NORTHWESTERN UNIVERSITY

Towards Efficient and Accurate Image Matting

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Electrical Engineering

by

Philip G. Lee

EVANSTON, ILLINOIS

June 2014

Contents

List of Tables

List of Figures

List of Algorithms

ABSTRACT

Towards Efficient and Accurate Image Matting

Philip G. Lee

Matting is an attempt to separate foreground from background layers. There are three main problems to address: improving accuracy, reducing required input, and improving efficiency. In our work L_1 *Matting* [16], we improved the accuracy by extending a method to include information from the L_1 norm via median filters, which provides noise resiliency and preserves sharp image structures. In our work *Nonlocal Matting* [17], we considerably reduce the amount of user effort required. The key observation is that the nonlocal principle, introduced to denoise images, can be successfully applied to the alpha matte to obtain sparsity in matte representation, and therefore dramatically reduce the number of pixels a user needs to manually label. We show how to avoid making the user provide redundant and unnecessary input, develop a method for clustering the image pixels for the user to label, and a method to perform high-quality matte extraction. To address the issue of efficiency, we develop specialized multigrid algorithms in *Scalable Matting* [18] taking advantage of previously unutilized priors, reducing the computational complexity from $\mathcal{O}\left(n^2\right)$ to $\mathcal{O}\left(n^{0.752}\right)$. Further, we show that the efficient methods used for image matting extend to a wide class of image formation problems.

Acknowledgment

Research is not a solitary art, but on the contrary is a collaborative one. Only a few names are attached to any single published article, but for each one of them there are many more who are to thank. For those whose names may not explicitly mentioned elsewhere in this book, I would like to give them full credit and my appreciative thanks here.

First, I thank my wife, Leidamarie Tirado-Lee for her consistent and ample moral support in my tenure as a Ph.D. student. Being not just my wife and best friend, she is also a Ph.D. student herself and no matter what was happening, she was always there to sympathize and empathize. My parents Marion and Kathy Isom also supported me throughout my studies, for which I thank them very much. I also thank the members of my committee, Dr. Aggelos Katsaggelos and Dr. Thrasyvoulos Pappas for guiding me during my candidacy. Further, my friends and colleagues Adam Barber, Alireza Bonakdar, Robert Brown, Iman Hassani, Steven Manuel, Timothy Rambo, Esteban Rangel, Vlad Seghete, Jian Shi, Cynthia Solomon, and Matthew Wampler-Doty deserve much credit in their ability to make me smile and share in the crazy world of academic research.

Northwestern is a great place for the free exchange of ideas, and I would like to thank several people for their intellectual discussions. First is Professor Jorge Nocedal, whose class in optimization I took, and who immediately understood the problems I was having in my own research. Second is Associate Professor Paul Umbanhowar, who visited like clockwork just to discuss whatever was currently happening in my Matlab session, and who was really great to bounce ideas with. I would also like to thank Professor Kevin Lynch for letting me help to construct his dynamic robotic manipulation system. It was a great change of pace, and refereshing to think about different problems.

There are a few individuals that deserve special recognition. First is Dr. Ilya Mikhelson. He was my first friend after moving to Evanston, and became one of my closest. Not only did we collaborate professionally, but we met at least once a week to discuss research, projects, philosophy, and personal subjects. I cannot imagine the past five years without him. Second is Yin Xia. He is another good friend who, in addition to discussing research, taught me much about Chinese culture. We shared many conversations about politics, culture, and computer vision, and enjoyed many craft beers together. Professor Alan Sahakian also deserves special recognition. Although I know he was always extremely busy with running the department and his own research, he never once turned me away if I had a question about anything, even if it was just an elementary question about some electronic component. I am most grateful for the advice he gave me during the middle of my tenure, when I was considering quitting my Ph.D. I am very glad it worked out otherwise.

Finally, I thank my advisor Ying Wu for the entire experience at Northwestern. He brought me onboard with enthusiasm from the very beginning. Unlike many, he is a true scientist searching for fundamental truths in our field. He impressed upon me the importance of simplicity and clear communication in everything. As time went on, he also became a paternal figure of mine, discussing many things including his philosophies of life and happiness. Mark Twain said: “Great people are those who make others feel that they, too, can become great.” Dr. Wu is such a person. I truly appreciate the wisdom he imparted.

Preface

I have little doubt this book is presently either *actively* under a projector, or waiting nearby to be propped under one. If by chance someone is actually reading it, then you, dear reader, must be in a state of great depression. I know this, because the only time I ever had to resort to reading someone else's thesis, it was out of complete and utter frustration, and I think I never once found whatever it was I was looking for in the first place.

Education: the path from cocky ignorance to miserable uncertainty. – Mark Twain

You see, graduate work is a tough and lonely business. At the beginning, you are bright-eyed and bushy-tailed and ready to take on the world. However, you quickly realize that you have just moved somewhere where you know noone and nobody in your lab is working with you on anything. You're just that new guy that asks all the annoying questions.

Finally, after a semester or two, you're forced into some distasteful collusion in order to do a group project that will reduce the number of papers this professor has to grade, and you crawl from your dungeon into the light and surprisingly meet a friend. He still doesn't work on the project, but *who cares* at this point? He's actually kind of normal and wants to go to the gym and watch movies and whatnot.

However, he still doesn't understand whatever it is that your advisor assigned you to work on. In fact, now that you yourself have passed your glazed eyes over the background material, you aren't sure either. What is this nonsense about? Why does it say "obviously" here when it really means "we hit the page limit, so figure it out?" And why is it that none of these fools post any code to clarify that one sentence that described all 42 parameters in their stupid algorithm? Even worse...a Windows® binary.

But, at least there are the really old papers that are pretty simple. You implemented them in MATLAB® (that god-awful hunk of junk) in a day. It took two weeks to run and the results looked like *Ecce Mono*[1], but you got there. There are even those two algorithms that you can both understand *and* that nearly work.

At least, that was my experience at the beginning of my Ph.D. As time went on, I gradually got used to the insanity of Computer Vision articles in places like CVPR, but there was something really wrong about it all. I didn't believe in Computer Vision. Almost every article I tried to read seemed like they were trying to build a Maserati by duct-taping rocks to a telephone pole, and then trying to sell me on it by showing me how much more robust it was than the other telephone poles.

Get your facts first, then you can distort them as you please. – Mark Twain

This was a big shock coming from a Math degree. In those papers, you can hardly hide a lie or even stretch the truth a bit. There is no salesmanship, and there is only what is true and what is not. If you can't prove something, you don't write a paper. In Computer Vision, if you can't prove something, you invent a dataset that is secretly biased in favor of your algorithm, and you tweak the parameters for every single input and make a table showing how you can beat the other algorithms by 0.1% 95% of the time. After I started to realize this, it made me very disappointed and I nearly quit because of it. There *is* something wrong with Computer Vision. In fact, there are many things.

First, this is basically Computer Science, and computer scientists should always post free code in the general interest of science. This does not happen in Computer Vision nearly often enough. Richard Stallman (founder of GNU[2]) always used to argue that free code was safe code, because it is very difficult to hide a backdoor in the software when you have so many eyeballs looking at the source code. In the same way, it is easy to hide dirty algorithmic tricks when you only give a binary, and it is possible to completely manufacture *all* of the results when not even that is

[1] *Ecce Homo*, as restored by Cecilia Giménez. Go look it up; you won't regret it.

[2] `http://www.gnu.org`

provided by the authors. If an author doesn't give his code so that I can compile it and run it on the same dataset and get the same results, it is a bullshit paper.

Let me provide a case study on the problems that lack of source code creates. Stay with me, because first we have to understand the problem to understand why we need source code. Some benchmarks, like the popular `alphamatting.com` [27] that I (am forced to) use in my work, have no way to prevent "human tuning." It is supposed to work like this: half of the dataset is given to you along with the ground truth result. You are supposed to use this for training to set the parameters of your algorithm. Then, you are asked to run the algorithm on the second half of the dataset, which does *not* contain the ground truth, and submit the results for blind analysis. Then, they compute a bunch of metrics using their secret ground truth and give it back to you. You yourself are to run your algorithm on your own machine and submit the results, asserting by the "honor system" that you did not manipulate the parameters or do anything nefarious to get the result. This is patently absurd, because in practice what happens is that I submit for the first time, then immediately start tuning my parameters according to the benchmark results until I start to see good results, completely ignoring the training dataset. The training dataset becomes the results of *other* alpha matting algorithms, and the learning algorithm becomes "learning by graduate student."

It gets *much* worse in this particular benchmark: I can reverse engineer the hidden ground truth and get a *perfect* score! The attack works as follows. One of the statistics you get back from the evaluation is the mean squared error (MSE) to the hidden ground truth. This is a "leaky channel" that spews out information about the ground truth. Suppose I am malicious (and I am), and for simplicity assume that there is only one matte to be evaluated. I can extract the *full ground truth* α^{gt} in the following manner:

1. Submit $\alpha = 0$ to the evaluation, and get back the MSE, which is a scalar multiple of $(\alpha^{\text{gt}})^T \alpha^{\text{gt}}$.
2. The metric you get is equivalent to $\|\alpha - \alpha^{\text{gt}}\|^2$ which is equal to $\alpha^T \alpha - 2\alpha^T \alpha^{\text{gt}} + (\alpha^{\text{gt}})^T \alpha^{\text{gt}}$.
3. We got the last term from the first step, and $\alpha^T \alpha$ is observable to us, so we can get $\alpha^T \alpha^{\text{gt}}$ by simple arithmetic after retrieving the MSE from the evaluation.
4. Set $\alpha_1 = 1$ and leave the rest 0. Submit, get the MSE, and calculate $\alpha^T \alpha^{\text{gt}} = \alpha_1^{\text{gt}}$.
5. Repeat for all pixels i to get α_i^{gt}.
6. Now, submit the calculated α^{gt} to the evaluation and get a perfect score!

What is the solution to this stupidity? It is to prohibit the authors of these algorithms from running their algorithm on their own machines. It is to force the authors to provide their *source code* to the benchmark people, and have the author of a *competing* algorithm review it, compile it, and run it. Better yet, the code should be attached to the paper in the review process, and the reviewers should be the ones to generate and analyze the benchmark results. No code? Good luck getting published without benchmark results.

Why is the current state of affairs acceptable in Computer Vision? Surely I am not the first person to see the lack of academic integrity in the evaluation of our algorithms. Maybe it's laziness; maybe it's the urgent need to publish; maybe it's just true ignorance.

Secondly, the review process for conferences like CVPR is horrendous. There are many reasons for that. First is that the reviewers are all graduate students with no idea and no *desire* to throughly read and understand the paper. I know, because I was one. It was impossible for me to ascertain the novelty of a paper, because I simply didn't have the experience necessary to do so. It was impossible for me to ascertain the importance of the work for the same reason. And finally, it was extremely difficult for me to understand the papers I *did* read thoroughly. This one is a chicken-and-egg problem. Suppose you write a paper that is impossible to work through, with 20 equations on every page. I can spend weeks to figure out what is going on, or I can do my homework. Further, if it looks complicated, there is a tendency among us to feel inept and unable to comprehend the "great mind" who wrote it. So, complicated papers get glossed over and accepted. Now, suppose you write a very simple paper with few equations and in a conversational tone. It is so easy to read that I can find some logical flaws. Rejected. The lesson you quickly learn is not to explain your ideas intuitively, but through symbols and layered notation that nobody will dare to read. CVPR reviewers will be familiar with the phrase "seems correct, but did not check completely."

All you need is ignorance and confidence and the success is sure. – Mark Twain

There is one more thing that mucks up the whole process: single stage review. You submit your paper. You wait 3 months. They come back terrible, as no one bothered to try to understand it, and you write your rebuttal. In practice, the rebuttals are never read, and no decisions are ever changed from them. To change my review would be to admit I was wrong, which is completely against human nature. No reviewer ever admits his uncertainty. We should not submit articles to conferences and journals where we are not expected to make significant changes after the reviews come back. This kind of expectation puts pressure on the reviewer to say under what conditions the paper must change so that he will accept it. Then, he must either read it enough to give suggestions or just be lazy and defer to the other reviewers.

Finally, very few researchers seem to care about the computational complexity of the algorithms they propose, which is absurd for computer scientists. In case you are not already aware, the complexity is truly spectacular for most of the proposed algorithms in the field. It is evident to see this by looking at the OpenCV[3] library. It is a collection of `C++` routines to create vision algorithms. While I dislike the design of the library, it is very good about including functions that run and could perhaps be composed to run near real time. Take a look, and you will see how small the number of functions is compared with the number of published vision algorithms being proposed every year (part of this is the unwilligness of researchers to produce open source code). I laugh out loud when I see someone requiring an Eigenvalue decomposition on some huge matrix. Almost no paper describes an algorithm fast enough to get included in OpenCV.

When I realized this mid-way through my Ph.D., I found it really irked me, and I finally found my passion in the field. I wanted to make scalable algorithms. In spite of the fact that half of Part II does not consider scalability, the real motivation for me to write this thesis is for the odd student to pick it up from under the projector, read it, and decide that scalability is important.

Let me sing its merits. First, it is much less fluffy than most of the field. For example, image segmentation has a gaping hole in its bid for existence: how do you define a good segmentation? However, if you can reduce the complexity by a factor of 10, great! That is a very clear goal and accomplishment. Second, don't you want someone to actually run your algorithm in the future? As I am writing this, there is a 41 Mpx camera/phone[4] on the consumer market. If you can't be efficient on a 256×256 patch, it is just going to be forgotten, and for a very good reason. Finally, complexity analysis and writing good code to demonstrate it are going to make you a good computer scientist and software engineer. The market for programmers is extremely hot, and I don't ever see it cooling down. There is now talk of including programming in high-school curricula as a core course. Everything is being programmed, and if you can do something faster than anyone else, that is a valuable skill in the job market.

Before you, dear graduate student reader, put this book down, I'd like to share some tips with you that I wish I would have had.

- Don't work on fluffy things (research or otherwise).
- Revision control *everything*! Learn to use `git` and Github or Gitorious or some other hosting service.
  - For unpublished papers, I highly recommend writing LaTeX code version-controlled with `git`, whose "origin" is a bare repository in your Dropbox or other cloud storage folder.
- Never write closed code. License your code under the GPL and share it (again, Github and Gitorious are great).
- Use `arXiv.org` to submit your pre-prints when you submit to a journal/conference. It lets you cite the paper and move on in your research immediately, prevents others from scooping you, and gives every other scientist a free way to access your work.
- Don't submit to CVPR.
- For the love of Thor, wipe your hard drive of all Microsoft products and install Linux. Debian[5] is a sane distribution.
- Thoroughly read and understand any articles assigned to you to review. If you can't understand it, admit it, and refuse to accept it until the arguments are simplified. Otherwise, provide a detailed review and give the conditions under which you will accept this article explicitly.

[3] `http://opencv.org`
[4] Nokia Lumia
[5] `http://www.debian.org`

- Spend your weekends with your friends or looking for a date.
- Collaborate with as many people as possible, even if just informally.
- Read *How to Win Friends and Influence People* by Dale Carnegie.
- Don't worry about failure in research (or anywhere else): 90% of your ideas will be crap.
- Go find your classmates in their offices/dungeons daily.
- If you're at Northwestern, go check out the craft beer scene like Revolution Brewpub, Half Acre, Piece, Finch, Temperance, etc. This beer is so good and the people are so nice!

To conclude, I want to remind you that you are a scientist. Be objective. Be open-minded. Reproduce others' work. Make your whole work freely available. Contribute to society. Take chances, and take criticism.

Part I

Prior Work

Chapter 1

Introduction

The problem of extracting an object from a natural scene is referred to as alpha matting. Each pixel i is assumed to be a convex combination of foreground and background colors F_i and B_i with $\alpha_i \in [0, 1]$ being the mixing coefficient:

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i. \tag{1.1}$$

Since there are 3 unknowns to estimate at each pixel, the problem is severely underconstrained. Most modern algorithms build complex local color or feature models in order to estimate α. One of the most popular and influential works is [19], which explored graph Laplacians based on color similarity as an approach to solve the matting problem.

The model that most Laplacian-based matting procedures use is

$$p\left(I \mid \alpha\right) \propto \exp\left(-\frac{1}{2}\left\|\alpha\right\|_{L^G}^2\right) \text{ and} \tag{1.2}$$

$$\alpha_{\text{ML}} = \arg\max_{\alpha} p(I \mid \alpha), \tag{1.3}$$

where L is a per-pixel positive semi-definite matrix that represents a graph Laplacian over the pixels, and is typically very sparse. However, L^G always has positive nullity (several vanishing eigenvalues), meaning that the estimate is not unique. So, user input is required to provide the crude prior:

$$p(\alpha) \propto \begin{cases} 1 & \text{if } \alpha \text{ is consistent with user input} \\ 0 & \text{o.w.} \end{cases}. \tag{1.4}$$

This is usually implemented by asking the user to provide "scribbles" to constrain some values of the matte to 1 (foreground) or 0 (background). This gives a maximum a-posteriori estimation

$$\alpha_{\text{MAP}} = \arg\max_{\alpha} p(I \mid \alpha) p(\alpha), \tag{1.5}$$

which may be unique depending on the constraints provided by the prior.

To get the constrained matting Laplacian problem into the linear form $Ax = b$, one typically chooses

$$(L^G + \gamma C)\alpha = \gamma f, \tag{1.6}$$

where C is diagonal, $C_{ii} = 1$ if pixel i is user-constrained to value f_i and 0 o.w., and γ is the Lagrange multiplier for those constraints. $\gamma \approx 10$ seems to work well in practice, and is not a very sensitive parameter.

Besides the motivation to solve the matting problem for graphics purposes like background replacement, there are many computer vision problems that can be cast as finding the matte for an image like dehazing [10], deblurring [3], and even tracking. The last is particularly attractive as shown by [7], because the object to be tracked can be considered the foreground, and the matting constraints can be propagated by very simple methods like SIFT [21] matching to provide a robust closed-loop tracker as demonstrated in Fig. 1.1. To be useful though, such a tracker and therefore the matting algorithm have to be real time. But, this becomes very difficult when the constraints are sparse as is the case with SIFT matching, or when the image size grows large.

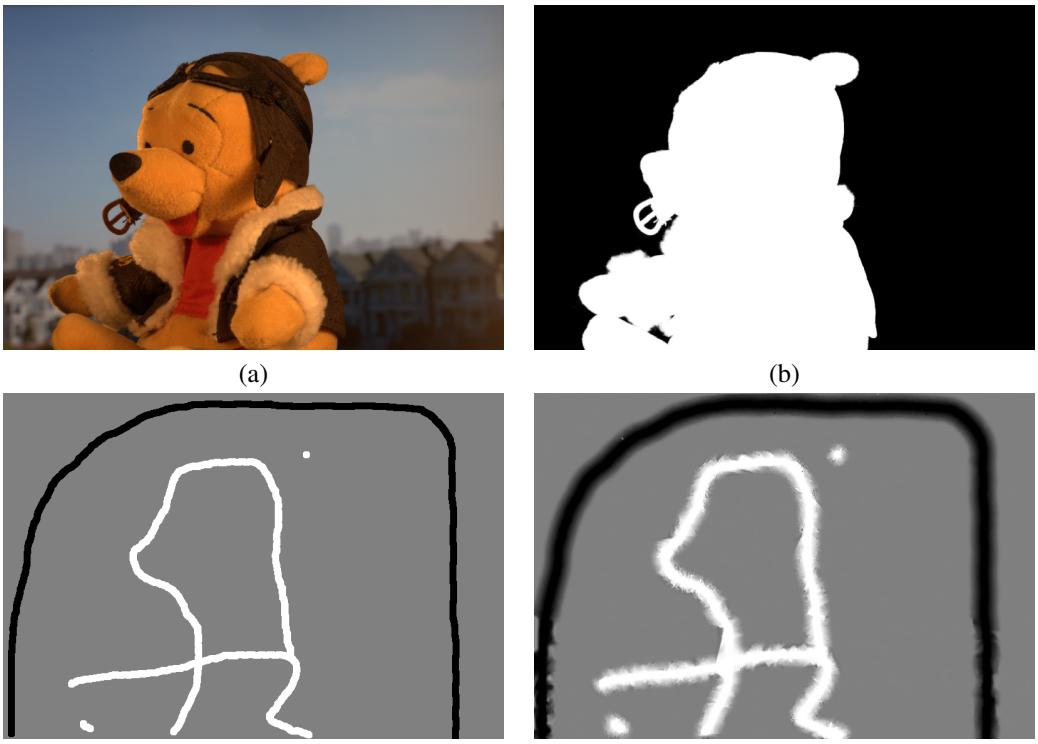
Figure 1.1: Robust closed-loop tracking by matting.

Since A is symmetric and positive definite (with enough constraints), it is tempting to solve the system by a Cholesky decomposition. However, this is simply impossible on a modest-size image on modern consumer hardware, since such decompositions rapidly exhaust gigabytes of memory when the number of pixels is larger than around 10 kilopixels (100×100).

It is also tempting to use the equality constraints directly, which reduces the system size by the number of constrained pixels. In order to solve this reduced system in memory would again require the number of unconstrained pixels to be less than about 10k. This is also impractical, because it places the burden on the user or tracking algorithm to constrain at least 99 % of the pixels in megapixel and larger images.

It is therefore necessary to resort to more memory-efficient iterative algorithms, which take advantage of the sparsity of A. There are many available iterative methods for solving the full-scale matting problem such as gradient descent, the Jacobi method, or conjugate gradient descent. However, although memory-efficient, they are all *incredibly* slow, especially as the resolution increases. There is a fundamental reason for this: the condition number of the system is much too large for stable iterations without dense constraints. Since the Laplacians are defined on a regular pixel lattice, its top eigenvectors correspond to high frequency kernels. Bad conditioning combined with the Laplacian structure causes the residual error at each iteration to be primarily composed of high-frequency components. So, the iterations focus on reducing the high-frequency errors first and the low-frequency errors last (see Fig. 1.2.d, which is mostly a lowpass version of Fig. 1.2.c). To have better effective conditioning, we draw inspiration from [11] who took the approach of defining a new Laplacian with very large kernels to trade off better conditioning at the expense of some quality. But, there is a more natural and more general approach that does not come at the sacrifice of quality or freedom of choice for the Laplacian.

Multigrid methods have been briefly mentioned by [31] and [11] for solving the matting problem, but they conclude the irregularity of the Laplacian is too much for multigrid methods to overcome. However, we show that this is not a problem at all when the upsample and downsample operations that define the grids are appropriately constructed. What we propose is a type of multigrid solver, which solves the full-scale problem *unaltered*, does not restrict the choice of the Laplacian, gives huge speedups without *any* sacrifice in quality, and scales *linearly* in the number of pixels. We further propose better priors that may partially or fully resolve the ambiguity of the problem in common situations.



(a) (b)

(c) (d)

Figure 1.2: Megapixel image (a), ground truth matte (b), user constraints (c), 20 iterations of conjugate gradient (d)

Chapter 2

Existing Matting Methods

2.1 Bayesian Matting

Bayesian matting [2] takes a naive Bayesian approach to matting where all of the distributions involved are multivariate Gaussians. So, they would like to solve

$$\arg\max_{F,B,\alpha} L(I \mid F, B, \alpha) + L(F) + L(B) \tag{2.1}$$

$$L(I \mid F, B, \alpha) = -\left\|I - (\alpha F + (1-\alpha)B)\right\|^2 / \sigma_C^2 \tag{2.2}$$

$$L(F) = (F - \bar{F})^T \Sigma_F^{-1} (F - \bar{F}) \tag{2.3}$$

$$L(B) = (B - \bar{B})^T \Sigma_B^{-1} (B - \bar{B}) \tag{2.4}$$

where L is the log likelihood. This is represented graphically in Fig. 2.1.

$L(I \mid F, B, \alpha) + L(F) + L(B)$ is not quadratic or convex necessarily. So, in order to solve it, they note that fixing α makes the equation quadratic w.r.t F and B and vice versa and attempt solve it by doing an alternating optimization on colors and α values. Notice this paper has no prior term $L(\alpha)$.

2.2 The Matting Laplacian

Matting by graph Laplacian is a common technique in matting literature [19, 33, 11, 17]. Laplacians play a significant role in graph theory and have wide applications, so we will take a moment to describe them here. Given an undirected graph $G = (V, E)$, the adjacency matrix of G is a matrix A_{ij}^G of size $|V| \times |V|$ that is positive $\forall v_i v_j \in E$. A_{ij}^G is the "weight" on the graph between v_i and v_j. Further define $D_{ii}^G = d(v_i) = \sum_{j \neq i} A_{ij}^G$ to be a diagonal matrix containing the degree of v_i, and let x be a real-valued function $V \to \mathbb{R}$. The Laplacian $L^G = D^G - A^G$ defines a particularly nice quadratic form on the graph

$$q(x) \triangleq x^T L^G x = \sum_{v_i v_j \in E} A_{ij}^G (x_i - x_j)^2. \tag{2.5}$$

By construction, it is easy to see that L is real, symmetric, and positive semi-definite, and has $\mathbf{1}$ (the constant vector) as its first eigenvector with eigenvalue $\lambda_1 = 0$.

This quadratic form plays a crucial role in matting literature, where the graph structure is the grid of pixels, and the function over the vertices is the alpha matte $\alpha \in [0, 1]^{|V|}$. By designing the matrix A_{ij}^G such that it is large when we expect α_i and α_j are the same and small when we expect they are unrelated, then minimizing $q(\alpha)$ over the alpha matte is finding a matte that best matches our expectations. The problem is that, as we have shown, one minimizer is always the constant matte $\mathbf{1}$, so extra constraints are always required on α.

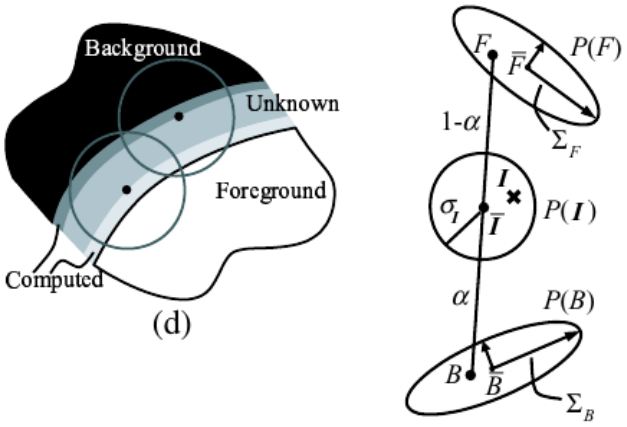
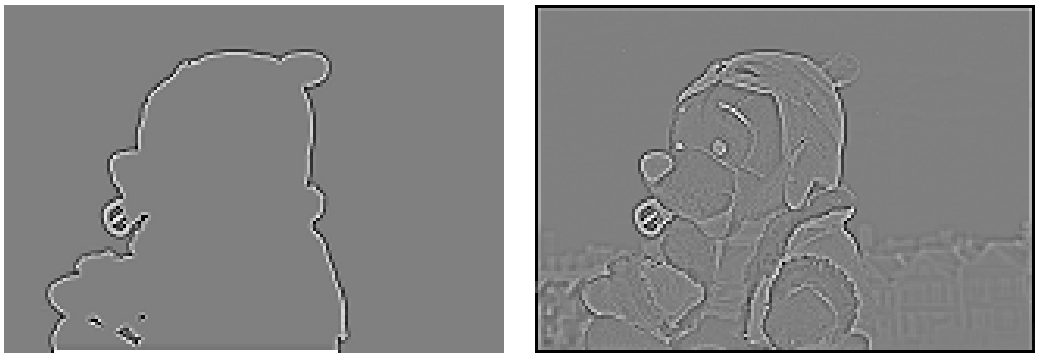


Figure 2.1: Pixel I is a linear combination of foreground and background colors F and B, which are normally distributed.



$\Delta\alpha$ ΔI

2.3 Poisson Matting

The idea presented by [30] was to use the concept of Poisson image editing [25] to construct the Laplacian. Somehow, it is assumed that the gradients of the given matte are known, but that the values themselves need to be reconstructed by solving the Poisson equations

$$\Delta\alpha = g \tag{2.6}$$

$$\text{s.t. } \alpha_\Omega = f, \tag{2.7}$$

where Δ is the Laplace operator, Ω is the domain over which the matte is known to be f, and g is a forcing function.

The forcing function they decided on is based on the observation that the alpha matte and the image have similar gradients along the foreground/background boundary up to a multiplicative scaling factor as demonstrated by Fig. 2.3. This is somewhat justifiable mathematically by the observation that since

$$I = \alpha(F - B) + B, \tag{2.8}$$

$$\Delta\alpha \approx \frac{1}{F - B}\Delta I. \tag{2.9}$$

Notice in this case that the matting Laplacian $L^G = \Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2$.

The main issue with this method is that it is not very accurate unless the constraints are very dense, because the underlying assumption is that the field $(F - B)$ is smooth and nonzero which is regularly violated. For this reason, detailed user input is required to make up for the deficiencies in the assumptions.

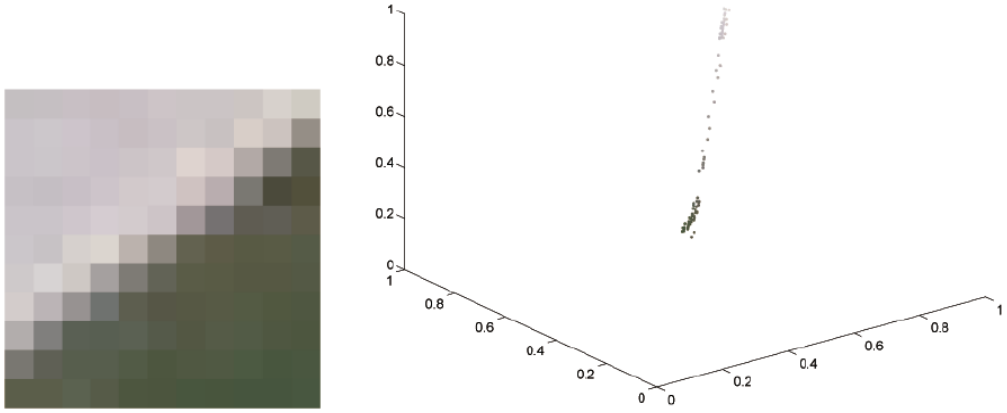


Figure 2.2: Colors tend to be locally linearly distributed.

2.4 Closed Form Matting

In [19], A_{ij}^G on the graph is given as

$$\sum_{k|(i,j)\in w_k} \frac{1}{|w_k|}\left(1+\langle I_i-\mu_k, I_j-\mu_k\rangle_{(\Sigma_k+\epsilon I)^{-1}}\right), \tag{2.10}$$

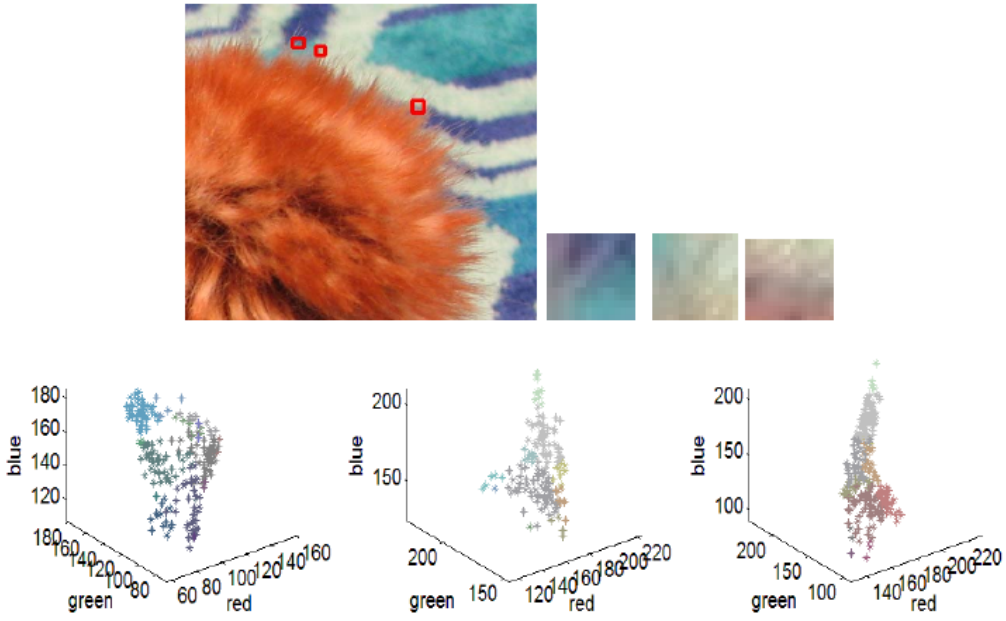
where w_k is a spatial window of pixel k, μ_k is the mean color in w_k, and Σ_k is the sample color channel covariance matrix in w_k. This is probably best understood as a Mahalanobis inner product in color space. Ignoring the regularizing ϵ term, defining the whitening transformation $J_i(k) = \Sigma_k^{-1/2}(I_i - \mu_k)$, the affinity above can be interpreted as an expectation $E_k[\langle J_i(k), J_j(k)\rangle]$, the covariance of the whitened pixels $J_i(k)$ and $J_j(k)$ over windows k. In essence, this is an affinity based on a Gaussian color model defined by Σ_k and μ_k, and the resulting matte can be interpreted as a backprojection (or likelihood map) of the combined local color models.

The reason that such a model is acceptable is due to the linearity of natural image colors in local windows known as the color line model (Fig. 2.2). Though this model does not always hold, it is good enough for many situations.

[29] demonstrated that [19]'s Laplacian intrinsically has a nullity of 4, meaning that whatever prior is used, it must provide enough information for each of the 4 subgraphs and their relationships to each other in order to find a non-trivial solution.

2.5 Learning-Based Matting

In [33], the color-based affinity in eq. (2.10) is extended to an infinite-dimensional feature space by using the kernel trick with a gaussian kernel to replace the RGB Mahalanobis inner product. [5] provides many other uses of graph Laplacians, not only in matting or segmentation, but also in other instances of *transductive learning*. In *inductive* learning, we are given a training set distinct from the test set, and must learn everything from the training set. The difference with other learning methods is that in transductive learning, we already know the whole dataset that we will be asked to label (the pixels in matting), and that gives us information about how they are to be labeled. For example, if A_{ij}^G is large, we should believe that samples i and j share the same label, even though we may not know what the label is.


blue
green
red
blue
green
red
blue
green
red

Chapter 3

Existing Solution Methods

Although we have already hinted at some ways to solve the matting problem, here we would like to take an in-depth look at the general structure of the problem and the approaches that others have taken to provide a solution.

3.1 Decomposition

It is often the case that certain decompositions of matrices can transform their solution into an easier form. The insight is that there are some matrices for which we can efficiently solve the inverse problem. For example, in the case that an $n \times n$ matrix A is diagonal (A_{ij} nonzero only if $i = j$), then the solution to

$$Ax = b \tag{3.1}$$

is simply $b_i = x_i / A_{ii}$. It is easy to see that such a solution requires only n divisions, and is therefore a very efficient $\mathcal{O}(n)$.

Another type of matrix that is easy to solve is the unitary matrix. We say a real square matrix U is *unitary* if and only if

$$U^T U = U U^T = I. \tag{3.2}$$

In other words, $U^T = U^{-1}$. In this case, solving

$$Ux = b \tag{3.3}$$

is as easy as

$$x = U^T b. \tag{3.4}$$

In this case, each element of x requires n multiplications, so the total complexity is $\mathcal{O}\left(n^2\right)$. Of course we are often not lucky enough to have unitary systems, but with some manipulation, unitary matrices often drop out of the analysis.

A slightly more complex type of matrix that we can still solve easily is the upper-triangular matrix. We say that a matrix R is *upper-triangular* if and only if all the nonzeros lie on or above the main diagonal. In other words:

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} & \dots \\ 0 & R_{22} & R_{23} & \dots \\ 0 & 0 & \ddots & \dots \\ 0 & 0 & 0 & R_{nn} \end{bmatrix}. \tag{3.5}$$

In this case, if we have $Rx = b$, then it is clear that $b_n = x_n / R_{nn}$. Then, we can see

$$R_{(n-1)(n-1)} x_{n-1} + R_{(n-1)n} x_{nn} = b_{n-1} \implies \tag{3.6}$$

$$x_{n-1} = \frac{1}{R_{(n-1)(n-1)}} \left(b_{n-1} - R_{(n-1)n} x_n \right). \tag{3.7}$$

We can follow this recursive logic for all elements of the solution to get the general algorithm known as backsubstitution (see Algorithm 1). Since it is clear that element x_i needs $(n-i)$ multiplications, the complexity of the solution is $\mathcal{O}\left(n^2\right)$.

Algorithm 1 Backsubstitution for upper-triangular systems

```
for i = n..1 do
    s ← b_i
    for j = i + 1..n do
        s ← s − R_ij x_j
    end for
    x_i ← s/R_ii
end for
```

The technique of matrix decomposition is to turn one matrix into some sort of combination of these "easy" matrices. For example, we might seek a decomposition that transforms the problem into a diagonal system. We say that a matrix is *diagonalizable* if and only if it can be decomposed as

$$A = V\Lambda V^{-1} \tag{3.8}$$

where Λ is a diagonal matrix, and V is non-singular. If such a decomposition can be found, we can find the solution by transforming the problem:

$$Ax = b \tag{3.9}$$
$$V\Lambda V^{-1}x = b \tag{3.10}$$
$$\Lambda V^{-1}x = V^{-1}b. \tag{3.11}$$

So, we can first solve the easy problem

$$\Lambda y = c, \text{ where } c = V^{-1}b, \tag{3.12}$$

so that

$$x = Vy. \tag{3.13}$$

It turns out that a matrix is diagonalizable if and only if the eigenvalue decomposition of A exists. This would mean we can take Λ and V as the eigenvalues and eigenvectors of A, satisfying

$$Av_i = \lambda_i v_i, \tag{3.14}$$

where $\Lambda_{ii} = \lambda_i$, and the i^{th} column of V is v_i. Unfortunately, the only way to get the eigenvalues is find the roots of

$$\det\left(A - \lambda_i\right), \tag{3.15}$$

which is an n^{th} order polynomial in n. It is well-known that no explicit formula exists for finding the roots of such a polynomial if $n > 5$, so unfortunately these decompositions are only useful for small matrices.

If the matrix A is symmetric, then we can imagine its decomposition into

$$A = LL^T, \tag{3.16}$$

where L is a lower triangular matrix (L^T is upper-triangular). This is called the Cholesky decomposition. In this case, the solution is

$$Ax = b \tag{3.17}$$
$$LL^Tx = b. \tag{3.18}$$

Algorithm 2 Cholesky decomposition

for $j = 1..n$ **do**
  $L_{jj} \leftarrow A_{jj}$
  for $k = 1..j-1$ **do**
    $L_{jj} \leftarrow L_{jj} - L_{jk}^2$
  end for
  $L_{jj} \leftarrow L_{jj}^{\frac{1}{2}}$
  for $i = j+1..n$ **do**
    $L_{ij} \leftarrow A_{ij}$
    for $k = 1..j-1$ **do**
      $L_{ij} \leftarrow L_{ij} - L_{ik}L_{jk}$
    end for
    $L_{ij} \leftarrow L_{ij}/L_{jj}$
  end for
end for

By letting $y = L^T x$, we can first solve $Ly = b$ using forward substitution, a variant of Algorithm 1, then solve $L^T x = y$ using Algorithm 1 in $\mathcal{O}\left(n^2\right)$ time. By examining Algorithm 2, we can see the decomposition itself requires j multiplications for element L_{ij}, and summing over all $j \leq i \leq n$ gives total complexity of $\mathcal{O}\left(n^3\right)$.

The QR decomposition uses a combination of a unitary matrix and a triangular matrix:

$$A = QR, \tag{3.19}$$

where Q is unitary and R is upper-triangular. If we have such a decomposition, then it can be solved in the following way.

$$Ax = b \tag{3.20}$$
$$QRx = b \tag{3.21}$$
$$Rx = Q^T b. \tag{3.22}$$

So, we can just solve $Rx = c$ using Algorithm 1 in $\mathcal{O}\left(n^2\right)$ time where $c = Q^T b$.

The QR decomposition can be realized by the Gram-Schmidt procedure in which orthogonal vectors are generated sequentially. With the procedure given in Algorithm 3, we see that each vector q_i needs $2n(i-1)$ multiplications, and summing over all n vectors gives $\mathcal{O}\left(n^3\right)$ as the total complexity. There are other algorithms utilizing Householder reflections [14] to obtain the decomposition that are more numerically stable.

Algorithm 3 Gram-Schmidt QR decomposition

$A \triangleq [a_1, \ldots, a_n]$
$Q \triangleq [q_1, \ldots, q_n]$
$q_1 \leftarrow a_1$
$q_1 \leftarrow q_1 / \|q_1\|$
for $i = 2..n$ **do**
  $q_i \leftarrow a_i$
  for $j = 1..i-1$ **do**
    $q_i \leftarrow q_i - \langle q_j, a_i \rangle q_j$
  end for
  $q_i \leftarrow q_i / \|q_i\|$
end for
$R \leftarrow Q^T A$

A slight modification to the eigenvalue decomposition makes the problem much easier. If we change the decom-

position a bit such that

$$A = U\Sigma V^T \tag{3.23}$$

for some diagonal Σ and unitary U and V, then

$$Ax = b \tag{3.24}$$
$$U\Sigma V^T x = b \tag{3.25}$$
$$\Sigma V^T x = U^T b. \tag{3.26}$$

This is called the *singular value* decomposition (SVD). So, to get the solution, we can apply the same strategy as we do for the eigenvalue decomposition:

$$\Sigma y = c, \text{ where } c = U^T b, \tag{3.27}$$

and then

$$x = Vy. \tag{3.28}$$

The procedure to construct the SVD is too long to include in algorithmic form here, but we state the result that it is $\mathcal{O}\left(n^3\right)$ complexity.

So, in increasing order of complexity, the typical matrix decompositions and solving routines are

1. Diagonal matrices ($\mathcal{O}\left(n\right)$)
2. Unitary matrices ($\mathcal{O}\left(n^2\right)$)
3. Triangular matrices ($\mathcal{O}\left(n^3\right)$)
4. Cholesky, QR, and singular value decompositions ($\mathcal{O}\left(n^3\right)$)
5. Eigenvalue decomposition (unclear, but usually $\mathcal{O}\left(n^3\right)$ in practice).

Since the QR and singular value decompositions apply to arbitrary matrices and can be performed in a numerically-stable manner, this suggests that solving dense matrix equations is fundamentally an $\mathcal{O}\left(n^3\right)$ problem.

This was the conventional wisdom for hundreds of years until [4] proved otherwise. They showed that matrix inversion can be performed stably in $O(n^{\omega+\eta})$ time if *and only if* the matrix multiplication can also be performed stably in $O(n^{\omega+\eta})$ time. For these dense matrices, $\omega = 2$ describes the $\mathcal{O}\left(n^2\right)$ complexity of dense multiplication, and $\eta = 0.376$ is a small extra factor required for stability. The argument comes from their proof that under the assumptions, the QR decomposition is stably computed in $O(n^{\omega+\eta})$ multiplications. Given that decomposition, the inversion as we showed above is simple.

3.2 Gradient Descent

Matrix decomposition is a great technique for solving dense matrix equations. However, we encounter practical problems when the system is sparse. We say a linear system is *sparse* if the number of nonzeros of the system is $\mathcal{O}\left(n\right)$. The reason this creates trouble is because although a compressed representation of such a system may fit comfortably in memory, the decomposition will be a full matrix (in general) and will tend to dominate the memory usage as n becomes large.

When the size of a sparse system becomes large, we require techniques that do not require an explicit decomposition. Rather, we favor techniques that can give succesively better approximations to the solution: that is to say *iterative* techniques. The general approach is to define some function that is smooth and that is minimized at the desired solution. Then, we can apply results from linear and nonlinear optimization to approach the solution.

One of the most natural ways to write such an objective function is

$$f(x) \triangleq \|Ax - b\|_2^2, \tag{3.29}$$

where $\|\cdot\|_2$ is the Euclidean norm. If a unique solution exists, then $f(x)$ is minimized at the point that satisfies $Ax = b$, since $f(x) \geq 0$ for any x by the properties of the norm, and $f(x) = 0$ at the solution.

We can rewrite the objective in terms of inner products:

$$f(x) = \langle x, A^T Ax \rangle - 2\langle x, A^T b \rangle + \langle b, b \rangle. \tag{3.30}$$

The gradient, which represents the direction of greatest instantaneous increase is then

$$\nabla f(x) = 2A^T Ax - 2A^T b. \tag{3.31}$$

Because we want to minimize this function, this suggests the simple scheme

$$x^n \triangleq x^{n-1} - \delta_{n-1} \nabla f(x^{n-1}), \tag{3.32}$$

where $\delta_n > 0$ is some step size that may or may not depend on n. For a given method to choose δ and x^0, we obtain an instantiation of *gradient descent*.

In our particular case for matting by graph Laplacian, A is positive semi-definite by construction, and we can get better stability by changing the objective function to

$$f(x) \triangleq \frac{1}{2}\langle x, Ax \rangle - \langle x, b \rangle. \tag{3.33}$$

Then, the gradient becomes

$$\nabla f(x) = Ax - b, \tag{3.34}$$

which is zero only at the solution. This provides more numerical stability because the quadratic operator is now A instead of $A^T A$, and if the condition number of $A^T A$ is κ^2, then the condition number of A is κ. A natural choice for the step size is one that minimizes x^n minimizes $f(x)$ along the descent direction. This is called *steepest descent*, and provides

$$\delta_n \triangleq \frac{\langle r^n, r^n \rangle}{\langle r^n, Ar^n \rangle}, \text{ where} \tag{3.35}$$

$$r^n \triangleq \nabla f(x^n). \tag{3.36}$$

The algorithm is given in Algorithm 4.

Algorithm 4 Steepest descent

$x \leftarrow x^0$
for $i = 1..K$ **do**
  $r \leftarrow Ax - b$
  $\delta \leftarrow \langle r, r \rangle / \langle r, Ar \rangle$
  $x \leftarrow x - \delta r$
end for

For a more rigorous analysis of the convergence of steepest descent, refer to [28] that the error reduction ratio is bounded by

$$\frac{\left\|e^K\right\|_A}{\|e^0\|_A} \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^K, \tag{3.37}$$

where κ is the condition number of A. If we require that the error reduction ratio

$$\frac{\left\|e^K\right\|_A}{\|e^0\|_A} \leq \epsilon, \tag{3.38}$$

then we can get a lower bound on the number of required iterations K:

$$\left(\frac{\kappa-1}{\kappa+1}\right)^K \leq \epsilon \tag{3.39}$$

$$K\log\left(\frac{\kappa-1}{\kappa+1}\right) \leq \log(\epsilon) \tag{3.40}$$

$$K\log\left(\frac{\kappa+1}{\kappa-1}\right) \geq \log\left(\frac{1}{\epsilon}\right) \tag{3.41}$$

$$K\log\left(\frac{1+1/\kappa}{1-1/\kappa}\right) \geq \log\left(\frac{1}{\epsilon}\right) \tag{3.42}$$

$$2K\left(\frac{1}{\kappa}+\frac{1}{3\kappa^3}+\cdots\right) \geq \log\left(\frac{1}{\epsilon}\right). \tag{3.43}$$

So, we can choose $K \geq 0.5\kappa\log(1/\epsilon)$ to sufficiently reduce the error. Since the number of iterations required is $\mathcal{O}(\kappa)$, and Algorithm 4 requires $\mathcal{O}(n)$ multiplications per iteration if A is sparse, the total complexity for steepest descent on sparse matrices is $\mathcal{O}(n\kappa)$.

3.3 Conjugate Gradient Descent

Conjugate gradient descent (CG) [12] is a modification of gradient descent that attempts to remedy the cycling problem of gradient descent.

Let the notation

$$\langle x, y\rangle_A = \langle x, Ay\rangle, \tag{3.44}$$

for a positive definite matrix A. We say that x and y are *conjugate* with respect to A iff. $\langle x, y\rangle_A = 0$. Suppose that

$$P \triangleq \{p_i \;\forall i \in [1,n] \mid \langle p_i, p_j\rangle_A = 0 \;\forall i \neq j\}. \tag{3.45}$$

That is, P is a complete set of mutually-conjugate vectors. We also write $p_i \perp_A p_j$ to denote conjugacy. Since $A > 0$, P is also a basis for $\mathbb{R}^n$.

Then, we can write

$$\hat{x} = \sum_{i=1}^{n} \alpha_i p_i, \tag{3.46}$$

with each $\alpha_i \in \mathbb{R}$, where $A\hat{x} = b$. Then,

$$b = A\hat{x} = \sum_{i=1}^{n} \alpha_i A p_i. \tag{3.47}$$

By choosing any $p_j \in P$, we have

$$\langle p_j, b\rangle = \sum_{i=1}^{n} \alpha_i \langle p_j, Ap_i\rangle \tag{3.48}$$

$$= \alpha_j \langle p_j, Ap_j\rangle, \tag{3.49}$$

due to the conjugacy of P. We can then recover

$$\alpha_j = \frac{\langle p_j, b\rangle}{\langle p_j, p_j\rangle_A}. \tag{3.50}$$

Algorithm 5 Conjugate gradient descent

$x \leftarrow x^0$
$r \leftarrow b - Ax$
$z \leftarrow r$
$p \leftarrow z$
for $i = 1..K$ **do**
  $\alpha \leftarrow \langle r, z\rangle / \langle p, p\rangle_A$
  $x \leftarrow x + \alpha Ap$
  $r^{\text{new}} \leftarrow r - \alpha Ap$
  $z^{\text{new}} \leftarrow r^{\text{new}}$
  $\beta \leftarrow \langle z^{\text{new}}, r^{\text{new}}\rangle / \langle z, r\rangle$
  $p \leftarrow z^{\text{new}} + \beta p$
  $z \leftarrow z^{\text{new}}$
  $r \leftarrow r^{\text{new}}$
end for

It turns out that we can generate p_i sequentially without significant effort, leading to Algorithm 5. Due to the fact that the iteration steps are along the p_i and that they are mutually conjugate and span the entire space, it is a fact that (excepting numerical errors) the algorithm converges exactly to $\hat{x}$ in at most n iterations.

If A is dense, then each iteration has $\mathcal{O}\left(n^2\right)$ multiplications. Since we have to perform $K = n$ iterations in the worst case, this gives an overall worst-case complexity of $\mathcal{O}\left(n^3\right)$ for dense matrices, and by the same logic $\mathcal{O}\left(n^2\right)$ for sparse systems.

For a more rigorous analysis of the convergence of CG, refer to [28] that the error reduction ratio is bounded by

$$\frac{\left\|e^K\right\|_A}{\left\|e^0\right\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^K, \tag{3.51}$$

where κ is the condition number of A. If we require that the error reduction ratio

$$\frac{\left\|e^K\right\|_A}{\left\|e^0\right\|_A} \leq \epsilon, \tag{3.52}$$

then we can get a lower bound on the number of required iterations K:

$$2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^K \leq \epsilon \tag{3.53}$$

$$K \log\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right) \leq \log\left(\frac{\epsilon}{2}\right) \tag{3.54}$$

$$K \log\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right) \geq \log\left(\frac{2}{\epsilon}\right) \tag{3.55}$$

$$K \log\left(\frac{1+1/\sqrt{\kappa}}{1-1/\sqrt{\kappa}}\right) \geq \log\left(\frac{2}{\epsilon}\right) \tag{3.56}$$

$$2K\left(\frac{1}{\sqrt{\kappa}} + \frac{1}{3\sqrt{\kappa}^3} + \cdots\right) \geq \log\left(\frac{2}{\epsilon}\right). \tag{3.57}$$

So, we can choose $K \geq \sqrt{\kappa}/2\log(2/\epsilon)$ to sufficiently reduce the error. Since the number of iterations required is $\mathcal{O}\left(\sqrt{\kappa}\right)$, the total complexity for CG on sparse matrices is $\mathcal{O}\left(n\sqrt{\kappa}\right)$, a significant improvement from the $\mathcal{O}\left(n\kappa\right)$ of steepest descent.

Table 3.1: Comparison of solvers on the matting problem

Algorithm	Space	Time
Complete Factorization	$\mathcal{O}\left(n^3\right)$	$\mathcal{O}\left(n^3\right)$
Steepest Descent	$\mathcal{O}\left(n\right)$	$\mathcal{O}\left(n\kappa\right) = \mathcal{O}\left(n^2\right)$
CG	$\mathcal{O}\left(n\right)$	$\mathcal{O}\left(n\sqrt{\kappa}\right) = \mathcal{O}\left(n^{1.5}\right)$

3.4 Preconditioning

Preconditioning is a general framework that describes a way to reduce the number of iterations required in iterative schemes by implicitly estimating the inverse of the system. The idea is to find a matrix T to modify the problem such that

$$TAx = Tb. \tag{3.58}$$

If T is chosen such that $\kappa(TA) \ll \kappa(A)$, then iterative schemes applied to this new system will converge faster.

One choice for sparse positive definite matrices is the *incomplete Cholesky factorization*. If we had a Cholesky factorization $A = LL^T$, then we could construct $T = A^{-1}$ and solve the problem in a single iteration. However, the Cholesky factorization is dense and unsuitable to store for large sparse matrices. The incomplete Cholesky factorization attempts to create a sparse matrix K that is similar to the dense factorization L. A simple way to create K is to perform the Cholesky decomposition, but to set K_{ij} to zero if A_{ij} is zero. This procedure is given in Algorithm 6. Then, the preconditioner $T = \left(KK^T\right)^{-1}$ can be applied to reduce the condition number and the required number of iterations. This is especialy popular to use with the conjugate gradient to obtain *preconditioned* conjugate gradient method [15].

Algorithm 6 Incomplete Cholesky decomposition

for $j = 1..n$ **do**
  $K_{jj} \leftarrow A_{jj}$
  for $k = 1..j-1$ **do**
    $K_{jj} \leftarrow K_{jj} - K_{jk}^2$
  end for
  $K_{jj} \leftarrow K_{jj}^{\frac{1}{2}}$
  for $i = j+1..n$ **do**
    if $A_{ij} \neq 0$ **then**
      $K_{ij} \leftarrow A_{ij}$
      for $k = 1..j-1$ **do**
        $K_{ij} \leftarrow K_{ij} - K_{ik}K_{jk}$
      end for
      $K_{ij} \leftarrow K_{ij}/K_{jj}$
    end if
  end for
end for

3.5 Applications to Matting

In the existing literature, the previous solution methods are used in various fashions. Since the iterative methods have complexity dependent on the condition number, we need to estimate that in order to get a clear picture of the practical complexity. Since the matting Laplacian defines a second-order elliptic PDE, our system tends to have a condition number $\kappa \in \mathcal{O}\left(n^{\frac{2}{d}}\right)$, where d is the spatial dimensionality, which is 2 in this case [28]. So, we can substitute n for κ in the complexity analyses to obtain Table 3.1.

Because all the methods summarized in Table 3.1 are not scalable as the resolution n grows, they are usually not used directly in practice. Some of the matting methods modify the previous algorithms directly, some use heuristics to simplify the solution, and some modify or decompose the input first.

In [30], the matting Laplacian is fixed across all instances of the problem to be the spatial Laplace operator. Although there are specialized efficient methods to solve with this operator, the work instead chooses to use successive over-relaxation (SOR), which is a straightforward iterative method that we will describe later. Since their method does not have a fixed right-hand side, they iterate between the SOR stage and recomputing the right-hand side.

In [19], the matting Laplacian is data-dependent. For small images, they say they use the "backslash" operator in MATLAB, which amounts to a Cholesky decomposition in this case. To deal with larger images, they downsample to a small resolution, solve in the same manner, then do interpolation and thresholding to get back to full resolution. They mention briefly that they implemented a multigrid solver to handle large problems, but give no details and state that it degrades the quality.

[11] use the conjugate gradient method. To work on large images, they split the problem into axis-aligned rectangles that contain both foreground and background constraints. They then solve for the matte in each segment independently.

Part II

My Work

Chapter 4

Addressing Quality & User Input

Quality and the amount of user input required to get that quality are two fundamental concerns of natural image matting. Below, we describe our works [16] and [17] that make progress on both fronts.

4.1 L_1 Matting

Many matting problems are formulated as minimization problems, where the objective function contains a term proportional to the energy encoded by the image and the matte. In other words, we often see problems formulated under L_2 as

$$\alpha_\Omega = \arg\min_\alpha \int_\Omega |T[I] - \alpha|^2 \, dx \tag{4.1}$$

where $T[I]$ is some transformation applied to the image, and Ω is small enough such that the matte is expected to be roughly constant there. If there is little interaction between alpha values on different domains, then it is clear that the optimum solution for alpha over each domain is the mean of $T[I]$.

In closed-form matting, the terms of the objective function are

$$\sum_{i \in w_j} |\alpha_i - a_j I_i - b_j|^2 , \tag{4.2}$$

and in poisson matting, the objective function is

$$\alpha_\Omega = \arg\min_\alpha \int_\Omega \left| \frac{1}{F-B} \nabla I - \nabla \alpha \right|^2 dx. \tag{4.3}$$

Such formulations utilizing the L_2 norm have some nice statistical properties and solutions. However, there is little reason to assume that the L_2 norm is the "right" norm. The mean that is associated with this norm also has drawbacks. It can be very sensitive to noise and outliers in some areas, and causes oversmoothing in other areas.

In our approach, we consider the usefulness of the L_1 norm. It turns out to be that the median filtered image $\Psi[I]$ plays a crucial role in L_1. Specifically, it is much smoother over foreground and background regions, but in contrast with the mean intrinsic to L_2 space, it tends to preserve information on the edges between foreground and background. The L_2 interpretation of this L_1 feature is that it forces foreground and background to be smooth. This leads to the intriguing conclusion that it is not always necessary to completely reformulate methods from L_2 to L_1 to utilize the benefits of L_1 space.

To illustrate this idea, we reformulate a the well-known method of closed-form matting. Since the local estimate of the matte depends directly on the image data, there are often patches within the result that exhibit too much fluctuation unless the smoothing parameter ϵ is chosen large enough, in which case the boundaries can become overly smooth. By letting the matte depend on the L_1 data $\Psi[I]$ instead, we improve the quality of the homogeneous regions of the matte without sacrificing as much of the quality along the boundaries. We show direct comparisons between the two methods in the mattes of blurred objects and static objects, and provide numerical evidence that our method generates better mattes.

We posit that matting using the median-filtered image $\Psi[I]$ in place of the image itself generally provides better results. The median filter is a non-linear filter that tends to preserve edges in an image while making the other regions more uniform. Techniques such as closed-form and Poisson matting make the assumption that the foreground and background images are smooth. But for most natural images, there is no reason to assume that they are, since there is no natural phenomenon that causes physical objects to be band-limited. This makes such assumptions troublesome. However, by using $\Psi[I]$ instead, we smooth out the foreground and background in the appropriate places and better satisfy the assumption, without compromising the information carried by edges.

4.1.1 Methods

Before we consider the L_1 norm and its usefulness toward matting, we first need to define the median. Take Ω to be a domain in $\mathbb{R}^n$ with finite Lebesgue measure and $f \in C(\Omega) \cap L_1(\Omega)$ to be a real-valued function on Ω. According to a theorem of [24], there exists a unique $m^* \in \mathbb{R}$ where

$$M(m) := \int_\Omega |f(x) - m| \, d\lambda^n(x) \tag{4.4}$$

is minimized. We call m^* *the* median of f over Ω with respect to λ^n.

The typical minimization problem under L_1 gives us

$$\alpha_\Omega = \arg\min \int_\Omega |T[I] - \alpha| \, dx \tag{4.5}$$

which we now recognize as the *median* of $T[I]$ according to equation (4.4). The median, in contrast with the mean, resolves the issue of noise sensitivity. It also does not cause oversmoothing issues near sharp edges, where most of the perceptual information is located.

Now we explain the concept of L_1 matting in an intuitive sense. For most of the image, α should be nearly constant and either 0 or 1. This means that even though Ψ is non-linear,

$$\Psi[I] \approx \alpha\Psi[F] + (1 - \alpha)\Psi[B] \tag{4.6}$$

except near transition regions. And of course, $\Psi[F]$ and $\Psi[B]$ are smoother than F and B. But near the transition regions, we have

$$\Psi[I] \approx I = \alpha F + (1 - \alpha)B. \tag{4.7}$$

What equations (4.6) and (4.7) mean, is that if we are concerned only with $\Psi[I]$, we eliminate the need for smoothness assumptions on F and B for the large majority of the image. Another interpretation is that we force the smoothness assumption to be true in regions that are clearly foreground or background.

Interestingly, in many cases it seems unnecessary to reformulate the whole problem to L_1 to take advantage of Ψ. As an example, we modify the closed-form approach. Notice that the individual terms of the objective function are of the form

$$\sum_{i \in w_j} |\alpha_i - a_j I_i - b_j|^2 . \tag{4.8}$$

If we use the L_1 norm instead of L_2, we do not lose the meaning of the cost function, since its use is only to make the solution easier to obtain. Taking w_j sufficiently small, $\alpha_i \approx \alpha_j \; \forall i \in w_j$. So if we make these changes,

$$\alpha^* := \arg\min \sum_{i \in w_j} |\alpha - a_j I_i - b_j| \tag{4.9}$$

which is nothing but a discrete version of equation (4.4). In other words, $\alpha^* = \Psi[a_j I_i - b_j]$, or

$$\alpha_i \approx a_j \Psi[I]_i + b_j \; \forall i \in w_j. \tag{4.10}$$

However, care must be taken when considering the radius of Ψ. Any feature with a thickness smaller than the radius will be obliterated. This can be a problem for images with fine structures (such as hair) near transition regions. However, one way around this is to introduce a second operator which preserves these structures. In other words:

$$\alpha_i \approx a_j \Psi_{r_1}[I]_i + b_j \Psi_{r_2}[I]_i + c_j \; \forall i \in w_j, \tag{4.11}$$

where r_1 is the radius of Ψ chosen to be smaller than the width of the finest structures, and r_2 is a radius chosen to sufficiently smooth the interior regions. $S := \Psi_{r_1}[I]$ and $L := \Psi_{r_2}[I]$. The closed-form objective function then becomes:

$$\begin{aligned} &J(\alpha, \mathbf{a}, \mathbf{b}, \mathbf{c}) := \\ &\frac{1}{2} \sum_{j \in I} \left(\sum_{i \in w_j} (\alpha_i - a_j S_i - b_j L_i - c_j)^2 + \epsilon_1 a_j^2 + \epsilon_2 b_j^2 \right). \end{aligned} \tag{4.12}$$

The ratio ϵ_1/ϵ_2 should be chosen high enough to preserve the small features, but not so large as to destroy the smoothness of interior regions, and both should be small enough to prevent global over-smoothing.

Since J is composed of quadratic functions of linear combinations of the variables, it is convex. This structure also means that the gradient components are linear in terms of the variables. For example,

$$\frac{\partial J}{\partial \alpha_k} = \alpha_k |w_k| - S_k \sum_{j \in w_k} a_j - L_k \sum_{j \in w_k} b_j - \sum_{j \in w_k} c_j, \tag{4.13}$$

and in vector notation,

$$\begin{aligned} \frac{\partial J}{\partial \alpha_k} =& [0, \ldots, |w_k|, 0, \ldots, \\ & 0, \ldots, -S_k, -S_k, \ldots, \\ & 0, \ldots, -L_k, -L_k, \ldots, \\ & 0, \ldots, -1, -1, \ldots, 0, \ldots] \mathbf{x}, \end{aligned} \tag{4.14}$$

if $\mathbf{x} := [\alpha, \mathbf{a}, \mathbf{b}, \mathbf{c}]^T$.

So, the unconstrained problem can be solved exactly by a sparse set of linear equations given by

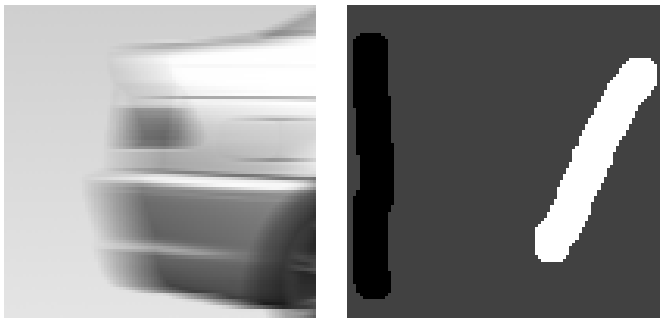
$$\begin{aligned} &\frac{\partial}{\partial x_k} J = \mathbf{A_k} \cdot \mathbf{x} = 0 \qquad \text{if } x_k \text{ unconstrained} \\ &\mathbf{A_k} \cdot \mathbf{x} = \mathbf{e_k} \cdot \mathbf{x} = v_k \quad \text{if } x_k \text{ constrained to be } v_k \end{aligned}$$

where $\mathbf{A_k}$ is the kth row of matrix $\mathbf{A} \in \mathbb{R}^{4n} \times \mathbb{R}^{4n}$.

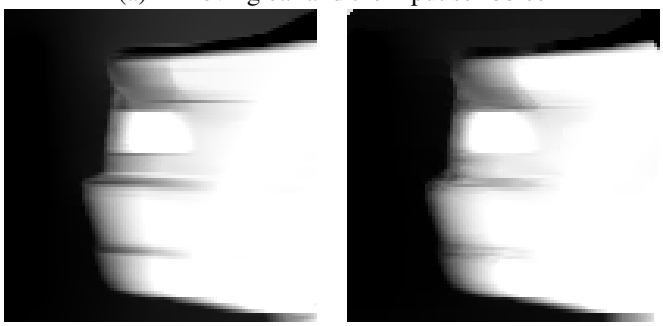
We have implemented this matrix method in Matlab and have been using it when the number of pixels is relatively small. However, notice that the number of non-zero entries is $\mathrm{O}(n)$. Therefore, the time complexity of the algorithm is $\mathrm{O}(n^2)$, which is quite steep for images as the resolution grows.

To get the solution more efficiently, it is worth noting that a well-constructed iterative gradient descent method will terminate *exactly* on the optimal solution in $4n$ iterations or less (excepting round-off errors). Therefore, gradient descent has a much better complexity of $\mathrm{O}(n)$. Each iteration of the descent forms the next estimate $\mathbf{x}^{p+1}$ by $\mathbf{x}^{p+1} = \mathbf{x}^p - \lambda \nabla J(\mathbf{x}^p)$, where λ is the step size that minimizes $J(\mathbf{x}^p - \lambda \nabla J(\mathbf{x}^p))$. Since this is a 1-d minimization over λ, we can find it easily, and the solution is

$$\lambda = \frac{T_1 - T_2}{T_3 - T_4}, \tag{4.15}$$



(a) A moving car and the input scribbles



(b) Closed-form result followed by our result

Figure 4.1: Notice the horizontal stripes begin to fade in our result.

where

$$
\begin{aligned}
T_1 &= \sum_{j \in I} (\epsilon_1 a_j^p \nabla a_j^p + \epsilon_2 b_j^p \nabla b_j^p) \\
T_2 &= \sum_{j \in I} \sum_{i \in w_j} (\alpha_i^p - a_j^p S_i - b_j^p L_i - c_j^p) \\
&\qquad \times (\nabla \alpha_i^p - \nabla a_j^p S_i - \nabla b_j^p L_i - \nabla c_j^p) \\
T_3 &= \sum_{j \in I} (\epsilon_1 (\nabla a_j^p)^2 + \epsilon_2 (\nabla b_j^p)^2) \\
T_4 &= \sum_{j \in I} \sum_{i \in w_j} (\nabla \alpha_i^p - \nabla a_j^p S_i - \nabla b_j^p L_i - \nabla c_j^p)^2
\end{aligned}
$$

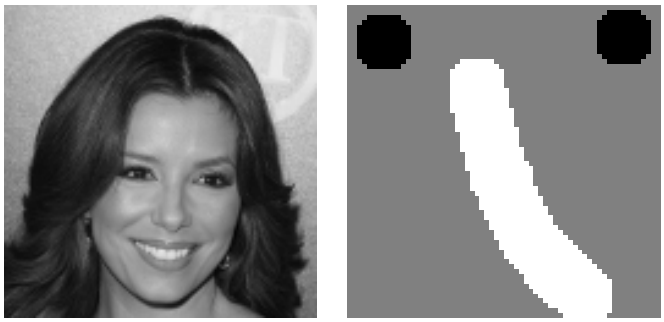
and ∇a_j^p is the component of $\nabla J(\mathbf{x}^p)$ corresponding to a_j, for example.

The effect of our formulation as given by equation (4.12) is like an adaptive median filter. In interior regions, b_j should be larger that a_j, meaning that the large-radius median filter is active. Near transition regions, a_j should be larger than b_j. This suggests another way to formulate the problem.

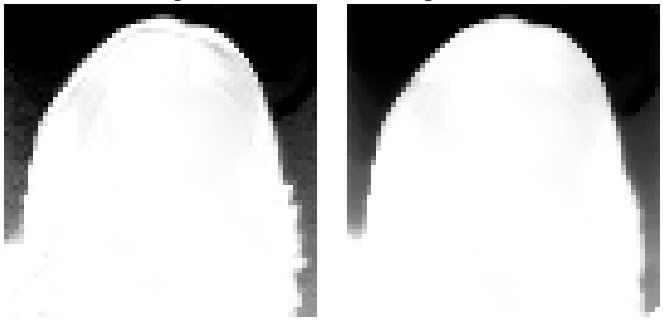
Instead of a fixed-radius Ψ, it should be possible to design an adaptive version which is cognizant of transition regions. Perhaps this could be done using gradient information. This is something we will pursue in the near future.

4.1.2 Experiments & Results

In figure 4.1, we show the tail of a car that is travelling from left to right, leaving a motion blur. Giving the image to the closed-form method produces the image in the lower left. Utilizing median information gives our result bottom right. Notice that not only have the horizontal stripes begun to fade, but the faint glow above the car has attenuated as well. In general, the alpha matte in the foreground and background regions is smoother, without over-smoothing in the transition region. The fact that we can achieve better results with motion-blurred images is quite significant, since



((a) A person's face and the input scribbles



(b) Closed-form result followed by our result

(c) Background replacement

Figure 4.2: Pink hair is remedied.

recovering the unblurred image depends on a high-quality alpha matte in some recent algorithms [3]. There are some artifacts near the edges and corners of the image, but they are only due to Matlab's implementation of the median filter.

In figure 4.2, we show a typical portrait view of a person. The closed-form method result in the lower left corner gives errors near the edges of the hair and where the hair changes color quickly due to highlights, where the assumption of smooth foreground fails. It also produces a bit of noise in the matte in the area between the hair and the left and right edges of the image. This means that some of the hair is assumed to be a bit transparent, which is obviously errant. In our result at the bottom right, the matte is much smoother over the hair, and does not show the same noise as in the closed-form result. As a result, background replacement with closed-form matting makes the highlights in the hair appear pink, while ours stays truer to the original color.

For numerical comparison, we obtained 27 images along with their ground truth alpha mattes from [27]. We then computed the alpha mattes for both methods with constant parameters, and found the absolute error with respect to the ground truth mattes. The parameters were chosen to give the least average error for each method, which meant $\epsilon = 10^{-5}$ for closed-form matting, and $r_1 = 0.5$, $r_2 = 2$, $\epsilon_1 = 10^{-5}$, $\epsilon_2 = 10^{-5}$ for our method. The input given was a sparse trimap, meaning that 80% to 90% of the image is classified as unknown. The result of the experiment is that the average error for our method is 93% of the closed-form error. Since the standard deviation for this figure was only 6.4%, we can say with over 99.99% confidence that using L_1 information is more effective at generating accurate mattes given sparse trimaps. This means that our method could be more amenable to automatic generation of alpha mattes, and we will pursue this in the near future.

4.2 Nonlocal Matting

So far, most papers on natural image matting have ignored one major question: what is good user input? In other words, what is the best input a user can provide so that the algorithm gives the most accurate results? To the best of our knowledge, there is very little in the literature to answer: what is good input, and how can we provide the least amount of input for the most accuracy? We answer these questions in [17] by showing that the nonlocal principle [1] acts as a very effective sparsity prior on the matte, and serves to *dramatically* reduce the human effort required to generate an accurate matte. Using this technique, we can effectively handle textured regions, edges, and structure in a way that no one has demonstrated before.

Suppose there is a subset of pixels that exactly cluster in the graph implied by Laplacian L, and that we constrain one of the pixels in that cluster to have alpha value $\overline{a}$. Then, the value of the objective function $\alpha^T L\alpha$ is minimized if the rest of the pixels in the cluster are also labeled $\overline{a}$. The reason is that, if we have K clusters, $\alpha^T L\alpha$ can be decomposed as

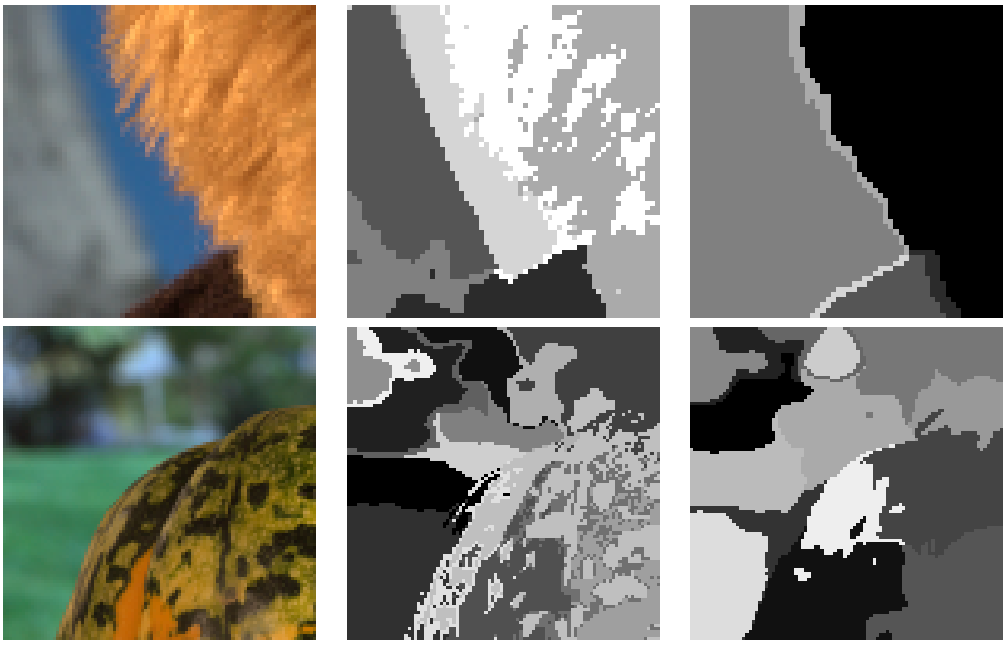
$$\alpha^T L\alpha = \sum_{k=1}^{K} \sum_{(i,j)\in E_k} A_{ij}(\alpha_i - \alpha_j)^2, \tag{4.16}$$

where $\{E_k \mid k = 1, 2, \ldots, K\}$ is a partition of the edge set E. Therefore, labeling the rest of the values in the cluster as $\overline{a}$ minimizes $\alpha^T L\alpha$. What this means is that if we can construct the affinities so that the graph clusters accurately on foreground and background objects, the user only needs to constrain a single pixel in each cluster, and the algorithm can do the rest of the work.

In many cases, obtaining a good clustering is difficult by using Levin et al.'s Laplacian [19]. The main reason is that there is an overly strong regularization of the problem with spatial patches. This causes many clusters to be small and localized (Fig. 4.3.b). This may be fine for large smooth regions, but is inappropriate for textured regions. In an attempt to remedy this, Levin et al. presented another algorithm in [20] that combined the small clusters into larger ones that they call *matting components* based on two priors: the components are heavily biased to be binary-valued, and they do not overlap. Their solution is a nonlinear optimization procedure with a non-convex and non-differentiable objective function.

4.2.1 Methods

In [17], we present a new approach that achieves better results *directly*. This new approach defines a new Laplacian based on the nonlocal principle. The nonlocal principle [1] essentially states that given an image I, the denoised pixel



(a) (b) (c)

Figure 4.3: (a) Input image. (b) Labels after clustering Levin's Laplacian. (c) Clustering nonlocal Laplacian.

i is a weighted sum of the pixels that have similar appearance, where the weights are given by some kernel $k(i,j)$. More formally,

$$E\left[I(i)\right] \approx \sum_j I(j)k(i,j)\frac{1}{D_i}, \text{ where} \tag{4.17}$$

$$k(i,j) \triangleq \exp\left(-\frac{1}{h_1^2}\left\|I(S(i)) - I(S(j))\right\|_g^2 - \frac{1}{h_2^2}d^2(i,j)\right) \tag{4.18}$$

$$D_i \triangleq \sum_j k(i,j). \tag{4.19}$$

Here, $S(i)$ is a small *spatial patch* around pixel i, and $d(i,j)$ is the pixel distance between pixels i and j. $\|\cdot\|_g$ indicates that the norm is weighted by point-spread function (PSF) g, which can be a center-weighted Gaussian PSF with standard deviation on the order of the radius of spatial neighborhood $S(i)$.

The nonlocal principle defines a discrete data distribution $k(i,\cdot)/D_i$ for each pixel i, and implies that the image can be sparsely represented by a few spatial patches. Suppose that, by analogy to Eq. (4.17), the alpha matte is such that

$$E\left[\alpha_i\right] \approx \sum_j \alpha_j k(i,j)\frac{1}{D_i}, \tag{4.20}$$

or, we can write

$$D_i\alpha_i \approx k(i,\cdot)^T\alpha,. \tag{4.21}$$

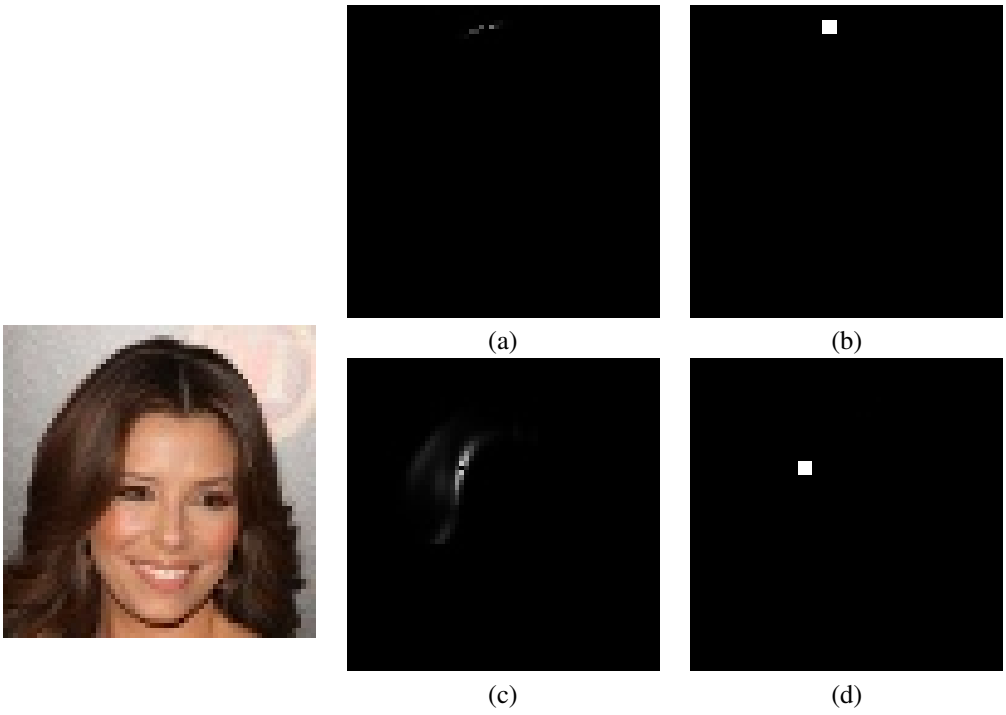


Figure 4.4: (a) Nonlocal neighbors at pixel (5,29). (b) Spatial neighbors at pixel (5,29). (c) Nonlocal neighbors at (24,23). (d) Spatial neighbors at (24,23).

It then follows that

$$D\alpha \approx A\alpha, \text{ where} \tag{4.22}$$

$$A = \begin{bmatrix} k(1,1) \cdots k(1,N) \\ \vdots \\ k(N,1) \cdots k(N,N) \end{bmatrix}, \text{ and} \tag{4.23}$$

$$D = \begin{bmatrix} D_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & D_N \end{bmatrix}. \tag{4.24}$$

Therefore, $(D-A)\alpha \approx 0$, and $\alpha^T(D-A)^T(D-A)\alpha \approx 0$. What this means is that we can effectively have a matting Laplacian based on the nonlocal principle $L_c = (D-A)^T(D-A)$.

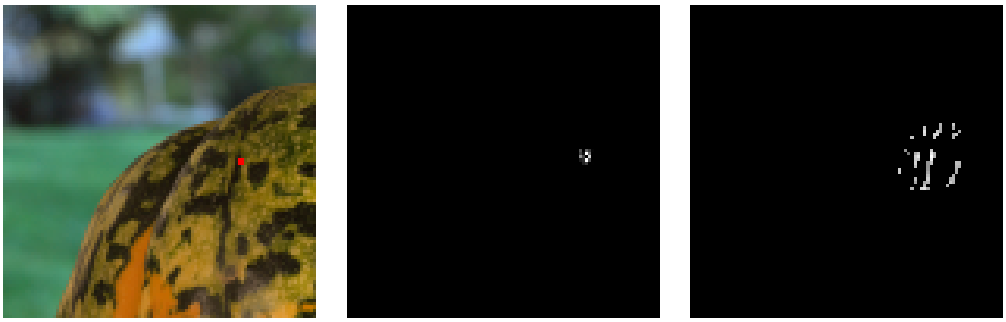
Due to (4.23), our affinity includes texture information, has little spatial regularization, and is nonlinear. Further, since we expect sparsity from the nonlocal principle, L_c should exhibit the property of having a small number of clusters for some permutation matrix P. For this reason, we refer to this L_c as the nonlocal *clustering* Laplacian.

To extract clusters from L_c, we adapt the method described in [23]. First, define

$$M = D^{\frac{-1}{2}} L_c D^{\frac{-1}{2}}. \tag{4.25}$$

Then, calculate the m smallest eigenvectors of M (we take $m = 10$), $\{V_1, \ldots, V_m\}$. Now, we create a matrix U such that each column corresponds to an eigenvector, but normalized such that each row has unit length under the 2-norm. In this way, each pixel is associated with a type of feature vector with m components. Then, we can treat U as the data matrix in the k-means clustering algorithm [9]. So, we use k-means to do the final segmentation to locate the clusters. Parameter k controls the final number of clusters and depends on the image complexity.

The key observation is that the number of clusters k that we need to describe the alpha matte is only on the order of 10 to 20, even when the number of pixels is tens of thousands. According to the observation in (4.16), we only need



(a) (b) (c)

Figure 4.5: (a) Input image, pixel (76,49) highlighted in red. (b) Affinities to pixel (76,49) in [19] (c) Nonlocal affinities to pixel (76,49).

the user to label k pixels in the entire image. This is in contrast to Levin et al.'s method, where their Laplacian does not cluster very well (Fig. 4.3), especially for small k, requiring dense user input in the form of large scribble regions, or even a nearly complete trimap.

To maintain accuracy of the matte near the edges, we present a modification to Levin et al.'s algorithm. First, some notation. If $C = \{\tau_1, \ldots, \tau_n\}$, define $\alpha(C)$ to be a column vector of length n: $[\alpha_{\tau_1}, \ldots, \alpha_{\tau_n}]^T$. Unless otherwise specified, vectors are column vectors. $S(i)$ represents a spatial neighborhood of pixel i (patch), and $N(i)$ represents the nonlocal neighborhood of pixel i.

Suppose

$$\begin{aligned}\mathbf{Q}\alpha(N(i)) &\approx \mathbf{Q}\left[X(N(i)), \mathbf{1}\right]\begin{bmatrix}\boldsymbol{\beta}\\ \beta_0\end{bmatrix} \\ &\triangleq X_0(N(i))\begin{bmatrix}\boldsymbol{\beta}\\ \beta_0\end{bmatrix},\end{aligned} \tag{4.26}$$

where X is a $N \times d$ vector of image features with N being the number of pixels, and d being the number of features or channels (we use RGB or grayscale values), and $\mathbf{Q} = \text{diag}(k(i,\cdot)/D_i)$, is a weighting matrix containing the nonlocal data distribution. If we assume α is fixed, the optimal linear parameters are

$$\begin{aligned}\begin{bmatrix}\boldsymbol{\beta}^*\\ \beta_0^*\end{bmatrix} &= \left(X_0^T(N(i))\mathbf{Q}X_0(N(i)) + \lambda\begin{bmatrix}\mathbf{I}_d & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\right)^{-1} \\ &\quad X_0^T(N(i))\mathbf{Q}\alpha(N(i)) \\ &\triangleq X_0^\dagger(i,\mathbf{Q},\lambda)\alpha(N(i)).\end{aligned} \tag{4.27}$$

Here, λ is a small number (10^{-5} or so) whose purpose is to ensure that the eigenvalues of $X^T(N(i))\mathbf{Q}X(N(i))$ are strictly positive so that the pseudoinverse defined in equation (4.27) exists and is unique.

Since we now have

$$\alpha(N(i)) \approx X_0(N(i))X_0^\dagger(i,\mathbf{Q},\lambda)\alpha(N(i)) \tag{4.28}$$

$$\triangleq \mathbf{B}_i\alpha(N(i)), \tag{4.29}$$

we can construct the quadratic form

$$\alpha^T L_m \alpha \triangleq q(\alpha) \tag{4.30}$$

$$= \sum_{i=1}^{N} \alpha(N(i))^T(\mathbf{I}-\mathbf{B}_i)^T(\mathbf{I}-\mathbf{B}_i)\alpha(N(i)). \tag{4.31}$$

The difference from [19] is that their data matrices $\mathbf{X}_0$ are local, and weighted by a uniform data matrix $\mathbf{Q} = c\mathbf{I}$, while our data matrices are nonlocal, and appropriately weighted by the nonlocal kernel, biasing the solution toward our assumption (4.20), and inserting texture information into the algorithm.

To find α, we solve the following optimization:

$$\begin{aligned} \alpha^* = \arg\min_{\alpha} \quad & q(\alpha) \\ \text{s.t.} \quad & \alpha(C) = \mathbf{k} \end{aligned}$$

where C is the set of user-constrained pixels and $\mathbf{k}$ is the vector of constrained values. This can be solved exactly and in closed form by a smaller quadratic program. Since we use this Laplacian L_m for matte extraction, we refer to it as the nonlocal matting *extraction* Laplacian.

The result of this nonlocal modification is very important. First, by exploiting the nonlocal kernel, we utilize some texture information on the image, effectively grouping pixels with similar texture. Second, the neighborhoods used in calculating the Laplacian are essentially adaptively varying in size as shown in Fig. 4.4, removing the highly spatial clustering effects of previous methods (see Fig. 4.3.b), and therefore reducing the amount of input needed. Another advantage is that the nonlocal neighborhoods can safely be much larger. The effect of large neighborhoods is interpreted by [11] as a large point spread function on the matte, causing convergence of iterative solvers to greatly speed up. Conversely, if the pixel's patch is not very similar to any other in the image, the nonlocal neighborhood will be very small and convergence there will be slow. Therefore, an iterative solver can provide feedback to the user based on the local convergence rates about the quality of the final matte, and where he or she should provide additional input.

4.2.2 Experiments & Results

To formally evaluate the algorithm we have developed, we test its accuracy on the dataset given by [27] by the mean squared error (MSE) as well as by visual inspection of matte quality. We use the algorithm in [19] for comparison, since it is the state of the art with respect to sparse user input. We provide it with the same user input as to our algorithm.

The results for the MSE are shown in Fig. 4.6, where we can immediately see a drastic reduction in MSE. In fact, the average ratio of the Levin's MSE to the nonlocal MSE is 4.04 with a sample standard deviation of 1.0. Therefore, we can say that our method outperforms Levin's with confidence $> 99.99\%$ when the input is sparse. Perhaps even more important is the observation that our MSE curve is much smoother, suggesting our method is more robust to variations in user input.

However, the MSE does not have a linear correlation to human perception. So, we need to visually verify that the extracted mattes are high quality despite the sparse input. For this, we provide several examples in Fig. 4.7. First, it is obvious that the amount of input required for our method is miniscule. The effects of spatial regularization can be seen as oversmoothing in column (c), while our results in (d) remain crisp. In column (e), background replacement highlights the limitations of [19], showing significant bleed-through, while ours in column (f) appears much cleaner. It is clear that even with such sparse input, the matte quality is much higher with our nonlocal matting.

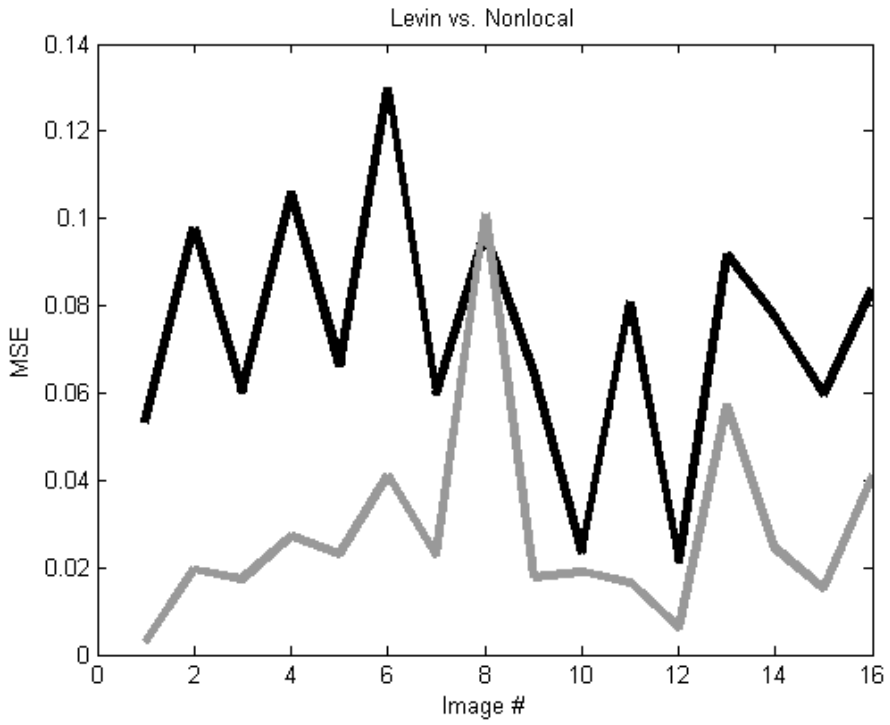


Figure 4.6: MSE of Levin's method vs. our method with the same amount of input on images from [27]. [19] = black, nonlocal = gray.

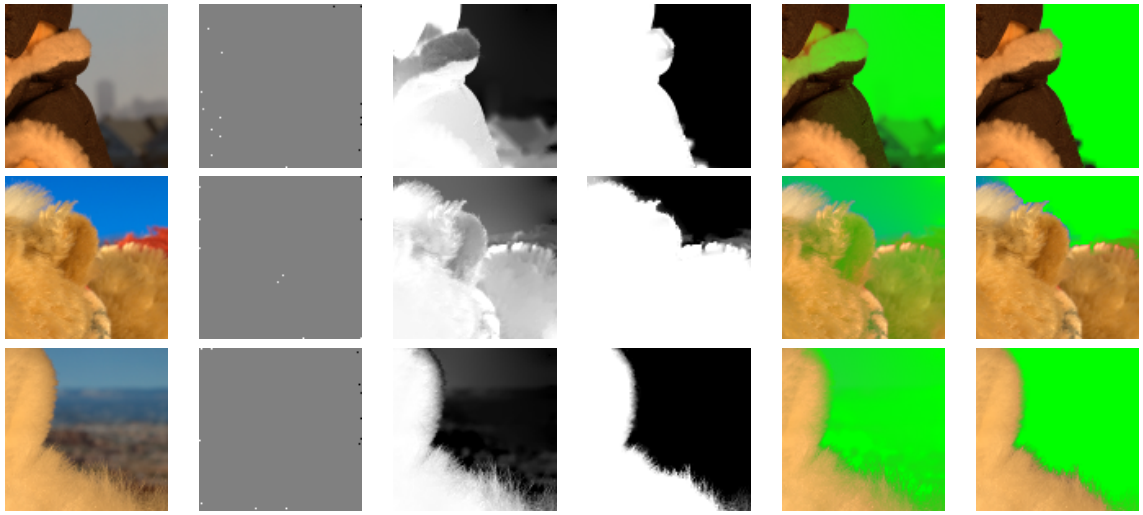


Figure 4.7: Visual inspection of matte quality. (a) Input image. (b) Sparse user input. (c) Levin's algorithm [19]. (d) Nonlocal algorithm. (e) Background replacement with (c). (f) Background replacement with (d).

Chapter 5

Multigrid Analysis

No matter what method we choose to construct the matting Laplacian, we end up with a large, sparse, ill-conditioned problem. There are a number of methods to solve large linear systems $Au = f$ for u. We have already covered several of these in chapter 3, finding that the best of these generally give $\mathcal{O}\left(n^{1.5}\right)$ time complexity on the matting problem. However, they are very generic solution methods and do not take advantage of several key features of our problem.

The first advantage we have is that u and f are images. Further, A must be a linear operator on the space of those images. This means that the problem has a sense of *resolution* built into it. Resolution allows us to get approximate answers to the full scale problem by downsampling the entire system, solving it quickly in small scale (since smaller n means smaller complexity), and then upsampling to full scale.

Secondly, we have good priors on u. For us, u is the alpha matte image, and in most cases we are matting opaque objects with little transparency. This means that u is nearly constant (0 or 1) everywhere, with fractional values near the object boundary. Further, unless the object is pathalogical like a fractal, the ratio of the number of boundary pixels to n goes to zero like $n^{-0.5}$ with increasing resolution. We know that if u is locally constant, that Au is locally minimized since A is a graph Laplacian in which constant vectors are in the null space. Since we expect u to be nearly constant everywhere as n grows, there *must* be a solver which can take advantage of this fact to converge much more quickly than those generic solvers.

Multigrid methods are designed to fully take advantage of both of these expectations. The general idea is to generate a set of succesively coarser resolution systems, and to solve for the low frequencies in small resolution, then upsample and iterate to solve for the high frequencies in high resolution, and to do so in a recursive manner. In this chapter, we review these methods and analysis briefly, and A full treatment is given in [22]. As we studied in [18], we will see that such methods are quite ideal in our circumstance and gives a ***sub***-linear complexity solver for matting.

5.1 Relaxation

First, we must discuss relaxation iterations. These are iterative linear solution methods, where the iteration is of the affine fixed-point variety:

$$u \leftarrow Ru + g. \tag{5.1}$$

Standard relaxtion methods include the Jacobi method, Gauss-Seidel, and successive over-relaxation (SOR). If $\hat{u}$ is the solution, it must be the case that it is a fixed point of the iteration: $\hat{u} = R\hat{u} + g$. For analysis, we define the error and residual vectors

$$e \triangleq \hat{u} - u \text{ and}$$
$$r \triangleq f - Au$$

so that original problem $Au = f$ is transformed to $Ae = r$. We can then see that

$$e \leftarrow Re \text{ i.e.}$$
$$e^{\text{new}} = Re \tag{5.2}$$

is the iteration with respect to the error. We can see that at the solution, $u = \hat{u}$ means that the error $e = 0$. It is important to realize that while r is the *observable* residual, e is *unobservable* in practice, obscured by requiring A^{-1}.

Since we are concerned with how quickly the error converges to 0, it would be nice to say how much the error is reduced in each iteration. The ratio $\|e^{\text{new}}\| \,/\, \|e\|$ captures the per-iteration error reduction and is what we call the *convergence rate*.

From Equation 5.2, we can see that the convergence rate $\|e^{\text{new}}\| \,/\, \|e\|$ is bounded from above by the spectral radius $\rho(R)$, since for any consist norm we have $\|Re\| \leq \|R\| \, \|e\|$, and $\rho(R) \leq \|R\|$. So, a simple upper bound for the error reduction after j iterations is $\left\|e^j\right\| \,/\, \left\|e^1\right\| \leq \rho(R)^j$. For quickest convergence and smallest error, we obviously want $\rho(R)$ to be much less than 1.

Recall $\rho(R) = \max_i \left|\mu_i\right|$, where μ_i are the eigenvalues of R. So, to analyze the convergence, we need to understand the eigenvalues. Since e can be quite a complicated function in practice, it is useful to decompose the error e in the basis of eigenvectors v_i of R:

$$e^1 = \sum_{i=1}^{n} a_i v_i. \tag{5.3}$$

In other words, a_i is the amount of spectral energy that e has at "frequency" i. We can think of it as frequency since the underlying graph is regular in structure, giving Eigenvectors that are increasingly oscillatory as i increases. This, this decomposition is very much a data-dependent version of the Fourier transform. Then, after j iterations, we can see that

$$e^j = \sum_{i=1}^{n} \mu_i^j a_i v_i. \tag{5.4}$$

So, the closer μ_i is to zero, the faster the v_i component of e converges to 0. For Laplace operators, $\rho(R) = \mu_1$, the lowest "frequency" Eigenvalue, and $\mu_1 < 1$ making the iteration always convergent.

Since μ_1 is closest to 1, the v_1 (low-frequency) component dominates the convergence rate, while the higher-frequency components are more easily damped out (for this reason, relaxation is often called *smoothing*). Further, we explain that the convergence rate rapidly approaches 1 as n grows. To break this barrier, we need a technique that can handle the low-frequencies just as well as the high frequencies.

5.2 Jacobi Relaxation

A first attempt to solve $Au = f$ is just to satisfy each equation independently:

$$u_{i(\text{new})} \leftarrow f_i - A_i u, \tag{5.5}$$

where A_i is the i^{th} row of A. This is the strategy of the **Jacobi Method**. More completely, if D, $-L$, and $-U$ are the diagonal, lower, and upper triangular parts of A resp., then $A = D - L - U$ and the Jacobi Method is:

$$u \leftarrow D^{-1}(L + U)u + D^{-1}f. \tag{5.6}$$

We can see that the relaxation matrix $R_J = D^{-1}(L + U)$. Since D is diagonal, it is trivial to invert and can be done once and amortized over all the iterations. Further, R_J can be constructed in-place from A, and the only additional storage required is a temporary vector $u_{(\text{new})}$. Also advantageous is the fact that each element of u is updated independently of all others, allowing trivial parallelization of the method. The only remaining question is how fast it converges.

In our case, $R_J = D^{-1}(L+U)$ is the normalized graph adjacency matrix $D^{-1}(A^G)$, also called the *random walk* Laplacian. From well-known facts, its spectrum gives $\rho(R_J) \approx 1 - c/n^2$ for some constant c. In order for the error to be reduced to some tolerance τ,

$$\left\|e^j\right\| \,/\, \left\|e^0\right\| \leq \tau \implies \tag{5.7}$$

$$(1 - c/n^2) \leq \tau \implies \tag{5.8}$$

$$j \approx n^2 \frac{-\log(\tau)}{c} \in \mathcal{O}\left(n^2\right). \tag{5.9}$$

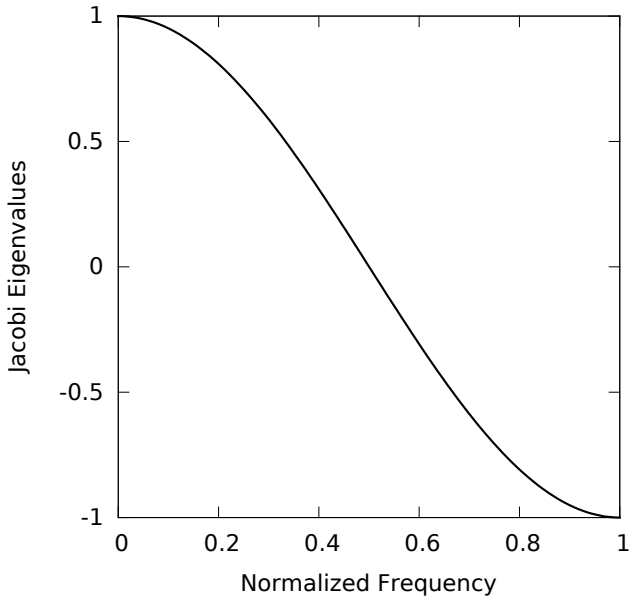


Figure 5.1: Jacobi method spectrum.

In fact, for typical Laplace operators on regular grids, common analysis gives

$$\mu_i \approx \cos\left(\frac{\pi i}{n+1}\right), \tag{5.10}$$

which plotted against normalized frequency $(\pi i)/(n+1)$ gives Fig. 5.1. We can see that while the medium frequencies converge quickly, the low- and high-frequency errors converge very slowly.

5.3 Gauss-Seidel Relaxation

Gauss-Seidel is another relaxation method very similar to the Jacobi method. Instead of independently solving each equation simultaneously, it attempts to do so *sequentially*. For example, it will update u_1, then update u_2 using the new value of u_1 it just computed. Unlike the Jacobi method, this means that it can be performed entirely in-place.

The Gauss-Seidel iteration can be described by

$$u \leftarrow (D-L)^{-1}Uu + (D-L)^{-1}f, \tag{5.11}$$

so, the iteration matrix in this case is

$$R_G = (D-L)^{-1}U. \tag{5.12}$$

Though we will not prove it here, it results that $\rho(R_G) \leq [\rho(R_J)]^2$, meaning the convergence rate is half of the Jacobi method, converging much more rapidly. In fact, the spectrum of R_G typically gives $\mu_i^h \approx \cos^2(i\pi h)$, which is plotted in Figure 5.2. However, the cost of the increased performance is the loss of the ability to parallelize the scheme.

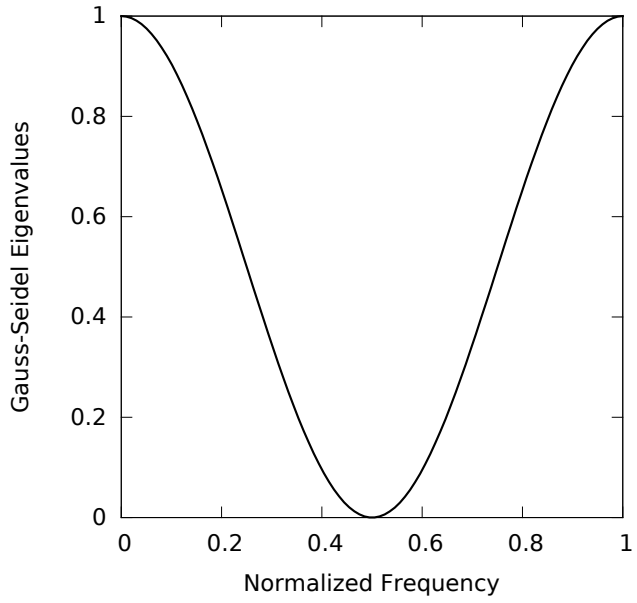


Figure 5.2: Gauss-Seidel spectrum.

5.4 Damping

One problem with the methods above is that the converge just as slowly at high frequencies as they do at low frequencies. This can be remediated by introducing damping.

$$R_J(\omega) \triangleq (1-\omega)I + \omega R_J, \text{ giving} \tag{5.13}$$

$$\mu(\omega) = 1 - \omega + \omega \cos\left(\frac{\pi i}{n+1}\right), \tag{5.14}$$

If we choose $\omega = 2/3$, then that reduces the magnitude of the mid and high-frequency eigenvalues to $1/3$, leaving the low-frequency eigenvalues relatively unchanged. The resulting eigenvalues are plotted in Figure 5.3.

Damping can also be applied to Gauss-Seidel. In that case, the iteration remains convergent for $0 \leq \omega < 2$, and for $\omega > 1$, it is called *successive over-relaxation* or SOR. Damped relaxation gives us the ability to focus on how to improve the convergence rate for the low-frequency error components.

5.5 Multigrid Methods

Multigrid methods are a class of solvers that attempt to fix the problem of slow low-frequency convergence by integrating a technique called nested iteration. The basic idea is that, if we can downsample the system, then the low-frequency modes in high-resolution will become high-frequency modes in low-resolution, which can be solved with relaxation.

Without loss of generality, we can assume for simplicity that the image is square, so that there are N pixels along each axis, $n = N^2$ total pixels, with $h = 1/(N-1)$ being the spacing between the pixels, corresponding to a continuous image whose domain is $[0,1]^2$. Since we will talk about different resolutions, we will let $n_h = (1/h+1)^2$ denote the total number of pixels when the resolution is h. We will also assume for simplicity that the spacings h differ by powers of 2, and therefore the number of pixels differ by powers of 4. The downsample operators are linear and denoted by $I_h^{2h} : \mathbb{R}^{n_h} \to \mathbb{R}^{n_{2h}}$, and the corresponding upsample operators are denoted by $I_{2h}^h = (I_h^{2h})^T$. The choice of these transfer operators is application-specific, and we will discuss that shortly.

The idea of nested iteration is to consider solutions to

$$I_h^{2h} A u^h = I_h^{2h} f^h, \tag{5.15}$$

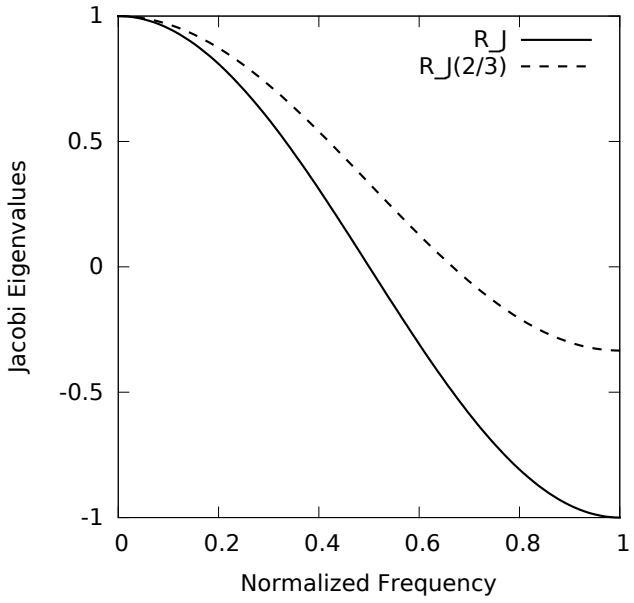


Figure 5.3: Damped Jacobi spectrum.

that is to say a high-resolution solution u^h that solves the downsampled problem exactly. Such a system is over-determined, so we can restrict our search to coarse-grid interpolants $u^h = I_{2h}^h u^{2h}$. So, we get

$$I_h^{2h} A I_{2h}^h u^{2h} = I_h^{2h} f^h. \tag{5.16}$$

Since $I_{2h}^h = (I_h^{2h})^T$, we can see that $A^{2h} \triangleq I_h^{2h} A I_{2h}^h$ is the equivalent coarse system. Of course we can repeat this to get A^{4h} and so on. The advantage of a smaller system is two-fold: first, the system is smaller and therefore is more efficient to iterate on, and second, the smaller system will converge faster due to better conditioning.

So, supposing we can exactly solve, say, u^{16h}, then we can get an initial approximation $u_0^h = I_{2h}^h I_{4h}^{2h} \dots I_{16h}^{8h} u^{16h}$. This is the simple approach taken by many pyramid schemes, but we can do much better. The strategy is good if we have no initial guess, but how can this be useful if we are already given u_0^h? The modification is subtle and important: iterate on the error e instead of the solution variable u.

Algorithm 7 Nested Iteration

1: $r^h \leftarrow f^h - A^h u^h$
2: $r^{2h} \leftarrow I_h^{2h} r^h$
3: Estimate $A^{2h} e^{2h} = r^{2h}$ for e^{2h} using $e_0^{2h} = 0$.
4: $u^h \leftarrow u^h + I_{2h}^h e^{2h}$

Of course, step 3 suggests a recursive application of Alg. 7. Notice that if the error is smooth ($e^h \approx I_{2h}^h e^{2h}$), then this works very well since little is lost by solving on the coarse grid. But, if the error is not smooth, then we make no progress. In this respect, nested iteration and relaxation are complementary, so it makes a lot of sense to combine both. This is called *v-cycle*, and is given in Alg. 8. A visualization of the algorithm is shown in Fig. 5.4.

The idea of multigrid is simple: incorporating coarse low-resolution solutions to the full-resolution scale. The insight is that the low-resolution parts propagate information quickly, converge quickly, and are cheap to compute due to reduced system size.

When the problem has some underlying geometry, multigrid methods can provide superior performance. It turns out that this is not just heuristic, but also has nice analytic properties. On regular grids, v-cycle is provably optimal, and converges in a constant number of iterations with respect to the resolution [8].

Algorithm 8 V-Cycle

function VCYCLE(u^h,f^h)
  if $h = H$ **then return** *exact* solution u^h
  end if
  Relax on $A^h\mathbf{u}^h = f^h$
  $r^h \leftarrow f^h - A^h u^h$ (compute residual)
  $e^{2h} \leftarrow$ VCYCLE($e^{2h} = 0$,$I_h^{2h} r^h$) (estimate correction)
  $u^h \leftarrow u^h + I_{2h}^h e^{2h}$. (apply correction)
  Relax on $A^h\mathbf{u}^h = f^h$
  return u^h
end function

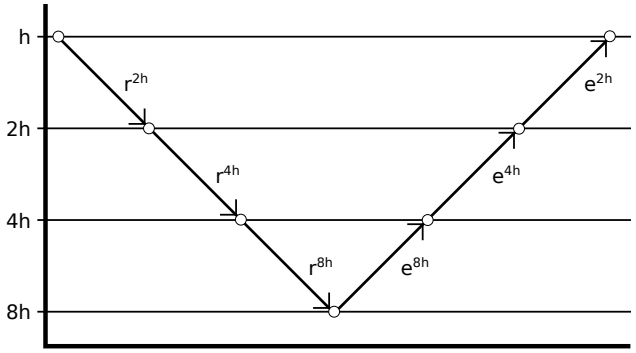


Figure 5.4: Visualization of the v-cycle schedule. The V shape is the reason for the name.

It is also important to see that the amount of work done per iteration is $\mathcal{O}(n)$ in the number of variables. To see this, we can take relaxation on the highest resolution to be 1 unit of work. Looking at Alg. 8, there is one relaxation call, followed by recursion, followed by another relaxation call. In our case, the recursion operates on a problem at $1/4$ size, so we have the total work as $W(n) = 1 + W(\frac{n}{4}) + 1 = 2 + W(\frac{n}{4})$. In general, the relaxation costs n_h/N, and the expression becomes $W(n_h) = 2\frac{n_h}{n} + W(\frac{n_h}{4})$. If we continue the expression, $W(n) = 2 + 2\frac{1}{4} + 2\frac{1}{16} + \cdots = 2\sum_{i=0}^{\infty} \frac{1}{4}^n$, which converges to $W(n) = 8/3$. In other words, the work done for v-cycle is just a constant factor of the work done on a single iteration of the relaxation at full resolution. Since the relaxation methods chosen are usually $\mathcal{O}(n)$, then a single iteration of v-cycle is also $\mathcal{O}(n)$. So, we get a lot of performance without sacrificing complexity.

V-cycle (and the similar w-cycle) multigrid methods work well when the problem has a regular geometry that allows construction of simple transfer operators. When the geometry becomes irregular, the transfer operators must also adapt to preserve the performance benefits. When the geometry is completely unknown or not present, algebraic multigrid algorithms can be useful, which attempt to automatically construct problem-specific operators, giving a kind of "black box" solver.

5.6 Multigrid (Conjugate) Gradient Descent

Here, we describe a relatively new type of multigrid solver from [26] that extends multigrid methods to gradient and conjugate gradient descent. Please note that our notation changes here to match that of [26] and to generalize the notion of resolution.

If the solution space has a notion of resolution on ℓ total levels, with downsample operators $I_\ell^i : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_i}$, and upsample operators $I_i^\ell : \mathbb{R}^{n_i} \to \mathbb{R}^{n_\ell}$ ($n_i < n_\ell \; \forall i < \ell$), then define the multi-level residuum space on x^k at the k^{th}

iteration as

$$\mathcal{R}^k = \text{span}\left\{r_\ell^k, \ldots, r_1^k\right\}, \text{ where} \tag{5.17}$$

$$r_i^k = I_i^\ell I_\ell^i r_\ell^k, \ \forall i < \ell \text{ and} \tag{5.18}$$

$$r_\ell^k = f - Au^k, \tag{5.19}$$

where r_ℓ^k is the residual at the finest level at iteration k, and r_i^k is the projection of the fine residual to the coarser levels. This residuum space changes at each iteration k. Gradient descent in this multi-level residuum space would be to find correction direction $d \in \mathcal{R}^k$ s.t. $\langle u^k + d, q\rangle_A = f^T q \ \forall q \in \mathcal{R}^k$. In other words, the correction direction is a linear combination of low-resolution residual vectors at iteration k, forcing the low-resolution components of the solution to appear quickly.

To find step direction d efficiently, one needs to orthogonalize $\mathcal{R}^k$ by the Gram-Schmidt procedure:

$$\hat{d}_i = r_i^k - \sum_{j=1}^{i-1} \langle r_i^k, d_j\rangle_A d_j, \ \forall i \in 1, \ldots, \ell \tag{5.20}$$

$$d_i = \frac{\hat{d}_i}{\left\|\hat{d}_i\right\|_A}, \tag{5.21}$$

$$d = \sum_{i=1}^{\ell} \langle r_\ell^k, d_i\rangle d_i. \tag{5.22}$$

These equations give an $\mathcal{O}\left(n_\ell \times \ell\right)$ algorithm if implemented naïvely. But, since all of the inner products can be performed at their respective downsampled resolutions, the efficiency can be improved to $\mathcal{O}\left(n_\ell\right)$. To get an idea how this can be done, consider computing the inner product $\langle d_i, d_i\rangle_A$. First see that [26] guarantees the existence of the low-resolution correction direction vector $d_i^{\text{co}} \in \mathbb{R}^{n_i}$ such that it can reconstruct the full-resolution direction by upsampling, i.e. $d_i = I_i^\ell d_i^{\text{co}}$. Now, notice that

$$\begin{aligned}
\langle d_i, d_i\rangle_A &= d_i^T A d_i \\
&= (I_i^\ell d_i^{\text{co}})^T A (I_i^\ell d_i^{\text{co}}) && (5.23)\\
&= (d_i^{\text{co}})^T (I_\ell^i A I_i^\ell) d_i^{\text{co}} && (5.24)\\
&= (d_i^{\text{co}})^T A_i d_i^{\text{co}}, && (5.25)
\end{aligned}$$

Where A_i is a lower-resolution system. Since A_i is only $n_i \times n_i$, and $n_i \ll n_\ell$ if $i < \ell$, then working directly on the downsampled space allows us to perform such inner products much more efficiently.

Standard gradient descent is simple to implement, but it tends to display poor convergence properties. This is because there is no relationship between the correction spaces in each iteration, which may be largely redundant, causing unnecessary work. This problem is fixed with conjugate gradient (CG) descent.

In standard conjugate gradient descent, the idea is that the correction direction for iteration k is A-orthogonal (conjugate) to the previous correction directions, i.e.

$$\mathcal{D}^k \perp_A \mathcal{D}^{k-1}, \tag{5.26}$$

where $\mathcal{D}^k$ is iteration k's correction space analagous to $\mathcal{R}^k$. This leads to faster convergence than gradient descent, due to never performing corrections that interfere with each other.

The conjugate gradient approach can also be adapted to the residuum space. It would require that $d \in \mathcal{D}^k$, where $\text{span}\{\mathcal{D}^k, \mathcal{D}^{k-1}\} = \text{span}\{\mathcal{R}^k, \mathcal{D}^{k-1}\}$ and $\mathcal{D}^k \perp_A \mathcal{D}^{k-1}$. However, as shown by [26], the resulting orthogonalization procedure would cause the algorithm complexity to be $\mathcal{O}\left(n_\ell \times \ell\right)$, even when performing the multiplications in the downsampled resolution spaces. By slightly relaxing the orthogonality condition, [26] develops a multigrid conjugate

gradient (MGCG) algorithm with complexity $\mathcal{O}(n_\ell)$.

$$\mathcal{D}^k = \text{span}\left\{d_1^k, \ldots, d_l^k\right\} \tag{5.27}$$

$$\left\|d_i^k\right\|_A = 1 \; \forall i \in 1, \ldots, \ell \tag{5.28}$$

$$d_i^k \perp_A d_j^k \; \forall i \neq j \tag{5.29}$$

$$\mathcal{D}^1 = \mathcal{R}^1 \tag{5.30}$$

$$d_i^{k-1} \perp_A d_j^k \; \forall k > 1, i \leq j \tag{5.31}$$

$$\mathcal{R}^k + \mathcal{D}^{k-1} = \mathcal{D}^k + \mathcal{D}^{k-1} \tag{5.32}$$

We present Alg. 10, which fixes some algorithmic errors made in [26].

Algorithm 9 Multigrid gradient descent.

while $numIter > 0$ **do**
  $numIter \leftarrow numIter - 1$
  $r_\ell \leftarrow b - Ax$
  $d_\ell \leftarrow r_\ell$
  for $i = \ell - 1..1$ **do**
    $r_i \leftarrow I_{i+1}^i r_{i+1}$
    $d_i \leftarrow r_i$
  end for
  for $i = 1..\ell$ **do**
    $k_i \leftarrow A_i d_i$
    for $j = i..2$ **do**
      $k_{j-1} \leftarrow I_j^{j-1} k_j$
    end for
    $s_1 = 0$
    for $j = 1..i-1$ **do**
      $s_j \leftarrow s_j + (k_j^T d_j) d_j$
      $s_{j+1} \leftarrow I_j^{j+1} s_j$
    end for
    $d_i \leftarrow d_i - s_i$
    $k_i \leftarrow A_i d_i$
    $d_i \leftarrow d_i (k_i^T d_i)^{-\frac{1}{2}}$
  end for
  $s_1 \leftarrow (d_1^T r_1) d_1$
  for $i = 2..\ell$ **do**
    $s_i \leftarrow I_{i-1}^i s_{i-1}$
    $s_i \leftarrow s_i + d_i$
  end for
  $x \leftarrow x + s_\ell$
end while

In multigrid algorithms such as these, we have freedom to construct the transfer operators. If we want to take full advantage of such algorithms, we must take into account our prior knowledge of the solution when designing the operators.

Algorithm 10 Multigrid CG descent.

Do one iteration of multigrid gradient descent.
while $numIter > 0$ **do**
  $numIter \leftarrow numIter - 1$
  $r_\ell \leftarrow b - Ax$
  $d_\ell^{\text{new}} \leftarrow r_\ell$
  for $i = \ell - 1..1$ **do**
    $r_i \leftarrow I_{i+1}^i r_{i+1}$
    $d_i^{\text{new}} \leftarrow r_i$
  end for
  for $i = 1..\ell$ **do**
    $k_i \leftarrow A_i d_i$
    for $j = i..2$ **do**
      $k_{j-1} \leftarrow I_j^{j-1} k_j$
    end for
    $s_1 = 0$
    for $j = 1..i-1$ **do**
      $s_j \leftarrow s_j + (k_j^T d_j^{\text{new}}) d_j^{\text{new}}$
      $s_{j+1} \leftarrow I_j^{j+1} s_j$
    end for
    $d_i \leftarrow d_i - s_i$
    $k_i \leftarrow A_i d_i$
    $d_i \leftarrow d_i (k_i^T d_i)^{-\frac{1}{2}}$
    $k_i \leftarrow A_i d_i^{\text{new}}$
    for $j = i..2$ **do**
      $k_{j-1} \leftarrow I_j^{j-1} k_j$
    end for

$s_1 \leftarrow 0$
for $j = 1..i-1$ **do**
$s_j \leftarrow s_j + d_j^{\text{new}}(k_j^T d_j^{\text{new}})$
$s_j \leftarrow s_j + d_j(k_j^T d_j^{\text{new}})$
$s_{j+1} \leftarrow I_j^{j+1} s_j$
end for
$d_i^{\text{new}} \leftarrow d_i^{\text{new}} - s_i$
$s_i \leftarrow d_i^{\text{new}} - d_i(k_i^T d_i)$
if $\|s_i\|_\infty > \epsilon \|d_i^{\text{new}}\|_\infty$ **then**
$d_i^{\text{new}} \leftarrow s_i$
else
$d_i^{\text{new}} \leftarrow d_i$
$d_i \leftarrow 0$
end if
$k_i \leftarrow A_i d_i^{\text{new}}$
$d_i^{\text{new}} \leftarrow d_i^{\text{new}}(k_i^T d_i^{\text{new}})^{-1/2}$
end for
$d_1 \leftarrow d_1^{\text{new}}$
$s_1 \leftarrow d_1(d_1^T r_1)$
for $i = 2..\ell$ **do**
$d_i \leftarrow d_i^{\text{new}}$
$s_i \leftarrow I_{i-1}^i s_{i-1}$
$s_i \leftarrow s_i + d_i(d_i^T r_i)$
end for
$x \leftarrow x + s_\ell$
end while

5.7 Evaluated Solution Methods

5.7.1 Conjugate Gradient

CG and variants thereof are commonly used to solve the matting problem, so we examine the properties of the classic CG method alongside the multigrid methods.

5.7.2 V-cycle

In order to construct a v-cycle algorithm, we need to choose a relaxation method and construct transfer operators. We chose Gauss-Seidel as the relaxation method, since it is both simple and efficient. Through experiment, we found that simple bilinear transfer operators (also called *full-weighting*) work very well for the matting problem. The downsample operator I_h^{2h} is represented by the stencil

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}.$$

If the stencil is centered on a fine grid point, then the numbers represent the contribution of the 9 fine grid points to their corresponding coarse grid point. The corresponding upsample operator I_{2h}^h is represented by the stencil

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}.$$

This stencil is imagined to be centered on a fine grid point, and its numbers represent the contribution of the corresponding coarse grid value to the 9 fine grid points. The algorithm is then given by Alg. 8.

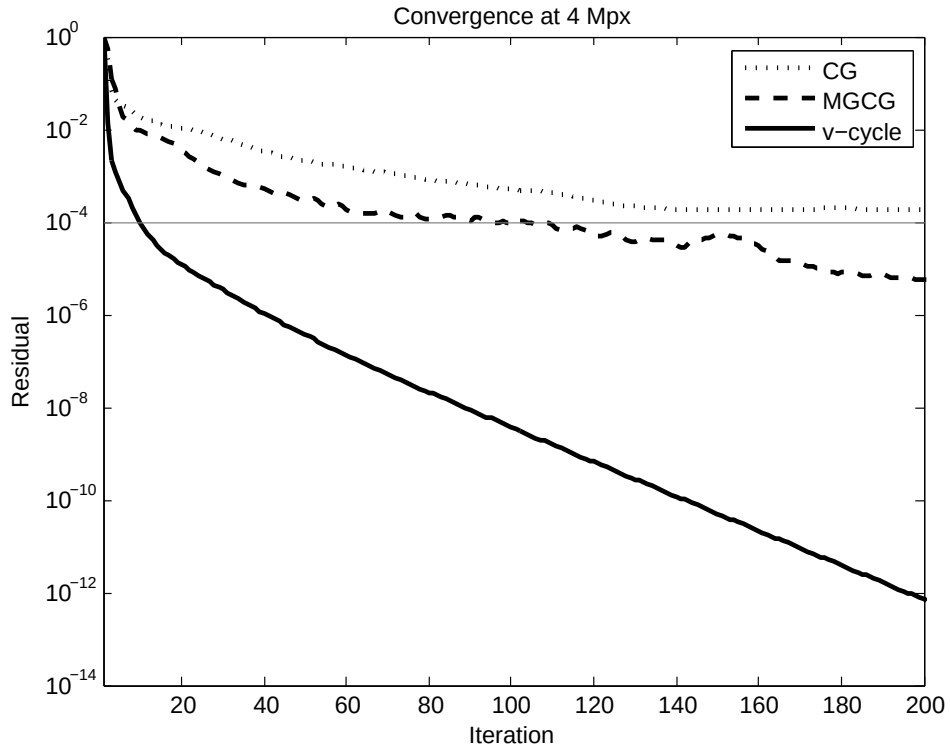


Figure 5.5: Convergence on a 4 Mpx image. Horizontal line denotes proposed termination value of 10^{-4}.

Resolution	CG	MGCG	v-cycle
0.5 Mpx	0.594	0.628	0.019
1 Mpx	0.357	0.430	0.017
2 Mpx	0.325	0.725	0.017
4 Mpx	0.314	0.385	0.015

Table 5.1: Initial convergence rates ρ_0 on image 5 in [27]. Lower is better.

5.7.3 Multigrid Conjugate Gradient

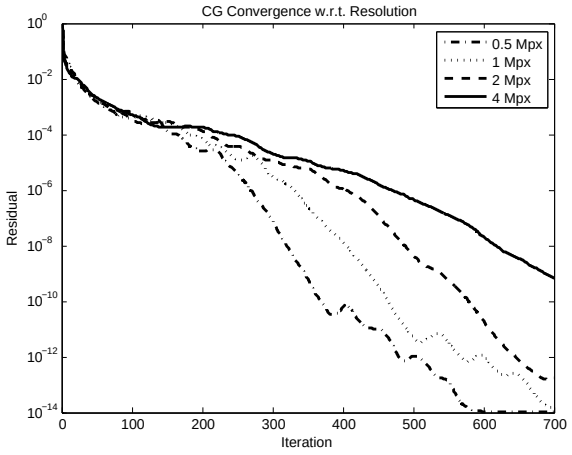
We also examine the multigrid conjugate gradient algorithm of [26], which is a more recent multigrid variant, and seems like a natural extension of CG. For this algorithm, we find that the bilinear (full-weighting) transfer operators also work very well. The algorithm is given by Alg. 10, where we give the full pseudo-code as the original pseudo-code given in [26] is erroneous.

5.8 Experiments & Results

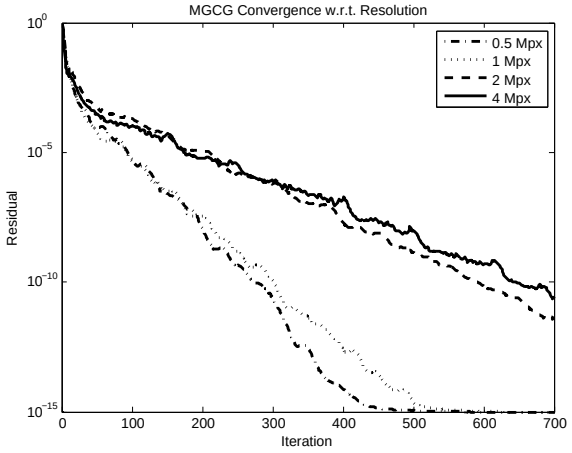
In order to compare the solution methods, we choose the matting Laplacian proposed by [19], as it is popular and provides source code. Please note that we are comparing the solvers, and not the performance of the matting Laplacian itself, as these solvers would equally apply to other matting Laplacians. We set their ϵ parameter to 10^{-3}, and we form our linear system according to Eq. (1.6) with $\gamma = 1$.

All methods are given the initial guess $\alpha_0 = 0$. To make the objective value comparable across methods and resolutions, we take $\|f - A\alpha_i\|_2 / \|f - A\alpha_0\|_2 = \|f - A\alpha_i\|_2 / \|f\|_2$ as the normalized residual at iteration i, or *residual* for short. We set the termination condition for all methods at all resolutions to be the descent of the residual below 10^{-4}.

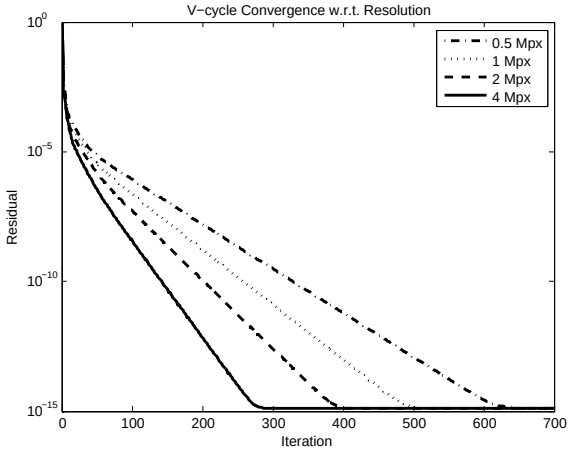
First, we evaluate the convergence of conjugate gradient, multigrid conjugate gradient, and v-cycle. Typical be-



(a)



(b)



(c)

Figure 5.6: CG (a) & MGCG (b) slow down as resolution increases, while v-cycle (c) *speeds up*.

	CG			MGCG			v-cycle		
Image	1 Mpx	2 Mpx	4 Mpx	1 Mpx	2 Mpx	4 Mpx	1 Mpx	2 Mpx	4 Mpx
1	143	159	179	81	81	94	12	11	11
2	151	176	185	73	86	99	12	10	10
3	215	280	334	103	108	118	12	9	8
4	345	439	542	143	171	230	16	14	11
5	180	211	236	44	120	95	13	12	10
6	126	148	180	65	91	61	11	10	9
7	152	179	215	86	84	96	11	10	8
8	314	385	468	135	167	180	16	12	9
9	237	285	315	87	119	91	12	11	8
10	122	149	180	72	72	95	10	9	8
11	171	192	78	77	96	32	12	11	7
12	127	151	179	65	86	77	9	8	7
13	266	291	317	143	156	159	18	15	11
14	125	145	175	76	79	83	13	12	11
15	129	156	190	67	69	105	12	12	11
16	292	374	468	123	171	178	15	14	12
17	153	190	230	63	81	86	10	8	7
18	183	238	289	86	104	107	26	22	16
19	91	104	116	52	58	61	12	10	8
20	137	164	199	81	82	101	12	9	7
21	195	288	263	118	83	137	22	16	15
22	118	139	161	62	75	72	11	9	7
23	140	166	196	64	77	74	11	9	8
24	213	266	325	76	131	104	14	14	10
25	319	455	629	142	170	249	39	31	24
26	370	432	478	167	192	229	41	32	24
27	271	304	349	121	144	147	17	15	12

Table 5.2: Iterations to convergence (residual less than 10^{-4}) at 1, 2, and 4 Mpx on images from [27]. CG and MGCG require more iterations as resolution increases while v-cycle requires less.

Image	a	p
1	11.8	-0.065
2	11.7	-0.140
3	11.8	-0.309
4	16.2	-0.261
5	13.2	-0.183
6	11.0	-0.144
7	11.2	-0.220
8	16.0	-0.415
9	12.3	-0.270
10	10.0	-0.160
11	12.5	-0.343
12	9.02	-0.180
13	18.2	-0.340
14	13.0	-0.120
15	12.2	-0.061
16	15.2	-0.156
17	9.90	-0.265
18	26.4	-0.333
19	12.1	-0.289
20	12.0	-0.394
21	21.4	-0.300
22	11.1	-0.321
23	10.9	-0.236
24	14.6	-0.216
25	39.1	-0.347
26	41.2	-0.381
27	17.2	-0.243

Table 5.3: Fitting of v-cycle required iterations to an^p (power law), with n being problem size in Mpx. Average p is $E[p] = -0.248$, meaning this solver is sublinear in n.

Method	SAD	MSE	Gradient	Connectivity
Ours	**11.1**	**10.8**	**12.9**	9.1
[19]	14	13.9	14.5	**6.4**
[11]	17	15.8	16.2	9.2

Table 5.4: Average rank in benchmark [27] on 11 Mar 2013 with respect to different error metrics.

havior on the dataset [27] is shown in Fig. 5.5 at 4 Mpx resolution. CG takes many iterations to converge, MGCG performs somewhat better, and v-cycle performs best.

Next, we demonstrate in Fig. 5.6 that the number of iterations v-cycle requires to converge on the matting problem does not increase with the resolution. In fact, we discover that v-cycle requires *fewer* iterations as the resolution increases. This supports a sub-linear complexity of v-cycle on this problem in practice since each iteration is $\mathcal{O}(n)$.

We list a table of the convergence behavior of the algorithms on all images provided in the dataset in Table 5.2. This demonstrates that MGCG performs decently better than CG, and that v-cycle always requires few iterations as the resolution increases. By fitting a power law to the required iterations in Table 5.3, we find the average number of required v-cycle iterations is $\mathcal{O}\left(n^{-0.248}\right)$. Since each iteration is $\mathcal{O}(n)$, this means that this is the first solver demonstrated to have sublinear time complexity on the matting problem. Specifically, the time complexity is $\mathcal{O}\left(n^{0.752}\right)$ on average. Theoretical guarantees [8] ensure that the worst case time complexity is $\mathcal{O}(n)$.

Though surprising, the sub-linear behavior is quite understandable. Most objects to be matted are opaque with a boundary isomorphic to a line segment. The interior and exterior of the object have completely constant α locally, which is very easy for v-cycle to handle at its lower levels, since that is where the low frequency portions of the solution get solved. This leaves the boundary pixels get handled at the highest resolutions, since edges are high-frequency. These edge pixels contribute to the residual value, but since the ratio of boundary to non-boundary pixels is $\mathcal{O}\left(n^{-0.5}\right)$, tending to 0 with large n, the residual tends to decrease more rapidly per iteration as the resolution increases as proportiontely more of the problem gets solved in lower resolution.

Another way to view it is that there is a fixed amount of information present in the underlying image, and increasing the number of pixels past some point does not really add any additional information. V-cycle is simply the first algorithm studied to exploit this fact by focusing its effort preferentially on the low resolution parts of the solution.

Perhaps if we had enough computational resources and a high-enough resolution, v-cycle could drop the residual to machine epsilon in a single iteration. At that point, we would again have a linear complexity solver, as we would always have to do 1 iteration at $\mathcal{O}(n)$ cost. In any case, $\mathcal{O}(n)$ is *worst* case behavior as guaranteed by [8], which is still much better than any solver proposed to date like [11] and [32] (which are at best $\mathcal{O}\left(n^2\right)$). We would also like to mention that the per-iteration work done by the proposed method is constant with respect to the number of unknowns, unlike most solvers including [11] and [32].

Table 5.1 lists the initial convergence rates ρ_0 (the factor by which the residual is reduced per iteration). [8] guarantees that the convergence rate for v-cycle is at most $\mathcal{O}(1)$ and bounded away from 1. We discover that on the matting problem the convergence rate is *sub*-constant. Yet again, v-cycle dramatically overshadows other solvers.

Although a MATLAB implementation of [19] is available, we found it to use much more memory than we believe it should. We therefore developed our own implementation in C++ using efficient algorithms and data structures to run experiments at increasing resolution. Since there are 25 bands in the Laplacian, and 4 bytes per floating point entry, the storage required for the matrix in (1.6) is 95 MB/Mpx, and the total storage including the downsampled matrices is about 127 MB/Mpx. The time to generate the Laplacian is 0.8 s/Mpx. The CPU implementation runs at 46 Gauss-Seidel iterations/s/Mpx and 11 v-cycle iterations/s/Mpx. Assuming 10 iterations of v-cycle are run, which is reasonable from Table 5.2, our solver runs in about 1.1 s/Mpx, which is nearly 5 times faster than [11] at low resolutions, and even faster at higher resolutions. Upon acceptance of this paper, we will post all the code under an open-source license.

Although we consider this to be a matting solver rather than a new matting algorithm, we still submitted it to the online benchmark [27] using [19]'s Laplacian. The results presented in Table 5.4 show some improvement over the base method [19], and more improvement over another fast method [11]. In theory, we should reach the exact same solution as [19] in this experiment, so we believe any performance gain is due to a different choice of their ϵ parameter.

Chapter 6

Extensions to Image Formation

The methods we utilize in [18] are not limited to image matting. Those multigrid methods extend to a wide range of problems we refer to as *image formation* problems in Computer Vision and Image Processing. In general, if a problem has the following characteristics, we can use multigrid methods to greatly enhance the efficiency:

1. Linear form $Au = f$, and $A > 0$
2. u is an image, or at least can be downsampled
3. A is large and sparse and $\kappa(A) \to \infty$ as $n \to \infty$

We investigate three such problems in our field, namely optical flow, image deconvolution, and texture modeling and synthesis to show the utility of these methods in greatly reducing the computational cost that has heretofore been associated with them.

6.1 Optical Flow

Optical flow is a classic Computer Vision problem. It asks for the displacement field for each pixel between two frames of video. It starts by assuming that a pixel i retains its color after movement δ_i from frame t to frame $t+1$:

$$I_{t+1}(i + \delta_i) = I_t(i). \tag{6.1}$$

Further, by assuming small displacement,

$$I_{t+1}(i + \delta_i) = I_{t+1}(i) + \delta_i^T \nabla I_{t+1}(i) + \mathcal{O}\left(\|\delta_i\|^2\right) \approx I_{t+1}(i) + \delta_i^T \nabla I_{t+1}(i). \tag{6.2}$$

Then, we have

$$I_t(i) \approx I_{t+1}(i) + \delta_i^T \nabla I_{t+1}(i), \text{ or} \tag{6.3}$$

$$\delta_i^T \nabla I_{t+1}(i) \approx I_t(i) - I_{t+1}(i). \tag{6.4}$$

Taken across all pixels i, and writing this in matrix notation:

$$\begin{bmatrix} \ddots & 0 & 0 & \ddots & 0 & 0 \\ 0 & \nabla_u I_t & 0 & 0 & \nabla_v I_t & 0 \\ 0 & 0 & \ddots & 0 & 0 & \ddots \end{bmatrix}_{n \times 2n} \begin{pmatrix} \delta_u \\ \delta_v \end{pmatrix}_{2n \times 1} = I_t - I_{t+1}, \text{ or} \tag{6.5}$$

$$[D_u \; D_v]\, \delta = g \tag{6.6}$$

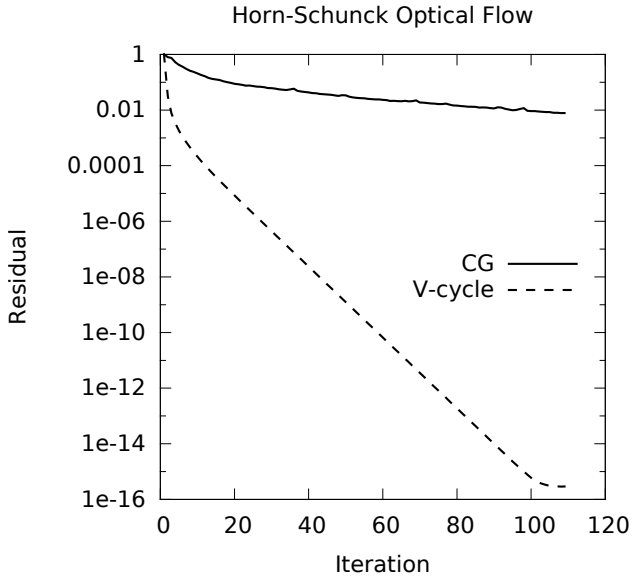


Figure 6.1: V-cycle applied to optical flow.

Obviously, there are $2n$ variables in the vector field, but only n constraining equations. Solving this requires prior information on the field. Horn and Schunck [13] used a Laplace operator L as the regularizing term, since they expect the motion field to be smooth. Also, to get the system to be square, we end up with

$$\left(\begin{bmatrix} D_u^2 & D_u D_v \\ D_v D_u & D_v^2 \end{bmatrix} + \lambda \begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix} \right) \delta = \begin{bmatrix} D_u \\ D_v \end{bmatrix} g \triangleq f. \tag{6.7}$$

Here, λ is a Lagrange parameter for the regularization on the smoothness of the motion field δ. Just as in matting, we end up with a large sparse system, whose variable represents something like an image.

1. ✓ Linear form $Au = f$, and $A > 0$
2. ✓ u is an image, or at least can be downsampled
3. ✓ A is large and sparse and $\kappa(A) \to \infty$ as $n \to \infty$

This problem is perfectly-suited to be solved efficiently by multigrid algorithms. In fact, multigrid methods *were* studied in the solution of optical flow in the 1980s [6], perhaps because multigrid methods were in their heyday at the same time as optical flow made its mark. However, nobody ever analyzed the computational complexity. When we apply the same methods from [18] to Horn-Schunck optical flow, we get nearly the same results as before, and again obtain a sub-linear solver (Figure 6.1).

6.2 Deconvolution

Deconvolution is a fundamental issue in Image Processing. If an image gets filtered by some process, we may wish to *deconvolve* the noisy filtered image to get an approximation of the original. In other words

$$Hu = v, \text{ or} \tag{6.8}$$

$$H^T H u = H^T v, \tag{6.9}$$

where H is a convolutional matrix for the filter, u is the original image, and v is the filtered image. Because H is rarely full-rank, we need to regularize the system in some way to be able to recover u. A very simple way is to regularize the 2-norm of u, since we do not expect solutions with very high energy:

$$(H^T H + \lambda I)u = H^T v, \tag{6.10}$$

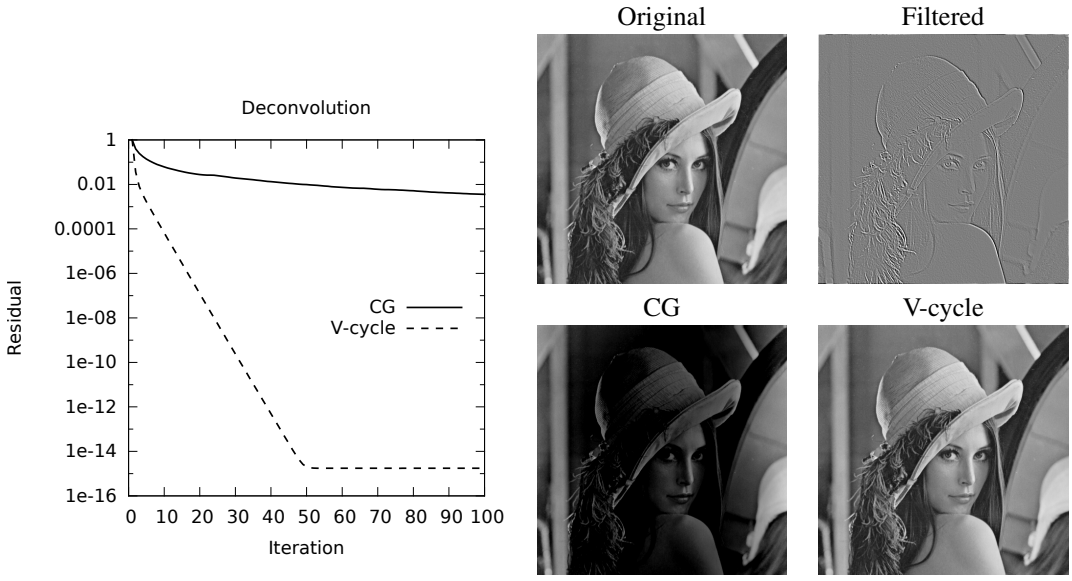


Figure 6.2: V-cycle on deconvolution

where again λ is the regularization parameter. If H represents a sort of highpass kernel, we again have nearly the same problem as in Chapter 5, and we expect multigrid algorithms to give a large increase in performance.

1. ✓ Linear form $Au = f$, and $A > 0$
2. ✓ u is an image, or at least can be downsampled
3. ✓ A is large and sparse and $\kappa(A) \to \infty$ as $n \to \infty$

6.3 Texture Modeling & Synthesis

There are two major veins of research about image textures. One concerns itself with describing or modeling the texture, and the other concerns itself with synthesizing textures. The goal of modeling is more or less to provide a prior $p(I \mid T)$ over images I for a given type of texture T. The usefulness of such a prior is apparent in applications like super-resolution, where textured regions may be superresolved by matching low-resolution textures to a set of high-resolution texture models. Texture synthesis is the opposite side of the coin, in which we ask to generate a sample texture image from $p(I \mid T)$, whether or not we have such an explicit model.

These two approaches are unified in [34] in a very elegant way, in which a texture model is learned from examples of the texture by creating an explicit Gibbs distribution, which is sampled using a Gibbs sampler in order to perform the learning. The result is an explicit form for $p(I \mid T)$ which can evaluate the likelihood of an image under the texture model, and can also generate new images. The disadvantage in both learning and synthesis is the reliance on the Gibbs sampler.

The Gibbs sampler is a mechanism by which a Gibbs distribution (equivalently a Markov Random Field) can be sampled by mariginalizing the distribution for a single pixel, drawing the pixel value from the marginal, and repeating over all pixels in the image for a very large number of iterations until the distribution of the entire image converges to that of the model. It is not at all clear how long or even *if* such an algorithm will converge, and in this particular context, it may be necessary to visit each pixel in the image hundreds or thousands of times before the generated texture can be said to be drawn from the model.

In practice, we observe for a fixed model, the complexity of generating a texture with n pixels is $\mathcal{O}(n)$. However, as the resolution of the learned texture increases, the model must incorporate features from larger and larger filters to capture the large-scale structures. This means that in order to accurately learn and synthesize a texture with n pixels,

the total complexity is something like $\mathcal{O}\left(n^2\right)$. Since even small textures on a 128×128 grid can take hours to converge on modern hardware, we need more efficient algorithms to make this type of modeling and synthesis practical.

We propose importing and modifying the concepts of multigrid analysis to the Gibbs sampler in order to make synthesis more efficient. First, let us look at the Gibbs sampling algorithm 11. Its inputs are the initial texture u, the marginal distributions $p(v,i) = p\left(u(v) = i \mid u(-v)\right)$, and the number of sweeps w to perform. The initial texture is typically one whose pixels are chosen uniformly randomly, the marginals are given by FRAME in this case and are constructed by computing histograms of filter responses, and w is a parameter that is increased until the algorithm appears to truly generate samples from the texture distribution.

Algorithm 11 Gibbs sampler

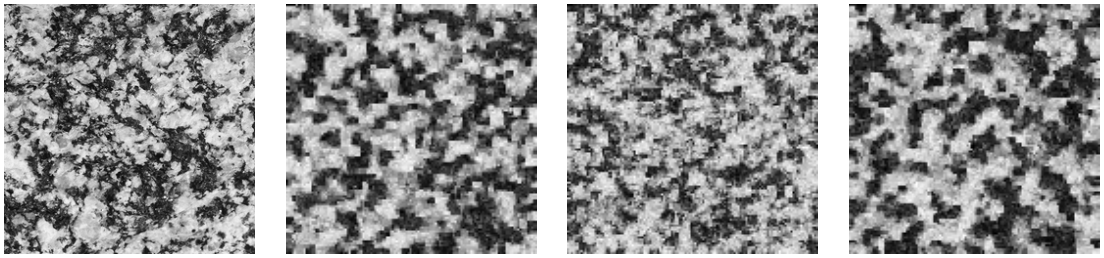
function GIBBS($u_{M\times N}$, $p(v,i)$, w)
  for numFlips = $1..wMN$ **do**
    $v \leftarrow$ draw $(U(0..M-1, 0..N-1))$
    for $i = 0..G-1$ **do**
      $f_i \leftarrow p(v,i)$
    end for
    $u(v) \leftarrow$ draw(f)
  end for
end function

The improvement we are proposing is sketched in Algorithm 12. In this case, we assume we have a texture model for the given texture at various resolutions so that $p^h(v,i)$ is the marginal for the texture at grid spacing h. The idea is that we should first do a few iterations of Gibbs sampling at full resolution to begin smoothing the initial texture, downsample it to initialize the next lower resolution estimate, recursively call the sampler on the lower resolution and obtain a low-resolution sample, then upsample it and do more iteration at full resolution.

Algorithm 12 Multigrid Gibbs sampler.

function MGGIBBS(u^h, $p^h(v,i)$, w)
  $u^h \leftarrow$ gibbs$(u^h, p^h(v,i), w)$ (pre-smoothing)
  if $h = H$ **then**
    return u^h
  end if
  $u^{2h} \leftarrow I_h^{2h} u^h$ (low-resolution estimate)
  $r^h \leftarrow u^h - I_{2h}^h u^{2h}$ (high frequency residual)
  $u^{2h} \leftarrow$ mgGibbs$(u^{2h}, I_h^{2h} p^h(v,i), w)$ (recurse on low resolution)
  $u^h \leftarrow r^h + I_{2h}^h u^{2h}$ (upsample and replace high frequencies)
  $u^h \leftarrow$ gibbs$(u^h, p^h(v,i), w)$ (post-smoothing)
  return u^h
end function

Although the results are preliminary, we are getting promising results from the proposed sampler. We trained full- and quarter-resolution models from a granite texture. See the original and synthesized textures from the learned models in Figure 6.3. We compared the convergence of the synthesis of using the full-scale model only with the Gibbs sampler vs. using both models in the proposed multigrid Gibbs sampler in Figure 6.4. Since we get much improved performance in this test case, we expect very promising results from this algorithm as we have more time to develop it in the future.



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

Figure 6.3: Granite texture (a), full-resolution sample (b), quarter-resolution sample (c), full-resolution sample obtained with multigrid Gibbs sampler (d).

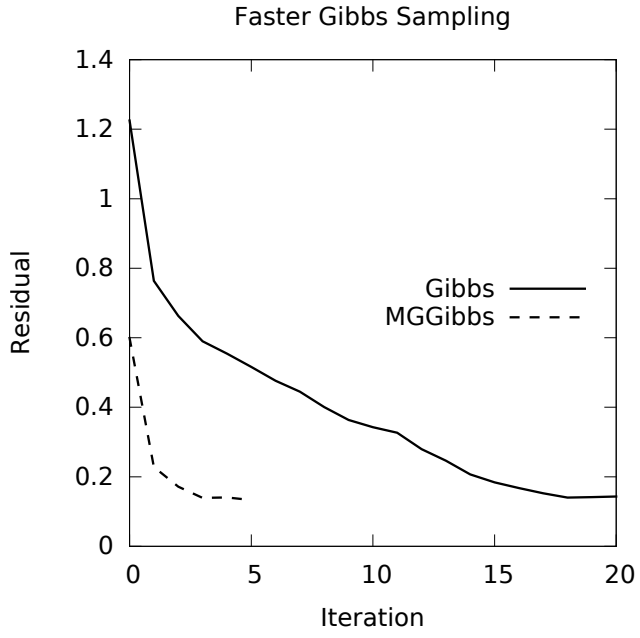


Figure 6.4: Two-level multigrid sampler converges much faster than simple Gibbs sampling.

Chapter 7

Conclusion

The research summarized in this thesis focuses on three major aspects of natural image matting: improving quality, decreasing required input, and most importantly, improving efficiency. To improve quality, we found in [16] that errors caused by oversmoothing can be better handled by incorporating median-filtered color channels into the color model. To decrease the required input, we found in [17] that constructing graph Laplacians from nonlocal patches is a much better regularizer than the typical spatial smoothness regularization, which can perform well even when the input is sparse. Finally, we found in [18] that the matting problem can be solved exactly by iterative multigrid methods in sublinear time complexity, a substantial improvement from the commonly-used conjugate gradient algorithm.

We consider the last work to be the most important. Not only does it work well for matting, but we show that the results extend to many similar problems in Computer Vision and Image Processing. This allows matting and other tasks to be implemented efficiently so that they are practical to use in the future as image resolution grows.

We did not invent multigrid algorithms, but were the first to do extensive studies on their utility in our field. Further, they are not always applicable in every image problem. In a strict sense, the problems must be well-behaved satisfying particular assumptions. However, the underlying concepts seem fundamental to image formation problems, and our results demonstrate that when approaching these problems, we should not blindly apply generic solution methods. Rather, we should seek to exploit the fact that we are solving for images, and that images should be treated differently from arbitrary data.

We hope this work reawakens not only interest in the utility of multigrid algorithms in Computer Vision, but also compels others to focus on the complexity of the algorithms in our field. In a world increasingly dominated by "big data," it is less and less acceptable to propose algorithms that cannot be implemented efficiently. We must endeavor to find techniques that can scale gracefully into the forseeable future.

Bibliography

[1] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 60–65. IEEE, 2005. 37

[2] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–264. IEEE, 2001. 18

[3] S. Dai and Y. Wu. Motion from blur. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 15, 37

[4] J. Demmel, I. Dumitriu, and O. Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91, 2007. 25

[5] O. Duchenne, J. Audibert, R. Keriven, J. Ponce, and F. Segonne. Segmentation by transduction. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008*, pages 1–8, 2008. 20

[6] W. Enkelmann. Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences. *Computer Vision, Graphics, and Image Processing*, 43(2):150–177, 1988. 59

[7] J. Fan, X. Shen, and Y. Wu. Closed-loop adaptation for robust tracking. In *Computer Vision–ECCV 2010*, pages 411–424. Springer, 2010. 15

[8] J. Gopalakrishnan and J. E. Pasciak. Multigrid convergence for second order elliptic problems with smooth complex coefficients. *Computer Methods in Applied Mechanics and Engineering*, 197(49):4411–4418, 2008. 47, 57

[9] J. Hartigan and M. Wong. A k-means clustering algorithm. *JR Stat. Soc., Ser. C*, 28:100–108, 1979. 39

[10] K. He, J. Sun, and X. Tang. Single image haze removal using dark channel prior. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1956–1963. IEEE, 2009. 15

[11] K. He, J. Sun, and X. Tang. Fast matting using large kernel matting laplacian matrices. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2165–2172. IEEE, 2010. 16, 18, 30, 41, 57

[12] M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS, 1952. 27

[13] B. Horn and B. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. 59

[14] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM (JACM)*, 5(4):339–342, 1958. 24

[15] D. S. Kershaw. The incomplete Cholesky—conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26(1):43–65, 1978. 29

[16] P. G. Lee and Y. Wu. L1 matting. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 4665–4668. IEEE, 2010. 8, 32, 63

[17] P. G. Lee and Y. Wu. Nonlocal Matting. In *2011 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2193–2200. IEEE, 2011. 8, 18, 32, 37, 63

[18] P. G. Lee and Y. Wu. Scalable Matting: A Sub-linear Approach. *arXiv:1404.3933 [cs.CV]*, 2014. 8, 43, 58, 59, 63

[19] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):228–242, 2008. 6, 15, 18, 20, 30, 37, 40, 41, 42, 53, 57

[20] A. Levin, A. Rav Acha, and D. Lischinski. Spectral matting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1699–1712, 2008. 37

[21] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999. 15

[22] S. F. McCormick, editor. *Multigrid Methods*. Society for Industrial and Applied Mathematics, 1987. 43

[23] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14: Proceeding of the 2001 Conference*, pages 849–856, 2001. 39

[24] S. Noah. The Median of a Continuous Function. *Real Analysis Exchange*, 33(1):269–274, 2007/2008. 33

[25] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 313–318. ACM, 2003. 19

[26] C. Pflaum. A multigrid conjugate gradient method. *Applied Numerical Mathematics*, 58(12):1803–1817, 2008. 48, 49, 50, 53

[27] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. In *Proc. CVPR*, pages 1826–1833. Citeseer, 2009. 5, 6, 11, 37, 41, 42, 53, 55, 57

[28] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994. 26, 28, 29

[29] D. Singaraju, C. Rother, and C. Rhemann. New appearance models for natural image matting. 2009. 20

[30] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum. Poisson matting. In *ACM Transactions on Graphics (ToG)*, volume 23, pages 315–321. ACM, 2004. 19, 30

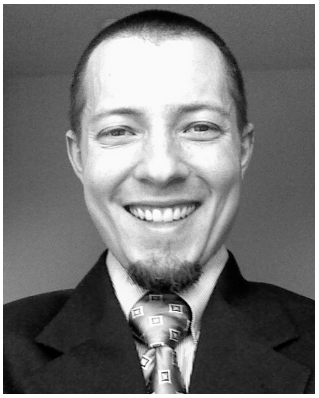
[31] R. Szeliski. Locally adapted hierarchical basis preconditioning. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 1135–1143. ACM, 2006. 16

[32] C. Xiao, M. Liu, D. Xiao, Z. Dong, and K.-L. Ma. Fast Closed-Form Matting Using a Hierarchical Data Structure. *Circuits and Systems for Video Technology, IEEE Transactions on*, 24:49–62, 2014. 57

[33] Y. Zheng and C. Kambhamettu. Learning Based Digital Matting. In *ICCV*, 2009. 18, 20

[34] S. C. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998. 60

Vita



Philip Lee was born in Oconee county South Carolina on 24 Feb 1986. He graduated from West-Oak High School and began studying at Clemson University in 2004. During the summer of 2006, he attented a research experience for undergraduates in analytic number theory at Clemson University under the supervision of Neil Calkin, Kevin James, and David Penniston, resulting in a 2008 publication. During the summer of 2007, he worked at the National Security Agency as an intern in the Director's Summer Program, where he improved the efficiency of a statistical language modeling algorithm by a factor of 10^3. He received the Bachelor of Science degree in Mathematics from Clemson, and graduated Summa Cum Laude in May 2008.

Philip began his Ph.D. career at Northwestern University in September 2009 under the supervision of Ying Wu in the department of Electrical Engineering and Computer Science. He has focused his research efforts on the natural image matting problem in Computer Vision. His published works include two papers that improve the accuracy and usability of matting, and another that fundamentally improves the efficiency of a whole class of matting algorithms as well as image formation problems in general. Philip also co-authored several other papers while at Northwestern that develop new realtime algorithms for tracking and depth/color camera calibration. During the summer of 2011, he worked as a research intern at Kodak, where he developed an algorithm to fuse sets of photos together to automatically remove unwanted objects in consumer photos. He received his Ph.D. from Northwestern University in Electrical Engineering in June 2014.