a horizontal cut edge, $x = \frac{\sqrt{2}}{2}$. Similarly for horizontal edges. Thus, after initialization, the x values from Figure 4.3 are defined as follows.

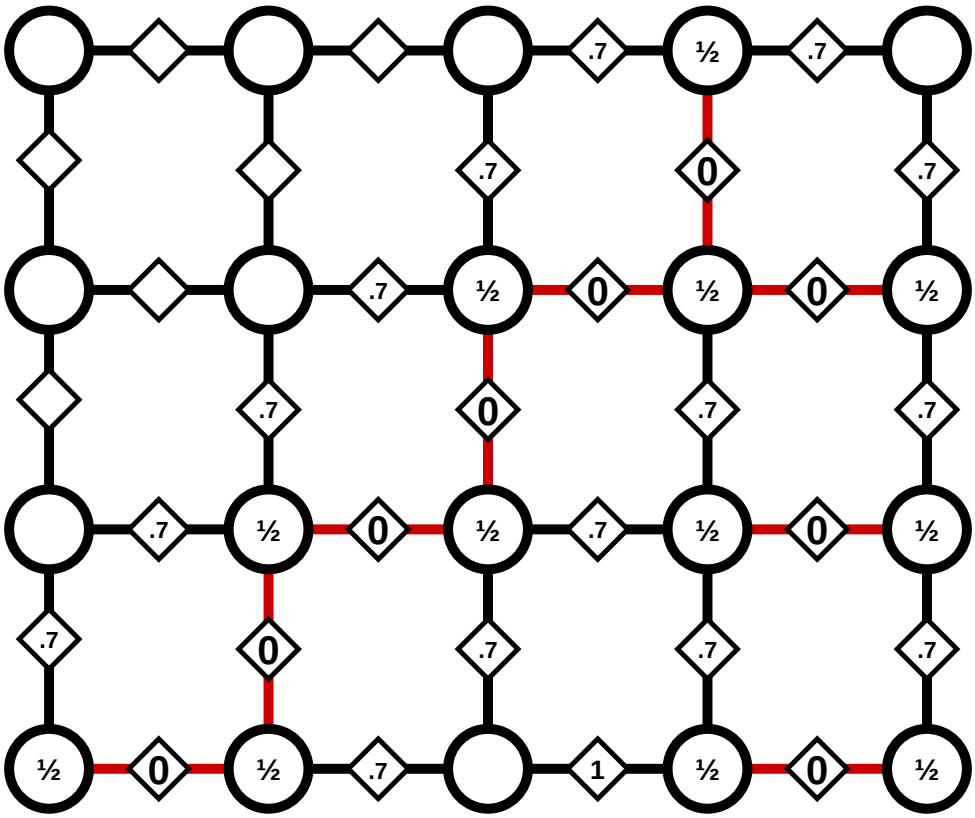


Figure 4.4: Distance Value Initialization

Each grid now has sufficient initial data to apply the nearest point algorithm given in Section 3.4.2. Along with defining an x value at all grid points, that algorithm associates each grid point with the set of cut edges that are closest to it. As demonstrated in Figure 4.5, the Voronoi diagrams associated with the set of cut edges will frequently have corners on grid points, and thus it is often the case that multiple cut edges are equally close to a given grid point. The k value at any grid point is defined to be the average of the k values defined at the closest cut edges.

It is possible to calculate k and x values at all grid locations by searching for the closest points included in the set of boundary line segments. However, this is a difficult operation to perform efficiently, in contrast to the nearest point algorithm, which can typically be performed in linear time. Furthermore, setting the value of x using the closest boundary line point would cause most cut edges to have x values slightly larger than 0, thus destroying the connection between the g_0 special case and the set of cut edges. That connection is critical in modeling sharp edges correctly. Therefore, using the nearest point algorithm to propagate x and k values is not only faster than performing a similar calculation directly from the boundary line data, it also makes

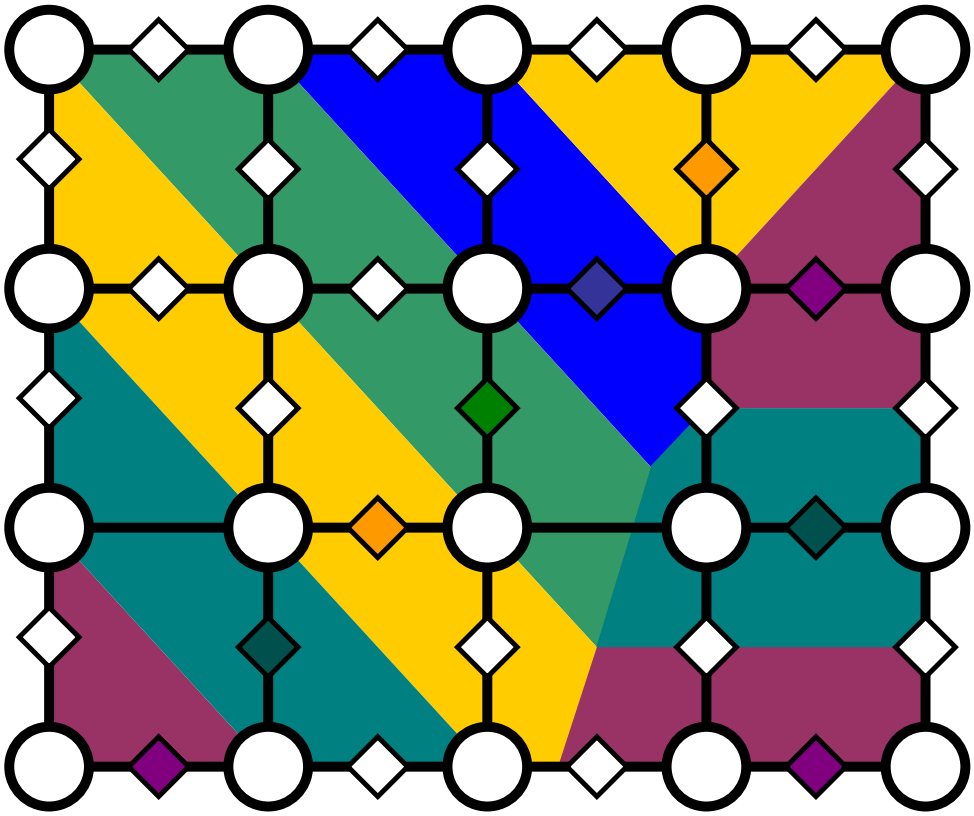


Figure 4.5: The cut edge Voronoi diagram associated with Figures 4.4 and 4.3. The cut edge locations are shown as colored diamonds, while the implied Vornoi regions have been drawn using a lighter shade of the same color.

it possible to take better advantage of the discretization error corrections derived in Sections 4.3.2 and 4.3.3.

After the completion of the nearest point algorithm, the linear system implied by equation (4.27) is solved, using the method of successive over-relaxation [63]. The result is the reconstructed image.

4.4 Results

4.4.1 One Dimensional Reconstruction Results

The arguments used to justify the anisotropic weight functions make several claims that require empirical validation. The first is that the reconstruction results will be accurate for the case of single edge images, despite the fact that the weight function derivation relies on the assumption that all edges will have infinite width. Similarly, the derived weighting equations are truncated, using the parameter ϵ, and the claim is made that this truncation does not lead to noticeable reconstruction errors, if $\epsilon \leq 0.005$.

Figures 4.9 and 4.11 demonstrate that, in the one dimensional case, very high quality reconstructions are achieved, even on relatively narrow, finite width images. Figures 4.8 and 4.10 show the relatively poor reconstruction created using a naive finite difference implementation of the continuous weight function definitions, thus justifying the additional complexity of the corrected discrete definitions derived in Section 4.3. Figure 4.7 plots reconstruction error over a range of discretization densities, and shows that, as expected, the corrected versions lead to better results at coarse discretization scales. However, as demonstrated in Figure 4.6, the corrected and uncorrected weight function definitions converge as h approaches 0, and thus the error plots shown in Figure 4.7 also converge for small h.

The quality of the edge reconstructions will be influenced the sharpness value that defines the edge model curve k, the parameters K and ϵ used in the the continuous weight function definitions, as well as the width m of the image and the distance h between discretization samples. The high dimensionality of this parameter space makes plotting the tradeoffs between all five terms impractical; however, creating a linear plot of any of these parameters, while the others are held fixed, is straightforward. For example, Figure 4.12 shows error as a function of edge sharpness. This error function shows a saw-tooth pattern, which likely results from the floor function used in the discrete weight function formulas. The same figure also includes a second plot made using a lows m value, which suggests that decreasing the width of the image will typically cause the reconstruction accuracy to become uniformly worse—an intuitive

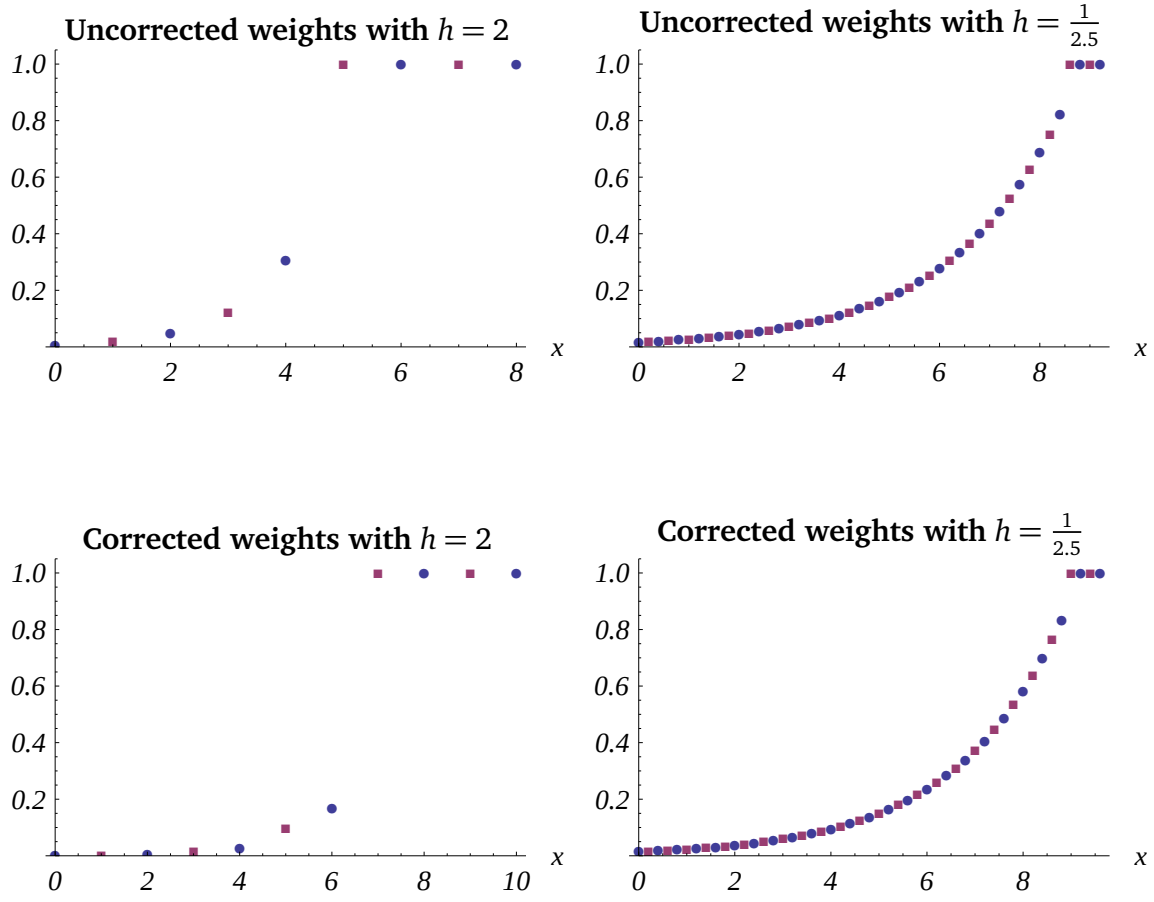


Figure 4.6: Comparison between the weight terms generated for a sharp edge case, with and without corrected definitions derived in Section 4.3. The edge weights g_i are plotted as blue circles, while data weights d_i are plotted as purple squares. In both cases, the parameters used are $k = 1$, $K = .3$, $h = 2$, and $\epsilon = .005$. The top two plots use w_1 and w_0 defined as per equation (4.8) then converted to g_i and d_i by applying equation (4.16) and multiplying both weights by h^2. The bottom plots show $\mathbf{g}$ and $\mathbf{d}$ defined using the formulas given in Section 4.3.3. In order to clarify the relation between the weight functions, all the weight function terms have been divided by their maximum values. As shown in Figures 4.8 and 4.9, in the case of large h, differences in the weight function definitions lead to significant improvements in the accuracy of the regularization results.

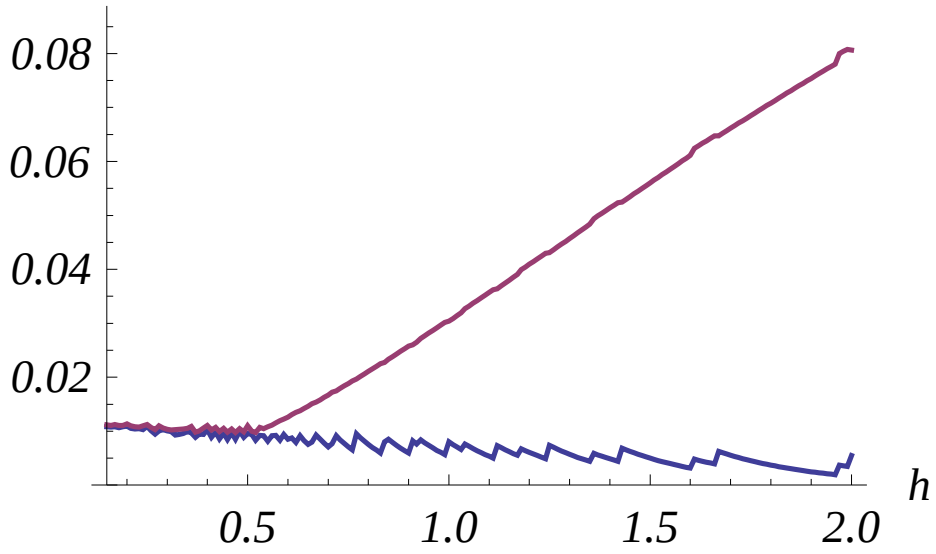


Figure 4.7: Maximum error as a Function of h. The blue line shows error when using the corrected edge weight definitions, while the purple line shows error using the uncorrected equations. The number of discretization points is set to $\frac{20}{h}$, thus modeling a single edge image of fixed width, sampled at a higher or lower resolution. Otherwise, the parameters are fixed at $k = .6$, $K = .3$, and $\epsilon = .005$. The corrected weights show significant improvement over the uncorrected weights for coarse discretizations, but, as h becomes small, the corrected weights converge towards the continuous definitions, and, thus, the error functions also converge. Note that the error does not approach 0. For any real image, some base reconstruction error is always expected, given the infinite width is condition assumed during the weight function derivations.

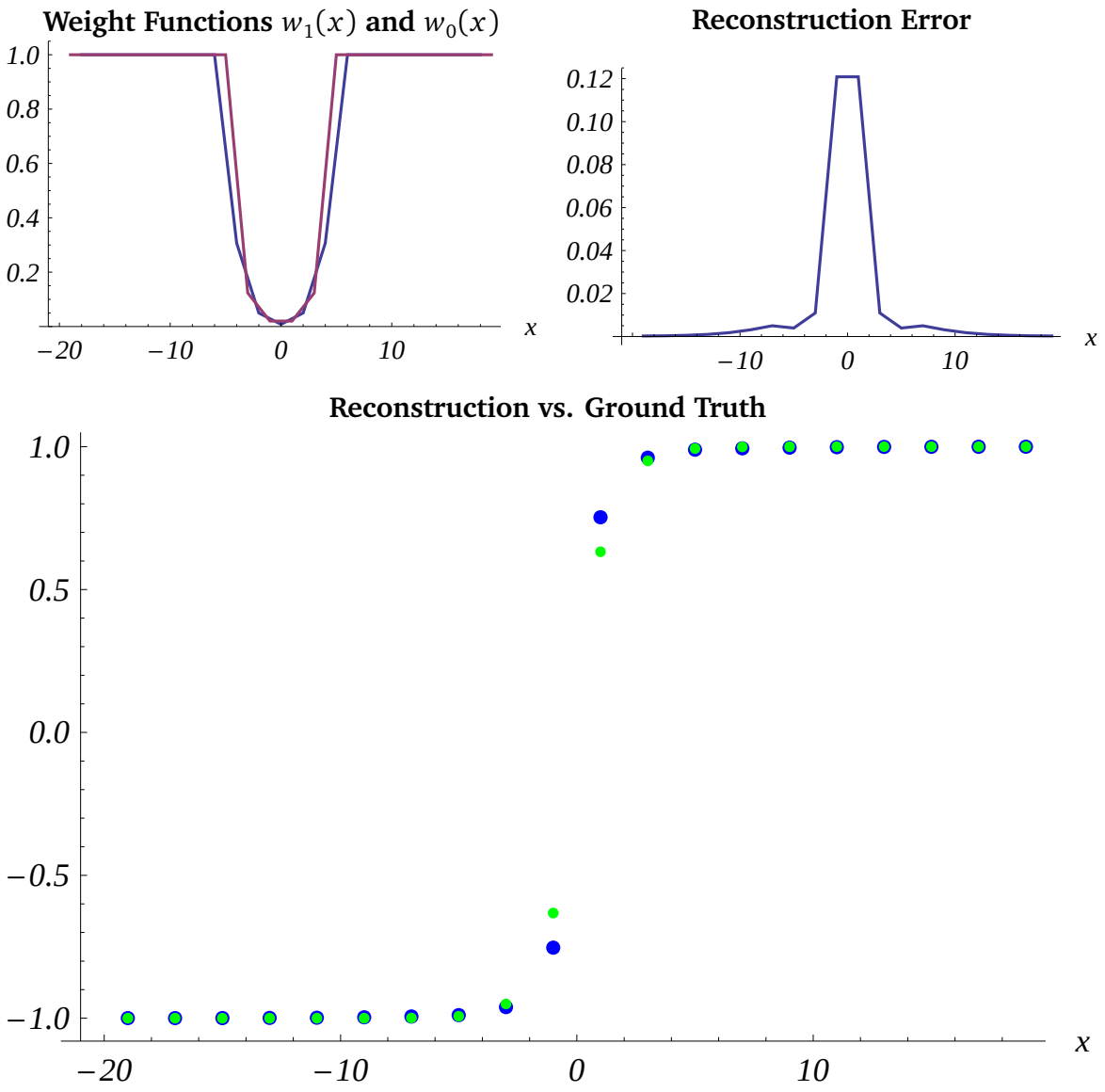


Figure 4.8: Discrete Reconstruction (Uncorrected). Sharp edge case, $k = 1$, $K = .3$, $h = 2$, $\epsilon = .005$, and $m = 20$. The ground truth, defined by esig(kx), is shown in blue, while the regularization result $\mathbf{u}$ is shown in green.

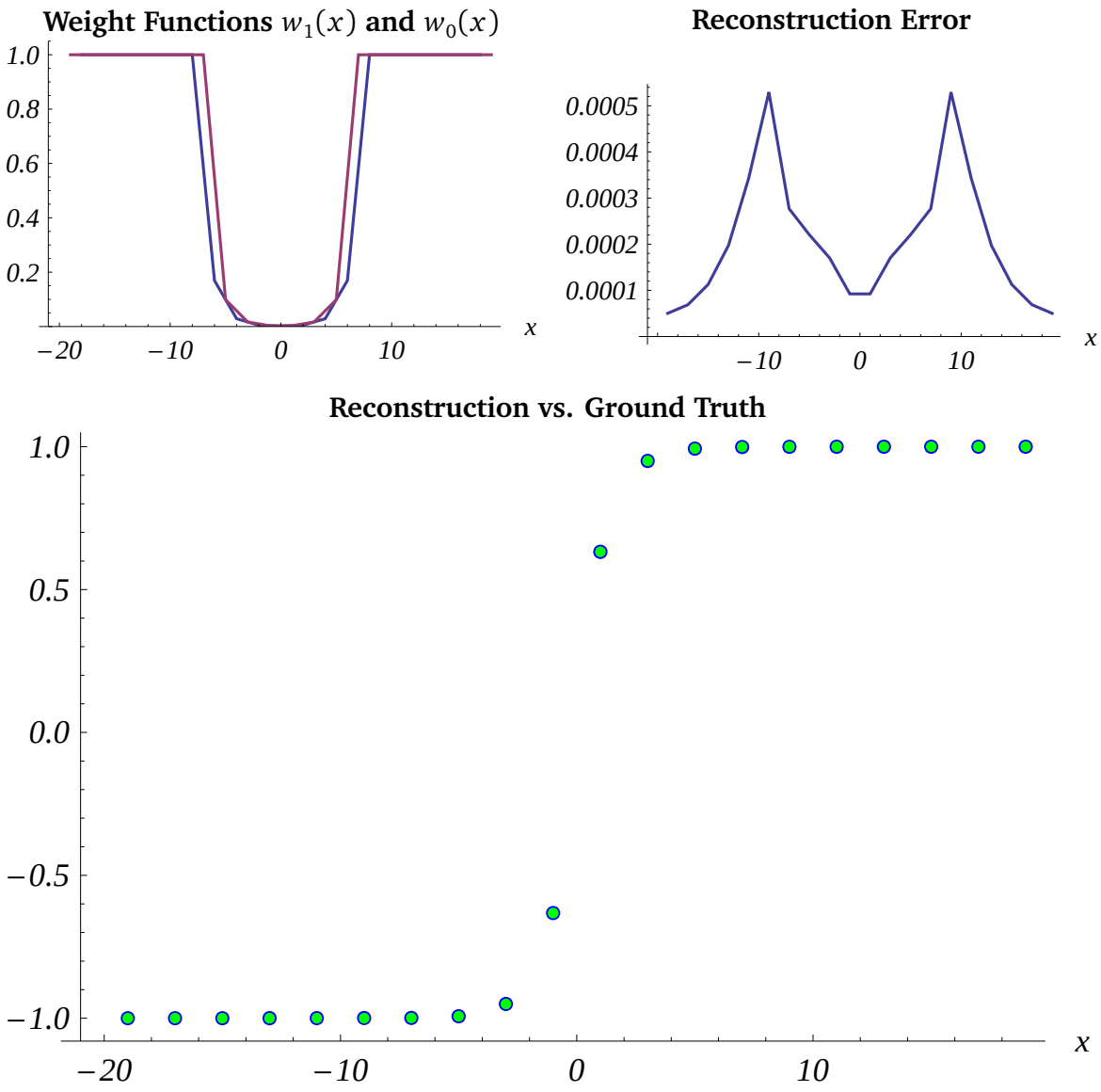


Figure 4.9: Discrete Reconstruction (Corrected). Sharp edge case, $k = 1$, $K = .3$, $h = 2$, $\epsilon = .005$, $m = 20$. The ground truth, defined by esig(kx), is shown in blue, while the regularization result $\mathbf{u}$ is shown in green.

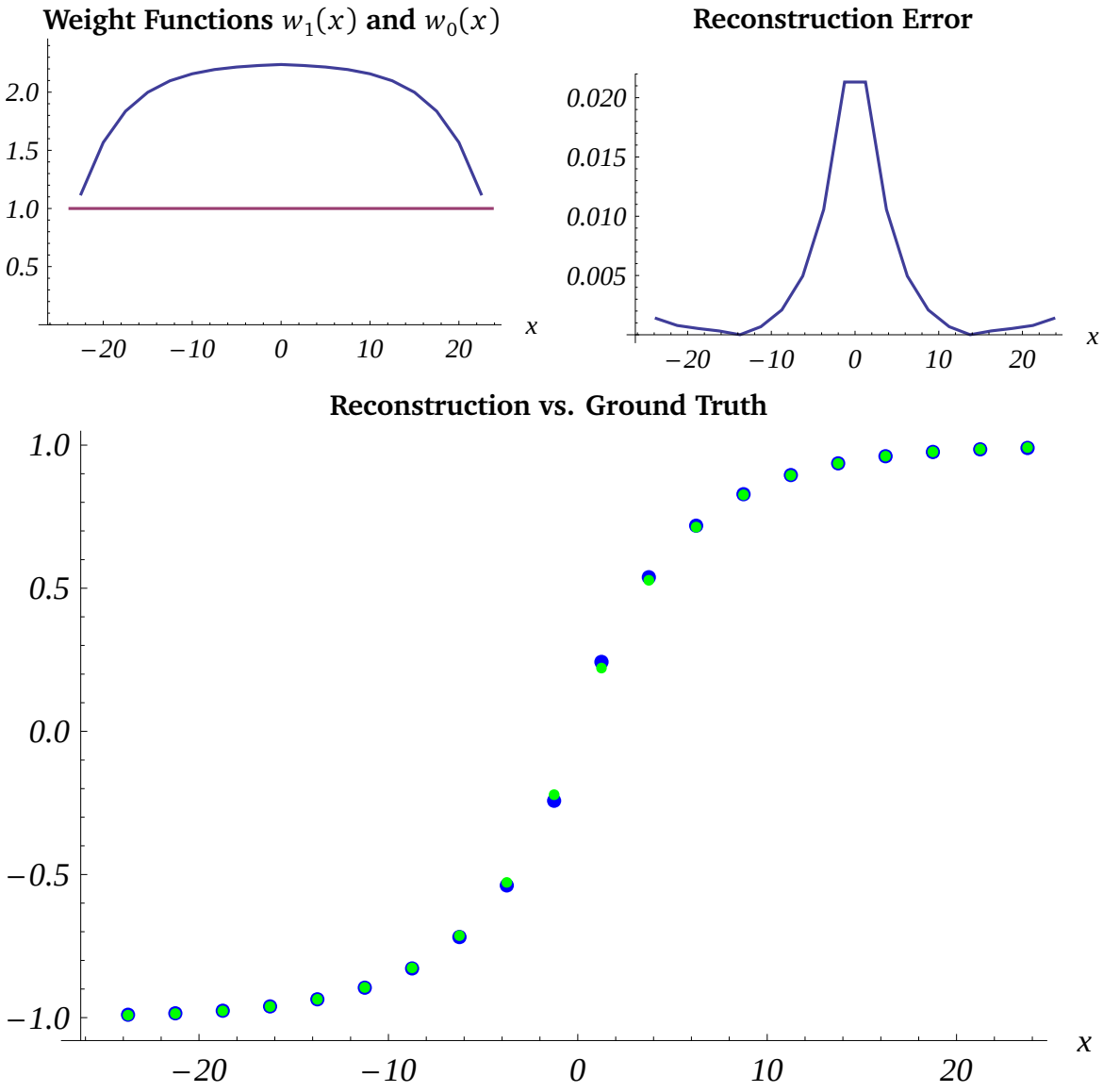


Figure 4.10: Discrete Reconstruction (Uncorrected). Soft edge case, $k = 0.1$, $K = .3$, $h = 2.5$, $\epsilon = .01$, $m = 20$. The ground truth, defined by esig(kx), is shown in blue, while the regularization result $\mathbf{u}$ is shown in green. In the weight function plot, the values d_i have been scaled by dividing through by the minimum value of d_i (unlike in the prior plots, in which d_i is scaled according to its maximum value).

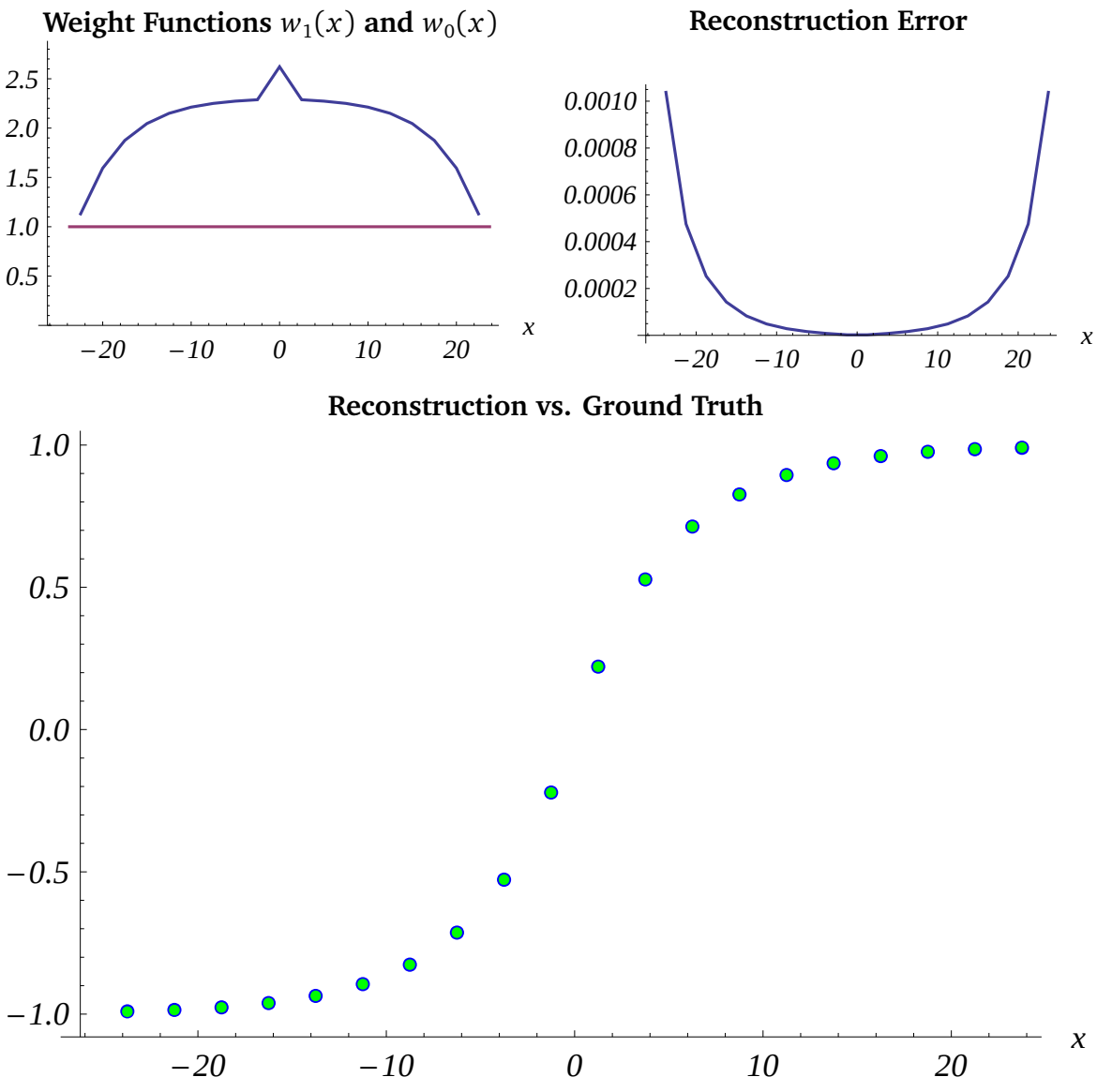


Figure 4.11: Discrete Reconstruction (Corrected). Soft edge case, $k = 0.1$, $K = .3$, $h = 2.5$, $\epsilon = .01$, $m = 20$. The ground truth, defined by esig(kx), is shown in blue, while the regularization result $\mathbf{u}$ is shown in green. Observe that most noticeable difference between the weight functions in the corrected and uncorrected cases is the small increase in the edge-crossing smoothness weight g_0. This minor change, however, leads to more than an order of magnitude improvement in the accuracy of the regularization result.

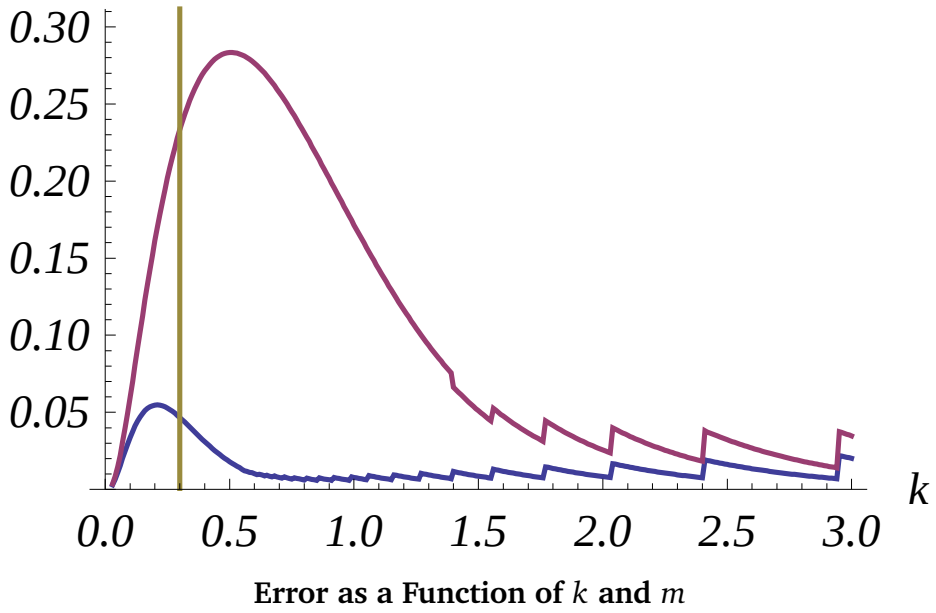


Figure 4.12: Maximum error as a Function of k. The blue line shows error with $m = 50$, the purple line shows error with $m = 20$. The yellow line marks $k = K$. Otherwise, the parameters used are fixed at $h = \frac{1}{2.5}$, $\epsilon = .005$.

result.

One less intuitive result discovered while investigating the relationships between the reconstruction parameters is shown in Figure 4.13. If the sharp edge definitions are applied in the case of soft edges, and vice versa, the reconstruction error in the single edge case reliably decreases. When k is outside of its intended domain, the weight function definitions tend to result in extremely large terms (or in the case of the soft edge functions, negative weighting terms with large absolute value). However, while these large values do make the weights unsuitable for generalization to the two dimensional case, they appear to lead to improved reconstruction accuracy in the one dimensional case.

4.4.2 Two-Dimensional Reconstruction Results

Quantifying the success or failure of edge reconstructions in the one dimensional case is straightforward, as the model curve provides a clear standard for correct behavior. However, when the regularization is generalized to the two dimensional, multiple edge case, there is no longer a clear ground truth.

Figure 4.17 shows a visualization of the intermediate data used when performing a regularization based on traced vector data. Notice that the weight function images

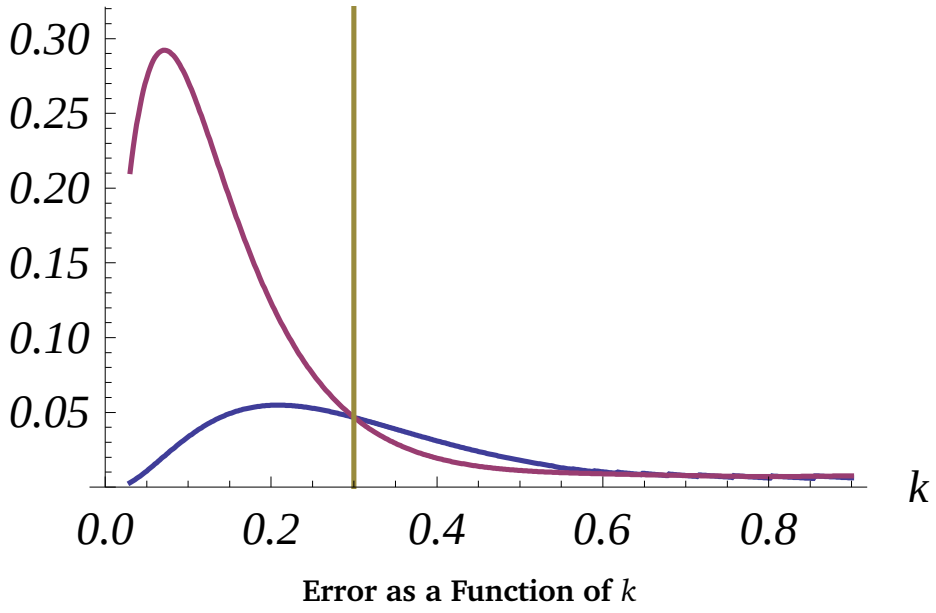


Figure 4.13: Maximum error as a Function of k. The blue line shows error when using the hard edge weight definitions, while the purple line shows error with using the soft edge definitions. The yellow line marks $k = K$. Otherwise, the parameters used are fixed at $m = 50$, $h = \frac{1}{2.5}$, $\epsilon = .005$. Notice that the errors are exactly the opposite of what would be expected. The soft edge formulas have lower errors when applied to the hard edge cases, and vice versa.

there are many strange shapes and sharp creases inside the regions bounded by the curve data. However, despite this, the only creases or edges apparent in the regularization result are those that occur at region boundaries.

Significance of the K Value

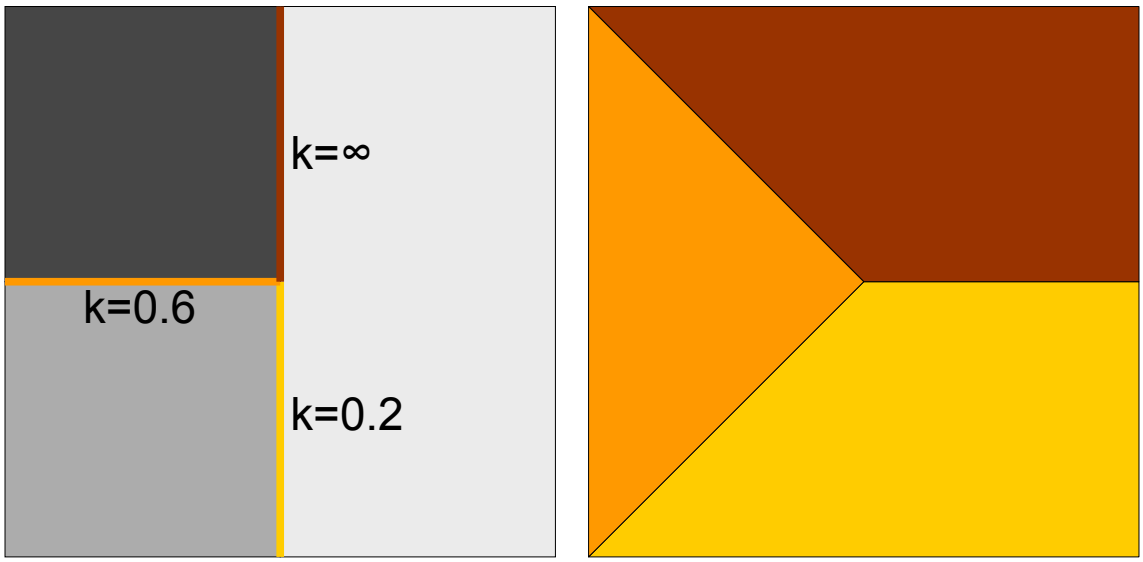
To better understand the tradeoffs implied by the K parameter, I have generated a series of results using a simple three edge test case, described in Figure 4.14. As demonstrated in Figure 4.15, smaller K imply a higher degree of smoothness across the "seams" between different Voronoi regions. Thus a low K is desirable. However, there are several negative impacts of choosing a very small K. One is that small K lead to wide, exponentially increasing weighting values near edges. Thus the linear solver requires more iterations to reach convergence. The other drawback of using very small K is that all pixels on the boundary of a soft edge's Voronoi region will often snap to the luminance value in the neighboring region. At high resolutions this is rarely leads to noticeable artifacts, but at lower resolutions it leads to a clear nonlinearity over what should be a smooth edge transition, as shown in Figure 4.16.

4.4.3 Comparison to Variable Width Blurs

The edge-only image representations used by Orzan et al. [52] and Elder are, in several important ways, very different from the representation used in my own system [20]. The largest of these differences is that I record only a single luminance value per region, while the other two systems store two luminance samples at each edge point. Thus, those systems require an initial reconstruction of a piecewise smooth image, which is achieved by treating the color samples at each edge as fixed points, and then solving Laplace's equation to find a smooth interpolating function for those values. This is quite different than the luminance reconstruction used by my own system, which generates an initial piecewise constant image using the connected components algorithms described in Section 3.4.1.

However, as a second pass, both Elder and Orzan et al.'s systems generate a set of blur widths defined at every point of the image. These widths are then used to guide a varying width blur. And in the case of a single edge image, the result of blurring will be that the transition between regions is perfectly modeled by the error function.

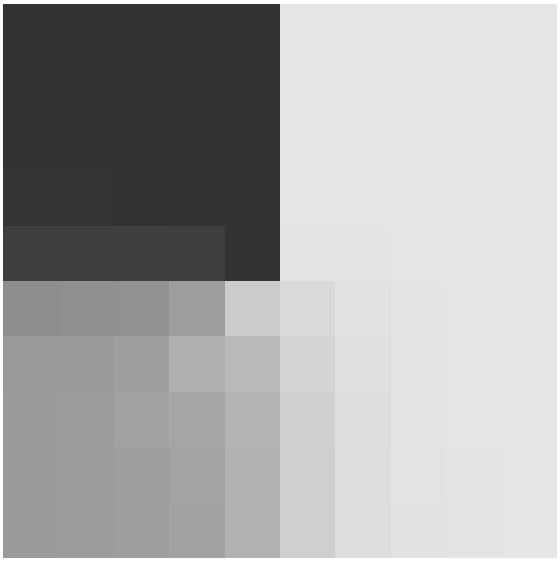
As explained in Section 4.6, the exponential sigmoid and the error function are similar curves, and it is possible to convert my sharpness samples k to σ values associated with an error function. Thus, Elder's blurring method can be applied in place of the regularization methods given in this chapter.



Boundary Lines and Target Image **Boundary Line Voronoi Diagram**

Figure 4.14: Specification for a 3 edge test case. This case is designed to study the impacts of the two dimensional generalization of the regularization problem. The image contains one smooth edge represented by the yellow $k = .2$ boundary, and one sharper edge, the orange $k = .6$ boundary. The three quantization bins represented in this image are $\mathbf{b} = (.2, .61, .95)$. Boundaries between regions with non-adjacent target values always correspond to the "infinitely sharp" special case, and such a boundary occurs along the red line. Artifacts are most likely to occur when the weight function definition switches from one k value to another; such transitions occur at the boundaries of the Voronoi diagram, shown left.

Reconstruction with $K = .03$



Reconstruction with $K = .3$

Reconstruction with $K = .03$

Reconstruction with $K = .3$

Figure 4.15: Artifacts caused by the choice of K.

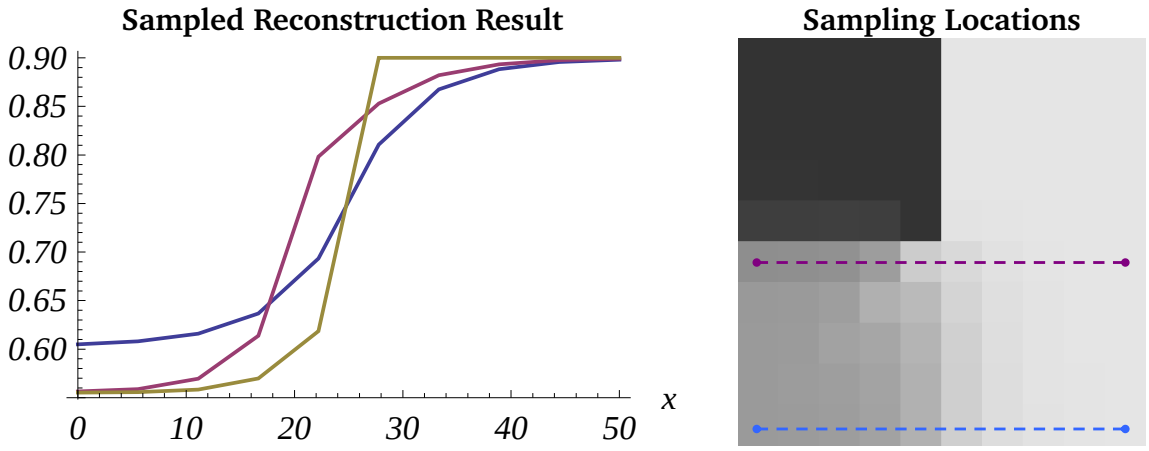
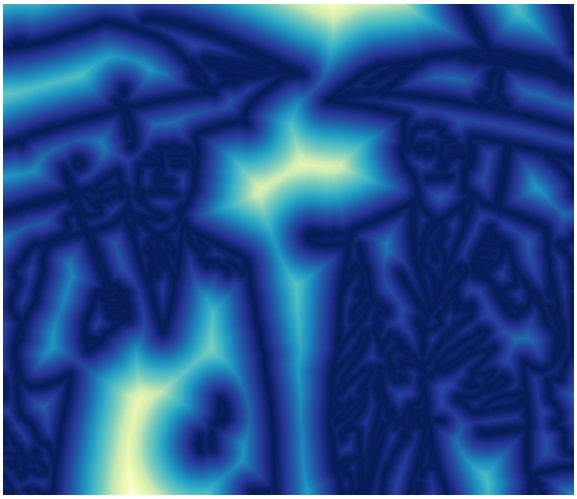


Figure 4.16: One dimensional samples from the three edge test case. The blue and purple lines are sampled from the low resolution $K = .3$ reconstruction result, as shown. The yellow line is taken from the $K = .03$ case, and uses the same sample locations as the purple line. Note the sharp discontinuity introduced when crossing the region boundary, in the $K = .03$ case.

In order to define blur widths at each point in the image, Elder solves a Laplace equation using the sampled blur widths as boundary conditions. Solving the Laplace equation requires a solving a sparse linear system much like the one used by my own anisotropic regularization method. Thus, using blurring for edge reconstruction is significantly slower than using anisotropic regularization, as it requires an expensive varying width blur operation, in addition to the already expensive linear solution required by either method.

However, as shown in Figures 4.18 and 4.19, using blurring to model soft transitions leads to a less accurate reconstruction of the shape and lighting information encoded in the sharpness data. Solving a Laplace equation in order to define a blur width at every point in the image does ensure that the blur result is free of crease or seam artifacts. However, it also implies that the smoothness of the result at any point will be influenced by the values stored at many neighboring edge points, not just the closest edge. I hypothesize that it this "blurring of the blur data" that causes Elder's approach to edge reconstruction to lead to relatively poor results, when applied in the context of my own system.



Distance Field

Sharpness Values

Data Weights

Edge Weights

Boundary Lines

Regularization Result

Figure 4.17: Data generated during the reconstruction process.

Reconstruction using Blurring

Reconstruction using Regularization

Blur Widths

Source Stylization

Figure 4.18: Comparison of varying width blurring and anisotropic regularization. The blur widths generated by solving Laplace's equation for each connected component are shown in the green image. Brighter pixels correspond to larger blur widths. The source stylization is the soft quantized image generated as per chapter 2.

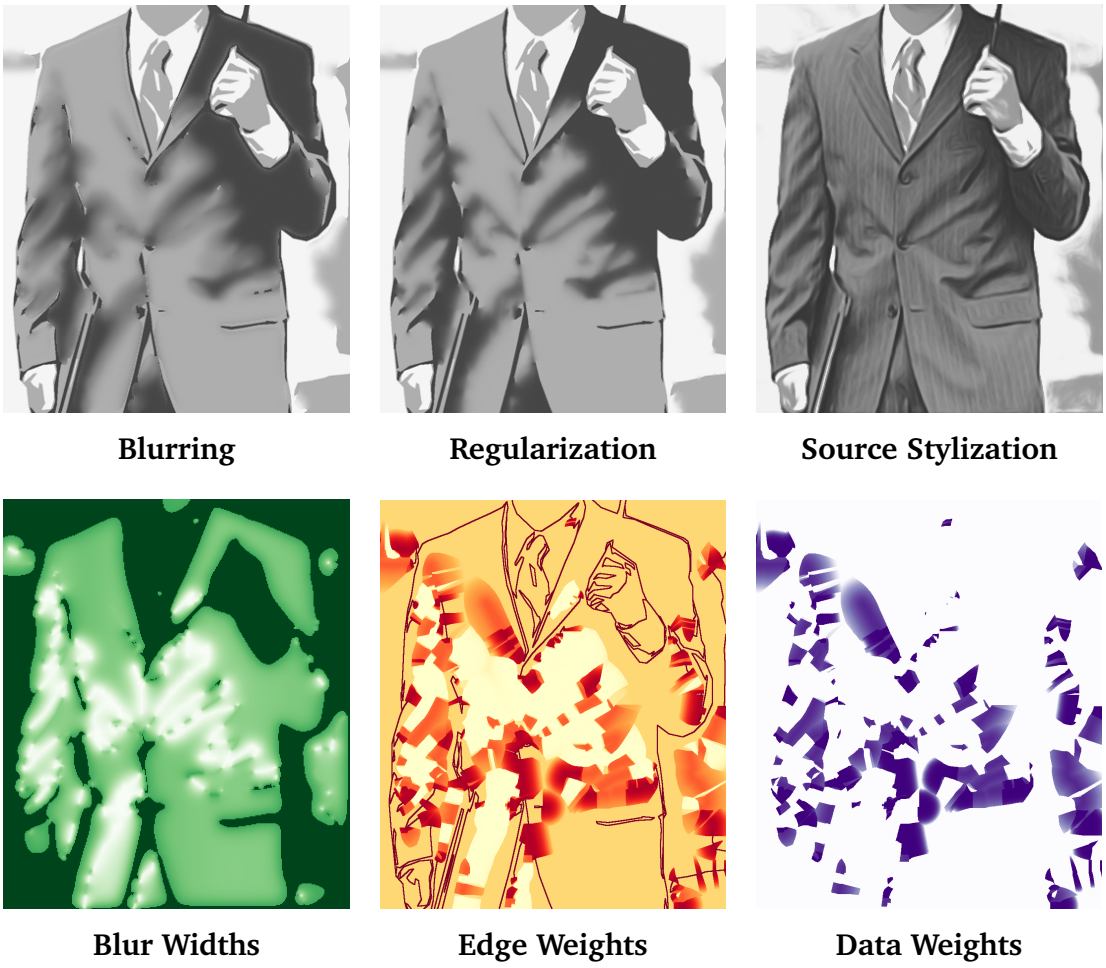


Figure 4.19: Reconstruction Comparison (detail). Notice that while the blur width data is smooth inside each connected component, the anisotropic weight functions show relatively complex features, which correspond to the Voronoi tessellations of the edge set, as well as the nonlinear weight function definitions. This increased complexity appears to lead to a more accurate reconstruction of the shading present in the source stylization.

4.5 Post Smoothing

The image reconstructions generated by the regularization process tend to accurately reproduce any smooth shading in the source stylized image. However, hard edges often suffer from "jaggies", aliasing artifacts that result from the conversion of the boundary data to a discrete image grid. One simple way to eliminate such artifacts is to reconstruct images at a much higher resolution than they will be displayed, and then downsample the results, taking advantage of the resolution independent nature of the data. However, I have found it both more computationally efficient and more effective to apply a slight smoothing effect to the result. Specifically, I apply coherence enhancing anisotropic diffusion, as defined by Weickert [75], using the coherence biasing parameter $\alpha = .001$, and scalespace parameter $t = 3.6$. As shown in Figure 4.20, in addition to removing aliasing artifacts, coherence enhancing diffusion biases the results towards images containing smooth, connected lines.

4.6 Sigmoid Function Conversion

The anisotropic regularization definitions developed in Section 4.2 assume that the soft quantization used to generate the source image is based on the exponential sigmoid curve. However, with a slight adjustment, it can be used to reconstruct soft quantized images generated using arbitrary sigmoid curves. Most soft quantization-based cartoon filters developed in the past have defined $\text{sig}(x) = \tanh(x)$, thus this sigmoid curve conversion step allows output from those systems to be used as source images.

Observe that the sigmoid curve $\tanh(x)$ is qualitatively very similar to $\text{esig}(1.4x)$. In fact, the maximum difference between those curves is less than 0.053. This suggests that a soft quantization $p(I,S)$ that uses $\text{sig}(x) = \tanh(x)$ would produce very similar results to the soft quantization $p(I,1.4S)$ with $\text{sig}(x) = \text{esig}(x)$.

Rather than experiment with candidate c values by hand, I have used numerical optimization to find,

$$\hat{c} := \min_c \int_{-\infty}^{\infty} (\text{sig}(x) - \text{esig}(cx))^2 dx. \tag{4.28}$$

In the case of $\text{sig}(x) = \tanh(x)$, this optimization yields $\hat{c} \approx 1.40236$. In the case of $\text{sig}(x) = \text{erf}(x)$, it yields $\hat{c} \approx 1.69361$.

This suggests a general recipe for creating anisotropic regularization terms corresponding to arbitrary $\text{sig}(x)$ definitions. First, solve equation (4.28) to find a multiplicative constant that makes the exponential sigmoid as similar as possible to the

Before Smoothing **After Smoothing**

Figure 4.20: Comparison of the raw regularization result, and the result after post smoothing with coherence enhancing anisotropic diffusion. In several places in this example, reconstructing the stored gradient information faithfully has led to undesirable smudging near the mouth and collar. Notice that, in addition to eliminating aliasing artifacts, coherence enhancing diffusion replaces those smudges with connected lines. The unsmoothed image shown here is a detail of the regularization result shown in Figure 4.17.

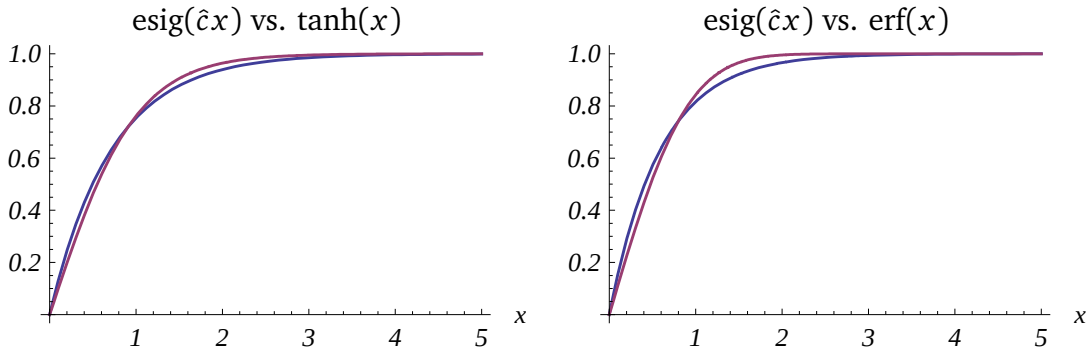


Figure 4.21: Comparison of sigmoid curves with their optimal approximation by an exponential sigmoid curve. The scaling factor $\hat{c}$ is defined separately for each graph, according to equation (4.28). Notice that while a good approximation is possible in the case of the hyperbolic tangent, the error function does not allow as close a fit.

sigmoid used in soft quantization. Scale the sharpness image by the resulting $\hat{c}$, and then proceed as in the case for reconstructing an esig-based soft quantization. While the scaling step does introduce some additional error into the reconstruction, this error will often be much smaller than errors introduced as a result of deviations of the source data from the piecewise linear ideal. As shown in Figure 4.21, tanh is similar enough to the approximating esig that converting from a tanh-based soft quantization to an esig-based representation should introduce negligible error. Conversion errors may be slightly more noticeable in the case of a soft quantization based on the error function, however.

5 Results and Examples

This example file is dedicated to Io, the Greek goddess of input and output.

– Donald Ervin Knuth

5.1 Computational Costs

The algorithms used for image space stylization, vector tracing, and image reconstruction are implemented in a mixture of Matlab and C. The dominant cost of image stylization and tracing is the calculation of the flow-guided blurs. These blurs are implemented in C, and run on a CPU. They can require more than a second of processing time per image. However, studies performed by Kyprianidis and Döllner demonstrate that flow-guided blurs can be performed at much greater speeds using GPU processing [39].

Given target resolutions in the megapixel range, image reconstruction can take several seconds, the overwhelming majority of which is spent performing the linear solve. As is the case with the stylization step, the costs of image reconstruction could likely be dramatically reduced by using an iterative linear solver implemented in CUDA or OpenCL.

5.2 Memory Efficiency

The goal of this work is to create simple, stylized images in which the key visual content of a source image has been clarified, rather than discarded. Given an input photograph, and the resulting vector image, the degree to which the results are successful in this goal is difficult to quantify.

For example, consider a system in which, for any input photograph, the "vector stylization" returned is always an arrangement of two black boxes on a white background. From a memory efficiency standpoint, it is clear that a dramatic simplification has been achieved. However, such a system could not reasonably claim to "preserve and clarify" the key visual content of the photograph. But, while quantifying the degree to which the stylizations are "good abstractions" is difficult, and, perhaps necessarily sub-

jective, identifying cases in which the vectorizations fail to improve memory efficiency is straightforward.

5.2.1 Simplification Relative to the Input Photograph

If the vector data, when stored to disk, has a higher encoding cost than a high fidelity compression of the input photograph, then the claim that the vector stylization is an effective *simplification* is suspect. For example, even if the vector image appears visually simpler than the source photograph, if the initial photograph can be encoded with a high degree of visual fidelity using 7kB, then any vector result that requires more than 7kB of storage space has actually made the source data more complex than it was in its original form.

The resolution independent nature of the data makes direct comparisons of encoding efficiency impossible. Vector data can be used to reconstruct smooth images at arbitrarily large sizes, thus storage efficiency per-pixel of output data is undefined. However, per-pixel efficiency relative to the input data is defined. Additionally, it is expected that vector encoding costs will increase as the complexity of the scene increases, thus it is reasonable to assume that even for a maximally efficient vector format, the resolution of the input photograph will be correlated with encoding costs.

For a collection of example vector stylizations, I have created compressed versions of the input photographs having similar file sizes. Cases in which the compressed images suffer from minimal visual distortions must be considered failure cases for the algorithm, in the sense that the generated vector data does not appear to represent a true simplification of the input photograph.

When the standard for simplification is set by JPEG image compression, all the vector images easily pass the data simplification test. In this case, is clear that encoding the initial photograph in a similar number of bits is not possible without introducing extreme visual artifacts.

Using JPEG2000 compression results in less extreme visual errors, but very noticeable artifacts still often result when images are compressed into file sizes that match those achieved by the vector stylization. See for example Figure 5.1. I also include some memory efficiency tests relative to the DLI research codec. As of 2009, DLI holds several records for minimum MSE compression of image data at very small file sizes [43].

Comparing the results of standard JPEG compression to either the DLI or JPEG2000 results should make it clear how dramatically image compression technology has improved over the last twenty years. In comparison to the highly sophisticated techniques

Source Image

JPEG2000 at 5.9kB

DLI at 5.3kB

Vector at 5.8kB

Figure 5.1: Memory Efficiency Comparisons. This is an example of a vector result that represents a successful simplification of the source photograph. For this test case, the DLI research codec significantly outperforms JPEG2000, preserving many more fine details, even when forced to encode the source photograph at less than 6kB. Compared to the compression results, the vector stylization is remarkable for having discarded nearly all background information, while better preserving important visual details, such as the locations of the eyes and noses. Also note the shading of the jacket worn by the student on the right; here the vector result does an excellent job of indicating the shape of the folded cloth, while discarding the high frequency texturing present in the original.

Source Image

JPEG at 20kB

JPEG2000 at 6kB

Vector at 6kB

Figure 5.2: Memory Efficiency Comparisons. This is an example of a vector result that fails to provide a useful simplification, as measured relative to a state of the art lossy compression codec. While JPEG compression is incapable of representing the source image accurately at 20kB or less, the more efficient wavelet encodings used by JPEG2000 are capable of representing the source quite faithfully, even when limited to the number of bits used by the vector stylization.

used by a system like DLI, the vector encoding method presented here is quite immature, and there is likely significant room for improvement. Even so, the encoding costs for simplified vector images are typically less than 7kB per image. And at such extremely small file sizes, even the DLI codec frequently results in noticeable artifacts; thus demonstrating that it is often reasonable to consider the vector results effective simplifications of the input photographs. For examples, see Figures 5.1, 5.3, and 5.4.

5.2.2 Simplification Relative to the Image Space Stylization

While the image space stylization step is designed to anticipate subsequent vectorization, the translation to a vector format will typically cause noticeable changes in the image content. Given my goals, such changes are not necessarily problematic. The aim of this research is to create effective abstractions of input photographs, and depending on the sorts of changes that exist between the image space stylization and the vector data, either one might be considered more successful in that aim. For example, the output of the Ardeco system is one example of a case in which the "artifacts" of vectorization can actually create attractive artistic effects [42]. The loss of texturing shown in Figure 4.19 acts to emphasize object shading, and is thus arguably a case in which the vector result is a more effective abstraction than the source stylization.

However, in many of the cases shown in this dissertation, the vector abstractions lose some important visual content relative to the source stylization. For example, the outline of an iris may disappear, or a small but important highlight may be eliminated. Figure 5.4 contains a good example of the iris simplification problem. In such cases, if conventional image compression can be used to store the *stylized image* at the same filesize as the vector data, we must conclude that the vectorization step of the pipeline is sacrificing visually important information unnecessarily.

Thus, I compare the vector renderings not only to lossy compressions of the source data, but, also, to lossy compressions of the image space stylizations from which the vector data is derived. Similar to the comparisons against compressions of the input photographs, the results often demonstrate that the vector encodings do occupy a level of memory efficiency at which the source stylization cannot be stored with good visual fidelity. For an example comparison, see Figure 5.3

5.3 Stylization Failure Cases

There are several cases in which the algorithms fail to create simple, curve-based representations of an initial image. If the stylization filter does not create a useful abstraction

Source DLI at 4.3kB **Stylization DLI at 4.3kB** **Vector at 4.3kB**

Figure 5.3: Memory Efficiency Comparisons. This figure compares the vector data to both a compressed version of the input photograph, and also to the compression of the image-space stylization from which the vector data is derived. Notice that there are fewer visual artifacts in the compressed stylization than in the compressed input photograph, which suggests that the DLI compression is able to take advantage of many of the visual simplifications achieved by the stylization step. However, the lines in the vector result are significantly sharper and smoother than those in the compressed stylization. (Image source: http://www.flickr.com/photos/bodoggirl.)

Source DLI at 2.6kB **Uncompressed Stylization** **Vector at 2.1kB**

Figure 5.4: In the case of this historical photograph, we begin to pass the limit at which state of the art compression is capable of storing the source image. DLI v1.3, set for maximum compression, creates a 2.6kB file. Meanwhile, the vector stylization of the same data requires 2.1kB of storage space, a 20% more efficient encoding. This test is also a case in which the vector stylization noticeably loses several visually important features that are present in the image-based stylization of the source. However, such losses of information are likely inevitable, given the extremely small file sizes being generated.

of the initial image, then the vectorization will also fail to be useful. Such failure cases often appear to be over-blurred or over simplified. Additionally, applying aggressive line simplification and region elimination settings can lead to significant shape distortions in the vectorized results. While errors of this last form can be fixed by using more conservative line simplification settings, doing so will increase curve complexity, leading to results which fail in the goal of creating data that is a simplification of the input, in the sense of memory efficiency.

6 Conclusions

Wow, look at how much we got done in the last three weeks!
We should have Siggraph every month!

– Jack Tumblin, January, 2005

The results of this research make it possible to revisit the fundamental question of how visual information can best be extracted from photographs. Edward's advice that artists must learn to draw spaces and shapes, rather than semantically loaded objects, may well reflect a deeper truth about the nature of images [19]. In the tradition of Leclerc [41], I would hypothesize that the most efficient possible representation of the core visual content of most natural images is as a vector tracing of patterns of shadows and highlights; augmented with a small amount of edge sharpness data. This vector efficiency hypothesis is sufficiently vague that it would be difficult to conclusively confirm or deny. Even so, given the results shown in Chapter 5, I suspect that there is a sense in which the hypothesis is true.

In a similar vein, it is interesting to note that the very low file size results created by cutting edge compression algorithms often contain artifacts not unlike the results of an artistic stylization. For example, compare the artifacts in the DLI compression shown in Figure 5.1 to the brush stroke stylizations created by Hertzmann [31]. The similarities suggest that, at extremely low bitrates, the most effective compression methods may be those that approach increasingly stylized images.

Developing a general purpose compression format in which images become increasingly stylized would certainly be a major project. However, it appears likely that creating such a format is possible, requiring relatively modest improvements over state of the art technologies in computer graphics, vision, and signal processing. For example, given further refinements of my tracing and encoding methods, the vector stylizations for single subject photographs could likely approach filesizes of 1kB or less, putting them comfortably outside the range at which visually accurate image compression is currently possible. And if such a compression format could be created, it would be a useful tool for web designers, as it would improve the visual quality and responsiveness of pages viewed under low bandwidth conditions.

The simplicity of the curve data also suggests that the vectorization system could prove useful as a tool for artists. For example, one of the drawbacks of diffusion curves, as discussed in Orzan et al., is that the number and complexity of the edges returned by the the tracing method makes the data difficult for artists to manipulate [52]. For this reason, most of the diffusion curve results shown in that paper are generated from scratch by an artist, even in cases where the vector image is based on a reference photograph. The curves traced by my own joint stylization/vectorization framework are simpler and thus likely easier to manipulate. As both my vector format and diffusion curves are variations on Elder's edge only format [20], converting the data to the diffusion curves format should be relatively straightforward.

Finally, observe that the image space stylizations described in Chapter 2 are an evolution of the stylization methods used by Kyprianidis [39], which have proven to yield good results on video. Thus, it is likely that the vectorization system could be extended to produce visually abstracted, memory efficient spacio-temporal tracings of video data. However, developing such an extension is nontrivial, given the need to maintain temporal coherence in any vector results.

Bibliography

Supplied by a sub-sub-librarian

It will be seen that this mere painstaking burrower and grubworm of a poor devil of a sub-sub appears to have gone through the long Vaticans and street-stalls of the earth, picking up whatever random allusions to whales he could anyways find in any book whatsoever.

– Herman Melville

[1] Tinku Acharya and Ping-Sing Tsai. *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. John Wiley & Sons, Hoboken, New Jersey, 2005.

[2] Aseem Agarwala, Aaron Hertzmann, David H. Salesin, and Steven M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Trans. Graph.*, 23(3):584–591, 2004.

[3] Richard C. Aster, Brian Borchers, and Clifford Thurber. *Parameter Estimation and Inverse Problems*. 2004.

[4] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial*. SIAM, second edition, 2000.

[5] Thomas Brox. *Von Pixeln zu Regionen: Partielle Differentialgleichungen in der Bildanalyse*. PhD thesis, Mathematische Bildverarbeitungsgruppe, Universität des Saarlandes, March 2005.

[6] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV 2004*, pages 25–36. Springer Berlin, 2004.

[7] Thomas Brox, Andrés Bruhn, and Joachim Weickert. Variational motion segmentation with level sets. In *ECCV 2006*, pages 471–483. Springer Berlin, 2006.

[8] Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270, New York, NY, USA, 1993. ACM.

[9] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *International Journal of Computer Vision*, 22:61–79, Feb 1997.

[10] Tony Chan and Jianhong Shen. *Image Processing And Analysis: Variational, Pde, Wavelet, And Stochastic Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005.

[11] Christopher M. Christoudias, Bogdan Georgescu, and Peter Meer. Synergism in low level vision. In *ICPR '02: Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 4*, page 40150, Washington, DC, USA, 2002. IEEE Computer Society.

[12] Charles K. Chui. *An introduction to wavelets*. Academic Press Professional, Inc., San Diego, CA, USA, 1992.

[13] L. D. Cohen. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing. Image Understanding*, 53(2):211–218, 1991.

[14] John P. Collomosse, David Rowntree, and Peter M. Hall. Stroke surfaces: Temporally coherent artistic animations from video. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):540–549, 2005.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[16] Daniel Cremers and Stefano Soatto. Motion competition: A variational approach to piecewise parametric motion segmentation. *Int. J. Comput. Vision*, 62(3):249–265, 2005.

[17] I. Daubechies. Orthonormal bases of compactly supported wavelets. 41:909–996, November 1988.

[18] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. In *Proceedings of SIGGRAPH '02*, pages 769–776, 2002.

[19] Betty Edwards. *Drawing on the right side of the brain*. J. P. Tarcher, Los Angeles, California, USA, 1979.

[20] James H. Elder. Are edges incomplete? *Int. J. Comput. Vision*, 34(2-3):97–122, 1999.

[21] Lawrence C. Evans. *Partial Differential Equations (Graduate Studies in Mathematics, V. 19) GSM/19*. American Mathematical Society, June 1998.

[22] George M. Ewing. *Calculus of Variations with Applications*. Dover, New York, 1969.

[23] Adam Finkelstein and David H. Salesin. Multiresolution curves. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 261–268, New York, NY, USA, 1994. ACM.

[24] F. Franke. A critical comparison of some methods for interpolation of scattered data. Technical Report NPS-53-79-03, Naval Postgraduate School, 1979.

[25] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. pages 452–472, 1990.

[26] Graham M.L. Gladwell. *Inverse Problems in Vibration*. Kluwer Academic Publishers, Dordrecht, second edition, 2004.

[27] Bruce Gooch, Erik Reinhard, and Amy Gooch. Human facial illustrations: Creation and psychophysical evaluation. *ACM Trans. Graph.*, 23(1):27–44, 2004.

[28] Steven J. Gortler and Michael F. Cohen. Hierarchical and variational geometric modeling with wavelets. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 35–ff., New York, NY, USA, 1995. ACM.

[29] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[30] James Hays and Irfan Essa. Image and video based painterly animation. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 113–120, New York, NY, USA, 2004. ACM Press.

[31] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460, New York, NY, USA, 1998. ACM.

[32] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[33] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098 –1101, sep. 1952.

[34] R. Illner and H. Neunzert. Relative entropy maximization and directed diffusion equations. *Mathematical Methods in the Applied Sciences*, 16:545–554, August 1993.

[35] Henry Kang, Seungyong Lee, and Charles K. Chui. Coherent line drawing. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 43–50, New York, NY, USA, 2007. ACM.

[36] Michael Kass and Andrew Witkin. Analyzing oriented patterns. *Comput. Vision Graph. Image Process.*, 37(3):362–385, 1987.

[37] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, Jan 1988.

[38] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286, 2003.

[39] Jan Eric Kyprianidis and J. Döllner. Image abstraction by structure adaptive filtering. In *Proc. EG UK Theory and Practice of Computer Graphics*, pages 51–58, 2008.

[40] Michael J. Langford. *Advanced Photography: A Grammar of Techniques*. Focal Press, London, 1974.

[41] Yvan G. Leclerc. Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3(1):73–102, 1989.

[42] Gregory Lecot and Bruno Lévy. Ardeco: Automatic region detection and conversion. In *Eurographics Symposium on Rendering*, 2006.

[43] Dennis Lee. DLI image compression . http://sites.google.com/site/dlimagecomp/software, 2010.

[44] Peter Litwinowicz. Processing images and video for an impressionist effect. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 407–414. ACM Press/Addison-Wesley Publishing Co., 1997.

[45] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987.

[46] S.G. Mallat. Multifrequency channel decompositions of images and wavelet models. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(12):2091 –2110, dec. 1989.

[47] P. Maragos and R. Schafer. Morphological skeleton representation and coding of binary images. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, 34(5):1228–1244, October 1986.

[48] Scott McCloud. *Understanding Comics*. HarperCollins, 1994.

[49] T. McInerney and D. Terzopoulos. Topologically adaptable snakes. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, page 840, Washington, DC, USA, 1995. IEEE Computer Society.

[50] Tim McInerney and Demetri Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4(2):73–91, 2000.

[51] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42:577–685, 1989.

[52] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–8, New York, NY, USA, 2008. ACM.

[53] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7):629–639, 1990.

[54] G. Plonka and M. Tasche. On the computation of periodic spline wavelets. *Appl. Comput. Harmonic Anal.*, 2:1–14, 1995.

[55] Merle C. Potter, Jack L. Goldberg, and Merle C. Potter. *Mathematical methods / Merle C. Potter, Jack Goldberg*. Prentice-Hall, Englewood Cliffs, N.J. :, 2nd ed. edition, 1987.

[56] Brian Price and William Barrett. Object-based vectorization for interactive image editing. *Vis. Comput.*, 22(9):661–670, 2006.

[57] E. Quak and N. Weyrich. Decomposition and reconstruction algorithms for spline wavelets on a bounded interval. *Applied and Computational Harmonic Analysis*, 1994.

[58] Ali Rahimi. Fast connected components on images. http://xenia.media.mit.edu/~rahimi/connected/, 2001.

[59] A. Ravishankar Rao and Brian G. Schunck. Computing oriented texture fields. *CVGIP: Graph. Models Image Process.*, 53(2):157–185, 1991.

[60] Goudong Rong. *Jump Flooding Algorithm on Graphics Hardware and Its Applications*. PhD thesis, National University of Singapore, 2007.

[61] Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, 1966.

[62] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.

[63] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, second edition, 2003.

[64] Thomas Schoenemann and Daniel Cremers. Near real-time motion segmentation using graph cuts. In *28th DAGM Symposium*, pages 455–464. Springer Berlin, 2006.

[65] Peter Selinger. Potrace: a polygon-based tracing algorithm. http://potrace.sourceforge.net/potrace.pdf, Sep 2003.

[66] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[67] James Stewart. Additional Topics: Nonhomogeneous linear equations. In *Calculus: Concepts and Contexts*. Brooks Cole, forth edition, 2009.

[68] Nicolas Stoiber. Compression of images and videos by geometrization. Master's thesis, Technische Universitat Munchen, September 2007.

[69] Eric J. Stollnitz, Tony D. Derose, and David H. Salesin. *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[70] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Trans. Graph.*, 26(3):11, 2007.

[71] Adobe Systems. *Photoshop 7.0 User Guide*. Adobe Systems, 2002.

[72] Demetri Terzopoulos. *Multiresolution Computation of Visible-Surface Representations*. PhD thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, January 1984.

[73] Luminita A. Vese and Tony F. Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *International Journal of Computer Vision*, 50:271–293, Dec 2002.

[74] Jue Wang, Yingqing Xu, Heung-Yeung Shum, and Michael F. Cohen. Video tooning. *ACM Trans. Graph.*, 23(3):574–583, 2004.

[75] Joachim Weickert. *Anisotropic diffusion in image processing*. PhD thesis, Dept. of Mathematics, University of Kaiserslautern, January 1996.

[76] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *ACM Trans. Graph.*, 25(3):1221–1226, 2006.

[77] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data Structure for Soft Objects. *The Visual Computer*, 2(4):227–234, February 1986.

[78] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Trans. Graph.*, 28(5):1–10, 2009.

[79] Song Chun Zhu and A. Yuille. Region competition: unifying snakes, region growing, and Bayes/MDL for multiband image segmentation. *Transactions on Pattern Analysis and Machine Intelligence*, 18(9):884–900, Sep 1996.