UNIVERSITY OF CAMBRIDGE

Shell-Based Geometric Image and Video Inpainting

L. Robert Hocking

Sidney Sussex College

This dissertation is submitted on August, 2017 for the degree of Doctor of Philosophy

I dedicate this thesis to Mikasa Ackerman
who was a role model
during the final, brutal months of the PhD.

Abstract

The subject of this thesis is a class of fast inpainting methods (image or video) based on the idea of filling the inpainting domain in successive shells from its boundary inwards. Image pixels (or video voxels) are filled by assigning them a color equal to a weighted average of either their already filled neighbors (the "direct" form of the method) or those neighbors plus additional neighbors within the current shell (the "semi-implicit" form). In the direct form, pixels (voxels) in the current shell may be filled independently, but in the semi-implicit form they are filled simultaneously by solving a linear system. We focus in this thesis mainly on the image inpainting case, where the literature contains several methods corresponding to the *direct* form of the method - the semi-implicit form is introduced for the first time here. These methods effectively differ only in the order in which pixels (voxels) are filled, the weights used for averaging, and the neighborhood that is averaged over. All of them are very fast, but at the same time all of them leave undesirable artifacts such as "kinking" (bending) or blurring of extrapolated isophotes.

This thesis has two main goals. First, we introduce new algorithms within this class, which are aimed at reducing or eliminating these artifacts, and also target a specific application - the 3D conversion of images and film. The first part of this thesis will be concerned with introducing 3D conversion as well as Guidefill, a method in the above class adapted to the inpainting problems arising in 3D conversion. However, the second and more significant goal of this thesis is to study these algorithms as a class. In particular, we develop a mathematical theory aimed at understanding the origins of artifacts mentioned. Through this, we seek is to understand which artifacts *can* be eliminated (and how), and which artifacts are inevitable (and why). Most of the thesis is occupied with this second goal.

Our theory is based on two separate limits - the first is a *continuum* limit, in which the pixel width $h \to 0$, and in which the algorithm converges to a partial differential equation. The second is an *asymptotic limit* in which h is very small but non-zero. This latter limit, which is based on a connection to random walks, relates the inpainted solution to a type of discrete convolution. The former is useful for studying kinking artifacts, while the latter is useful for studying blur. Although all the theoretical work has been done in the context of image inpainting, experimental evidence is presented suggesting a simple generalization to video.

Finally, in the last part of the thesis we explore shell-based video inpainting. In particular, we introduce spacetime transport, which is a natural generalization of the ideas of Guidefill and its predecessor, coherence transport, to three dimensions (two spatial dimensions plus one time dimension). Spacetime transport is shown to have much in common with shell-based image inpainting methods. In particular, kinking and blur artifacts persist, and the former of these may be alleviated in exactly the same way as in two dimensions. At the same time, spacetime transport is shown to be related to optical flow based video inpainting. In particular, a connection is derived between spacetime transport and a generalized Lucas–Kanade optical flow that does not distinguish between time and space.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or am concurrently submitting, for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or is being concurrently submitted, for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text.

Chapter 4 describes Guidefill, an algorithm which grew out of the time that I spent working in industry at Gener8 with my friend Russell MacKenzie. Russell and I spent many lunchhours discussing my plans for this algorithm, and his feedback was very helpful. The concept of "smart order" (Section 4.5) is something we dreamed up together during one of our discussions. However, aside from this, both the algorithm (which evolved considerably after I left Gener8) and the mathematics surrounding it are my own work. This work is now due to be published in SIAM imaging in collaboration with Russell as well as my supervisor Carola-Bibiane Schönlieb, who was very helpful in providing feedback as the paper evolved [37].

Chapters 5 and 6 contain work that was done in collaboration with Tom Holding and my supervisor Carola-Bibiane Schönlieb, which is available as a preprint here [36]. The work done in collaboration with Tom is entirely contained in Lemma 6.4.3 and step 1 of stage 4 of the proof of Theorem 6.3.1. In addition, the proof of Lemma A.8.4 in Appendix A.8 was done in collaboration with Dömötör Pálvölgyi. My supervisor Carola was invaluable in reminding me to keep the motivation for the paper clear and not let it get too abstract, as well as giving other helpful feedback and suggestions. All other work is my own.

L. Robert Hocking
August, 2017

Acknowledgements

There are several people without whom this thesis would not have been possible. In addition to acknowledging them, I would like to use this space to acknowledge those who enriched my experience at Cambridge, and to reflect upon that experience.

There are at least three people without whom this thesis would not exist. First, I acknowledge my parents for their support throughout the PhD. Without their encouragement I likely would never have applied to Cambridge in the first place. Second, I acknowledge my supervisor Carola-Bibiane Schönlieb for introducing me to image inpainting and for her guidance and patience. Third, I owe a debt of gratitude to my long time friend Russell MacKenzie, who convinced me near the start of my PhD to intermit and come work with him in industry on inpainting as it applies to 3D conversion. This experience in industry set the direction that my PhD would take. Russell and I have now co-authored a paper together and I hope that in doing so, conversely, I have given him at least a small taste of academia. Thanks are also owed to one of my other co-authors, Tom Holding, whose incredible energy made him a joy to work with, and whose perspective on the correct way to attack a problem is usually right. I should also acknowledge the Cambridge Overseas Trust and the William and Margaret Brown Scholarship, which funded me throughout my PhD.

The most rewarding aspect of life in Cambridge has been the incredible people that I have had the privilege to know: my friends Tom and Henry, who at the same time manage to be two of the smartest and two of the funniest people I know; my friend Kim, who is possibly even funnier and always has a good book recommendation; my friend Karen, who is the nicest; and finally Martin, who is so down to earth and yet works on problems that are of such deep significance and so fundamental to our understanding of the universe (and are of course extremely challenging) - I salute you for your bravery in using your time on earth to go after the big questions. It really is an honor to be able to count all of you among my friends.

Living in college accommodation had its ups and downs. The College's obsession with rules has been both vexing and a source of constant amusement. On one particularly memorable occasion, the rice cooker of my flatmate from Shanghai was confiscated as an electrical hazard. The college replaced it a few days later with a special “college approved” rice cooker, only to confiscate that as well a week later, apparently unable to recognize their own rice cookers.

The university's long history also comes with pros and cons. On the one hand, you have a sense of connection to the past that is difficult to find elsewhere. On the other hand, there is a chronic shortage of space. An experience from my third year illustrates this trade off. That year, when I was living in a converted entrance hall just big enough to fit a bed (there was no desk), on my way home from work one day I noticed a brass plaque on the brick wall outside our house. I stepped closer to take a look: “CHARLES DARWIN LIVED HERE 1836-7”. My tiny room suddenly didn't seem so bad.

Living in Cambridge offers a connection both to history and to what is happening in the world that I will sorely miss. It is certainly not everywhere that I can come into the office one day, only to be told

that Stephen Hawking has just won £1,000,000 and everyone is invited to Pavilion B for Champagne. Or where I can read a paper on five-dimensional black holes only to realize that the author works in an office five hundred meters from my own. It's not everywhere that I can come into work, discover that a celebration of Stephen Hawking's 75th birthday is in full tilt, go to a talk in honor of the occasion and hear that "the proof of the existence of the maximal Cauchy development for the Einstein equations was viewed with dissatisfaction by mathematicians for many years, until a bright young man right here in the audience managed to prove it without using Zorn's lemma", and realize that that "bright young man" is the person who supervised my own research project on general relativity in first year.

However, perhaps the most memorable anecdote came when I finished reading a book about Joseph Needham, a Cambridge professor and the creator of the series of books "Science and Civilization in China". The book concluded by mentioning the existence of a memorial to Needham, located in Caius College, bearing the inscription

人去留影

(roughly, the man departs - his shadow remains). Noticing that the park bench upon which I sat reading was a ten minute walk from Caius College, I promptly set off to find it.

Contents

Chapter 1

Introduction

Image (video) inpainting refers to the filling in of a region in an image (video) where information is missing or needs to be replaced (due to, for example, an occluding object), in such a way that the result looks plausible to the human eye. Hereafter, we use “inpainting” as a blanket term to refer to either image or video inpainting - in either case, we refer to the region to be filled as the inpainting domain. Since the seminal work of Bertalmio et al. [11], inpainting has become increasingly important, with applications ranging from removing an undesirable or occluding object from a photograph, to painting out a wire in an action sequence, to 3D conversion of film [37], as well as 3D TV and novel viewpoint construction [15, 42, 23] in more recent years. See [29] for a recent survey of image inpainting techniques, as well as [3] for a survey of video inpainting.

In this thesis we focus on a simple class of inpainting methods (image or video) in which the inpainting domain is filled in concentric shells. At each iteration, a given pixel (or voxel, in the case of video) in the current shell is “filled” by assigning it a color equal to a weighted average of the colors of either its *already* filled neighbors (we call this the direct method), or of those neighbors *as well as* unknown neighbors within the same shell (we call this the semi-implicit extension or the semi-implicit form, as it involves solving a linear system) - see Figure 1.1 for an illustration of shell based inpainting and Figure 1.2 for an illustration of the difference between the direct method and the semi-implicit extension. Examples of the direct method that have appeared in the literature prior to this thesis are Telea’s Algorithm [64], coherence transport [12], and coherence transport with adapted distance functions [44]. The literature now contains one additional method, Guidefill [37], which was created as part of this thesis. Chapter 4 of this thesis will be devoted to a description of Guidefill, which is designed for inpainting as it applies to a specific application, namely the 3D conversion of images or film. 3D conversion refers to the generation of a right and left eye stereo pair of images (videos) given the image (video) corresponding to just one eye, or possibly corresponding to some other vantage point (in industry, it is common to assume that the original camera position is half way between the final left and right eye stereo pair). We will cover 3D conversion in some detail in Chapter 3 - see also [59, Chapter 9] and [10, Chapters 12 and 14]. Although designed for 3D conversion of images or video, in the latter case Guidefill operates on a frame by frame basis, and therefore is most appropriately thought of as an image inpainting algorithm. Frame by frame inpainting, while adequate for some situations (for example, a foreground object occluding a static or slowly moving background), has an obvious disadvantage from the point of view temporal coherence. Since the frames are inpainted independently of one another, there is nothing to ensure that they vary smoothly when viewed through time. Near the end of the thesis (Chapter 7) we will explore spacetime transport, a generalization of Guidefill aimed at overcoming this. Spacetime transport is conceptually similar to Guidefill, but rather than inpainting each frame independently it treats the video as a whole.

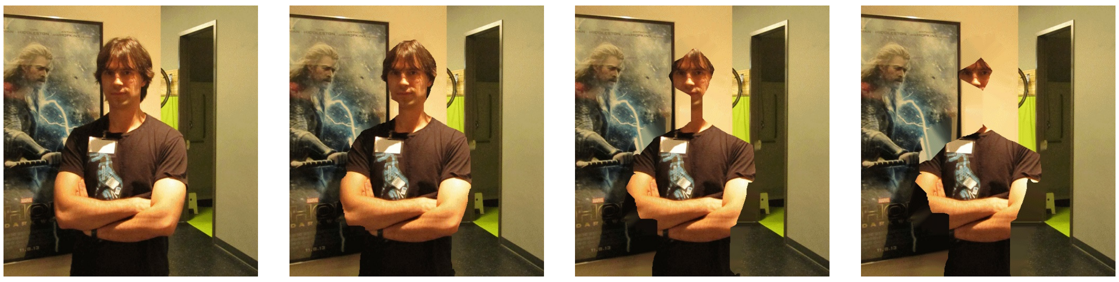
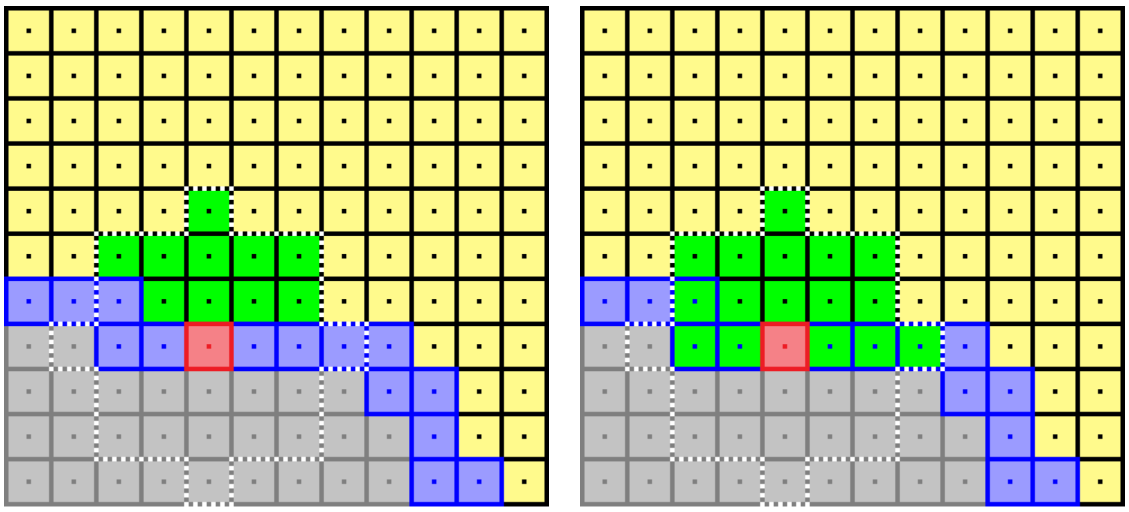
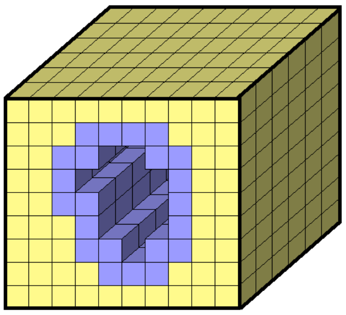

(a) Original image. (b) After filling 7 shells. (c) After filling 20 shells. (d) After filling 31 shells.

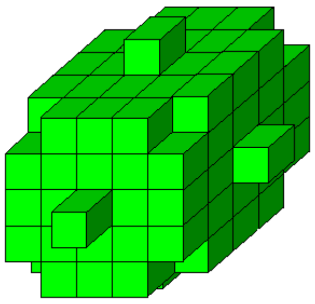
Figure 1.1: **Shell-based inpainting:** Here we illustrate the shell-based inpainting of an image including an undesirable human to be removed. In (a), we see the original image, including a human that is gradually eroded in (b)-(d) as we fill more and more shells. In this case the inpainting method is Guidefill [37], and the application is disocclusion for 3D conversion, which means that the human does not need to be removed entirely. Guidefill ensures isophotes are extrapolated in the correct direction by adapting its weights to the image content - a similar strategy to coherence transport [12] before it. See Chapter 3 of this thesis, [37], [59, Chapter 9], or [10, Chapters 12 and 14] for more details on this application.


(a) **The direct method:** the color of a given pixel (highlighted in red) on the current inpainting domain boundary (blue) is computed as a weighted sum of its already known neighbors in the filled portion of the image (pale yellow). Pixels included in the sum are highlighted in green.

(b) **The semi-implicit extension:** the color of a given pixel (highlighted in red) on the current inpainting domain boundary (blue) is given implicitly as a weighted sum of its already known neighbors in the filled portion of the image (pale yellow), as well as unknown neighboring pixels on the current boundary. Pixels included in the sum are highlighted in green.

Figure 1.2: **The direct method and its semi-implicit extension:** In this illustration, the filled portion of an image is highlighted in pale yellow, and the current inpainting domain is highlighted in both grey and blue, the former denoted pixels in the interior of the inpainting domain and the latter, pixels on its current boundary. The boundary is the "shell" that is currently being filled - note that the filled (known) portion of the image consists not only of the original undamaged region, but also of previously filled shells. At this stage, the grey and blue pixels are both unknown. In the direct method (a), the color of a given pixel (highlighted in red) is computed directly as a weighted average of pixels within a given neighborhood (outlined in white) that are already known. In the semi-implicit extension, the sum also includes pixels on the current boundary of the inpainting domain, but not pixels in its interior. This results in a linear system that needs to be solved, but this can be done relatively cheaply (see Section 6.2) and has benefits in terms of artifact reduction (see Section 6.6).


(a) An example video inpainting problem. Readable pixels are highlighted in yellow, the shell currently being filled (that is, the current boundary of the inpainting domain) is highlighted in blue, and the interior of the inpainting domain is transparent.


(b) An example 3D averaging neighborhood, in this case the 3D discrete solid ball of radius three voxels, which we denote by $\mathbf{B}_{\epsilon,h}(\mathbf{x})$ (note the bold).

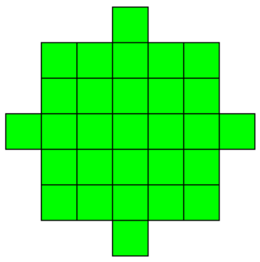

(c) The corresponding 2D averaging neighborhood, the 2D discrete ball of radius three pixels, which we denote by $B_{\epsilon,h}(\mathbf{x})$.

Figure 1.3: **Shell-based inpainting in 3D:** One approach to shell-based video inpainting is to inpaint each frame independently, as a sequence of image inpainting problems. Another approach, which we illustrate here, treats the video as a solid cube of voxels, with an inpainting domain that is a 3D solid. In (a), we illustrate an example inpainting problem with the readable voxels highlighted in yellow, the current boundary of the inpainting domain highlighted in blue, and the interior of the inpainting domain transparent. In any given iteration, all pixels in the current boundary are filled by taking a weighted average over their already filled neighbors (or those neighbors plus neighbors in the same shell, if we use the semi-implicit form) within a 3D neighborhood, such as the one shown in (b). For reference, we also show the corresponding 2D averaging neighborhood in (c). Although more expensive computationally and potentially much more memory intensive, this approach comes with significant advantages, including the improved temporal stability gained by averaging over a 3D region.

That is, the video is now treated as a solid cube of voxels, and both the inpainting domain and averaging neighborhood are now 3D solids. Each shell in the inpainting process now resembles a discretized two dimensional surface - see Figure 1.3 for an illustration. In addition to designing new algorithms, this thesis also studies the theoretical properties of this class of algorithms as a whole. This is done in Chapter 6, and represents the bulk of the work in this thesis. However, for the sake of simplicity, our analysis only covers image inpainting. Shell-based video inpainting in the sense of Figure 1.3 is something that we hope to explore theoretically in the future. In summary, this thesis therefore aims to do three things:

1. To introduce new algorithms within this class (Chapter 4 and Chapter 7).

2. To introduce 3D conversion as an application, and show how these methods may be adapted to this application (Chapter 3).

3. To develop a mathematical theory of these algorithms as a class, and prove general statements about them (Chapter 6).

One of the algorithms developed in this thesis, Guidefill (Chapter 4), has been accepted for publication in SIIMS (a preprint can be found here [37]). The other, spacetime transport, will be published but is not yet ready. The theory presented in Chapter 6 will also be published, and a preprint can be found here [36].

1.1 Background on Image and Video Inpainting

Image inpainting methods can loosely be categorized as *exemplar-based* and *geometric*. Exemplar-based methods generally operate by copying pieces of the undamaged portion of the image (typically small square patches) into the inpainting domain, in such a way as to make the result appear seamless. Examples include [22], [69], [5]. Exemplar-based methods also operate behind the scenes in Photoshop's famous *Content-Aware Fill* tool. These methods excel at interpolating texture across large gaps, but may produce artifacts in structure dominated images with strong edges, or be needlessly expensive if the inpainting domain is thin.

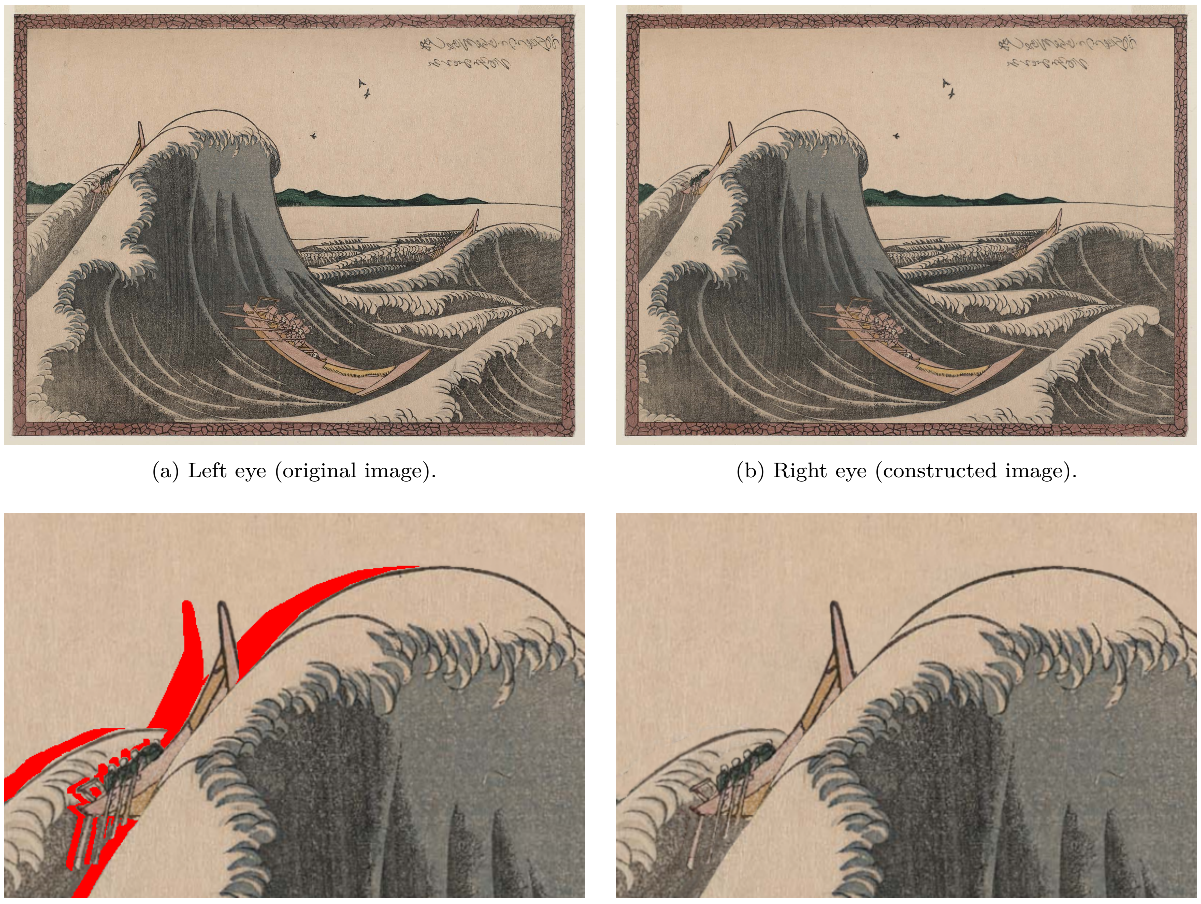
Geometric image inpainting methods aim to smoothly extend image structure into the inpainting domain, typically using partial differential equations or variational principles. Continuation may be achieved by either *interpolation* or *extrapolation*. Examples of methods based on interpolation include the seminal work of Bertalmio et al. [11], TV, TV-H^{-1}, Mumford-Shah, Cahn-Hilliard inpainting [18, 13], Euler's Elastica [46, 19], as well as the joint interpolation of image values and a guiding vector field in Ballester et al. [6]. These approaches are typically iterative and convergence is often slow, implying that such methods are usually not suitable for real-time applications. Telea's algorithm [64] and coherence transport [12, 44] (which can be thought of as an improvement of the former) are based on *extrapolation* and visit each pixel only once, filling them in order according to their distance from the boundary of the inpainting domain. These latter two methods belong to the class of inpainting methods considered in this thesis. Unlike their iterative counterparts, shell-based geometric methods have the advantage of being very fast.

Geometric methods are designed to propagate structure, but fail to reproduce texture. Similarly, exemplar-based approaches excel at reproducing texture, but are limited regarding their ability to propagate structure. A few attempts have been made at combining geometric and exemplar-based methods, such as Cao et al. [17], which gives impressive results but is relatively expensive.

Video inpainting adds an additional layer of complexity, because now temporal information is available, which is exploited by different algorithms in different ways. For example, when inpainting a moving object in the foreground, one can expect to find the missing information in nearby frames - this type of strategy is utilized in for example [38]. Another strategy is to generalize exemplar-based image inpainting methods to video by replacing 2D image patches with 3D spacetime cubes. This approach is taken in [49, 50], which also present a generalized patchmatch algorithm for video. While producing impressive results, this method is also very expensive, both in terms of runtime and memory requirements. Finally, the authors of [35] present a strategy for video inpainting of planar or almost-planar surfaces, based on inpainting a single frame and then propagating the result to neighboring frames using an estimated homography. To the best of our knowledge, no *geometric* video inpainting methods have been proposed in the literature.

Despite their fast runtime, simple methods such as the shell based geometric approach considered in this thesis have fallen somewhat out of fashion within the academic community in recent years, as sophisticated exemplar-based approaches have become faster. However, beyond their fast runtime, these methods carry a secondary advantage in terms of fast implementation time. This latter consideration, which tends to be undervalued in academia, has a significant impact on which algorithms get implemented in industry - in fact, it was while working in industry at the start of my PhD that I first encountered these methods. However, unless implemented carefully, they are known to produce disturbing visual artifacts that limit their applicability - we discuss these artifacts in detail in Section 2.2. While simple changes are often enough to eliminate or minimize these effects, without an analysis of *why* they occur in the first place, it is difficult to know which changes to make. More significantly, without careful analysis, it is not clear which artifacts *can* be eliminated and which are inevitable. In order to properly understand these issues, we feel that an in depth analysis is warranted. Therefore, one of the main objectives of this thesis is to fill a gap in the analysis of these methods with the aim of understanding the origins of the artifacts they produce and to what extent they are avoidable. In other words, our objective is to understand the limits of these algorithms in order to push them to their maximum potential.

1.2 The story of this PhD



(a) Left eye (original image).

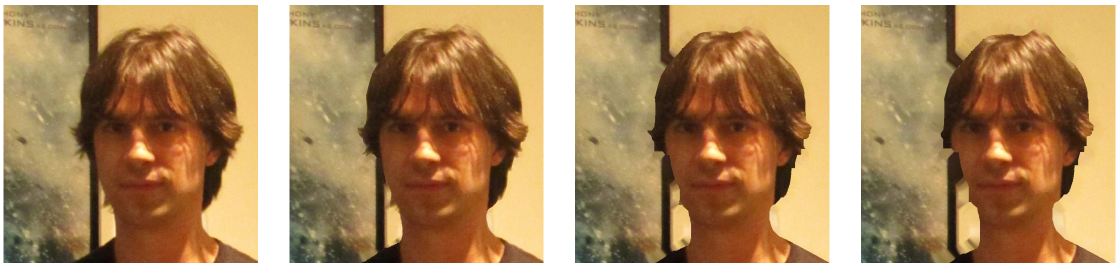
(b) Right eye (constructed image).

(c) Closeup of right eye view including cracks to be filled (red).

(d) Closeup of right eye view after cracks have been filled.

Figure 1.4: **3D conversion of a painting:** conversion of the painting *Oshiokuri hato tsuusen no zu* (1805), by the Japanese artist Hokusai (more famous for *Great Wave off Kanagawa*), into stereo 3D. The left eye view is the original painting, while the right eye view is synthetic, generated by myself using a pipeline analoguous to the one I encountered during my time working at Gener8 - details of this pipeline are provided in Chapter 3. The synthesized right eye view contains disoccuded areas not visible in the original painting that need to be inpainted - a few of these are shown in (c), while the same area post inpainting is shown in (d). Like the artists at Gener8, I did this using a variety of inpainting methods, combined with a considerable amount of touching up by hand for the more difficult areas. The result in anaglyph 3D is shown in Figure 1.6.

My PhD began rather unconventionally. After picking inpainting as PhD topic, but before doing any work on it, I was convinced by a friend to take a break in order to work with him at Gener8, a company specializing in the 3D conversion of Hollywood films. Examples of recent films converted in whole or in part by Gener8 include Maleficent, Thor, and Guardians of the Galaxy [1]. Inpainting comes up as one



(a) Original image. (b) After filling 10 shells. (c) After filling 20 shells. (d) After filling 30 shells.

Figure 1.5: **Kinking artifacts and Erodefill:** Inpainting a human using Erodefill (uniform weights and a 5×5 pixel box neighborhood). In this case, weights are not adapted to the image content and extrapolated isophotes kink in the direction of the normal to the inpainting domain boundary. Telea's algorithm [64], despite its somewhat more complex choice of weights, suffers from nearly identical problems.

step in a pipeline that also includes problems such as camera parameter estimation and segmentation. During my time at Gener8, inpainting was done by teams of artists who used a toolbox of algorithms to get a rough solution which they would then touch up by hand. This touching up was necessary because the algorithms available to the artists almost never yielded results meeting the high standards required for film. My job was to develop inpainting algorithms which, rather than giving these artists a perfect solution, instead provided them with another tool that would enable them to spend *less time* touching up. Guidefill [37], which was designed during my time at Gener8, was designed based on discussions with artists about the type of algorithm they would *like* to have.

Gener8 was also where I became familiar with the shell-based inpainting framework that is the subject of this thesis. The programmers there had implemented a series of shell-based image inpainting methods (equivalently, frame by frame video inpainting methods), all of which were essentially variants of Telea's algorithm [64] (which they were unaware of) in order to solve the inpainting problem arising in their 3D conversion pipeline. This type of algorithm was attractive to them because it is fast and simple to implement, while generally adequate for their application (where inpainting domains are typically very thin). The simplest of these methods, which they called “Erodefill”, was implemented by my friend Russell MacKenzie in an afternoon. It filled the inpainting domain on a frame by frame basis, assigning each pixel on the boundary of the inpainting domain a color equal to the average color of the already filled neighbors within a 5×5 pixel box. While sufficient for inpainting simple backgrounds of either uniform or slowly varying color, it was observed that Erodefill produced an odd “kinking” behaviour - isophotes extrapolated into the inpainting domain would change direction and become parallel to the normal vector of the inpainting domain boundary - regardless of their orientation outside of the inpainting domain (see Figure 1.5). My first assignment was to figure out why, and to modify Erodefill to eliminate this behaviour. In fact, as we will discuss in Section 2.3, this phenomenon had already been studied by Bornemann and März [12]. I had heard Tom März speak at Cambridge, so I was vaguely aware of their work, and I thought that completing my first assignment would amount to little more than downloading the code März had made available on the web. However, after a couple of test problems, it became clear that coherence transport wasn't free of kinking artifacts either.

Coherence transport, which will be discussed in more detail in the review of shell-based inpainting methods in Section 2.5, operates roughly as follows: A given pixel $\mathbf{x}$ on the current boundary of the inpainting domain is assigned a color equal to a weighted average of its already filled neighbors within a discrete ball of radius ϵ centered at $\mathbf{x}$. This ball is denoted by $B_{\epsilon,h}(\mathbf{x})$, and consists of those pixels distance at most ϵ from $\mathbf{x}$ (here h denotes the width of one pixel). An example with $\epsilon = 3h$ has been illustrated in Figure 1.3(c). In this, coherence transport is no different from Telea's algorithm [64] which came before

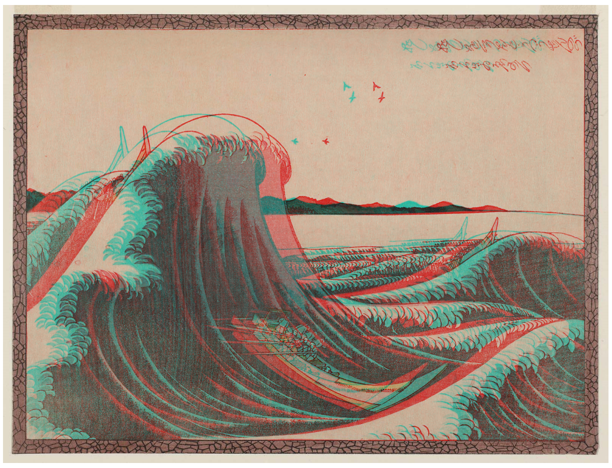


Figure 1.6

Figure 1.7: The 3D conversion problem from Figure 1.4 in anaglyph 3D (anaglyph 3D glasses required).

it. The key innovation of coherence transport is that if there are no edges in the undamaged region approaching $\mathbf{x}$, then the weights should correspond to a smooth blur, but if an edge with orientation $\mathbf{g}$ is approaching $\mathbf{x}$, then it should be extrapolated parallel to $\mathbf{g}$ into the inpainting domain. The idea is to accomplish this by combining a robust procedure for measuring $\mathbf{g}$ with carefully selected weights designed to favor extrapolation in the direction of $\mathbf{g}$.

However, computer experiments suggested that coherence transport could only successfully accomplish this when $\mathbf{g}$ was of the form $\mathbf{g} = \lambda(\mathbf{y} - \mathbf{x})$ for some $\mathbf{y} \in B_{\epsilon,h}(\mathbf{x})$ and some $\lambda \in \mathbb{R}$ (see Section 2.3 and Figure 2.6). Intuitively, it seemed that the problem might be resolved if the discrete ball $B_{\epsilon,h}(\mathbf{x})$ was replaced with a *rotated* discrete ball $\tilde{B}_{\epsilon,h}(\mathbf{x})$ which was constructed in such a way that $\mathbf{g} = \lambda(\mathbf{y} - \mathbf{x})$ for some $\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x})$ was *always* true - see Figure 2.9(b). However, since the elements of such a rotated ball would in general lie between pixel centers, it became necessary to define what I called “ghost pixels” - virtual pixels lying between pixel centers whose color was defined based on bilinear interpolation of their “real pixel” neighbors. This change did indeed appear to solve the problem, at least provided the angle between $\mathbf{g}$ and the inpainting domain boundary is not too shallow (Figure 2.6(c)). The desire to extend the method to handle even these very shallow angles eventually led to semi-implicit extension, which is able to resolve this issue (Section 6.6). Guidefill thus began as a simple modification of coherence transport incorporating the use of ghost pixels. However, the methods quickly diverged as more issues in need of resolution presented themselves (see Section 4.3.1) and as the desires of the artists became incorporated. For example, one of the main things the artists asked for was an intuitive interface for influencing the result of inpainting. This eventually led to a system where edges to be extrapolated are first represented as splines which the artist can adjust. Inpainting then proceeds once the artist has indicated they are happy with the splines. Eventually, it made more sense to think of Guidefill as a separate method.

When I left Gener8, it was still unclear to me why coherence transport created the kinking artifacts I had observed, or why the use of ghost pixels was able to resolve them. Neither of these things was explained by the mathematical theory presented by Bornemann and März in [12], which analyzed shell-based geometric inpainting methods by studying their behaviour in a high resolution and vanishing viscosity continuum limit, where they reduce to a transport partial differential equation (PDE). While I was still at Gener8, my colleagues and I disagreed over the reasons for this. The programmers, who were most comfortable with discrete mathematics and generally distrustful of PDEs and the continuum

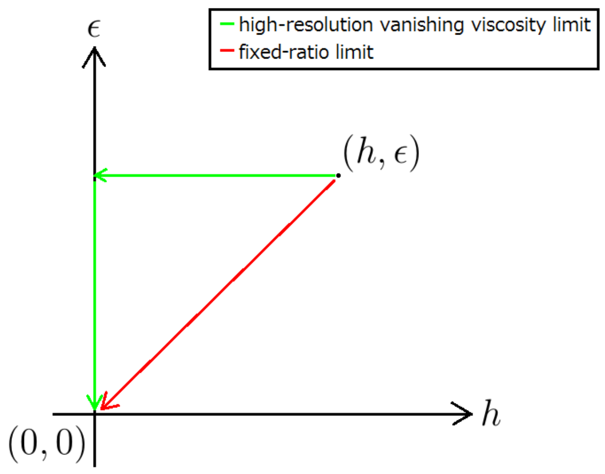


Figure 1.8: **Graphical distinction between two possible continuum limits of shell-based inpainting algorithms:** One way of understanding the shell-based framework studied in this thesis is via a continuum limit in which the pixel width h and averaging radius ϵ both go to 0, that is, $(h, \epsilon) \to (0, 0)$. However, this limit is not unique and exhibits path dependence. The path in green corresponds to the high-resolution and vanishing viscosity proposed by Bornemann and März [12], where first $h \to 0$ and then $\epsilon \to 0$. The path in red corresponds to a "fixed ratio" continuum limit studied in this thesis, where $(h, \epsilon) \to (0, 0)$ along the diagonal $\epsilon = rh$, where $r \in \mathbb{N}$ is a fixed integer representing the averaging radius measured in pixels.

in general, believed that it is because information is lost when one passes from the discrete world of real images into the idealized continuum. I was not convinced, and after my return to Cambridge I began to develop my own mathematical theory. Eventually I realized that the programmers were right, but also wrong. They were right because the failure of [12] was indeed due to a loss of information when one passes to the continuum. However, they were also wrong, because they falsely attributed this to an irreconcilable difference between the continuous and the discrete, rather than an accident of the *particular way* in which the limit in [12] is carried out. The reality is that all the kinking artifacts we had observed *can* be explained in the continuum, one just needs to take the limit differently.

Bornemann and März's continuum limit, which is based on first taking the pixel width $h \to 0$ and then the averaging radius $\epsilon \to 0$, is not unique. This should come as no surprise - one of the first things that second year calculus students are taught is that when taking a limit of the form $(x, y) \to (x_0, y_0)$, one should expect the result to depend on the *path* along which you approach (x_0, y_0). This is exactly what is happening here. Bornemann and März's original limit, which is illustrated as the green path in Figure 1.8, takes a different value from the limit approached along the red path, which represents an alternative "fixed ratio" limit studied in this thesis. This limit is so named because the ratio $r = \epsilon/h$ is fixed along this path.

Much of my PhD became an exploration of this alternative limit, which is able to explain a great deal about kinking artifacts and their resolution. However, this limit turned out still not to be the entire story, as it was inadequate for exploring blur artifacts. For this, one needs yet another limit - specifically, an *asymptotic limit* where the pixel width h is very small, but not yet 0. The systematic study of these two limits as well as their relationship to Bornemann and März's original limit eventually blossomed into the contents of Chapter 6. The takeaway messages are as follows:

- Bornemann and März's original continuum limit loses the most information, and is sufficient only for explaining some kinking artifacts.
- My alternative *fixed ratio* is in the middle - it retains enough information to uncover the fine

structure of kinking artifacts, but can only hint at the existence of blur artifacts.

- The asymptotic limit loses the least information, and is able to explain both kinking and blur artifacts with high accuracy.

One of the most interesting aspects of these three limits is their differing predictions regarding ghost pixels and the semi-implicit extension. Specifically, Bornemann and März's original limit predicts no difference at all between the use of coherence transport with the original discrete ball $B_{\epsilon,h}(\mathbf{x})$ and its rotated counterpart $\tilde{B}_{\epsilon,h}(\mathbf{x})$, as well as no difference at all between the direct method and the semi-implicit extension. The fixed ratio limit, on the other hand, predicts three distinct limits that behave very differently from the point of view of kinking artifacts (Section 6.6.2), and that are shown to be in excellent agreement with experiments (Section 6.8). However, the fixed ratio limit isn't precise enough to capture blur artifacts. In particular, there is some evidence in the literature (in the context of inpainting based on optical flow, which we discuss in Chapter 7) that replacing bilinear interpolation with higher order interpolation schemes can result in the reduction of blur [38, Section 4.5.1]. This suggests that blur may be reduced if the bilinear interpolation scheme used in the definition of ghost pixels is replaced with a higher order scheme. However, the fixed ratio limit is the same for *all* interpolation schemes satisfying a few properties that we list in Section 5.1, the most critical being that the interpolation scheme preserves polynomials of degree one (that is, the interpolant of a degree one polynomial relative to a given lattice is the polynomial back). To analyze the potential merits of redefining ghost pixels based on alternative interpolants, a new tool is needed, and the asymptotic limit seems a like a good candidate. However, this is beyond the scope of this thesis.

Chapter 6 provides a fairly comprehensive account of the kinking artifacts that arise in these methods and their resolution. This is done analytically using the fixed ratio continuum limit. At the same time, the asymptotic limit is able to make accurate predictions regarding blur artifacts, but does not currently help us to resolve them (Section 6.7). Unfortunately, despite its ability to make accurate predictions, we are not currently able to prove that the methods studied in this thesis converge to the asymptotic limit. The reasons for this are technical and are explained in Section 6.7. For the time being, convergence to the asymptotic limit remains a conjecture and the main piece of unfinished business in my PhD.

For the sake of fairness, I should mention that while the two limits I propose in this thesis as alternatives to Bornemann and März's high-resolution vanishing viscosity are able to capture fine behaviour that is missed by the latter, this comes at a price. The price is that to make our analysis tractable, we are forced to work within a far more restrictive setting than they do, with greater simplifying assumptions. Due to this, it is also true that, conversely, there are phenomena their model is able to explain which ours misses completely. In particular, the high-resolution vanishing viscosity limit provides a good theoretical framework for analyzing *shocks* (another type of artifact that we will discuss in Section 2.2), see in particular [45]. Our analysis, on the other hand, ignores this completely. Therefore, we see the different models as being complementary, each with their own strengths and weaknesses.

After Guidefill was in use for some time at Gener8, it became clear that the main limitation of the method was that when applied to video, it lacks temporal stability. That is, since frames are filled independently, there is nothing to ensure that pixel colors change smoothly over time, meaning that the results may appear to "jitter" when viewed through time. The final portion of my PhD was devoted to spacetime transport, a shell-based video inpainting algorithm that treats the video as a whole as in Figure 1.3, rather than inpainting frame by frame. The current "shell" of the inpainting domain becomes, essentially, a discretized 2D surface, while the "neighborhood" of a pixel that gets averaged over becomes a 3D solid. Spacetime transport is the subject of Chapter 7. However, this portion of the thesis is much less comprehensive than earlier sections, and no supporting theory is provided.

1.3 Organization and style

Here we describe the organization of the chapters, including the organization of the sections within each chapter. The chapters themselves do not have a section devoted to their internal organization. Some of the bigger chapters conclude with a section entitled "conclusions". Some of what is said there will be reiterated in an abridged form in the final chapter. A detailed discussion of future research directions is deferred until the final chapter, and so the body of the thesis will not discuss this beyond making the occasional remark.

Chapter 2 provides a detailed description of both the inpainting framework studied in this thesis as well as its limitations (including the artifacts it is known to produce), both of which we have already discussed informally in Chapter 1. Existing methods and related research are reviewed, and the difference between our analytical approach with earlier approaches is described clearly. The internal breakdown is as follows:

- The opening paragraphs give a quantitative description of the framework including pseudo code.
- Sections 2.1 and 2.2 describe the advantages and known problems (artifacts) of the framework.
- Section 2.3 provides a review of related work.
- Section 2.4 describes how our theoretical approach differs from previous approaches.
- Section 2.5 provides a description of the main inpainting methods within this framework.

Chapter 3 provides an overview of 3D conversion, with an emphasis on inpainting as a subproblem of the 3D conversion pipeline. Two possible pipelines are discussed, but the one arising in film, which is of primary interest in this thesis, is stressed. The reason for this is that Guidefill, which is discussed in Chapter 4, is designed for the film pipeline. It is not designed for the other pipeline (used mainly for 3D TV and other applications where the demands on visual quality are lower), which leads to an inpainting problem with different mathematical properties. A review is made of existing methods for inpainting as it applies to 3D conversion, however, it is unfortunately rather one-sided because Guidefill is as far as we know the only method in the literature explicitly designed for the film pipeline. The internal breakdown is as follows:

- Section 3.1 provides an overview of the two main 3D conversion pipelines.
- Section 3.2 discusses related work on inpainting for 3D conversion.
- Section 3.3 describes in detail the 3D conversion pipeline relevant to film. The other pipeline is also sketched in more detail, and some of its disadvantages are discussed.

Chapter 4 describes in detail Guidefill, an inpainting algorithm for 3D conversion of film that was designed as part of this thesis. The internal breakdown is as follows:

- Section 4.1 describes the design of Guidefill, the problem it aims to solve, and some of its properties.
- Section 4.2 gives a high-level overview of the different components of the algorithm.
- Sections 4.3, 4.4, 4.5, and 4.6 describe the components of the algorithm in detail. These are (respectively) the generation of the guide field, the use of ghost pixels for overcoming kinking artifacts, Guidefill's pixel ordering strategy (smart order), and a description of some GPU implementation considerations. Section 4.3.1 also discusses an issue that arises in coherence transport from the use of the modified structure tensor.

- Section 4.7 gives a proof of the convergence of coherence transport and Guidefill to the fixed ratio continuum limit proposed in this thesis (under some simplifying assumptions) and discusses the consequences of this result. This is a preview of the much more general (and more difficult) convergence proof in Chapter 6.
- Section 4.8 analyzes the time complexity and processor complexity of Guidefill as a parallel algorithm.
- Section 4.9 contains numerical experiments, aimed both at demonstrating the viability of Guidefill as an inpainting method for 3D conversion, and at validating the complexity analysis of the previous section.
- Section 4.10 draws some conclusions.

Chapter 5 describes in detail the semi-implicit form of the algorithm and introduces the concept of equivalent weights. The internal breakdown is as follows:

- Section 5.1 explains how to convert between weighted sums of ghost pixels and equivalent sums over real pixels with modified weights, and establishes a list of properties satisfied by these modified weights.
- Section 5.2 uses the result of the previous section to derive an explicit formula for the linear system arising in the semi-implicit extension, which is shown to be strictly diagonally dominant. A simple iterative method is proposed for its solution, which is proven to be equivalent to damped Jacobi or successive over-relaxation (SOR).
- Section 5.2.2 discusses the semi-implicit form of Guidefill, introduces some changes to its pixel ordering strategy, and provides numerical evidence that this form of the algorithm alleviates certain kinking artifacts present in the direct form.

Chapter 6 is the longest chapter and contains most of our analysis. The internal breakdown is as follows:

- Section 6.1 describes the simplifying assumptions under which our subsequent analysis takes place.
- Section 6.2 derives sharp bounds on the convergence rates of damped Jacobi and SOR for semi-implicit Guidefill. SOR is shown to converge extremely quickly.
- Sections 6.3 and 6.4 are the longest and most difficult portion of the thesis. It is here that convergence of the class of inpainting methods under scrutiny to our proposed fixed ratio limit is proven. Bounds on rates of convergence are also given, which are later shown experimentally to be sharp in most situations (Appendix A.10).
- Section 6.5 fills a gap in the literature by also proving convergence (taking the limit a different way) to März and Bornemann's original high-resolution vanishing viscosity limit. Conditions are derived under which the two limits coincide.
- Section 6.6 uses the results from Sections 6.3 and 6.4 to prove results about kinking artifacts. First, a fundamental distinction is proven between the direct and semi-implicit forms of the class of algorithm under study. Next, in Section 6.6.2 the continuum limits of coherence transport, Guidefill, and semi-implicit Guidefill are derived. Kinking behaviour (or lack thereof) is then studied for each algorithm.

- Section 6.7 introduces the asymptotic limit, which is used to explain and predict blur artifacts. Convergence to the asymptotic limit is left as a conjecture, but a sketch of the proof is given, with an explanation of where the technical problems come up and how they might be resolved.
- Section 6.8 contains numerical experiments backing up the theoretical results of this chapter. Additional experiments are given in Appendices A.9 and A.10.
- Section 6.9 summarizes and draws some conclusions.

Chapter 7 introduces spacetime transport, a shell-based video inpainting method that is inspired by Guidefill, but rather than inpainting frame by frame, treats the video as a 3D solid of voxels. Spacetime transport is shown to have elements in common with the shell-based image inpainting methods we have seen so far, as well as optical flow based video inpainting. However, a significant difference between spacetime transport and Guidefill is that in the former, both the guide field and the video are inpainted based on two separate shell-based approaches. In the case of Guidefill, the computation of the guide field is done based on splines and is not shell-based. The internal breakdown is as follows:

- Section 7.1 introduces optical flow based video inpainting, and provides a review of some optical flow based algorithms. Sections 7.1.1 and 7.1.2 note parallels with two dimensional shell based image inpainting, and discuss the known issue of blur in optical flow based algorithms.
- Sections 7.2 and 7.3 describe spacetime transport in detail, explaining how the components of Guidefill generalize to three dimensions (for example, ghost pixels become ghost voxels) and providing pseudo code. The Section 7.4 briefly discusses voxel ordering strategies for both stages of inpainting, while Section 7.5 goes over some implementation considerations.
- Section 7.6 makes a connection between spacetime transport and Lucas–Kanade optical flow.
- Section 7.7 shows by way of examples how kinking and blur artifacts generalize to 3D. It is shown that the same strategies that worked in 2D for resolving kinking artifacts also work in 3D. Blur artifacts are shown to persist and are possibly a more serious problem than before. In this section ground truth values are given for the guide field.
- Section 7.8 provides a few examples of the full spacetime transport algorithm, where the guide field is not given and must be computed. Some of the additional difficulties this entails are discussed.
- Section 7.9 summarizes and draws some conclusions.

Chapter 8 summarizes the findings of the thesis, discusses open questions, and sketches possible directions for future research. The internal breakdown is as follows:

- Section 8.1 summarizes the thesis and sketches directions for future research.
- Section 8.2 singles out the three most promising future research directions and discusses them in more detail.

Appendices: This thesis also contains a number of appendices. These include additional numerical results and implementation details that were considered too disruptive or not important enough to include in the main text, as well as details of proofs that were either too long, too tangentially related to the main ideas, or simply not interesting enough to justify including in the main text. The breakdown is as follows:

- Appendix A.1 contains further justification for Remark 2.5.1.
- Appendix A.2 contains additional details of GPU implementation of Guidefill.
- Appendix A.3 contains additional discussion of the algorithmic complexity of Guidefill.
- Appendix A.4 contains 3D conversion results from Guidefill in anaglyph 3D, which certain readers (in possession of a pair of anaglyph 3D glasses) may find interesting.
- Appendix A.5 briefly discusses GPU architecture in more detail.
- Appendix A.6 contains proofs of the properties of ghost pixels and equivalent weights that we listed in Section 5.1.
- Appendix A.7 contains the proof of Theorem 6.5.1, which proves convergence to März and Bornemann's original high-resolution vanishing viscosity limit.
- Appendix A.8 derives an explicit formula for the kinking behaviour of coherence transport in terms of an object we call the "angular spectrum" of a discrete set, which is defined.
- Appendix A.9 contains a list of the examples used in the numerical experiments given in Section 6.8.
- Appendix A.10 contains a series of additional numerical experiments testing the tightness of bounds on the rates of convergence to the fixed ratio limit given in Theorem 6.3.1.

1.4 Notation

- $h =$ the width of one pixel.
- $\mathbb{Z}_h^2 := \{(nh, mh) : (n,m) \in \mathbb{Z}^2\}$.
- Given $\mathbf{v} \in \mathbb{R}^2$, we denote by $L_{\mathbf{v}} := \{\lambda \mathbf{v} : \lambda \in \mathbb{R}\}$ the line through the origin parallel to $\mathbf{v}$.
- Given $\mathbf{x} \in \mathbb{R}^2$, we denote by $\theta(\mathbf{x}) \in [0, \pi)$ the counter-clockwise angle $L_{\mathbf{x}}$ makes with the x-axis. $\theta(\mathbf{x})$ can also be thought of as the counterclockwise angle $\mathbf{x}$ makes with the x-axis, modulo π.
- Given $\mathbf{v} \in \mathbb{R}^2$, we denote by $\mathbf{v}^\perp$ the counterclockwise rotation of $\mathbf{v}$ by $90°$.
- $\Omega = [a,b] \times [c,d]$ and $\Omega_h = \Omega \cap \mathbb{Z}_h^2$ are the continuous and discrete image domains.
- $D_h = D_h^{(0)} \subset \Omega_h$ is the (initial) discrete inpainting domain.
- $D_h^{(k)} \subseteq D_h^{(0)}$ is the discrete inpainting domain on step k of the algorithm.
- $B_h \subset \Omega_h \backslash D_h$ is the set of "bystander pixels" (defined in Section 3.3.1) that are neither inpainted nor used for inpainting.
- $D \subset \Omega := \{\mathbf{x} \in \Omega : \exists \mathbf{y} \in D_h \text{ s.t. } \|\mathbf{y} - \mathbf{x}\|_\infty < h\}$ is the continuous inpainting domain.
- $D^{(k)}$ is the continuous inpainting domain on step k of the algorithm, defined in the same way as D.
- $B \subset \Omega$ is the continuous bystander set, defined in terms of B_h in the same way as D.
- $u_0 : \Omega_h \backslash D_h \to \mathbb{R}^d$ is the given (discrete) image. In an abuse of notation, we also use u_0 to denote an assumed underlying continuous image $u_0 : \Omega \backslash D \to \mathbb{R}^d$.

- $u_h : \Omega_h \to \mathbb{R}^d$ is the inpainted completion of u_0.
- $\mathbf{g}(\mathbf{x})$ is the guidance vector field used by coherence transport and Guidefill.
- $B_\epsilon(\mathbf{x})$ the solid ball of radius ϵ centered at $\mathbf{x}$.
- $A_{\epsilon,h}(\mathbf{x}) \subset B_\epsilon(\mathbf{x})$ denotes a generic discrete (but not necessarily lattice aligned) neighborhood of radius ϵ surrounding the pixel $\mathbf{x}$ and used for inpainting.
- $\mathrm{Supp}(A_{\epsilon,h}(\mathbf{x})) \subset \mathbb{Z}^2_h$ denotes the set of real pixels needed to define $A_{\epsilon,h}(\mathbf{x})$ biased on bilinear interpolation.
- $B_{\epsilon,h}(\mathbf{x}) = \{\mathbf{y} \in \Omega_h : \|\mathbf{x} - \mathbf{y}\| \leq \epsilon\}$, the discrete ball of radius ϵ centerd at $\mathbf{x}$ and the choice of $A_{\epsilon,h}(\mathbf{x})$ used by coherence transport.
- $r = \epsilon/h$ the radius of $B_{\epsilon,h}(\mathbf{x})$ measured in pixels.
- $\tilde{B}_{\epsilon,h}(\mathbf{x}) = R(B_{\epsilon,h}(\mathbf{x}))$, where R is the rotation matrix taking $(0,1)$ to $\mathbf{g}(\mathbf{x})$, the choice of $A_{\epsilon,h}(\mathbf{x})$ used by Guidefill.
- $\mathcal{N}(\mathbf{x}) = \{\mathbf{x} + \mathbf{y} : \mathbf{y} \in \{-h, 0, h\} \times \{-h, 0, h\}\}$ is the 9-point neighborhood of $\mathbf{x}$.
- Given $A_h \subset \mathbb{Z}^2_h$, we define the dilation of A_h by

$$D_h(A_h) = \cup_{\mathbf{x} \in A_h} \mathcal{N}(\mathbf{x}).$$

  If $h = 1$ we write D instead of D_1.
- Given $A_h \subset \mathbb{Z}^2_h$, we define the discrete (inner) boundary of A_h by

$$\partial A_h := \{\mathbf{x} \in A_h : \mathcal{N}(\mathbf{x}) \cap \mathbb{Z}^2_h \backslash A_h \neq \emptyset\}.$$

  For convenience we typically drop the word “inner” and refer to ∂A_h as just the boundary of A_h.
- O denotes the zero matrix.
- Given $c \in \mathbb{R}$, we define $\{y \leq c\} := \{(x, y) \in \mathbb{R}^2 : y \leq c\}$.
- Given x, $y \in \mathbb{R}$, we define $x \wedge y := \min(x, y)$.

1.4.1 3D-specifc notation

Here we list notation specific to the 3D case considered in Chapter 7. In some cases, such as $B_{\epsilon,h}(\mathbf{x})$ versus $\mathbf{B}_{\epsilon,h}(\mathbf{x})$, we use bold to explicitly distinguish a 3D object from its 2D counterpart. In other cases, however, we rely on the reader to work out based on context what we mean. This should not be difficult as all of the 3D content is in one chapter.

- $\mathbb{Z}^3_h := \{(nh, mh, lh) : (n, m, l) \in \mathbb{Z}^3\}$.
- $\Omega = [a, b] \times [c, d] \times [e, f]$ and $\Omega_h = \Omega \cap \mathbb{Z}^3_h$ are the continuous and discrete video domains.
- $D_h = D_h^{(0)} \subset \Omega_h$ is the (initial) discrete inpainting domain.
- $u_0 : \Omega_h \backslash D_h \to \mathbb{R}^d$ is the given (discrete) video.
- $u_h : \Omega_h \to \mathbb{R}^d$ is the inpainted completion of u_0.

- $G(\mathbf{x}) : D_h \to \mathbb{R}^{3\times 3}$ is the guide field of symmetric positive semi-definite matrices used by spacetime transport.
- $\mathbf{B}_\epsilon(\mathbf{x})$ the solid ball 3D ball of radius ϵ centered at $\mathbf{x}$.
- $\mathbf{A}_{\epsilon,h}(\mathbf{x}) \subset \mathbf{B}_\epsilon(\mathbf{x})$ denotes a generic discrete (but not necessarily lattice aligned) 3D neighborhood of radius ϵ surrounding the voxels $\mathbf{x}$ and used for inpainting.
- $\mathbf{B}_{\epsilon,h}(\mathbf{x}) = \{\mathbf{y} \in \Omega_h : \|\mathbf{x} - \mathbf{y}\| \leq \epsilon\}$, the discrete 3D ball of radius ϵ centerd at $\mathbf{x}$.
- $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ and $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ denote the eigenvalues and eigenvectors of G.
- $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x}) = R(B_{\epsilon,h}(\mathbf{x}))$, where $R = [\mathbf{v}_1\mathbf{v}_2\mathbf{v_3}]$ is the rotation matrix aligning the coordinate vectors e_1, e_2, e_3 with the eigenvectors of G.
- $\mathcal{N}(\mathbf{x}) = \{\mathbf{x} + \mathbf{y} : \mathbf{y} \in \{-h, 0, h\} \times \{-h, 0, h\} \times \{-h, 0, h\}\}$ is the 27-point neighborhood of $\mathbf{x}$.
- Given $A_h \subset \mathbb{Z}_h^3$, we define the discrete boundary of A_h by

$$\partial A_h := \{\mathbf{x} \in A_h : \mathcal{N}(\mathbf{x}) \cap \mathbb{Z}_h^3 \backslash A_h \neq \emptyset\}.$$

Chapter 2

Framework and Motivation

In this chapter we describe in detail our general framework, as well as its advantages and disadvantages, for the case of image inpainting (equivalently, frame by frame video inpainting). We will not return to video inpainting in the sense of Figure 1.3 until Chapter 7. As already stated, the class of methods under scrutiny in this thesis fill the inpainting domain in successive shells from the boundary inwards, as was illustrated in Figure 1.1. In the direct form of the method, the color of a given pixel $\mathbf{x}$ due to be filled is computed as a weighted average of its already filled neighbors within a discrete neighborhood $A_{\epsilon,h}(\mathbf{x}) \subset B_\epsilon(\mathbf{x})$. In the semi-implicit extension (which is the subject of Chapter 5), this sum also includes unknown pixels within the current shell, resulting in a linear system (Figure 1.2). We will cover the resulting linear system in detail in Section 5.2, where we also propose an iterative method for its solution, the convergence of which is analyzed in Section 6.2. The direct method as well as this proposed iterative solution to the semi-implicit extension are illustrated in Algorithm 1 with pseudo code (the blue code indicates the parts relevant to the semi-implicit extension). The neighborhood $A_{\epsilon,h}(\mathbf{x})$ need not be axis aligned and may contain "ghost pixels" lying between pixel centers - see Figure 2.1 for an illustration. As explained in Section 1.2, ghost pixels are helpful for reducing kinking artifacts (see Figure 2.4), and the color of a given ghost pixel is defined as the bilinear interpolation of its four real pixel neighbors, but is undefined if one or more of them has not yet been assigned a color. We denote by $\text{Supp}(A_{\epsilon,h}(\mathbf{x})) \subset \mathbb{Z}_h^2$ the set of real pixels needed to define $A_{\epsilon,h}(\mathbf{x})$ in this way. Here h and ϵ denote respectively the width of one pixel and the radius of the bounding disk $B_\epsilon(\mathbf{x}) \supset A_{\epsilon,h}(\mathbf{x})$. The averaging weights w_ϵ are non-negative and are allowed to depend on $\mathbf{x}$, but must scale proportionally with the size of the neighborhood $A_{\epsilon,h}(\mathbf{x})$, like so:

$$w_\epsilon(\mathbf{x},\mathbf{y}) = \hat{w}\left(\mathbf{x}, \frac{\mathbf{y}-\mathbf{x}}{\epsilon}\right) \tag{2.0.1}$$

for some function $\hat{w}(\cdot,\cdot) : \Omega \times B_1(\mathbf{0}) \to [0,\infty]$. Note that we will sometimes write w_r or w_1 in place of w_ϵ - in this case we mean (2.0.1) with ϵ replaced by r or 1 in the denominator on the right hand side. As the algorithm proceeds, the inpainting domain shrinks, generating a sequence of inpainting domains $D_h = D_h^{(0)} \supset D_h^{(1)} \supset \ldots \supset D_h^{(K)} = \emptyset$. We will assume the non-degeneracy condition

$$\sum_{\mathbf{y}\in A_{\epsilon,h}(\mathbf{x})\cap(\Omega\setminus D^{(k)})} w_\epsilon(\mathbf{x},\mathbf{y}) > 0 \tag{2.0.2}$$

holds at all times, this ensures that the average (2.0.3) in Algorithm 1 is always well defined. One trivial way of ensuring this is by having strictly positive weights $\hat{w}$, which all the methods considered do (see Section 2.5). At iteration k, only pixels belonging to the current boundary $\partial D_h^{(k)}$ are filled, but moreover we fill only a subset $\partial_{\text{ready}} D_h^{(k)} \subseteq \partial D_h^{(k)}$ of pixels deemed to be "ready" (In Section 2.5 we will review the main methods in the literature and give their “ready” functions).

Algorithm 1 Shell Based Geometric Inpainting

```
$u_h$ = damaged image, initialized to 0 on inpainting domain.
$\Omega = [a,b] \times [c,d]$ = continuous image domain.
$\Omega_h = \Omega \cap \mathbb{Z}_h^2$ = discrete image domain.
$D_h^{(0)}$ = initial inpainting domain.
$\partial D_h^{(0)}$= initial inpainting domain boundary.
semiImplicit = false, unless we use the semi implicit extension (Section 5.2).
for $k = 0, \ldots$ do
    if $D_h^{(k)} = \emptyset$ then
        break
    end if
    $\partial_{\text{ready}} D_h^{(k)} = \{\mathbf{x} \in \partial D_h^{(k)} : \text{ready}(\mathbf{x})\}$
    $u_h = \textsc{FillBoundary}(u_h,\, D_h^{(k)},\, \partial_{\text{ready}} D_h^{(k)})$
    $D_h^{(k+1)} = D_h^{(k)} \backslash \partial_{\text{ready}} D_h^{(k)}$
    if semiImplicit then
        $u_h^{(0)} = u_h$
        for $n = 1, 2, \ldots$ (until convergence) do
            $u_h^{(n)} = \textsc{FillBoundary}(u_h^{(n-1)},\, D_h^{(k+1)},\, \partial_{\text{ready}} D_h^{(k)})$
        end for
    end if
    $\partial D_h^{(k+1)} = \{\mathbf{x} \in \partial D_h^{(k+1)} : \mathcal{N}(\mathbf{x}) \cap (\Omega_h \backslash D_h^{(k+1)}) \neq \emptyset\}$.
end for

function $u_h = \textsc{FillBoundary}(u_h,\, D_h, \partial D_h)$
    for $\mathbf{x} \in \partial D_h$ do
        compute $A_{\epsilon,h}(\mathbf{x})$ = neighborhood of $\mathbf{x}$.
        compute non-negative weights $w_\epsilon(\mathbf{x},\mathbf{y}) \geq 0$ for $A_{\epsilon,h}(\mathbf{x})$.
        if ready($\mathbf{x}$) then
            $$u_h(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in (A_{\epsilon,h}(\mathbf{x}) \backslash \{\mathbf{x}\}) \cap (\Omega \backslash D)} w_\epsilon(\mathbf{x},\mathbf{y}) u_h(\mathbf{y})}{\sum_{\mathbf{y} \in (A_{\epsilon,h}(\mathbf{x}) \backslash \{\mathbf{x}\}) \cap (\Omega \backslash D)} w_\epsilon(\mathbf{x},\mathbf{y})} \qquad (2.0.3)$$
        end if
    end for
end function
```

See (4.5.3) for a definition of the ready function for Guidefill. Coherence transport and Guidefill use the neighborhoods $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$, $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ respectively - see Figure 2.9. They also both use the same weights (2.5.2). Blue text is only relevant for the semi-implicit extension we introduce in Section 5.2.

The main inpainting methods in the literature of the general form given by Algorithm 1 are (in chronological order) Telea's Algorithm [64], coherence transport [12], coherence transport with adapted distance functions [44], and Guidefill [37]. As we will see, these methods essentially differ only in the choice of weights (2.0.1), the choice of fill order as dictated by the “ready” function, and the choice of neighborhood $A_{\epsilon,h}(\mathbf{x})$. We will review these methods in Section 2.5.

Remark 2.0.1. *It is worth mentioning that this class of algorithms is nearly exactly the same as the “generic single-pass algorithm” first systematically studied by Bornemann and März in [12]. The two main differences are*

1. *They assume $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$, while we allow for greater flexibility.*

2. *They consider only the direct form of Algorithm 1, not the semi-implicit extension.*

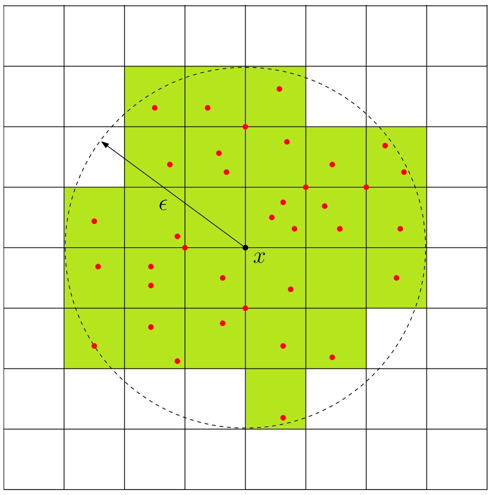


Figure 2.1: **Illustration of a generic set $A_{\epsilon,h}(\mathbf{x})$ containing ghost pixels:** In this illustration the overlaid grid is the lattice $\mathbb{Z}_h^2$ with pixel centers at its vertices. The elements of a generic $A_{\epsilon,h}(\mathbf{x})$ are represented as red dots - they do not need to occupy pixel centers, but they must all lie within distance ϵ of $\mathbf{x}$. Ghost pixels are defined based on bilinear interpolation of their real pixel neighbors. Here we have highlighted in green the squares whose vertices are the real pixels needed to define the colors of the ghost pixels in $A_{\epsilon,h}(\mathbf{x})$. We call this set of real pixels $\text{Supp}(A_{\epsilon,h}(\mathbf{x}))$. Note that while $A_{\epsilon,h}(\mathbf{x}) \subset B_\epsilon(\mathbf{x})$, this inclusion is not in general true of $\text{Supp}(A_{\epsilon,h}(\mathbf{x}))$.

Beyond this, they also phrase things in terms of a pre-determined fill order, rather than a "ready" function, but the former may easily be seen, mathematically at least, to be a special case of the latter (Section 2.5).

Remark 2.0.2. *When Algorithm 1 is adapted for 3D conversion (as it is in the case of Guidefill, which we cover in Chapter 4) it must be modified to include the use of a bystander set $B_h \subseteq \Omega_h \backslash D_h$ of pixels that are neither inpainted nor used for inpainting. The bystander set B_h is explained in Chapter 3, which covers 3D conversion, in Section 3.3.1. The net effect is that the update formula* (2.0.3) *must be modified slightly to*

$$u_h(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in (A_{\epsilon,h}(\mathbf{x}) \backslash \{\mathbf{x}\}) \cap (\Omega \backslash (D \cup B))} w_\epsilon(\mathbf{x},\mathbf{y}) u_h(\mathbf{y})}{\sum_{\mathbf{y} \in (A_{\epsilon,h}(\mathbf{x}) \backslash \{\mathbf{x}\}) \cap (\Omega \backslash (D \cup B))} w_\epsilon(\mathbf{x},\mathbf{y})} \tag{2.0.4}$$

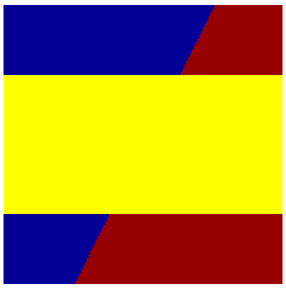
where B is the continuous bystander set defined in Section 1.4. However, for the sake of simplicity, in this thesis outside of Chapters 3 and 4 we assume $B_h = \emptyset$.

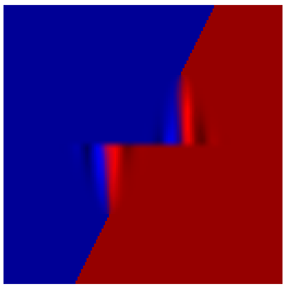
2.1 Advantages of Algorithm 1

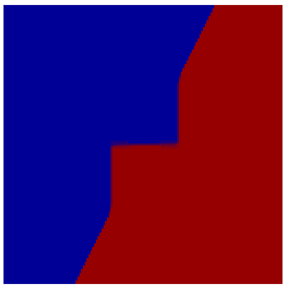
The main appeal of Algorithm 1 is its simplicity and parallelizability, which enable it to run very fast. A second advantage, first noted by Bornemann and März [12], is the stability property

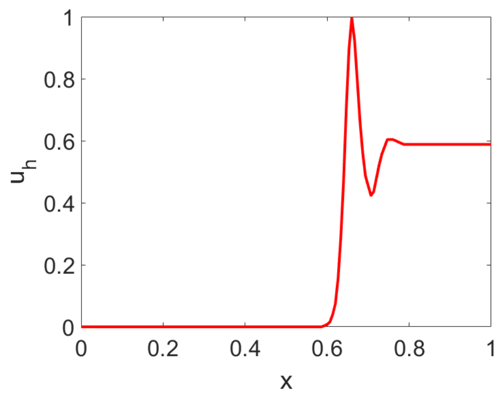
$$\min_{\mathbf{y} \in B} u_0(\mathbf{y}) \leq u_h(\mathbf{x}) \leq \max_{\mathbf{y} \in B} u_0(\mathbf{y}) \qquad \text{for all } \mathbf{x} \in D_h, \tag{2.1.1}$$

(which holds channelwise) where $u_0 : \Omega_h \backslash D_h \to \mathbb{R}^d$ is the given image and B is the band of width ϵ pixels surrounding ∂D_h. This property holds because we have chosen non-negative weights summing to one.

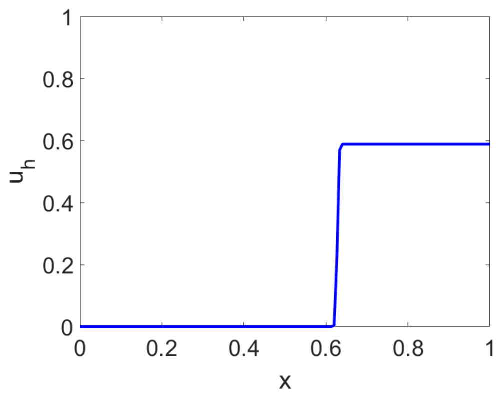

(a) Inpainting problem (inpainting domain in yellow).


(b) Inpainting with Telea's algorithm.


(c) Inpainting with coherence transport.



(d) Red channel cross-section for Telea's algorithm.



(e) Red channel cross-section for coherence transport.

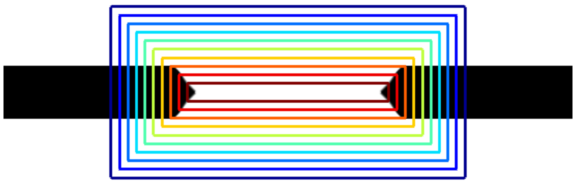
Figure 2.2: **Instabilities in Telea's algorithm:** In this example we consider the inpainting problem shown in (a) consisting of a line separating a region of dark blue from a region of dark red. We inpaint both using Telea's algorithm (b) and coherence transport (c). Coherence transport obeys the stability property (2.1.1) and hence the brightness of the inpainted solution remains bounded above by the brightness on the exterior of the inpainting domain. This is not true of Telea's algorithm, which exhibits bright spots outside the the original color range. These are not visible in Figure 2.4, because the brightness of each color channel is already saturated, and Telea's algorithm uses clamping to prevent the solution from going outside the admissible color range. This is further illustrated in (d)-(e), where we plot horizontal cross sections of the red channel of each inpainted solution. These slices are located ten rows of pixels above the midpoint of the inpainting domain.

Remark 2.1.1. *Although we have presented Telea's algorithm [64] as an example of Algorithm 1, this is not strictly true as its update formula* (2.5.1) *(see Section 2.5) contains a gradient term that, after it has been approximated with finite differences, effectively violates the rule of non-negative weights summing to one. This means that Telea's algorithm does not satisfy the stability property* (2.1.1)*, as we illustrate in Figure 2.2.*

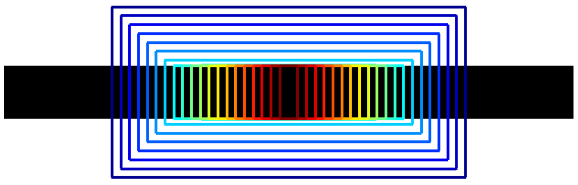
2.2 Disadvantages and artifacts

The main disadvantage of Algorithm 1 is obviously loss of texture, and in cases where texture is important, these methods should not be used. However, beyond loss of texture, inpainting methods of the general form given in Algorithm 1 can also introduce a host of of other artifacts, which we list below.

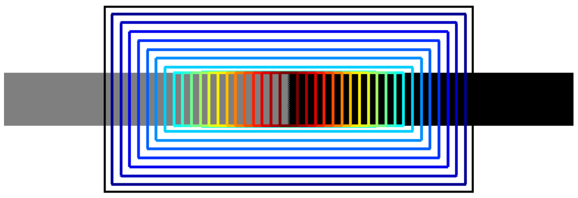
- **"kinking" of isophotes** where extrapolated isophotes change direction at the boundary of the inpainting domain - see Figure 2.4 and Figure 2.5.



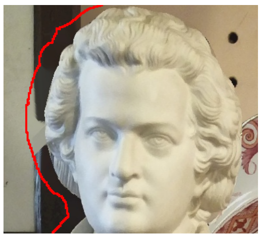
(a) Coherence transport with default onion shell ordering. Isophotes are cut off.



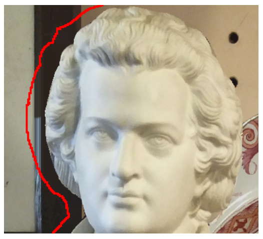
(b) Guidefill with smart pixel ordering is able to make a successful connection (März's adapted distance functions [44] would also do the job).



(c) Guidefill's smart pixel ordering is not able to prevent a shock in this case because of incompatible boundary conditions.



(d) Guidefill with onion shell ordering results in a cut off picture frame that ends abruptly.



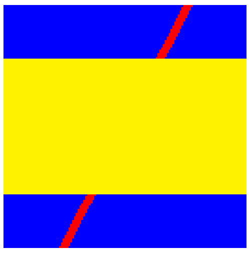
(e) Guidefill with smart ordering connects the picture frame, but creates a shock.

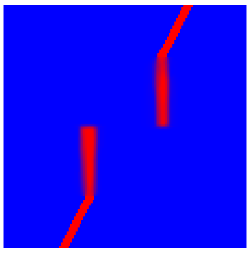
Figure 2.3: **Cut off isophotes and shocks:** Because Algorithm 1 fills the inpainting domain from many directions at once, "cut off isophotes" or shocks can sometimes be formed. In (a), this is due to the (superimposed) fill order, which is the default onion shell ordering and a bad choice in this case. In (b), we have a chosen a new fill order better adapted to the image and the problem is solved in this case. However, the shock in (c) is due to incompatible boundary conditions and it is unlikely any special fill order could solve the problem. If (c) seems a little contrived, consider the "real life" examples (d)-(e). In (d), we inpainted using Guidefill with the default onion shell ordering, resulting in the picture frame being cut off. In (e) we used Guidefill's build in smart order, which successfully completes the picture frame, but creates a shock in the middle. This shock is due to incompatible lighting conditions at either end of the inpainting domain, which is outlined in red.

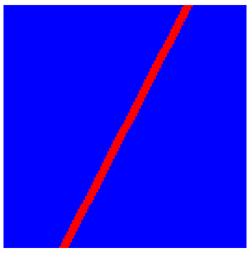
- **"blurring" of isophotes** where edges that are sharp in the undamaged region may blur when extended into the inpainting domain - see Figure 2.4 and Figure 2.5.
- **"cutting off" of isophotes** where isophotes extrapolated into the inpainting domain end abruptly - see Figure 2.3.
- **formation of shocks** where sharp discontinuities may form in the inpainting domain - see Figure 2.3.
- **bright or dark spots** that are only a problem if the stability condition (2.1.1) is violated, as it is for Telea's algorithm. See Figure 2.2 and Figure 2.5.

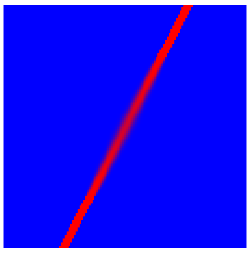
2.3 Related work

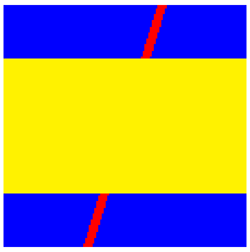
Here we discuss briefly discuss related work on this class of methods, both in terms of the progression of newer methods reducing the artifacts discussed in the previous section, and mathematical analysis. We include Guidefill in the discussion even thought it was developed as part of this thesis. This is because it fits into the discussion very naturally, and failing to do so would detract from the readers understanding of the issues that motivate our theoretical work in Chapter 6.

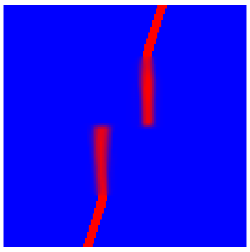

(a) Inpainting problem with $\theta = 63°$.

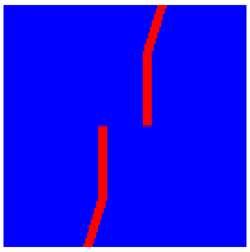

(b) Telea's algorithm.

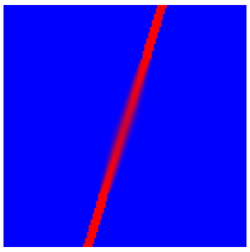

(c) coherence transport.

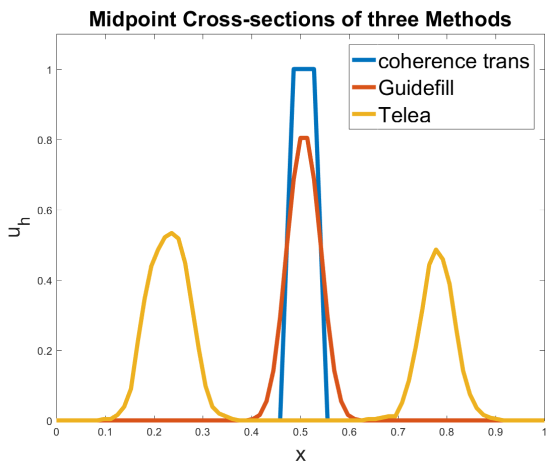

(d) Guidefill.


(e) Inpainting problem with $\theta = 73°$.

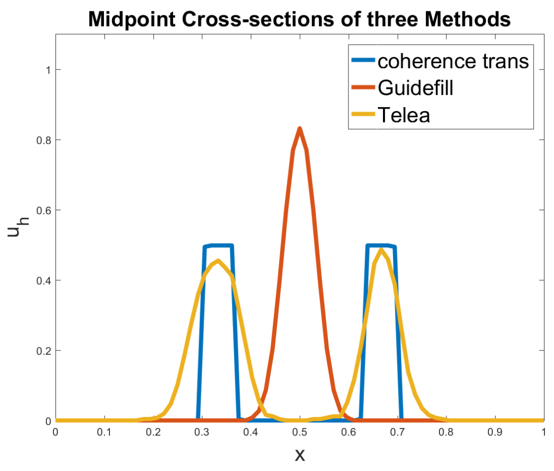

(f) Telea's algorithm.


(g) coherence transport.


(h) Guidefill.



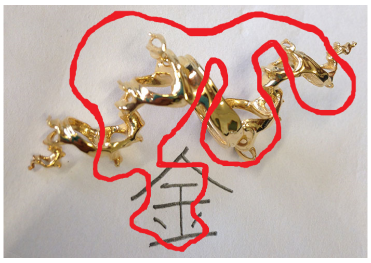
(i) Midpoint cross-sections for $\theta = 63°$.

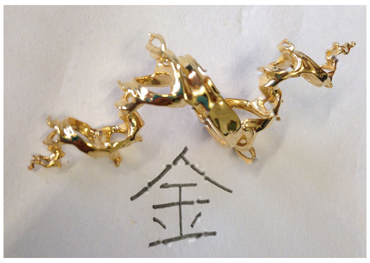


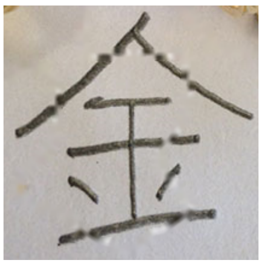
(j) Midpoint cross-sections for $\theta = 73°$.

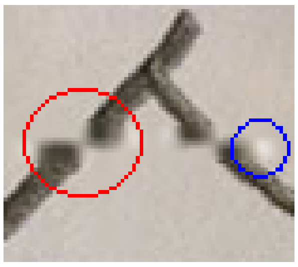
Figure 2.4: **A tale of two inpainting problems:** In (a)-(d), a line making an angle of $\theta = 63°$ with the horizontol is inpainted using each of Telea's algorithm [64], coherence transport, [12, 44], and Guidefill [37] (the inpainting domain is shown in yellow). In this case the radius of $A_{\epsilon,h}(\mathbf{x})$ is $\epsilon = 3$px, and since $63° \approx \arctan(2) \approx 63.44°$ is close to one of the "special directions" in which coherence transport can extend isophotes successfully for this value of ϵ (see Figure 2.6), both coherence transport and Guidefill make a successful connection. In (e)-(h) we change the angle of the line slightly to $\theta = 73°$. This isn't one of coherence transport's admissable directions for $\epsilon = 3$px, so it fails to make the connection, while Guidefill continues to have no problems, at the expense of some blurring. Telea's algorithm, on the other hand, propagates in the direction of the normal to the inpainting domain boundary regardless of the undamaged image content, and thus fails to make the connection in both cases while also introducing significant blur. In (i)-(j), we examine horizontal cross sections (of the red channel) of all three methods at the midpoint of the inpainting domain. Here, a disadvantage of Guidefill in terms of blur becomes more apparent - coherence transport by contrast produces a much sharper result. The reasons for this are explored in Section 6.7.

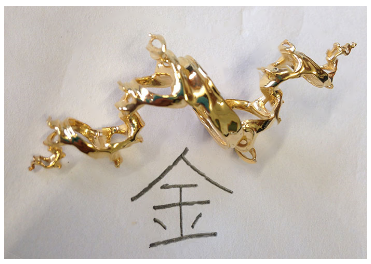
2.3.1 Artifact reduction


(a) Damaged image with inpainting domain in red.


(b) The result of Telea's algorithm [64].


(c) Closeup of (b). Note the bright spots and disconnected isophotes.


(d) Further closeup of (b), with blurry, disconnected isophotes circled in red and a bright spot circled in blue.


(e) The result of coherence transport [12].

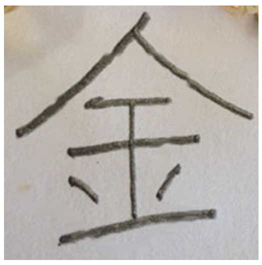
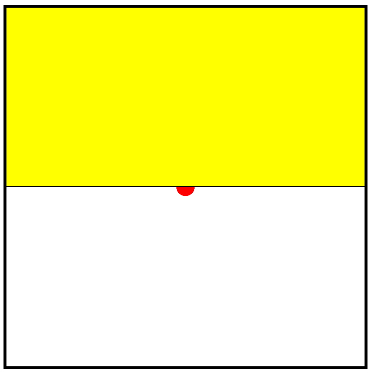
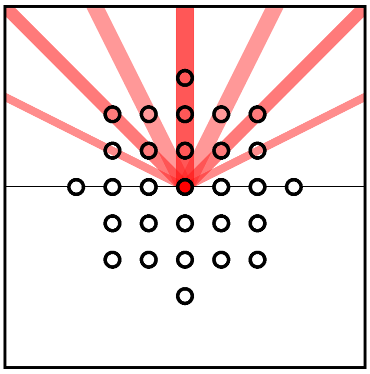

(f) Closeup of (d) - note the better reconstruction of 金.

Figure 2.5: **Blurring, kinking, and bright spots with Telea's algorithm:** Even for problems such as (a) where the inpainting domain is very thin, Telea's algorithm (b)-(d) still creates strong blurring artifacts and fails to connect isophotes effectively. Also, due to the presence of the gradient term in (2.5.1), Telea's algorithm violates the stability condition (2.1.1) and as a result can "overshoot" when filling pixels close to edges in the filled area, where the (numerical) gradient changes rapidly. This leads to the bright spots near the reconstructed 金 in (c)-(d). In this case coherence transport (e)-(f) is a much better choice.

Broadly speaking, there has been incremental progress as follows: Telea's algorithm [64], the earliest variant to appear in the literature, suffers from strong artifacts of every type. In particular, the weights make no attempt to take into account the orientation of undamaged isophotes in $\Omega_h \backslash D_h$, and the result shows strong kinking artifacts (see Figure 2.4). Bornemann and März identified and sought to address this problem with coherence transport [12], which proposed carefully chosen weights that are proven (in a high resolution and vanishing viscosity limit) to extend isophotes in any desired guidance direction $\mathbf{g}$ not parallel to the inpainting domain boundary. This was combined with a method aimed at robustly measuring the orientation of isophotes at the boundary, so that a suitable $\mathbf{g}$ allowing for a seamless transition could be found. The problem of "kinking" ostensibly resolved, in a follow up work März proposed coherence transport with adapted distance functions [44] designed to minimize the problem of "cut off" isophotes and shocks. This was accomplished by recognizing that artifacts such as the incomplete line in Figure 2.3(a) are often the byproduct of a suboptimal fill order such as the one superimposed (in this case the default onion shell ordering). The situation can often be corrected as in Figure 2.3(b), by using an ordering better adapted to the image such as the one illustrated there. Rather than filling pixels in an order proportional to their distance from the boundary, i.e. having the ready function in Algorithm 1 always return "true", März proposed a number of ways of generating improved orderings based on non-Euclidean distance from boundary maps. At the same time, recognizing that the presence


(a) Inpainting problem with D_h colored yellow and outlined in black.


(b) Superposition of multiple inpaintings of (a) using coherence transport with guidance direction $\mathbf{g}$ sweeping out an arc from 1° up to 179°. In this case $\epsilon = 3$px and $B_{\epsilon,h}(\mathbf{0})$ is superimposed.

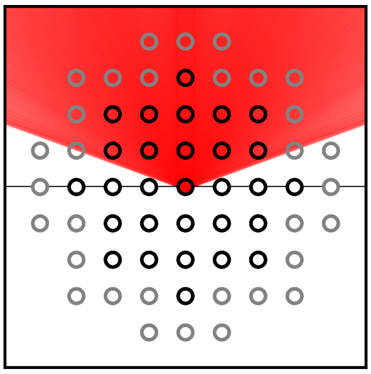
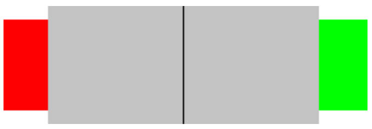

(c) Now Guidefill is used instead of coherence transport, but all parameters including $\epsilon = 3$px and $\mu = 100$ are kept the same. This time we superimpose the dilated ball $D_h(B_{\epsilon,h}(\mathbf{0}))$. New points are shown in grey.

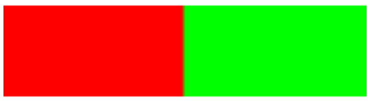
Figure 2.6: **Special directions:** For a given guidance direction $\mathbf{g} = (\cos\theta, \sin\theta)$, coherence transport [12, 44] can successfully extrapolate isophotes parallel to $\mathbf{g}$ only if $\mathbf{g} = \lambda\mathbf{v}$, for some $\mathbf{v} \in B_{\epsilon,h}(\mathbf{0})$. This is illustrated in (b), where have solved the inpainting problem posed in (a) multiple times using coherence transport with $\epsilon = 3$px with a sequence of guidance directions $\mathbf{g}_k = (\cos\theta_k, \sin\theta_k)$ (with θ_k ranging from 1° up to 179° in increments of one degree), combined the results (specifically, we took a weighted average of all 179 solutions in which the weight of a given pixel decreases exponentially as its color moves away from pure red), and superimposed $B_{\epsilon,h}(\mathbf{0})$ (the parameter μ in (2.5.2) is $\mu = 100$). Instead of a smoothly varying line sweeping through the upper half plane and filling it with red, we see a superposition of finitely many lines, each passing through some $\mathbf{v} \in B_{\epsilon,h}(\mathbf{0})$. When we repeat the experiment in (c) using Guidefill [37], we see that it is not free of problems either. In this case Guidefill can extrapolate along $\mathbf{g} = (\cos\theta, \sin\theta)$ so long as $0 < \theta_c \leq \theta \leq \pi - \theta_c < \pi$, where θ_c is a critical angle, and we get a red cone bounded on either side by θ_c. Here we have superimposed the dilated ball $D_h(B_{\epsilon,h}(\mathbf{0}))$, and it is evident that θ_c is in some way related to this dilation - this will be explained in Section 6.6.1.

of shocks was related to the "stopping set" [44] of the distance map, März was able to exert some measure of control over those as well, if not prevent them entirely. Guidefill [37] brought the focus back to the reduction of kinking artifacts, by noting that coherence transport is actually only able to propagate along a given guidance vector $\mathbf{g}$ if it points in one of a finite set of special directions - see Figure 2.6(b).

Whereas previous improvements to Algorithm 1 had focused first on improving the choice of weights, then the fill order (equivalently the choice of ready function), Guidefill proposed for the first time to change the averaging neighborhood $A_{\epsilon,h}(\mathbf{x})$, which until now had always been the discrete ball $B_{\epsilon,h}(\mathbf{x})$ (Figure 2.9(a)). Specifically, it proposed to replace $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$ with $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$, where $\tilde{B}_{\epsilon,h}(\mathbf{x})$ is the *rotated* discrete ball shown in Figure 2.9(b), aligned with the guidance direction $\mathbf{g}$. Since $A_{\epsilon,h}(\mathbf{x})$ is in this case no longer axis aligned, it contains what the authors called "ghost pixels" lying between pixel centers, which they defined based on bilinear interpolation. This small change enabled Guidefill to propagate along most guidance directions, but it too has problems when the angle between $\mathbf{g}$ and the boundary to the inpainting domain is too shallow - see Figure 2.6(c). However, Guidefill pays a price for its reduction in kinking artifacts in the form of an increase in blur artifacts. See Figure 2.4, where coherence transport produces a sharp extension of image isophotes, albeit possibly in the wrong direction, whereas Guidefill extrapolates in the right direction, but the extrapolation suffers from blur. Guidefill also proposed its own "smart order" computed on the fly as an alternative to März's adapted distance functions, but this does not have any serious advantage in terms of the quality of the results. Either approach will do for preventing "cut off" isophotes.

2.3.2 Related theoretical work


(a) An inpainting problem with incompatible boundary conditions. The inpainting domain D_h is in grey, and the skeleton Σ is drawn in black.


(b) Inpainting using Guidefill. A shock is formed on the skeleton set Σ shown in (a).

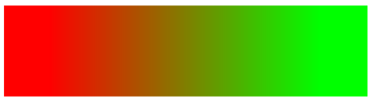

(c) Inpainting by solving the second order elliptic equation $-\epsilon\Delta u + u_x = 0$ with $\epsilon = 10^{-7}$. Shocks are prevented, but the solution (using GMRES) becomes much more expensive.

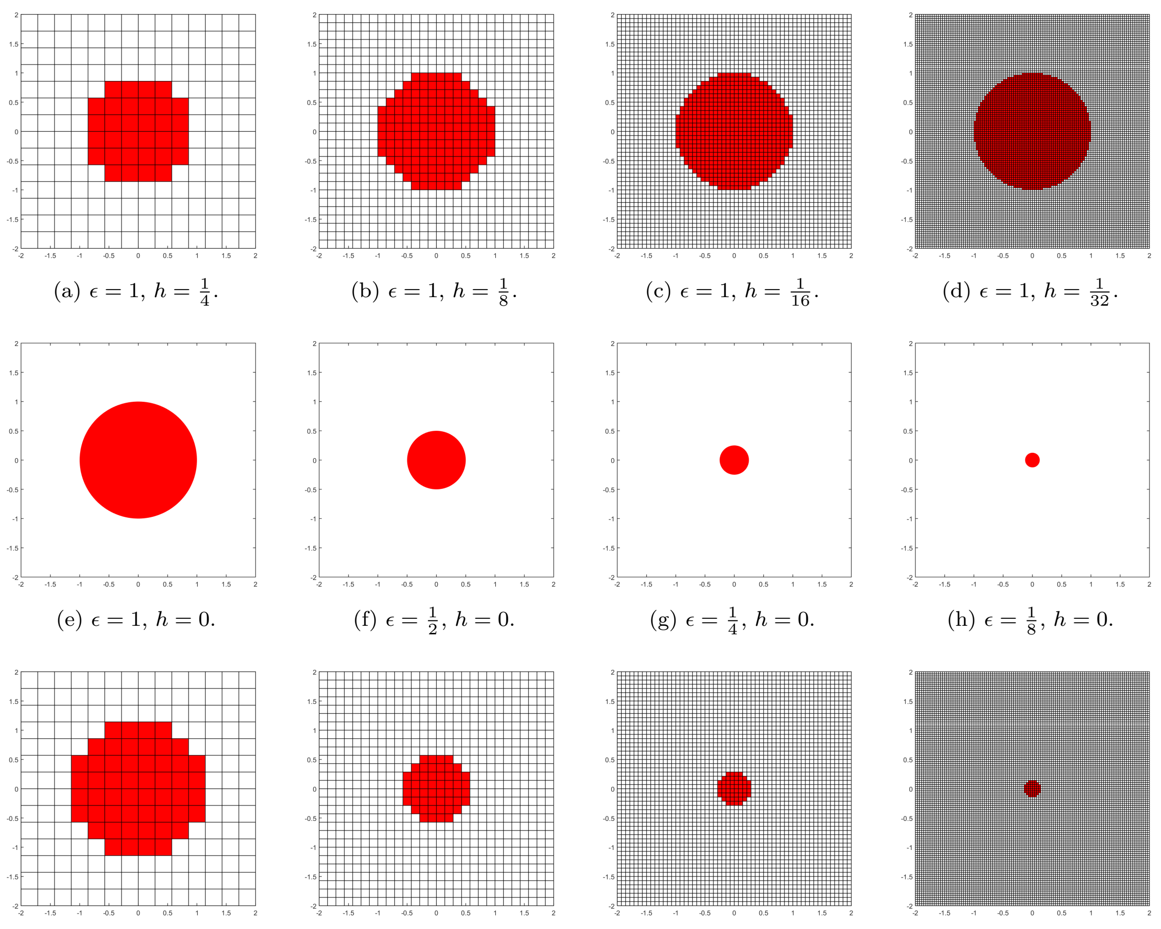
Figure 2.7: **Creation of shocks by Algorithm 1:** When Algorithm 1 is used to inpaint problems with incompatible boundary conditions, such as the problem illustrated in (a) of inpainting a stripe that is red on one end and green on the other, the result may contain shocks as in (b). These shocks can be understood by adopting the framework proposed in [12, 45], where the output of Algorithm 1 under a high resolution and vanishing viscosity limit is shown to be equivalent to the solution of a first order transport equation on $D\backslash\Sigma$, where Σ is a set of measure zero containing any potential shocks. Ballester et al. [6] suggested overcoming this problem by adding a diffusive term $-\epsilon\Delta u$ to the transport equation and taking $\epsilon \to 0$. As can be seen in (c), in this case the formation of shocks is prevented, but the algorithm becomes much more expensive, in this case requiring several minutes for GMRES [56] to converge on a 200×200px inpainting domain (Guidefill by contrast took only 60ms).

Algorithm 1 has been studied by Bornemann and März in high-resolution and vanishing viscosity double limit where first $h \to 0$ and then $\epsilon \to 0$. They study the direct form of Algorithm 1, but their analysis applies equally well to the semi-implicit form, and the limit obtained is identical. Under this limit, Algorithm 1 becomes a first order (possibly non-linear) transport equation boundary value problem

$$\nabla u(\mathbf{x}) \cdot \mathbf{c}(\mathbf{x}, u) = 0 \text{ for } \mathbf{x} \in D\backslash\Sigma, \quad u\Big|_{\partial D} = u_0\Big|_{\partial D}, \tag{2.3.1}$$

where u_0 denotes the given data in the undamaged region, and Σ is a set of measure zero related to a distance map prescribing the order in which pixels are filled. The above equation is linear ($\mathbf{c}$ independent of u) if the weights w_ϵ are independent of the inpainted solution u_h (as is the case in Guidefill, for example, see Chapter 4) - or *nonlinear* ($\mathbf{c}$ depends on u) if w_ϵ is allowed to depend on u_h (as it does in coherence transport). This transport equation is then analyzed from two perspectives. Firstly, its well-posedness (which is non-trivial, due to the unusual boundary conditions) is studied [12, 45], where it is shown that (2.3.1) is well posed on $D\backslash\Sigma$, but not on the bigger set D. This analysis is then related to cut off isophotes and shocks as illustrated in Figure 2.3, which are shown to occur within the exceptional set Σ. These ideas also play a role in coherence transport with adapted distance functions, which aims to control the position of Σ within D via a suitable manipulation of the fill order in order to minimize these artifacts.

Bornemann and März appear to be the first to perform an in depth study the well-posedness of (2.3.1) and its relationship to the formation of shocks. However, this problem was arguably anticipated by Ballester et al. [6], who considered both the joint interpolation of image values and a guiding vector field, as well as the propagation of image values along a known vector field. In the latter case, they noted that their approach is equivalent to (2.3.1) with $\Sigma = \emptyset$ and $\mathbf{c}$ independent of u, which they note does not have an obvious solution. Indeed, the integral curves of $\mathbf{c}(\mathbf{x})$, each with a beginning and endpoint on ∂D, may have incompatible values of u_0 at those endpoints (indeed, this is exactly the reason for the formation of shocks). To resolve this issue they suggested, among other things, adding a diffusive term $-\epsilon\Delta u$ to (2.3.1) to make it well posed, and then taking $\epsilon \to 0$. This approach not only resolves the well-posedness, it also prevents the formation of shocks, as illustrated in Figure 2.7(c), where we solve the resulting nonsymmetric linear system with $\epsilon = 10^{-7}$ using GMRES (the Generalized Minimum RESidual method for nonsymmetric linear systems) [56]. However, the resulting increase in runtime is substantial.



(a) $\epsilon = 1$, $h = \frac{1}{4}$.
(b) $\epsilon = 1$, $h = \frac{1}{8}$.
(c) $\epsilon = 1$, $h = \frac{1}{16}$.
(d) $\epsilon = 1$, $h = \frac{1}{32}$.

(e) $\epsilon = 1$, $h = 0$.
(f) $\epsilon = \frac{1}{2}$, $h = 0$.
(g) $\epsilon = \frac{1}{4}$, $h = 0$.
(h) $\epsilon = \frac{1}{8}$, $h = 0$.

(i) $\epsilon = 1$, $h = \frac{1}{4}$.
(j) $\epsilon = \frac{1}{2}$, $h = \frac{1}{8}$.
(k) $\epsilon = \frac{1}{4}$, $h = \frac{1}{16}$.
(l) $\epsilon = \frac{1}{8}$, $h = \frac{1}{32}$.

Figure 2.8: **Two distinct continuum limits:** In this thesis we study two separate continuum limits of Algorithm 1, illustrated here for $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$. The first, illustrated in (a)-(h), is the high-resolution vanishing viscosity double limit proposed by Bornemann and März [12], in which $h \to 0$ (a)-(d) and then $\epsilon \to 0$ (e)-(h). The second is the fixed-ratio limit single limit $(\epsilon, h) \to (0, 0)$ with $r = \frac{\epsilon}{h}$ fixed proposed in our previous work [37], illustrated in (i)-(l) for $r = 4$. We will prove that while they are both valid limits of Algorithm 1, they predict very different behaviour. Moreover, we will see that the predictions of the latter correspond much more closely than the former to the actual behaviour of Algorithm 1, especially when r is small (which it typically is in applications - [12] recommends r between 3 and 5, for example). See Theorems 6.3.1 and 6.5.1, as well as Sections 6.6.2 and 6.8.2.

In addition to their well-posedness analysis, Bornemann and März also used the continuum model (2.3.1) for studying kinking artifacts. In particular, they showed that for Telea's algorithm, the resulting transport direction $\mathbf{c}(\mathbf{x})$ is always normal to the boundary of the inpainting domain, while for coherence transport, it can be made to point parallel to the guideance direction $\mathbf{g}$, so long as the latter is not parallel to ∂D. As we have seen in Figure 2.6 and Figure 2.4, the former conclusion is consistent with numerical experiments, but the latter is not. While useful, something is missing in Bornemann and März's continuum limit.

2.4 An alternative continuum limit

One of the key ideas of this thesis is to instead study Algorithm 1 through a fixed ratio continuum limit in which $(h, \epsilon) \to (0, 0)$ along the ray $\epsilon = rh$. The non-negative integer r is simply the radius of $A_{\epsilon,h}(\mathbf{x})$ measured in pixels and will play a key role in our analysis. In words, our limit takes $h \to 0$ while keeping the size of the neighborhood $A_{\epsilon,h}(\mathbf{x})$ constant, as measured in pixels. Figure 2.8 illustrates the difference between our fixed ratio limit and Bornemann and März's high-resolution vanishing viscosity limit. Although both are perfectly valid mathematically, our fixed ratio limit captures fine details in the behaviour of Algorithm 1 that the other limit glosses over. For example, Bornemann and März's continuum limit is identical for coherence transport, Guidefill, and the semi-implicit extension of Guidefill, whereas we obtain three distinct limits. Their limit predicts no kinking artifacts for any of these algorithms, unless the guidance direction $\mathbf{g}$ is exactly parallel to ∂D, whereas our limit predicts that this is only true of semi-implicit Guidefill. Our limit depends on r, and we will see that as $r \to \infty$, we recover Bornemann and März's original limit. The fine behaviour uncovered by our limit is thus most significant when r is a small integer.

To uncover even finer behaviour, in this thesis we also consider an asymptotic limit in which $h/\epsilon = r$ is constant, and h is small but positive. This will be critical for analyzing blur artifacts (Chapter 6, Section 6.7).

2.4.1 Motivation for ghost pixels.

In Section 5.1 we will prove that any weighted sum over a set $A_{\epsilon,h}(\mathbf{x})$ of ghost pixels is equivalent to a sum over the real pixels in $\text{Supp}(A_{\epsilon,h}(\mathbf{x}))$ with equivalent weights. While this makes ghost pixels in some sense redundant, they are useful concept. Specifically, in Theorem 6.3.1 we will prove that the fixed ratio continuum limit described above and illustrated in Figure 2.8 is a partial differential equation with coefficients that depend continuously on the weights w_ϵ and on the elements of $A_{\epsilon,h}(\mathbf{x})$. It will be desirable to control this limit by making suitable choices for w_ϵ and $A_{\epsilon,h}(\mathbf{x})$, and this is easier if the elements of the latter may be varied continuously.

2.5 Review of main methods

Here we briefly review the main inpainting methods of the general form sketched in Algorithm 1.

Telea's algorithm. The earliest algorithm (to our knowledge) appearing in the literature and of the form sketched in Algorithm 1, Telea's algorithm [64] is also the only such algorithm to use a different formula for $u_h(\mathbf{x})$ than the expression (2.0.3) appearing in Algorithm 1 (see Remark 2.1.1). Instead of computing $u_h(\mathbf{x})$ a weighted average of $u_h(\mathbf{y})$ evaluated at nearby already filled pixels $\mathbf{y}$, it takes a weighted average of the *predictions* that each of these pixels makes, based on linear extrapolation, for $u_h(\mathbf{x})$. That is,

$$u_h(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in B_{\epsilon,h}(\mathbf{x}) \cap (\Omega_h \backslash D_h^{(k)})} w_\epsilon(\mathbf{x}, \mathbf{y})(u_h(\mathbf{y}) + \nabla_h u_h(\mathbf{y}) \cdot (\mathbf{x} - \mathbf{y}))}{\sum_{\mathbf{y} \in B_{\epsilon,h}(\mathbf{x}) \cap (\Omega_h \backslash D_h^{(k)})} w_\epsilon(\mathbf{x}, \mathbf{y})}, \tag{2.5.1}$$

where $\nabla_h u_h(\mathbf{y})$ denotes the centered difference approximation to the gradient of u_h at $\mathbf{y}$, that is

$$\nabla_h u_h(\mathbf{y}) := \frac{1}{2}\left(u_h(\mathbf{y} + e_1) - u_h(\mathbf{y} - e_1), u_h(\mathbf{y} + e_2) - u_h(\mathbf{y} - e_2)\right).$$

As we have already noted in Remark 2.1.1, this approach has a disadvantage in that it results in the loss of the stability property (2.1.1). Moreover, the "predictions"

$$u_{\text{predicted}}(\mathbf{x}) := u_h(\mathbf{y}) + \nabla_h u_h(\mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})$$

can become highly inaccurate when $\mathbf{y}$ is on an edge $\Omega_h \backslash D_h^{(k)}$, leading to significant over or undershooting, visible as bright or dark spots as in Figure 2.2 and Figure 2.5. Perhaps in recognition of this, the gradient term was dropped from (2.5.1) in all subsequent algorithms. The weights in this case are

$$w_\epsilon(\mathbf{x}, \mathbf{y}) = \text{dir}(\mathbf{x}, \mathbf{y}) \cdot \text{dst}(\mathbf{x}, \mathbf{y}) \cdot \text{lev}(\mathbf{x}, \mathbf{y}),$$

where

$$\text{dir}(\mathbf{x}, \mathbf{y}) := \frac{\mathbf{x} - \mathbf{y}}{\|\mathbf{x} - \mathbf{y}\|} \cdot N(\mathbf{x}), \ \text{dst}(\mathbf{x}, \mathbf{y}) := \frac{d_0^2}{\|\mathbf{x} - \mathbf{y}\|^2}, \ \text{lev}(\mathbf{x}, \mathbf{y}) := \frac{T_0}{1 + |T(\mathbf{y}) - T(\mathbf{x})|},$$

and $T(\mathbf{x})$ denotes the Euclidean distance from $\mathbf{x}$ to the (original) boundary of the inpainting domain, and $N(\mathbf{x}) = \nabla_h T(\mathbf{x})$ (estimated based on central differences). T is precomputed using the fast marching method. Telea's algorithm uses the default onion shell ordering, that is "ready$(\mathbf{x}) \equiv$ true".

Coherence transport. Coherence transport [12] improves upon Telea's algorithm by adapting the weights in order to encourage extrapolation of isophotes in the direction of their tangent. This is done by calculating a "local coherence direction" $\mathbf{g}(\mathbf{x})$ in terms of a modified structure tensor. Coherence transport calculates the color of a given pixel to be filled using the formula (2.0.3) in Algorithm 1 with weights

$$w_\epsilon(\mathbf{x}, \mathbf{y}) = \frac{1}{\|\mathbf{y} - \mathbf{x}\|} \exp\left(-\frac{\mu^2}{2\epsilon^2}(\mathbf{g}^\perp(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}))^2\right), \tag{2.5.2}$$

and with $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$ - see Figure 2.9(a) and Figure 2.9(c). Like Telea's algorithm, coherence transport uses the default onion shell ordering, that is "ready$(\mathbf{x}) \equiv$ true".

Coherence transport with adapted distance functions. In a subsequent work [44], März made improvements to coherence transport by replacing the default onion shell ordering with one based on a variety of non-Euclidean distance functions. One such distance function defines an "active boundary" $\Gamma_h \subseteq \partial D_h$ defined by

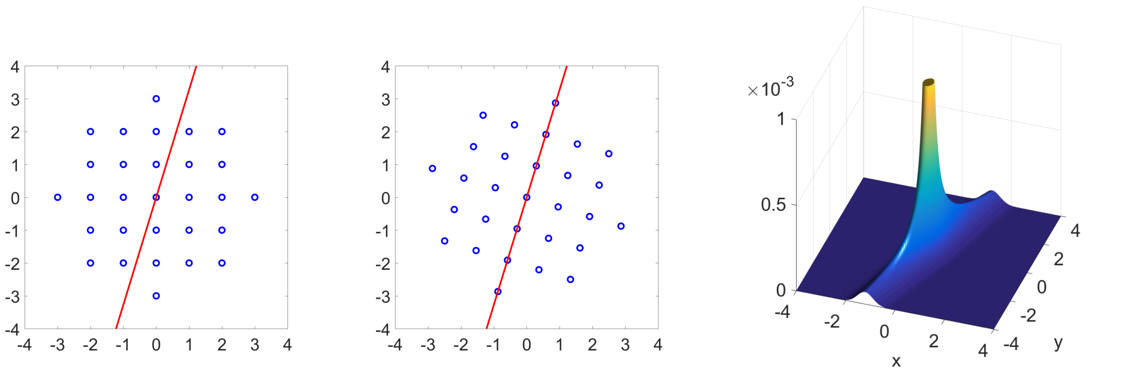
$$\Gamma_h := \{\partial D_h : \langle \mathbf{g}(\mathbf{x}), \mathbf{N}(\mathbf{x}) \rangle^2 > \gamma\}$$

where $\gamma > 0$ is a small constant. The non-Euclidean distance to boundary T_h^* is then computed as the Euclidean distance to the active boundary. The algorithm is modified so that at any given iteration, only a subset of boundary pixels are filled - namely those minimizing T_h^*. That is

$$\text{ready}(\mathbf{x}) = \text{true} \Leftrightarrow \mathbf{x} \in \text{argmin}_{\mathbf{y} \in \partial D_h} T_h^*(\mathbf{y}).$$

This adaptation leads to improvements in the long range extrapolation of isophotes, as in Figure 2.3.

Guidefill. Guidefill [37] is a recent inpainting algorithm designed to address, among other things, the kinking issues in Figure 2.6(b) and Figure 2.4. While coherence transport is able to extrapolate along guidance direction $\mathbf{g}(\mathbf{x})$ only if $\mathbf{g}(\mathbf{x}) = \lambda(\mathbf{v} - \mathbf{x})$ for some $\mathbf{v} \in B_{\epsilon,h}(\mathbf{x})$ (see Figure 2.6(b)), Guidefill replaces the lattice aligned discrete ball $B_{\epsilon,h}(\mathbf{x})$ with the *rotated discrete ball* $\tilde{B}_{\epsilon,h}(\mathbf{x})$ aligned with the local transport direction $\mathbf{g}(\mathbf{x})$, so that $\mathbf{g}(\mathbf{x}) = \lambda(\mathbf{v} - \mathbf{x})$ for some $\mathbf{v} \in \tilde{B}_{\epsilon,h}(\mathbf{x})$ is *always* true. The rotated ball $\tilde{B}_{\epsilon,h}(\mathbf{x})$ contains "ghost pixels" lying between pixel centers which are defined using bilinear interpolation. See Section 5.1 for a deeper discussion of ghost pixels, as well as Figure 2.9(a)-(b) for an illustration of



(a) $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$. (b) $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$. (c) Illustration of the (normalized) weights (2.5.2) for $\mu = 10$.

Figure 2.9: **Neighborhoods and weights for coherence transport and Guidefill:** Here we illustrate the neighborhoods $A_{\epsilon,h}(\mathbf{x})$ and weights (2.5.2) used by coherence transport and Guidefill. In each case $\epsilon = 3\text{px}$ and $\mathbf{g}(\mathbf{x}) = (\cos 73°, \sin 73°)$. Coherence transport (a) uses the lattice-aligned discrete ball $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$, while Guidefill (b) uses the rotated discrete ball $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$. The ball $\tilde{B}_{\epsilon,h}(\mathbf{x})$ is rotated so that it is aligned with the line L (shown in red) passing through $\mathbf{x}$ parallel to $\mathbf{g}(\mathbf{x})$. In general $\tilde{B}_{\epsilon,h}(\mathbf{x})$ contains "ghost pixels" lying between pixel centers, which are defined using bilinear interpolation of their "real" pixel neighbors. Both use the same weights (2.5.2) illustrated in (c). The parameter μ controls the extent to which the weights are biased in favor of points lying on or close to the line L.

$B_{\epsilon,h}(\mathbf{x})$ and $\tilde{B}_{\epsilon,h}(\mathbf{x})$. Guidefill uses the same weights (2.5.2) as coherence transport (illustrated in Figure 2.9(c)) and similarly to the latter's extension [44], it has a way of automatically determining a good fill order. Unlike coherence transport which computes $\mathbf{g}(\mathbf{x})$ concurrently with inpainting, Guidefill computes a guide field $\mathbf{g}(\mathbf{x}) : D_h \to \mathbb{R}^2$ prior to inpainting. The guide field is computed based on splines which the user may adjust in order to influence the results. It is used to automatically compute a good fill order by computing for each $\mathbf{x} \in \partial D_h$ a confidence $C(\mathbf{x}) \in [0,1]$ inspired by Criminisi et al. [22] and given by

$$C(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in\tilde{B}_{\epsilon,h}(\mathbf{x})\cap(\Omega\backslash D^{(k)})} w_\epsilon(\mathbf{x},\mathbf{y})}{\sum_{\mathbf{y}\in\tilde{B}_{\epsilon,h}(\mathbf{x})} w_\epsilon(\mathbf{x},\mathbf{y})}, \tag{2.5.3}$$

and then only filling those pixels for which $C(\mathbf{x}) > c$, where $c \in (0,1)$ is a small constant. That is

$$\text{ready}(\mathbf{x}) = 1(C(\mathbf{x}) > c) \tag{2.5.4}$$

Guidefill was designed for use as part of a 3D conversion pipeline, and as such makes use of a set B_h of "bystander pixels" which are neither inpainted nor may be used for inpainting. However, this is not relevant to our current investigation and we will assume $B_h = \emptyset$ throughout. As shown in Figure 2.6(c) - Guidefill is able to largely, but not completely, eliminate kinking artifacts. It was in the hope of overcoming this that we designed the semi-implicit version of Algorithm 1 discussed in Section 5.2.

Remark 2.5.1. *Note that we have deliberately excluded the point* $\mathbf{x}$ *from the update formula* (2.0.3) *in Algorithm 1, even if the set* $A_{\epsilon,h}(\mathbf{x})$ *contains* $\mathbf{x}$*. This is* not *done in any of the methods [64, 12, 44, 37] we have just discussed, but it makes no difference to them or any other variant of the direct form of Algorithm 1, because the subroutine FillRow only involves sums taken over* $A_{\epsilon,h}(\mathbf{x}) \cap (\Omega\backslash D^{(k)})$*, which never contains* $\mathbf{x}$*. However, the semi-implicit extension of Algorithm 1 expresses* $u_h(\mathbf{x})$ *as a sum of* $u_h(\mathbf{y})$ *over a set of points that might include* $\mathbf{x}$*. This creates problems with weights such as* (2.5.2) *for which* $w_\epsilon(\mathbf{x},\mathbf{x}) = \infty$*. See Appendix A.1 for further details.*

Chapter 3

3D Conversion

The increase in demand over the past decade for 3D content has resulted in the emergence of a multi-million dollar industry devoted to the conversion of 2D films into stereo 3D. This is partly driven by the demand for 3D versions of old films, but additionally many current filmmakers are choosing to shoot in mono and convert in post production [60]. Examples of recent films converted in whole or in part include Maleficent, Thor, and Guardians of the Galaxy [1].

Mathematically, 3D conversion amounts to constructing the image or video shot by a camera at the perturbed position $p + \delta p$ and orientation $O + \delta O$, given the footage at (p, O).

3.1 Two primary conversion pipelines.

There are essentially two pipelines for achieving this. The first pipeline assumes that each frame of video is accompanied by a depth map (and hence is more applicable to footage from RGB-D cameras). The new viewpoint is generated by "warping" the original footage based on the given depth map and known or estimated camera parameters - see [15] for an excellent recent overview. This pipeline has applications including 3D TV and free-viewpoint rendering [74, 28]. However, it is not typically used in the movie industry - this is for a number of reasons (including, for example, the fact that much of what is being converted are older movies created before RGB-D cameras were invented) - see [73, 60] for more details and discussion.

In this thesis we focus on a second pipeline, which is of greater interest in film. This pipeline does not assume that a depth map is given. Instead, it is based on teams of artists generating a plausible 3D model of the scene, reprojecting the original footage onto that model from a known or estimated camera position, and then rerendering the scene from a novel viewpoint. Unlike the previous pipeline this one involves a step whereby teams of artists create masks for every relevant object in the original scene. Crucially, these masks include occluded parts of objects - see Figure 3.1(c). We go over this pipeline in detail in Section 3.3.

One thing both pipelines have in common is a hole-filling or disocclusion step whereby missing information in the form of RGB values visible from $(p + \delta p, O + \delta O)$ but not from (p, O) is "inpainted". This step is considered one of the most technical and time-consuming pieces of the pipeline [60]. However, while the disocclusion step arising in the first pipeline has received a lot of attention in the literature, see for example [15, 72, 71, 28, 74, 39, 53, 23, 42, 48] to name a few, the disocclusion step arising in the second pipeline relevant to film has received far less attention. To the best of our knowledge our paper [37] (created as part of this thesis) is the first paper to address it directly. While related, these two disocclusion problems have important differences. Most significantly, the fact that our pipeline comes with an explicit mask for every scene object - even occluded parts - and the fact that we have a full 3D model instead of just a single depth map from a single viewpoint, has two major consequences. Firstly, while the methods above need to inpaint both the color information at the new view and the corresponding new depth map, we get the depth map at the new viewpoint for free. This is important because most of the methods in the literature either devote quite a bit of effort to inpainting the depth map [15], or else do so based on rough heuristics [72, 71, 28, 74, 39, 53, 23, 42, 48], which, as noted in [15, Sec. II.C.], tend to fail. Secondly, these masks give an explicit segmentation of the scene into relevant objects both in the old viewpoint and the new one. The methods in the other pipeline, by contrast, have access to neither. This means that we, unlike the above approaches, always know which pixels to use for inpainting and do not have to worry about (for example) inpainting a piece of the foreground into the background. By contrast, all of the above methods have to rely on imperfect heuristics to guess based on the depth map which pixels belong to which object - see [15, Sec. II.B.].

Additionally, in our pipeline, the inpainting is done by teams of artists armed with a “toolbox” of inpainting algorithms. These algorithms provide a starting point which artists may then touch up by hand. Hence interactive speeds and the ability for the user to influence the results of inpainting, which may not be a priority in the other pipeline, are important in ours.

3.2 Related Work on Disocclusion Inpainting for 3D Conversion

Over the past decade considerable attention has been given in the literature to the design of algorithms for automatic or semi-automatic 3D conversion - at least for the first pipeline based on depth maps. As we have already stated, the pipeline used in film, on which we focus in this thesis, has received little to no attention. Nevertheless, we review here briefly the work on 3D conversion using the first pipeline. In regards to the hole filling step, there is great variability in how it is handled. At one extreme are cheap methods that inpaint each frame independently using very basic rules such as clamping to the color of the nearest useable pixel [39], or taking a weighted average of the closest useable pixels along a small number $(8-12)$ of fixed directions [74, 28]. Slightly more sophisticated is the approach in [53] which applies a depth-adapted variant of Telea’s algorithm [64]. These methods are so basic that they do not appear to inpaint the depth map. In the midrange are a variety of methods based on first inpainting the depth map, and then applying a depth aided variant of Criminisi’s method - examples include [71, 72, 23, 42, 48, 15], see also [15] for an overview of the state of the art. Unfortunately, until recently most of these approaches have been limited in the sense that too little attention has been given to the depth inpainting step, which is done based on crude heuristics, while most of the attention is given to the subsequent color inpainting step. To our knowledge, [15] is the first paper to acknowledge this gap in the literature, and addresses it with a sophisticated approach to depth inpainting.

Finally, at the most expensive extreme are methods taking temporal information explicitly into account, such as [20] which copies spacetime patches into the inpainting domain via a process similar to Criminisi et al.

3.3 A 3D Conversion Pipeline for Film

Here we briefly review a 3D conversion pipeline commonly used in film - see for example [73] for a more detailed description. The pipeline relevant to us involves three main steps (typically done by separate teams of specialized artists) which must be completed before inpainting can proceed:

1. If camera data (including position, orientation and field of view) is not known, it must be estimated.

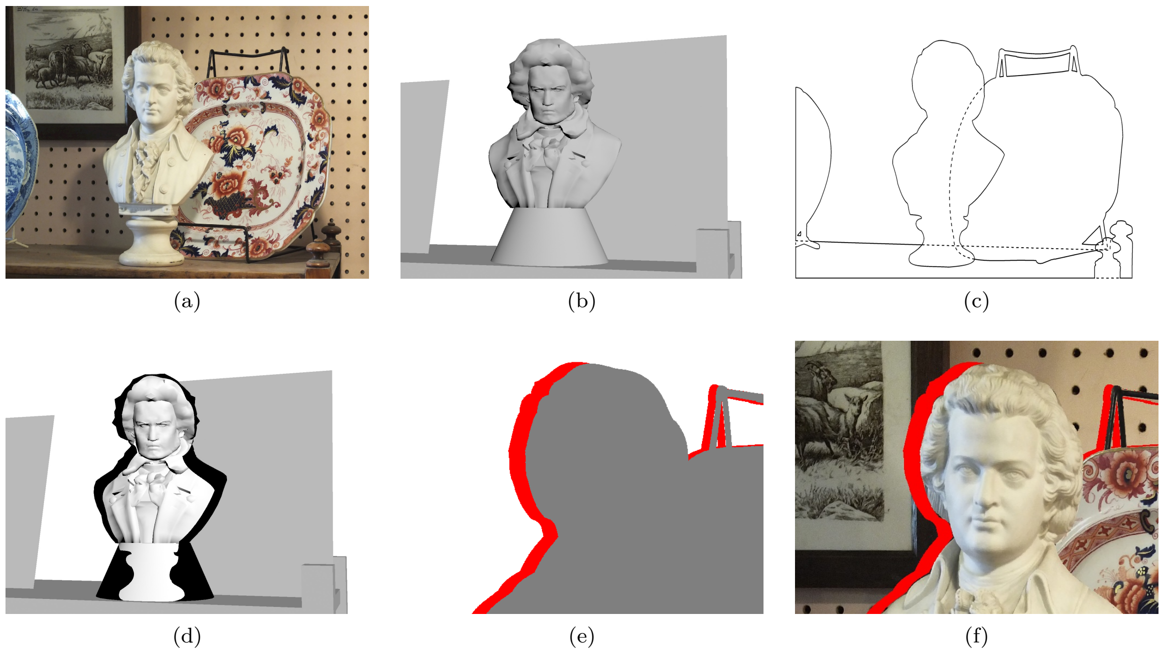


Figure 3.1: **Intermediate data generated in a 3D conversion pipeline prior to inpainting:** (a) original image, (b) rough 3D geometry, (c) object masks including occluded areas, (d) projection of an object mask onto the corresponding object geometry, (e) example labeling of pixels in the new view according to object and visibility (in this case the object in question is the wall, white pixels are visible from both viewpoints, red are visible from the new viewpoint but occluded in the original view, grey are occluded in both views), and (f) the generated new view with red "cracks" requiring inpainting.

This process is often called "match-move" and is typically done with the aid of semi-automatic algorithms based on point tracking [58, 25].

2. Accurate masks must be generated for all objects and for every frame, including occluded areas (See Figure 3.1(c)) . This is typically done to a subpixel accuracy using editable Bézier splines called "roto". These masks play three important roles:

   (a) generating the depth discontinuities visible from the new viewpoint(s).

   (b) generating the scene segmentation in the old viewpoint.

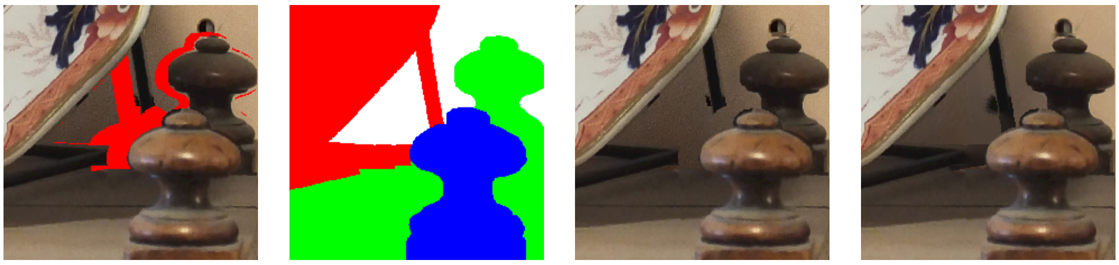
   (c) generating the scene segmentation in the new viewpoint(s).

   These masks need to be as accurate as possible [60].

3. A plausible 3D model of the scene must be generated (see Figure 3.1(b) for an example). This will effectively be used to generate the "smooth" component of the depth map as viewed from the new viewpoint(s) and does not have to be perfect. It is however very important that each object's mask generated in the previous step fits entirely onto its geometry when projected from the assumed camera position, as in Figure 3.1(d). For this reason 3D geometry is typically designed to be slightly larger than it would be in real life [73].

4. For each object, a multi-label mask must be generated assigning a label to each pixel in the new view as either:

   - belonging to the object and visible from the original viewpoint, or



(a) Detail from "Bust": A complex hole involving several objects at multiple depths.

(b) Segmentation of the new view available to our pipeline.

(c) Midground structure cut off by "bleeding" of the background into the midground, when (b) is not taken into account.

(d) Our result.

Figure 3.2: **Importance of the pixel labeling step:** Unlike our pipeline, which has an explicit scene segmentation (b) available to it from the new viewpoint, the depth map based pipeline does not have this information and must rely on heuristics. As noted in [15], these heuristics tend to fail for complex holes involving multiple objects at different depths, such as (a). Most methods in the literature (especially those based on scanlines such as [72, 71, 53]) with the exception of [15] itself (a very recent paper designed to cope with these situations) would struggle to correctly inpaint this hole and would likely produce artifacts similar to (c), where the midground structure is cut off by "bleeding" of background into the midground. Our pipeline does not have this problem as it is able to take advantage of the segmentation in (b).

- belonging to the object and occluded in the original viewpoint, but visible in the new viewpoint, or
- belonging to the object and occluded in both the original and new viewpoints, or
- belonging to another object.

See Figure 3.1(e) for an example where the four labels are colored white, red, grey, and black respectively, and the object in question is the background.

Once these components are in place, the original footage, clipped using the provided masks, is projected onto the geometry from the assumed camera position and orientation. The new view is then generated by rendering the 3D scene from the perspective of a new virtual camera. This new view, however, contains disoccluded regions - formerly hidden by geometry in the old view - which must be inpainted (see Figure 3.1(f)). Inpainting then proceeds on an object by object basis, with each object inpainted separately.

3.3.1 Bystander Pixels

In most image inpainting algorithms it is assumed that all pixels in $\Omega_h \backslash D_h$ may be used for inpainting. However, for this application, each object is inpainted separately, so some of the pixels in $\Omega_h \backslash D_h$ belong to other objects (according to the labelling in step 4) and should be excluded. Failure to do so will result in "bleeding" artifacts, where, for example, a part of the background is extended into what is supposed to be a revealed midground object - see Figure 3.2(c).

Pixels which are neither inpainted nor used as inpainting data are called "bystander pixels", and the set of all such pixels is denoted by B_h. Pixels in $\Omega_h \backslash (D_h \cup B_h)$ are called "readable".

3.3.2 An Alternative Pipeline

Here we briefly review the depth-map based pipeline that has so far received the most attention in the literature. We will go over some of the heuristics employed and give a simple example to show how these heuristics can fail. Please also see [15], which covers the same issues we raise but in more detail, and aims at overcoming them.

The general setup is that we have an initial image/video frame u_0 with an accompanying depth map d_0 taken from a known camera position, and we wish to know the image/video frame u'_0 from a new virtual camera position. The key idea is that of a warping function $\mathcal{W}$, constructed from the known camera positions and parameters, that determines where a pixel $\mathbf{x}$ in u_0 at depth $d_0(\mathbf{x})$ "lands" in u'_0. u'_0 and d'_0 are then constructed by applying $\mathcal{W}$ to all pixels in u_0, d_0 (note that some care may be required as in general $\mathcal{W}(\mathbf{x}, d_0(\mathbf{x}))$ may lie between pixel centers). It is typically assumed that the camera positions are related by a translation orthogonal to the optical axis and parallel to the horizon so that $\mathcal{W}$ is a simple horizontal translation. The result is a new image u'_0 and depth map d'_0 with "gaps" due to disocclusion.

The main disadvantage of this approach is that it has access to neither a depth map of the new view nor a segmentation thereof, whereas we have both. When confronted with a complex hole as in Figure 3.2(a), our pipeline also has access to the segmentation in Figure 3.2(b), and hence while it does not know what RGB values a given pixel in the hole is meant to have, it at least knows which object it belongs to. Without this information, algorithms in this pipeline instead have to make guesses based on heuristics. One common approach is to first inpaint the depth map based on heuristics, then use the inpainted depth map to guess which pixels belong to which objects. For depth map inpainting, a very common heuristic, used in, for example, [72, 71], is to divide the inpainting domain in horizontal scanlines. Each scanline is then filled with a constant depth value that may be that of the endpoint with the greater depth [71], or the minimal extrema of depth patch statistics centered at the endpoints of the scanline as well as their inverse images under the warping function $\mathcal{W}$ [72]. In [53], the authors do not inpaint the depth map, but divide the inpainting domain into horizontal scanlines as usual, declaring the endpoint with greater depth "background" and hence useable for inpainting, while discarding the other endpoint. These approaches will work for most of the hole in Figure 3.2(a), but all of them will incorrectly cut off the vertical plate leg as in Figure 3.2(c). Another approach, used in, for example, [42], is to inpaint using a modified variant of Criminisi that assigns higher priority to pixels with greater depth. This approach is also likely to fail to extend either leg of the plate, since as an object lying in the midground, it will be given a lower priority than background pixels.

In fact, of the approaches currently in the literature, the only one likely to give the correct result in this case is [15], which was designed to address this gap in the literature by incorporating an explicit structure propagation step. By contrast, our algorithm, taking advantage of the segmentation in Figure 3.2(b), produces the result in Figure 3.2(d).

Chapter 4

Guidefill

In this chapter we cover in detail the Guidefill algorithm designed as part of this thesis. As Guidefill was already discussed from a high level in Chapter 2, there will be a slight redundancy here - in particular a few figures and one equation from Chapter 2 will appear again. We felt this was better than forcing the reader to flip back. We will also restate a few things, in order to make the chapter more self contained. We begin by summarizing the unique contributions and aims of Guidefill.

4.1 Summary of contributions and objectives

First, Guidefill is the next logical step in the progression of algorithms sketched in Section 2.3, aimed at incrementally reducing the artifacts listed in Section 2.2. This aspect of the algorithm has already been discussed.

Second, Guidefill is also an adaptation of the shell-based framework presented in Chapter 2 to the 3D conversion pipeline for film presented in Section 3.3. While any of the disocclusion algorithms reviewed in Section 3.2 for the alternative pipeline discussed in Section 3.3.2 could be adapted to this pipeline, they are not designed to take advantage of its particular characteristics. In particular, none of them are designed to take advantage of the scene segmentation available in our pipeline, and with the possible exception of the recent high-quality approach [15], this is likely to lead to needless "bleeding" artifacts when pixels from the wrong object are used for inpainting, as seen in Figure 3.2(c) (see also the discussion in [15, Sec. II.C]). Therefore, Guidefill also repressents an attempt to create an inpainting algorithm designed to take advantage of this extra information explicitly, which it does by making use of the set of "bystander pixels" not to be used for inpainting (Section 3.3.1).

Third, even if the methods from Section 3.2 were adapted to our pipeline, what appears to be missing is an algorithm suitable for the "middle ground" of cases where Telea's algorithm and coherence transport are inadequate, but exemplar-based approaches are needlessly expensive. In particular, because the inpainting domains in 3D conversion tend to be thin "cracks" (see Figure 3.1), there are many situations in which one can safely ignore texture. Guidefill attempts to fill that middle ground.

Fourth, Guidefill is designed to meet a demand in the 3D conversion industry for an interactive inpainting algorithm in which the user has the ability to influence the results of inpainting.

In summary, Guidefill is a fast, geometric, user guided inpainting algorithm intended for use by artists for the hole-filling step of 3D conversion of film. It is designed with two primary objectives in mind:

- The method retains interactive speeds even when applied to the HD footage used in film.
- Although the method is automatic, the artist is kept "in the loop" with a means of possibly adjusting

the result of inpainting that is *intuitive* (that is, they are not simply adjusting parameters).

The first of these goals is accomplished via an efficient GPU implementation based on a novel algorithm for tracking the boundary of the inpainting domain as it evolves. Since our method only operates on the boundary of the inpainting domain in any given step, knowing where the boundary is means that we can assign GPU processors only to boundary pixels, rather than all pixels in the image. For very large images ($\sqrt{N} \gg p$, where N denotes the number of pixels in the inpainting domain, and p denotes the number of available processors), our tracking algorithm leads to a time and processor complexity of $T(N, M) = O(N \log N)$, $P(N, M) = O(\sqrt{N + M})$ respectively (where $N + M$ is the total number of pixels in the image), versus $T(N, M) = O((N + M)\sqrt{N})$, $P(N, M) = O(N + M)$ without tracking - see Theorem 4.8.1 and Theorem 4.8.2. Moreover, for moderately large problems ($\sqrt{N} \lessapprox p$ and $N + M \gg p$) the gains are larger - $T(N, M) = O(\sqrt{N} \log N)$ with tracking in this case.

The second goal is accomplished by providing the user with automatically computed splines showing how key image isophotes are to be extended. These splines may be edited if necessary. In this regard, our algorithm is not unlike Sun et al. [63] and Barnes et al. [7], both of which allow the user to similarly promote the extension of important structures by drawing them onto the image directly. However, both of these approaches are exemplar-based, the former of is relatively expensive and the latter, while less expensive, is limited to linear edges. As far as we know our method is the first *geometric* method to give the user this type of control over the results of inpainting.

Guidefill is intended as a practical tool that is fast and flexible, and applicable to many, but not all, situations. It is not intended as a black box capable of providing the correct result in any situation given enough time. Our method was originally designed for the 3D conversion company Gener8 and a version of it is in use by their stereo artists.

Similarly to many state of the art 3D conversion approaches we treat the problem frame by frame. An extension that uses temporal information is explored in Chapter 7.

4.1.1 Relationship to coherence transport

Guidefill is inspired by the coherence transport algorithm [12, 44], but improves upon it by correcting some of its shortcomings. In particular, both methods proceed by measuring the orientation of image isophotes in the undamaged region near the inpainting domain and then extrapolating them into the inpainting domain. However, in the case of coherence transport both of these steps have problems. Firstly, the procedure for measuring the orientation $\mathbf{g}$ of isophotes in the undamaged region is inaccurate and leads to "kinking" in the extrapolation. See Figure 4.3 as well as Section 4.3.1 for a discussion of this problem and our resolution. Second, as we have seen in Figure 2.6, once fed a desired extrapolation direction $\mathbf{g}$ (which may or may not be accurate based on the last point), coherence transport instead extrapolates along a direction $\mathbf{g}^*$ such that $\mathbf{g}^* \neq \mathbf{g}$ unless $\mathbf{g}$ points in one of a small number of special directions. The result is a secondary "kinking" effect of extrapolated isophotes, which was illustrated in Figure 2.4. This behaviour is explored in Section 4.4 and rigorously analyzed in Chapter 6, Theorem 6.3.1. We also present a "teaser" version of Theorem 6.3.1 in Section 4.7, where a special case is proven (Theorem 4.7.1). As already discussed in Section 2.3 and 2.5, Guidefill resolves this issue by introducing the concept of ghost pixels - virtual pixels lying between pixel centers and defined based on bilinear interpolation. However, our ability to transport along these additional directions comes at a price in the sense that our method introduces some blurring into extrapolated edges. This blurring is most significant for low resolution images and wide inpainting domains, but appears to be minimal for HD images and narrow inpainting domains. A theoretical investigation of these issues is deferred until Section 6.7.

In summary, coherence transport has the following drawbacks:

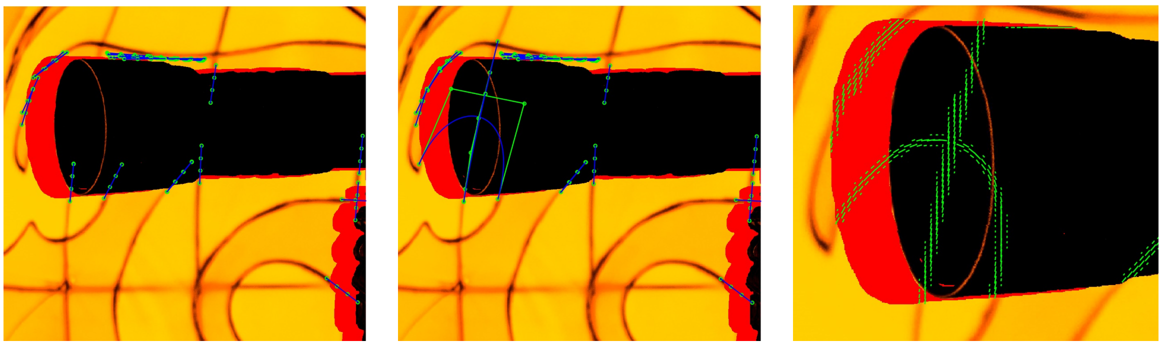

(a) Automatically generated splines. (b) After user adjustment. (c) Closeup of the resulting guide field.

Figure 4.1: Generating the guide field $\mathbf{g}$ (c) based on splines automatically generated by Guidefill (a) and edited by the user (b).

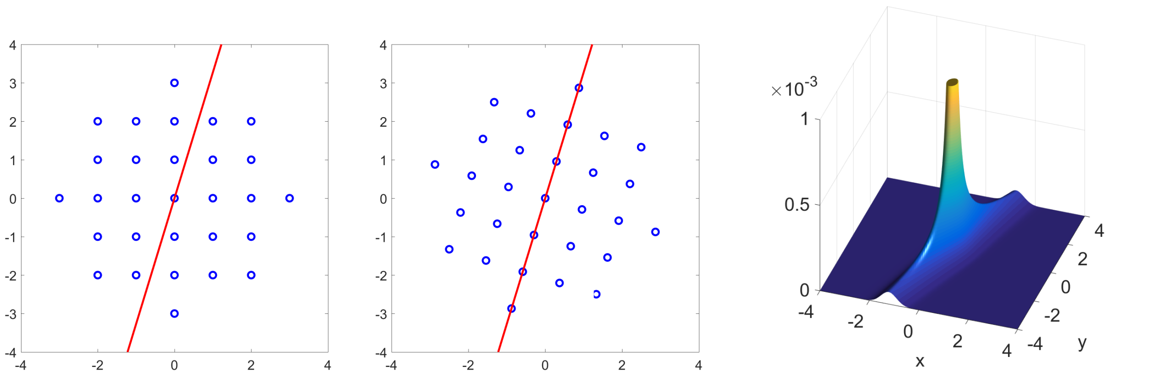
1. Users may need to tune parameters in order to obtain a good result.
2. Extrapolated isophotes may "kink" due to inaccurate computation of the guidance direction $\mathbf{g}$ (see Figure 4.3 and Section 4.3.1).
3. Even if $\mathbf{g}$ is computed correctly, extrapolated isophotes may still "kink" if $\mathbf{g}$ does not belong to a finite set of special directions (see Figures 4.4, 4.5, 4.7 and Sections 4.4, 4.7).
4. The method is a black box with no artist control.
5. The quality of the result can be strongly influenced by the order in which pixels are filled - see Figure 4.6. This is partially addressed in [44], where several methods are proposed for pre-computing improved pixel orderings based on non-Euclidean distance functions. However, these methods all either require manual intervention or else have other disadvantages - see Section 4.5.

Guidefill is aimed at overcoming these difficulties while providing an efficient GPU implementation (the implementation of coherence transport in [12, 44] was sequential, despite the inherent parallelizability of the method), in order to create a tool for 3D conversion providing intuitive artist control and improved results.

4.2 Overview

The main idea behind Guidefill is to generate, possibly based on user input, a suitable vector field $\mathbf{g} : D_h \to \mathbb{R}^2$ to guide the inpainting process, prior to inpainting. The vector field $\mathbf{g}$, which we call the "guide field", is generated based on a small set of curves carrying information about how key image edges in $\Omega_h \backslash (D_h \cup B_h)$ should be continued into D_h. These curves provide an intuitive mechanism by which the user can influence the results of inpainting (see Figure 4.1).

Coherence transport also utilizes a vector field $\mathbf{g}(\mathbf{x})$, but it is calculated concurrently with inpainting. Precomputing the guide field ahead of time is an advantage because the guide field contains information that can be used to automatically compute a good pixel ordering, avoiding artifacts such as Figure 4.6. At step k of our algorithm, given any pixel $\mathbf{x} \in \partial_{\text{active}} D_h^{(k)}$ due to be filled, our algorithm decides based on $\mathbf{g}(\mathbf{x})$ whether to allow $\mathbf{x}$ to be filled, or to wait for a better time. Our test amounts to checking whether or not enough pixels have already been inpainted in the area pointed to by $\mathbf{g}(\mathbf{x})$, and is discussed in greater detail in Section 4.5.



(a) $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$. (b) $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$. (c) Illustration of the (normalized) weights (4.2.1) for $\mu = 10$.

Figure 4.2: Illustration of the neighborhoods $A_{\epsilon,h}(\mathbf{x})$ and weights (4.2.1) used by coherence transport and Guidefill. In each case $\epsilon = 3$px and $\mathbf{g}(\mathbf{x}) = (\cos 73°, \sin 73°)$. Coherence transport (a) uses the lattice-aligned discrete ball $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$, while Guidefill (b) uses the rotated discrete ball $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$. The ball $\tilde{B}_{\epsilon,h}(\mathbf{x})$ is rotated so that it is aligned with the line L (shown in red) passing through $\mathbf{x}$ parallel to $\mathbf{g}(\mathbf{x})$. In general $\tilde{B}_{\epsilon,h}(\mathbf{x})$ contains "ghost pixels" lying between pixel centers, which are defined using bilinear interpolation of their "real" pixel neighbors. Both use the same weights (4.2.1) illustrated in (c). The parameter μ controls the extent to which the weights are biased in favor of points lying on or close to the line L.

The method begins with the user either drawing the desired edges directly onto the image as Bézier splines using a GUI, or else by having a set of splines automatically generated for them based on the output of a suitable edge detection algorithm run on $\Omega_h \backslash (D_h \cup B_h)$. In the latter case, the user may either accept the result or else use it as a starting point which they may improve upon by editing and/or removing existing splines as well as drawing new ones. This is illustrated in Figure 4.1.

Next, the idea is to choose $\mathbf{g}(\mathbf{x})$ to be $\mathbf{0}$ when $\mathbf{x}$ is far away from any splines (e.g. more than a small number of pixels, around ten by default), and "parallel" to the splines when $\mathbf{x}$ is close. Details are provided in Section 4.3.

The purpose of the guide field is to ensure that the inpainting will tend to follow the splines wherever they are present. To accomplish this, at step k of our algorithm a given pixel $\mathbf{x} \in \partial_{\text{active}} D_h^{(k)}$ due to be inpainted is "filled" by assigning it a color equal to a weighted average of its already filled neighbors, with weights biased in favor of neighboring pixels $\mathbf{y}$ such that $\mathbf{y} - \mathbf{x}$ is parallel to $\mathbf{g}(\mathbf{x})$. This is accomplished using the weight function

$$w_\epsilon(\mathbf{x}, \mathbf{y}) = \frac{1}{\|\mathbf{y} - \mathbf{x}\|} \exp\left(-\frac{\mu^2}{2\epsilon^2}(\mathbf{g}^\perp(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}))^2\right), \tag{4.2.1}$$

(introduced in coherence transport [12]) where $\mu > 0$ is a positive parameter and $\epsilon > 0$ is the radius of the neighborhood $A_{\epsilon,h}(\mathbf{x})$. However, whereas the sum in coherence transport is taken over the filled portion of the discrete ball $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$ aligned with the image lattice, we sum over the available "pixels" within a *rotated* ball $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ aligned with the local guide direction $\mathbf{g}(\mathbf{x})$ - see Figure 4.2 for an illustration. The color $u_h(\mathbf{x})$ is then computed based in Algorithm 1 from Chapter 2, taking $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ and using weights (4.2.1), but with the update formula (2.0.3) replaced by (2.0.4) which incorporates the bystander set B_h from Section 3.3.1, as discussed in Remark 2.0.2. Coherence transport "fills" a pixel using exactly the same formula (minus the use of B_h), except that now $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$.

Unlike in coherence transport, however, our neighbourhood $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ is not axis aligned (unless $\mathbf{g}(\mathbf{x})$ is parallel to e_1 or e_2), and this means that in general we have to evaluate u_h *between* pixel centers, which we accomplish by extending the domain of u_h at step k from $\Omega_h \backslash (D_h^{(k)} \cup B_h)$ to $\Omega \backslash (D^{(k)} \cup B)$ using bilinear interpolation. That is, we define

$$u_h(\mathbf{x}) = \sum_{y \in \Omega_h} \Lambda_{\mathbf{y},h}(\mathbf{x}) u_h(\mathbf{y}) \qquad \text{for all } \mathbf{x} \in \Omega \backslash (D^{(k)} \cup B), \tag{4.2.2}$$

where $\{\Lambda_{\mathbf{y},h}\}_{\mathbf{y}\in\Omega_h}$ denotes the basis functions of bilinear interpolation. Note that the continuous sets B and $D^{(k)}$ have been defined so that they include a one pixel wide buffer zone around their discrete counterparts, ensuring that bilinear interpolation is well defined outside $D^{(k)} \cup B$. The reason for the introduction of $\tilde{B}_{\epsilon,h}(\mathbf{x})$ is to avoid a “kinking” phenomena whereby isophotes given a guidance direction $\mathbf{g}(\mathbf{x})$ instead extrapolate along $\mathbf{g}^*(\mathbf{x}) \neq \mathbf{g}(\mathbf{x})$. This is discussed in detail in Section 4.4 and Section 4.7. But first we describe our process of spline detection and the generation of the guide field, and how this is done in such a way as to avoid a second “kinking” phenomena in the computation of $\mathbf{g}(\mathbf{x})$ itself.

Remark 4.2.1. *Note that although the weights* (4.2.1) *have a pole at* $\mathbf{y} = \mathbf{x}$*, because* $u_h(\mathbf{x})$ *is expressed as a weighted average of its* already *filled neighbors, the weights* $w_\epsilon(\mathbf{x}, \mathbf{y})$ *are never evaluated at* $\mathbf{y} = \mathbf{x}$ *and so this has no effect.*

4.3 Automatic Spline Detection and Creation of the Guide Field

The goal of the automatic spline detection is to position splines as straight lines in areas near the active boundary of the inpainting domain where we have detected a strong edge. These splines are lengthened so that they extend into the inpainting domain, and may be edited by the user before being used to construct the guide field.

A one pixel wide ring R is computed a small distance from $\partial_{\text{active}} D_h$ in the undamaged area $\Omega_h \backslash (D_h \cup B_h)$ (as we will see in the next subsection, this dilation of R from $\partial_{\text{active}} D_h$ is crucial for obtaining an accurate orientation of extrapolated isophotes).

We then run a version of Canny edge detection [16] on an annulus of pixels containing the ring, and check to see which pixels on the ring intersect a detected edge. Portions of the annulus not labelled as belonging to the current object are ignored. For those pixels which do intersect a detected edge, we draw a spline in the direction of the edge beginning at that pixel and extending linearly into the inpainting domain.

The direction of the edge is calculated based on the *structure tensor* [68]

$$\mathcal{J}_{\sigma,\rho} := g_\rho * (\nabla u_\sigma \otimes \nabla u_\sigma) \qquad \text{where } u_\sigma := g_\sigma * u_h, \tag{4.3.1}$$

(and where g_σ is a Gaussian centered at $\mathbf{0}$ with variance σ^2, $\otimes$ denotes the tensor product, ∇ denotes the centered difference approximation to the gradient, and $*$ denotes convolution) evaluated at the point $\mathbf{x}_{\text{base}} \in R$. By $\mathbf{x}_{\text{base}}$, we mean a pixel on the annulus R intersecting one of the edges detected by Canny edge detection. It is called $\mathbf{x}_{\text{base}}$ because it is the base point from which we will draw a spline extending into D_h. In practice the above convolutions are truncated to windows of size $(4\sigma+1)^2$, $(4\rho+1)^2$ respectively, so in order to ensure that $\mathcal{J}_{\sigma,\rho}(\mathbf{x}_{\text{base}})$ is computed accurately we have to ensure R is far enough away from $D_h \cup B_h$ that neither patch overlaps it. Note that our approach is different from that of coherence transport [12, 44] (and later adopted by Cao et al. [17]) which proposes calculating a modified structure tensor directly on $\partial_{\text{active}} D_h$. As we will show shortly, the modified structure tensor introduces a kinking effect and so we do not use it. Once $\mathcal{J}_{\sigma,\rho}(\mathbf{x}_{\text{base}})$ has been calculated for a given spline Γ, we assign Γ a direction based on the vector $\mathbf{g}_\Gamma$

$$\mathbf{g}_\Gamma := \pm \tanh\left(\frac{\lambda^+ - \lambda^-}{\Lambda}\right)\mathbf{v}^-,$$

where $(\lambda^\pm, \mathbf{v}^\pm)$ are the maximal and minimal eigenpairs of $\mathcal{J}_{\sigma,\rho}(\mathbf{x}_{\text{base}})$ respectively, Λ is a constant that we fix at $\Lambda = 10^{-5}$ by default, and the sign of $\mathbf{g}_\Gamma$ is chosen in order to point into D_h. Then, the guide field $\mathbf{g}$ at a point $\mathbf{x} \in D_h$ is computed by first finding the spline $\Gamma_\mathbf{x}$ closest to $\mathbf{x}$, and then applying the formula

$$g(\mathbf{x}) = \mathbf{g}_{\Gamma_\mathbf{x}} e^{-\frac{d(\mathbf{x},\Gamma_\mathbf{x})^2}{2\eta^2}}$$

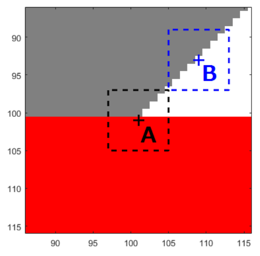
where $d(\mathbf{x}, \Gamma_\mathbf{x})$ is the distance from $\mathbf{x}$ to $\Gamma_\mathbf{x}$ and $\eta > 0$ is a constant that we set at $\eta = 3$px by default. In practice, if $d(\mathbf{x}, \Gamma_\mathbf{x}) > 3\eta$ we set $\mathbf{g}(\mathbf{x}) = \mathbf{0}$.

4.3.1 Kinking Artifacts Created by the Modified Structure Tensor and their Resolution

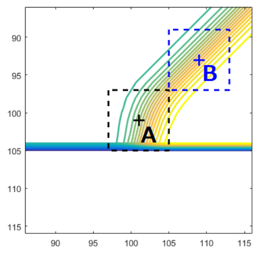
Coherence transport operates by computing for each $\mathbf{x} \in \partial D_h$ a local "coherence direction" $\mathbf{g}(\mathbf{x})$ representing the orientation of isophotes in $\Omega_h \backslash (D_h \cup B_h)$ near $\mathbf{x}$. Inspired by the success of the *structure tensor* (4.3.1) as a robust descriptor of the local orientation of *complete* images, but also noting that $\mathcal{J}_{\sigma,\rho}(\mathbf{x})$ is undefined when $\mathbf{x} \in \partial D_h$, the authors proposed the following *modified structure tensor*

$$\hat{\mathcal{J}}_{\sigma,\rho}(\mathbf{x}) := \frac{\left(g_\rho * \left(1_{\Omega_h \backslash (D_h^{(k)} \cup B_h)} \nabla v_\sigma \otimes \nabla v_\sigma\right)\right)(\mathbf{x})}{\left(g_\rho * 1_{\Omega_h \backslash (D_h^{(k)} \cup B_h)}\right)(\mathbf{x})} \quad \text{where } v_\sigma := \frac{g_\sigma * \left(1_{\Omega_h \backslash (D_h^{(k)} \cup B_h)} u_h\right)}{g_\sigma * 1_{\Omega_h \backslash (D_h^{(k)} \cup B_h)}}, \tag{4.3.2}$$

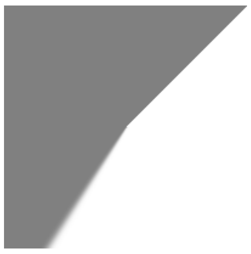
which has the advantage that it is defined even for $\mathbf{x} \in \partial D_h$ (note the use of v_σ as opposed to u_σ in (4.3.2). This notation was introduced in [12] because u_σ is already defined in (4.3.1)). The authors provide no theoretical justification for $\hat{\mathcal{J}}_{\sigma,\rho}(\mathbf{x})$ but instead argue that it solves the problem "experimentally". However, closer inspection shows that the modified structure tensor is an inaccurate description of the orientation of undamaged isophotes near $\mathbf{x}$ when the latter is on or near ∂D_h. We illustrate this using the simple example of inpainting the lower half plane given data in the upper half plane consisting of white below the line $y = x$ and grey above it ($B_h = \emptyset$ in this case). We take $\sigma = 2$, $\rho = 4$. This is presented in Figure 4.3(a), where the inpainting domain is shown in red and where we also show two square neighborhoods of size $(4\sigma + 1)^2$, both centered at points on the line $y = x$, but with one center at point A on ∂D_h, and the other at point $B \in \Omega_h \backslash D_h$, which is far away enough from D_h that neither it nor the larger neighborhood of size $(2\rho + 1)^2$ (not shown) overlap with D_h. The core problem lies in the "smoothed" version v_σ of u, which for pixel A is computed based on a weighted average of pixel values *only in the top half of the box above* $y = 0$. Ideally, v_σ sitting on the line $y = x$ should be half way between grey and white. However, as the weights are radially symmetric and the "angular wedge" of the partial box centered at A contains far more grey pixels than it does white, at pt A we end up with a color much closer to grey. This results in a curvature of the level curves of v_σ that can be seen in Figure 4.3(b). The result is that the modified structure tensor at point A has an orientation of 57° (off by 12°), whereas the regular structure tensor, which is defined at point B since point B is far enough away from D_h to be computed directly, predicts the correct orientation of 45°. Figure 4.3(c)-(d) show the results of inpainting using respectively the minimal eigenvalue of modified structure tensor at point A and the structure tensor at point B as the guidance direction. This is why in Section 4.3 we backed our splines up from the inpainting domain and computed their orientation using the structure tensor rather than the modified structure tensor.



(a) Points A, B and their respective neighborhoods of size $(4\sigma+1)^2$.



(b) The isocontours of v_σ used to compute the modified structure tensor (4.3.2) bend near point A.


(c) Inpainting using Guidefill with $\mathbf{g}$ calculated at pt A using the modified structure tensor (4.3.2).

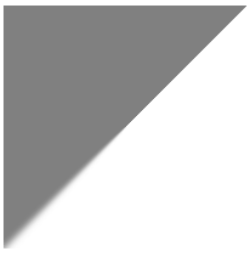

(d) Inpainting using Guidefill with $\mathbf{g}$ calculated at pt B using the ordinary structure tensor (4.3.1).

Figure 4.3: Kinking induced by the modified structure tensor. Consider the simple problem shown in (a) of extending a 45° line into the inpainting domain (red). A first step is to measure the orientation of this line, which coherence transport proposes to do directly on ∂D_h, at point A, using the modified structure tensor $\hat{\mathcal{J}}_{\sigma,\rho}$ (4.3.2) (with $\sigma = 2, \rho = 4$). However, as can be seen in (b), the level lines of v_σ, a smoothed version of u computed as an intermediate in (4.3.2) (as noted in the text, the notation v_σ was introduced in [12] because the ordinary structure tensor (4.3.1) already defines a u_σ), bend in the vicinity of ∂D_h. The resulting guidance direction $\mathbf{g}_A$ (computed at A using the modified structure tensor) makes an angle of 57° with the horizontal, off by 12° from the correct value of 45° obtained by evaluating the ordinary structure tensor (4.3.1) at B. (c)-(d) show the results of inpainting using Guidefill with guidance directions $\mathbf{g}_A$ and $\mathbf{g}_B$ respectively.

Remark 4.3.1. *In some ways our spline-based approach resembles the earlier work by Masnou and Morel [46] and later Cao et al. [17] in which level lines are interpolated across the inpainting domain by joining pairs of "compatible T-junctions" (level lines with the same grey value intersecting the boundary with opposite orientations). This was done first as straight lines [46] , and later as Euler spirals [17]. An $O(N^3)$ algorithm was proposed in [17] for joining compatible T-junctions, where N is the number of such junctions. This could be beneficial in situations such as Figure 4.1(a)-(b), where a similar process was done by hand in the editing step.*

However, our situation is different because we no longer have a simple interpolation problem - in particular, instead of an inpainting domain surrounded on both sides by useable pixels, we now typically have D_h with usable pixels on one side, and bystander pixels on the other (for example, pixels belonging to some foreground object as in Figure 3.1(f)). In some cases we might get around this by searching the perimeter of $D_h \cup B_h$, as opposed to just the perimeter of D_h, for compatible T-junctions. However, this will not always work. For example, consider the problem of inpainting a compact object in the midground partially occluded by something in the foreground. In this case the usable pixels $\Omega_h \backslash (B_h \cup D_h)$ may be a small island entirely surrounded by $B_h \cup D_h$. In such cases our problem is clearly no longer interpolation but extrapolation, and it doesn't make sense to talk about joining compatible T-junctions.

Nevertheless, following the definition of "compatibility" given in [17], one way of incorporating this idea would be to declare two splines S_1 and S_2 based at $\mathbf{x}^{(1)}_{base}$ and $\mathbf{x}^{(2)}_{base}$ "compatible" if

$$(\nabla u_\sigma(\mathbf{x}^{(1)}_{base}) \cdot T_R(\mathbf{x}^{(1)}_{base}))(\nabla u_\sigma(\mathbf{x}^{(2)}_{base}) \cdot T_R(\mathbf{x}^{(2)}_{base})) < 0,$$

where u_σ is given by (4.3.1) and $T_R(\mathbf{x})$ denotes the unit positively oriented tangent vector to the ring R evaluated at $\mathbf{x} \in R$. Compatible splines could then be further tested by comparing patches around the base of each, with the patches rotated according to the orientation of the spline. Those with a high match score could be tentatively joined, with the user given the option to accept or reject this. However, this is beyond

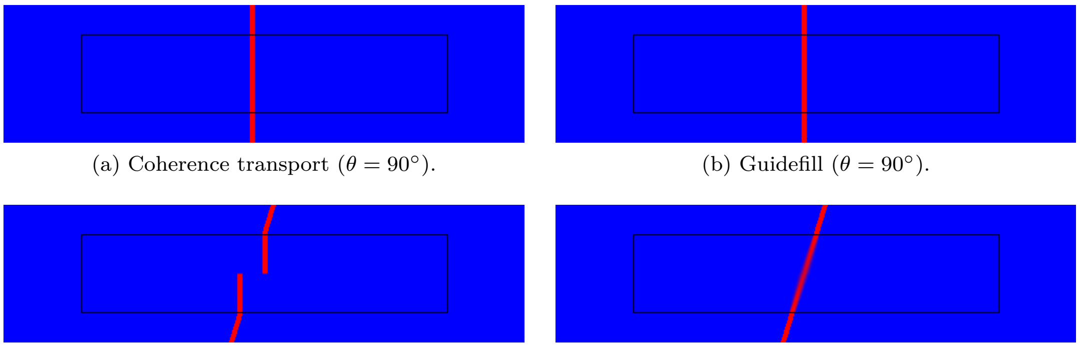

(a) Coherence transport ($\theta = 90°$). (b) Guidefill ($\theta = 90°$). (c) Coherence transport ($\theta = 73°$). (d) Guidefill ($\theta = 73°$).

Figure 4.4: Connecting broken lines using coherence transport (left column) and Guidefill (right column). When the line to be extrapolated is vertical ($\theta = 90°$), both methods are successful. However, when the line is rotated slightly ($\theta = 73°$) coherence transport causes the extrapolated line to "kink", whereas Guidefill continues to produce a successful connection. A theoretical explanation for this phenomena is provided in Theorem 4.7.1 and illustrated in Figure 4.7.

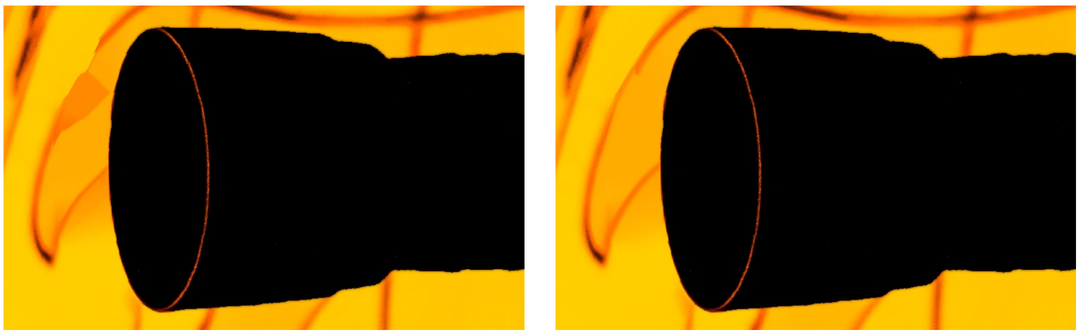

(a) Ghost pixels disabled. (b) Ghost pixels turned on.

Figure 4.5: The effect of ghost pixels on a non-synthetic example ($\epsilon = 3$px, $\mu = 50$). When ghost pixels are disabled, the extrapolated isophotes are unable to smoothly curve as only finitely many transport directions are possible.

the scope of the present work.

4.4 Resolving Additional Kinking Artifacts using Ghost Pixels

The last section showed how coherence transport can cause extrapolated isophotes to "kink" due to an incorrect measurement of the guidance direction $\mathbf{g}$, and how this is overcome in Guidefill. In this section, we briefly go over a second kinking effect that can occur even when $\mathbf{g}$ is known exactly, and how Guidefill overcomes this as well. More details and a theoretical explanation are provided by our continuum analysis in Section 4.7.

Figure 4.4 illustrates the use of coherence transport and Guidefill - each with $\epsilon = 3$px and $\mu = 50$ - for connecting a pair of broken lines. In each case both methods are provided the correct value of $\mathbf{g}$. When the line to be extrapolated is vertical ($\theta = 90°$), both methods are successful. However, when the line is rotated slightly ($\theta = 73°$) coherence transport causes the extrapolated line to "kink", whereas Guidefill makes a successful connection. This happens because coherence transport is trying to bias inpainting in favor of those pixels $\mathbf{y}$ in the partial ball $B_{\epsilon,h}(\mathbf{x}) \cap (\Omega_h \backslash (D^{(k)} \cup B))$ sitting on the line L passing through $\mathbf{x}$ in the direction $\mathbf{g}(\mathbf{x})$, but in this case $B_{\epsilon,h}(\mathbf{x})$ contains no such pixels (other than $\mathbf{x}$ itself, which is

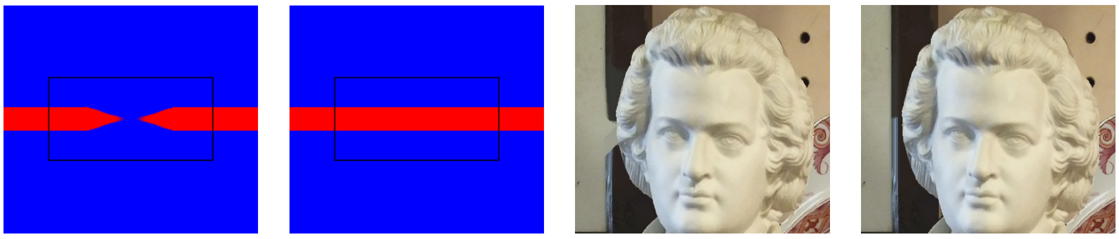

(a) Onion shell order (synthetic). (b) Smart order (synthetic). (c) Onion shell order (non-synthetic). (d) Smart order (non-synthetic).

Figure 4.6: Importance of Pixel Order. When pixels are filled in a simple "onion shell" order (i.e. filled as soon as they appear on the boundary of the inpainting domain), this creates artifacts including "clipping" of isophotes. Our smart order (4.5.3) avoids this by using information from the pre-computed guide field to automatically decide when pixels should be filled.

excluded as it hasn't been inpainted yet) - see Figure 4.2(a). Instead coherence transport favors the pixel(s) *closest* to L, which in this case happens to be $\mathbf{y} = \mathbf{x} + (0, h)$. Since $\mathbf{y} - \mathbf{x}$ is in this case parallel to $(0, 1)$, isophotes are extrapolated along $\mathbf{g}^*(\mathbf{x}) = (0, 1)$ instead of along $\mathbf{g}(\mathbf{x})$ as desired. This implies that inpainting can only be expected to succeed when $\mathbf{g}(\mathbf{x})$ is of the form $\mathbf{g}(\mathbf{x}) = (\lambda n, \lambda m)$ for $\lambda \in \mathbb{R}$, $n, m \in \mathbb{Z}$ and $n^2 + m^2 \leq 9$.

We resolve this problem by replacing $B_{\epsilon,h}(\mathbf{x})$ with the rotated ball of ghost pixels $\tilde{B}_{\epsilon,h}(\mathbf{x})$, which is constructed in order to contain at least one "pixel" on L besides $\mathbf{x}$, as illustrated in Figure 4.2(b).

In Figure 4.5 we also illustrate the importance of ghost pixels on the non-synthetic example with a smoothly varying guide field shown in Figure 4.1. When ghost pixels are not used, the extrapolated isophotes are unable to smoothly curve as only finitely many transport directions are possible. The result is a break in the extrapolated isophote. On the other hand, when ghost pixels are turned on we get a smoothly curving isophote with no break.

4.5 Automatic Determination of a Good Pixel Order (Smart Order)

Figure 4.6(a) and 4.6(c) shows the result of inpainting using an "onion shell" fill order (where pixels are filled as soon as they appear on the boundary of the inpainting domain), for a synthetic and non-synthetic example. In these cases extrapolated lines are cut off due to certain pixels being filled too early. Figure 4.6(b) and 4.6(d) show the same examples using our improved fill order defined by the ready function (4.5.3).

Review of pixel ordering strategies in the literature. There are at least three pixel ordering strategies for shell based inpainting methods currently in the literature. Sun et al. [63] proposed having the user draw critical curves over top of the image, and then filling patches centered on those curves first. März [44] suggested calculating non-standard distance from the boundary functions, and then filling pixels in an order based on those functions. Finally, Criminisi et al. [22] computes for each $\mathbf{p} \in \partial D_h$ a patch priority function $P(\mathbf{p})$ as a product of a confidence term $C(\mathbf{p})$ and a data term $D(\mathbf{p})$, that is $P(\mathbf{p}) = C(\mathbf{p})D(\mathbf{p})$ where

$$C(\mathbf{p}) = \frac{\sum_{\mathbf{q}\in\Psi_{\mathbf{p}}\cap(\mathcal{I}\setminus\Omega)} C(\mathbf{q})}{|\Psi_{\mathbf{p}}|} \quad \text{and } D(\mathbf{p}) = \frac{|\nabla I_{\mathbf{p}}^{\perp} \cdot \mathbf{n}_{\mathbf{p}}|}{\alpha}, \tag{4.5.1}$$

where $\Psi_{\mathbf{p}}$ denotes the patch centered at $\mathbf{p}$, $\nabla^{\perp} I_{\mathbf{p}}$ is the orthogonal gradient to the image I at $\mathbf{p}$, $\alpha = 255$, and $\mathbf{n_p}$ denotes the inward facing unit normal to the current boundary of the inpainting domain.

Patches are then filled sequentially, with the highest priority patch filled first (note that after a patch has been filled, the boundary has now changed, and certain patch priorities must be recomputed).

Our approach. The approach of März [44] based on distance maps might seem the most natural - and indeed there are very simple ways one might imagine constructing a distance map given our already known splines and guide field. For example, distance could grow more "slowly" along or close to splines, while growing at the normal rate far away from splines where the guide field is zero. However, we chose not to go to this route because we wanted to avoid the extra computational effort involved in computing such a map.

Instead, our approach most closely resembles the approach in Criminisi et al. [22]. For each $\mathbf{x} \in \partial_{\text{active}} D_h$, we compute the ratio

$$C(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x}) \cap (\Omega \backslash (D^{(k)} \cup B))} w_\epsilon(\mathbf{x}, \mathbf{y})}{\sum_{\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x})} w_\epsilon(\mathbf{x}, \mathbf{y})}, \tag{4.5.2}$$

where $w_\epsilon(\mathbf{x}, \mathbf{y})$ is the weight function (4.2.1) depending implicitly on $\mathbf{g}$. The ratio C measures what fraction of ghost pixels in $\tilde{B}_{\epsilon,h}(\mathbf{x})$ have been filled, weighted according to their importance, and plays a role similar to the confidence term in (4.5.1). However, because our definition of $C(\mathbf{x})$ also implicitly depends on the guide field $\mathbf{g}(\mathbf{x})$, it will be small when not much information is available in the direction $\mathbf{g}(\mathbf{x})$, even if the majority of the ghost pixels in $\tilde{B}_{\epsilon,h}(\mathbf{x})$ have already been filled. In this sense it also plays a role analogous to the data term in (4.5.1), which tries to ensure that the angle between $\nabla^{\perp} I_{\mathbf{p}}$ and $\mathbf{n_p}$ is not too large. However, unlike Criminisi et al. [22], our algorithm is parallel and not sequential. Therefore, instead of every iteration filling the pixel $\mathbf{x} \in \partial D_h$ with the highest value of $C(\mathbf{x})$, at every iteration we fill *all* pixels $\mathbf{x} \in \partial D_h$ for which $C(\mathbf{x})$ is greater than a threshold. That is, we define

$$\text{ready}(\mathbf{x}) = 1(C(\mathbf{x}) > c) \tag{4.5.3}$$

where $c > 0$ is some small user supplied constant ($c = 0.05$) by default.

Possible extensions. Unlike [22], which assigns high priority to pixels with a large gradient, (4.5.3) does not take into account the size of $\|\mathbf{g}(\mathbf{x})\|$. The result is that areas where $\mathbf{g} = \mathbf{0}$ fill concurrently with areas where $\|\mathbf{g}\| > 0$. However, if one wanted to obtain an algorithm along the lines of Sun et al. [63] where the region with $\|\mathbf{g}\| > 0$ filled first, one would only have to add a data term

$$D(\mathbf{x}) = \|\mathbf{g}(\mathbf{x})\|$$

and then modify (4.5.3) as

$$\text{ready}(\mathbf{x}) = 1(D(\mathbf{x}) > c_2) 1(C(\mathbf{x}) > c_1), \tag{4.5.4}$$

where $c_1 = c = 0.05$ by default. For c_2, one would take $c_2 = 0$ initially, until it was detected that the entire region $\|\mathbf{g}\| > 0$ had been filled, after which point one could revert back to (4.5.3).

4.6 GPU Implementation

Here we sketch two GPU implementations of Guidefill, differing in how they assign GPU threads to pixels in Ω_h. In Section 4.8 we will analyze the time and processor complexity of these algorithms, and show that they belong to different complexity classes. The motivation behind these algorithms is the observation that a typical HD image contains millions of pixels, but the maximum number of concurrent threads in a typical GPU is in the tens of thousands[1]. Hence, it can be advantageous to ensure that GPU threads are only assigned to the subset of pixels being currently worked on.

1. **Guidefill without tracking.** This implementation assigns one GPU thread per pixel in Ω_h, regardless of whether or not that pixel is currently being worked on. This implementation is simplest, but for the reason above does not scale well to very large images.
2. **Guidefill with tracking.** This implementation maintains a list of the coordinates of every pixel in $\partial_{\text{active}} D_h$, which it updates every iteration using a method that requires $O(|\partial_{\text{active}} D_h|)$ threads to do $O(\log|\partial_{\text{active}} D_h|)$ work each. This extra overhead means a longer runtime for very small images, but leads to massive savings for large images as we can assign GPU threads only to pixels in $\partial_{\text{active}} D_h$.

Implementation details of both methods can be found in Appendix A.2.

4.7 Continuum Limit

Here we present a special case of the analysis in Section 6.3. The statement of the main result of that section, Theorem 6.3.1, is quite general and the proof is more than ten pages long, which is unfortunate in that some of the underlying simplicity is obscured. Here we build intuition by presenting a special case that is less abstract - we also prove only a special case within the special case, which may be tackled in roughly half a page. We briefly go over consequences of this result relevant to explaining the kinking artifacts observed in coherence transport and their resolution in Guidefill using ghost pixels, as well as Guidefill's tendancy to introduce blur, as noted in Figure 2.4 and the discussion of Section 2.3. This can be thought of as a preview of Sections 6.6.2 and 6.7 which cover these issues in greater detail.

We consider the fixed ratio continuum limit of u_h introduced in Section 2.4, where $h \to 0$ with $r := \epsilon/h \in \mathbb{N}$, the radius of the neighborhood $A_{\epsilon,h}(\mathbf{x})$ measured in pixels, fixed. Recall that this is different from the limit considered in [12], where first $h \to 0$ and then $\epsilon \to 0$ - see Remark 4.7.4. Our objective is to assume enough complexity to explain the phenomena we have observed, but otherwise to keep our analysis as simple as possible. We aim to prove convergence of u_h, when computed by inpainting using coherence transport or Guidefill with guidance direction $\mathbf{g}$, to u obeying a (weak) transport equation

$$\nabla u \cdot \mathbf{g}_r^* = 0, \tag{4.7.1}$$

where $\mathbf{g}_r^* \neq \mathbf{g}$ in general (indeed this inequality is the source of our observed "kinking"). We will define convergence relative to discrete L^p norms defined shortly by (6.3.5), and we will see that convergence is always guaranteed for $p < \infty$, but not necessarily when $p = \infty$. We then connect this latter point back to the issue of blurring raised in Section 2.3, but a full investiagation of blur has to wait until Section 6.7.

Assumptions. We assume a constant guide direction

$$\mathbf{g}(\mathbf{x}) := \mathbf{g},$$

as this is all that is required to capture the phenomena in question. We assume no bystander pixels ($B = \emptyset$), and that the image domain Ω, inpainting domain D, and undamaged region $\mathcal{U} := \Omega \backslash D$ are all simple rectangles

$$\Omega = (0,1] \times (-\delta, 1] \qquad D = (0,1]^2 \qquad \mathcal{U} = (0,1] \times (-\delta, 0]$$

[1]For example, the GeForce GTX Titan X is a flagship NVIDIA GPU at the time of writing and has a total of 24 multiprocessors [2] each with a maximum of 2048 resident threads [52, Appendix G.1].

equipped with periodic boundary conditions at $x = 0$ and $x = 1$. We discretize $D = (0, 1]^2$ as an $N \times N$ array of pixels $D_h = D \cap \mathbb{Z}_h^2$ with width $h := 1/N$. We assume that $h < \delta/r$ so that $\epsilon < \delta$. Given $f_h : D_h \to \mathbb{R}$, we introduce the following discrete L^p norm for $p \in [0, \infty]$

$$\|f_h\|_p := \Big(\sum_{\mathbf{x} \in D_h} |f_h(\mathbf{x})|^p h^2 \Big)^{\frac{1}{p}}, \qquad \|f_h\|_\infty := \max_{\mathbf{x} \in D_h} |f_h(\mathbf{x})|. \tag{4.7.2}$$

We similarly define $\Omega_h = \Omega \cap \mathbb{Z}_h^2$, $\mathcal{U}_h = \mathcal{U} \cap \mathbb{Z}_h^2$, and assume that the pixels are ordered using the default onion shell ordering, so that at each iteration $D_h^{(k)} = \{(ih, jh) : j > k\}_{i=1}^N$.

Regularity. In Section 6.3 we will consider general boundary data $u_0 : \mathcal{U} \to \mathbb{R}^d$ with low regularity assumptions, including but not limited to nowhere differentiable boundary data with finitely many jump discontinuities. Here, we limit ourselves to piecewise C^2 boundary data because this is the case most relevant to image processing. To be more precise, we assume that u_0 is C^2 everywhere on $\mathcal{U}$ except for on a (possibly empty) finite set of smooth curves $\{\mathcal{C}_i\}_{i=0}^N$ where $N \geq 0$. We assume that the $\mathcal{C}_i$ intersect neither themselves nor each other, and moreover that within $0 < x \leq 1$, $-\delta < y \leq 0$ each $\mathcal{C}_i$ can be parametrized as a smooth monotonically increasing (or monotonically decreasing) function $y = f_i(x)$ each of which makes a non-zero angle with the line $y = 0$ (that is, if $f_i(x^*) = 0$, then $f_i'(x^*) \neq 0$).

Weak Solution. As we have allowed discontinuous boundary data u_0, the solution to (6.3.1) given boundary data u_0 must be defined in a weak sense. Since we have assumed a constant guidance direction $\mathbf{g}(\mathbf{x}) := \mathbf{g}$ and due to the symmetry of the situation, the resulting transport direction $\mathbf{g}_r^*$ will also be constant (we will prove this), so this is simple. So long as $\mathbf{g}_r^* \cdot e_2 \neq 0$, we simply define the solution to the transport problem (6.3.1) with boundary conditions $u(x, 0) = u_0(x, 0)$, u(0,y)=u(1,y) to be

$$u(x, y) = u_0(x - \cot(\theta_r^*)y \bmod 1, 0) \tag{4.7.3}$$

where the mod 1 is due to our assumed periodic boundary conditions and where

$$\theta_r^* = \theta(\mathbf{g}_r^*) \in (0, \pi)$$

is the counterclockwise angle between the x-axis and a line parallel to $\mathbf{g}_r^*$.

Theorem 4.7.1. *Let the image domain Ω, inpainting domain D, undamaged region $\mathcal{U}$, as well as their discrete counterparts, be defined as above. Similarly, assume the boundary data $u_0 : \mathcal{U} \to \mathbb{R}^d$ obeys the regularity conditions above, in particular that it is C^2 except for on a finite, possibly empty set of smooth curves $\{\mathcal{C}_i\}_{i=1}^N$, $N \geq 0$ with the assumed properties.*

Assume D_h is inpainted using Algorithm 1, with neighbourhood

$$A_{\epsilon,h}(\mathbf{x}) \in \{B_{\epsilon,h}(\mathbf{x}), \tilde{B}_{\epsilon,h}(\mathbf{x})\},$$

(that is, either the neighborhood used by coherence transport or the one used by Guidefill). Let $w_\epsilon(\mathbf{x}, \mathbf{y})$ be given by (4.2.1) with guidance direction $\mathbf{g}(\mathbf{x}) := \mathbf{g}$ constant. Suppose we fix $r := \epsilon/h \in \mathbb{N}$, assume $r \geq 2$ (that is, the radius of $A_{\epsilon,h}(\mathbf{x})$ is at least two pixels) and let $h \to 0$. Define the transport direction $\mathbf{g}_r^*$ by

$$\mathbf{g}_r^* = \frac{\sum_{\mathbf{y} \in A_r^-} w_r(\mathbf{0}, \mathbf{y})\mathbf{y}}{\sum_{\mathbf{y} \in A_r^-} w_r(\mathbf{0}, \mathbf{y})} \qquad A_r^- := \{(y_1, y_2) \in \frac{1}{h} A_{\epsilon,h}(0) : y_2 \leq -1\}. \tag{4.7.4}$$

Note that A_r^- depends only on r. Also note that we have written w_r to mean the weights (4.2.1) with ϵ replaced by r. Let $u : (0, 1]^2 \to \mathbb{R}^d$ denote the weak solution (6.3.3) to the transport PDE (6.3.1) with transport direction $\mathbf{g}^*$ and with boundary conditions $u(x, 0) = u_0(x, 0)$, u(0,y)=u(1,y).

Then u exists and for any $p \in [1, \infty]$ and for each channel[2] *of u, u_h we have the bound*

$$\|u - u_h\|_p \le K h^{\frac{1}{2p}} \tag{4.7.5}$$

if $\{\mathcal{C}_i\}$ is non-empty and

$$\|u - u_h\|_p \le K h. \tag{4.7.6}$$

independent of p otherwise (that is, if u_0 is C^2 everywhere). Here K is a constant depending only on u_0 and r.

Remark 4.7.2. *The transport direction $\mathbf{g}_r^*$ predicted by Theorem 4.7.1 has a simple geometric interpretation. It is the average position vector* or center of mass *of the set A_r^- with respect to the normalized weights w_r (4.2.1). This is true regardless of whether or not A_r^- is axis aligned. For coherence transport and Guidefill, we give the set A_r^- the special names b_r^- and $\tilde{b}_r^-$ respectively. For $\mathbf{g} \neq \mathbf{0}$ they are given by*

$$\begin{aligned} b_r^- &:= \{(n,m) \in \mathbb{Z}^2 : n^2 + m^2 \le r^2, m \le -1\} \\ \tilde{b}_r^- &:= \{n\hat{\mathbf{g}} + m\hat{\mathbf{g}}^\perp : (n,m) \in \mathbb{Z}^2, n^2 + m^2 \le r^2, n\hat{\mathbf{g}} \cdot e_2 + m\hat{\mathbf{g}}^\perp \cdot e_2 \le -1\}. \end{aligned}$$

where $\hat{\mathbf{g}} := \mathbf{g}/\|\mathbf{g}\|$ (if $\mathbf{g} = \mathbf{0}$ we set $\tilde{b}_r^- = b_r^-$). These sets can be visualized by looking at the portion of the balls in Figure 4.2(a)-(b) below the line $y = -1$. The limiting transport directions for coherence transport and Guidefill - denoted by $\mathbf{g}_{c.t.}^*$ and $\mathbf{g}_{g.f.}^*$ respectively - are then given by

$$\mathbf{g}_{c.t.}^* = \frac{\sum_{\mathbf{y} \in b_r^-} w_r(\mathbf{0}, \mathbf{y})\mathbf{y}}{\sum_{\mathbf{y} \in b_r^-} w_r(\mathbf{0}, \mathbf{y})} \qquad \textit{and} \qquad \mathbf{g}_{g.f.}^* = \frac{\sum_{\mathbf{y} \in \tilde{b}_r^-} w_r(\mathbf{0}, \mathbf{y})\mathbf{y}}{\sum_{\mathbf{y} \in \tilde{b}_r^-} w_r(\mathbf{0}, \mathbf{y})}. \tag{4.7.7}$$

Although these formulas differ only in the replacement of a sum over b_r^- with a sum of over $\tilde{b}_r^-$, this difference is significant, as is explored in Figure 4.7.

Proof. Here we prove the easy case where u_0 is C^2 everywhere and $A_{\epsilon,h}(\mathbf{x})$ contains no ghost pixels, that is $A_{\epsilon,h}(\mathbf{x}) \subset \mathbb{Z}_h^2$. The case where $A_{\epsilon,h}(\mathbf{x})$ contains ghost pixels lying between pixel centers and for u_0 with lower regularity is covered in Section 6.3. We also only prove the case $p = \infty$, as $p < \infty$ follows trivially since the bound is independent of p in this case. We use the notation $\mathbf{x} := (ih, jh)$ interchangeably throughout.

First note that the symmetry of the situation allows us to rewrite (2.0.3) in Algorithm 1 as

$$u_h(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in A_r^-} w_r(\mathbf{0}, \mathbf{y}) u_h(\mathbf{x} + \mathbf{y}h)}{\sum_{\mathbf{y} \in A_r^-} w_r(\mathbf{0}, \mathbf{y})}.$$

Next we note that A_r^- is nonempty, which follows from our assumption $A_{\epsilon,h}(\mathbf{x}) \in \{B_{\epsilon,h}(\mathbf{x}), \tilde{B}_{\epsilon,h}(\mathbf{x})\}$ and $r \ge 2$ (we leave it as an exercise to the reader that no matter how we rotate $\tilde{B}_{\epsilon,h}(\mathbf{x})$, this is always true). Since $A_r^- \neq \emptyset$, it follows that $\mathbf{g}_r^*$ (4.7.4) is defined, and moreover $\mathbf{g}_r^* \cdot e_2 \neq 0$. This was the condition we needed to ensure that u is defined.

Now that we know u exists, let us define $e_h := u_h - u$. Then it suffices to prove

$$|e_h(\mathbf{x})| \le K h \tag{4.7.8}$$

for all $\mathbf{x} \in D_h$, where $K > 0$ is a constant independent of $\mathbf{x}$. To prove this, we make use of the fact that since u_0 is C^2, u is as well and so there is a $D > 0$ s.t. $\|Hu\|_2 \le D$ uniformly on $(0, 1]^2$, where Hu

[2]Remember, u, u_0, and u_h are all vector valued. We could have made this more explicit by writing $u^{(i)} - u_h^{(i)}$ in (4.7.5), (4.7.6) to emphasize that it holds channel-wise for each $i = 1, \ldots, d$, but felt that this would lead to too much clutter.

denotes the Hessian of u and $\|\cdot\|_2$ is usual operator norm induced by the vector 2-norm (moreover, this D depends only on u_0). We will use this to prove the stronger condition that for any $1 \leq i,j \leq N$ we have

$$|e_h(ih,jh)| \leq jDr^2h^2, \tag{4.7.9}$$

from which (4.7.8) follows with $K = Dr^2$ since $j \leq N = 1/h$.

We proceed by induction, supposing that (4.7.9) holds for all $(i'h, j'h)$ with $1 \leq i' \leq N$ and $j' < j$ (the base case $j = 0$ is obvious). Applying our inductive hypothesis and expanding u to second order we obtain:

$$\begin{aligned}
|e_h(ih,jh)| &\leq \frac{\sum_{\mathbf{y}\in A_r^-} w_r(\mathbf{0},\mathbf{y})|e_h(\mathbf{x}+\mathbf{y}h)|}{\sum_{\mathbf{y}\in A_r^-} w_r(\mathbf{0},\mathbf{y})} + \left|\frac{\sum_{\mathbf{y}\in A_r^-} w_r(\mathbf{0},\mathbf{y})u(\mathbf{x}+\mathbf{y}h) - u(\mathbf{x})}{\sum_{\mathbf{y}\in A_r^-} w_r(\mathbf{0},\mathbf{y})}\right| \\
&\leq (j-1)Dr^2h^2 + \left|\nabla u(\mathbf{x}) \cdot \frac{\sum_{\mathbf{y}\in A_r^-} w_r(\mathbf{0},\mathbf{y})\mathbf{y}h}{\sum_{\mathbf{y}\in A_r^-} w_r(\mathbf{0},\mathbf{y})}\right| + Dr^2h^2 \\
&= jDr^2h^2 + |h\underbrace{\nabla u(\mathbf{x}) \cdot \mathbf{g}_r^*}_{=0}|,
\end{aligned}$$

where we have used the fact that when $\mathbf{x} = (ih, jh)$ and $\mathbf{y} \in A_r^-$, then $(\mathbf{x} + \mathbf{y}h)$ is of necessary form $(i'h, j'h)$ with $1 \leq i' \leq N$ and $j' < j$ needed for our inductive hypothesis to hold. □

4.7.1 Kinking artifacts

Theorem 4.7.1 provides us with a theoretical basis from which to understand the kinking phenomena discussed in Section 4.4 and its resolution using Ghost pixels. This is accomplished by analyzing the relationship between the phase $\theta_r^* = \theta(\mathbf{g}_r^*)$ of the theoretical limiting transport direction and the phase $\theta = \theta(\mathbf{g})$ of the guidance direction. If $\theta_r^* = \theta$, then there is no kinking - otherwise, there is. Figure 4.7 illustrates this by plotting the phase $\theta(\mathbf{g}^*_{\text{c.t.}})$ and $\theta(\mathbf{g}^*_{\text{g.f.}})$ of the theoretical limiting transport directions of coherence transport and Guidefill respectively (4.7.7) as a function of the phase $\theta(\mathbf{g})$ of the guidance direction $\mathbf{g}$. The cases $\epsilon = 3h$ and $\epsilon = 5h$ are considered (coherence transport [12] recommends $\epsilon = 5h$ by default) with $\mu \to \infty$. For coherence transport we have $\theta(\mathbf{g}_r^*) \neq \theta(\mathbf{g})$ except for finitely many angles, explaining the kinking observed in practice. On the other hand, for Guidefill we have $\theta(\mathbf{g}_r^*) = \theta(\mathbf{g})$ for all angles greater than a minimum value. This will be covered in greater detail in Section 6.6.2, where we also consider the semi-implicit form of Guidefill (which obeys $\theta_r^* = \theta$ unless θ is a multiple of π).

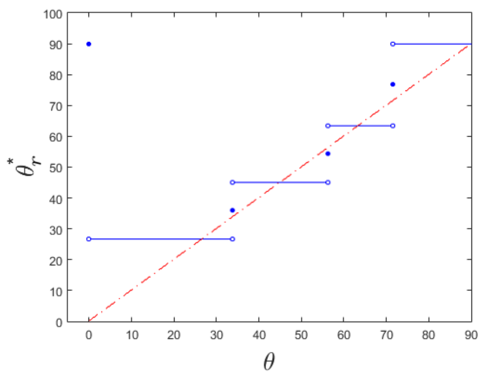
Remark 4.7.3. *In order to understand the kinking of Guidefill shown in Figure 4.7(c)-(d) at small angles for* $\mathbf{g} \neq \mathbf{0}$ *and* $\mu \gg 1$*, it is helpful to consider the decomposition*

$$\tilde{B}_{\epsilon,h}(\mathbf{x}) = \ell_{\epsilon,h}(\mathbf{x}) \cup (\tilde{B}_{\epsilon,h}(\mathbf{x}) \backslash \ell_{\epsilon,h}(\mathbf{x})) \quad \textit{where} \quad \ell_{\epsilon,h}(\mathbf{x}) := \{\mathbf{x} + \epsilon k \hat{\mathbf{g}}\}_{k=-r}^{r},$$

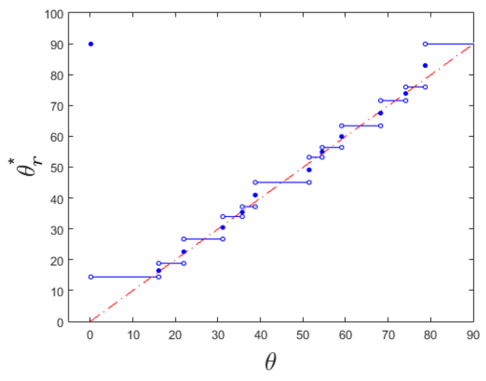
where $\hat{\mathbf{g}} := \mathbf{g}/\|\mathbf{g}\|$*, and where* $r := \epsilon/h \in \mathbb{N}$ *as usual. If* $\ell_{\epsilon,h}(\mathbf{x}) \cap (\Omega_h \backslash (D_h^{(k)} \cup B_h)) \neq \emptyset$*, that is, if* $\ell_{\epsilon,h}(\mathbf{x})$ *contains readable ghost pixels, then under the assumptions of Theorem 4.7.1 one may readily show that* $\mathbf{g}^*_{g.f.}$ *given by* (4.7.7) *obeys* $\mathbf{g}^*_{g.f.} = \mathbf{g}$*. The kinking observed for small angles in Figure 4.7(c)-(d) occurs when* $\ell_{\epsilon,h}(\mathbf{x})$ *contains no readable pixels, that is*

$$\ell_{\epsilon,h}(\mathbf{x}) \cap (\Omega_h \backslash (D_h^{(k)} \cup B_h)) = \emptyset. \tag{4.7.10}$$

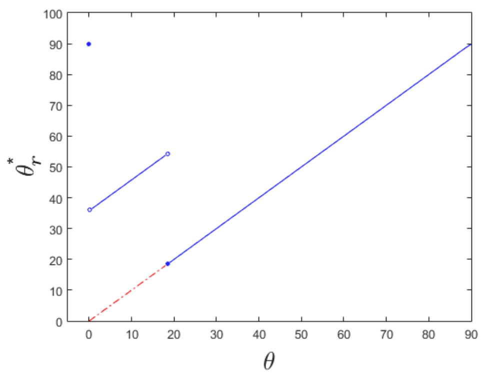
In practice, the smart order proposed in Section 4.5 is likely to detect this situation. Since the weights (4.2.1) *concentrate most of their mass in* $\ell_{\epsilon,h}(\mathbf{x})$ *when* μ *is large, in this case we can expect the confidence*



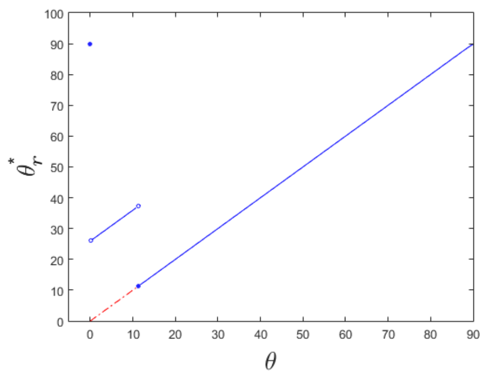
(a) Coherence transport, $r = 3$.



(b) Coherence transport, $r = 5$.



(c) Guidefill, $r = 3$.



(d) Guidefill, $r = 5$.

Figure 4.7: The theoretical limiting curves $\theta_r^* = \theta(\mathbf{g}^*_{\text{c.t.}})$ (coherence transport (a)-(b)) and $\theta_r^* = \theta(\mathbf{g}^*_{\text{g.f.}})$ (Guidefill (c)-(d)) as a function of $\theta = \theta(\mathbf{g})$, with $\mathbf{g}^*_{\text{c.t.}}$ and $\mathbf{g}^*_{\text{g.f.}}$ given by (4.7.7), and where $\mathbf{g}$ is the desired guidance direction fed into the weights (4.2.1). We set $r := \epsilon/h = 3, 5$ and consider $\mu \to \infty$. The ideal curve $\theta_r^* = \theta$ is highlighted in red. The limiting guide directions $\mathbf{g}^*_{\text{c.t.}}$ and $\mathbf{g}^*_{\text{g.f.}}$ are related by (4.7.4) to the weights (4.2.1) as well as the distribution of sample points within $A_{\epsilon,h}(\mathbf{x})$. Coherence transport makes the choice $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$, leading to the "kinking" observed in (a)-(b), where $\theta_r^* \neq \theta$ for all but finitely many angles. The choice $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ made by Guidefill is largely able to avoid this and exhibits no kinking for all angles greater than a critical minimum - see Remark 4.7.3 as well as Section 6.6.2 for more details.

term (4.5.2) *to be small and the smart order test* (4.5.3) *will tell the algorithm to delay the filling of* $\mathbf{x}$. *If at a later iteration* (4.7.10) *no longer holds,* (4.5.3) *should be satisfied and inpainting can resume with no kinking.*

Remark 4.7.4. *The limiting transport direction* $\mathbf{g}_r^*$ *predicted by Theorem 4.7.1 is similar to the transport direction predicted by März in [12] (Theorem 1). However, as already stated, while März considered the double limit where* $h \to 0$ *and then* $\epsilon \to 0$*, we consider the single limit* $(h, \epsilon) \to (0, 0)$ *with* $r = \epsilon/h$ *fixed. The result is that whereas [12] obtains a formula for* $\mathbf{g}^*$ *as an integral over a (continuous) half-ball, our* $\mathbf{g}_r^*$ *is a finite sum over a discrete half-ball. In particular, when* $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$ *as in coherence transport, the following predictions are obtained for the limiting transport direction of coherence transport (note that we write* w_r *and* w_1 *to mean the weights* (4.2.1) *with* ϵ *replaced by* r *and* 1 *respectively):*

$$\mathbf{g}^*_{\text{\textit{märz}}} = \frac{\int_{\mathbf{y}\in B_1^-(\mathbf{0})} w_1(\mathbf{0},\mathbf{y})\mathbf{y}d\mathbf{y}}{\int_{\mathbf{y}\in B_1^-(\mathbf{0})} w_1(\mathbf{0},\mathbf{y})} \qquad \mathbf{g}^*_{ours} = \frac{\sum_{\mathbf{y}\in b_r^-} w_r(\mathbf{0},\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y}\in b_r^-} w_r(\mathbf{0},\mathbf{y})},$$

where

$$\begin{aligned} B_1^-(\mathbf{0}) &:= \{(x,y) \in \mathbb{R}^2 : x^2 + y^2 \leq 1 \text{ and } y < 0\} \\ b_r^- &:= \{(n,m) \in \mathbb{Z}^2 : n^2 + m^2 \leq r^2 \text{ and } m \leq -1\}. \end{aligned}$$

Our discrete sum $\mathbf{g}^*_{ours}$ *predicts the kinking observed by coherence transport in practice, whereas the integral* $\mathbf{g}^*_{\text{\textit{märz}}}$ *does not.*

4.7.2 Blur

Theorem 4.7.1 does not allow us to explain the origins of blur artifacts - for that we will need the asymptotic limit introduced in Section 6.7. However, the L^p convergence result in Theorem 4.7.1 for piecewise C^2 boundary data does imply that blur has to become less significant for higher resolution images. In particular, Figure 2.4 suggests that the most significant blur artifacts consist of the smoothing out of what should be sharp jump discontinuities over a non-zero blur radius. Theorem 4.7.1 implies that this blur radius must go to zero as $h \to 0$, otherwise we not converge in L^p for $p < \infty$. See Section 6.7 and in particular Figure 6.14 for more details.

4.8 Algorithmic Complexity

In this section we analyze the complexity of the two implementations of Guidefill sketched in Section 4.6 as parallel algorithms. Specifically, we analyze how both the *time complexity* $T(N, M)$ and *processor complexity* $P(N, M)$ vary with $N = |D_h|$ and $M = |\Omega_h \backslash D_h|$, where a time complexity of $T(N, M)$ and processor complexity of $P(N, M)$ means that the algorithm can be completed by $O(P(N, M))$ processors in $O(T(N, M))$ time per processor. See for example [55, Ch. 5] for a more detailed discussion of the time and processor complexity formalism for parallel algorithms.

We assume that Guidefill is implemented on a parallel architecture consisting of p processors working at the same time in parallel. We further assume that when Guidefill attempts to run $P > p$ parallel threads such that there are not enough available processors to comply, the P threads are run in $\lceil P/p \rceil$ sequential steps. In reality, GPU architecture is not so simple - see for example [52, Ch. 4] for a discussion of GPU architecture, and for example, [43] for a more realistic theoretical model. We do not consider these additional complexities here.

In Theorem 4.8.1 we derive a relationship between the time and processor complexities $T(N,M)$, $P(N,M)$ and the number of iterations $K(N)$ required for Guidefill to terminate. This relationship is valid in general but does not allow us to say anything about $K(N)$ itself. Next, in Theorem 4.8.2 we establish bounds on $K(N)$ under two simplifying assumptions. Firstly, we assume that the inpainting domain is surrounded entirely by readable pixels - that is $(\partial_{\text{outer}} D_h) \cap B_h = \emptyset$. In particular, this means that we assume the inpainting domain does not include the edge of the image and is not directly adjacent to pixels belonging to another object (such as an object in the foreground). Secondly, we assume that the smart ordering of Section 4.5 is turned off. We also include a discussion in Appendix A.3 of what to expect in the general case. Our analysis considers only the filling step of Guidefill after the guide field has already been constructed.

Theorem 4.8.1. *Let $N = |D_h|$, $M = |\Omega_h \backslash D_h|$ denote the problem size and let $T(N,M)$ and $P(N,M)$ denote the time complexity and processor complexity of the filling step of Guidefill implemented on a parallel architecture as described above with p available processors. Let $K(N)$ denote the number of iterations before Guidefill terminates. Then the processor complexity of Guidefill with and without boundary tracking is given by*

$$P(N,M) = \begin{cases} O(N+M) & \textit{without tracking} \\ O(\sqrt{N+M}) & \textit{with tracking} \end{cases}$$

while the time complexity is given by

$$T(N,M) = \begin{cases} O(K(N)) & \textit{if } P(N,M) \leq p \\ O((N+M)K(N)) & \textit{if } P(N,M) > p \end{cases} \quad \textit{without tracking}$$

$$T(N) = \begin{cases} O((\sqrt{N} + K(N))\log(N)) & \textit{if } P(N,M) \leq p \\ O((N + K(N))\log(N)) & \textit{if } P(N,M) > p \end{cases} \quad \textit{with tracking.}$$

Proof. For the case of no boundary tracking Guidefill allocates one thread per pixel in Ω_h, hence $P(N,M) = O(|\Omega_h|) = O(N+M)$. In this case if $|\Omega_h| := N+M < p$, then each thread fills only one pixel, and hence does $O(1)$ work. On the other hand, if $N+M > p$, each thread must fill $\lceil \frac{N+M}{p} \rceil$ pixels. It follows that

$$T(N,M) \leq \sum_{k=1}^{K(N)} \left\lceil \frac{N+M}{p} \right\rceil \leq \begin{cases} K(N) & \text{if } N+M < p \\ \frac{2}{p}(N+M)K(N) & \text{otherwise.} \end{cases}.$$

Guidefill with tracking allocates $O(|\partial D_h^{(k)}|)$ threads per iteration of Guidefill, each of which do $O(\log|\partial D_h^{(k)}|)$ work. This is because, as stated in Section 4.6, the boundary is updated over a series of $O(\log|\partial D_h^{(k)}|)$ parallel steps. In order to keep the processor complexity at $O(\sqrt{N+M})$, we assume that in the unlikely event that more than $\sqrt{N+M}$ threads are requested, then Guidefill runs them in $O\left(\left\lceil \frac{|\partial D_h^{(k)}|}{\sqrt{N+M}} \right\rceil\right)$ sequential steps each involving $\sqrt{N+M}$ processors. We therefore have, for $\sqrt{N+M} < p$

$$T(N,M) \leq \sum_{k=1}^{K(N)} \left(\frac{|\partial D_h^{(k)}|}{\sqrt{N+M}} + 1 \right) C \log|\partial D_h^{(k)}| \leq C\log(N) \sum_{k=1}^{K(N)} \left(1 + \frac{|\partial D_h^{(k)}|}{\sqrt{N+M}} \right)$$

where the factor $C > 0$ comes from the hidden constants in the Big O notation. But we know that

$\{\partial D_h^{(k)}\}_{k=1}^{K(N)}$ forms a partition of D_h, so that $\sum_{k=1}^{K(N)} |\partial D_h^{(k)}| = N$. Therefore

$$T(N,M) \leq C\log(N)\left(K(N) + \frac{N}{\sqrt{N+M}}\right) \leq C(\sqrt{N} + K(N))\log(N).$$

An analogous argument with $\sqrt{N+M}$ in the denominator replaced by p handles the case $P(N,M) > p$. □

Theorem 4.8.2. *If we make the same assumptions as in Theorem 4.8.1 and if we further suppose $(\partial_{outer} D_h) \cap B_h = \emptyset$ and that the smart order test from Section 4.5 is turned off, then we additionally have*

$$K(N) = O(\sqrt{N}) \tag{4.8.1}$$

so that, in particular, we have $T(N,M) = O(\sqrt{N})$, $T(N,M) = O(\sqrt{N}\log(N))$ for Guidefill without and with tracking given sufficient processors, and
$T(N,M) = O((N+M)\sqrt{N})$, $T(N,M) = O(N\log(N))$ respectively when there is a shortage of processors.

Proof. Now assume $(\partial_{\text{outer}} D_h) \cap B_h = \emptyset$ and (4.5.3) is disabled. Then after k iterations all pixels $\mathbf{x}$ such that $\mathcal{N}^{(k)}(\mathbf{x}) \cap \Omega_h \backslash D_h \neq \emptyset$ will have been filled, where

$$\mathcal{N}^{(k)}(\mathbf{x}) = \bigcup_{\mathbf{y} \in \mathcal{N}^{(k-1)}(\mathbf{x})} \mathcal{N}(\mathbf{y}), \quad \mathcal{N}^{(1)}(\mathbf{x}) = \mathcal{N}(\mathbf{x}).$$

Therefore, if D_h has not be completely filled after k iterations, there must exist a pixel $\mathbf{x}^* \in D_h$ such that $\mathcal{N}^{(k)}(\mathbf{x}^*) \subseteq D_h$. However, it is easy to see that $|\mathcal{N}^{(k)}(\mathbf{x}^*)| = (2k+1)^2$. But since $|D_h| = N$, after $k = \lceil \sqrt{N}/2 \rceil$ iterations $\mathcal{N}^{(k)}(\mathbf{x}^*)$ will contain more pixels than D_h itself, and cannot possibly be a subset of the latter.

This proves that Guidefill terminates in at most $\lceil \sqrt{N}/2 \rceil$ iterations, and hence $K(N) = O(\sqrt{N})$. □

4.9 Numerical Experiments

In this section we aim to validate our method as a practical tool for 3D conversion, and also to validate the complexity analysis of Section 4.8. We have implemented Guidefill in CUDA C and interfaced with MATLAB. Our experiments were run on a laptop with a 3.28GHz Intel i7 − 4710 CPU with 20GB of RAM, and a GeForce GTX 970M GPU[3].

4.9.1 3D Conversion Examples

We tested our method on a number of HD problems, including the four photographs shown in Figure 4.8 and the video illustrated in Figure 4.9. The photographs were converted into 3D by building rough 3D geometry and creating masks for each object, as outlined in Section 3.3. For the movie, we used a computer generated model with existing 3D geometry and masks[4], as generating these ourselves on a frame by frame basis would have been far too expensive (indeed, in industry this is done by teams of artists and is extremely time-consuming). One advantage of this approach is that it gave us a ground truth to compare against, as in Figure 4.9(k). Additional results in anaglyph 3D are provided in Appendix

[3]The experiments involving nl-means and nl-Poisson are an exception. Because the implementation available online does not support Windows, these experiments had to be done on a separate Linux machine with a 3.40GHz Intel i5 − 4670 CPU with 16GB of RAM. As a comparison, we measured the time to solve a 500×500 Poisson problem to a tolerance of 10^{-6} using the conjugate gradient method in MATLAB, which took 8.6s on our Windows laptop, and 5.2s on the Linux box.

[4]Downloaded from `http://www.turbosquid.com/` in accordance with the Royalty Free License agreement.

A.4 (anaglyph glasses required). Timings for Guidefill are given both with and without the boundary tracking as described in Section 4.6.

As has been noted in Section 3.2, the literature abounds with depth-guided variants of Criminisi's method [22] designed for the disocclusion step arising in 3D conversion using the depth-map based pipeline discussed in Section 3.3.2 (see for example [71, 72, 23, 42, 48, 15]), but not for the pipeline relevant to us. In particular, none of these methods are designed to make explicit use of the bystander set B_h available to us and instead rely on heuristics. In Section 3.3.2 Figure 3.2, we have shown a simple example where with the exception of [15] these heuristics are likely fail. Adapting these methods to our pipeline where depth-map inpainting is unnecessary would require considerable effort and fine tuning. Therefore, rather than comparing with these methods, we considered it more natural to compare with our own "bystander-aware" variant of Criminisi, adapted in an extremely simple way to incorporate the set B_h. We simply modify Criminisi's algorithm by setting the priority equal to 0 on $\partial D_h^{(k)} \backslash \partial_{\text{active}} D_h^{(k)}$ and restricting the search space to patches that do not overlap $\Omega_h \backslash (D_h \cup B_h)$. However, we acknowledge that many of these methods also make further optimizations to Criminisi et al. from the point of view of running time - for example [15] incorporates the running time improvements originally published in their earlier work [14]. We also could have based our "bystander-aware" Criminisi on the improvement in [14], however, instead we note that the running time published in [15] is about 1500px/s, which is still much slower than Guidefill, especially for high-resolution problems (see Table 4.1).

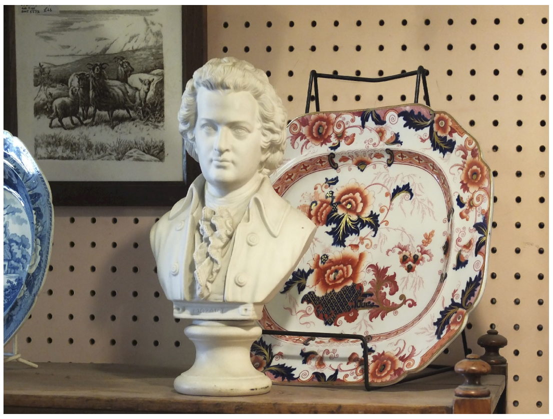
For the photographs, in addition to our "bystander-aware" Criminisi, we also compare the output of Guidefill with four other inpainting methods: coherence transport [12, 44], the variational exemplar-based methods nl-means and nl-Poisson from Arias et al. [5], and Photoshop's Content-Aware fill. For the movie, we compare with the exemplar-based video inpainting method of Newson et al. [50, 49]. However, generating "bystander-aware" versions of all of these methods would have been a significant undertaking, so we arrived at a compromise. To avoid bleeding-artifacts like Figure 3.2(c), we first ran each method using $D_h \cup B_h$ as the inpainting domain, giving the results shown. However, as this led to an unfair running time due to the need to fill B_h, we then ran each method again using only D_h as the inpainting domain, in order to obtain the given timings. All methods are implemented in MATLAB + C (mex) and are available for download online[5].

Figure 4.9 shows a few frames of a 1280px $\times$ 960px $\times$ 101fr video, including the inpainting domain and the results of inpainting with both Guidefill and Newson's method. With the exception of a few artifacts such as those visible in Figure 4.9(j), Newson's method produces excellent results. However, it took 5hr37min to run, and required more than 16GB of RAM. In comparison Guidefill produces a few artifacts, including the incorrectly completed window shown in Figure 4.9(e). In this case the failure is because the one pixel wide ring described in Section 4.3 fails to intersect certain edges we would like to extend. However, Guidefill requires only 19s (if boundary tracking is employed, 31s if it is not) to inpaint the entire video and these artifacts can be corrected as in Figure 4.9(f). However, due to the frame by frame nature of the computation, the results do exhibit some flickering when viewed temporally, an artifact which Newson's method avoids.

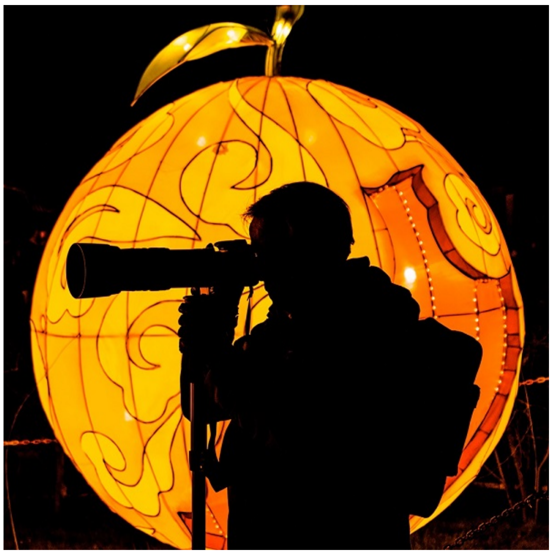
Timings for the images are reported in Table 4.1, with the exception of Content-Aware fill which is difficult to time as we do not have access to the code. We also do not provide timings for Bystander-Aware Criminisi, nl-means, and nl-Poisson for the "Pumpkin" and "Planet" examples as the former ran out of memory while nl-means and nl-Poisson did not finish within two hours. However, for the "Pumpkin" example we do provide the result of nl-Poisson run on a small region of interest. Results are given in Figures 4.10, 4.11, and 4.13. We do not show the output of every method and have included only the

[5]Coherence transport: `http://www-m3.ma.tum.de/bornemann/InpaintingCodeAndData.zip`, Criminisi's method: `https://github.com/ikuwow/inpainting_criminisi2004`, nl-means and nl-Poisson: `http://www.ipol.im/pub/art/2015/136/`, Newson's method: `http://perso.telecom-paristech.fr/~gousseau/video_inpainting/`.

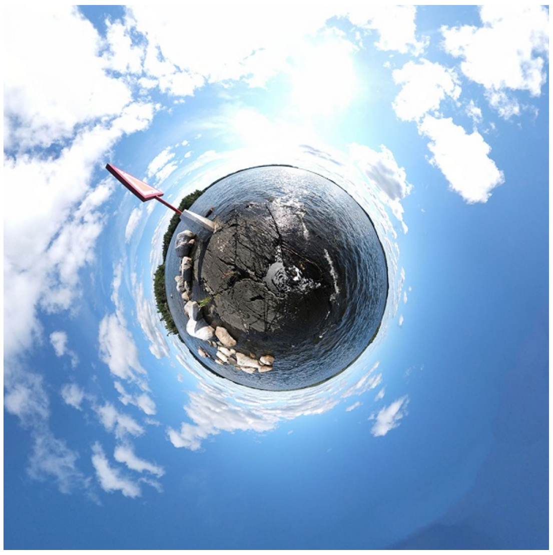
(a) Wine.



(b) Bust.



(c) Pumpkin.



(d) Planet.

Figure 4.8: Example photographs used for 3D conversion, of different sizes (a) 528×960px (b) 1500×1125px (c) 4000×4000px (d) 5000×5000px.

Table 4.1: Timings of different inpainting algorithms used in the conversion of the four examples in Figure 4.8. The inpainting domains of "Wine", "Bust", "Pumpkin", and "Planet" contain 15184px, 111277px, 423549px, and 1160899px respectively. "Guidefill n.t." refers to Guidefill without boundary tracking, "B.A.C." stands for Bystander-Aware Criminisi and "C.T." refers to coherence transport.

	C.T.	B.A.C.	nl-means	nl-Poisson	Guidefill n.t.	Guidefill
Wine	340ms	1 min 40s	41s	2min11s	**233ms**	261ms
Bust	2.13s	37min	23min	1hr 10min	1.34s	**559ms**
Pumpkin	15.7s	–	–	–	6.66s	**1.14s**
Planet	28.5s	–	–	–	4.27s	**923ms**

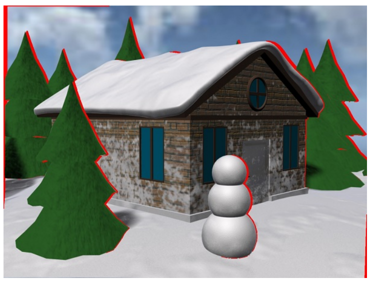
most significant.

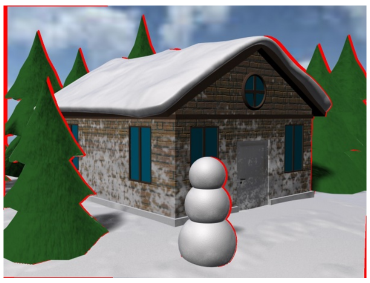
The first example, "Wine", is a 528 × 960px photo. Timings are reported only for the background object, which has an inpainting domain containing 15184px. Figure 4.12 shows the detected splines for the background object and illustrates the editing process. Results are shown in Figure 4.10 in two particularly challenging areas. In this case the highest quality results are provided by nl-means and nl-Poisson, but both are relatively slow. Bystander-Aware Criminisi and Content-Aware fill each produce noticeable artifacts. Guidefill also has problems, most notably in the area behind the wine bottle, where the picture frame is extended incorrectly (this is due to a spline being too short) and where additional artifacts have been created next to the Chinese characters. These problems, however, are mostly eliminated by lengthening the offending spline and editing some of the splines in the vicinity of Chinese characters as illustrated in Figure 4.12. Guidefill is also the fastest method, although in this case the gains aren't as large as for bigger images.

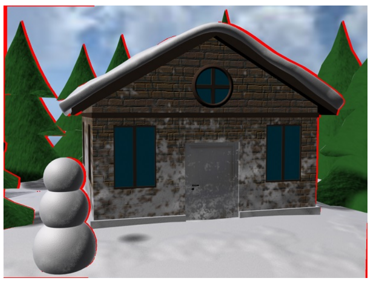
The second example "Bust" is a 1500 × 1125px image. Timings are reported only for inpainting the background object, which has an inpainting domain containing 111277px, and results are shown in Figure 4.11(a)-(f). In this case we chose to edit the automatically detected splines, in particular rotating one that was crooked. Once again, the nicest result is probably nl-Poisson, but an extremely long computation time is required. All other algorithms, including Bystander-Aware Criminisi and nl-means which are not shown, left noticeable artifacts. The fully automatic version of Guidefill also leaves some artifacts, but these are largely eliminated by the adjustment of the splines. The exception is a shock visible in the inpainted picture frame in Figure 4.11(f). As we noted in Section 2.3.2, shock artifacts are an unfortunate feature of the class of methods under consideration.

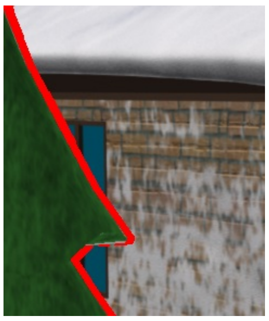
Our third example "Pumpkin" is a very large 4000 × 4000px image. Timings are reported only for the pumpkin object, which has an inpainting domain containing 423549px. Results are shown in Figure 4.11(g)-(l). We ran nl-Poisson on only the detail shown in Figure 4.11(g), because it did not finish within two hours when run on the image as a whole. In this case we edited the automatically detected splines as shown in Figure 4.1(a)-(b). In doing so we are able to recover smooth arcs that most fully automatic methods would struggle to produce. Guidefill in this case is not only the fastest method by far, it also produces the nicest result. In this example we also see the benefits of our boundary tracking algorithm, where it leads to a speed up by a factor of $2-3$. The gains of boundary tracking are expected to be greater for very large images where the pixels greatly outnumber the available processors.

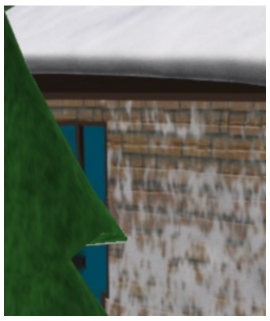
Our final example is a fun example that illustrates how 3D conversion may be used to create "impossible" 3D scenes. In this case the image is a 5000×5000px "tiny planet" panorama generated by stitching together dozens of photographs. The choice of projection creates the illusion of a planet floating in space - however, a true depth map would appear as an elongated finger, as in reality the center of the sphere is only a few feet from the camera, while its perimeter is at a distance of several kilometers. In order to preserve the illusion we created fake spherical 3D geometry. See Appendix A.4 for the full 3D effect -

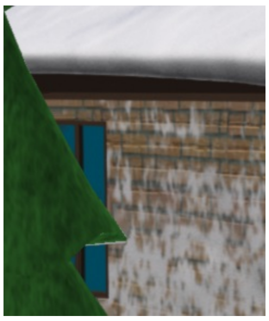

(a) Frame 0 (pre inpainting).

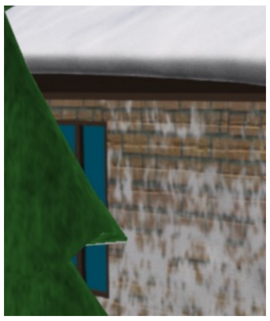

(b) Frame 26 (pre inpainting).

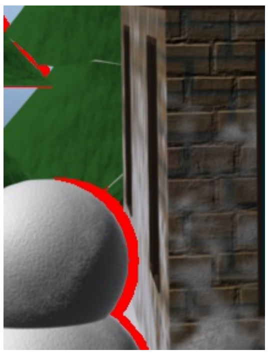

(c) Frame 100 (pre inpainting).

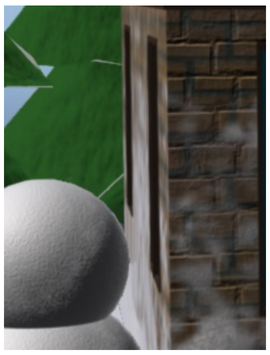

(d) Frame 26 detail.

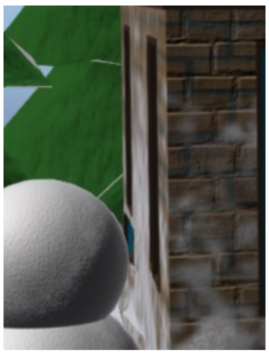

(e) Guidefill (pre edit).


(f) Guidefill (post edit).


(g) Newson's Method.


(h) Frame 100 detail.


(i) Guidefill (no edit).


(j) Newson's Method.

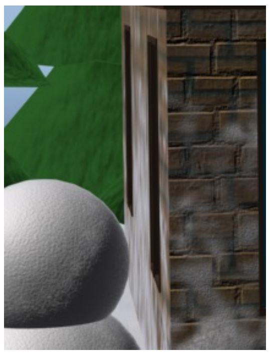
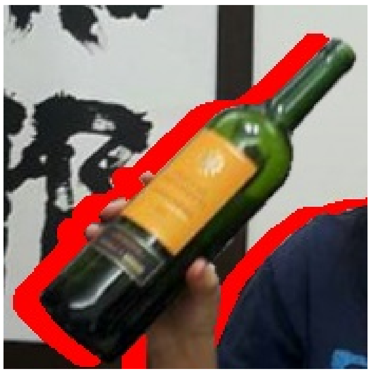
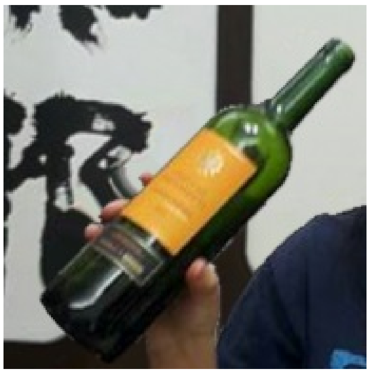
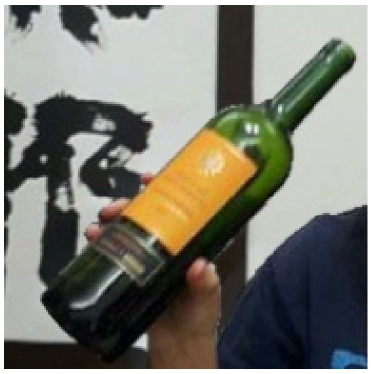
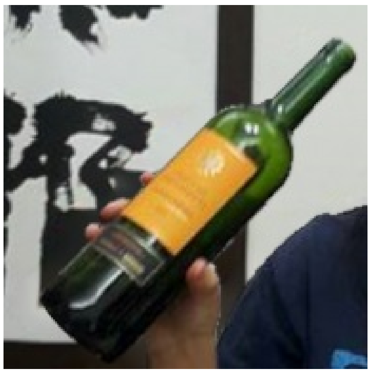

(k) Ground Truth.

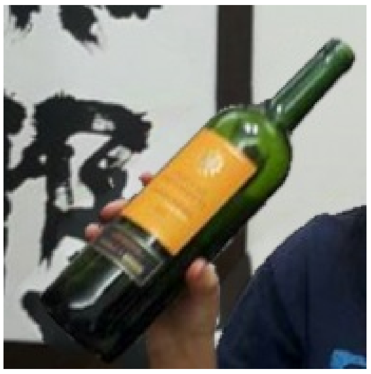
Figure 4.9: Comparison of Guidefill (19s with tracking, 31s without) and Newson's method (5hr37min) for inpainting the "cracks" (shown in red) arising in the 3d conversion of an HD video (1280px×960px×101fr). Guidefill produces artifacts such as the incorrectly extrapolated window in (e), but these can be corrected as in (f) and it is several orders of magnitude faster than Newson's method (which also required more than 16GB of RAM in this case). The latter produces very high quality results, but is prohibitively expensive and still produces a few artifacts as in (j), which the user has no recourse to correct. A disadvantage of Guidefill is a flickering as the video is viewed through time due to the frames being inpainted independently.

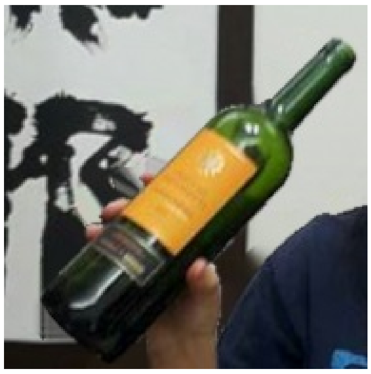

(a) Detail one.

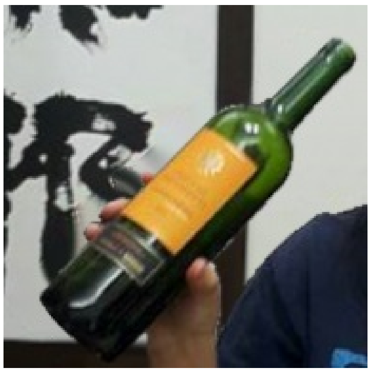

(b) Coherence transport - note the bending of the picture frame.

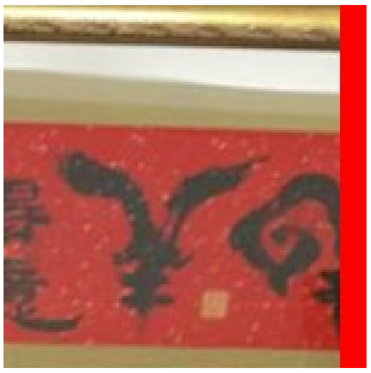

(c) nl-means - good result, but slow.

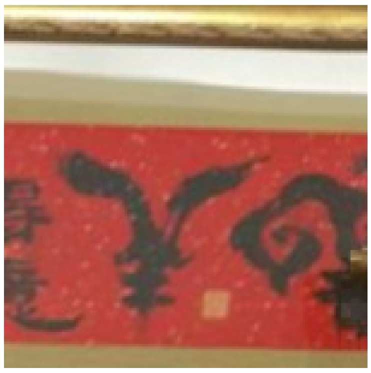

(d) nl-Poisson - Chinese characters are a solid block.

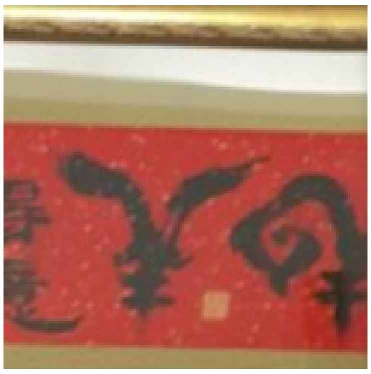

(e) Content-Aware Fill - distorted picture frame.

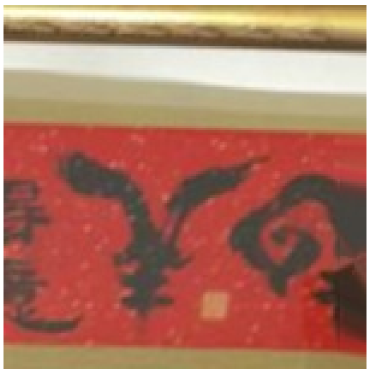

(f) Guidefill (before spline adjustment) - numerous issues.


(g) Guidefill (after adjustment) - issues are mostly resolved.


(h) Detail two.


(i) Bystander-Aware Criminisi - a piece of the picture frame is used to extrapolate the drawing.


(j) nl-Poisson - good result, but slow.


(k) Guidefill (before spline adjustment) - extension of drawing does not look natural.

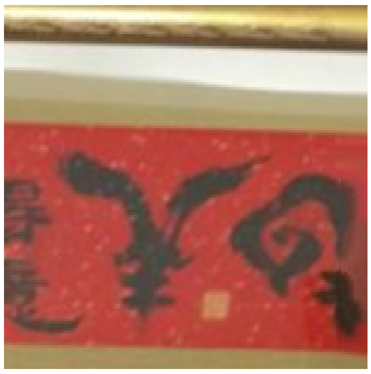
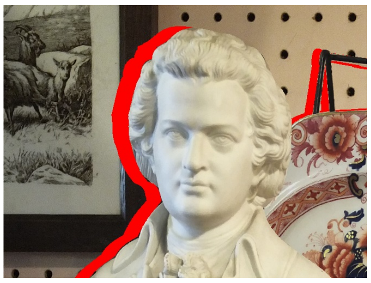
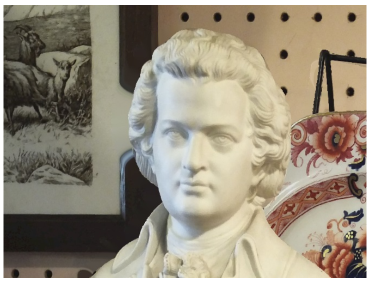
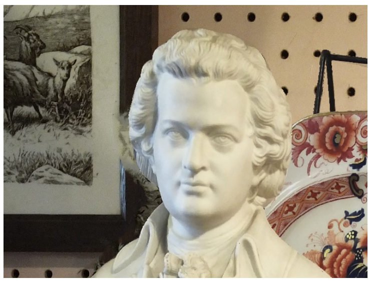
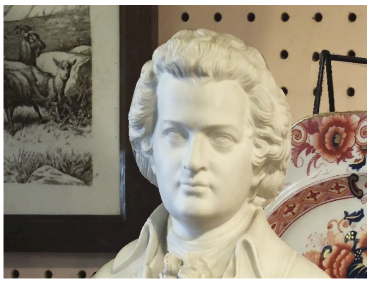

(l) Guidefill (after adjustment) - more believable extrapolation.

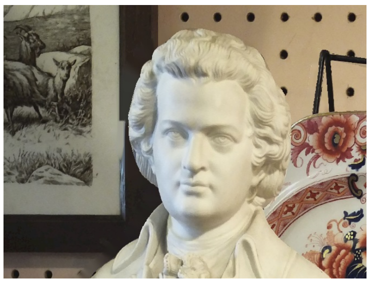
Figure 4.10: Comparison of different inpainting methods for the "Wine" example. Two challenging areas are shown.

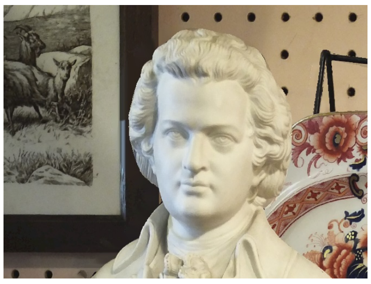

(a) Detail of "Bust".

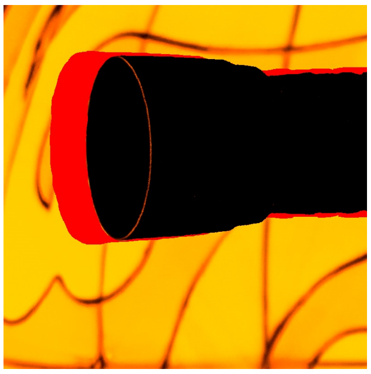

(b) Coherence transport.

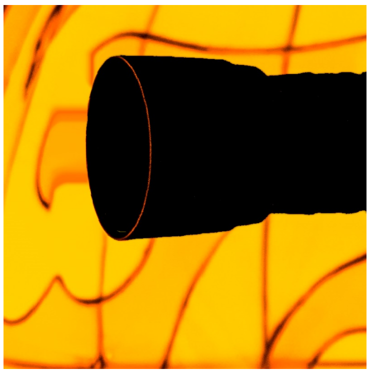

(c) Content-Aware Fill.

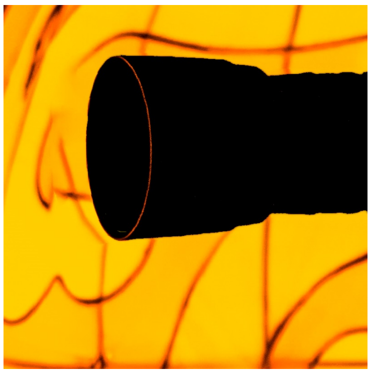

(d) nl-Poisson.

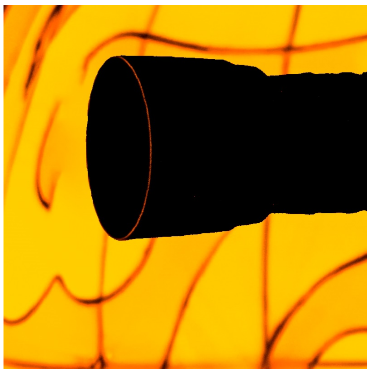

(e) Guidefill (before spline adjustment).

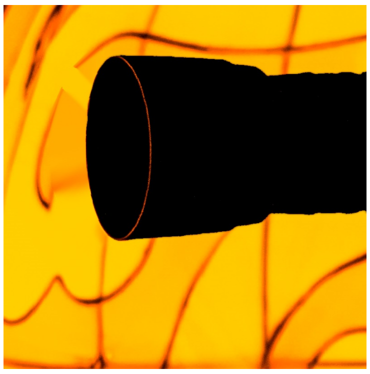

(f) Guidefill (after adjustment).


(g) Detail of "Pumpkin".


(h) Coherence transport.


(i) Content-Aware Fill.


(j) nl-Poisson.


(k) Guidefill (before spline adjustment).

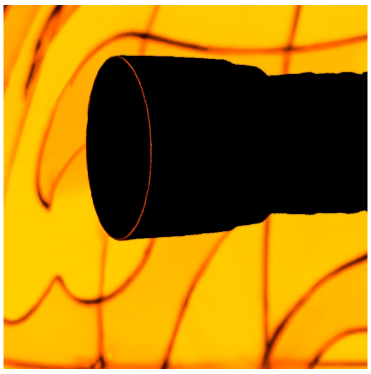
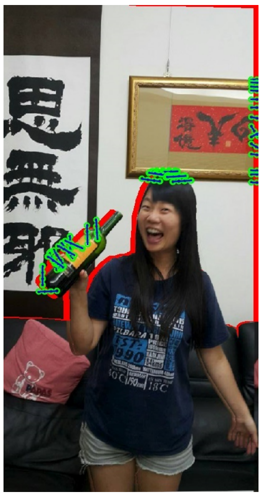
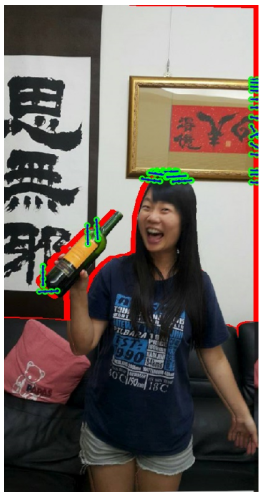
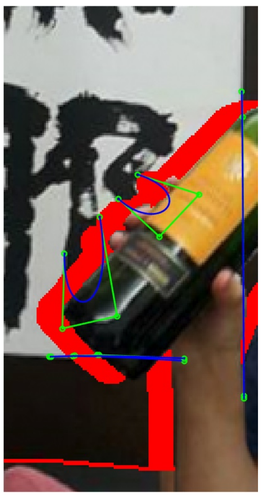

(l) Guidefill (after adjustment).

Figure 4.11: Comparison of different inpainting methods for the "Bust" and "Pumpkin" examples. Note the shock visible in the inpainted picture frame in (f), as discussed in Section 2.3.2.


(a)


(b)


(c)

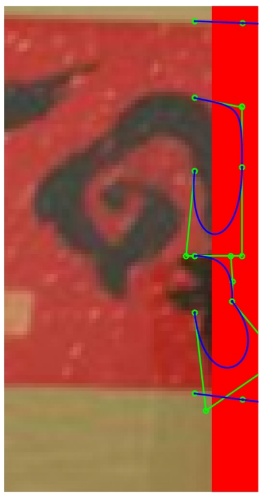
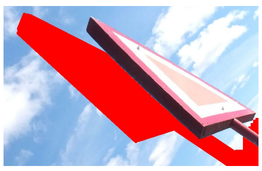
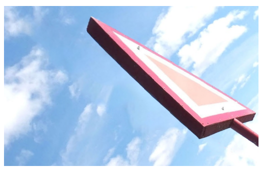
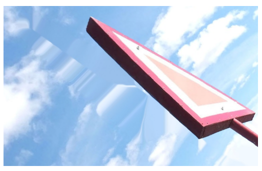

(d)

Figure 4.12: Stages of spline adjustment for the "Wine" example: (a) The automatically detected splines for the background object. (b) Undesirable splines are deleted. (c) Deleted splines are replaced with new splines, drawn by hand, which form a plausible extension of the disoccluded characters. (d) Some of the remaining splines on the painting in the upper right corner are edited to form a more believable extension.


(a) Detail of "Planet".


(b) Content-Aware Fill.


(c) coherence transport.

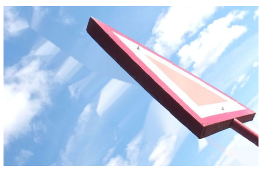
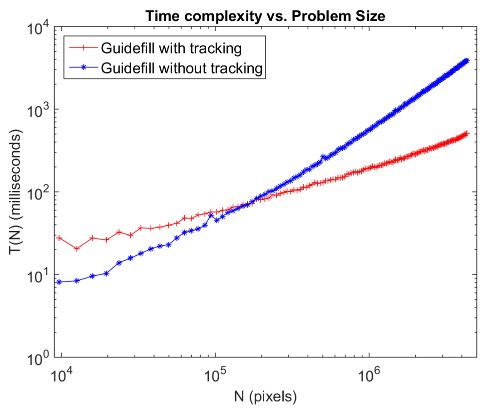
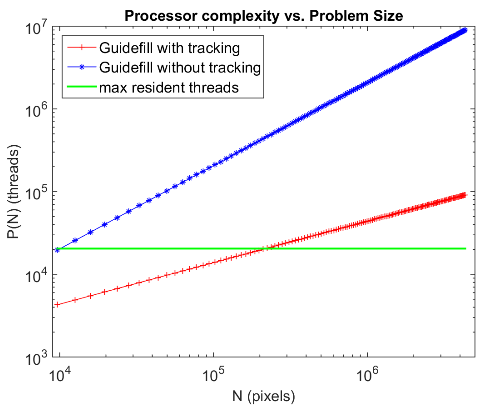

(d) Guidefill (no spline adjustment).

Figure 4.13: Comparison of different inpainting methods for the "Planet" example. In this case geometric methods leave noticeable artifacts and exemplar-based methods like Content-Aware Fill are a better choice.



(a) Time Complexity



(b) Processor Complexity

Figure 4.14: **Experimental time complexity $T(N)$ and processor complexity $P(N)$ of Guidefill with and without boundary tracking:** The continuum inpainting problem $\Omega = [0,4] \times [0,1]$, $D = [0.4, 3.96] \times [0.2, 0.8]$ was discretized at a variety of resolutions leading to inpainting domains with $N := |D_h|$ varying from $N \approx 10^4$px up to $N \approx 10^6$px. Results are given on a loglog scale to emphasize the approximate power law $T(N) \approx AN^\alpha$, $P(N) \approx BN^\beta$. A least squares fit gives $\alpha = 1.1$, $\beta = 1.0$ without tracking, and $\alpha = 0.54$, $\beta = 0.5$ with tracking (see Section 4.6 for a review of these terms). The superior scaling law of Guidefill with tracking kicks in around $N \approx 2 \cdot 10^5$. Processor complexity is compared with the maximum number of resident threads (green line).

here we show only a detail in Figure 4.13. In this example the inpainting domain is relatively wide and the image is dominated by texture. As a result, geometric methods are a bad choice and exemplar-based methods are more suitable.

4.9.2 Validation of Complexity Analysis

As stated in Section 4.8, our analysis assumes that Guidefill is implemented on a parallel architecture consisting of p identical processors acting in parallel. In reality, GPU architecture is more complex than this, but as a rough approximation, we assume $p = 20480$, the maximum number of resident threads allowed for our particular GPU. See Appendix A.5 for a deeper discussion.

In order to explore experimentally the time and processor complexity of Guidefill, we considered the continuum problem of inpainting the line $0.45 \le y \le 0.55$ across the inpainting domain $D = [0.4, 3.96] \times [0.2, 0.8]$ with image domain $\Omega = [0,4] \times [0,1]$. This continuum problem was then rendered at a series of resolutions varying from as low as 280×70px all the way up to 4000×1000px. The resulting series of discrete inpainting problems were solved using Guidefill. For simplicity, smart order was disabled and splines were turned off. In each case, we measured the execution time $T(N)$ of Guidefill as well as the maximum number of requested threads $P(N)$, with and without tracking. Results are shown in Figure 4.14 - note the loglog scale. In Figure 4.14(b) we have also indicated the value of p for comparison - note that for Guidefill without tracking we have $P(N) \gg p$ for all but the smallest problems, but for Guidefill with tracking we have $P(N) < p$ up until $N \approx 2 \times 10^5$.

Based on Theorem 4.8.1 and Theorem 4.8.2 for Guidefill without tracking we expect $T(N) \in O(N^{1.5})$ for all N, but for Guidefill with tracking we expect
$T(N) \in O(N^{0.5} \log(N))$ for N up to about 10^5px (where $P(N) \approx p$), with somewhat worse performance as N grows larger, converging to $O(N \log(N))$ when $P(N) \gg p$. To test these expectations we assume a power law of $T(N) \approx AN^\alpha$ and solve for α using least squares. The results are $\alpha = 0.54$ and $\alpha = 1.10$ for Guidefill with and without tracking respectively. Assuming a similar power law $P(N) \approx BN^\beta$ gives

$\beta = 1.0$, $\beta = 0.5$ for Guidefill without and with tracking respectively. These results suggest that the analysis in Section 4.8 does a reasonable job of predicting the rough behaviour of our method in practice.

4.10 Conclusions

We have presented a fast inpainting method suitable for use in the hole-filling step of a 3D conversion pipeline used in film, which we call Guidefill. Guidefill is non-texture based, exploiting the fact that the inpainting domains in 3D conversion tend to be in the form of a thin "crack" such that texture can often be neglected. Its fast processing time and its setup allowing intuitive, user-guided amendment of the inpainting result render Guidefill into a user-interactive inpainting tool. A version of Guidefill is in use by the stereo artists at the 3D conversion company Gener8, where it has been used in major Hollywood blockbusters such as Mockingjay, Pan, and Maleficent. In those cases where it is suitable, especially scenes dominated by structure rather than texture and/or thin inpainting domains, Guidefill produces results that are competitive with alternative algorithms in a tiny fraction of the time. In practice, Guidefill was found to be particularly useful for movies with many indoor scenes dominated by structure, and less useful for movies taking place mainly outdoors, where texture dominates. Because of its speed, artists working on a new scene may apply our method first. If the results are unsatisfactory, they can edit the provided splines or switch to a more expensive method.

In addition to its use as an algorithm for 3D conversion, Guidefill belongs to a broader class of fast geometric inpainting algorithms also including Telea's Algorithm [64] and coherence transport [12, 44]. Similarly to these methods, Guidefill is based on the idea of filling the inpainting domain in shells while extrapolating isophotes based on a transport mechanism. However, Guidefill improves upon these methods in several important respects, including the elimination of two forms of kinking of extrapolated isophotes. In one case this is done by summing over a non-axis aligned ball of "ghost pixels", which as far as we know has never been done in the literature.

We have also presented a theoretical analysis of our method and methods like it, by considering a relevant continuum limit. Our limit, which is different from the one explored in [12, Theorem 1], is able to theoretically explain some of the advantages and disadvantages of both our method and coherence transport. In particular, our analysis predicts a kinking phenomenon observed in coherence transport in practice but not accounted for by the analysis in [12]. It is also to explain how our ghost pixels are able to fix this problem, while also shedding light on a new problem that they introduce - the progressive blurring of the extrapolated signal. Nonetheless, our analysis predicts that this latter effect becomes less and less significant as the image resolution increases, and our method is designed with HD in mind. More details of our analytic framework are explored in Chapter 6.

In order to make our method as fast as possible, we have implemented it on the GPU where we consider two possible implementations. A naive implementation, suitable for small images, simply assigns one GPU thread per pixel. For our second implementation, we propose an algorithm to track the inpainting interface as it evolves, facilitating a massive reduction in the number of threads required by our algorithm. This does not lead to speed up by a constant factor - rather, it changes the complexity class of our method, leading to improvements that become arbitrarily large as $N = |D_h|$ increases. In practice we observed a slight decrease in speed (compared with the naive implementation) for small images ($N \lesssim 10^5$px), and gains ranging from a factor of $2 - 6$ for larger images.

A current disadvantage of our method is that temporal information is ignored. In particular, splines are calculated for each frame separately, and inpainting is done on a frame by frame basis without consideration for temporal coherence. As a result of the former, artists must perform separate spline adjustments for every frame. In practice we find that only a minority of frames require adjustment, however one potential direction for improvement is to design a system that proposes a series of *animated* splines to the user, which they may then edit over time by adjusting control points and setting key frames. Further, a procedure for enforcing temporal coherence, if it could be implemented without significantly increasing the runtime, would be beneficial. These ideas are discussed more in Chapter 7, where we introduce spacetime transport, a 3D generalization of Guidefill that takes temporal information into account.

Chapter 5

The Semi-implicit Extension

In this chapter we go over the semi-implicit form of Algorithm 1 described in Chapter 2 and illustrated in Figure 1.2. We begin in Section 5.1 by introducing the concept of equivalent weights, which will allow us to transform any weighted sum of the colors of a set of ghost pixels into an equivalent weighted sum of the colors of a related set of real pixels, with modified weights. This will allow us in Section 5.2 to write down an explicit expression for the linear system arising in the semi-implicit form of Algorithm 1. Next, in Section 5.2.1 we propose a simple iterative method for solving this linear system (the blue text in Algorithm 1), and prove that it is equivalent to damped Jacobi or successive over-relaxation (SOR). In Section 5.2.2 we discuss the semi-implicit extension of Guidefill. Convergence analysis of our proposed iterative method, for the semi-implicit form of Guidefill, is given in Proposition 6.2.1 of Chapter 6.

5.1 Ghost pixels and equivalent weights

Because ghost pixels are defined using bilinear interpolation, any sum over a finite set of ghost pixels $A(\mathbf{x})$ can be converted into a sum over an equivalent set of real pixels with equivalent weights[1], that is

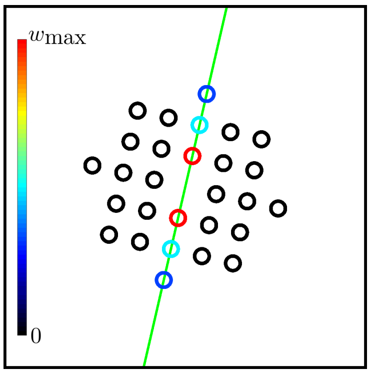
$$\sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y})u_h(\mathbf{y}) = \sum_{\mathbf{y}\in \mathrm{Supp}(A(\mathbf{x}))} \tilde{w}(\mathbf{x},\mathbf{y})u_h(\mathbf{y})$$

where $\mathrm{Supp}(A(\mathbf{x}))$ denotes the set of real pixels needed to define $u_h(\mathbf{y})$ for each $\mathbf{y}\in A(\mathbf{x})$ and $\tilde{w}$ denotes a set of equivalent weights. This works because each $u_h(\mathbf{y})$ is itself a weighted sum of the form

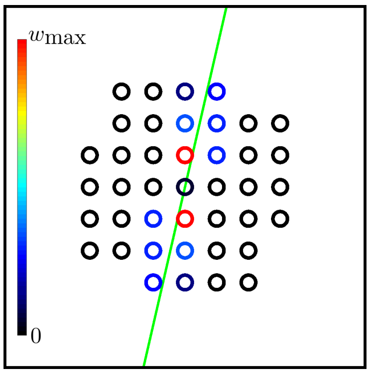
$$u_h(\mathbf{y}) = \sum_{\mathbf{z}\in\mathbb{Z}_h^2} \Lambda_{\mathbf{z},h}(\mathbf{y})u_h(\mathbf{z}),$$

where $\{\Lambda_{\mathbf{z},h}\}_{\mathbf{z}\in\mathbb{Z}_h^2}$ denote the basis functions of bilinear interpolation associated with the lattice $\mathbb{Z}_h^2$. This is illustrated in Figure 5.1(a)-(b), where we show a heat map of the weights (2.5.2) over the set $\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\}$ for $\mu = 50$ and $\epsilon = 3$px, as well as a similar heat map of the modified weights over $\mathrm{Supp}(\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\}) \subseteq D_h(B_{\epsilon,h}(\mathbf{x}))$. Note that even though $\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\}$ does not contain the point $\mathbf{x}$, the support of this set does. This will be important in Section 5.2. Here we briefly list some properties of equivalent weights, including an explicit formula. A proof is sketched, but details are deferred to Appendix A.6.

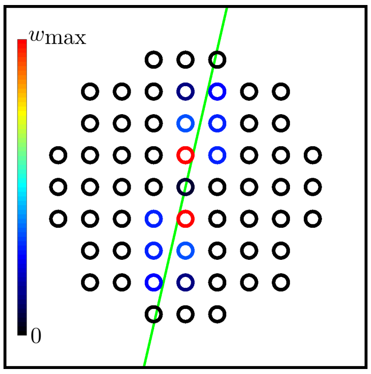
[1]note that here we mean a general family of finite sets $A(\mathbf{x}) \in \mathbb{R}^2$ and general weights $w(\mathbf{x},\mathbf{y})$. We do not mean the specific family of sets $A_{\epsilon,h}(\mathbf{x})$ or the specific weights $w_\epsilon(\mathbf{x},\mathbf{y})$, which have special properties.



(a) Heatmap of weights (2.5.2) over $\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\}$.



(b) Heatmap of equivalent weights over $\mathrm{Supp}(\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\})$.



(c) Heatmap of equivalent weights over $D_h(B_{\epsilon,h}(\mathbf{x}))$.

Figure 5.1: **Ghost pixels and equivalent weights:** Because ghost pixels are defined using bilinear interpolation, any weighted sum over a set of ghost pixels $A_{\epsilon,h}(\mathbf{x})$ is equivalent to a sum with equivalent weights over real pixels in the set $\mathrm{Supp}(A_{\epsilon,h}(\mathbf{x}))$, defined as the set of real pixels needed to define each ghost pixel $\mathbf{y}$ in $A_{\epsilon,h}(\mathbf{x})$. We illustrate this in (a)-(c) using Guidefill with $\epsilon = 3$px, $\mathbf{g} = (\cos 77°, \sin 77°)$, and $\mu = 100$. In (a), the (normalized) weights (2.5.2) are visualized as a heat map over $\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\}$. In (b), we show the equivalent weights over $\mathrm{Supp}(\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\}) \subseteq D_h(B_{\epsilon,h}(\mathbf{x}))$ (this containment comes from (5.1.7) in Remark 5.1.2). Note that even though $\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\}$ does not contain the point $\mathbf{x}$, the support of this set does. In (c), we visualize the equivalent weights over the set $D_h(B_{\epsilon,h}(\mathbf{x}))$, which is strictly larger than $\mathrm{Supp}(\tilde{B}_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\})$. For reference, we include the line parallel to $\mathbf{g}$ in green.

Properties of equivalent weights Properties 1-3 deal with a general finite set $A(\mathbf{x})$ and general weights $w(\mathbf{x},\mathbf{y})$, while properties 4-6 deal with the specific set $A_{\epsilon,h}(\mathbf{x}) \subset B_\epsilon(\mathbf{x})$ and the specific weights $w_\epsilon(\mathbf{x},\mathbf{y})$ obeying (2.0.2).

1. Explicit formula:

$$\tilde{w}(\mathbf{x},\mathbf{z}) = \sum_{\mathbf{y}\in A(\mathbf{x})} \Lambda_{\mathbf{y},h}(\mathbf{z}) w(\mathbf{x},\mathbf{y}) \tag{5.1.1}$$

2. Preservation of total mass:

$$\sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y}) = \sum_{\mathbf{y}\in \mathrm{Supp}(A(\mathbf{x}))} \tilde{w}(\mathbf{x},\mathbf{y}). \tag{5.1.2}$$

3. Preservation of center of mass (or first moment):

$$\sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y})\mathbf{y} = \sum_{\mathbf{y}\in \mathrm{Supp}(A(\mathbf{x}))} \tilde{w}(\mathbf{x},\mathbf{y})\mathbf{y}. \tag{5.1.3}$$

4. Inheritance of non-negativity:

$$\tilde{w}_\epsilon(\mathbf{x},\mathbf{z}) \geq 0 \quad \text{ for all } \mathbf{z} \in \mathrm{Supp}(A_{\epsilon,h}(\mathbf{x})). \tag{5.1.4}$$

5. Inheritance of non-degeneracy condition (2.0.2):

$$\sum_{\mathbf{y}\in \mathrm{Supp}(A_{\epsilon,h}(\mathbf{x})\cap(\Omega\backslash D^{(k)}))} \tilde{w}_\epsilon(\mathbf{x},\mathbf{y}) > 0. \tag{5.1.5}$$

6. Universal support: For any $n \in \mathbb{Z}$, we have

$$\mathrm{Supp}(A_{\epsilon,h}(\mathbf{x}) \cap \{y \leq nh\}) \subseteq D_h(B_{\epsilon,h}(\mathbf{x})) \cap \{y \leq nh\} \subseteq B_{\epsilon+2h,h}(\mathbf{x}) \cap \{y \leq nh\}. \tag{5.1.6}$$

   where $\{y \leq nh\} := \{(x,y) \in \mathbb{R}^2 : y \leq nh\}$, and where D_h is the dilation operator defined in our section on notation.

Proof. Most of these properties are either obvious or are derived based on a simple exercise in changing the order of nested finite sums. Properties (5.1.2) and (5.1.3) are slightly more interesting - they follow from the fact that the bilinear interpolant of a polynomial of degree at most one is just the polynomial again. Note that an analogous formula for preservation of the second moment does *not* hold, because a quadratic function and its bilinear interpolant are not the same thing. The last identity is based on an explicit formula for the support of a point. Details are provided in Appendix A.6. □

Remark 5.1.1. *Although we have explicit formula* (5.1.1) *for the equivalent weights which will occasionally be useful, most of the time it is more fruitful to think about them in the following way: To compute* $\tilde{w}_\epsilon(\mathbf{x}, \mathbf{y})$ *for some real pixel* $\mathbf{y}$*, loop over the ghost pixels* $\mathbf{z}$ *such that* $\mathbf{y} \in$ *Supp*$(\mathbf{z})$*. Then, each such* $\mathbf{z}$ *redistributes to* $\tilde{w}_\epsilon(\mathbf{x}, \mathbf{y})$ *a fraction of its weight* $w_\epsilon(\mathbf{x}, \mathbf{z})$ *equal to the proportion of* $u_h(\mathbf{y})$ *that went into* $u_h(\mathbf{z})$*.*

Remark 5.1.2. *An obvious corollary of the universal support property* (5.1.6) *is that we also have the containment*

$$\mathit{Supp}(A_{\epsilon,h}(\mathbf{x})) \subseteq D_h(B_{\epsilon,h}(\mathbf{x})) \subseteq B_{\epsilon+2h,h}(\mathbf{x}). \tag{5.1.7}$$

Figure 5.1 illustrates an example where this containment is strict, and in fact it is not hard to show that this holds in general. However, (5.1.6) *is tight enough for our purposes in this thesis.*

5.2 Semi-implicit extension of Algorithm 1

Here we present a semi-implicit extension of Algorithm 1, to our knowledge not previously proposed in the literature, in which instead of computing $u_h(\mathbf{x})$ for each $\mathbf{x} \in \partial_{\text{ready}} D_h^{(k)}$ independently, we solve for them simultaneously by solving a linear system. We call our method semi-implicit in order to distinguish it from fully implicit methods in which the entire inpainting domain $\{u_h(\mathbf{x}) : \mathbf{x} \in D_h\}$ is solved simultaneously, as is typically the case for most inpainting methods based on PDEs or variational principles, e.g. [11, 19, 13]. Specifically, we solve

$$\mathcal{L}\mathbf{u} = \mathbf{f} \quad \text{where } \mathbf{u} = \{u_h(\mathbf{x}) : \mathbf{x} \in \partial_{\text{ready}} D_h^{(k)}\}. \tag{5.2.1}$$

and $\mathbf{f}$ is a vector of length $|\partial_{\text{ready}} D_h^{(k)}|$. The explicit entries of $\mathcal{L}$ are written in terms of the equivalent weights $\tilde{w}_\epsilon$ introduced in Section 5.1. Defining

$$S_{\epsilon,h}^{(k)}(\mathbf{x}) := \mathrm{Supp}(A_{\epsilon,h}(\mathbf{x})) \cap \partial_{\text{ready}} D_h^{(k)},$$

it follows that $\mathcal{L}$ couples each $\mathbf{x} \in \partial_{\text{ready}} D_h^{(k)}$ to its immediate neighbors in $S_{\epsilon,h}^{(k)}(\mathbf{x})$. In particular, we have

$$(\mathcal{L}\mathbf{u})(\mathbf{x}) = \left(1 - \frac{\tilde{w}_\epsilon(\mathbf{x},\mathbf{x})}{W}\right) u_h(\mathbf{x}) - \sum_{\mathbf{y} \in S_{\epsilon,h}^{(k)}(\mathbf{x}) \setminus \{\mathbf{x}\}} \frac{\tilde{w}_\epsilon(\mathbf{x},\mathbf{y})}{W} u_h(\mathbf{y}), \tag{5.2.2}$$

where W is the total mass and can be computed in one of two ways, using either the original weights w_ϵ or the equivalent weights $\tilde{w}_\epsilon$, exploiting preservation of mass (5.1.2):

$$\begin{aligned} W &:= \sum_{\mathbf{y}\in S_{\epsilon,h}^{(k)}(\mathbf{x})\cup\left(\mathrm{Supp}(A_{\epsilon,h}(\mathbf{x}))\cap(\Omega_h\backslash D_h^{(k)})\right)} \tilde{w}_\epsilon(\mathbf{x},\mathbf{y}) && (5.2.3)\\ &= \sum_{\mathbf{y}\in(A_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\})\cap(\Omega\backslash D^{(k)})} w_\epsilon(\mathbf{x},\mathbf{y}). && (5.2.4)\end{aligned}$$

Generally, (5.2.4) is more convenient to work with than (5.2.3), but (5.2.3) combined with the inherited non-degeneracy condition (5.1.5) tells us that

$$\sum_{\mathbf{y}\in S_{\epsilon,h}^{(k)}(\mathbf{x})} \tilde{w}_\epsilon(\mathbf{x},\mathbf{y}) < W,$$

because the latter implies that a non-zero proportion of the total weight goes into the known pixels in $\mathrm{Supp}(A_{\epsilon,h}(\mathbf{x}))\cap(\Omega\backslash D_h^{(k)})$ rather than the unknown pixels in $S_{\epsilon,h}^{(k)}(\mathbf{x})$. From this it immediately follows that $\mathcal{L}$ is strictly diagonally dominant - a property we will use later. To compute $\mathbf{f}$, we do not need the concept of equivalent weights. We have

$$\mathbf{f}(\mathbf{x}) = \sum_{\mathbf{y}\in(A_{\epsilon,h}(\mathbf{x})\backslash\{\mathbf{x}\})\cap(\Omega\backslash D^{(k)})} \frac{w_\epsilon(\mathbf{x},\mathbf{y})}{W}u_h(\mathbf{y}). \qquad (5.2.5)$$

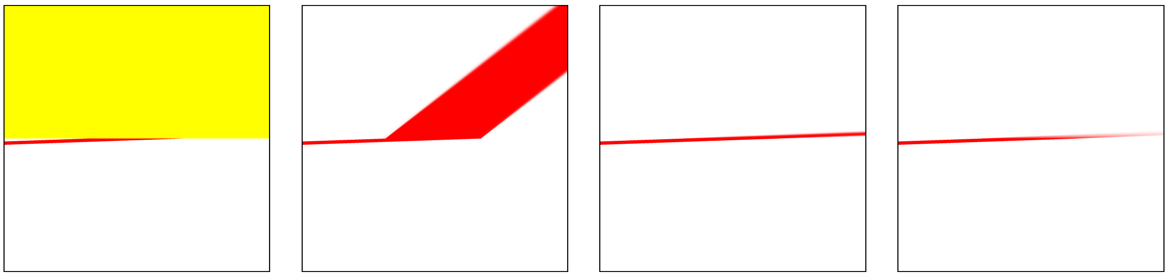
5.2.1 Solving the linear system

Designing maximally efficient methods for solving (5.2.1) is beyond the scope of this thesis - our main purpose lies in understanding the effect of this extension on the continuum limit that we will derive later. Therefore, we consider only two very simple methods: damped Jacobi and SOR (successive over-relaxation). These are natural choices for a number of reasons. First, since $\mathcal{L}$ is strictly diagonally dominant, these methods are both guaranteed to converge [67, Theorem 3.10, pg. 79], at least in the case $\omega = 1$, where they reduce to Jacobi and Gauss-Seidel. Second, at least for the semi-implicit extension of Guidefill, the performance of SOR is already satisfactory, (see Section 6.2, Proposition 6.2.1). Third, both methods can be implemented with minimal changes to the direct form of Algorithm 1. In fact, changing the variable "semiImplicit" from "false" to "true" in Algorithm 1 and executing the "FillBoundary" subroutine in parallel is equivalent to solving (5.2.1) using damped Jacobi, with underrelaxation parameter

$$\omega^* = \left(1 - \frac{\tilde{w}_\epsilon(\mathbf{x},\mathbf{x})}{W}\right) \leq 1. \qquad (5.2.6)$$

Similarly, executing the "FillBoundary" subroutine sequentially results in SOR with the same underrelaxation parameter. We will prove this in a moment in Proposition 5.2.2, however first we build intuition with Remark 5.2.1. Note that ω^* is typically very close 1, so these methods are very similar to plain Jacobi and Gauss-Seidel.

Remark 5.2.1. *The reason Algorithm 1 with "semiImplicit" set to "true" results in damped Jacobi/SOR, rather than plain Jacobi/Gauss-Seidel, is because even though the update formula* (2.0.3) *for the nth iterate $u_h^{(n)}(\mathbf{x})$ is expressed as a sum of the $(n-1)$st iterate evaluated at* ghost pixels *that do not include $\mathbf{x}$, some of those ghost pixels may indirectly depend on $u_h^{(n-1)}(\mathbf{x})$ because they are defined using bilinear interpolation. The result is that the nth iterate $u_h^{(n)}(\mathbf{x})$ depends on $u_h^{(n-1)}(\mathbf{x})$, which is true of damped Jacobi/SOR but not of Jacobi/Gauss-Seidel.*



(a) Original inpainting problem, inpainting domain in yellow.

(b) Inpainting with Guidefill.

(c) Inpainting with semi-implicit Guidefill, 5 SOR iterations per shell.

(d) Inpainting with semi-implicit Guidefill, 50 damped Jacobi iterations per shell.

Figure 5.2: **Semi-Implicit Guidefill and Shallow Inpainting Directions:** In (a), a line makes a very shallow angle of just 2° with the inpainting domain, shown in yellow. This line is then inpainted using first Guidefill (b) and then the semi-implicit extension thereof (c). The latter uses 5 iterations of SOR per shell to solve the linear system (5.2.1) arising in every shell (in this case, the original image is 2000×2000px, so there are 1000 shells). Visually identical results can be obtained using damped Jacobi, but more than 100 iterations per shell are required - see Proposition 6.2.1. Both methods use relaxation parameter $\omega = \omega^*$ (5.2.6) and are given $\mathbf{g} = (\cos 2°, \sin 2°)$, $\mu = 100$, $\epsilon = 3$px, and use the default onion shell ordering (smart-order is turned off). Guidefill kinks while its extension does not. In (d), we see the result of failing to solve the linear system (5.2.1) to sufficient accuracy by applying too few iterations of damped Jacobi. In this case only 50 iterations per shell are used and the extrapolated line gradually fades away.

Proposition 5.2.2. *Changing the boolean "semiImplicit" to true in Algorithm 1 is equivalent to solving* (5.2.1) *with damped Jacobi if "FillBoundary" is executed in parallel, and to SOR if it is executed sequentially. In either case, the relaxation parameter is given by*

$$\omega^* = \left(1 - \frac{\tilde{w}_\epsilon(\mathbf{x},\mathbf{x})}{W}\right).$$

Proof. First, note that the Jacobi iteration for solving the linear system (5.2.1) may be written as

$$\tilde{u}_h^{(n+1)}(\mathbf{x}) = \frac{1}{1 - \frac{\tilde{w}_\epsilon(\mathbf{x},\mathbf{x})}{W}} \left(\sum_{\mathbf{y} \in S^{(k)}_{\epsilon,h}(\mathbf{x}) \setminus \{\mathbf{x}\}} \frac{\tilde{w}_\epsilon(\mathbf{x},\mathbf{y})}{W} u_h^{(n)}(\mathbf{y}) + \mathbf{f} \right)$$

with $\mathbf{f}$ defined as in (5.2.5). By comparison, repeated (parallel) executation of FillBoundary$(D_h^{(k+1)}, \partial D_h^{(k)})$ is equivalent (after applying the definition of equivalent weights) to

$$\begin{aligned} u_h^{(n+1)}(\mathbf{x}) &= \sum_{\mathbf{y} \in S^{(k)}_{\epsilon,h}(\mathbf{x})} \frac{\tilde{w}_\epsilon(\mathbf{x},\mathbf{y})}{W} u_h^{(n)}(\mathbf{y}) + \mathbf{f} \\ &= \frac{\tilde{w}(\mathbf{x},\mathbf{x})}{W} u_h^{(n)}(\mathbf{x}) + \sum_{\mathbf{y} \in S^{(k)}_{\epsilon,h}(\mathbf{x}) \setminus \{\mathbf{x}\}} \frac{\tilde{w}_\epsilon(\mathbf{x},\mathbf{y})}{W} u_h^{(n)}(\mathbf{y}) + \mathbf{f} \\ &= (1 - \omega^*) u_h^{(n)}(\mathbf{x}) + \omega^* \tilde{u}_h^{(n+1)}(\mathbf{x}), \end{aligned}$$

which is a definition of damped Jacobi. The proof for SOR is analogous. □

5.2.2 Semi-implicit Guidefill

In this thesis we particularly interested in the extension of Guidefill since, as illustrated in Figure 5.2 and as we will prove in Section 6.6.2, it is able to overcome the issue of kinking that we saw with Guidefill for shallow angles in Figure 2.6 (and see again in 5.2(b)). In Section 6.2 we will analyze the convergence of damped Jacobi and SOR for solving the linear system (5.2.1) arising in this method - see Proposition 6.2.1 in particular. Special attention will be paid to the parameter value $\omega = \omega^*$ (5.2.6) that naturally arises from the implementation we have proposed in Algorithm 1 (blue text). First, however, we note that beyond changing the boolean variable "semiImplicit" to "true" in Algorithm 1, this extension performs optimally only if we also change the definition of the "ready" function. If we didn't do this, whenever a very shallow line such as the one in Figure 5.2 was encountered, the ready function would tell us not to try to inpaint it. This is because (4.5.3) takes into account only information about pixels that have already been filled, but now $u_h(\mathbf{x})$ is constructed using information from its neighbors in $\partial D_h^{(k)}$ that are being filled at the same time. The modified "ready" function must reflect this. The idea is to allow $\text{ready}(\mathbf{x})$ to depend on pixels belonging to the current shell, but only if those pixels are also "ready" to be filled. Thus, just like the colors of pixels in the current shell are now coupled together, the binary values $\{\text{ready}(\mathbf{x}) : \mathbf{x} \in \partial D_h^{(k)}\}$ are coupled as well.

First, let us fix some notation. We define $\tilde{D}_h^{(k+1)} = D_h^{(k)} \backslash \partial D_h^{(k)}$, that is, the inpainting domain as it would be on the next step if all of $\partial D_h^{(k)}$ were filled (clearly $D_h^{(k)} \supseteq \tilde{D}_h^{(k)}$, as we never fill *more* than $\partial D_h^{(k)}$ on iteration k). Then we denote the continuous version of $\tilde{D}_h^{(k+1)}$ by $\tilde{D}^{(k+1)}$, defined in the usual way as in the notational section. Next, defining a pixel that has already been filled to be "ready" by default, and a ghost pixel to be "ready" if and only if the real pixels needed to define it are all "ready", we write the modified confidence $C^*(\mathbf{x})$ in terms of $\text{ready}(\mathbf{y})$ for $\mathbf{y}$ neighboring $\mathbf{x}$ as

$$\begin{aligned} C^*(\mathbf{x}) &= \frac{\sum_{\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x}) \cap (\Omega \backslash \tilde{D}^{(k+1)})} w_\epsilon(\mathbf{x},\mathbf{y}) \text{ready}(\mathbf{y})}{\sum_{\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x})} w_\epsilon(\mathbf{x},\mathbf{y})} \\ &= C(\mathbf{x}) + \frac{\sum_{\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x}) \cap (D^{(k)} \backslash \tilde{D}^{(k+1)})} w_\epsilon(\mathbf{x},\mathbf{y}) \text{ready}(\mathbf{y})}{\sum_{\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x})} w_\epsilon(\mathbf{x},\mathbf{y})} \\ &= C(\mathbf{x}) + \Delta C(\mathbf{x}, \mathbf{r}) \end{aligned}$$

where $C(\mathbf{x})$ is defined by (4.5.2) and $\mathbf{r} : \partial D_h^{(k)} \to \{0,1\}$ is defined by

$$\mathbf{r}(\mathbf{x}) = \text{ready}(\mathbf{x}).$$

Finally the "ready" function at $\mathbf{x}$ is coupled to that of its neighbors by

$$\text{ready}(\mathbf{x}) = 1(C(\mathbf{x}) + \Delta C(\mathbf{x}, \mathbf{r}) > c) \tag{5.2.7}$$

Let $\partial^0_{\text{ready}} D_h^{(k)}$, $\partial_{\text{ready}} D_h^{(k)}$ denote the portion of $\partial D_h^{(k)}$ that is "ready" to be filled, according to standard and semi-implicit Guidefill respectively. Then, since $\Delta C(\mathbf{x}, \mathbf{r}) \geq 0$, we clearly have

$$\partial^0_{\text{ready}} D_h^{(k)} \subseteq \partial_{\text{ready}} D_h^{(k)} \subseteq \partial D_h^{(k)}.$$

In other words, semi-implicit Guidefill will always agree with Guidefill that a pixel is ready to be filled, but may decide that other pixels Guidefill believes are not ready to be filled are actually ready. In Algorithm 2 we propose a simple, iterative, and parallel algorithm to solve for $\partial_{\text{ready}} D_h^{(k)}$, at least approximately.

Note that in iteration 0, we have $\Delta C(\mathbf{x}, \mathbf{r}) \equiv 0$, so that $C^*(\mathbf{x}) = C(\mathbf{x})$ and hence after one iteration $\partial_{\text{ready}} D_h^{(k)} = \partial^0_{\text{ready}} D_h^{(k)}$ (that is, it is the same as the set of ready pixels determined by standard,

Algorithm 2 Semi-Implicit Guidefill's ready function

```
$\mathbf{r} : \partial D_h^{(k)} \to \{0,1\}$, initialized to 0 everywhere.
$\partial_{\text{ready}} D_h^{(k)} = \{\mathbf{x} \in \partial D_h^{(k)} : r(\mathbf{x}) = 1\}$.
maxIt : maximum number of iterations.
$k = 0$.
while $|\partial_{\text{ready}} D_h^{(k)}|$ keeps growing and $k <$ maxIt do
    for $\mathbf{x} \in \partial D_h^{(k)}$ do
        if $C(\mathbf{x}) + \Delta C(\mathbf{x}, \mathbf{r}) > c$ then
            $\mathbf{r}(\mathbf{x}) = 1$.
        end if
    end for
    $\partial_{\text{ready}} D_h^{(k)} = \mathbf{r}^{-1}(\{1\})$.
    $k = k + 1$
end while
```

non-implicit Guidefill). In subsequent iterations $\partial_{\text{ready}} D_h^{(k)}$ can only grow, but it must stop growing within finitely many iterations, even if we set maxIt $= \infty$, since $|\partial D_h^{(k)}| < \infty$. We will not attempt to prove that the $\partial_{\text{ready}} D_h^{(k)}$ output by Algorithm 2 is equal the $\partial_{\text{ready}} D_h^{(k)}$ defined by (5.2.7). In the present work we do not analyze the benefits of this modified "ready" function - this will be the subject of future work.

Chapter 6

Analysis

This chapter contains our core analysis, firstly of the convergence properties of semi-implicit Guidefill, then of the convergence of both Algorithm 1 and its semi-implicit extension to a continuum limit as $(h, \epsilon) \to (0, 0)$ along the ray $\epsilon = rh$, then finally how under certain additional hypotheses we can also converge to the original high-resolution and vanishing viscosity limit proposed by Bornemann and März ($h \to 0$ first and then $\epsilon \to 0$). In Section 4.7.1, we will apply our results to explain some of the artifacts discussed in Section 2.2. We begin with some symmetry assumptions that will hold throughout the chapter.

6.1 Symmetry assumptions

We will assume throughout that the inpainting domain D is the unit square
$(0, 1] \times (0, 1]$ while the image domain is $\Omega = (0, 1] \times (-\delta, 1]$ equipped with Dirichlet or periodic boundary conditions at $x = 0$ and $x = 1$, and no condition at $y = 1$. We denote the undamaged portion of the image by

$$\mathcal{U} := (0, 1] \times (-\delta, 0] \qquad \mathcal{U}_h := \mathcal{U} \cap \mathbb{Z}^2_h. \tag{6.1.1}$$

We discretize $D = (0, 1]^2$ as an $N \times N$ array of pixels $D_h = D \cap (h \cdot \mathbb{Z}^2)$ with pixel width $h := 1/N$. In order to ensure that the update formula (2.0.3) is well defined, we need $\epsilon + 2h < \delta$, which we achieve by assuming $h < \frac{\delta}{r+2}$ (this follows from the inclusion (5.1.7)). We assume that the default onion shell ordering is used, so that

$$\partial D_h^{(k)} = \{(jh, kh)\}_{j=1}^{N}.$$

We also assume that the sets $A_{\epsilon,h}(\mathbf{x})$ are translations of one another, and the weights $w_\epsilon(\mathbf{x}, \mathbf{y})$ depend only on $\frac{\mathbf{y}-\mathbf{x}}{\epsilon}$, that is

$$w_\epsilon(\mathbf{x}, \mathbf{y}) = \hat{w}\left(\frac{\mathbf{y} - \mathbf{x}}{\epsilon}\right)$$

For coherence transport and Guidefill this means that the guidance direction $\mathbf{g}$ is a constant.

Remark 6.1.1. *We make the above assumptions not because we believe they are necessary, but because they enable us to make our analysis as simple as possible while still capturing the phenomena we would like to capture. In particular, the above two assumptions on the weights w_ϵ and neighborhood $A_{\epsilon,h}(\mathbf{x})$ ensure that the matrix $\mathcal{L}$ given by* (5.2.1) *is either Toeplitz or circulant (depending on the boundary conditions), and also ensures that the random walk we connect Algorithm 1 to in Section 6.3 has i.i.d. (independent identically distributed) increments. Without these simplifications, our already lengthy analysis would* *become even more technical. Numerical experiments in Section 6.8 suggest that these assumptions can be weakened, but proving this is beyond the scope of the present work.*

These assumptions give us a high level of symmetry with which we may rewrite the update formula (2.0.3) of Algorithm 1 in the generic form

$$u_h(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in a_r^*} w_r(\mathbf{0},\mathbf{y}) u_h(\mathbf{x}+h\mathbf{y})}{\sum_{\mathbf{y}\in a_r^*} w_r(\mathbf{0},\mathbf{y})} \tag{6.1.2}$$

where

$$a_r^* = \left(\frac{1}{h}A_{\epsilon,h}(\mathbf{0})\backslash\{\mathbf{0}\}\right) \cap \{y \le \delta\},$$

and $\delta = -1$ for the direct method, while $\delta = 0$ for the semi-implicit extension. In particular, for coherence transport we have $a_r^* = b_r^-$ for the direct method and $a_r^* = b_r^0$ for the extension, where

$$\begin{aligned} b_r^0 &:= \{(n,m)\in\mathbb{Z}^2 : 0 < n^2+m^2 \le r^2, m \le 0\}. & (6.1.3)\\ b_r^- &:= \{(n,m)\in\mathbb{Z}^2 : n^2+m^2 \le r^2, m \le -1\}. & (6.1.4)\end{aligned}$$

Similarly, for Guidefill, we have $a_r^* = \tilde{b}_r^-$ for the direct method and $a_r^* = \tilde{b}_r^0$ for the semi-implicit extension, where

$$\begin{aligned} \tilde{b}_r^0 &:= \{n\hat{\mathbf{g}} + m\hat{\mathbf{g}}^\perp : (n,m)\in\mathbb{Z}^2, 0 < n^2+m^2 \le r^2, n\hat{\mathbf{g}}\cdot e_2 + m\hat{\mathbf{g}}^\perp\cdot e_2 \le 0\}\\ \tilde{b}_r^- &:= \{n\hat{\mathbf{g}} + m\hat{\mathbf{g}}^\perp : (n,m)\in\mathbb{Z}^2, n^2+m^2 \le r^2, n\hat{\mathbf{g}}\cdot e_2 + m\hat{\mathbf{g}}^\perp\cdot e_2 \le -1\},\end{aligned}$$

and $\hat{\mathbf{g}} := \mathbf{g}/\|\mathbf{g}\|$ (if $\mathbf{g} = \mathbf{0}$ we set $\tilde{b}_r^- = b_r^-$). The sets b_r^-, b_r^0, $\tilde{b}_r^-$, $\tilde{b}_r^0$ may be visualized by looking at the portion of Figure 2.9(a)-(b) on or below the lines $y = 0$ and $y = -1$ respectively. Also important are the dilated sets $\bar{b}_r^0 = D(b_r^0) \cap \{y \le 0\}$ and $\bar{b}_r^- = D(b_r^-) \cap \{y \le -1\}$. These sets are given explicitly by

$$\begin{aligned} \bar{b}_r^0 &:= \{(n+\Delta n, m+\Delta m)\\ &: (n,m)\in b_r^0, (\Delta n,\Delta m)\in\{-1,0,1\}\times\{-1,0,1\}, m+\Delta m \le 0\} & (6.1.5)\\ \bar{b}_r^- &:= \{(n+\Delta n, m+\Delta m)\\ &: (n,m)\in b_r^-, (\Delta n,\Delta m)\in\{-1,0,1\}\times\{-1,0,1\}, m+\Delta m \le -1\}. & (6.1.6)\end{aligned}$$

This is because the universal support property (5.1.6) gives us the inclusion

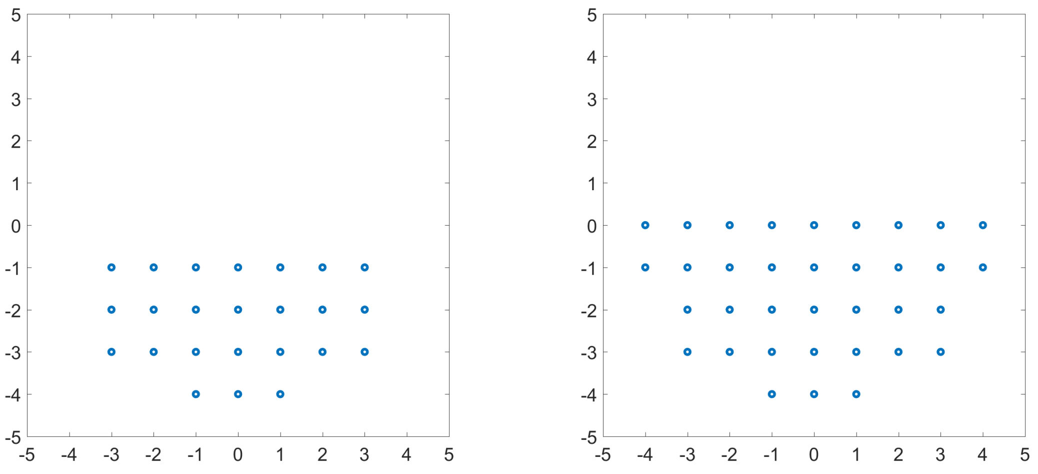
$$\text{Supp}(a_r^*) \subseteq \begin{cases} \bar{b}_r^- \subseteq b_{r+2}^- & \text{if we use the direct form of Algorithm 1.}\\ \bar{b}_r^0 \subseteq b_{r+2}^0 & \text{if we the semi-implicit extension,}\end{cases} \tag{6.1.7}$$

which will be critical later. The sets $\bar{b}_r^-$ and $\bar{b}_r^0$ are illustrated in Figure 6.1.

Definition 6.1.2. *We call the set a_r^* and the weights $\{w_r(\mathbf{0},\mathbf{y}) : \mathbf{y}\in a_r^*\}$ the* stencil *and* stencil weights *of a method. The* center of mass *of a_r^* is defined in the following two equivalent ways:*

$$C.M. = \frac{\sum_{\mathbf{y}\in a_r^*} w_r(0,\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y}\in a_r^*} w_r(0,\mathbf{y})} = \frac{\sum_{\mathbf{y}\in Supp(a_r^*)} \tilde{w}_r(0,\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y}\in Supp(a_r^*)} \tilde{w}_r(0,\mathbf{y})}.$$

Here $\tilde{w}_r$ denote the equivalent weights from Section 5.1, and the above identity follows from (5.1.2) *and* (5.1.3)*. The center of mass of a_r^* will play a critical role both in the continuum limit of Algorithm 1 (where it is the transport direction of the resulting transport PDE) and in the connection to random walks*



(a) Illustration of $\bar{b}_r^-$ for $r = 3$.

(b) Illustration of $\bar{b}_r^0$ for $r = 3$.

Figure 6.1: **Visualization $\bar{b}_r^-$ and $\bar{b}_r^0$:** Here we illustrate the sets $\bar{b}_r^- := D(b_r^-) \cap \{y \le -1\}$ and $\bar{b}_r^0 := D(b_r^0) \cap \{y \le 0\}$ in the case $r = 3$. These sets are defined explicitly in (6.1.4) and (6.1.5), and D is the dilation operator defined in the notation section. These sets are important because depending on whether we use the direct form of Algorithm 1 or its semi-implicit extension, $\mathrm{Supp}(a_r^*)$ is always contained in one or the other. See (6.1.7) in the text.

(where, after multiplication by h, it is the mean of the increments of the walk).

Under these assumptions, the matrix $\mathcal{L}$ from (5.2.1) is independent of k (that is, we solve the same linear system for every shell), and moreover $\mathcal{L}$ becomes a Toeplitz matrix (Dirichlet boundary conditions) or circulant matrix (periodic boundary conditions). For a given pixel $\mathbf{x}$ at least $r+2$ pixels away from the boundary at $x = 0$ and $x = 1$, that is $\mathbf{x} \in \{(jh, kh) : r+2 \le j \le N-r-2\}$, it takes on the form

$$(\mathcal{L}\mathbf{u})(\mathbf{x}) = \left(1 - \frac{\tilde{w}_r(\mathbf{0},\mathbf{0})}{W}\right) u_h(\mathbf{x}) - \sum_{\mathbf{y} \in s_r \setminus \{\mathbf{0}\}} \frac{\tilde{w}_r(\mathbf{0},\mathbf{y})}{W} u_h(\mathbf{x} + h\mathbf{y}), \tag{6.1.8}$$

where by (6.1.7) we have

$$s_r = \mathrm{Supp}(a_r^*) \cap (\mathbb{Z} \times \{0\}) \subseteq \{-(r+2)e_1, -(r+1)e_1, \dots, (r+1)e_1, (r+2)e_1\}.$$

If $\mathbf{x}$ is *not* at least $r+2$ pixels away from the boundaries, then the formula changes in the usual way for Toeplitz and circulant matrices - we assume the reader is familiar with this and no further discussion is needed. Under the same assumptions the vector $\mathbf{f}$ becomes

$$\mathbf{f} = \sum_{\mathbf{y} \in \mathrm{Supp}(a_r^*) \setminus (\mathbb{Z} \times \{0\})} \frac{\tilde{w}_r(\mathbf{0},\mathbf{y})}{W} u_h(\mathbf{x} + h\mathbf{y})$$

where $\mathrm{Supp}(a_r^*) \setminus (\mathbb{Z} \times \{\mathbf{0}\}) \subseteq b_{r+2}^-$. We also define

$$\tilde{W} := \sum_{j=-r-2}^{r+2} \tilde{w}_r(\mathbf{0}, je_1) \quad \text{and} \quad \tilde{w}_{0,0} := \tilde{w}_r(\mathbf{0},\mathbf{0}). \tag{6.1.9}$$

For a given point $\mathbf{x} \in \partial D_h^{(k)}$, (again, assuming $\mathbf{x}$ is far enough from the boundary) the ratio $\frac{\tilde{W}}{W}$ gives the fraction of the mass of the stencil (Definition 6.1.2) centered at $\mathbf{x}$ that gets "leaked" to the unknown pixels in $\partial D_h^{(k)}$, while $\frac{\tilde{w}_{0,0}}{W}$ gives the fraction that gets leaked to $\mathbf{x}$. Together, these give a measure of the diagonal dominance of $\mathcal{L}$, as we have

$$\frac{\sum_{j \neq i} |\mathcal{L}_{ij}|}{|\mathcal{L}_{ii}|} = \frac{\tilde{W} - \tilde{w}_{0,0}}{W - \tilde{w}_{0,0}}.$$

The smaller this ratio is, the stronger the diagonal dominance of $\mathcal{L}$, and the faster damped Jacobi and SOR can be expected to converge - see Proposition 6.2.1 for explicit formulas.

For semi-implicit Guidefill with guidance direction $\mathbf{g} = (\cos\theta, \sin\theta)$, it can be shown that $\mathcal{L}$ becomes a lower triangular matrix in the limit $\mu \to \infty$, provided we order unknowns left to right if $\cos\theta > 0$ and right to left otherwise (see the proof of Proposition 6.2.1). This gives us a hint that Gauss-Seidel and SOR might be very effective for the solution of (5.2.1) in this case, and indeed Proposition 6.2.1 confirms this.

6.2 Convergence rates of damped Jacobi and SOR for semi-implicit Guidefill

Here we derive tight bounds on the convergence rates of damped Jacobi and SOR for solving (5.2.1) in the semi-implicit extension of Guidefill described in Section 5.2, under the symmetry assumptions discussed above, and in the limit $\mu \to \infty$ (recall that μ is the parameter from the weights (2.5.2) controlling the extent to which weights are biased in favor of pixels in the directions $\pm\mathbf{g}$). We will prove that in each case the parameter value $\omega = 1$ is optimal, but also pay special attention to the case $\omega = \omega^*$ given by (5.2.6), since this is the value of ω that our proposed implementation of the semi-implicit extension in Algorithm 1 uses. We consider general ω mainly in order to demonstrate that the choice $\omega = \omega^*$, while not optimal, is close enough to optimal not to matter in practice.

We will assume that $D = (0,1]^2$ with Dirichlet boundary conditions, as this simplifies our analysis of SOR - for damped Jacobi, we could just as easily have assumed periodic boundary conditions. We will measure convergence rates with respect to the induced infinity norm, which obeys the identity

$$\|A\|_\infty = \max_{i=1}^{N} \sum_{j=1}^{N} |a_{ij}| \tag{6.2.1}$$

for any $N \times N$ matrix A. Note that the iterates of the error $\mathbf{e}^{(0)}$, $\mathbf{e}^{(1)}$, ... associated with any stationary iterative method with iteration matrix M obey the bounds

$$\|\mathbf{e}^{(n)}\| \leq \|M\|^n \|\mathbf{e}^{(0)}\| \quad \text{and} \quad R(\mathbf{e}) := \sqrt[n]{\frac{\|\mathbf{e}^{(n)}\|}{\|\mathbf{e}^{(0)}\|}} \leq \|M\| \tag{6.2.2}$$

for *any* vector norm $\|\cdot\|$ and induced matrix norm. We will be interested in these identities in the particular case that the vector norm is $\|\cdot\|_\infty$, and the stationary iterative method is damped Jacobi or SOR. Here

$$\mathbf{e}^{(n)} := u_h - u_h^{(n)}$$

denotes the difference between the exact solution to (5.2.1), found by first solving (5.2.1) to machine precision, with the approximate solution at iteration n.

Proposition 6.2.1. *Suppose semi-implicit Guidefill with guidance direction* $\mathbf{g} = (\cos\theta, \sin\theta)$ *is used to inpaint the square* $[0,1)^2$ *under the assumptions above, using either damped Jacobi or SOR to solve* (5.2.1). *Suppose that in the case of SOR,* $\partial D_h^{(k)} = \{kh\}_{k\in\mathbb{Z}}$ *is ordered from left to right if* $\cos\theta \geq 0$ *and from right to left otherwise. Let* $\mathcal{L}$ *be as in* (5.2.2) *and define* $\mathcal{L} = D - L - U$*, where* D*,* $-L$*, and* $-U$ *are the diagonal, strictly lower triangular, and strictly upper triangular parts of* $\mathcal{L}$ *respectively. Let* J_ω *and* G_ω *denote the iteration matrices of damped Jacobi and SOR respectively with relaxation parameter* ω*, that is*

$$J_\omega = I - \omega D^{-1}\mathcal{L} \qquad G_\omega = (I - \omega D^{-1}L)^{-1}((1-\omega)I + D^{-1}U).$$

Let $r = \epsilon/h$ *and define* $\theta_c = \arcsin(1/r)$*. Define, for* $\theta_c \leq \theta \leq \pi - \theta_c$*,* $j^* = \lfloor \frac{1}{\sin\theta} \rfloor \leq r$*. Let* W *be the total weight* (5.2.4)*, let* $\tilde{W}$ *and* $\tilde{w}_{0,0}$ *be as in* (6.1.9)*. Then, in the limit as* $\mu \to \infty$*, we have*

$$\begin{aligned} W &= \sum_{j=1}^{r} \frac{1}{j}, \qquad \tilde{w}_{0,0} = (1-\sin\theta)(1-|\cos\theta|) \\ \tilde{W} &= \begin{cases} \sum_{j=1}^{r} \frac{1}{j} - r\sin\theta & \text{if } \theta \in (0,\theta_c] \cup [\pi-\theta_c, \pi) \\ \sum_{j=1}^{j^*} \frac{1}{j} - j^*\sin\theta & \text{if } \theta \in (\theta_c, \pi-\theta_c) \end{cases}. \end{aligned}$$

and

$$\begin{aligned} \|J_\omega\|_\infty &= |1-\omega| + \omega\left(\frac{\tilde{W} - \tilde{w}_{0,0}}{W - \tilde{w}_{0,0}}\right) \quad \text{for } \omega \in (0,2) \\ \|G_\omega\|_\infty &= \frac{|1-\omega|}{1 - \omega\frac{\tilde{W}-\tilde{w}_{0,0}}{W-\tilde{w}_{0,0}}} \quad \text{for } \omega \in (0,1], \end{aligned}$$

where $\|\cdot\|_\infty$ *is the induced infinity matrix norm* (6.2.1)*. The optimal* $\omega \in (0,2)$ *is in both cases independent of* θ *and equal to* $\omega = 1$*, where we obtain*

$$\begin{aligned} \|J_1\|_\infty &= \begin{cases} 1 - \frac{r\sin\theta}{\sum_{j=1}^{r}\frac{1}{j} - (1-\sin\theta)(1-|\cos\theta|)} & \text{if } \theta \in (0,\theta_c] \cup [\pi-\theta_c,\pi) \\ 1 - \frac{\sum_{j=j^*+1}^{r}\frac{1}{j} + j^*\sin\theta}{\sum_{j=1}^{r}\frac{1}{j} - (1-\sin\theta)(1-|\cos\theta|)} & \text{if } \theta \in (\theta_c, \pi-\theta_c). \end{cases} \\ \|G_1\|_\infty &= 0. \end{aligned}$$

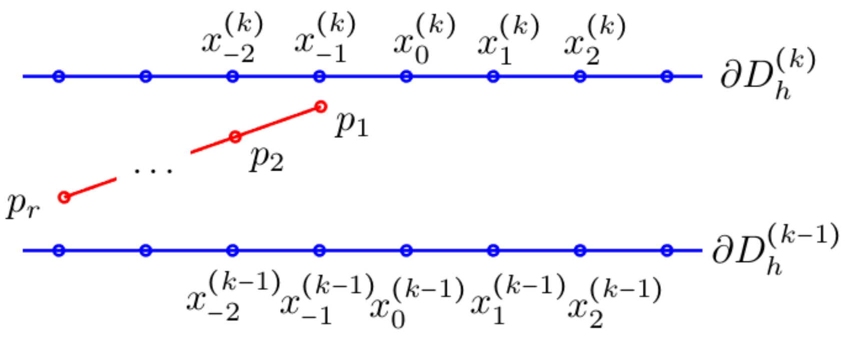
Proof. First we fix some notation. Suppose we are on iteration k of semi-implicit Guidefill and let $\mathbf{x} := x_0^{(k)}$ denote a fixed but arbitrary member of $\partial D_h^{(k)}$. The pixel $x_0^{(k)}$ is coupled by (5.2.1) to its immediate neighbors $x_j^{(k)}$ for $-r-2 \leq j \leq r+2$, and also depends on the pixels $x_j^{(k-\delta)} := \mathbf{x} + h(j,\delta) \in \partial D_h^{(k-\delta)}$ for $(j,\delta) \in b_{r+2}^-$ which appear in the right hand side of (5.2.1) within the vector $\mathbf{f}$.

Next, note that since $\mu \to \infty$, the weights w_r have vanished on all of $\tilde{b}_r^0$ except for the line of ghost pixels

$$\ell_r^- := \{-j\mathbf{g}\}_{j=1}^{r}.$$

For convenience, we enumerate ℓ_r^- as $\ell_r^- := \{p_j\}_{j=1}^{r}$ where $p_j = -j\mathbf{g}$. Each p_j receives weight

$$w_j := \frac{1}{j}.$$



$x_{-2}^{(k)}$ $x_{-1}^{(k)}$ $x_0^{(k)}$ $x_1^{(k)}$ $x_2^{(k)}$
$\partial D_h^{(k)}$
p_1
p_2
$\ldots$
p_r
$\partial D_h^{(k-1)}$
$x_{-2}^{(k-1)}$ $x_{-1}^{(k-1)}$ $x_0^{(k-1)}$ $x_1^{(k-1)}$ $x_2^{(k-1)}$

Figure 6.2: **Illustration of the position of the line ℓ_r^- relative to the current shell $\partial D_h^{(k)}$ and previous shell $\partial D_h^{(k-1)}$:** Here we visualize the line $\ell_r^- := \{-j\mathbf{g}\}_{j=1}^r \subseteq \tilde{b}_r^-$ when $\mathbf{g} = (\cos\theta, \sin\theta)$ with $0 < \theta < \theta_c = \arcsin(1/r)$. In this case ℓ_r^- fits entirely into the space between $\partial D_h^{(k)}$ and $\partial D_h^{(k-1)}$. For convenience, we enumerate ℓ_r^- as $\ell_r^- := \{p_j\}_{j=1}^r$ where $p_j = -j\mathbf{g}$. We write the current pixel of interest $\mathbf{x}$ as $x_0^{(k)}$ for convenience, and its unknown neighbors in $\partial D_h^{(k)}$ as $x_j^{(k)}$ for $-r-2 \le j \le r+2$, while its already filled neighbors (we only show the ones in $\partial D_h^{(k-1)}$) are denoted by $x_j^{(k-\delta)} := \mathbf{x} + h(j,\delta) \in \partial D_h^{(k-\delta)}$ for $(j,\delta) \in b_{r+2}^-$.

This situation is illustrated in Figure 6.2 for the case $0 < \theta < \theta_c$, where ℓ_r^- fits entirely into the space between $\partial D_h^{(k)}$ and $\partial D_h^{(k-1)}$.

To compute the entries of $\mathcal{L}$, we follow the idea of Section 5.1 and consider how the weight w_j of each ghost pixel p_j gets distributed to its real pixel neighbors. For example, in Figure 6.2, the weight w_1 of p_1 gets redistributed amongst the four pixels $x_0^{(k)}$, $x_{-1}^{(k)}$, $x_0^{(k-1)}$, and $x_{-1}^{(k-1)}$.

How exactly this weight gets redistributed is for the most part not something we need to know precisely. For example, it is already clear from Figure 6.2 that if $0 < \theta \le \frac{\pi}{2}$, then none of the weight of any of the p_j make it into any of $x_1^{(k)}, x_2^{(k)}, x_3^{(k)} \ldots$. Similarly, if $\frac{\pi}{2} \le \theta < \pi$, no weight makes it to any of $x_{-1}^{(k)}, x_{-2}^{(k)}, x_{-3}^{(k)} \ldots$. This means that, given our assumed ordering of pixels within each layer $\partial D_h^{(k)}$, we already know that $\mathcal{L}$ is a lower triangular matrix. Hence $L = \mathcal{L}$, $U = O$. Therefore, G_ω takes on the simplified form

$$G_\omega = (1-\omega)(I - \omega D^{-1}L)^{-1}.$$

We begin with $\|G_\omega\|_\infty$, the harder case. In this case, defining

$$A := I - \omega D^{-1}L,$$

we have

$$\|G_\omega\|_\infty = |1-\omega|\|A^{-1}\|_\infty \tag{6.2.3}$$

We know $\mathcal{L} = D - L$ is strictly diagonally dominant, so the following computation shows that A is as well, provided $0 < \omega \le 1$:

$$\sum_{j\neq i} |A_{ij}| = \frac{|\omega|}{|D_{ii}|}\sum_{j\neq i}|L_{ij}| < |\omega| \le 1 = |A_{ii}|.$$

Hence, the following classical bound due to Jim Varah [66, Theorem 1] applies:

$$\|A^{-1}\|_\infty \le \frac{1}{\min_{i=1}^N \Delta_i(A)} \qquad \text{where} \qquad \Delta_i(A) := \left||A_{ii}| - \sum_{j\neq i}|A_{ij}|\right|.$$

Since A is a Toeplitz matrix with band width $r+2$ and at the same time a lower triangular matrix , we know that $\Delta_i(A)$ is the same for all $i \ge r+3$, but increases somewhat for $i \le r+2$ as there are fewer off diagonal terms (due to our assumed Dirichlet boundary conditions). In particular, the first row has *no* off diagonal terms, so we have $\Delta_1(A) = A_{11} = 1$. It follows that

$$\Delta_1(A) \geq \Delta_2(A) \geq \ldots \geq \Delta_{r+3}(A) = \Delta_{r+4}(A) = \ldots = \Delta_N(A).$$

Choosing row N as a representative row for convenience gives

$$\|A^{-1}\| \leq \frac{1}{\Delta_N(A)}.$$

However, the identity

$$\|A^{-1}\|^{-1} = \inf_{\mathbf{x}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$$

(valid for any induced norm) means that in particular, for the vector e containing $r+2$ zeros followed by $N-r-2$ ones, that is

$$e = (\underbrace{0,\ldots,0}_{r+2},\underbrace{1,\ldots,1}_{N-r-2}),$$

we have

$$\|A^{-1}\|_\infty^{-1} \leq \frac{\|Ae\|_\infty}{\|e\|_\infty} = \max_{i=1}^{N}\left|\sum_{j=1}^{N} A_{ij}e_j\right| = \max_{i=r+3}^{N}\left||A_{ii}| - \sum_{j\neq i}|A_{ij}|\right| = \Delta_N(A).$$

where we have used the fact that for all i we have $A_{ii} > 0$ and $A_{ij} \leq 0$ for $j \neq i$. The vector e was chosen deliberately in order to avoid the first $r+2$ rows of A, which we have already said are different due to boundary conditions. Hence

$$\|A^{-1}\|_\infty \geq \frac{1}{\Delta_N(A)}$$

as well, and having proven the bound in both directions we conclude

$$\|A^{-1}\|_\infty = \frac{1}{\Delta_N(A)}. \tag{6.2.4}$$

Remark 6.2.2. *It appears that Varah's bound [66, Theorem 1] should generalize to equality not only in our case, but to general strictly diagonally dominant Toeplitz matrices obeying $A_{ii} > 0$ for all i and $A_{ij} \leq 0$ whenever $j \neq i$, using a very similar argument. However, this generalization does not appear in [66] and we have been unable to find it in the literature.*

The next step is to compute $\Delta_N(A)$. To that end, note that by definition $A = I - \omega D^{-1}L$ obeys $A_{ii} = 1$ for all i and

$$A_{ij} = -\omega\frac{\left(\frac{\tilde{w}_r(\mathbf{0},(j-i)\mathbf{e}_1)}{W}\right)}{\left(1-\frac{\tilde{w}_r(\mathbf{0},\mathbf{0})}{W}\right)} = -\omega\frac{\tilde{w}_r(\mathbf{0},(j-i)\mathbf{e}_1)}{W-\tilde{w}_r(\mathbf{0},\mathbf{0})} \quad \text{for} \quad \max(i-r-2,1) \leq j < i.$$

by (6.1.8). Here W are the total weight and equivalent weights $\tilde{w}_r$ defined in Section 6.1. Hence

$$\Delta_N(A) = \left|1 - \omega\frac{\sum_{j=-r-2}^{-1}\tilde{w}_r(\mathbf{0},j\mathbf{e}_1)}{W-\tilde{w}_r(\mathbf{0},\mathbf{0})}\right| = \left|1-\omega\left(\frac{\tilde{W}-\tilde{w}_{0,0}}{W-\tilde{w}_{0,0}}\right)\right|,$$

where $\tilde{W}$ and $\tilde{w}_{0,0}$ are defined as in (6.1.9). Combining the above with (6.2.3) and (6.2.4) we finally obtain

$$\|G_\omega\|_\infty = \frac{|1-\omega|}{1-\omega\left(\frac{\tilde{W}-\tilde{w}_{0,0}}{W-\tilde{w}_{0,0}}\right)}$$

as claimed. We leave deriving expressions for W, $\tilde{W}$, and $\tilde{w}_{0,0}$ until the end. First we derive an expression for $\|J_\omega\|_\infty$ in terms of these three quantities. Since $U = O$ we have

$$J_\omega = I - \omega D^{-1}\mathcal{L} = I - \omega D^{-1}(D-L) = (1-\omega)I + \omega D^{-1}L.$$

By definition we have

$$\|J_\omega\|_\infty := \max_{i=1}^{N} \sum_{j=1}^{N} |(J_\omega)_{ij}|.$$

So long as $i \geq r+3$, this sum becomes

$$\sum_{j=1}^{N} |(J_\omega)_{ij}| = |1-\omega| + \omega\left(\frac{\tilde{W}-\tilde{w}_{0,0}}{W-\tilde{w}_{0,0}}\right).$$

If $i \leq r+2$, then this sum includes fewer terms and is potentially smaller. Hence

$$\|J_\omega\|_\infty = |1-\omega| + \omega\left(\frac{\tilde{W}-\tilde{w}_{0,0}}{W-\tilde{w}_{0,0}}\right).$$

Our remaining task is to derive the claimed expressions for W, $\tilde{W}$, and $\tilde{w}_{0,0}$. The easiest is W. By (5.2.4) we have

$$W = \sum_{j=1}^{r} w_j = \sum_{j=1}^{r} \frac{1}{j}.$$

It is also not difficult to compute $\tilde{w}_{0,0}$, which represents fraction of the mass $w_1 = 1$ of the point p_1 that gets redistributed back to $x_0^{(k)}$ (see Figure 6.2). Since p_1 sits $h\sin\theta$ units below $\partial D_h^{(k)}$ and $h(1-\sin\theta)$ units above $\partial D_h^{(k-1)}$, and either $h\cos\theta$ units to the left $x_0^{(k)}$ and $h(1-\cos\theta)$ units right of $x_{-1}^{(k)}$ if $0 \leq \theta \leq \frac{\pi}{2}$ or $h|\cos\theta|$ units right of $x_0^{(k)}$ and $h(1-|\cos\theta|)$ units left of $x_1^{(k)}$ otherwise, it follows that

$$\tilde{w}_{0,0} = (1-\sin\theta)(1-|\cos\theta|)w_1 = (1-\sin\theta)(1-|\cos\theta|).$$

For $\tilde{W}$, we split into cases. If $0 \leq \theta \leq \theta_c$ or $\pi - \theta_c \leq \theta \leq \pi$, then ℓ_r^- fits entirely between $\partial D_h^{(k)}$ and $\partial D_h^{(k-1)}$, as in Figure 6.2. If $\theta_c < \theta < \pi - \theta_c$, then only p_1 up to p_{j^*} fit (recall the definition of j^* from the statement of the proposition). As a result, in the first case every p_j for $1 \leq j \leq r$ contribute mass to $\tilde{W}$. In the second case, only the first j^* contribute. Each contributing p_j is situated $hj\sin\theta$ units below $\partial D_h^{(k)}$ and $h(1-j\sin\theta)$ units above $\partial D_h^{(k-1)}$. Hence each contributing p_j contributes $(1-j\sin\theta)w_j$ towards $\tilde{W}$. Hence, in the first case we have

$$\tilde{W} = \sum_{j=1}^{r}(1-j\sin\theta)\frac{1}{j} = \sum_{j=1}^{r}\frac{1}{j} - r\sin\theta,$$

while in the second we have

$$\tilde{W} = \sum_{j=1}^{j^*}\frac{1}{j} - j^*\sin\theta.$$

Our final claim on the expressions for $\|J_1\|_\infty$ and $\|G_1\|_\infty$ and the optimality of $\omega = 1$ is now a simple exercise and is left to the reader.

□

Corollary 6.2.3. *For the special case of*

$$\omega^* = \left(1 - \frac{\tilde{w}_{0,0}}{W}\right) = \left(1 - \frac{(1-\sin\theta)(1-|\cos\theta|)}{\sum_{j=1}^{r}\frac{1}{j}}\right)$$

that is, the parameter value equivalent to running Algorithm 1 with "semiImplicit" set to true, we obtain

$$\begin{aligned} \|G_{\omega^*}\|_\infty &= \frac{\tilde{w}_{0,0}}{W - \tilde{W}_k + \tilde{w}_{0,0}} \\ &= \begin{cases} \frac{(1-\sin\theta)(1-|\cos\theta|)}{r\sin\theta + (1-\sin\theta)(1-|\cos\theta|)} & \textit{if } \theta \in (0,\theta_c] \cup [\pi-\theta_c,\pi) \\ \frac{(1-\sin\theta)(1-|\cos\theta|)}{\sum_{j=j^*+1}^{r}\frac{1}{j} + j^*\sin\theta + (1-\sin\theta)(1-|\cos\theta|)} & \textit{if } \theta \in (\theta_c, \pi-\theta_c). \end{cases} \\ \|J_{\omega^*}\|_\infty &= \frac{\tilde{W}_k}{W} = \begin{cases} 1 - \frac{r\sin\theta}{\sum_{j=1}^{r}\frac{1}{j}} & \textit{if } \theta \in (0,\theta_c] \cup [\pi-\theta_c,\pi) \\ 1 - \frac{\sum_{j=j^*+1}^{r}\frac{1}{j} + j^*\sin\theta}{\sum_{j=1}^{r}\frac{1}{j}} & \textit{if } \theta \in (\theta_c, \pi-\theta_c). \end{cases} \end{aligned}$$

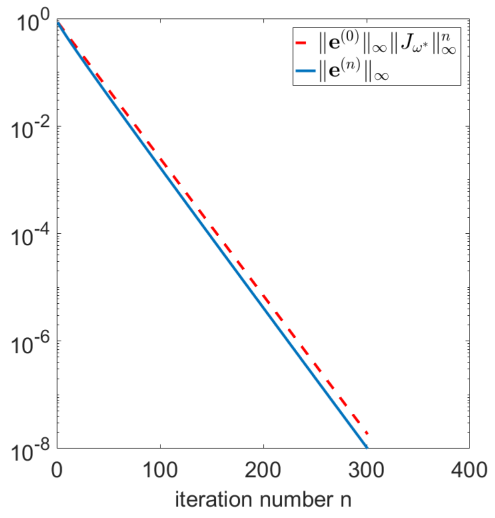
See Figure 6.3 for a plot of $\|G_{\omega^*}\|_\infty$ and $\|J_{\omega^*}\|_\infty$ as a function of θ for $r = 3$.

Proof. This follows from direct substitution of ω^* given by (5.2.6) into Proposition 6.2.1. □

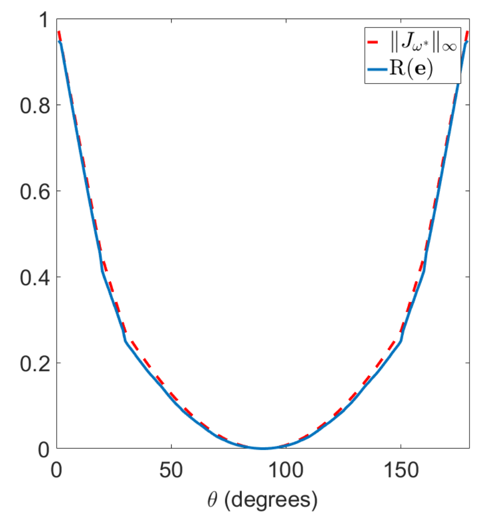
Remark 6.2.4. *Although our choice of ω^* is non-optimal, it is convenient to implement and the difference in performance is negligible. For example, even for the optimal value $\omega = 1$, for Jacobi we have $\|J_1\|_\infty \to 1$ as $\theta \to 0$ or $\theta \to \pi$. At the same time, for SOR we have $\|G_{\omega^*}\|_\infty \leq 0.06$ (Figure 6.3) for all θ and all $r \geq 0$ (it follows trivially from Corollary 6.2.3 that $\|G_{\omega^*}\|_\infty$ is decreasing function of r for each fixed θ) - while not quite as good as the optimal value $\|G_1\|_\infty \equiv 0$, it is more than satisfactory and moreover, we have $\|G_{\omega^*}\|_\infty \to 0$ as $\theta \to 0$ or $\theta \to \pi$. As we will see in Section 6.6.1, it is precisely these shallow angles where the semi-implicit extension has an advantage over the direct method. Unfortunately, however, Guidefill was designed to be a parallel algorithm but SOR is a sequential. While ideally we would like a method to solve the linear system* (5.2.1) *that is both fast and parallel, this is beyond the scope of the present work.*

6.3 Convergence of Algorithm 1 to a continuum limit

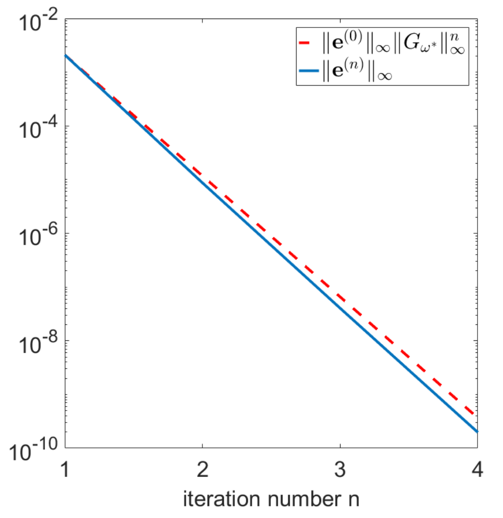
Our objective in this section is to prove that the direct and semi-implicit forms of Algorithm 1 both converge to a continuum limit u when we take $(h, \epsilon) \to (0,0)$ along the path $\epsilon = rh$. Before we can do that, we need to define what that limit is, in what sense u_h converges to it, and to lay out our assumptions regarding the regularity of u_0. We also describe a property which, if satisfied, will allow us to connect our limit with the one studied by Bornemann and März in [12] (we also show that all methods considered in this thesis have this property). When u_0 is smooth, convergence is straightforward to prove - we did so in [37, Theorem 1] for the direct form of Algorithm 1 using a simple argument based on Taylor series. However, in this thesis we are interested in a more general setting where u_0 may not be smooth and Taylor series may not be available. Instead of Taylor series, our main tool will be a connection to stopped random walks. We will explain this connection briefly before continuing on to the main result of this section. Throughout this section we assume for convenience that u_0 and u_h are both greyscale images, that is $u_0 : \Omega\backslash D \to \mathbb{R}$, $u_h : \Omega_h \to \mathbb{R}$. In the color case $u_0 : \Omega\backslash D \to \mathbb{R}^d$, $u_h : \Omega_h \to \mathbb{R}^d$ one may easily show that our results hold channel-wise.



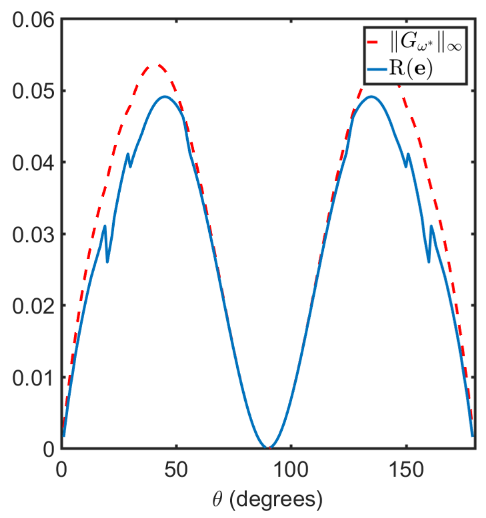
(a)



(b)



(c)



(d)

Figure 6.3: **Convergence rates of damped Jacobi and SOR for semi-implicit Guidefill:** As noted in Section 5.2, the implementation of semi-implicit Guidefill outlined in Algorithm 1 (blue text) is equivalent to solving the linear system (5.2.1) iteratively using damped Jacobi (parallel implementation) or SOR (sequential implementation), with relaxation parameter ω^* given by (5.2.6). Here we compare the experimentally measured convergence rates of this implementation ($r = 3$, $\mathbf{g} = (\cos\theta, \sin\theta)$ and $\mu = 100$) with the theoretical bounds on $\|J_{\omega^*}\|_\infty$ and $\|G_{\omega^*}\|_\infty$ from Corollary 6.2.3. Specifically, (a) and (c) confirm experimentally the first bound in (6.2.2) in the cases $M = J_{\omega^*}$ and $M = G_{\omega^*}$, that is, damped Jacobi and SOR with relaxation parameter ω^*, for the case $\theta = 2°$. The inpainting problem in this case is the same as in Figure 5.2(a), and all the parameters of semi-implicit Guidefill are the same. The "exact" solution u_h was found by first solving (5.2.1) to machine precision. In each case, we measured convergence rates only within the first "shell" of the inpainting problem. Next, (b)-(d) confirm experimentally the second bound in (6.2.2), as a function of θ. The inpainting problem is the same as the one in Figure 2.6(a), and all parameters are the same. In this case we vary θ from 1° up to 179° in increments of one degree, in each case iteratively solving (5.2.1) (again, only for the first shell), computing $R(\mathbf{e})$, and comparing with $\|J_{\omega^*}\|_\infty$ and $\|G_{\omega^*}\|_\infty$. Note the excellent performance of SOR in comparison with damped Jacobi.

Definition of the continuum limit. We wish to prove convergence of the direct and semi-implicit forms of Algorithm 1 to a continuum limit u given by the transport equation

$$\nabla u \cdot \mathbf{g}_r^* = 0, \qquad u\Big|_{y=0} = u_0\Big|_{y=0} \qquad u\Big|_{x=0} = u\Big|_{x=1} \tag{6.3.1}$$

Because of the assumptions we have made in Section 6.1, $\mathbf{g}_r^*$ will turn out to be a constant equal to the center of mass of the stencil a_r^* with respect to the stencil weights $\{w_r(\mathbf{0},\mathbf{y}) : \mathbf{y} \in a_r^*\}$ (Definition 6.1.2), that is

$$\mathbf{g}_r^* = \frac{\sum_{\mathbf{y}\in a_r^*} w_r(0,\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y}\in a_r^*} w_r(0,\mathbf{y})}. \tag{6.3.2}$$

As we will allow discontinuous boundary data u_0, the solution to (6.3.1) must be defined in a weak sense. However, since $\mathbf{g}_r^*$ is a constant, this is simple. So long as $\mathbf{g}_r^* \cdot e_2 \neq 0$, we simply define the solution to the transport problem (6.3.1) to be

$$u(\mathbf{x}) = u_0(\Pi_{\theta_r^*}(\mathbf{x})), \qquad \text{where} \qquad \Pi_{\theta_r^*}(x,y) = (x - \cot(\theta_r^*)y \bmod 1, 0). \tag{6.3.3}$$

We call $\Pi_{\theta_r^*} : D \to \partial D$ the transport operator associated with (6.3.3). The mod 1 is due to our assumed periodic boundary conditions and

$$\theta_r^* = \theta(\mathbf{g}_r^*) \in (0,\pi)$$

is the counterclockwise angle between the x-axis and the line $L_{\mathbf{g}_r^*} := \{\lambda \mathbf{g}_r^* : \lambda \in \mathbb{R}\}$.

Modes of convergence (discrete L^p norms). Given $f_h : D_h \to \mathbb{R}$, we introduce the discrete L^∞ norm

$$\|f_h\|_\infty := \max_{\mathbf{x}\in D_h} |f_h(\mathbf{x})| \tag{6.3.4}$$

and the discrete L^p norm

$$\|f_h\|_p := \Big(\sum_{\mathbf{x}\in D_h} |f_h(\mathbf{x})|^p h^2\Big)^{\frac{1}{p}}. \tag{6.3.5}$$

These norms will be used to measure the convergence of u_h to the continuum limit u.

Convergence of center of mass. There is one additional property which, if satisfied, will enable us to connect our limit with the one studied by Bornemann and März. Specifically, if a method obeys

$$\lim_{r\to\infty} \frac{\mathbf{g}_r^*}{r} \to \mathbf{g}^* \in \mathbb{R}^2 \qquad \text{with rate at least } O(r^{-q}) \quad \text{for some } q > 0, \tag{6.3.6}$$

then it is possible to make sense of a (generalized version) of Bornemann and März's limit. Moreover, as we will see in Section 6.5, the vector $\mathbf{g}^*$ *is* the transport direction obtained in their limit. When $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$ as in Telea's algorithm and coherence transport, or $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ as in Guidefill, it is easy to see that this condition is satisfied. In fact, if $a_r^* \in \{b_r^-, b_r^0, \tilde{b}_r^-, \tilde{b}_r^0\}$ we have

$$\lim_{r\to\infty} \frac{\mathbf{g}_r^*}{r} = \lim_{r\to\infty} \frac{\sum_{\mathbf{y}\in a_r^*} \hat{w}\left(0, \frac{\mathbf{y}}{r}\right) \frac{\mathbf{y}}{r}\frac{1}{r}}{\sum_{\mathbf{y}\in a_r^*} \hat{w}\left(0, \frac{\mathbf{y}}{r}\right) \frac{1}{r}} \to \frac{\int_{\mathbf{y}\in B_1^-(\mathbf{0})} \hat{w}(0,\mathbf{y})\mathbf{y}d\mathbf{y}}{\int_{\mathbf{y}\in B_1^-(\mathbf{0})} \hat{w}(0,\mathbf{y})d\mathbf{y}}$$

(where $\hat{w}$ is the function we assumed existed in (2.0.1)), because the LHS can be interpreted as a Riemann sum for the RHS. We also know, by an elementary argument in quadrature, that the convergence rate is $O(r^{-1})$ or better, so $q = 1$ in this case. In fact, it is straightforward to show that the integral on the RHS of the above equation is the same for coherence transport, Guidefill, and semi-implicit Guidefill. In other words, the original limit of Bornemann and März is the same for all three of these methods. However,

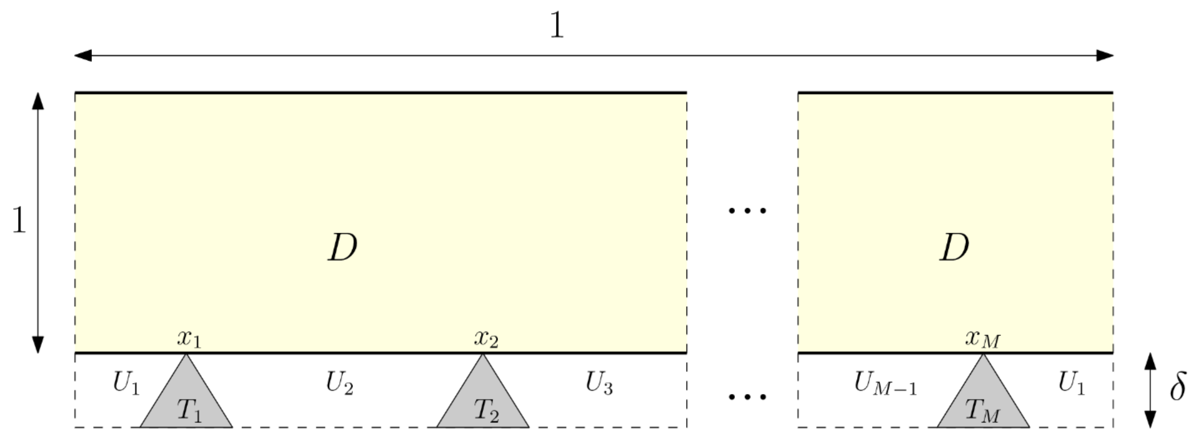


Figure 6.4: **Regularity assumptions:** The inpainting domain $D = (0,1]^2$ together with data region $\mathcal{U} = (0,1] \times (-\delta, 0]$. The image data u_0 has high regularity when restricted to each of the sets U_i but has (possibly) lower regularity on $\mathcal{U}$ as a whole due to problems within finitely many triangles T_i (where we might have, for example, jump discontinuities in u_0 or its derivatives, if they exist). In particular, we have $u_0 \in W^{s,\infty}(U_i)$ for all i and for some $s > 0$ but only $u_0 \in W^{s',\infty}(\mathcal{U})$ for some $0 \leq s' \leq s$, where $W^{s,\infty}(U_i)$, $W^{s',\infty}(\mathcal{U})$ denote the fractional sobolev spaces described in the text.

our limit predicts that they behave very differently - see Section 6.6.2.

Regularity of the boundary data

Our goal is to prove convergence under very weak regularity conditions on u_0. In particular, our analysis should include the case that u_0 is piecewise smooth, as this case is particularly relevant to images. Indeed we actually develop an even more general setting that includes piecewise smooth u_0 as a special case.

Specifically, we consider u_0 that belongs to the fractional Sobolev space $W^{s,\infty}(U_i)$ where $0 < s < \infty$, when restricted to each of a series of subsets $\{U_i\}_{i=1}^N \subseteq \mathcal{U}$ such that $\mathcal{U} \backslash \{U_i\}_{i=1}^N$ is "small" in some sense, while belonging to a potentially lower regularity fractional Sobolev space $W^{s',\infty}(\mathcal{U})$ for some $0 \leq s' \leq s$ when considered on $\mathcal{U}$ as a whole. Note that for $s' > 0$ we have the identity $W^{s',\infty}(\mathcal{U}) = C^{s', s' - \lfloor s' \rfloor}(\mathcal{U})$ - see also [51] for a review of fractional Sobolev spaces. While most authors avoid defining $W^{s',\infty}(\mathcal{U})$ in the case $s' = 0$, for the purposes of this thesis we define

$$W^{0,\infty}(\mathcal{U}) = L^\infty(\mathcal{U}). \tag{6.3.7}$$

This will enable us to seamlessly incorporate the case of u_0 that is piecewise continuous, which is of particular interest. With this definition for every $s \geq 0$ the space $W^{s,\infty}(\mathcal{U})$ is a Banach space with norm $\| \cdot \|_{W^{s,\infty}(\mathcal{U})}$ such that any $u_0 \in W^{s,\infty}(\mathcal{U})$ obeys the Hölder property

$$\|\partial_\alpha u_0(\mathbf{x}) - \partial_\alpha u_0(\mathbf{y})\| \leq \|u_0\|_{W^{s,\infty}(\mathcal{U})} \|\mathbf{x} - \mathbf{y}\|^{s - \lfloor s \rfloor} \tag{6.3.8}$$

for all multi-indices α s.t. $|\alpha| = \lfloor s \rfloor$. This property will be one of the key tools of our analysis in this section.

Detailed Assumptions. We assume that the strip $\mathcal{U} = (0,1] \times (-\delta, 0]$ contains M closed triangles $\{T_i\}_{i=1}^M$ with tips touching the x-axis at M points $\mathbf{x}_i := (x_i, 0)$ with $1 \leq i \leq M$ as in Figure 6.4. Each T_i is defined by the inequality

$$T_i = \{(x,y) \in (0,1] \times [-\delta, 0] : y \leq -L|x_i - x|\}$$

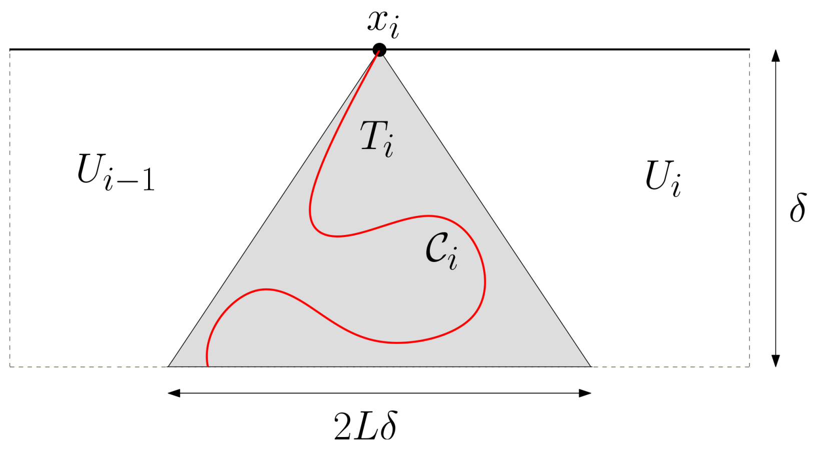


Figure 6.5: **Close up of one of the triangles T_i:** The source of the problems in T_i may be, for example, a curve $\mathcal{C}_i$ such that the regularity condition (6.3.8) with exponent s fails if $\mathbf{x}$ and $\mathbf{y}$ lie on opposite sides of $\mathcal{C}_i$ (in which case it still holds with exponent $s' \le s$). In this case T_i is the "bounding cone" of $\mathcal{C}_i$, ensuring that it intersects D in a simple way.

for some constant $L > 0$. We label the region between triangles T_{i-1} and T_i as U_i, where $T_0 := T_M$.

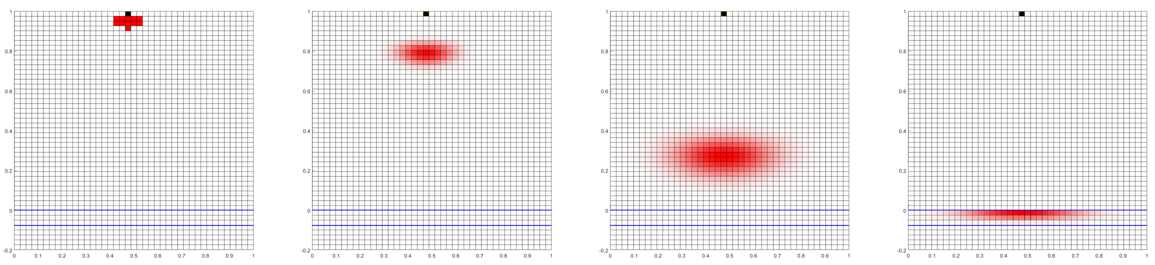
We assume that $u_0 \in W^{s,\infty}(U_i)$ for each U_i, but has a smaller Sobolev exponent $s' < s$ on $(0,1]\times(-\delta,0]$ as a whole. The T_i can be thought of as "bounding cones" of a series of curves $\mathcal{C}_i$ intersecting $\partial\Omega$ at $\{\mathbf{x}_i\}$, such that u_0 has lower regularity on each $\mathcal{C}_i$. For example, these curves might be places where either u_0 or ∇u_0 (if it exists) have jump discontinuities.

The assumption $\mathcal{C}_i \subset T_i$ ensures that each $\mathcal{C}_i$ intersects $\partial\Omega$ "nicely". In particular, if $\mathcal{C}_i$ is smooth, this means that the angle between the tangent to $\mathcal{C}_i$ at x_i and ∂D is bounded away from 0 (however, our assumption does not require that $\mathcal{C}_i$ be smooth). See Figure 6.5.

Connection to stopped random walks. Note that the update formula (6.1.2) gives a relationship between $u_h(\mathbf{x})$ and its immediate neighbors in a_r^*, which for now we assume obeys $a_r^* \subseteq b_r^-$ or $a_r^* \subseteq b_r^0$ (if this is not the case we can apply the method of equivalent weights from Section 5.1). Now suppose we modify (6.1.2) iteratively by repeated application of the following rule: for each $\mathbf{y} \in a_r^*$, if $\mathbf{x}+h\mathbf{y} \in D_h$, replace $u_h(\mathbf{x}+h\mathbf{y})$ in the RHS of (6.1.2) with the RHS of a version of (6.1.2) where the LHS is evaluated at $\mathbf{x}+h\mathbf{y}$ (in other words, we are substituting (6.1.2) into itself). Otherwise, if $\mathbf{x}+h\mathbf{y} \in \mathcal{U}_h$, we are already in the undamaged portion of the image, and we may replace $u_h(\mathbf{x}+h\mathbf{y})$ with $u_0(\mathbf{x}+h\mathbf{y})$. Repeat this procedure until $u_h(\mathbf{x})$ is entirely expressed as a weighted sum of $u_0\Big|_{\mathcal{U}_h}$, that is

$$u_h(\mathbf{x}) = \sum_{\mathbf{y}\in\mathcal{U}_h} \rho(\mathbf{y})u_0(\mathbf{y}), \tag{6.3.9}$$

for some as of yet unknown weights ρ. Denoting $\mathbf{x} := (nh, mh)$, then for the direct form of Algorithm 1 this procedure will terminate after m steps, as in this case (6.1.2) expresses $u_h(\mathbf{x})$ in terms of neighbors at least h units below it. On the other hand, for the semi-implicit extension, (6.3.9) has to be interpreted as a limit.



(a) $k = 0$ (b) $k = 4$ (c) $k = 17$ (d) $k = 40$

Figure 6.6: **Connection to Stopped Random Walks:** Here we illustrate the connection between the elimination procedure described in the text and stopped random walks. In (a) the pixel $\mathbf{x}$ (colored black) is expressed as a weighted average (2.0.3) of its neighbors in in b_r^- (colored in red). In this case we use the direct form of Algorithm 1 with $r = 3$ and uniform weights (which is why each neighbor in b_r^- is the same shade of red). In (c)-(d) we have applied $k = 4$, $k = 17$, and $k = 40$ steps of our elimination procedure (which essentially consists of substituting (2.0.3) into itself repeatedly - details in the text), and $u_h(\mathbf{x})$ is now expressed as weighted sum of $u_h(\mathbf{y})$ for whichever pixels $\mathbf{y}$ are colored red, with darker shades of red indicating greater weight. Since the stencil weights $\{\frac{w_r(\mathbf{0},\mathbf{y})}{W} : \mathbf{y} \in b_r^-\}$ are nonegative and sum to one, this is procedure is related to a stopped random walk $\mathbf{X}_\tau := (X_\tau, Y_\tau)$ started from $\mathbf{x}$ with stopping time $\tau = \inf\{n : Y_n \leq 0\}$. Specifically, after k steps of elimination we have $u_h(\mathbf{x}) = \sum_{\mathbf{y}\in\Omega_h} \rho_{\mathbf{X}_{\tau\wedge k}}(\mathbf{y})u_h(\mathbf{y})$, where $\rho_{\mathbf{X}_{\tau\wedge k}}$ is the density of $\mathbf{X}_{\tau\wedge k} := \mathbf{x} + h\sum_{i=1}^{\tau\wedge k} \mathbf{Z}_i$, a random walk with increments $\mathbf{Z}_i$ i.i.d. taking values in b_r^- with density $P(\mathbf{Z}_i = \mathbf{y}) = \frac{w_r(\mathbf{0},\mathbf{y})}{W}$. Since b_r^- is below the line $y = -1$, the density of $\rho_{\mathbf{X}_{\tau\wedge k}}$ necessarily moves at least one pixel downwards each iteration, and since $\mathbf{x}$ is only 40 pixels above $y = 0$, by $k = 40$ it the entire density is below $y = 0$ and the walk terminates. This process is illustrated in (b)-(d), where we see the density after 4 steps (b), 17 steps (c), and the final stopped density (d). Note that as $\mathbf{X}_\tau$ has increments of size at most rh, the density $\rho_{\mathbf{X}_\tau}$ necessarily lies between the lines $y = -rh$ and $y = 0$ (outlined in blue). The purpose of this procedure is to express the color of a given pixel $\mathbf{x}$ deep inside the inpainting domain entirely in terms of the colors of known pixels below $y = 0$.

This elimination procedure has a natural interpretation in terms of stopped random walks. Since the weights $\{\frac{w(\mathbf{0},\mathbf{y})}{W}\}_{\mathbf{y}\in a_r^*}$ are non-negative and sum to 1, we can interpret them as the density of a two dimensional random vector $\mathbf{Z} := (V, W)$ taking values in b_r^- or b_r^0 . Moreover, defining the random walk

$$\mathbf{X}_j := (X_j, Y_j) = (nh, mh) + h\sum_{i=1}^{j} \mathbf{Z}_i \tag{6.3.10}$$

with $\{\mathbf{Z}_i\}$ i.i.d. and equal to $\mathbf{Z}$ in distribution, and defining

$$\tau = \inf\{j : \mathbf{X}_j \in \mathcal{U}_h\} = \inf\{j : Y_j \leq 0\}, \tag{6.3.11}$$

then after k steps of elimination, we have

$$u_h(\mathbf{x}) = \sum_{\mathbf{y}\in\Omega_h} \rho_{\mathbf{X}_{j\wedge\tau}}(\mathbf{y})u_h(\mathbf{y}),$$

where $\rho_{\mathbf{X}_{j\wedge\tau}}$ denotes the density of $\mathbf{X}_{j\wedge\tau}$. Denoting the mean of $\mathbf{Z}$ by (μ_x, μ_y) note that by (6.3.2) we have the equivalence

$$(\mu_x, \mu_y) = \mathbf{g}_r^*.$$

In other words, the mean of $\mathbf{Z}$ is precisely the transport direction of our limiting equation (6.3.1). The condition $\mathbf{g}_r^* \cdot e_2 \neq 0$, which we needed for (6.3.1) to be defined, implies $\mu_y < 0$. In the nomenclature of random walks, this means that $\mathbf{X}_k$ has negative drift in the y direction, while τ is the first passage time through $y = 0$. Fortunately, this type of random walk and this type of first passage time have been studied and are well understood. See for example [32, Chapter 4], [33], [31], [30]. The book [32] also provides an good overview of stopped random walks in general. In particular, we know immediately that τ is a stopping time, $P(\tau = \infty) = 0$, and τ has finite moments of all orders [32, Chapter 3, Theorems 1.1 and 3.1]. It follows that

$$u_h(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{U}_h} \rho_{\mathbf{X}_\tau}(\mathbf{y}) u_0(\mathbf{y}) = E[u_0(\mathbf{X}_\tau)]. \tag{6.3.12}$$

This insight is central to our convergence argument and will also play a central role when we analyze smoothing artifacts in Section 6.7. See Figure 6.6 for an illustration of these ideas.

Theorem 6.3.1. *Let the continuous inpainting domain D and undamaged area $\mathcal{U}$, as well as their discrete counterparts D_h, $\mathcal{U}_h$ be as described in Section 6.1. Suppose we inpaint D_h using the direct form of Algorithm 1 or the semi-implicit extension, and denote the result by $u_h : D_h \to \mathbb{R}$. Assume the assumptions of Section 6.1 hold and let a_r^* and $\{w_r(\mathbf{0}, \mathbf{y}) : \mathbf{y} \in a_r^*\}$ denote the stencil and stencil weights respectively (defined in Definition 6.1.2) of our inpainting method. Let u denote the weak solution to the transport equation* (6.3.1)*, with transport direction equal to the center of mass of our stencil with respect to the stencil weights, that is*

$$\mathbf{g}_r^* = \sum_{\mathbf{y} \in a_r^*} \frac{w(\mathbf{0}, \mathbf{y})}{W} \mathbf{y} \quad \text{where } W = \sum_{\mathbf{y} \in a_r^*} w(\mathbf{0}, \mathbf{y}).$$

Suppose the boundary data $u_0 : \mathcal{U} \to \mathbb{R}$ obeys the regularity assumptions above, and let $\{U_i\}_{i=1}^M$ be as defined above and illustrated in Figure 6.4. Then for any $p \in [0, \infty]$ we have

$$\|u - u_h\|_p \leq K \cdot (rh)^{\left(\frac{s'}{2} + \frac{1}{2p}\right) \wedge \frac{s}{2} \wedge 1}$$

(the case $p = \infty$ is included by defining $1/\infty := 0$) where K is a constant depending only on u_0, $\mathcal{U}$, $\{U_i\}_{i=1}^M$, $\frac{r}{\mathbf{g}_r^* \cdot e_2}$ and θ_r^*. Moreover, K depends continuously on θ_r^* and $\frac{r}{\mathbf{g}_r^* \cdot e_2}$, is a monotonically increasing function of $\frac{r}{\mathbf{g}_r^* \cdot e_2}$, and $K \to \infty$ as $\theta_r^* \to 0$ or $\theta_r^* \to \pi$.

Remark 6.3.2. *Here we list some notable special cases of Theorem 6.3.1, together with their associated convergence rates:*

Uniform regularity: *If $u_0 \in C^{k,\alpha}(\mathcal{U})$, then*

$$\|u - u_h\|_p \leq \begin{cases} K \cdot (rh)^{\frac{k+\alpha}{2}} & \text{if } k \in \{0, 1\} \\ K \cdot (rh) & \text{if } k \geq 2. \end{cases}$$

In other words, the rate of convergence depends smoothly on the regularity of u_0 up until the C^2 level, beyond which additional smoothness has no effect. Note also that in this case the rate of convergence is independent of p.

Piecewise smooth: *If $u_0 \in C^{k,\alpha}(\mathcal{U} \backslash \{\mathcal{C}_i\}_{i=1}^M)$ where $k \geq 1$ and $\{\mathcal{C}_i\}_{i=1}^M$ are a series of curves such that $\mathcal{C}_i \subseteq T_i$ for each i as in Figure 6.5, but u_0 is discontinuous on $\mathcal{U}$ as a whole, then we have*

$$\|u - u_h\|_p \leq K \cdot (rh)^{\frac{1}{2p}}.$$

Thus, in the piecewise smooth case our rates of convergence are independent of the regularity of u_0 on $\mathcal{U} \backslash \{\mathcal{C}_i\}_{i=1}^M$ so long as it is at least C^1, but now depend on p. In particular, the convergence rate is a *monotonically decreasing function of p, and we converge in L^p for every $1 \leq p < \infty$, but not necessarily in L^∞.*

Piecewise Hölder continuous: *If $u_0 \in C^{0,\alpha}(\mathcal{U}\backslash\{\mathcal{C}_i\}_{i=1}^M)$ but discontinuous on $\mathcal{U}$ as a whole (where $\{\mathcal{C}_i\}_{i=1}^M$ are the same as in the previous example), then we have*

$$\|u - u_h\|_p \leq \begin{cases} K \cdot (rh)^{\frac{\alpha}{2}} & \text{if } 1 \leq p \leq \frac{1}{\alpha} \\ K \cdot (rh)^{\frac{1}{2p}} & \text{if } \frac{1}{\alpha} < p \leq \infty. \end{cases}$$

In this case the convergence rate is independent of p for $1 \leq p \leq \frac{1}{\alpha}$, but is the same as the previous example for $p > \frac{1}{\alpha}$.

6.4 Proof of Theorem 6.3.1

First a note on notation. Let us define for convenience $\hat{\mathbf{x}} = \Pi_{\theta_r^*}(\mathbf{x})$, so that the solution to (6.3.1) may be more compactly written as

$$u(\mathbf{x}) = u_0(\hat{\mathbf{x}}).$$

For brevity, even though $\mathbf{X}_\tau$, $\hat{\mathbf{x}}$, and τ depend implicitly on $\mathbf{x} = (nh, mh)$, we do not write this dependence down explicitly. We will also adopt the notation that if $\mathbf{X} = (X_1, X_2)$ and X denote vector and scalar valued random variables respectively, then

$$\|\mathbf{X}\|_{L^p} := E[\|\mathbf{X}\|^p]^{\frac{1}{p}} \quad \text{and} \quad \|X\|_{L^p} := E[|\mathbf{X}|^p]^{\frac{1}{p}},$$

where $\|\mathbf{X}\| := \|\mathbf{X}\|_2$ denotes the *euclidean* norm of $\mathbf{X}$. The identity

$$\|\mathbf{X}\|_{L^p} \leq \|X_1\|_{L^p} + \|X_2\|_{L^p} \tag{6.4.1}$$

(which follows trivially from the identity $\|\mathbf{X}\|_2 \leq \|\mathbf{X}\|_1$ and the triangle inequality with respect to the norm $\|\cdot\|_{L^p}$) will occasionally be useful. Finally, note that by the linearity of expectation, we have

$$\|u - u_h\|_p = \|E[u_0(\mathbf{X}_\tau) - u_0(\hat{\mathbf{x}})]\|_p \tag{6.4.2}$$

Roadmap. Our proof consists of six stages, which we go through before proceeding. Stages 1-5 assume that $a_r^* \subseteq b_r^-$ or $a_r^* \subseteq b_r^0$ for simplicity. Stage 6 generalizes to arbitrary a_r^*.

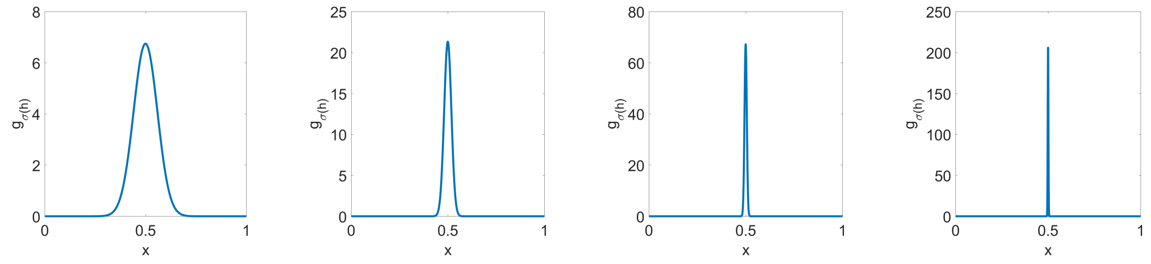
1. **Stage 1:** Obtain bounds, as a function of h, on the rate at which $\mathbf{X}_\tau$ is concentrating around its mean $E[\mathbf{X}_\tau]$, as well as the rate that $E[\mathbf{X}_\tau]$ itself is converging to $\hat{\mathbf{x}}$. For technical reasons, we will require in particular bounds on $\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4}$. In effect what is happening is that

   $$\rho_{\mathbf{X}_\tau} \to \delta_{\hat{\mathbf{x}}} \qquad \text{in } \mathcal{D}'(\mathcal{U}) \text{ as } h \to 0,$$

   where $\delta_{\hat{\mathbf{x}}}$ denotes the Dirac delta distribution centered at $\hat{\mathbf{x}}$ (see Figure 6.7). Formally speaking, we have

   $$u_h(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{U}_h} \rho_{\mathbf{X}_\tau}(\mathbf{y}) u_0(\mathbf{y}) = \int_{\mathcal{U}} u_0(\mathbf{y}) d\mu_{\mathbf{X}_\tau}(\mathbf{y}) \to \langle \delta_{\hat{\mathbf{x}}}, u_0 \rangle = u_0(\hat{\mathbf{x}}),$$

   where $\mu_{\mathbf{X}_\tau}$ denotes the measure associated with the density of $\mathbf{X}_\tau$. We will not attempt to make this argument rigorous, and mention it only for the sake of building intuition. For this part of the proof, known results from [32, Chapter 4] will save us some effort.



(a) $h = 1e-2$. (b) $h = 1e-3$. (c) $h = 1e-4$. (d) $h = 1e-5$.

Figure 6.7: **Theorem 6.3.1 through the lense of distribution theory:** Theorem 6.3.1 says that for $\mathbf{x} \in D_h$, $u_h(\mathbf{x}) \to u_0(\hat{\mathbf{x}})$ in L^p as $h \to 0$ for every $p \in [1,\infty]$ if u_0 is continuous, but possibly not in L^∞ if u_0 contains discontinuities. Here $\hat{\mathbf{x}} = \Pi_{\theta_r^*}(\mathbf{x})$ is the transport operator (6.3.3) applied to $\mathbf{x}$. One way of understanding this is via the identity $u_h(\mathbf{x}) = \sum_{\mathbf{y}\in\mathcal{U}_h} \rho_{\mathbf{X}_\tau}(\mathbf{y})u_0(\mathbf{y})$ (6.3.12). As we will see in Section 6.7, for each fixed $\mathbf{x} \in D_h$, the x-coordinate X_τ of the stopped random walk $\mathbf{X}_\tau$ started from $\mathbf{x}$ converges in distribution to a Gaussian $g_{\sigma(h)}$ with mean $\hat{\mathbf{x}}$ and h-dependent variance $\sigma(h)^2$, such that $g_{\sigma(h)}$ itself is converging to a one dimensional Dirac delta distribution centered at $\hat{\mathbf{x}}$ as $h \to 0$. At the same time, since $\mathbf{X}_\tau$ can only overshoot $y = 0$ by distance at most rh, it follows that the density of $\mathbf{X}_\tau$ lies entirely between the lines $y = -rh$ and $y = 0$. These two facts together imply that $\mathbf{X}_\tau$ converges in distribution as $h \to 0$ to a two dimensional Dirac delta distribution centered at $\hat{\mathbf{x}}$. Hence we expect $u_h(\mathbf{x}) \to u_0(\hat{\mathbf{x}})$, at least when $\hat{\mathbf{x}}$ is a continuity point of u_0 (if $\hat{\mathbf{x}}$ is not a continuity point, then we need to apply the theory of distributions with discontinuous test functions - see for example [24] - and we do not expect $u_h(\mathbf{x}) \to u_0(\hat{\mathbf{x}})$ in general). We illustrate this in (a)-(d), which show plots of $g_{\sigma(h)}$ for Guidefill with $r = 3$, $\mathbf{g} = (\cos 45°, \sin 45°)$ and $\mu \to \infty$ for various values of h. We fix $\mathbf{x} = (0.5, 1)$ so that $\hat{\mathbf{x}} = (0.5, 0)$ (remember the periodic boundary conditions).

2. **Stage 2:** Use the assumed regularity of u_0 to obtain bounds on $|E[u_0(\mathbf{X}_\tau) - u_0(\hat{\mathbf{x}})]|$. In particular, we will obtain:
   (a) A bound that holds for *any* starting position $\mathbf{x}$ of the random walk $\mathbf{X}_\tau$, and depends only on $\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4}$ and $\|E[\mathbf{X}_\tau] - \hat{\mathbf{x}}\|$.
   (b) A tighter bound that holds assuming $E[\mathbf{X}_\tau]$ and $\hat{\mathbf{x}}$ both belong to one of the well behaved sets U_i (see Figure 6.4), and depends not only on $\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4}$ and $\|E[\mathbf{X}_\tau] - \hat{\mathbf{x}}\|$, but also on the "rogue event" that even though $\hat{\mathbf{x}} \in U_i$ and $E[\mathbf{X}_\tau] \in U_i$, we nonetheless have $\mathbf{X}_\tau \notin U_i$.
3. **Stage 3:** Partition D into bands $\{B_i\}_{i=1}^M$ and sub-bands $\tilde{B}_i \subset B_i$ such that the area of the complement $B_i \backslash \tilde{B}_i$ goes to zero as $h \to 0$, and such that the starting position $\mathbf{x} \in \tilde{B}_i$ guarantees $E[\mathbf{X}_\tau] \in U_i$ and $\hat{\mathbf{x}} \in U_i$. See Figure 6.8 for an illustration.
4. **Stage 4:** Bound the probability of the rogue event from stage 2, under the assumption that the starting position $\mathbf{x} \in \tilde{B}_i$ for some $1 \le i \le M$.
5. **Stage 5:** Substitute the bounds from the previous four stages into (6.4.2) to obtain a bound for $\|u - u_h\|_p$.
6. **Stage 6:** Generalize to arbitrary a_r^* containing ghost pixels by exploiting the idea of equivalent weights from Section 5.1.

Steps 1 and 4 will occasionally require us to use some elementary results from martingale theory, including Wald's first identity and Azuma's inequality. For a review of martingale theory, see for example [70].

Stage 1. Throughout this stage we will be using the following Theorem from [32, Chapter 1]:

Theorem 6.4.1. *Suppose $E[|X_1|^r] < \infty$ for some $0 < r < \infty$, $E[X_1] = 0$, and define the stopped random walk $S_\tau = \sum_{i=1}^{\tau} X_i$, where $\{X_i\}$ are i.i.d. Then*

$$E[|S_\tau|^p] \leq B_p E[|X_1|^p] E[\tau^{\frac{p}{2}}].$$

where B_p is a constant depending only on p.

The result of this stage is the following lemma:

Lemma 6.4.2. *Let $\mathbf{X}_\tau := (X_\tau, Y_\tau) = \mathbf{x} + h\sum_{i=1}^{\tau} \mathbf{Z}_i = (nh, mh) + h\sum_{i=1}^{\tau}(V_i, W_i)$ with $\{\mathbf{Z}_i\}$ i.i.d. as above. Then*

(i) $\|E[\mathbf{X}_\tau] - \hat{\mathbf{x}}\| \leq rh$.

In addition, there is a constant $C > 0$ dependent only on $\frac{\mu_y}{r}$ and θ_r^* such that

(ii) $\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4} \leq C\sqrt{rh}$.

Moreover, $C \to \infty$ as $\theta_r^* \to \{0, \pi\}$ or $\frac{\mu_y}{r} \to 0$.

Proof. The idea of the proof is to combine repeated application of Theorem 6.4.1 with geometric observations of the situation at hand - specifically, the fact that $|\mathbf{Z}_i| \leq r$ means that $\mathbf{X}_\tau$ can overshoot $y = 0$ by a distance of at most rh. First, note that since $E[\tau] < \infty$, Wald's first identity

$$E[\mathbf{X}_\tau] = (nh, mh) + E[\tau](\mu_x h, \mu_y h)$$

is applicable. This implies that the ray joining (nh, mh) with $E[\mathbf{X}_\tau]$ is parallel to the ray from (nh, mh) to $\hat{\mathbf{x}}$. At the same time, since $|\mathbf{X}_{k+1} - \mathbf{X}_k| \leq rh$, it follows that $E[\mathbf{X}_\tau]$ "overshoots" $\hat{\mathbf{x}}$ by a distance of at most rh, from which we immediately have

$$\|E[\mathbf{X}_\tau] - \hat{\mathbf{x}}\| \leq rh.$$

This proves the first claim. For the second claim, first note that by (6.4.1), we have

$$\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4} \leq \|X_\tau - E[X_\tau]\|_{L^4} + \|Y_\tau - E[Y_\tau]\|_{L^4} \leq \|X_\tau - E[X_\tau]\|_{L^4} + rh, \tag{6.4.3}$$

where we have used the overshooting observation again to obtain the bound
$|Y_\tau - E[Y_\tau]| \leq rh$. Thus it suffices to bound $\|X_\tau - E[X_\tau]\|_{L^4}$. To do so, we are going to use Theorem 6.4.1. This means we are going to need an estimate for $E[\tau^2]$, and our first task is to find one. To that end, first note that for any $1 \leq p < \infty$ we have

$$\|\mu_y\tau - \mu_y E[\tau]\|_{L^p} \leq \|\sum_{i=1}^{\tau} W_i - \tau\mu_y\|_{L^p} + \|(m + \sum_{i=1}^{\tau} W_i) - (m + E[\tau]\mu_y)\|_{L^p}.$$

The second term on the RHS is exactly $\frac{1}{h}\|Y_\tau - E[Y_\tau]\|_{L^p}$, and is bounded above by r since, by our overshooting observation, Y_τ and $E[Y_\tau]$ both must lie in the interval $[-rh, 0]$. For the first term, applying Theorem 6.4.1 and noting $E[|W_1|^p] \leq r^p$ gives $\|\sum_{i=1}^{\tau} W_i - \tau\mu_y\|_{L^p} \leq (B_p)^{\frac{1}{p}} r E[\tau^{\frac{p}{2}}]^{\frac{1}{p}}$, from which it follows that

$$\|\tau - E[\tau]\|_{L^p} \leq \frac{1}{|\mu_y|}\left[(B_p)^{\frac{1}{p}} r E[\tau^{\frac{p}{2}}]^{\frac{1}{p}} + r\right] \leq \frac{1}{|\mu_y|}[(B_p)^{\frac{1}{p}} + 1] r E[\tau^{\frac{p}{2}}]^{\frac{1}{p}} \tag{6.4.4}$$

Since $\tau \geq 1$. Next, we take $p = 2$ and square both sides to obtain

$$E[\tau^2] - E[\tau]^2 \leq \frac{1}{|\mu_y|^2}[(B_2)^{\frac{1}{2}} + 1]^2 r^2 E[\tau].$$

But, since $-rh \leq E[Y_\tau] = mh + E[\tau]\mu_y h \leq 0$, it follows that

$$E[\tau] \leq \frac{m+1}{|\mu_y|} \leq \frac{2N}{|\mu_y|} = \frac{2}{|\mu_y|h}.$$

After some short algebra this gives

$$E[\tau^2] \leq \left(4 + 2\frac{[(B_2)^{\frac{1}{2}} + 1]^2 r^2}{\mu_y^2}\right) \frac{1}{|\mu_y|^2 h^2}.$$

Next, note that

$$\begin{aligned} \|X_\tau - E[X_\tau]\|_{L^p} &= \|(nh + h\sum_{i=1}^{\tau} V_i - (nh + E[\tau]\mu_x)\|_{L^p} \\ &\leq h\|\sum_{i=1}^{\tau} V_i - \tau\mu_x\|_{L^p} + h|\mu_x|\|\tau - E[\tau]\|_{L^p}. \end{aligned}$$

The first term in the RHS we bound with another application of Theorem 6.4.1, together with the observation $E[|V_1|^p] \leq r^p$. For the second term we already have the bound (6.4.4). Together we get

$$\|X_\tau - E[X_\tau]\|_{L^p} \leq \left((B_p)^{\frac{1}{p}} + \frac{|\mu_x|}{|\mu_y|}(1 + (B_p)^{\frac{1}{p}})\right) rhE[\tau^{\frac{p}{2}}]^{\frac{1}{p}}.$$

Setting now $p = 4$ and applying our bound on $E[\tau^2]$ gives

$$\|X_\tau - E[X_\tau]\|_{L^4} \leq \left[(B_4)^{\frac{1}{4}} + \frac{|\mu_x|}{|\mu_y|}(1 + (B_4)^{\frac{1}{4}})\right] \left[4 + 2(1 + (B_2)^{\frac{1}{2}})^2 \frac{r^2}{|\mu_y|^2}\right]^{\frac{1}{4}} \sqrt{rh}.$$

Since $rh < 1$ it follows that $rh < \sqrt{rh}$. Combining the above bound with (6.4.3) gives

$$\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4} \leq C\sqrt{rh}$$

as claimed, with

$$C := 1 + \left((B_4)^{\frac{1}{4}} + \cot(\theta_r^*)(1 + (B_4)^{\frac{1}{4}})\right)\left(4 + 2(1 + (B_2)^{\frac{1}{2}})^2 \frac{r^2}{|\mu_y|^2}\right)^{\frac{1}{4}}.$$

□

Stage 2. The previous stage established quantitative bounds on the rate at which $\mathbf{X}_\tau$ is concentrating around its mean, while its mean converges to $\hat{\mathbf{x}}$. The next step is to use the regularity of u_0 to express the rate at which $|E[u_0(\mathbf{X}_\tau) - u_0(\hat{\mathbf{x}})]|$ is tending towards zero in terms of these rates, as well as the probability of a "rogue" event (which we denote by E_s^c) that $E[\mathbf{X}_\tau]$ and $\hat{\mathbf{x}}$ each belong to one of the well behaved sets U_i from Figure 6.4, but $\mathbf{X}_\tau$ does not. Specifically, our bound will depend on $P(E_s^c)^{\frac{1}{2}}$, and it was in order to obtain a power of $\frac{1}{2}$ that we needed the fourth moment of $\mathbf{X}_\tau$ - if we had only the variance, we would end up with a power that depends on the regularity constants $0 \leq s' \leq s$. This is accomplished by the following lemma, which, although we have stated it in a slightly more general setting with a general random vector $\mathbf{X}$ taking values in a general convex set $\Omega \subset \mathbb{R}^2$, general convex sets $\{U_i\}_{i=1}^M$ contained within Ω, a general function $u : \Omega \to \mathbb{R}$, a general norm $\|\mathbf{X} - E[\mathbf{X}]\|_{L^{p'}}$ with $p' \geq 4$, and a general point $\hat{\mathbf{x}} \in \Omega$, we will ultimately be applying this lemma only to $\mathbf{X}_\tau$, $\Omega = \mathcal{U}$, $p' = 4$, and $\{U_i\}$ and $\hat{\mathbf{x}}$ as described above.

Lemma 6.4.3. *Let $\Omega \subset \mathbb{R}^2$ be convex and suppose $u \in W^{s',\infty}(\Omega)$, and in addition $u \in W^{s,\infty}(U_i)$ $(s \geq s')$ for each of a finite collection of convex sets $\{U_i\}_{i=1}^M$ where each $U_i \subseteq \Omega$. Let $\hat{\mathbf{x}} \in \Omega$ and let $\mathbf{X}$ be a random vector taking values in Ω. Define*

1. $C_{u,\Omega} := \|u\|_{W^{s',\infty}(\Omega)}$,

2. $C_{u,\Omega,\{U_i\}} := \max\left\{\|u\|_{W^{s',\infty}(\Omega)}, \|u\|_{W^{s,\infty}(U_1)}, \dots, \|u\|_{W^{s,\infty}(U_M)}\right\}$

Then for any $p' \geq 1$ we have

1.
$$|E[u(\mathbf{X}) - u(\hat{\mathbf{x}})]| \leq \quad C_{u,\Omega}\left\{(\|\mathbf{X} - E[\mathbf{X}]\|_{L^{p'}})^{s'\wedge 2} + \|E[\mathbf{X}] - \hat{\mathbf{x}}\|^{s'\wedge 1}\right\},$$

Next, suppose $\hat{\mathbf{x}} \in U_i$, $E[\mathbf{X}] \in U_i$ for some $1 \leq i \leq M$. Denote by E_s the event that $\mathbf{X} \in U_i$ as well. Then for any $p' \geq 4$ such that $\|\mathbf{X} - E[\mathbf{X}]\|_{L^{p'}} \leq 1$ we have

2.
$$\begin{aligned}|E[u(\mathbf{X}) - u(\hat{\mathbf{x}})]| \quad &\leq \quad C_{u,\Omega,\{U_i\}}\{(\|\mathbf{X} - E[\mathbf{X}]\|_{L^{p'}})^{s\wedge 2} \\ &+ \quad 2(\|\mathbf{X} - E[\mathbf{X}]\|_{L^{p'}})^{s'\wedge 2}P(E_s^c)^{\frac{1}{2}} + \|E[\mathbf{X}] - \hat{\mathbf{x}}\|^{s\wedge 1}\},\end{aligned}$$

Proof. First let us define for convenience

$$\mathbf{x}^* := E[\mathbf{X}], \qquad \mathbf{x}^*_{E_s} := E[\mathbf{X}|E_s], \qquad \mathbf{x}^*_{E_s^c} := E[\mathbf{X}|E_s^c].$$

For both statements, the first step is to divide the expectation into two pieces.

$$|E[u(\mathbf{X}) - u(\hat{\mathbf{x}})]| \leq |E[u(\mathbf{X}) - u(\mathbf{x}^*)]| + |u(\mathbf{x}^*) - u(\hat{\mathbf{x}})| := \Pi_1 + \Pi_2.$$

To prove statement one, we apply the Hölder condition (6.3.8) to find
$\Pi_2 \leq C_{u,\Omega}\|\mathbf{x}^* - \hat{\mathbf{x}}\|^{s'\wedge 1}$. Hence it suffices to prove $\Pi_1 \leq C_{u,\Omega}E[\|\mathbf{X} - E[\mathbf{X}]\|^{p'}]^{\frac{s'\wedge 2}{p'}}$. We proceed by cases. If $s' < 1$, then

$$\Pi_1 \leq C_{u,\Omega}E[\|\mathbf{X} - \mathbf{x}^*\|^{s'}] \leq C_{u,\Omega}E[\|\mathbf{X} - \mathbf{x}^*\|^{p'}]^{\frac{s'}{p'}} = C_{u,\Omega}E[\|\mathbf{X} - \mathbf{x}^*\|^{p'}]^{\frac{s'\wedge 2}{p'}}.$$

where we have used Jensen's inequality together with the concavity of $x^{\frac{s'}{p'}}$ on $[0, \infty)$ for the second inequality. On the other hand, if $s' \geq 1$, then ∇u exists and by Taylor's theorem $u(\mathbf{X}) - u(\mathbf{x}_*) = \nabla u(\mathbf{z}) \cdot (\mathbf{X} - \mathbf{x}_*)$ where $\mathbf{z} = (1-t)\mathbf{x}_* + t\mathbf{X}$ for some $t \in [0, 1]$. Therefore

$$\Pi_1 = |E[\nabla u(\mathbf{z}) \cdot (\mathbf{X} - \mathbf{x}^*)]| = |E[(\nabla u(\mathbf{z}) - \nabla u(\mathbf{x}^*)) \cdot (\mathbf{X} - \mathbf{x}^*)] + E[\nabla u(\mathbf{x}^*) \cdot (\mathbf{X} - \mathbf{x}^*)]|.$$

But
$$E[\nabla u(\mathbf{x}^*) \cdot (\mathbf{X} - \mathbf{x}^*)] = \nabla u(\mathbf{x}^*) \cdot E[\mathbf{X} - \mathbf{x}^*] = \nabla u(\mathbf{x}^*) \cdot (\mathbf{x}^* - \mathbf{x}^*) = 0,$$

therefore

$$\begin{aligned}\Pi_1 &= |E[(\nabla u(\mathbf{z}) - \nabla u(\mathbf{x}^*)) \cdot (\mathbf{X} - \mathbf{x}^*)]| \\ &\leq E[\|\nabla u(\mathbf{z}) - \nabla u(\mathbf{x}^*)\| \|\mathbf{X} - \mathbf{x}^*\|] \\ &\leq C_{u,\Omega} E[\|\mathbf{z} - \mathbf{x}^*\|^{(s'-1)\wedge 1} \|\mathbf{X} - \mathbf{x}^*\|] \\ &\leq C_{u,\Omega} E[\|\mathbf{X} - \mathbf{x}^*\|^{s'\wedge 2}] \\ &\leq C_{u,\Omega} E[\|\mathbf{X} - E[\mathbf{X}]\|^{p'}]^{\frac{s'\wedge 2}{p'}},\end{aligned}$$

where we have used the Cauchy-Schwarz inequality on line two, the Hölder condition (6.3.8) together with the convexity of Ω on line three, the bound
$\|\mathbf{z} - \mathbf{x}^*\| \leq \|\mathbf{X} - \mathbf{x}^*\|$ on line four, and Jensen's inequality again on line five.

For the second statement, the idea is to split the expectation up as a sum of conditional expectations conditioning on E_s and E_s^c, apply the corresponding regularity estimates, and express the results in terms of the conditional moments of $\mathbf{X}$. Then, using the contractive property of conditional expectation, namely

$$E[\|E[\mathbf{X}|Y]\|^{p'}] \leq E[\|\mathbf{X}\|^{p'}]$$

valid for all random variables Y adapted to the same sigma algebra as $\mathbf{X}$ and all $p' \geq 1$, we can eliminate the conditional moments. We do not use the contractive property itself, but rather two identities which may be easily derived from it. In particular, let $A \in \sigma(\mathbf{X})$ and define $\mathbf{x}_A^* = E[\mathbf{X}|A]$. Then for any $p' \geq 1$

$$\begin{aligned}\|\mathbf{x}_A^* - \mathbf{x}^*\|^{p'} P(A) &\leq \|E[\mathbf{X} - \mathbf{x}^*|A]\|^{p'} P(A) + \|E[\mathbf{X} - \mathbf{x}^*|A^c]\|^{p'} P(A^c) \\ &= E[\|E[\mathbf{X} - \mathbf{x}^*|1_A]\|^{p'}] \\ &\leq E[\|\mathbf{X} - \mathbf{x}^*\|^{p'}],\end{aligned} \tag{6.4.5}$$

and a similar manipulation gives

$$E[\|\mathbf{X} - \mathbf{x}^*\|^{p'}|A] P(A) \leq E[\|\mathbf{X} - \mathbf{x}^*\|^{p'}]. \tag{6.4.6}$$

Similarly to part one, we have $\Pi_2 \leq C_{u,\Omega,\{U_i\}} \|\mathbf{x}^* - \hat{\mathbf{x}}\|^{s\wedge 1}$, since $\mathbf{x}^*, \hat{\mathbf{x}} \in U_i$. It remains to prove

$$\begin{aligned}\Pi_1 \leq \Pi_1^* &:= C_{u,\Omega,\{U_i\}} E[\|\mathbf{X} - E[\mathbf{X}]\|^{p'}]^{\frac{s\wedge 2}{p'}} \\ &+ 2C_{u,\Omega,\{U_i\}} E[\|\mathbf{X} - E[\mathbf{X}]\|^{p'}]^{\frac{s'\wedge 2}{p'}} P(E_s^c)^{\frac{1}{2}}.\end{aligned}$$

We split Π_1 up as

$$\begin{aligned}\Pi_1 &= |E[u(\mathbf{X}) - u(\mathbf{x}^*)]| \\ &= |E[u(\mathbf{X}) - u(\mathbf{x}^*)|E_s] P(E_s) \\ &+ E[u(\mathbf{X}) - u(\mathbf{x}^*)|E_s^c] P(E_s^c)| \\ &:= |\Pi_{1,1} + \Pi_{1,2}|\end{aligned}$$

and proceed by cases. First assume $0 \leq s' \leq s < 1$. Then, by a manipulation identical to the case $s' < 1$ in part one, we have

$$\Pi_{1,1} \leq C_{u,\Omega,\{U_i\}} (E[\|\mathbf{X} - \mathbf{x}^*\|^{p'}|E_s])^{\frac{s\wedge 2}{p'}} P(E_s)$$

and

$$\Pi_{1,2} \le C_{u,\Omega,\{U_i\}}(E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}|E_s^c])^{\frac{s'\wedge 2}{p'}}P(E_s^c).$$

We therefore have

$$\begin{aligned}
\Pi_1 &\le C_{u,\Omega,\{U_i\}}(E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}|E_s]P(E_s))^{\frac{s\wedge 2}{p'}}P(E_s)^{1-\frac{s\wedge 2}{p'}} \\
&+ C_{u,\Omega,\{U_i\}}(E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}|E_s^c]P(E_s^c))^{\frac{s'\wedge 2}{p'}}P(E_s^c)^{1-\frac{s'\wedge 2}{p'}} \\
&\le C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s\wedge 2}{p'}} + C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s'\wedge 2}{p'}}P(E_s^c)^{\frac{1}{2}} \\
&\le \Pi_1^*.
\end{aligned}$$

Where we have used (6.4.6) and $p' \ge 4$ on line three. Next suppose $0 \le s' \le 1$ but $s \ge 1$. Then ∇u exists on U_i and has norm bounded by $C_{u,\Omega,\{U_i\}}$, and hence whenever $\mathbf{X} \in U_i$ we can find a $\mathbf{z} = t\mathbf{x}^* + (1-t)\mathbf{X} \in U_i$ for some $t \in [0,1]$ such that $u(\mathbf{X}) = \nabla u(\mathbf{z}) \cdot (\mathbf{X}-\mathbf{x}^*)$, by Taylor's theorem and the convexity of U_i. Therefore

$$\begin{aligned}
\Pi_{1,1} &\le |E[(\nabla u(\mathbf{z}) - \nabla u(\mathbf{x}^*)) \cdot (\mathbf{X}-\mathbf{x}^*)|E_s]P(E_s) \\
&+ \nabla u(\mathbf{x}^*) \cdot E[\mathbf{X}-\mathbf{x}^*|E_s]P(E_s)|,
\end{aligned}$$

For the first term we can apply an argument that is identical to the case $s' \ge 1$ in part one. However, unlike in part one, the second term does not vanish as our expectation is now conditional. We obtain

$$\begin{aligned}
\Pi_{1,1} &\le (E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}|E_s]P(E_s))^{\frac{s\wedge 2}{p'}}P(E_s)^{1-\frac{s\wedge 2}{p'}} + \|\nabla u(\mathbf{x}^*)\|\|\mathbf{x}^*_{E_s}-\mathbf{x}^*\|P(E_s) \\
&\le C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s\wedge 2}{p'}} + \|\nabla u(\mathbf{x}^*)\|\|\mathbf{x}^*_{E_s^c}-\mathbf{x}^*\|P(E_s^c) \\
&\le C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s\wedge 2}{p'}} + C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{1}{p'}}P(E_s^c)^{1-\frac{1}{p'}}.
\end{aligned}$$

Where we have used the manipulation

$$\mathbf{x}^*_{E_s}P(E_s) + \mathbf{x}^*_{E_s^c}P(E_s^c) = \mathbf{x}^* = \mathbf{x}^*P(E_s) + \mathbf{x}^*P(E_s^c)$$

$$\Rightarrow \quad (\mathbf{x}^* - \mathbf{x}^*_{E_s})P(E_s) = (\mathbf{x}^*_{E_s^c} - \mathbf{x}^*)P(E_s^c). \tag{6.4.7}$$

together with (6.4.6) on line two, and (6.4.5) on line three. The final trick is to note that since $E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}] \le 1$ and $s' \le 1$, we have $E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{1}{p'}} \le E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s'\wedge 2}{p'}}$. Hence, bounding $\Pi_{1,2}$ in exactly the same way as in the previous case, we have

$$\Pi_1 \le C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s\wedge 2}{p'}} + 2C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s'\wedge 2}{p'}}P(E_s^c)^{\frac{1}{2}} \le \Pi_1^*,$$

where we have used $p \ge 4$ again to write $P(E_s^c)^{\frac{1}{p'}} \le P(E_s^c)^{\frac{1}{2}}$. Finally, we consider $1 \le s' \le s$. In this case ∇u exists everywhere and applying Taylor's theorem twice we have

$$\begin{aligned}
\Pi_1 &= |E[(\nabla u(\mathbf{z}_1) - \nabla u(\mathbf{x}^*)) \cdot (\mathbf{X}-\mathbf{x}^*)|E_s]P(E_s) \\
&+ E[(\nabla u(\mathbf{z}_2) - \nabla u(\mathbf{x}^*)) \cdot (\mathbf{X}-\mathbf{x}^*)|E_s^c]P(E_s^c) \\
&+ \nabla u(\mathbf{x}^*) \cdot \underbrace{\left\{(\mathbf{x}^*_{E_s}-\mathbf{x}^*)P(E_s) + (\mathbf{x}^*_{E_s^c}-\mathbf{x}^*)P(E_s^c)\right\}}_{=0 \text{ by } (6.4.7)}|,
\end{aligned}$$

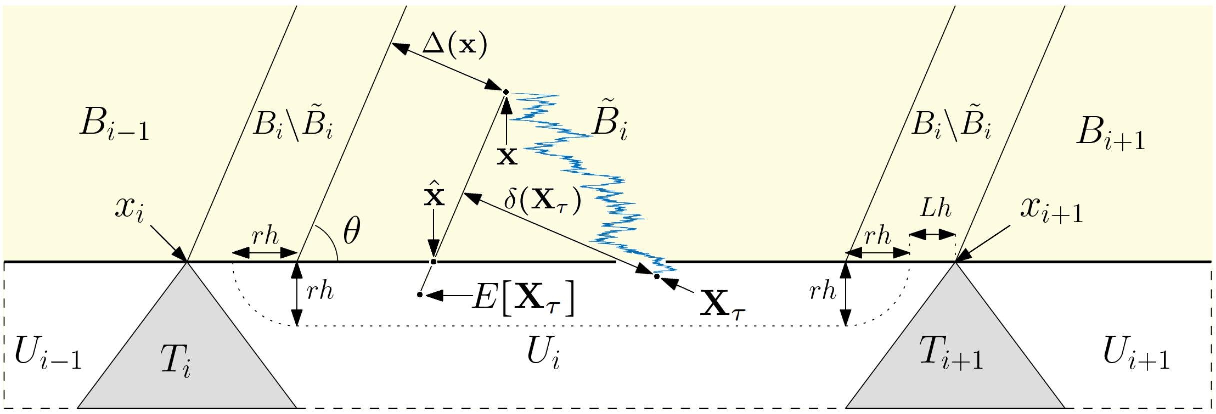


Figure 6.8: **Illustration of the stopped random walk $\mathbf{X}_\tau$ as well as the subdivision of D into bands:** The inpainting domain $D = (0,1]^2$ is partitioned into bands $\{B_i\}_{i=1}^M$, such that each band $B_i := \Pi_{\theta_r^*}^{-1}((x_i, x_{i+1}] \times \{0\})$, where $\Pi_{\theta_r^*}$ is the transport map (6.3.3). With in each band we identify a subband $\tilde{B}_i$, of width dependent on h, such that if $\mathbf{X}_\tau$ starts in $\tilde{B}_i$, then $E[\mathbf{X}_\tau] \in U_i$ and $\hat{\mathbf{x}} \in U_i$. As $h \to 0$, the area of $B_i \backslash \tilde{B}_i$ goes to zero. At the same time we illustrate an example stopped random walk $\mathbf{X}_\tau$ started from $\mathbf{x}$ (blue curve) as well the relationship between $\mathbf{x}$, $E[\mathbf{X}_\tau]$ and $\hat{\mathbf{x}} = \Pi_{\theta_r^*}(\mathbf{x})$. The orthogonal distances $\Delta(\mathbf{x})$ from $\mathbf{x}$ to $\partial\tilde{B}_i$ and $\delta(\mathbf{X}_\tau)$ from $\mathbf{X}_\tau$ to the ray from $\mathbf{x}$ to $\hat{\mathbf{x}}$ are also illustrated. Note that $\hat{\mathbf{x}}$ is always on the line $y = 0$, whereas $\mathbf{X}_\tau$ and $E[\mathbf{X}_\tau]$ will typically overshoot it. However, they can only overshoot by distance at most rh. That is, $\mathbf{X}_\tau$ and $E[\mathbf{X}_\tau]$ must always sit between the lines $y = -rh$ and $y = 0$. Note that Pythagorean theorem implies $\Delta(\mathbf{x}) \le 1$ regardless of where $\mathbf{x}$ is placed in D and regardless of the number and position of the bands.

where $\mathbf{z}_1 = t_1\mathbf{x}^* + (1-t_1)\mathbf{X} \in U_i$, $\mathbf{z}_2 = t_2\mathbf{x}^* + (1-t_2)\mathbf{X} \in U_i$ for some $t_1, t_2 \in [0,1]$ (again, by the convexity of U_i). Applying an argument almost identical to the previous step, we then obtain

$$
\begin{aligned}
\Pi_1 &\le C_{u,\Omega,\{U_i\}}(E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}|E_s]P(E_s))^{\frac{s\wedge 2}{p'}}P(E_s)^{1-\frac{s\wedge 2}{p'}} \\
&+ C_{u,\Omega,\{U_i\}}(E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}|E_s]P(E_s^c))^{\frac{s'\wedge 2}{p'}}P(E_s^c)^{1-\frac{s'\wedge 2}{p'}} \\
&\le C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s\wedge 2}{p'}} + C_{u,\Omega,\{U_i\}}E[\|\mathbf{X}-\mathbf{x}^*\|^{p'}]^{\frac{s'\wedge 2}{p'}}P(E_s^c)^{\frac{1}{2}} \\
&\le \Pi_1^*,
\end{aligned}
$$

where we have used (6.4.6) and $p' \ge 4$ on line three. □

Stage 3. In this stage our objective is to partition D into a series of bands $\{B_i\}_{i=1}^M$ and sub-bands $\tilde{B}_i \subset B_i$ such that the area of the complement $B_i \backslash \tilde{B}_i$ goes to zero as $h \to 0$, and such that the starting position $\mathbf{x} \in \tilde{B}_i$ guarantees $E[\mathbf{X}_\tau] \in U_i$ and $\hat{\mathbf{x}} \in U_i$. Each band B_i is associated with the interval $(x_i, x_{i+1}] \times \{0\}$ (see Figure 6.4 for a reminder of what x_i is) and is equal to its inverse image under the transport map $\Pi_{\theta_r^*} : D \to (0,1] \times \{0\}$ given by (6.3.3). Within each band B_i we define a sub-band $\tilde{B}_i$ given by the inverse image of $(x_i^+, x_{i+1}^-] \times \{0\}$ under Π_{θ_r}, where $x_i^+ := x_i + (L+r)h$ and $x_{i+1}^- := x_{i+1} - (L+r)h$. Note that the width of $\tilde{B}_i$ is dependent on h, even though our choice of notation does not make this obvious. As usual, we denote by $B_{i,h}$, $\tilde{B}_{i,h}$ the intersection of these bands with D_h. The set $\tilde{B}_{i,h}$ is significant because we know $\hat{\mathbf{x}}, E[\mathbf{X}_\tau] \in U_i$ if $\mathbf{x} \in \tilde{B}_{i,h}$. This situation is illustrated in Figure 6.8.

Stage 4. In stage two we gave a bound for $|E[u_0(\mathbf{X}_\tau) - u_0(\hat{\mathbf{x}})]|$, valid when $E[\mathbf{X}_\tau]$ and $\hat{\mathbf{x}}$ each belong to one of the well behaved sets U_i, that depends on $\|E[\mathbf{X}_\tau] - \hat{\mathbf{x}}\|$ and $\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^{p'}}$ for $p' \ge 4$, as well as the probability of the event E_s^c, a "rogue" event where even though $E[\mathbf{X}_\tau]$ and $\hat{\mathbf{x}}$ each belong to U_i, the stopped random walk $\mathbf{X}_\tau$ nevertheless lands outside U_i. In stage three we broke D into a series of bands $\{B_i\}_{i=1}^M$ and sub-bands $\tilde{B}_i \subset B_i$ such that the starting position $\mathbf{x} \in \tilde{B}_i$ guarantees $E[\mathbf{X}_\tau] \in U_i$ and

$\hat{\mathbf{x}} \in U_i$. We already have bounds on $\|E[\mathbf{X}_\tau] - \hat{\mathbf{x}}\|$ and $\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4}$ from stage one. What remains is to bound $P(E_s^c)$. We will accomplish this in two steps using elementary arguments based on martingales and Azuma's inequality. In step one, we show that τ is bounded above by a constant (dependent on h) with a high probability. In step two, we show that so long as τ is smaller than this constant, then $\mathbf{X}_\tau$ is unlikely to drift very far from $E[\mathbf{X}_\tau]$. We then put these facts to together to bound $P(E_s^c)$.

Step 1. $P\left(\tau > \lceil \frac{2}{|\mu_y|h} \rceil\right) \leq \exp\left(-\frac{1}{4\frac{r}{|\mu_y|}rh}\right)$.

Proof. We define

$$M_k := Y_{k\wedge\tau} - (\tau \wedge k)\mu_y h - mh$$

which the reader may verify is a zero mean martingale with bounded increments

$$|M_{k+1} - M_k| \leq rh.$$

Next we note that for any k the following events are equal:

$$\{\tau \geq k\} = \{Y_{\tau\wedge k} \geq 0\} = \{M_k \geq -(k\wedge\tau)\mu_y h - mh\} = \{M_k \geq -k\mu_y h - mh\}.$$

Therefore

$$\left\{\tau \geq \left\lceil \frac{2}{|\mu_y|h} \right\rceil\right\} = \left\{M_{\lceil \frac{2}{|\mu_y|h} \rceil} \geq -\left\lceil \frac{2}{|\mu_y|h} \right\rceil \mu_y h - mh\right\} \subseteq \left\{M_{\lceil \frac{2}{|\mu_y|h} \rceil} \geq 1\right\}$$

where we have used the inequality

$$-\left\lceil \frac{2}{|\mu_y|h} \right\rceil \mu_y h - mh = \left\lceil \frac{2}{|\mu_y|h} \right\rceil |\mu_y|h - mh \geq 2 - mh \geq 1.$$

Noting that $M_0 = 0$ we apply Azuma's inequality to find

$$\begin{aligned} P\left(\tau > \left\lceil \frac{2}{|\mu_y|h} \right\rceil\right) &\leq P\left(M_{\lceil \frac{2}{|\mu_y|h} \rceil} \geq 1\right) \\ &\leq \exp\left(-\frac{1}{2\left\lceil \frac{2}{|\mu_y|h} \right\rceil r^2h^2}\right) \leq \exp\left(-\frac{1}{4\frac{r}{|\mu_y|}rh}\right). \end{aligned}$$

□

Step 2. Let $\Delta(\mathbf{x})$ denote the orthogonal distance from $\mathbf{x}$ to $\partial\tilde{B}_i$ (see Figure 6.8). Then

$$P(E_s^c) \leq 3\exp\left(-\frac{\Delta(\mathbf{x})^2}{4\frac{r}{|\mu_y|}rh}\right).$$

Proof. This time we define

$$M_k := (\mathbf{X}_{\tau\wedge k} - \mathbf{x}) \cdot (-\sin\theta_r^*, \cos\theta_r^*).$$

Once again, M_k is a zero-mean martingale with bounded increments
$|M_{k+1} - M_k| \leq rh$ obeying $M_0 = 0$. Moreover, if $\tau \leq k$, then $|M_k|$ has a geometric interpretation as the orthogonal distance $\delta(\mathbf{X}_\tau)$ from $\mathbf{X}_\tau$ to the line passing through the points $\mathbf{x}$ and $\hat{\mathbf{x}}$. In addition, we clearly have $\mathbf{X}_\tau \notin U_i$ only if $\delta(\mathbf{X}_\tau) > \Delta(\mathbf{x})$, where $\Delta(\mathbf{x})$ denotes the orthogonal distance from $\mathbf{x}$ to $\partial\tilde{B}_i$. See Figure 6.8 for an illustration. Hence we have the containment

$$E_s^c \cap \{\tau \leq k\} \subseteq \{|M_k| > \Delta(\mathbf{x})\}.$$

Applying Azuma's inequality again gives, for any $k \in \mathbb{N}$,

$$P(|M_k| > \Delta(\mathbf{x})) \le 2\exp\left(-\frac{\Delta(\mathbf{x})^2}{2kr^2h^2}\right).$$

At the same time, for any integer $k \in \mathbb{N}$ we have

$$P(E_s^c) \le P(E_s^c \cap \{\tau \le k\}) + P(E_s^c \cap \{\tau > k\}) \le P(|M_k| > \Delta(\mathbf{x})) + P(\tau > k).$$

Taking $k = \lceil \frac{2}{|\mu_y|h} \rceil$, substituting the above bound as well as the one from Step 1 gives

$$P(E_s^c) \le 2\exp\left(-\frac{\Delta(\mathbf{x})^2}{4\frac{r}{|\mu_y|}rh}\right) + \exp\left(-\frac{1}{4\frac{r}{|\mu_y|}rh}\right) \le 3\exp\left(-\frac{\Delta(\mathbf{x})^2}{4\frac{r}{|\mu_y|}rh}\right),$$

since $\Delta(\mathbf{x}) \le 1$ (this follows trivially from the Pythagorean theorem - see Figure 6.8). □

Stage 5. We are now ready to prove our main result, which we do assuming $p < \infty$ (the case $p = \infty$ is treated as a limit). The idea is to split $\|u - u_h\|_p$, which is defined as a sum over D_h, into separate sums over the bands $\{B_{i,h}\}$ defined in stage three. Within in each band we split further into a sum over $\tilde{B}_{i,h}$ and $B_{i,h}\backslash\tilde{B}_{i,h}$, that is

$$\begin{aligned} \|u - u_h\|_p &= \|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]\|_p \\ &\le \sum_{i=1}^{M} \Big\{ \|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{B_{i,h}\backslash\tilde{B}_{i,h}}\|_p \\ &+ \|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{\tilde{B}_{i,h}}\|_p \Big\}. \end{aligned} \tag{6.4.8}$$

This stage is itself divided into three steps, where first to derive a bound for $\|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{B_{i,h}\backslash\tilde{B}_{i,h}}\|_p$, then derive one for $\|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{\tilde{B}_{i,h}}\|_p$, and then put the bounds together. Step one is by far the hardest.

Step 1: A bound for $\|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{B_{i,h}\backslash\tilde{B}_{i,h}}\|_p$.

On each $\tilde{B}_{i,h}$, we can apply our estimate from statement two of Lemma 6.4.3 from Stage 2.

$$\begin{aligned} \|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{\tilde{B}_{i,h}}\|_p &\le C_{u_0,\mathcal{U},\{U_i\}} \Big\| \Big(\{\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4}\}^{s\wedge 2} \\ &+ 2\{\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4}\}^{\frac{s'\wedge 2}{4}} P(E_s^c)^{\frac{1}{2}} \\ &+ \|E[\mathbf{X}_\tau] - \hat{\mathbf{x}}\|^{s\wedge 1} \Big) 1_{\tilde{B}_{i,h}} \Big\|_p \end{aligned}$$

Substituting in our estimates of $\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4}$ and $\|E[\mathbf{X}_\tau] - \hat{\mathbf{x}}\|$ from Stage 1 gives

$$\|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{\tilde{B}_{i,h}}\|_p \le C_{u_0,\mathcal{U},\{U_i\}}(C+1)(rh)^{\frac{s}{2}\wedge 1} + 2(rh)^{\frac{s'}{2}\wedge 1}\|P(E_s^c)1_{\tilde{B}_{i,h}}\|_p,$$

where we have used $rh < 1$ to bound $(rh)^{s\wedge 1} \le (rh)^{\frac{s}{2}\wedge 1}$. Our next job is to bound $\|P(E_s^c)1_{\tilde{B}_{i,h}}\|_p$, which we do by applying our bound from stage four and then making an argument based on numerical quadrature to replace the sum in $\|\cdot\|_p$ with an integral that is easier to evaluate. Specifically, let us define

$$f(\mathbf{x}) = \exp\left(-\frac{p\Delta(\mathbf{x})^2}{4\frac{r}{|\mu_y|}rh}\right)$$

for convenience (f is one third of the bound on $P(E_s^c)$ we derived in stage 4). Then

$$\begin{aligned}\|P(E_s^c)1_{\tilde{B}_{i,h}}\|_p &\leq 3\|f(\mathbf{x})1_{\tilde{B}_{i,h}}\|_p \\ &\leq 3\left(\int_{\tilde{B}_i} f(\mathbf{x})d\mathbf{x} + \max_{\mathbf{x}\in\tilde{B}_i}\|\nabla f(\mathbf{x})\|h + \max_{\mathbf{x}\in\tilde{B}_i}|f(\mathbf{x})|h\right)^{\frac{1}{p}}.\end{aligned}$$

This is because $\|f(\mathbf{x})1_{\tilde{B}_{i,h}}\|_p^p$ can be interpreted as a Riemann sum for $\int_{\tilde{B}_i} f(\mathbf{x})d\mathbf{x}$. The two error terms on the left come respectively from the error in the rectangle rule and error in the approximation of our non-rectangular domain by squares. One may readily show $|f(\mathbf{x})| \leq 1$ while

$$\|\nabla f(\mathbf{x})\|h \leq \|\nabla f(\mathbf{x})\|rh \leq \frac{e^{-\frac{1}{2}}p^{\frac{1}{2}}}{\sqrt{2\frac{r}{|\mu_y|}}}\sqrt{rh}.$$

At the same time, changing coordinates to (u, v) with u parallel to $\partial\tilde{B}_i$ and v perpendicular one easily obtains

$$\int_{\tilde{B}_i} f(\mathbf{x})d\mathbf{x} \leq |\csc(\theta_r^*)| \int_{-\infty}^{\infty} 2\exp\left(-\frac{pv^2}{4\frac{r}{|\mu_y|}rh}\right)dv = 4|\csc(\theta_r^*)|\sqrt{\frac{\pi r}{p|\mu_y|}}(rh)^{\frac{1}{2}},$$

where the factor of $|\csc(\theta_r^*)|$ has appeared because this is the length of each side of $\partial\tilde{B}_i$ (see Figure 6.8). Putting it together gives

$$\|P(E_s^c)1_{\tilde{B}_{i,h}}\|_p \leq A_p(rh)^{\frac{1}{2p}},$$

where A_p is given by

$$\begin{aligned}A_p &= 3\left(1 + \frac{e^{-\frac{1}{2}}p^{\frac{1}{2}}}{\sqrt{2\frac{r}{|\mu_y|}}} + 2|\csc(\theta_r^*)|\sqrt{\frac{\pi r}{p|\mu_y|}}\right)^{\frac{1}{p}} \\ &\leq 3\left(1 + \frac{e^{-\frac{1}{2}}}{\sqrt{2\frac{r}{|\mu_y|}}} + 2|\csc(\theta_r^*)|\sqrt{\frac{\pi r}{|\mu_y|}}\right)e^{\frac{1}{2e}} := A,\end{aligned}$$

and where we have used $1 \leq p$ to pull out a common factor of $p^{\frac{1}{2}}$, applied the bound $p^{\frac{1}{2p}} \leq e^{\frac{1}{2e}}$, and noticed that what is left over is maximized at $p = 1$. Thus we have

$$\|P(E_s^c)1_{\tilde{B}_{i,h}}\|_p \leq A(rh)^{\frac{1}{2p}},$$

where A is a constant depending only on $\frac{r}{|\mu_y|}$ and θ_r^*. Finally, we obtain the bound

$$\begin{aligned}\|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{\tilde{B}_{i,h}}\|_p &\leq C_{u_0,\mathcal{U},\{U_i\}}(C+1)(rh)^{\frac{s}{2}\wedge 1} + 2A(rh)^{\left(\frac{s'}{2}+\frac{1}{2p}\right)\wedge\left(1+\frac{1}{2p}\right)} \\ &\leq \max(C_{u_0,\mathcal{U},\{U_i\}}(C+1), 2A)(rh)^{\frac{s}{2}\wedge\left(\frac{s'}{2}+\frac{1}{2p}\right)\wedge 1}. \qquad (6.4.9)\end{aligned}$$

Step 2: A bound for $\|E[u(\mathbf{X}_\tau) - u(\hat{\mathbf{x}})]1_{\tilde{B}_{i,h}}\|_p$.

This step is easier. Applying statement one of Lemma 6.4.3 from Stage 2 this time, and substituting in our results from stage one as before gives

$$\begin{aligned}\|E[u(\mathbf{X}_\tau)-u(\hat{\mathbf{x}})]1_{B_{i,h}\backslash\tilde{B}_{i,h}}\|_p &\le C_{u_0,\mathcal{U}}\Big\|\Big(\{\|\mathbf{X}_\tau - E[\mathbf{X}_\tau]\|_{L^4}\}^{s'\wedge 2}\\ &+ \|E[\mathbf{X}_\tau]-\hat{\mathbf{x}}\|^{s'\wedge 1}\Big)1_{B_{i,h}\backslash\tilde{B}_{i,h}}\Big\|_p\\ &\le C_{u_0,\mathcal{U}}(C+1)(rh)^{\frac{s'}{2}\wedge 1}\|1_{B_{i,h}\backslash\tilde{B}_{i,h}}\|_p\end{aligned}$$

It remains to bound $\|1_{B_{i,h}\backslash\tilde{B}_{i,h}}\|_p$. But

$$\|1_{B_{i,h}\backslash\tilde{B}_{i,h}}\|_p = (|B_{i,h}\backslash\tilde{B}_{i,h}|h^2)^{\frac{1}{p}} = (2(L+r)Nh^2)^{\frac{1}{p}} \le 2(L+1)(rh)^{\frac{1}{p}} \le 2(L+1)(rh)^{\frac{1}{2p}},$$

hence

$$\|E[u(\mathbf{X}_\tau)-u(\hat{\mathbf{x}})]1_{B_{i,h}\backslash\tilde{B}_{i,h}}\|_p \le 2C_{u_0,\mathcal{U}}(C+1)(L+1)(rh)^{\frac{s'}{2}+\frac{1}{2p}}. \tag{6.4.10}$$

Step 3: Putting the bounds together.

Combining (6.4.8), (6.4.9), and (6.4.10), we at last obtain

$$\|u-u_h\|_p \le K^*(rh)^{\frac{s}{2}\wedge\left(\frac{s'}{2}+\frac{1}{2p}\right)\wedge 1}, \tag{6.4.11}$$

where K^* is the constant

$$K^* = \max(C_{u_0,\mathcal{U},\{U_i\}}(C+1), 2A, 2C_{u_0,\mathcal{U}}(C+1)(L+1))$$

which depends only on u_0, $\mathcal{U}$, $\{U_i\}$, $r/|\mu_y|$, and θ_r^* as claimed. Moreover, K is a monotonically increasing function of $\frac{r}{|\mu_y|}$ and $K^* \to \infty$ as $\theta_r^* \to 0$ or $|\mu_y| \to 0$ as claimed.

Stage 6. The previous five stages all assumed that $a_r^* \subseteq b_r^-$ or $a_r^* \subseteq b_r^0$ so that we could exploit the connection between Algorithm 1 and stopped random walks. Now all that remains is to generalize to arbitrary stencils a_r^* possibly containing ghost pixels. This follows easily from the idea of equivalent weights from Section 5.1.

Now assume a_r^* is arbitrary. By (6.1.7) we have $\mathrm{Supp}(a_r^*) \subseteq b_{r+2}^0$, we know that (6.4.11) holds with r replaced by $r+2$, where u is the solution to the transport equation (6.3.1) with transport direction $\mathbf{g}_r^*$ given in terms of $\mathrm{Supp}(a_r^*)$ and the equivalent weights $\tilde{w}_r$. But we have already noted in Definition 6.1.2 that it doesn't matter whether the center of mass is calculated in terms of the original weights w_r and stencil a_r^*, or the equivalent weights $\tilde{w}_r$ and modified stencil $\mathrm{Supp}(a_r^*)$. Hence we have

$$\mathbf{g}_r^* = \frac{\sum_{\mathbf{y}\in\mathrm{Supp}(a_r^*)}\tilde{w}_r(\mathbf{0},\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y}\in\mathrm{Supp}(a_r^*)}\tilde{w}_r(\mathbf{0},\mathbf{y})} = \frac{\sum_{\mathbf{y}\in a_r^*} w_r(\mathbf{0},\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y}\in a_r^*} w_r(\mathbf{0},\mathbf{y})}$$

as claimed.

We are now almost done. All that remains is some tidying up of the constant in (6.4.11), which we are forced to do since we had to replace r with $r+2$. Specifically, we know

$$\|u-u_h\|_p \le K\left(\frac{r+2}{|\mu_y|}\right)^*((r+2)h)^{\frac{s}{2}\wedge\left(\frac{s'}{2}+\frac{1}{2p}\right)\wedge 1}$$

where we have written down the dependence of K^* on $\frac{r}{|\mu_y|}$ explicitly. Since K^* increases monotonically with $\frac{r}{|\mu_y|}$, and since $(\cdot)^{\left(\frac{s'}{2}+\frac{1}{2p}\right)\wedge 1}$ is itself a monotonically increasing function, we can replace all occurrences of $r+2$ with $3r$. This gives

$$\|u-u_h\|_p \leq 3K\left(3\frac{r}{|\mu_y|}\right)^{*}(rh)^{\frac{s}{2}\wedge\left(\frac{s'}{2}+\frac{1}{2p}\right)\wedge 1}.$$

We certainly have $K\left(\frac{r}{|\mu_y|}\right)^{*} \leq 3K\left(3\frac{r}{|\mu_y|}\right)^{*}$, so we now define

$$K = 3K\left(3\frac{r}{|\mu_y|}\right)^{*}$$

to obtain our final constant that works for both ghost pixels and real pixels. □

6.5 Convergence to März's limit

In the previous section we proved convergence of the inpainted function u_h to the fixed ratio continuum limit u given by the weak solution (6.3.3) to the transport problem (6.3.1), as $h \to 0$ with $r = \epsilon/h$ fixed. In [12], März and Bornemann also considered convergence of Algorithm 1 (which they called "the generic single pass algorithm") to a continuum limit, under a high resolution and vanishing viscosity double limit where first $h \to 0$ and then $\epsilon = rh \to 0$. Their limit, which we refer to here as $u_{\text{märz}}$, is also expressed as the solution to a transport equation, but with different coefficients. As already stated in Section 6.3, if the additional condition (6.3.6) is satisfied, then we can draw a connection between these limits. We have also already stated that coherence transport, Guidefill, and semi-implicit Guidefill all satisfy (6.3.6) and converge to the same limit $u_{\text{märz}}$, in this case with transport direction given by

$$\mathbf{g}^* = \frac{\int_{B_1^-(\mathbf{0})} w_1(\mathbf{0},\mathbf{y})\mathbf{y}d\mathbf{y}}{\int_{B_1^-(\mathbf{0})} w_1(\mathbf{0},\mathbf{y})d\mathbf{y}}$$

where as before $B_1^-(\mathbf{0})$ is the unit disk intersected with the lower half plane and $w_1(\mathbf{0},\mathbf{y})$ are the weights (2.5.2) used by all three methods with $\epsilon = 1$. Note that $\mathbf{g}^*$ depends implicitly on the parameter μ in the weights (2.5.2). Let us make this explicit for a moment by writing $\mathbf{g}^*_\mu$ in place of $\mathbf{g}^*$. Then, if $\mathbf{g} = (\cos\theta, \sin\theta)$ with $\theta \in [0,\pi)$ denotes the guidance direction, we have from [12, pg. 14, equation (14)]

$$\lim_{\mu\to\infty} \theta(\mathbf{g}^*_\mu) = \begin{cases} \theta & \text{if } \theta \neq 0 \\ \frac{\pi}{2} & \text{if } \theta = 0 \end{cases} \tag{6.5.1}$$

where $\theta(\mathbf{g}^*_\mu)$ denotes as usual the counterclockwise angle that the line
$L_{\mathbf{g}^*_\mu} := \{\lambda\mathbf{g}^*_\mu : \lambda \in \mathbb{R}\}$ makes with the x-axis. In other words, the continuum limit from [12] predicts no kinking unless $\mathbf{g}$ is exactly parallel to ∂D_h, and moreover, predicts that the three methods listed above exhibit exactly the same behaviour as $\mu \to \infty$. This is not what is observed in practice and we will draw very different conclusions in Section 6.6.2. We have three objectives:

1. To rigorously establish convergence of u_h to $u_{\text{märz}}$, which was treated as a "formal limit" in [12], at least under the simplifying assumptions of Section 6.1. In fact, although $u_{\text{märz}}$ was treated as an iterated double limit in [12], we will see that $u_h \to u_{\text{märz}}$ in the single *simultaneous* limit where $h \to 0$ and $\epsilon \to 0$ at the same time, but with a nuance. It is not in general enough that h and ϵ separately vanish - in general they do so in such a way that their ratio $r = \epsilon/h$ diverges.

2. To show that for $r \gg 1$, the two limits u and $u_{\text{märz}}$ are similar, i.e. $u \to u_{\text{märz}}$.

3. To argue that when $r \in \mathbb{N}$ is small (say close to $r = 5$, the recommended default value in [12]), our limit u typically does a better job of capturing the behaviour of u_h than $u_{\text{märz}}$ does, i.e. $\|u_h - u\|_p \ll \|u_h - u_{\text{märz}}\|_p$.

The following Theorem accomplishes objectives 1-2. Objective 3 is discussed in Remark 6.5.2 and supported numerically in Section 6.8.

Theorem 6.5.1. *Suppose $\mathbf{g}^* = \lim_{r\to\infty} \frac{\mathbf{g}_r^*}{r}$ exists and converges with rate at least $O(r^{-q})$ as in (6.3.6), for some $q > 0$. Let $u_{märz}(\mathbf{x}) := u_0(\Pi_{\theta^*}(\mathbf{x}))$ denote the weak solution to the transport equation (6.3.1) with $\mathbf{g}_r^*$ replaced by $\mathbf{g}^*$, and Π_{θ^*} the transport operator defined as in (6.3.3) but with $\theta_r^* = \theta(\mathbf{g}_r^*)$ replaced by $\theta^* = \theta(\mathbf{g}^*)$. Assume $h \le r^{-q}$. Then under the same conditions as in Theorem 6.3.1 we have the bound*

$$\|u_h - u_{märz}\|_p \le K_1 \cdot (rh)^{(\frac{s'}{2}+\frac{1}{2p})\wedge\frac{s}{2}\wedge 1} + K_2 r^{-q\{s\wedge(s'+\frac{1}{p})\wedge 1\}} \tag{6.5.2}$$

where $K_1 > 0$, $K_2 > 0$ are constants depending only on $\theta^* = \theta(\mathbf{g}^*)$, $\mathbf{g}^* \cdot e_2$, u_0, $\mathcal{U}$, and $\{U_i\}_{i=1}^M$ (where $\{U_i\}_{i=1}^M$ are the sets illustrated in Figure 6.4). Moreover, The dependence on θ^* and $\mathbf{g}^* \cdot e_2$ is continuous but K_1 and K_2 diverge as $\theta^* \to 0$ or $\theta^* \to \pi$. Additionally, we have

$$\|u - u_{märz}\|_p \to 0 \qquad \text{as } r \to \infty \tag{6.5.3}$$

for all $1 \le p < \infty$ if $s' = 0$ and for $p = \infty$ as well if $s' > 0$.

Proof. Appendix A.7. □

Remark 6.5.2. *Theorem 6.5.1 implies our claim in objective one, because the bound (6.5.2) implies that if $\epsilon = rh \to 0$ and $r \to \infty$, then u_h converges to $u_{märz}$ in L^p. This does not, however, mean that $r \to \infty$ is* required. *In Section 6.6.2, we will see that there are many ways for (6.5.3) to be satisfied. We will see examples where:*

- *$\|u - u_{märz}\|_p \to 0$ as $r \to \infty$, but remains strictly positive for all r.*
- *$\|u - u_{märz}\|_p > 0$ for all $r < R$ and then $u = u_{märz}$ for all $r \ge R$.*
- *$u = u_{märz}$ independent of r.*

Hence the bound (6.5.2) is not in general tight and $r \to \infty$ is a sufficient but not necessary condition. This means that our third objective, which is to argue that when r is a small integer then

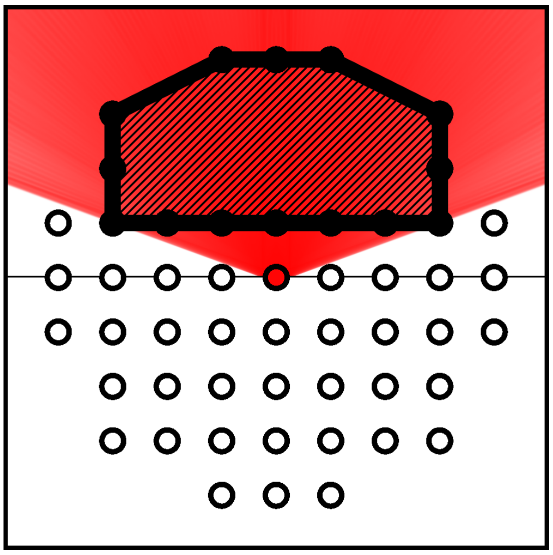
$$\|u_h - u\|_p \ll \|u_h - u_{märz}\|_p$$

is not in general true. However, numerical experiments in Section 6.8.2 provide strong numerical evidence that it does hold in many cases. Finally, objective two is clearly implied by (6.5.3).

6.6 Kinking artifacts and the continuum limit

In this section we apply our analysis, in particular Theorem 6.3.1, in order to explain:

1. The kinking artifacts listed in Section 2.2 and illustrated in Figures 2.6, 2.4.
2. The differences between the direct form of Algorithm 1 and its semi-implicit extension in terms of coping with these artifacts.


(a) Repeat of the experiment in Figure 2.6 for Guidefill with $r = 3$, with $\text{Conv}(-\bar{b}_r^-)$ superimposed.

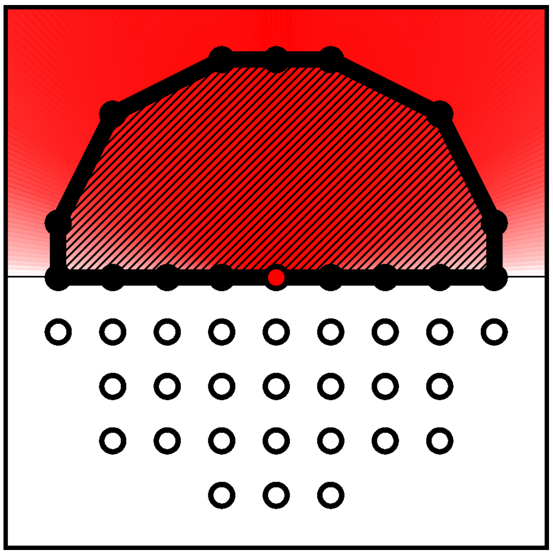

(b) Repeat of the experiment in Figure 2.6 for semi-implicit Guidefill with $r = 3$, with $\text{Conv}(-\bar{b}_r^0)$ superimposed.

Figure 6.9: **Convex hulls and transport directions:** An immediate corollary of Theorem 6.3.1 is that the limiting transport direction $\mathbf{g}_r^*$ lies in the convex hull of $\bar{b}_r^-$ for the direct form of Algorithm 1, and the convex hull of $\bar{b}_r^0$ for the semi-implicit extension. Since $\text{Conv}(\bar{b}_r^-)$ contains only a cone of directions while $\text{Conv}(\bar{b}_r^0)$ contains the full arc from 0 to π, this explains why Guidefill (and more generally *all* direct methods of the general form given by Algorithm 1) fails for shallow angles, while this is not true of the semi-implicit extension. To illustrate this, we have repeated the experiment from Figure 2.6 using Guidefill (a) and semi-implicit Guidefill (b), while superimposing the sets $\text{Conv}(-\bar{b}_r^-)$ and $\text{Conv}(-\bar{b}_r^0)$ respectively (we have negated the sets for convenience which we can do since $\mathbf{g}_r^*$ and $-\mathbf{g}_r^*$ define the same transport equation). Note that in the case of semi-implicit Guidefill, the lines appear to be getting fainter as $\theta(\mathbf{g}_r^*) \to 0$ and $\theta(\mathbf{g}_r^*) \to \pi$. This likely has two causes. For one, the angular footprint of the red dot in Figure 2.6(a), i.e. the width of the dot multiplied by $\sin\theta_r^*$ (which tells you how much of the dot is "visible" from direction θ_r^*) is going to zero. Secondly, we will see in Section 6.7 (see in particular Figure 6.16) that semi-implicit Guidefill has blur artifacts that become arbitrarily bad as $\theta(\mathbf{g}_r^*) \to 0$ or $\theta(\mathbf{g}_r^*) \to \pi$. Nevertheless, Figure 5.2 demonstrates that semi-implict Guidefill can successfully extrapolate lines with θ very close to 0 without serious issues of faintness. All parameters are the same as in Figure 2.6.

First we prove a fundamental distinction between the direct form of Algorithm 1 and the semi-implicit extension. Then we go on to look at the limiting transport directions associated with three specific methods: coherence transport, Guidefill, and semi-implicit Guidefill.

6.6.1 The direct form Algorithm 1 kinks, the semi-implicit extension need not

An immediate corollary of Theorem 6.3.1 is that the direct form of Algorithm 1 is limited in terms of what directions it can extrapolate along, while this is not true of the semi-implicit extension. This follows from the geometric interpretation of the limiting transport direction $\mathbf{g}_r^*$ as the center of mass of the stencil a_r^* with the respect to the stencil weights $\{w_r(\mathbf{0},\mathbf{y})/W : \mathbf{y} \in a_r^*\}$. Specifically, since the original weights $\{\frac{w_r(\mathbf{0},\mathbf{y})}{W}\}_{\mathbf{y}\in a_r^*}$ are non-negative and sum to one, by the preservation of total mass (5.1.2) and inheritance of non-negativity (5.1.4) properties of equivalent weights, the same is true of the equivalent weights $\{\frac{\tilde{w}_r(\mathbf{0},\mathbf{y})}{W}\}_{\mathbf{y}\in\text{Supp}(a_r^*)}$. Hence

$$\mathbf{g}_r^* = \sum_{\mathbf{y}\in\text{Supp}(a_r^*)} \frac{\tilde{w}_r(\mathbf{0},\mathbf{y})}{W}\mathbf{y} \in \text{Conv}(\text{Supp}(a_r^*)),$$

where Conv(Supp(a_r^*)) denotes the convex hull of Supp(a_r^*). But by (6.1.7) we have

$$\text{Supp}(a_r^*) \subseteq \begin{cases} \bar{b}_r^- & \text{if we use the direct form of Algorithm 1} \\ \bar{b}_r^0 & \text{if we use the semi-implicit extension} \end{cases} \tag{6.6.1}$$

where $\bar{b}_r^0$ and $\bar{b}_r^-$ are the dilated sets defined by (6.1.5) and (6.1.6) respectively. It follows that $\mathbf{g}_r^*$ has to lie in the convex hull of $\bar{b}_r^-$ if the direct form of Algorithm 1 is used, or the convex hull of $\bar{b}_r^0$ if the semi-implicit form is used instead. That is

$$\mathbf{g}_r^* \in \begin{cases} \text{Conv}(\bar{b}_r^-) & \text{if we use the direct form of Algorithm 1.} \\ \text{Conv}(\bar{b}_r^0) & \text{if we the semi-implicit extension.} \end{cases} \tag{6.6.2}$$

This in turn immediately implies that if we use the direct form of Algorithm 1, then $\theta(\mathbf{g}_r^*)$ is restricted to the cone

$$\theta_c \le \theta(\mathbf{g}_r^*) \le \pi - \theta_c \tag{6.6.3}$$

where

$$\theta_c = \arcsin\left(\frac{1}{r}\right) \tag{6.6.4}$$

is the smallest possible angle one can make given the restriction (6.6.2), while for the semi-implict method we have

$$0 < \theta(\mathbf{g}_r^*) < \pi \tag{6.6.5}$$

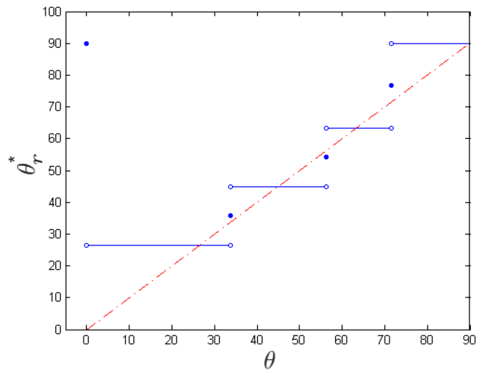
where the angles $\theta = 0$ and $\theta = \pi$ are omitted not because of the restriction (6.6.2), but because Theorem 6.3.1 does not apply in either case (indeed, the continuum limit u given by (6.3.1) is not even defined). The cone (6.6.3) is exactly what we saw in Figure 2.6(c). At the same time, the lack of restrictions implied by (6.6.5) is consistent with our experience in Figures 5.2 and 6.3, where semi-implicit Guidefill was able to successfully extrapolate lines making an angle as small as $1°$ with the inpainting domain boundary for $r = 3$, well under the critical angle $\theta_c = \arcsin(\frac{1}{3}) \approx 19.5°$ where standard Guidefill breaks down. Figure 6.9 illustrates this result.

6.6.2 Limiting Transport Directions for Coherence Transport, Guidefill, and semi-implicit Guidefill

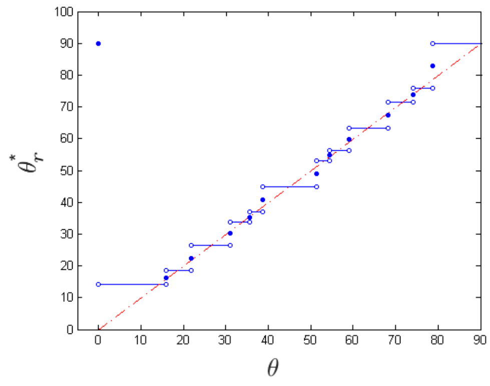
Here we derive formulas for the limiting transport directions of coherence transport, Guidefill, and semi-implicit Guidefill in the limit as $\mu \to \infty$. For convenience, in this section we rescale the limiting transport direction $\mathbf{g}_r^*$ by a factor of $W = \sum_{\mathbf{y}\in a_r^*} w_r(\mathbf{0},\mathbf{y})$, giving

$$\mathbf{g}_r^* = \sum_{\mathbf{y}\in a_r^*} w_r(\mathbf{0},\mathbf{y})\mathbf{y}. \tag{6.6.6}$$

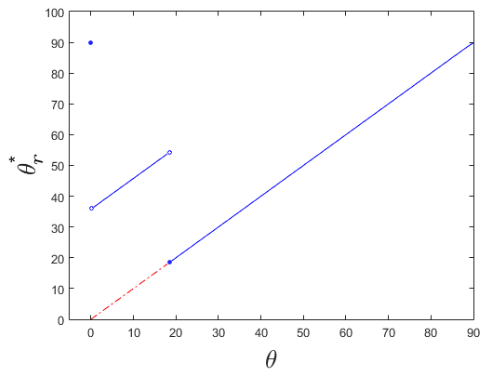
This is more convenient to work with and defines the same transport equation. In fact, the ability to rescale $\mathbf{g}_r^*$ by any $\lambda \neq 0$ without changing the underlying transport equation is a tool that we will use repeatedly in our arguments throughout this section. To make the dependence of the transport direction



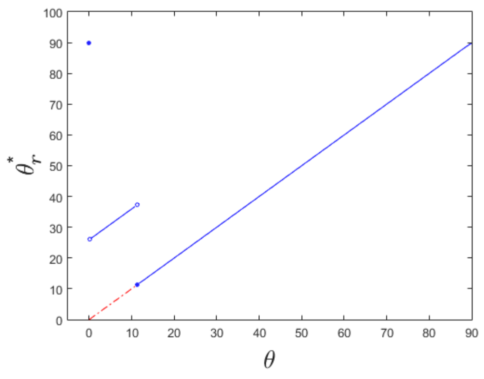
(a) Coherence transport, $r = 3$.



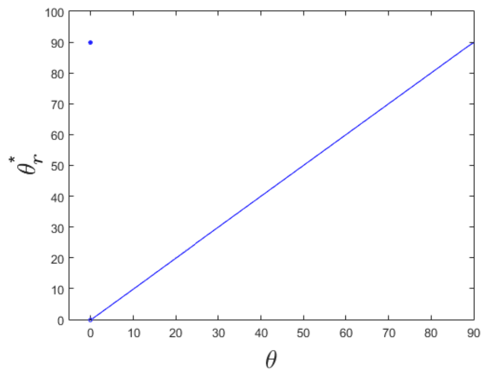
(b) Coherence transport, $r = 5$.



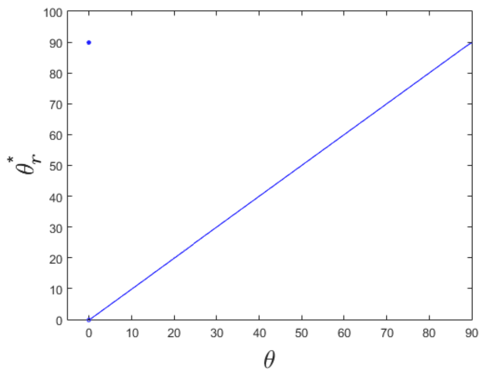
(c) Guidefill, $r = 3$.



(d) Guidefill, $r = 5$.



(e) Semi-implicit Guidefill $r = 3$.



(f) Semi-implicit Guidefill $r = 5$.

Figure 6.10: **Limiting transport direction $\theta^*_r = \theta(\mathbf{g}^*_r)$ as a function of the guideance direction $\theta = \theta(\mathbf{g})$ for the three main methods:** The theoretical limiting curves $\theta^*_r = F(\theta)$ obtained for coherence transport (a)-(b), Guidefill (c)-(d), and semi-implicit Guidefill (e)-(f). The desired curve $F(\theta) = \theta$ is highlighted in red. Coherence transport exhibits a staircase pattern with $F(\theta) \neq \theta$ for all but finitely many θ, Guidefill obeys $F(\theta) = \theta$ for all $\theta \in (\theta_c, \pi - \theta_c)$ where θ_c is the critical angle (6.6.4), and semi-implicit Guidefill obeys $F(\theta) = \theta$ for all $\theta \neq 0$.

on μ explicit, within this section we write this direction as $\mathbf{g}^*_{r,\mu}$, and define

$$\mathbf{g}^*_r := \lim_{\mu\to\infty} \mathbf{g}^*_{r,\mu}.$$

In order to resolve the ambiguity that $\mathbf{g}^*_r$ and $-\mathbf{g}^*_r$ define the same transport equation, we will frequently be taking angles modulo π. This is reflected in the definition we have selected for $\theta(\mathbf{v})$ - recall that this refers to the angle that the line $L_{\mathbf{v}} = \{\lambda\mathbf{v} : \lambda \in \mathbb{R}\}$ makes with the x-axis, and hence always lies in the range $[0, \pi)$. We define $\mathbf{g} = (\cos\theta, \sin\theta)$, $\theta^*_{r,\mu} := \theta(\mathbf{g}^*_{r,\mu})$, $\theta^*_r := \theta(\mathbf{g}^*_r)$ and consider the function $\theta^*_r = F(\theta)$. Our main concern is to determine when $\theta^*_r = \theta$ (no kinking) and when $\theta^*_r \neq \theta$ (kinking). Results are illustrated in Figure 6.10.

Coherence Transport. We have already stated in Section **??** and illustrated in Figure 2.6 that coherence transport in the limit $\mu \to \infty$ kinks unless $\mathbf{g} = \lambda\mathbf{v}$ for some $\mathbf{v} \in B_{\epsilon,h}(\mathbf{0})$ and some $\lambda \in \mathbb{R}$. Under the assumptions of Section 6.1, a more precise statement is that coherence transport in the limit $\mu \to \infty$ fails to kink if and only if $\mathbf{g} = \lambda\mathbf{v}$ for some $\mathbf{v} \in b^-_r$ and $\lambda \in \mathbb{R}$. Before we prove this, let us make some definitions. We define the *angular spectrum* of b^-_r as

$$\Theta(b^-_r) := \{\theta_1, \theta_2, \ldots, \theta_n : \text{ each } \theta_i = \theta(\mathbf{y}) \text{ for some } \mathbf{y} \in b^-_r\}. \tag{6.6.7}$$

In other words, $\Theta(b^-_r)$ is the collection of angles modulo π that are representable using members of b^-_r (or which elements of the projective space $\mathbb{RP}^1$ are representable, to be more mathematically precise), see Figure 6.11 for an illustration. The angular spectrum may be similarly defined in the obvious way for more general sets, and we do so in Appendix A.8. We will show that for coherence transport, when $0 < \theta < \pi$, we either have

$$\theta^*_r \in \Theta(b^-_r) \quad \text{or} \quad \theta^*_r = \frac{\theta_i + \theta_{i+1}}{2} \quad \text{for two consecutive } \theta_i, \theta_{i+1} \in \Theta(b^-_r).$$

To begin, note that in this case (6.6.6) becomes

$$\mathbf{g}^*_{r,\mu} := \sum_{\mathbf{y}\in b^-_r} e^{-\frac{\mu^2}{2r^2}(\mathbf{y}\cdot\mathbf{g}^\perp)^2} \frac{\mathbf{y}}{\|\mathbf{y}\|},$$

where b^-_r is the discrete half ball (6.1.4). Denote by Ψ the set of minimizers of $|\mathbf{y}\cdot\mathbf{g}^\perp|$ for $\mathbf{y} \in b^-_r$, meaning that $|\mathbf{y}\cdot\mathbf{g}^\perp| := \Delta \geq 0$ for all $\mathbf{y} \in \Psi$ and $|\mathbf{y}\cdot\mathbf{g}^\perp| > \Delta$ for all $\mathbf{y} \in b^-_r\backslash\Psi$. After rescaling by $e^{\frac{\mu^2}{2r^2}\Delta^2}$, the transport direction $\mathbf{g}^*_{r,\mu}$ becomes

$$\begin{aligned}\mathbf{g}^*_{r,\mu} &= \sum_{\mathbf{y}\in\Psi} \frac{\mathbf{y}}{\|\mathbf{y}\|} + \sum_{\mathbf{j}\in b^-_r\backslash\Psi} e^{-\frac{\mu^2}{2r^2}\{(\mathbf{y}\cdot\mathbf{g}^\perp)^2-\Delta^2\}} \frac{\mathbf{y}}{\|\mathbf{y}\|} \\ &\to \sum_{\mathbf{y}\in\Psi} \frac{\mathbf{y}}{\|\mathbf{y}\|} \qquad \text{as } \mu \to \infty. \end{aligned} \tag{6.6.8}$$

Note that $|\mathbf{y}\cdot\mathbf{g}^\perp|$ represents the distance from the point $\mathbf{y}$ to the line through the origin $L_{\mathbf{g}}$. Thus computing the set Ψ is equivalent to finding the set of points in b^-_r closest to a given line through the origin. In Appendix A.8 we prove that as θ sweeps out an arc from 0 to π, for all but finitely many θ the set Ψ is a singleton, containing a sequence of lone minimizers that we enumerate (in order of occurrence) as $\mathbf{y}_1, \mathbf{y}_2, \ldots \mathbf{y}_{n'}$ (for some finite n'). Now, it turns out that $n' = n$ and moreover for every $\theta_i \in \Theta(b^-_r)$ we have

$$\theta_i = \theta(\mathbf{y}_i)$$

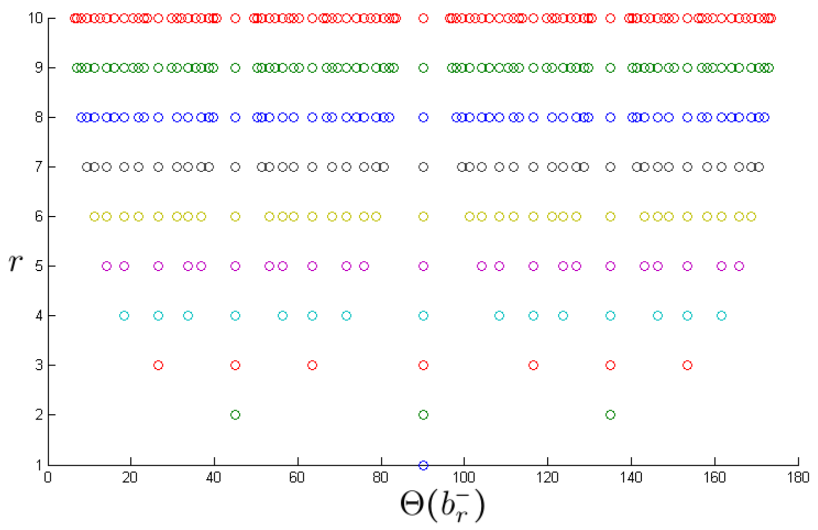
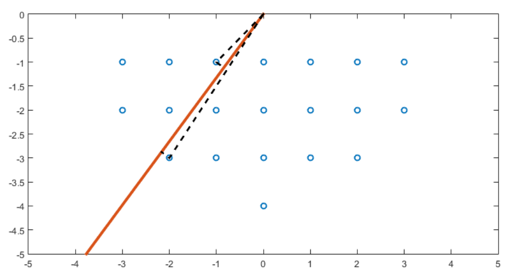


Figure 6.11: **Illustration of the angular spectrum:** The angular spectrum $\Theta(b_r^-)$ tells us which angles (modulo π), are representable using elements of b_r^-. Here we have illustrated $\Theta(b_r^-)$, measured in degrees, for $r = 1, 2, \ldots$.


(a)

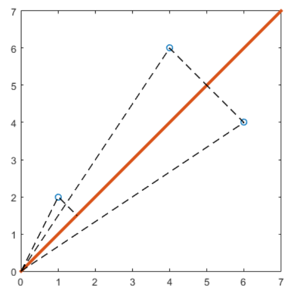

(b)

Figure 6.12: **Closest points and shallowest angles:** We claim that the closest point in the set $b_r^- \subset \mathbb{Z}^2$ to a given line $L_{\mathbf{g}}$ ($\mathbf{g} = (\cos\theta, \sin\theta)$) is always one of the two points casting the shallowest angle with $L_{\mathbf{g}}$ on either side, as illustrated in (a) for $\theta = 53°$, $r = 4$. This statement does *not* hold if b_r^- is replaced with a generic $A \subset \mathbb{Z}^2$, as demonstrated by the counterexample $A = \{(1,2),(4,6),(6,4)\}$ and $\theta = 45°$ in (b).

(Appendix A.8, Proposition A.8.3). In other words, we have a 1-1 correspondence between (singleton) minimizers of $|\mathbf{y} \cdot \mathbf{g}^\perp|$ and the angular spectrum $\Theta(b_r^-)$. Moreover, it can be shown that if $\theta_i < \theta < \theta_{i+1}$ for some $\theta_i, \theta_{i+1} \in \Theta(b_r^-)$, then either $\Psi = \{\mathbf{y}_i\}$ (for θ close to θ_i) or $\Psi = \{\mathbf{y}_{i+1}\}$ (for θ close to θ_{i+1}) or $\Psi = \{\mathbf{y}_i, \mathbf{y}_{i+1}\}$ if $\theta = \theta_{i,i+1}$, where $\theta_{i,i+1} \in (\theta_i, \theta_{i+1})$ is a critical angle given by

$$\theta_{i,i+1} = \theta(\mathbf{y}_i + \mathbf{y}_{i+1}) \quad \text{ for } 1 \le i \le n-1.$$

For convenience, we also define $\theta_{0,1} = 0$ and $\theta_{n,n+1} = \pi$. Since one can also prove $\Psi = \{\mathbf{y}_1\}$ for $0 := \theta_{0,1} < \theta < \theta_1$ and $\Psi = \{\mathbf{y}_n\}$ for $\theta_n < \theta < \theta_{n,n+1} := \pi$, with this notation we have the general result

$$\Psi = \begin{cases} \{\mathbf{y}_i\} & \text{if } \theta_i < \theta < \theta_{i,i+1} \quad \text{for some } i = 1, \ldots, n \\ \{\mathbf{y}_i, \mathbf{y}_{i+1}\} & \text{if } \theta = \theta_{i,i+1} \quad \text{for some } i = 1, \ldots, n-1 \\ \{\mathbf{y}_{i+1}\} & \text{if } \theta_{i,i+1} < \theta < \theta_{i+1} \quad \text{for some } i = 0, \ldots, n-1 \end{cases}$$

(Appendix A.8, Proposition A.8.4 - note that we have carefully excluded $\theta_{0,1} := 0$ and $\theta_{n,n+1} = \pi$ from the middle case). In words, this means that the element(s) of b_r^- closest to the line $L_{\mathbf{g}}$ are also the member(s) of b_r^- that cast the shallowest angles with $L_{\mathbf{g}}$ from above and below. This statement is not true if b_r^- is replaced with a generic subset of $\mathbb{Z}^2$ - see Figure 6.12. The remaining cases are $\theta = \theta_i \in \Theta(b_r^-)$ and $\theta = 0$. We deal with the former first - in the first case we have

$$\Psi = \{\mathbf{y} \in b_r^- : \theta(y) = \theta_i\},$$

a set containing up to r members, all of which are parallel to each other and to $\mathbf{g}$. In order to make these ideas more concrete, Example 6.6.1 gives explicit expressions for $\Theta(b_r^-)$ and Ψ in the case $r = 3$. Also, in Appendix A.8 Remark A.8.7, we give an algorithm for computing $\Theta(b_r^-)$ and $Y(b_r^-) := \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$ for any r.

Example 6.6.1. *When $r = 3$ we have*

$$\begin{aligned} \Theta(b_3^-) &:= \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7\} \\ &= \{\arctan(1/2), \frac{\pi}{4}, \arctan(2), \frac{\pi}{2}, \frac{\pi}{2} + \arctan(1/2), \frac{3\pi}{4}, \frac{\pi}{2} + \arctan(2)\}. \end{aligned}$$

For $0 < \theta \leq \frac{\pi}{2}$, (we omit $\frac{\pi}{2} < \theta < \pi$ for brevity) the set of minimizers Ψ is given by

$$\Psi = \begin{cases} \{(-2,-1)\} := \{\mathbf{y}_1\} & \textit{if } 0 < \theta < \theta_{1,2}. \\ \{(-2,-1),(-1,-1)\} := \{\mathbf{y}_1, \mathbf{y}_2\} & \textit{if } \theta = \theta_{1,2}. \\ \{(-1,-1)\} := \{\mathbf{y}_2\} & \textit{if } \theta_{1,2} < \theta < \theta_2. \\ \{(-1,-1),(-2,-2)\} := \{\mathbf{y}_2, 2\mathbf{y}_2\} & \textit{if } \theta = \theta_2. \\ \{(-1,-1)\} := \{\mathbf{y}_2\} & \textit{if } \theta_2 < \theta < \theta_{2,3}. \\ \{(-1,-1),(-1,-2)\} := \{\mathbf{y}_2, \mathbf{y}_3\} & \textit{if } \theta = \theta_{2,3}. \\ \{(-1,-2)\} = \{\mathbf{y}_3\} & \textit{if } \theta_{2,3} < \theta < \theta_{3,4}. \\ \{(-1,-2),(0,-1)\} = \{\mathbf{y}_3, \mathbf{y}_4\} & \textit{if } \theta = \theta_{3,4}. \\ \{(0,-1)\} := \{\mathbf{y}_4\} & \textit{if } \theta_{3,4} < \theta < \theta_4. \\ \{(0,-1),(0,-2),(0,-3)\} := \{\mathbf{y}_4, 2\mathbf{y}_4, 3\mathbf{y}_4\} & \textit{if } \theta = \theta_4. \end{cases}$$

where $\theta_{0,1} = 0$, $\theta_{1,2} = \arctan(2/3)$, $\theta_{2,3} = \arctan(3/2)$, $\theta_{3,4} = \arctan(3)$.

When Ψ is a singleton set, that is $\Psi = \{\mathbf{y}_i\}$ for some $1 \leq i \leq n$, (6.6.8) becomes $\mathbf{g}_r^* = \frac{\mathbf{y}_i}{\|\mathbf{y}_i\|}$ and we have

$$\theta(\mathbf{g}_r^*) = \theta\left(\frac{\mathbf{y}_i}{\|\mathbf{y}_i\|}\right) = \theta_i$$

On the other hand, if $\theta = \theta_i \in \Theta(b_r^-)$, $\mathbf{g}_r^*$ is a sum of vectors all parallel to one another and to $\mathbf{g}$, and we get $\theta_i^* = \theta_i$ again. This is the lone case in which coherence transport doesn't kink. Next, at the transition angles $\theta_{i,i+1}$ where $\Psi = \{\mathbf{y}_i, \mathbf{y}_{i+1}\}$, we have $\mathbf{g}_r^* = \frac{\mathbf{y}_i}{\|\mathbf{y}_i\|} + \frac{\mathbf{y}_{i+1}}{\|\mathbf{y}_{i+1}\|}$, so that

$$\theta(\mathbf{g}_r^*) = \theta\left(\frac{\mathbf{y}_i}{\|\mathbf{y}_i\|} + \frac{\mathbf{y}_{i+1}}{\|\mathbf{y}_{i+1}\|}\right) = \frac{\theta_i + \theta_{i+1}}{2},$$

where we have used the observation (proved in Appendix A.8, Observation A.8.5) that $\theta(\mathbf{v} + \mathbf{w}) = \frac{\theta(\mathbf{v}) + \theta(\mathbf{w})}{2}$ holds for all unit vectors $\mathbf{v}, \mathbf{w} \in S^1$. Finally, suppose $\theta = 0$. Here we have $\Psi = \{(i, -1) :$

$-r+1 \leq i \leq r-1\}$, giving

$$\mathbf{g}_r^* = e_2 \qquad \text{for } \theta = 0$$

after rescaling. In summary, for $\theta \in [0,\pi)$ we then have

$$\theta_r^* = \begin{cases} \frac{\pi}{2} & \text{if } \theta = 0 \\ \theta_i & \text{if } \theta_{i-1,i} < \theta < \theta_{i,i+1} \text{ for some } i = 1,\dots n \\ \frac{\theta_i + \theta_{i+1}}{2} & \text{if } \theta = \theta_{i,i+1} \text{ for some } i = 1,\dots n \end{cases} \tag{6.6.9}$$

See Figure 6.10(a)-(b) for an illustration of (6.6.9) for $r = 3$ and $r = 5$. In Appendix A.8, Corollary A.8.6 we prove a generalization of this result that applies if, for example, the discrete ball used by coherence transport is replaced with a discrete square.

Remark 6.6.2. *Since $\theta_r^* = \theta$ if and only if $\theta \in \Theta(b_r^-)$, and since the latter is generated by vectors with integer coordinates, it follows that for fixed $\theta = \arctan(x)$, we have $\theta_r^* = \theta$ for all r sufficiently large if x is rational and $\theta_r^* \neq \theta$ for all r if x is irrational. In light of* (6.5.1) *and Theorem 6.5.1, this means that in the former case we have $u = u_{\text{märz}}$ for all r sufficiently large, but in the latter case $\|u - u_{\text{märz}}\|_p \to 0$ for all $1 \leq p < \infty$, but we always have $u \neq u_{\text{märz}}$ for generic boundary data u_0.*

Guidefill. In this case (6.6.6) becomes

$$\mathbf{g}_{r,\mu}^* := \sum_{\mathbf{y} \in \tilde{b}_r^-} e^{-\frac{\mu^2}{2r^2}(\mathbf{y}\cdot\mathbf{g}^\perp)^2} \frac{\mathbf{y}}{\|\mathbf{y}\|},$$

where $\tilde{b}_r^-$ is given by (6.1.4). It is useful to patition $\tilde{b}_r^-$ into a disjoint union of sets ℓ_k^- such at each ℓ_k^- is the collection of points in $\tilde{b}_r^-$ distance k from the line $L_{\mathbf{g}} = \{\lambda \mathbf{g} : \lambda \in \mathbb{R}\}$, that is

$$\ell_k^- = \{n\mathbf{g} + m\mathbf{g}^\perp \in \tilde{b}_r^- : m = \pm k\}.$$

Since the weights (2.5.2) exponentially decay with distance from $L_{\mathbf{g}}$, then so long as ℓ_0^- is non-empty, we expect the contribution of the other ℓ_k^- to vanish. For Guidefill, ℓ_0^- can be explicitly parametrized as

$$\begin{aligned} \ell_0^- &:= \{n\mathbf{g} \in \tilde{b}_r^- : n\mathbf{g}\cdot e_2 \leq -1\} \\ &= \{n\mathbf{g} : n = -r,\dots,-\lceil \csc\theta \rceil\}. \end{aligned}$$

For ℓ_0^- to be non-empty, we need $\lceil \csc\theta \rceil \leq r$, which occurs only if $\theta_c \leq \theta \leq \pi - \theta_c$, where θ_c is the same critical angle (6.6.4) from Section 6.6.1. If $\ell_0^- \neq \emptyset$, we have

$$\begin{aligned} \mathbf{g}_{r,\mu}^* &= \sum_{\mathbf{y}\in\ell_0^-} \frac{\mathbf{y}}{\|\mathbf{y}\|} + \sum_{k=1}^{r} \sum_{\mathbf{y}\in\ell_k^-} e^{-\frac{\mu^2}{2r^2}k^2\|\mathbf{g}\|^2} \frac{\mathbf{y}}{\|\mathbf{y}\|} \\ &\to \sum_{\mathbf{y}\in\ell_0^-} \frac{\mathbf{y}}{\|\mathbf{y}\|} \qquad \text{as } \mu \to \infty. \\ &= \sum_{n=-r}^{-\lceil \csc\theta \rceil} n\mathbf{g} \\ &= \mathbf{g} \qquad \text{after rescaling.} \end{aligned}$$

Hence we transport in the correct direction in this case.

One the other hand, if $\ell_0^- = \emptyset$ but $\theta \neq 0$, then the weights (2.5.2) concentrate all their mass into ℓ_1^-, with all other ℓ_k^- vanishing. Unfortunately, unlike ℓ^0, the set ℓ_1^- is not parallel to $\mathbf{g}$ so we expect kinking in this case. In general ℓ_1^- can consist of two parallel components on either side of ℓ_0^-. But in this case, since ℓ_0^- lies entirely above the line $y = -1$, we know ℓ_1^- consists of just one component and we can write it explicitly as

$$\ell_1^- = \{n\mathbf{g} - \text{sgn}(\cos\theta)\mathbf{g}^\perp : n = -r+1, \dots, -1\}$$

(remember that $\mathbf{g}^\perp$ denotes the *counterclockwise* rotation of $\mathbf{g}$ by 90°). After rescaling by $e^{\frac{\mu^2}{2r^2}\|\mathbf{g}\|^2}$ we obtain

$$\begin{aligned}
\mathbf{g}^*_{r,\mu} &= \sum_{\mathbf{y}\in\ell_1^-} \frac{\mathbf{y}}{\|\mathbf{y}\|} + \sum_{k=2}^{r} \sum_{\mathbf{y}\in\ell_k^-} e^{-\frac{\mu^2}{2r^2}(k^2-1)\|\mathbf{g}\|^2} \frac{\mathbf{y}}{\|\mathbf{y}\|} \\
&\to \sum_{\mathbf{y}\in\ell_1^-} \frac{\mathbf{y}}{\|\mathbf{y}\|} \qquad \text{as } \mu \to \infty. \\
&= \left(\sum_{n=-r+1}^{-1} \frac{n}{\sqrt{1+n^2}}\right)\mathbf{g} - \text{sgn}(\cos\theta)\left(\sum_{n=-r+1}^{-1} \frac{1}{\sqrt{1+n^2}}\right)\mathbf{g}^\perp \\
&= \mathbf{g} + \text{sgn}(\cos\theta)\alpha_r\mathbf{g}^\perp \qquad \text{after rescaling.}
\end{aligned}$$

where

$$\alpha_r = \frac{\sum_{n=1}^{r-1} \frac{1}{\sqrt{1+n^2}}}{\sum_{n=1}^{r-1} \frac{n}{\sqrt{1+n^2}}}.$$

Finally, if $\theta = 0$ we have $\tilde{b}_r^- = b_r^-$ and we obtain $\mathbf{g}_r^* = e_2$ as for coherence transport. Defining $\Delta\theta_r = \arctan(\alpha_r)$, for $\theta \in [0, \pi)$ we obtain

$$\theta_r^* = \begin{cases} \frac{\pi}{2} & \text{if } \theta = 0 \\ \theta + \Delta\theta_r & \text{if } 0 < \theta < \theta_c \\ \theta & \text{if } \theta_c \le \theta \le \pi - \theta_c \\ \theta - \Delta\theta_r & \text{if } \pi - \theta_c < \theta < \pi. \end{cases} \tag{6.6.10}$$

In other words, aside from exceptional case $\theta = 0$, we have $\theta_r^* = \theta$ for the "well behaved" range of values $\theta_c \le \theta \le \pi - \theta_c$, but θ_r^* jumps by a constant angle $\Delta\theta_r$ near $\theta = 0$ and $\theta = \pi$. The first few values of $\Delta\theta_r$ are $\Delta\theta_3 \approx 35.8°$, $\Delta\theta_4 \approx 30.0°$, $\Delta\theta_5 \approx 25.9°$. See Figure 6.10(c)-(d) for an illustration of (6.6.10) for $r = 3$ and $r = 5$.

Remark 6.6.3. *Since $\theta_c = \arcsin(1/r) \to 0$ as $r \to \infty$, it follows that for each fixed $\theta \in (0, \pi)$ we have $\theta_r^* = \theta$ and hence $u = u_{\text{märz}}$ for all r sufficiently large. On the other hand, since $\theta_c > 0$, $\alpha_r > 0$ for all r, we never have $\theta_r^* = \theta$ independent of θ for any fixed r. There are always angles we can't reach.*

Semi-implicit Guidefill. The analysis of semi-implicit Guidefill is the same as for Guidefill except that the set $\tilde{b}_r^-$ is replaced by $\tilde{b}_r^0$. Defining ℓ_k^0 as the collection of points in $\tilde{b}_r^0$ distance k from the line $L_\mathbf{g} = \{\lambda\mathbf{g} : \lambda \in \mathbb{R}\}$ as before, we find that in this case

$$\ell_0^0 := \{n\mathbf{g} \in \tilde{b}_r : n\mathbf{g} \cdot e_2 \le 0\}$$

is *never* empty. In fact, for $0 \le \theta < \pi$ we have

$$\ell_0^0 = \begin{cases} \{n\mathbf{g} : -r \le n \le -1\} & \text{if } 0 < \theta < \pi \\ \{n\mathbf{g} : -r \le n \le r, n \ne 0\} & \text{if } \theta = 0. \end{cases}$$

If $\theta > 0$, we proceed as before and find

$$\begin{aligned}
\mathbf{g}^*_{r,\mu} &= \sum_{\mathbf{y}\in\ell_0^0} \frac{\mathbf{y}}{\|\mathbf{y}\|} + \sum_{k=1}^{r} \sum_{\mathbf{y}\in\ell_k^0} e^{-\frac{\mu^2}{2r^2}k^2\|\mathbf{g}\|^2} \frac{\mathbf{y}}{\|\mathbf{y}\|} \\
&\to \sum_{\mathbf{y}\in\ell_0^0} \frac{\mathbf{y}}{\|\mathbf{y}\|} \qquad \text{as } \mu \to \infty. \\
&= \sum_{n=-r}^{-1} n\mathbf{g} \\
&= \mathbf{g} \qquad \text{after rescaling.}
\end{aligned}$$

However, if $\theta = 0$, this argument doesn't work because the elements of ℓ_0^0 all cancel each other out. In fact, in this case we have

$$\mathbf{g}^*_{r,\mu} = \underbrace{\sum_{\mathbf{y}\in\ell_0^0} \frac{\mathbf{y}}{\|\mathbf{y}\|}}_{=0} + \underbrace{\sum_{\mathbf{y}\in b_r^-} e^{-\frac{\mu^2}{2r^2}(\mathbf{y}\cdot\mathbf{g}^\perp)^2} \frac{\mathbf{y}}{\|\mathbf{y}\|}}_{\text{the } \mathbf{g}^*_{r,\mu} \text{ from coherence transport.}}.$$

Hence, in this case $\mathbf{g}^*_r = e_2$ yet again, just like for Guidefill and coherence transport. In general, for $0 \le \theta < \pi$, we have

$$\theta^*_r = \begin{cases} \frac{\pi}{2} & \text{if } \theta = 0 \\ \theta & \text{if } 0 < \theta < \pi \end{cases} \tag{6.6.11}$$

In other words, semi-implicit Guidefill kinks only if $\mathbf{g}$ is exactly parallel to boundary of the inpainting domain. See Figure 6.10(e)-(f) for an illustration of (6.6.11) for $r = 3$ and $r = 5$ (the curves are of course the same, since (6.6.11) is independent of r).

Remark 6.6.4. *In contrast to Guidefill and coherence transport,* (6.6.11) *tells us that for semi-implicit Guidefill in the limit $\mu \to \infty$ we have $\theta^*_r = \theta$ for all $\theta \in (0, \pi)$, independent of r. This is in fact exactly the same prediction* (6.5.1) *(albeit under stronger simplifying assumptions) that März and Bornemann obtained for coherence transport in their own continuum limit $u_{\text{märz}}$ as $\mu \to \infty$* (6.5.1)*. So in this case we have $u = u_{\text{märz}}$ independent of r. We have in some sense come full circle - the original predictions of [12] for coherence transport under their high resolution and vanishing viscosity limit are the same as ours for semi-implicit Guidefill under our fixed ratio limit.*

6.7 Asymptotic limit and blur

In this section we utilize the connection between Algorithm 1 and stopped random walks to explore the origins of blur artifacts. The main idea is that the continuum limit studied in Theorem 6.3.1 - while useful for studying kinking artifacts - is inadequate for studying blur. For the latter, one instead needs to consider an *asymptotic limit* where h is very small but still non-zero. This is accomplished by leveraging a central limit theorem for the type of stopped random walk relevant to us, which states that a suitably rescaled version of our stopped random walk (or more precisely, its x-coordinate) is converging in distribution to a standard normal distribution as $h \to 0$. Since we have the identity

$$u_h(\mathbf{x}) = \sum_{\mathbf{y}\in\mathcal{U}_h} \rho_{\mathbf{X}_\tau}(\mathbf{y}) u_0(\mathbf{y}), \tag{6.7.1}$$

this suggests that u_h can be related to a mollified version of u_0, with a Gaussian mollifier. This idea is made precise in Conjecture 6.7.1, and is supported by numerical experiments (Figures 6.13 and 6.15). However, because the central limit theorem we utilize gives no rates of convergence, Conjecture 6.7.1 remains a conjecture - its rigorous proof will be the subject of future work. Before elaborating on these ideas, we first provide some motivation with a closer look at our L^p convergence rates from Theorem 6.3.1, and a case in which they can be tightened - what we call *degenerate* stencils. If Conjecture 6.7.1 is true, then degenerate stencils give a sufficient condition under which blur artifacts can be avoided. However, they also tend to increase kinking artifacts, so there is an apparent trade-off (recall from Figure 2.4 that coherence transport kinks but doesn't blur, while Guidefill blurs but doesn't kink).

6.7.1 Degerate stencils and L^p convergence rates

The first clue that blur artifacts might be present in u_h comes from Theorem 6.3.1, where we see convergence to u in L^p for every $p \in [1, \infty)$ but not necessarily in L^∞. This is consistent with blur that is present for every $h > 0$ but becomes less and less pronounced as $h \to 0$. A clue that these artifacts might be in some cases be avoidable comes from the fact that if the stencil weights put all of their mass into a single $\mathbf{y} \in a_r^*$ (we call such weights *degenerate*), as occurs in coherence transport with $\mathbf{g} = (\cos\theta, \sin\theta)$ for all but finitely many θ (Section 6.6.2), then the bounds in Theorem 6.3.1 can be tightened. This is because the random walk $\mathbf{X}_\tau$ has become deterministic. All terms related to variances or probabilities are killed off, and the bound immediately changes to

$$\|u - u_h\|_p \leq K(rh)^{s \wedge \left(s' + \frac{1}{p}\right) \wedge 1}. \tag{6.7.2}$$

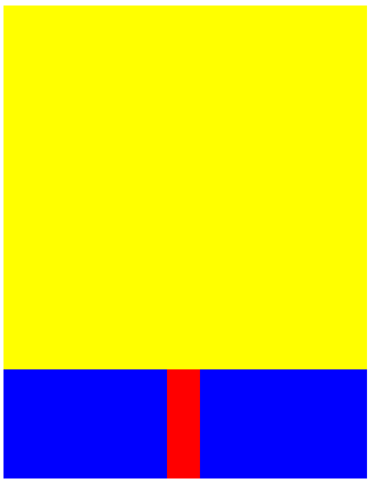
In some cases, dependent on the boundary data $u_0 : \mathcal{U}_h \to \mathbb{R}^d$, it is even possible to prove convergence in L^∞ for u_0 with jump discontinuities.

6.7.2 An asymptotic limit for Algorithm 1

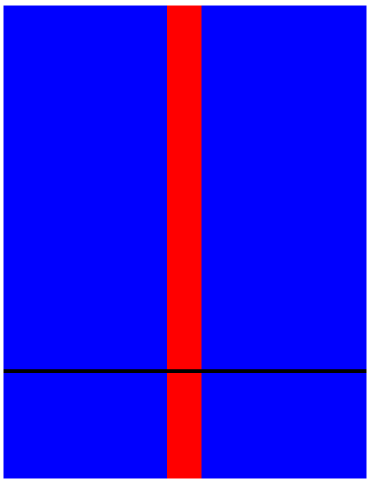
The ideas in this section rely on the book [32], which covers the asymptotic distribution functions of certain stopped random walks. Recall that we have the identity (6.7.1), where $\rho_{\mathbf{X}_\tau}$ is the probability density function of the stopped random walk $\mathbf{X}_\tau = \mathbf{x} + h\sum_{i=1}^{j} \mathbf{Z}_i$. The steps $\{\mathbf{Z}_i\} = \{(V_i, W_i)\}$ are i.i.d. and take values in $\mathrm{Supp}(a_r^*) \subseteq \bar{b}_r^-$ (if we use the direct version of Algorithm 1) or $\mathrm{Supp}(a_r^*) \subseteq \bar{b}_r^0$ (if we use the semi-implicit extension) with probability density

$$P(\mathbf{Z}_i = \mathbf{y}) = \frac{\tilde{w}_r(\mathbf{0}, \mathbf{y})}{W}, \tag{6.7.3}$$

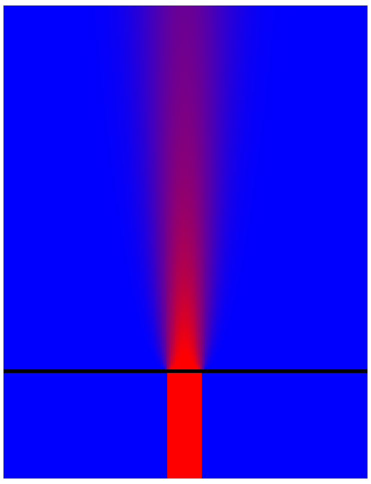
where $\tilde{w}_r$ are the equivalent weights defined in Section 5.1. Denote the mean of $\mathbf{Z}_i := (V_i, W_i)$ by (μ_x, μ_y) and let τ given by (6.3.11) be the first passage time through $y = 0$. The bound $\mu_y < 0$ is guaranteed (otherwise, the transport equation (6.3.1) is not well posed), and hence $\mathbf{X}_\tau$ has negative drift in the y-direction. As we have already stated in Section 6.3, this type of random walk is well understood [32, Chapter 4], [33], [31], [30]. The way to understand blur artifacts is to recognize that $\rho_{\mathbf{X}_\tau}$ is a mollifier of sorts and moreover, because these types of stopped random walks obey a central limit theorem [32, Chapter 4, Theorem 2.3], $\rho_{\mathbf{X}_\tau}$ is asymptotically approaching a Gaussian mollifier. In fact, if Conjecture 6.7.1 is correct, then the precise form of mollification is identical to the definition of discrete mollifaction presented in applications such as [47, Section 3], in the context of stabilizing noisy data for the purposes of numerical differentiation, and in [9] for more general applications. The mollifier is also equivalent to the one proposed in [47, Section 2].



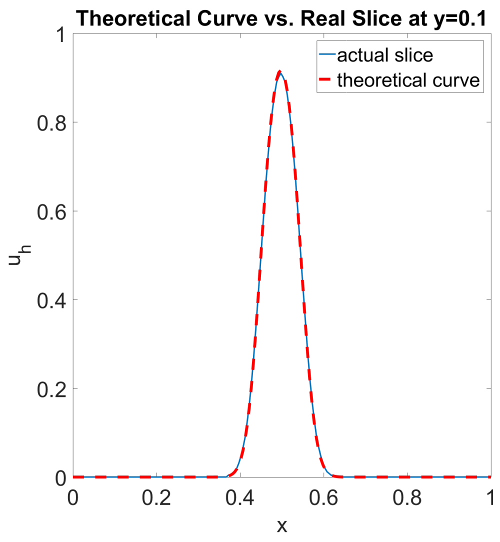
(a) Original inpainting problem. $D = [0,1]^2$ is 200×200px and shown in yellow.



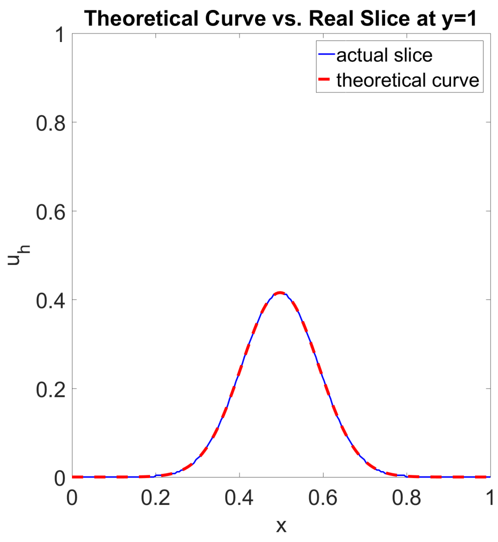
(b) Inpainting with Guidefill, $\mu = 100$, $r = 3$, $\mathbf{g} = e_2$.



(c) Inpainting with Guidefill, $\mu = 100$, $r = 3$, $\mathbf{g} = e_1$.

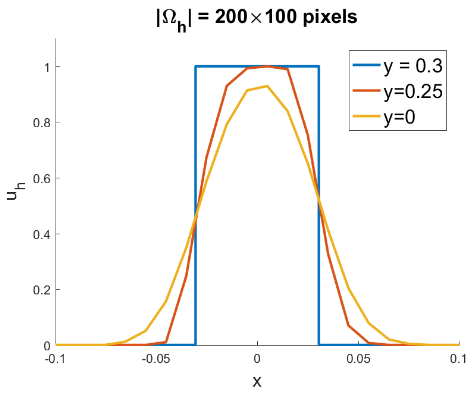


(d) Slice of the result from (c) at $y = 0.1$, compared with the theoretical curve from Conjecture 6.7.1.

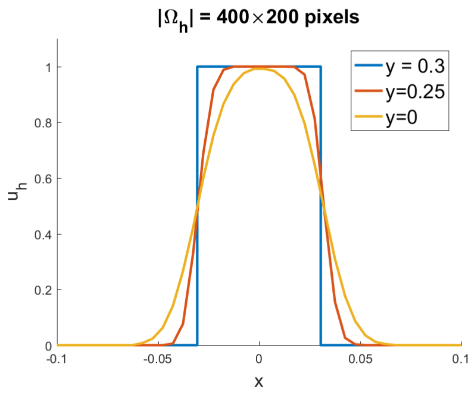


(e) Slice of the result from (c) at $y = 1.0$, compared with the theoretical curve from Conjecture 6.7.1.

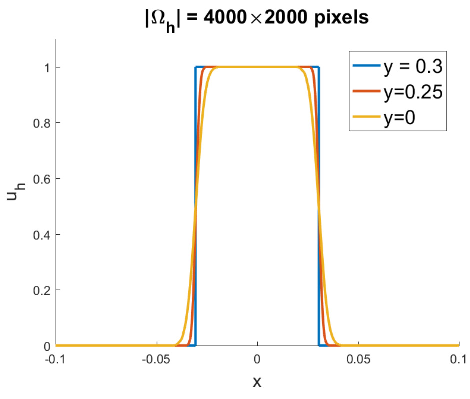
Figure 6.13: **Transport is not the whole story:** In this experiment, the problem shown in (a) of inpainting $D = [0,1]^2$ (200×200px) given data on $[0,1] \times [-0.3, 0)$ is solved using Guidefill with $\mu = 100$, $r = 3$ and $\mathbf{g} = (\cos\theta, \sin\theta)$ for $\theta = \frac{\pi}{2}$ (b) and $\theta = 0$ (c). Theorem 6.3.1 and the subsequent analysis of the limiting transport direction for Guidefill in Section 6.6.2 (Figure 6.10 and (6.6.10)) suggests that these two situations are in some sense the same, as they are both converging to the same transport equation (6.3.1) with the same transport direction $\mathbf{g}_r^* = e_2$. However, one case yields a clean extrapolation while the other suffers from heavy blur. This means the continuum limit of Algorithm 1 presented in Theorem 6.3.1, while useful, is inadequate for studying blur artifacts. Instead of a continuum limit, Conjecture 6.7.1 proposes an *asymptotic* limit where h is very small but non-zero. As we illustrate here, this limit is able to make quantitative predictions that are in excellent agreement with blur artifacts measured in practice. In (c)-(d) we compare horizontal slices of (c) at $y = 0.1$ and $y = 1$ respectively with the predictions of Conjecture 6.7.1. In this case the predictions are accurate to within an error of $1/255$, the minimum increment of an image on our machine.



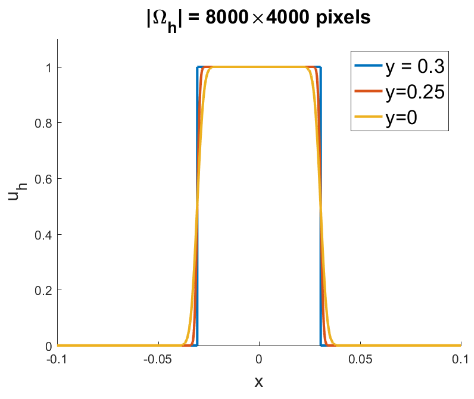
(a)



(b)

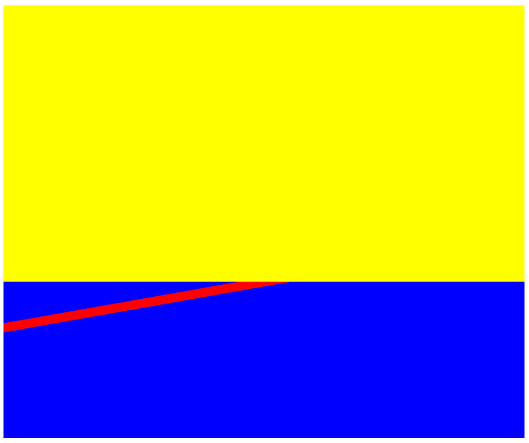


(c)

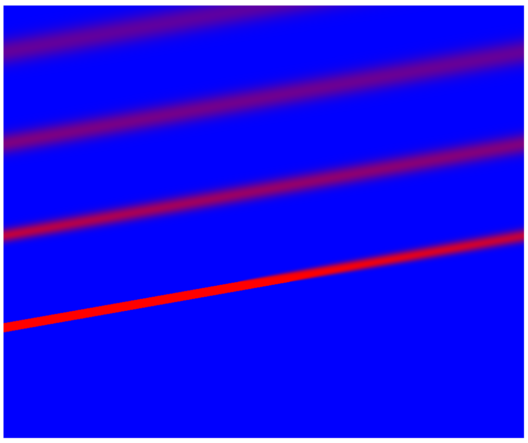


(d)

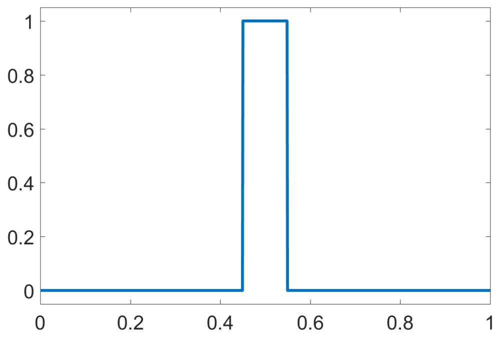
Figure 6.14: **Signal degradation and L^p convergence of Guidefill:** The continuum problem of inpainting the line $\tan(73°) - 0.1 \leq y \leq \tan(73°) + 0.1$ with image domain $\Omega = [-1, 1] \times [-0.5, 0.5]$ and inpainting domain $D = [-0.8, 0.8] \times [-0.3, 0.3]$ is rendered at a variety of resolutions and inpainted each time using Guidefill. Examining cross-sections of u_h at $y = 0.3$ (on the boundary of D_h), $y = 0.25$ (just inside), and $y = 0$ (in the middle of D_h) we notice a gradual deterioration of the initially sharp signal. If Conjecture 6.7.1 is correct, then this deterioration is to be expected, as $u_h(\mathbf{x})$ is related to a mollified version of u_0 with a Gaussian mollifier $g_{\sigma(h)}$. This does not contradict our L^p convergence results in Theorem 6.3.1, since $\sigma(h) \to 0$ as $h \to 0$ (it does, however, shed some light on why Theorem 6.3.1 can establish convergence in L^p for every $p < \infty$ but not in L^∞ in general). Here we see directly how decreasing h leads to less and less loss of signal. Another perspective is that since Guidefill is based on iterated bilinear interpolation, we should expect this effect as iterated bilinear interpolation is known to lead to signal degradation [57, Sec. 5]. However, despite this we have less degradation in higher resolution images, even though we have applied more bilinear interpolation operations.



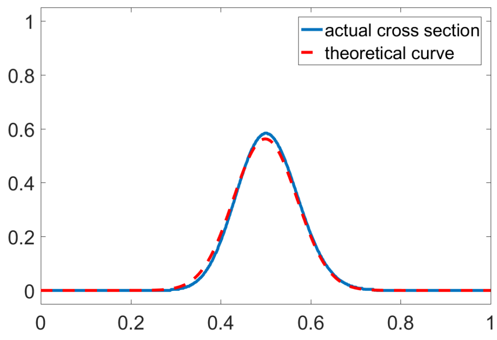
(a) Original inpainting problem (D_h in yellow).



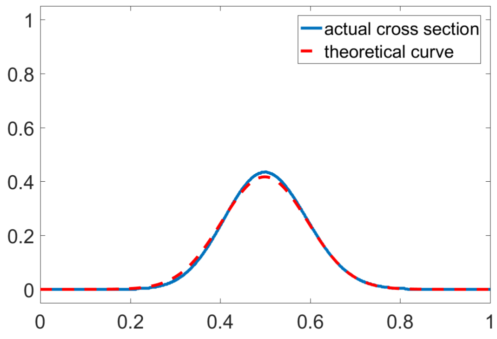
(b) Inpainted using semi-implicit Guidefill with periodic boundary conditions.



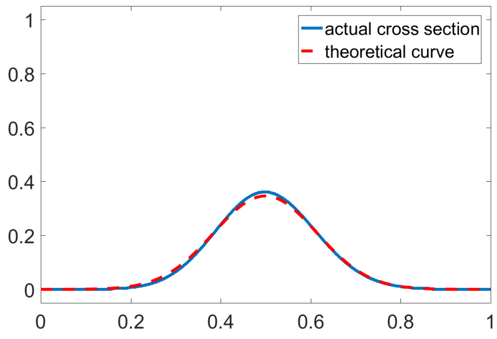
(c) Boundary data at $y = 0$.



(d) $y = \tan(10°) \approx 0.18$.



(e) $y = 2\tan(10°) \approx 0.35$.



(f) $y = 3\tan(10°) \approx 0.53$.

Figure 6.15: **Numerical evidence for Conjecture 6.7.1:** Similarly to Figure 6.13, here we consider again the problem of inpainting $D = [0,1]^2$ given data on $[0,1] \times [-0.3, 0)$. This time u_0 consists of a line making a an angle of 10° with the horizontal, but the slice $u_0(x, 0)$ is the same step as in Figure 6.13. This time D_h is 1000×1000px. Inpainting is done using semi-implicit Guidefill ($r = 3$, $\mathbf{g} = (\cos 10°, \sin 10°)$ $\mu = 100$). In (c) we show the initially sharp signal at $y = 0$, while (d)-(f) compare horizontal slices at $y = \tan(10°) \approx 0.18$, $y = 2\tan(10° \approx 0.35$ and $y = 3\tan(10°) \approx 0.53$ with the predictions of Conjecture 6.7.1. Even though in this case u_0 is not independent of y, we ignore this and use (6.7.4) for our predictions, once again obtaining a very good prediction. Compared with Figure 6.13, notice that despite the fact that h has decreased by an order of magnitude, our loss of signal is much more rapid. This is consistent with the divergence $\sigma(h) \to \infty$ as $\theta \to 0$ we will encounter later in Figure 6.16.

Conjecture 6.7.1. *Let the inpainting domain D and undamaged area $\mathcal{U}$, as well as their discrete counterparts $D_h, \mathcal{U}_h$ be as described in Section 6.1. Let $u_0 : \mathcal{U} \to \mathbb{R}^d$ denote as usual the undamaged portion of the image, and assume u_0 is non-negative and bounded, that is, there is an $M > 0$ such that $0 \le u_0 \le M$ on $\mathcal{U}$. Suppose we inpaint D_h using Algorithm 1 or its semi-implicit extension, and denote the result by $u_h : D_h \to \mathbb{R}^d$. Assume the assumptions of Section 6.1 hold and let a^*_r denote the stencil of our inpainting method. Let $Z_i = (V_i, W_i)$ taking values in Supp(a^*_r) with mean (μ_x, μ_y) denote the increments of the random walk described above, with probability density given by (6.7.3). Let $\Pi_{\theta^*_r} : D \to \partial D$ denote the transport operator defined in (6.3.3) (recall that that the transport direction $\mathbf{g}^*_r$ obeys $\mathbf{g}^*_r = (\mu_x, \mu_y)$). Then, if $u_0 : \mathcal{U} \to \mathbb{R}^d$ is independent of its y-coordinate, we have as $h \to 0$*

$$u_h(x,y) = (u_0\Big|_{y=0} * g_{\sigma(h)})(\Pi_{\theta^*_r}(x,y)) + o(1) \tag{6.7.4}$$

where $u_0\Big|_{y=0} * g_{\sigma(h)}$ denotes the discrete mollification of $u_0\Big|_{y=0}$ with mollifier $g_{\sigma(h)}$ defined for any $x \in (0,1]$ by

$$\left(u_0\Big|_{y=0} * g_{\sigma(h)}\right)(x) := \sum_{i=1}^{N} \left[\int_{(i-1)h}^{ih} g_{\sigma(h)}(x-t)dt\right] u_0\Big|_{y=0}(ih).$$

and where $g_{\sigma(h)}$ is a one dimensional Gaussian kernel with h-dependent variance given by

$$\sigma(h)^2 = \frac{\gamma^2 y h}{|\mu_y|^3} \qquad \text{where } \gamma^2 = \mathrm{Var}(\mu_x W_1 - \mu_y V_1) \tag{6.7.5}$$

Moreover, for general u_0 we have as $h \to 0$

$$u_h(x,y) = (\tilde{u}_0 * g_{\sigma(h)})(\Pi_{\theta^*_r}(x,y)) + o(1), \tag{6.7.6}$$

where $\tilde{u}_0 : \partial D_h \to \mathbb{R}^d$ obeys $\tilde{u}_0(ih) = \sum_{j=-r-1}^{0} \alpha_j(x) u_0(ih, jh)$ where for each $(ih, 0) \in \partial D_h$ we have $0 \le \alpha_j(ih) \le 1$ and $\sum_{j=-r-1}^{0} \alpha_j(ih) = 1$ (in other words, for each ih, $\tilde{u}_0(ih)$ is a convex combination of $u_0(ih, 0)$ and the $r+1$ pixels directly below it).

Proof Idea: Although proving this conjecture is beyond the scope of our current work, we briefly sketch how a proof might go and where the technical challenges arise. First, note that it suffices to prove claim two, as the first claim is then a special case. Our first job is to define $\tilde{u}_0$. To that end, note that

$$\begin{aligned} u_h(x,y) &= \sum_{i=1}^{N} \sum_{j=-r-1}^{0} \rho_{\mathbf{X}_\tau}(ih, jh) u_0(ih, jh) \\ &= \sum_{i=1}^{N} \rho_{X_\tau}(ih) \underbrace{\sum_{j=-r-1}^{0} \rho_{Y_\tau|X_\tau}(jh|ih) u_0(ih, jh)}_{:=\tilde{u}_0(ih)}. \end{aligned}$$

This definition of $\tilde{u}_0$ obeys the claimed properties since $\alpha_j(ih) := \rho_{Y_\tau|X_\tau}(jh|ih)$ obeys $0 \le \alpha_j(ih) \le 1$ and $\sum_{j=-r-2}^{0} \alpha_j(ih) = 1$ for all $i = 1, \ldots N$ as claimed. Next, define $\hat{x}(x,y) := \Pi_{\theta^*_r}(x,y)$ for convenience. Fix $y > 0$. Our goal is to show that for any $\epsilon > 0$ for h sufficiently small we have

$$|u_h(x,y) - (\tilde{u}_0 * g_{\sigma(h)})(\hat{x}(x,y))| < \epsilon.$$

To that end, note that

$$|u_h(x,y) - (\tilde{u}_0 * g_{\sigma(h)})(\hat{x}(x,y))| = \left|\sum_{i=1}^{N} \tilde{u}_0(ih)\left[\rho_{X_\tau}(ih) - \int_{(i-1)h}^{ih} g_{\sigma(h)}(\hat{x}-t)dt\right]\right|. \tag{6.7.7}$$

It follows from [32, Chapter 4, Theorem 2.3] that

$$\rho_{X_\tau}(ih) - \int_{(i-1)h}^{ih} g_{\sigma(h)}(\hat{x}-t)dt \to 0$$

as $h \to 0$, at a rate independent of $ih = x$. However, unfortunately, no quantitative rate of convergence is given - without this we cannot prove that the RHS of (6.7.7) goes to 0 (since the number of terms in the RHS diverges to infinity as $h \to 0$ while the terms themselves shrink to 0, a convergence rate is needed to quantify the balance between these competing effects). In the future, we hope to derive rates ourselves, so that we can upgrade Conjecture 6.7.1 to a theorem.

Remark 6.7.2. *Conjecture 6.7.1 says that $u_h(x,y)$, which we already know from Theorem 6.3.1 converges as $h \to 0$ to the solution of the transport equation* (6.3.1)*, can also be viewed asymptotically for small but non-zero h as a solution to the same transport equation but where the boundary data has been mollified by a Gaussian kernel $g_{\sigma(h)}$. Moreover, the degree of mollification* (6.7.5) *increases as we move further into the inpainting domain (that is, $\sigma(h)$ increases) and decreases as $h \to 0$ ($\sigma(h)$ decreases). In fact, we have already observed in Figure 6.7 that $g_{\sigma(h)}$ is converging as $h \to 0$ to a Dirac delta distribution.*

6.7.3 Angular dependence of blur artifacts

Conjecture 6.7.1 proposes an asymptotic relationship between $u_h(x,y)$ and the convolution of $u_0(x,0)$ with a Gaussian kernel of h-dependent variance $\sigma(h)^2$ given by (6.7.5). Although currently only a conjecture and valid only valid asymptotically as $h \to 0$, Figures 6.13 and 6.15 suggest that the conjecture is not only valid but gives an extremely good approximation even for very small images (The example in Figure 6.13 is only 200×200px). Conjecture 6.7.1 has three takeaway messages:

- Blur gets *worse* as one moves further into the inpainting domain and (e.g. y increases) - Figures 6.13 and 6.15.

- Blur gets *better* as $h \to 0$ - Figure 6.14.

- Blur is *non-existent* if the stencil weights are *degenerate* - that is, put all of their mass into a single *real* pixel $\mathbf{y}$ (since in this case $\mathbf{Z}_i$ is deterministic and all variances are 0).

All three of these observations are consistent with experience. The third point in particular explains why coherence transport, which has a degenerate stencil in the limit $\mu \to \infty$ for all but finitely many $\theta = \theta(\mathbf{g})$ (Section 6.6.2), does not appear to suffer from blur artifacts at all, e.g. Figure 2.4. However, it also predicts that for those few special angles where coherence transport puts its mass into more than one $\mathbf{y} \in b_r^-$, it will blur just like everything else (experiments, which we omit for the sake of space, confirm this).

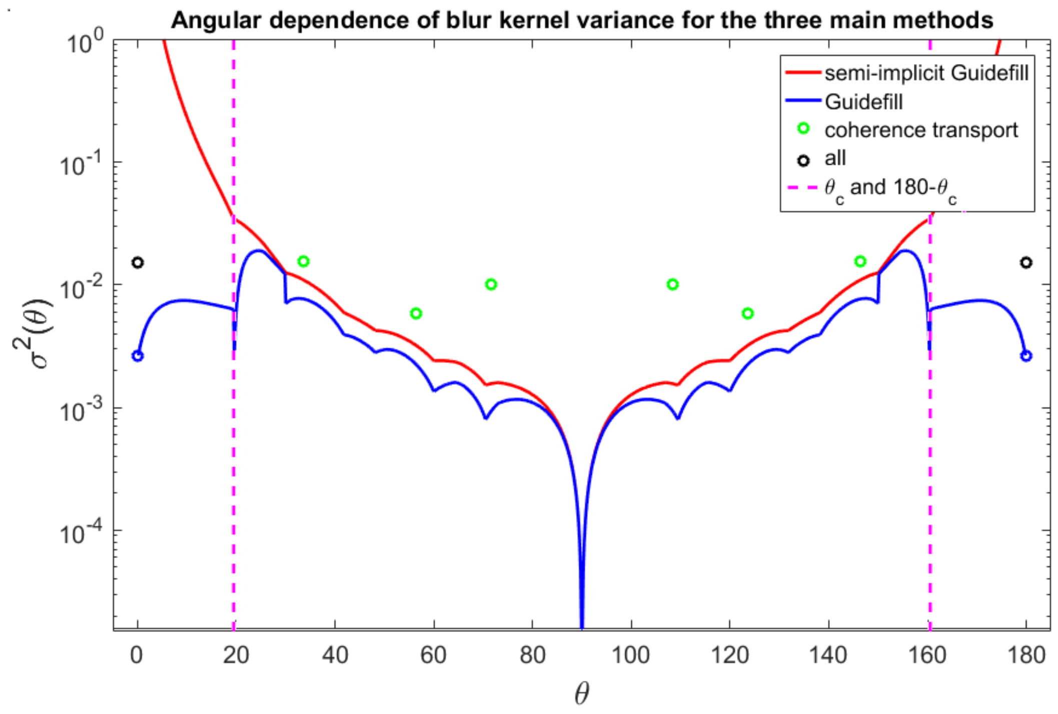


Figure 6.16: **Angular variation of blurring artifacts:** Here we plot the angular dependence of the variance $\sigma^2(h)$ from the asymptotic blur kernel $g_{\sigma(h)}$ in Conjecture 6.7.1, for the three methods coherence transport, Guidefill, and semi-implicit Guidefill. We fix with $r = 3$, take $\mu \to \infty$, vary $\mathbf{g} = (\cos\theta, \sin\theta)$, and plot $\sigma^2(h)$ as a function of θ. We fix $y = 1$ and $h = 0.01$. Note that for coherence transport, $\sigma^2(h) = 0$ for all but finitely many angles, explaining the methods apparent lack of blur artifacts (Figure 2.4). These special angles correspond precisely to the jumps in Figure 6.10(a), where coherence transport puts its mass into more than one $\mathbf{y} \in b_r^-$ (Section 6.6.2). Note also that while $\sigma(h)^2$ remains bounded for Guidefill, for the semi-implicit extension it grows arbitrarily large as $\theta \to 0$ or $\theta \to \pi$. Note also that all three methods agree at $\theta = 0$ and $\theta = \pi$, just like they did in Section 6.6.2. Note the log scale.

What is less clear from (6.7.5) is how $\sigma(h)^2$ depends on $\mathbf{g} = (\cos\theta, \sin\theta)$, as $\frac{\gamma^2}{|\mu_y^3|}$ can be quite complex in general (certainly for semi-implicit Guidefill at least, where $|\mu_y| = g_r^* \cdot e_2$ can get arbitrarily close to 0, we expect severe blur artifacts for shallow angles). Figure 6.16 illustrates the angular dependence of $\sigma(h)$ as a function of $\theta \in [0, \pi]$ with $y = 1$ and $h = 0.01$ fixed, for the three main methods of interest - coherence transport, Guidefill, and semi-implicit Guidefill (note the log scale). In every case we have $r = 3$ and $\mu = 100$.

Remark 6.7.3. *Figure 6.16 suggests a trade-off of sorts between kinking artifacts and blur artifacts. If Conjecture 6.7.1 is true, then semi-implicit Guidefill, the only method considered so far capable of avoiding kinking artifacts unless* $\mathbf{g} = (\cos\theta, \sin\theta)$ *is exactly parallel to the inpainting domain, that is* $\mathbf{g}_r^* = \mathbf{g}$ *unless* $\theta \in \{0, \pi\}$*, pays a price for this ability with blurring artifacts that become arbitrarily bad as* $\theta \to 0$ *or* $\theta \to \pi$*. At the opposite extreme, coherence transport suffers from no blur at all for all but finitely many angles, but also kinks (that is* $\mathbf{g}_r^* \neq \mathbf{g}$*) for all but finitely many angles (Figure 2.6). Guidefill is in the middle:* $\mathbf{g}_r^* = \mathbf{g}$ *so long as* θ *is not too shallow, and blur exists but remains bounded. It remains to be seen whether a method be completely free of both kinking and blur artifacts, but it seems unlikely. A more modest goal would be to design a method which like semi-implicit Guidefill suffers from no kinking artifacts, but for which* $\sigma(h)$ *defined by* (6.7.5) *remains bounded for all* θ*, like Guidefill. Whether or not this is possible also remains to be seen, and is something we would like to investigate in the future.*

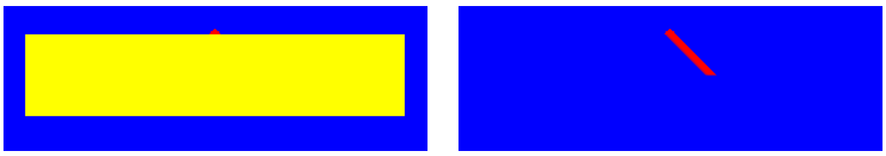

(a) Original Inpainting problem, inpainting domain shown in yellow. (b) Inpainting using coherence transport with $\mathbf{g} = (\cos 45°, \sin 45°)$, $r = 3$, $\mu = 40$.

Figure 6.17: **Stretching a dot into a line:** In (a) we have an inpainting problem consisting of a red dot on a blue background, with the inpainting domain in yellow. In (b), we see the result of inpainting using coherence transport with $\mu = 40$, $r = 3$, and $\mathbf{g} = (\cos 45°, \sin 45°)$. The dot is now stretched into a line, the orientation of which may be measured to deduce $\mathbf{g}^*_r$.

6.8 Numerical Experiments

In this section we have three objectives:

1. To compare the limiting transport directions derived in Section 6.6.2 for coherence transport, Guidefill, and semi-implicit Guidefill as $\mu \to \infty$ with the orientation of extrapolated isophotes when the algorithm is run in practice.

2. To check experimentally our claim in Section 6.5 that when r is a small integer, our limit u typically does a better job of capturing the behaviour of u_h than $u_{\text{märz}}$ does, that is $\|u_h - u\|_p \ll \|u_h - u_{\text{märz}}\|_p$.

3. To verify our bounds in Theorem 6.3.1 and check that they are tight. Further, we wish to see whether or not our results continue to hold if some of our simplifying assumptions are relaxed (specifically, Guidefill with a constant guidance direction $\mathbf{g}$ replaced by a smoothly varying transport field $\mathbf{g}(\mathbf{x})$).

We will perform one experiment for each objective, but for the sake of space, Experiment III is deferred to Appendix A.10. For Experiments II and III, we will experiment using a variety of boundary data u_0 satisfying the hypotheses of Section 6.3, as well as a few different inpainting methods that are each special cases of Algorithm 1. These are presented in detail in Appendix A.9.

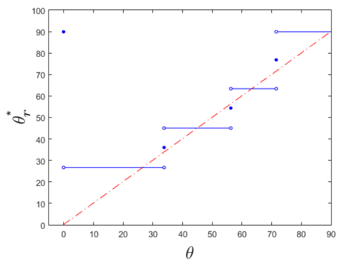
6.8.1 Experiment I: Validation of limiting transport directions for coherence transport, Guidefill, and semi-implicit Guidefill.

In this experiment we compare the limiting transport directions derived in Section 6.6.2 for coherence transport, Guidefill, and semi-implicit Guidefill as $\mu \to \infty$ with the orientation of extrapolated isophotes in an actual inpainted image u_h obtained in practice with finite μ. In each case we choose as our boundary data the image shown in Figure 6.17(a), consisting of a red dot on a blue background, with the inpainting domain shown in yellow. We run each algorithm with

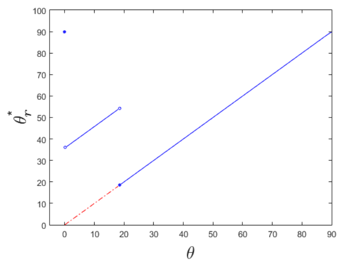
$$\mathbf{g} = (\cos(k°), \sin(k°))$$

for $k = 0, 1, \ldots, 90$, with $\mu = 40$ fixed and for various values of r. The dot is then stretched into a line as in Figure 6.17(b), the orientation of which gives $\mathbf{g}^*_{\mu,r}$ and which can be measured numerically.

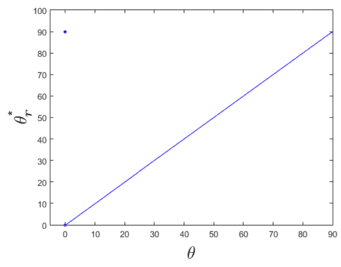
Results are shown in Figure 6.18 for $r = 3$ and $r = 5$. The top row (a)-(c) gives the theoretical curves for $r = 3$, with the actual measured results in the row underneath (d)-(f). The third row (g)-(i) gives the theoretical curves for $r = 5$, while the final row (j)-(l) gives corresponding real measurements. While we see some smoothing out of the jump discontinuities in the case of coherence transport, this is expected



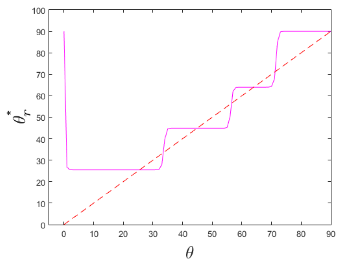
(a) Coherence transport, theoretical curve, $r = 3$.



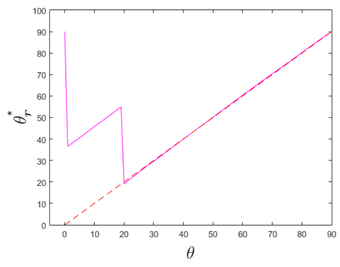
(b) Guidefill, theoretical curve, $r = 3$.



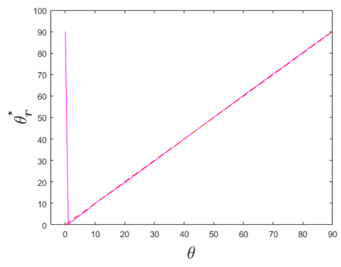
(c) Semi-implicit Guidefill, theoretical curve, $r = 3$.



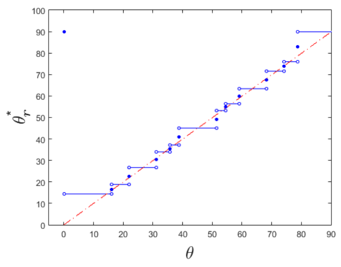
(d) Coherence transport, actual result, $r = 3$.



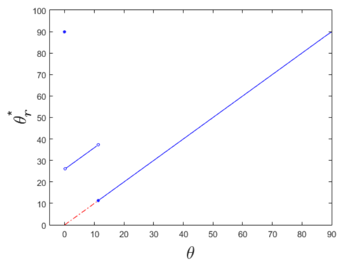
(e) Guidefill, actual result, $r = 3$.



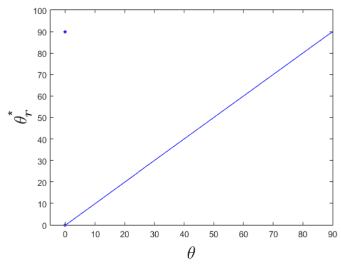
(f) Semi-implicit Guidefill, actual result, $r = 3$.



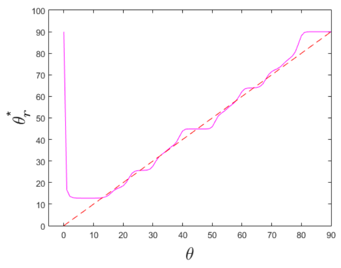
(g) Coherence transport, theoretical curve, $r = 5$.



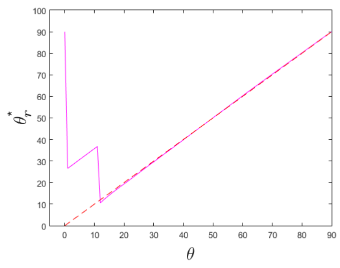
(h) Guidefill, theoretical curve, $r = 5$.



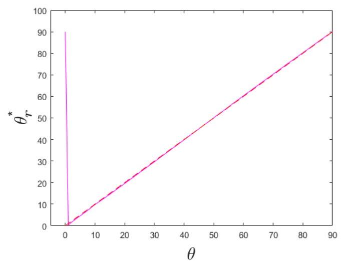
(i) Semi-implicit Guidefill, theoretical curve, $r = 5$.



(j) Coherence transport, actual result, $r = 5$.



(k) Guidefill, actual result, $r = 5$.



(l) Semi-implicit Guidefill, actual result, $r = 5$.

Figure 6.18: **Validation of limiting transport directions for coherence transport, Guidefill, and semi-implicit Guidefill:** Here we compare the limiting transport directions $\theta_r^* = \theta(\mathbf{g}_r^*)$ as a function of $\theta = \theta(\mathbf{g})$ derived in Section 6.6.2 for coherence transport, Guidefill, and semi-implicit Guidefill as $\mu \to \infty$ with the orientation of extrapolated isophotes in the inpainting problem shown in Figure 6.17(a) (where $\mu = 40$). The top row (a)-(c) gives the theoretical curves for $r = 3$, with the actual measured results in the row underneath (d)-(f). The third row (g)-(i) gives the theoretical curves for $r = 5$, while the final row (j)-(l) gives corresponding real measurements.

as one can easily show that the convergence to $\theta^* = \lim_{\mu\to\infty} \theta^*_{r,\mu}$ is pointwise but not uniform, becoming arbitrarily slow in the vicinity of the jumps. On the other hand, for Guidefill and semi-implicit Guidefill we see excellent agreement with theory even in the vicinity of jump discontinuities. This is to be expected as well, since one can easily show that the relevant limits are uniform in this case.

6.8.2 Experiment II: Comparison of convergence rates to both continuum limits

Our second experiment explores the relationship between $\|u_h - u\|_p$ and $\|u_h - u_{\text{märz}}\|_p$, for u_h such that the limit $u_{\text{märz}}$ exists, and under the assumption that we are not in the trivial case $u = u_{\text{märz}}$. From Section 6.5 we have the bound

$$\|u - u_{\text{märz}}\|_p \leq K_1 \cdot (rh)^{(\frac{s'}{2}+\frac{1}{2p})\wedge\frac{s}{2}\wedge 1} + K_2 r^{-q\{s\wedge(s'+\frac{1}{p})\wedge 1\}}. \tag{6.8.1}$$

If this bound is tight, then when r is a small integer and h is small, we expect $\|u_h - u\|_p \ll \|u_h - u_{\text{märz}}\|_p$. At the same time, Theorem 6.5.1 shows that $u \to u_{\text{märz}}$ as $r \to \infty$, so we expect $\|u_h - u\|_p \approx \|u_h - u_{\text{märz}}\|_p$ when r is large. Figure 6.19 tests this by plotting $\|u_h - u\|_p$ and $\|u_h - u_{\text{märz}}\|_p$ as a function of r with h fixed and for various choices of p, and for a few choices of boundary data u_0 and inpainting methods from Appendix A.9. Specifically, the boundary data u_0 is given by (A.9.1) for various values of s and s', and the weights and neighborhoods are as in Example 1 and Example 2 from Appendix A.9. The results confirm our expectations. For the sake of space, we only show a few illustrative examples.

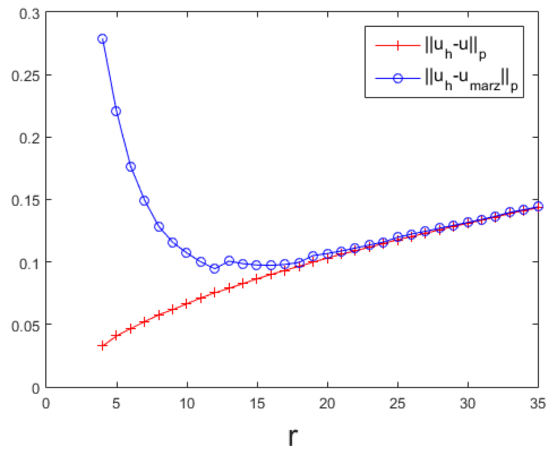
6.9 Conclusions

In this chapter we have presented a detailed analysis of the class of geometric inpainting algorithms introduced in Chapter 2. Methods in the literature falling within this framework include

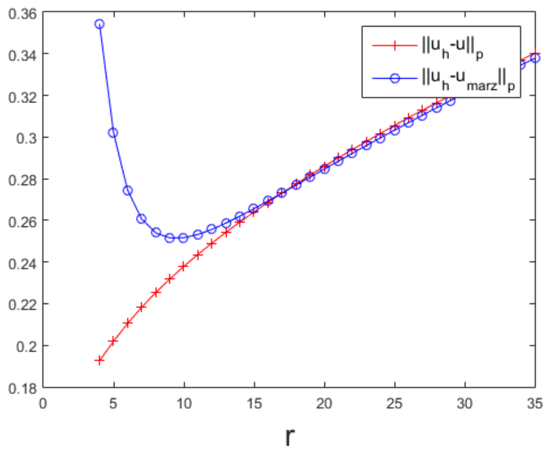
- Telea's Algorithm [64].
- Coherence Transport [12, 44].
- Guidefill [37].

A subtle but important point about the above methods is that pixels in the current inpainting boundary are filled independently. Noting this, we proposed in this thesis a semi-implicit extension (or semi-implicit form) of these methods in which pixels in the current shell are instead filled simultaneously by solving a linear system. An implementation of the semi-implicit form equivalent to solving this linear system using damped Jacobi or successive over-relaxation (SOR) was highlighted in blue in Algorithm 1 (the equivalence was proven in Proposition 5.2.2). The matrix associated with the linear system was in Chapter 5, where it was shown to be strictly diagonally dominant. In the same chapter, a semi-implicit version of Guidefill was introduced. In this chapter, a theoretical convergence analysis was presented for the semi-implicit extension of Guidefill, where we showed that SOR converges extremely quickly. This analysis is backed up by numerical experiments. We also presented in some detail additional features of semi-implicit Guidefill relating to how the method decides on a good order to fill pixels, but this is not the main focus of our work.

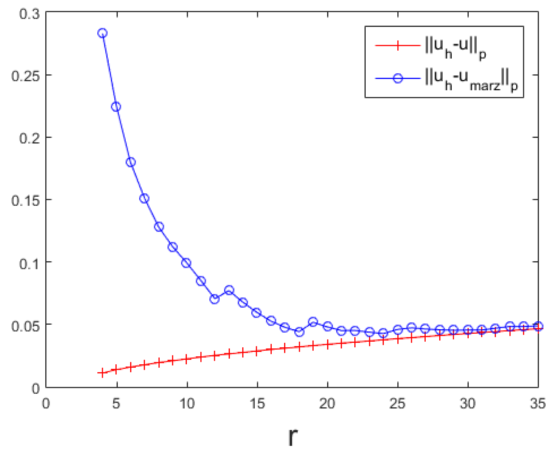
One obvious question is whether or not the semi-implicit extension makes any difference. Our theoretical analysis of kinking and blur artifacts in this chapter shows that it does. Our analysis proves that within the class of algorithms under scrutiny, kinking artifacts will always be present unless one uses the



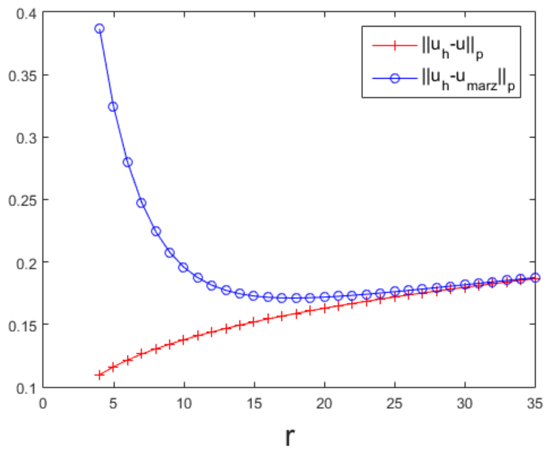
(a) Example 1, $h = 2^{-10}$, u_0 given by (A.9.1) with $s = 2$, $s' = 0$, L^1 norm.



(b) Example 2, $h = 2^{-10}$, u_0 given by (A.9.1) with $s = s' = 0.5$, L^2 norm.



(c) Example 1, $h = 2^{-13}$, u_0 given by (A.9.1) with $s = 2$, $s' = 0$, L^1 norm.



(d) Example 2, $h = 2^{-13}$, u_0 given by (A.9.1) with $s = s' = 0.5$, L^2 norm.

Figure 6.19: **Quantitative comparison of the L^p distance between u_h and its two continuum limits:** Plots of $\|u_h - u\|_p$ (starred line) and $\|u_h - u_{\text{märz}}\|_p$ (dotted line) as a function of r for $h = 2^{-10}$ (top row) and $h = 2^{-13}$ (bottom row) fixed. In each case u_0 is given by (A.9.1) for different values of s' and s. In the the left column we have $s = 2$, $s' = 0$, $w(\cdot,\cdot)$ is given by März's weights (2.5.2) with $\mu = 10$, $\mathbf{g} = (\cos 20°, \sin 20°)$ as in Appendix A.9, Example 1, and error is measured using the L^1 norm. In the right column we have $s = s' = 0.5$, $w(\cdot,\cdot)$ is given by the offset Gaussian (A.9.3) as in Appendix A.9, Example 2, and error is measured using the L^2 norm. In every case we have $\|u_h - u\|_p \ll \|u_h - u_{\text{märz}}\|_p$ when r is small. This sheds some light on why our continuum limit u appears to much more closely match the actual inpainted solution u_h than the alternative limit $u_{\text{märz}}$ does. Results for other values of s and s' as well as different choices of norm are similar but omitted.

semi-implicit extension. However, our analysis also shows that the semi-implicit extension tends to make blur artifacts a little bit worse.

As discussed in Section 2.2, all of the algorithms listed above are known to create some disturbing visual artifacts, and the main objective of our analysis is to understand why these occur and whether or not they are inevitable. We focus specifically on kinking artifacts and blur artifacts. Other artifacts including the formation of shocks and extrapolated isophotes that end abruptly are discussed but not analyzed, as they have already been studied elsewhere [12, 45] and are well understood. Our analysis is based on two key ideas:

- A continuum limit, which we use to explain kinking artifacts.
- A connection to the theory of stopped random walks, which we use both to prove convergence to our continuum limit, and to explore blur artifacts.

Similarly to the earlier work of Bornemann and März [12], our continuum limit is a transport equation. However, as discussed in Section 2.4 our limit process is different and so are the coefficients of the resulting transport equation. Moreover, numerical experiments show that our transport equation is a better reflection of the behaviour of Algorithm 1 (the direct form and our proposed semi-implicit extension) in practice, capable of accurately predicting kinking phenomena that is not captured by the alternative continuum limit proposed in [12]. The second core idea of our analysis, which is to relate Algorithm 1 and its extension to stopped random walks, is critical for two reasons. Firstly, it allows us to prove convergence to our continuum limit even for boundary data with low regularity, such as (finitely many) jump discontinuities. By contrast, the analysis in [12] assumes smooth boundary data, which is an unrealistic assumption for images. Secondly, this connection is central to our analysis of blur artifacts, which we analyze based not on a continuum limit where $h \to 0$, but rather an asymptotic limit where h very small but nonzero. While we have not (yet) been able to prove convergence to this asymptotic limit, it allows us to make quantitative predictions that are in excellent agreement with numerical experiments, even for relatively low resolution images (e.g. Figure 6.13 which is only 200×200px). Our analysis operates in a highly idealized setting (Section 6.1), but our conclusions are far reaching. In particular, we prove the following:

1. The rate of convergence to our continuum limit depends on the regularity of the boundary data, and convergence is slower for boundary data with lower regularity. Thus, our continuum limit u does a better job of approximating the actual inpainted image u_h when the boundary data u_0 is smooth (Theorem 6.3.1).

2. The difference between our continuum limit and the one proposed in [12] is most significant when the radius r of the averaging neighborhood $A_{\epsilon,h}(\mathbf{x})$ (measured in pixels) is small, and goes to zero as $r \to \infty$ (Theorem 6.5.1).

3. In the direct form of Algorithm 1, kinking artifacts will always be present. That is, certain isophotes cannot be extended into the inpainting domain without bending (Section 6.6.1).

4. This is not true of the semi-implicit extension of Algorithm 1. In particular, semi-implicit Guidefill can extrapolate isophotes with any orientation[1], and moreover is able to do so efficiently by using SOR to solve the required linear system (Section 6.6.1, Section 6.6.2, Corollary 6.2.3).

The following results, which we do not prove, are implied by Conjecture 6.7.1 if it is true:

[1]that is, unless the isophotes are exactly parallel to the boundary of the inpainting domain. But in this case extrapolation is not defined.

1. Blur artifacts exhibit an angular dependence, which for semi-implicit Guidefill becomes arbitrarily bad as the angle an extrapolated isophote makes with the boundary of the inpainting domain goes to zero. Thus, semi-implicit Guidefill pays a heavy price (on top of the need to solve a linear system for every shell) for its ability to successfully extrapolate such isophotes (Conjecture 6.7.1, Figure 6.16).

2. Blur artifacts become less significant as the resolution of the image goes up, and get worse the further into the inpainting domain you extrapolate (Conjecture 6.7.1).

3. Methods that put all of their weight into a single pixel exhibit no blur, but can only extrapolate without kinking in finitely many directions (Remark 6.7.3).

The last of these conclusions suggest that there is an apparent trade off between kinking artifacts and blur artifacts, however, this is something that requires additional investigation. Our inability to prove Conjecture 6.7.1 is due to a result on stopped random walks from [32] - that our result depends on - failing to give rates of convergence. In the future we hope to derive those rates ourselves so that we can prove or disprove this conjecture.

Beyond proving Conjecture 6.7.1, there are a number of interesting new directions this analysis could take in the future. These are discussed in Chapter 8.

Chapter 7

Spacetime Transport: A 3D Generalization

In this second to last chapter, we discuss a generalization of the ideas of coherence transport and Guidefill to shell-based video inpainting of the form illustrated in Figure 1.3. The framework is now exactly the same as the one given for shell-based image inpainting in Algorithm 1, except that the underlying objects are now 3D. Please refer to Section 1.4.1 for a list of 3D specific notations.

Our generalization, which we call spacetime transport, aims to preserve, as much as possible, the speed of frame by frame shell-based approaches like Guidefill, while improving temporal coherence. At the same time, it makes a connection between shell-based image inpainting and what is called optical flow based video inpainting.

7.1 Optical flow based video inpainting

One of the important differences between image and video inpainting is that for the latter, in many cases where either the inpainting domain or the background is in motion, the missing information in a given frame can be found in some other frame. This is illustrated in Figure 7.1, where in (a)-(c) we have a ball-shaped inpainting domain moving in front of a static background, while in (d)-(f) we have a background moving relative to a stationary inpainting domain.

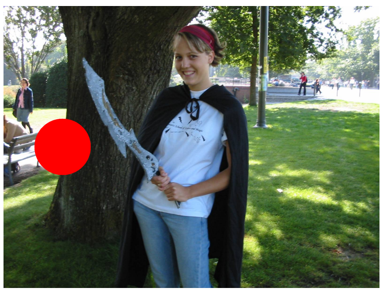
The connection between frames is encapsulated in the idea of optical flow, which assigns to each pixel $\mathbf{x}$ in frame t of a video a vector $\mathbf{v}(\mathbf{x},t) \in \mathbb{R}^2$ representing the displacement from the underlying object at pixel $\mathbf{x}$ in frame t to its new location in frame $t+\Delta t$. Optical flow is calculated by first assuming that the brightness of the underlying object is constant, that is

$$u(\mathbf{x},t) = u(\mathbf{x}+\mathbf{v}(\mathbf{x},t), t+\Delta t). \tag{7.1.1}$$

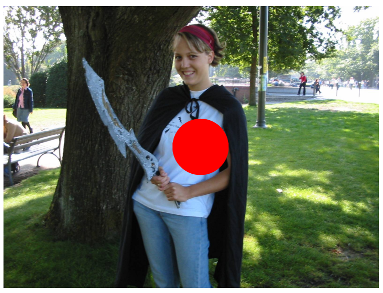
Second, one assumes that the unknown displacement $\mathbf{v}(\mathbf{x},t)$ is small enough that one may expand the above equation to first order, yielding the optical flow equations

$$\nabla u(\mathbf{x},t) \cdot \mathbf{v}(\mathbf{x},t) - \frac{\partial u}{\partial t}(\mathbf{x},t) = 0, \tag{7.1.2}$$

(where we have assumed $\Delta t = 1$ for convenience, as is typically done). Note that the above equation is overdetermined as there are two unknowns (the two components of $\mathbf{v}$) for every pixel. To obtain a unique optical flow, additional constraints must be added, which different algorithms accomplish in different



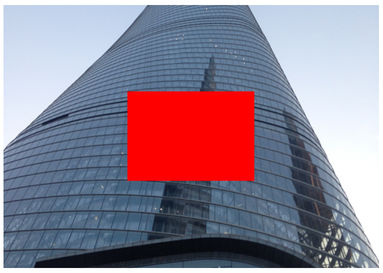
(a) Fauve: frame 1



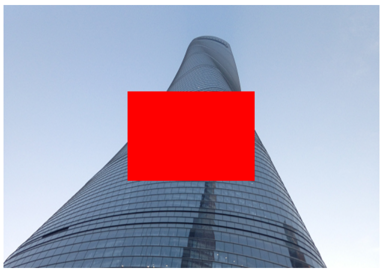
(b) Fauve: frame 50

(c) Fauve: frame 100

(d) Shanghai tower: frame 1



(e) Shanghai tower: Frame 51



(f) Shanghai tower: Frame 101

Figure 7.1: **Moving inpainting domain or background:** In cases where the inpainting domain and background are in motion relative to one another, the missing information in a given frame can often be found in another frame. This is illustrated in (a)-(c), where we see sample frames from a video consisting a ball shaped inpainting domain in motion relative to a static background, and in (d)-(e), where we have a moving background and a stationary inpainting domain.

ways. Here we discuss only one approach based on least squares (Section 7.6). See [8] for an overview of optical flow techniques. The concept of optical flow is illustrated in Figure 7.2, where we have provided the ground truth optical flow for the two examples shown in Figure 7.1.

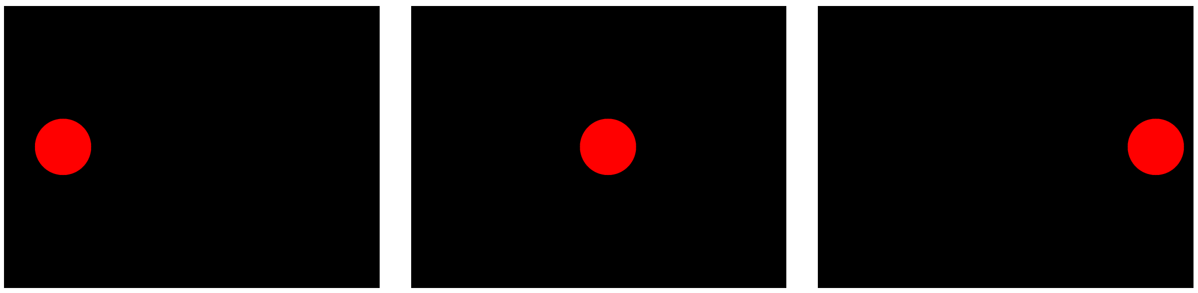
There are a number of video inpainting methods in the literature based on optical flow. In this chapter, we focus on a specific optical flow based inpainting strategy consisting of two steps:

1. Estimate the optical flow on the exterior of the inpainting domain using an algorithm of choice, and then inpaint the optical flow into the inpainting domain.

2. Use the inpainted optical flow to inpaint the video (this is typically, but not always, done based on some form of transport along the optical flow vector field).

These two steps are essentially independent problems, and there is great variability in how each is handled. For example, [38] completes step 1 using a Bayesian framework to estimate the most likely optical flow in the occluded area, then inpaints each frame sequentially by taking a weighted average of "motion corrected" versions of earlier frames. These "motion corrected" frames are estimates for the current frame created by processing earlier frames using the inpainted optical flow. In [61] an exemplar-based approach, using 3D spacetime "patches" of optical flow data, is first used to inpaint the optical flow, while the video inpainting itself is done utilizing a graph model constructed from the inpainted optical flow. The authors of [62] inpaint the optical flow frame by frame using Telea's algorithm [64], then inpaint the video using a frame by frame variant of Criminisi's algorithm [22] constrained to be consistent with the inpainted optical flow. They also propose a convenient user guided mask definition procedure (for defining the inpainting domain), based on user drawn scribbles transported along the optical flow field.

Remark 7.1.1. *There are other video inpainting algorithms making use of optical flow that do not consist of the above two steps. For example, [40] is a recent high-quality exemplar-based approach that inpaints*


(a) Fauve: frame 1 (b) Fauve: frame 50 (c) Fauve: frame 100

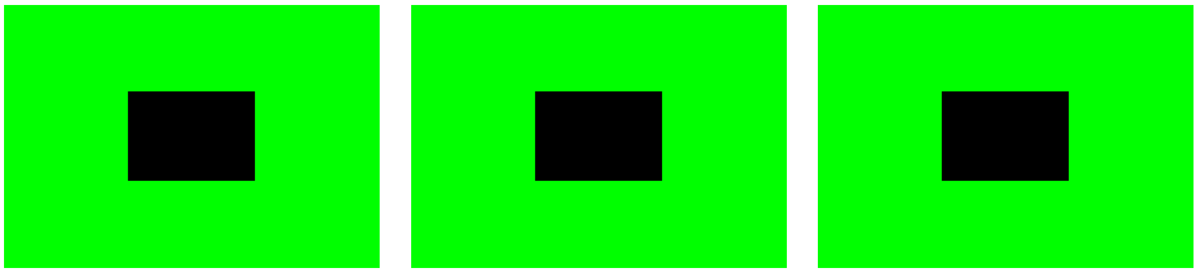

(d) Shanghai tower: frame 1 (e) Shanghai tower: Frame 51 (f) Shanghai tower: Frame 101

Figure 7.2: **Optical Flow:** Ground truth optical flow for the two examples in Figure 7.1. In each case, the optical flow is piecewise constant. In (a)-(c), we have $\mathbf{v}_n \equiv \mathbf{0}$ on the stationary background, while $\mathbf{v}_n \equiv (12, 0)$ on the inpainting domain, which moves at a rate of 12 pixels per frame (we represent this as a pure red, since the flow is purely in the x-direction). In (d)-(f), we have $\mathbf{v}_n \equiv 0$ on the stationary inpainting domain, while $\mathbf{v}_n \equiv (0, -3)$ on the moving background, which moves down by three pixels every frame (we represent this by pure green as the flow is purely in the y direction).

the optical flow and the video simultaneously by minimizing a global energy, using "patches" that are 3D parallelepipeds of spacetime skewed based on the optical flow. However, for the purposes of this chapter, when we say "optical flow based video inpainting", we mean an algorithm based on the above two steps.

7.1.1 Parallels with shell-based image inpainting

There are parallels between the two steps of optical flow based video inpainting and the two steps of shell-based image inpainting common to coherence transport and Guidefill. In the case of image inpainting, the first step is the determination of the orientation of image isophotes in the exterior of the inpainting domain, done using the structure tensor in the case of Guidefill, and the modified structure tensor in the case of coherence transport. This is analogous to the computation of an optical flow field on the exterior of the (video) inpainting domain in optical flow based video inpainting. Indeed, as we will show in Section 7.6, there is a form of Lucas–Kanade optical flow that is calculated in terms of the structure tensor. Next, the guide field used by both coherence transport and Guidefill is analogous to the inpainted optical flow. This connection is more obvious in the case of Guidefill, where the guide field is computed on its own as a separate step, similarly to the separate optical flow inpainting step. Finally, the color inpainting step done using the guidance field in shell-based image inpainting is analogous to the color inpainting step based on the inpainted optical flow in optical flow based video inpainting, particularly when the latter is done based on an explicit transport mechanism (that is, based on some procedure for enforcing or at least encouraging constant color along the integral curves of the inpainted optical flow field).

7.1.2 Blur artifacts in optical flow based video inpainting

Something else that shell-based image inpainting and optical flow based video inpainting have in common is blur artifacts. In the latter case, these arise because the optical flow is rarely integer-valued, meaning that pixel centers in a given frame are typically mapped to locations between pixel centers in nearby frames. This necessitates some form of interpolation which, as a number of authors have noted, leads to progressive blur. For example, [38] first noted that when bilinear interpolation is used "after a few frames the image in the region of the [inpainting domain] rapidly loses its detail", and proposed to use windowed sinc or bicubic interpolation instead. However, these schemes both involve negative weights and can lead to instabilities, as the authors themselves note. They are thus combined with heuristics for clamping to zero small weights deemed likely to lead to instabilities. In [57], the authors note that while higher order interpolation schemes are helpful, the problem persists. They propose a more sophisticated approach based on defining a *convective derivative*, and then alternating between forward and backward difference schemes based on this derivative. This leads to further improvements, but still does not completely resolve the issue of blur. Blur artifacts in the case of shell or transport based image inpainting have already been discussed in Section 6.7.

7.2 Spacetime transport - intuitive idea

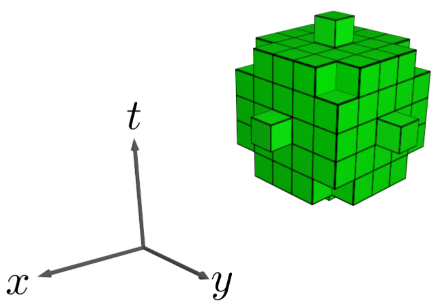
We know from Chapter 6 that shell-based inpainting, at least in the 2D case and under a suitable continuum limit, is equivalent to inpainting based on transport along a guidance field $\mathbf{g}(\mathbf{x}) \in \mathbb{R}^2$. Based on this observation, the idea of spacetime transport is to combine shell-based image inpainting and optical flow based video inpainting into a unified framework based on the idea of transporting along a general spacetime vector

$$\mathbf{g}(\mathbf{x}) = g_x(\mathbf{x})e_x + g_y(\mathbf{x})e_y + g_t(\mathbf{x})e_t,$$

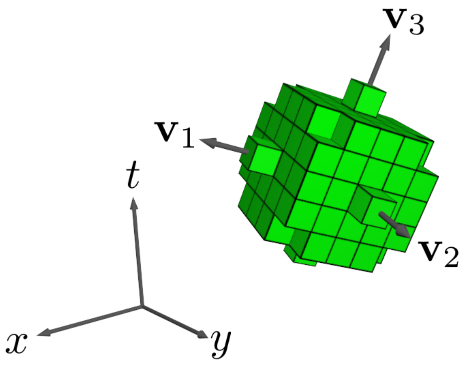
(where e_x, e_y, and e_t are the unit vectors along the x, y, and time dimensions respectively) that reduces to an optical flow vector in cases (such as those illustrated in Figure 7.1) where this approach makes sense. When this doesn't make sense, as in the case of an inpainting domain and background that are not in motion relative to one another, the temporal component of $\mathbf{g}$ should become zero so that the method reduces to extrapolation of spatial isophotes. Like coherence transport and Guidefill, spacetime transport is shell-based and operates by taking weighted averages. The connection to transport is only made explicit in the continuum limit. On the one hand, spacetime transport can be seen as a natural generalization of coherence transport and Guidefill to three dimensions - this is the perspective we present over the next several sections, which cover the components of spacetime transport in detail. However, at the same time, spacetime transport can be seen as inpainting based on a generalized Lucas–Kanade optical flow that doesn't distinguish between time and space. This latter interpretation is explored in Section 7.6.

7.3 Detailed Discussion

Like coherence transport, a core idea of spacetime transport is to first obtain information about u_h in the undamaged region $\Omega_h \backslash D_h$ based on a structure tensor $\mathcal{J}_{\sigma,\rho}$, with the weights used for inpainting then adapted based on the structure tensor. The structure tensor $\mathcal{J}_{\sigma,\rho}$ is defined in terms of the discrete gradient ∇u_h in the same way as in Section 4.3. However, now ∇u_h is a vector with three components instead of two, so $\mathcal{J}_{\sigma,\rho}$ is now a 3×3 symmetric positive semi-definite matrix, with three eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ (we denote the corresponding eigenvectors by $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$), rather than the two in



(a) $\mathbf{A}_{\epsilon,h}(\mathbf{x}) = \mathbf{B}_{\epsilon,h}(\mathbf{x})$.



(b) $\mathbf{A}_{\epsilon,h}(\mathbf{x}) = \tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$.

Figure 7.3: **Lattice aligned and rotated discrete 3D balls:** When spacetime transport uses as its averaging neighborhood $\mathbf{A}_{\epsilon,h}(\mathbf{x})$ the discrete axis aligned 3D ball of voxel centers shown in (a), it suffers from kinking artifacts analogous to those of coherence transport. Similarly to Guidefill, better results are obtained by using the rotated ball of ghost voxels in (b), which has been rotated into alignment with the orthonormal basis of eigenvectors $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ of the guidance tensor G.

the case of coherence transport and Guidefill. Coherence transport and Guidefill each have two cases of interest involving the eigenvalues $\lambda_1 \geq \lambda_2 \geq 0$ (eigenvectors $\{\mathbf{v}_1, \mathbf{v}_2\}$) and two desired behaviours for the corresponding weights $w_\epsilon(\cdot,\cdot)$, namely

1. $\lambda_1 \approx \lambda_2$: weights should be homogeneous, not favoring any direction.

2. $\lambda_1 \gg \lambda_2$: weights should be biased in favor of (ghost) pixels lying on the line parallel to $\mathbf{v}_2$.

By contrast, spacetime transport has three cases and three desired behaviours:

1. $\lambda_1 \approx \lambda_2 \approx \lambda_3$: weights should be homogeneous, not favoring any direction.

2. $\lambda_1 \gg \lambda_2 \approx \lambda_3$: weights should be biased in favor of (ghost) voxels lying on the *plane* spanned by $\mathbf{v}_2$ and $\mathbf{v}_3$.

3. $\lambda_1 \approx \lambda_2 \gg \lambda_3$ or $\lambda_1 \gg \lambda_2 \gg \lambda_3$: weights should be biased in favor of (ghost) voxels lying on the *line* parallel to $\mathbf{v}_3$.

We will see in the next section one way of adapting the weights (2.5.2) used by coherence transport and Guidefill in order to produce this behaviour. At the same time, since we anticipate some form of the kinking artifacts present in 2D to persist in 3D, we also adapt the rotated ball $\tilde{B}_{\epsilon,h}(\mathbf{x})$ of ghost pixels used by Guidefill into the present setting. Specifically, we use as our averaging neighborhood $\mathbf{A}_{\epsilon,h}(\mathbf{x})$ the rotated 3D ball of ghost voxels $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ aligned with the local orthonormal coordinate frame $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ defined by the eigenvectors of the structure tensor. If one instead uses the unrotated discrete ball of voxel centers $\mathbf{B}_{\epsilon,h}(\mathbf{x})$, then kinking artifacts are produced, just like in the 2D case. See Figure 7.3 for an illustration of the rotated and unrotated 3D balls $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ and $\mathbf{B}_{\epsilon,h}(\mathbf{x})$, as well as Section 7.7 for a discussion of kinking artifacts in 3D. As we will see, due to the temporal component that is now involved, these artifacts manifest themselves as, among other things, abrupt changes in the *velocity* of moving objects. Just like in the 2D cases, the use of $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ is not in general enough to prevent these artifacts. In general, it must be combined with the solution of the linear system arising in the semi-implicit form of the algorithm.

7.3.1 Weights

Let us denote by $d_L(\mathbf{x})$ and $d_\Pi(\mathbf{x})$ the distance from $\mathbf{x} \in \mathbb{R}^3$ to the line $L_{\mathbf{v}_3}$ and plane $\Pi_{\mathbf{v}_2,\mathbf{v}_3}$ respectively. That is,

$$d_L(\mathbf{x}) = \|\mathbf{x} \times \mathbf{v}_3\| \qquad d_\Pi(\mathbf{x}) = |\det([\mathbf{x}, \mathbf{v}_2, \mathbf{v}_3])|. \tag{7.3.1}$$

We generalize the weights (2.5.2) used by coherence transport and Guidefill as follows:

$$w_\epsilon(\mathbf{x}, \mathbf{y}) = \frac{\exp\left(-\frac{\mu_L(\mathbf{x})^2}{2\epsilon^2} d_L(\mathbf{y}-\mathbf{x})^2 - \frac{\mu_\Pi(\mathbf{x})^2}{2\epsilon^2} d_\Pi(\mathbf{y}-\mathbf{x})^2\right)}{\|\mathbf{y}-\mathbf{x}\|}. \tag{7.3.2}$$

In order to achieve our stated objectives, we need $\mu_L(\mathbf{x}) \gg 0$ when $\lambda_2 \gg \lambda_3$ and $\mu_L(\mathbf{x}) \approx 0$ when $\lambda_2 \approx \lambda_3$, while $\mu_\Pi(\mathbf{x}) \gg 0$ when $\lambda_1 \gg \lambda_2$ and $\mu_\Pi(\mathbf{x}) \approx 0$ when $\lambda_1 \approx \lambda_2$. This gives us

1. $\lambda_1 \approx \lambda_2 \approx \lambda_3$: weights should be homogeneous, not favoring any direction.

$$w_\epsilon(\mathbf{x}, \mathbf{y}) \approx \frac{1}{\|\mathbf{y}-\mathbf{x}\|}.$$

2. $\lambda_1 \gg \lambda_2 \approx \lambda_3$: weights should be biased in favor of (ghost) voxels lying on the *plane* spanned by $\mathbf{v}_2$ and $\mathbf{v}_3$.

$$w_\epsilon(\mathbf{x}, \mathbf{y}) \approx \frac{\exp\left(-\frac{\mu_\Pi(\mathbf{x})^2}{2\epsilon^2} d_\Pi(\mathbf{y}-\mathbf{x})^2\right)}{\|\mathbf{y}-\mathbf{x}\|}.$$

3. $\lambda_1 \approx \lambda_2 \gg \lambda_3$ or $\lambda_1 \gg \lambda_2 \gg \lambda_3$: weights should be biased in favor of (ghost) voxels lying on the *line* parallel to $\mathbf{v}_3$.

$$w_\epsilon(\mathbf{x}, \mathbf{y}) \approx \frac{\exp\left(-\frac{\mu_L(\mathbf{x})^2}{2\epsilon^2} d_L(\mathbf{y}-\mathbf{x})^2\right)}{\|\mathbf{y}-\mathbf{x}\|}.$$

There are various ways we could design the functions $\mu_L(\mathbf{x})$ and $\mu_\Pi(\mathbf{x})$ that are consistent with the above constraints. We use

$$\begin{aligned} \mu_L(\mathbf{x}) &= \mu \tanh\left(\frac{|\lambda_1 - \lambda_3|}{\Lambda_{1,3}}\right) \tanh\left(\frac{|\lambda_2 - \lambda_3|}{\Lambda_{2,3}}\right) \\ \mu_\Pi(\mathbf{x}) &= \mu \tanh\left(\frac{|\lambda_1 - \lambda_3|}{\Lambda_{1,3}}\right) \left[1 - \tanh\left(\frac{|\lambda_2 - \lambda_3|}{\Lambda_{2,3}}\right)\right] \end{aligned}$$

where $\mu \geq 0$, $\Lambda_{1,3} > 0$ and $\Lambda_{2,3} > 0$ are parameters. We take $\mu = 50$, $\Lambda_{1,3} = 10^{-3}/\operatorname{arctanh}(0.99)$, and $\Lambda_{2,3} = 10^{-8}/\operatorname{arctanh}(0.99)$ by default. The parameters $\Lambda_{1,3}$ and $\Lambda_{2,3}$ were determined experimentally - see Remark 7.3.1. This particular choice satisfies the above constraints, but also obeys

$$\mu_L(\mathbf{x}) + \mu_\Pi(\mathbf{x}) = \tanh\left(\frac{|\lambda_1 - \lambda_3|}{\Lambda_{1,3}}\right),$$

that is, added together μ_L and μ_Π measure the overall spread of the eigenvalues of G. Note that we could replace the denominator with $\|\mathbf{y}-\mathbf{x}\|^p$ for any $p \in [0, 2]$ without having a serious impact on the behaviour of the method or its theory. In particular, since the weights remain integrable over $\mathbf{B}_1(\mathbf{0})$ for any $p \in [0, 2]$, a limit in the style of Bornemann and März's high-resolution vanishing viscosity should exist in this case. We have chosen $p = 1$ for the sake of consistency with the 2D case.

Remark 7.3.1. *The parameter value* $\Lambda_{1,3} = 10^{-3}/\operatorname{arctanh}(0.99)$ *was selected because it was noted that an edge with a jump in brightness by the maximum value of* 1 *(we are assuming all color channels lie in* $[0, 1]$*) translated into a maximum eigenvalue of about* 10^{-3} *for the structure tensor with* $\sigma = \rho = 2px$*,* *which are the values we use by default. For other values of ρ and σ, or if the image is dominated by weak edges, one may want to use a different value of $\Lambda_{1,3}$. On the other hand, $\Lambda_{2,3} = 10^{-8}/\operatorname{arctanh}(0.99)$ was selected so that weights would remain baised in favor of the line $L_{\mathbf{v}_3}$ as often as possible, reverting over to bias in favor of the whole plane $\Pi_{\mathbf{v}_1,\mathbf{v}_3}$ only the vectors $\mathbf{v}_2$, $\mathbf{v}_3$ very nearly form a degenerate eigenspace.*

Algorithm 3 Spacetime transport

```
$u_h$ = damaged video, initialized to 0 on inpainting domain.
$\Omega = [a,b] \times [c,d] \times [e,f]$ = continuous video domain.
$\Omega_h = \Omega \cap \mathbb{Z}_h^3$ = discrete video domain.
$\sigma$ = pre-smoothing blur kernel radius for structure tensor.
$\rho$ = post-smoothing blur kernel radius for structure tensor.
$D_h^{(0)}$ = initial inpainting domain.
$\tilde{D}_h^{(0)} = \textsc{Dilate}(D_h^{(0)}, \max(4\sigma, 4\rho))$ = the extended inpainting domain, obtained from $D_h^{(0)}$ by dilating outwards by a distance of $\max(4\sigma, 4\rho)$ voxels.
$\partial D_h^{(0)}$= initial inpainting domain boundary.
$\partial \tilde{D}_h^{(0)}$= initial extended inpainting domain boundary.
$G$ = guidance tensor, equal to the structure tensor on $\Omega_h \backslash \tilde{D}_h^{(0)}$ and initialized to $O$ on $\tilde{D}_h^{(0)}$.
innerIt = number of inner iterations for computing guidance tensor.
semiImplicit = false, unless we use the semi implicit extension.
$G = \textsc{FillGuidanceTensor}(G, \tilde{D}_h^{(0)}, \partial \tilde{D}_h^{(0)}, \text{innerIt}, \text{semiImplicit})$
$u_h = \textsc{FillVideo}(u_h, G, \tilde{D}_h^{(0)}, \partial \tilde{D}_h^{(0)}, \text{semiImplicit})$
```

Like coherence transport and Guidefill, the weights described in this section will be used to fill the inpainting domain in successive shells. However, in spacetime transport this occurs in two shell-based waves, where first the guide field is inpainted and then the video itself (the reasons for this are explained in the next section). This is illustrated with pseudo code in Algorithms 3-6, which cover respectively the algorithm from a high level, the subroutines devoted to filling the guide field and the video, and finally the subroutine for filling generic data (guide field or video) within the current shell.

Remark 7.3.2. *The derivation of the weights $w_\epsilon(\mathbf{x},\mathbf{y})$ given by* (7.3.2) *was done under the assumption that $G(\mathbf{x})$ is a positive semi-definite matrix. This is certainly true on $\Omega_h \backslash \tilde{D}_h$, where the guide field G coincides with the 3D structure tensor. However, for our weights to continue to make sense within $\tilde{D}_h$, we need to make sure that our procedure for inpainting yields a guide field that is symmetric positive semi-definite. We will see shortly in Lemma 7.3.3 that it does. This is due to a generalization of the stability property* (2.1.1) *from Section 2.1.*

7.3.2 The guidance tensor field

There are various ways we might generalize Guidefill's guide field to the present 3D case - moreover, due to the modular nature of the algorithm, these may be interchanged without affecting the other components of the algorithm. One method would be to compute the guide field concurrently with inpainting in the same way as is done for coherence transport, based on a generalized modified structure tensor. However, as we have already in Section 4.3.1, this leads to additional kinking artifacts, and one may easily verify that the obvious generalization suffers from the same type of problem. Another approach, following Guidefill, would be to calculate the guide field based on *animated splines* that are computed automatically, but may be edited by the user - this time both spatially and temporally. Specifically, in this case, the user would specify not only the control points of each spline, but also how these points move over time. This could be done by specifying the position of each control point at each of a sequence of key frames. This is something that would be interesting to explore in the future, however, for now we do something simpler, following neither coherence transport nor Guidefill.

Our basic idea is to note that, as pointed out in Section 4.3.1, the structure tensor $J_{\rho,\sigma}(\mathbf{x})$ is only defined for $\mathbf{x}$ sufficiently far enough away from the inpainting domain that the spacetime patch $P_\Delta(\mathbf{x})$, a cube of size $(2\Delta+1)^3$ voxels centered at $\mathbf{x}$, where $\Delta = \max(4\sigma, 4\rho)$, does not overlap the inpainting domain ($P_\Delta(\mathbf{x})$ is the neighborhood used for pre and post blurring in the computation of the structure tensor). We formalize this by defining the extended inpainting domain $\tilde{D}_h$ by

$$\tilde{D}_h := \cup_{\mathbf{x}\in D_h} P_\Delta(\mathbf{x}). \tag{7.3.3}$$

Then, $J_{\rho,\sigma}$ is well defined on $\Omega_h \backslash \tilde{D}_h$, and the first step of spacetime transport is to extend it to all of Ω_h by inpainting. Like for Guidefill, we refer to the result as the "guide field". Unlike Guidefill, however, the guide field, which we denote by $G(\mathbf{x})$, is a tensor field rather than a vector field.

We inpaint using exactly the same procedure as we will later use for inpainting the video - i.e. a shell based approach using the weights from Section 7.3.1. However, since these weights themselves depend on $G(\mathbf{x})$, we end up with an implicit relationship even in the direct form of the method. Specifically, in this case we get:

$$G(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in\tilde{\mathbf{B}}_{G(\mathbf{x}),\epsilon,h}(\mathbf{x})\cap(\Omega\backslash D^{(k)})} w_{G(\mathbf{x}),\epsilon}(\mathbf{x},\mathbf{y})G(\mathbf{y})}{\sum_{\mathbf{y}\in\tilde{\mathbf{B}}_{G(\mathbf{x}),\epsilon,h}(\mathbf{x})\cap(\Omega\backslash D^{(k)})} w_{G(\mathbf{x}),\epsilon}(\mathbf{x},\mathbf{y})}, \tag{7.3.4}$$

where we have highlighted the implicit dependence of the neighborhood $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ and weights w_ϵ on $G(\mathbf{x})$ by writing them as $\tilde{\mathbf{B}}_{G(\mathbf{x}),\epsilon,h}(\mathbf{x})$ and $w_{G(\mathbf{x}),\epsilon}(\mathbf{x},\mathbf{y})$. We solve the above non-linear equation using the fixed point iteration

$$G^{(0)}(\mathbf{x}) = O, \quad G^{(n+1)}(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in\tilde{\mathbf{B}}_{G^{(n)}(\mathbf{x}),\epsilon,h}(\mathbf{x})\cap(\Omega\backslash D^{(k)})} w_{G^{(n)}(\mathbf{x}),\epsilon}(\mathbf{x},\mathbf{y})G(\mathbf{y})}{\sum_{\mathbf{y}\in\tilde{\mathbf{B}}_{G^{(n)}(\mathbf{x}),\epsilon,h}(\mathbf{x})\cap(\Omega\backslash D^{(k)})} w_{G^{(n)}(\mathbf{x}),\epsilon}(\mathbf{x},\mathbf{y})} \quad \text{for } n = 1,2,\ldots \tag{7.3.5}$$

We do not attempt to prove that the above iteration converges to the desired solution (or at all), or make any attempt at the analysis of a convergence rate. We simply note that empirically, it does converge and the convergence is very rapid - we fix the number of iterations at five by default. Proving this analytically is something we may consider in the future.

One important feature of the inpainting procedure (7.3.5) is that it enjoys a "generalized stability property" extending Bornemann and März's stability property (2.1.1) to matrices. This is formalized in the following lemma which states, among other things, that if the guide tensor field G is symmetric positive semi-definite on $\Omega_h\backslash\tilde{D}_h$, then after inpainting using (7.3.5) it is symmetric positive semi-definite within $\tilde{D}_h$ as well. As noted in remark 7.3.2, this is a requirement for the weights (7.3.2) derived in Section 7.3.1 to make sense.

Lemma 7.3.3. *Suppose $G : \Omega_h\backslash\tilde{D}_h \to \mathbb{R}^{3\times3}$ is a matrix valued function such that $G(\mathbf{x})$ is symmetric positive semi-definite for each $\mathbf{x}\in\Omega_h\backslash\tilde{D}_h$. Suppose G is extended to Ω_h by inpainting based on* (7.3.5)*. Then the inpainted function $G : \Omega_h \to \mathbb{R}^{3\times3}$ has the property that $G(\mathbf{x})$ is symmetric positive semi-definite for every $\mathbf{x}\in\Omega_h$. Moreover*

$$\max_{\mathbf{x}\in\Omega_h} \|G(\mathbf{x})\| = \max_{\mathbf{x}\in\Omega_h\backslash\tilde{D}_h} \|G(\mathbf{x})\|$$

for any induced matrix norm $\|\cdot\|$.

Proof. One easily verifies that if $\{\alpha_i\}_{i=1}^N$ are non-negative weights summing to 1 and $\{A_i\}_{i=1}^N$ are symmetric positive semi-definite matrices in $\mathbb{R}^{n\times n}$, then

$$A = \sum_{i=1}^{N} \alpha_i A_i$$

is symmetric positive semi-definite as well, and obeys

$$\|A\| \le \max_{i=1}^{N} \|A_i\|$$

for any induced matrix norm $\|\cdot\|$. The claim is now an easy exercise in induction, noting that the iteration formula (7.3.5) is executed finitely many times and is based on non-negative weights summing to 1. $\square$

Algorithm 4 Guidance Tensor inpainting

```
function $G$ = FillGuidanceTensor($G$, $\tilde{D}_h^{(0)}$, $\partial\tilde{D}_h^{(0)}$, innerIt, semiImplicit)
    for $k = 0, \ldots$ do
        if $\tilde{D}_h^{(k)} = \emptyset$ then
            break
        end if
        $\partial_{\text{ready}}\tilde{D}_h^{(k)} = \{\mathbf{x} \in \partial\tilde{D}_h^{(k)} : \text{readyGuide}(\mathbf{x})\}$.
        $G$ = FillBoundary($G$, $G$, $\tilde{D}_h^{(k)}$, $\partial_{\text{ready}}\tilde{D}_h^{(k)}$,innerIt).
        $\tilde{D}_h^{(k+1)} = \tilde{D}_h^{(k)} \backslash \partial_{\text{ready}}\tilde{D}_h^{(k)}$.
        if semiImplicit then
            $G^{(0)} = G$
            for $n = 1, 2, \ldots$ (until convergence) do
                $G^{(n)}$ = FillBoundary($G^{(n-1)}$, $G^{(n-1)}$, $\tilde{D}_h^{(k+1)}$, $\partial_{\text{ready}}\tilde{D}_h^{(k)}$, readyGuide, innerIt)
            end for
        end if
        $\partial\tilde{D}_h^{(k+1)} = \{\mathbf{x} \in \partial\tilde{D}_h^{(k+1)} : \mathcal{N}(\mathbf{x}) \cap (\Omega_h \backslash \tilde{D}_h^{(k+1)}) \neq \emptyset\}$.
    end for
end function
```

In the semi-implicit case, this iteration becomes an inner loop within a larger iterative method to solve the bigger system of equations relating $G(\mathbf{x})$ to its unknown neighbors $G(\mathbf{y})$ with $\mathbf{y} \in \partial D_h^{(k)}$. This is illustrated in Algorithms 3, 4, and 5, where just like in Algorithm 1, we use damped Jacobi (parallel implementation) and SOR (sequential implementation) to solve the larger linear system. However, unlike in 2D, we have chosen not to devote a section to explicitly deriving the matrix components of this linear system, nor to analyzing the convergence of damped Jacobi and SOR for its solution. While this could likely be done with little effort, there would probably be large overlap with the 2D case and it seems unlikely that the results would be very different. Therefore we have instead simply assumed that just like in the 2D case, a couple of iterations of SOR per shell will solve the system to sufficient accuracy, and have designed Algorithms 3 to 6 based on this assumption. Numerical experiments suggest that this assumption is valid.

Another thing that we do not do in the 3D case is prove convergence to a continuum limit - either a high-resolution vanishing viscosity one like März and Bornemann proposed, or a fixed ratio limit like the one studied in this thesis for the 2D case. This would also likely be quite easy to do, and there would again likely be significant overlap with the proof in the 2D case. For now, when we investigate kinking artifacts in Section 7.7, we will instead argue based on symmetry that the problem is essentially two dimensional, and use our existing results from Chapter 6, which appear to be adequate. Generalizing the results of Chapters 5 and 6 to three dimensions (or perhaps d-dimensions) is something we may consider in the future.

Algorithm 5 Video inpainting

```
function $u_h$ = FillVideo($u_h$, $G$, $\tilde{D}_h^{(0)}$, $\partial\tilde{D}_h^{(0)}$, semiImplicit)
    for $k = 0, \ldots$ do
        if $D_h^{(k)} = \emptyset$ then
            break
        end if
        $\partial_{\text{ready}} D_h^{(k)} = \{\mathbf{x} \in \partial D_h^{(k)} : \text{readyVid}(\mathbf{x})\}$.
        innerIt = 1.
        $u_h$ = FillBoundary($u_h$, $G$, $D_h^{(k)}$, $\partial_{\text{ready}} D_h^{(k)}$, readyVid, innerIt).
        $D_h^{(k+1)} = D_h^{(k)} \backslash \partial_{\text{ready}} D_h^{(k)}$
        if semiImplicit then
            $u_h^{(0)} = u_h$
            for $n = 1, 2, \ldots$ (until convergence) do
                $u_h^{(n)}$ = FillBoundary($u_h^{(n-1)}$, $G$, $D_h^{(k+1)}$, $\partial_{\text{ready}} D_h^{(k)}$, readyVid, innerIt)
            end for
        end if
        $\partial D_h^{(k+1)} = \{\mathbf{x} \in \partial D_h^{(k+1)} : \mathcal{N}(\mathbf{x}) \cap (\Omega_h \backslash D_h^{(k+1)}) \neq \emptyset\}$.
    end for
end function
```

Algorithm 6 Fill Boundary Subroutine

```
function $v$ = FillBoundary($v$, $G$, $D_h$, $\partial D_h$, ready, innerIt) // here "$v$" refers to generic data to be inpainted, which could be $u_h$ or $G$, and ready refers to a generic "ready" function, which could be either readyGuide or readyVid.
    for $\mathbf{x} \in \partial D_h$ do
        for $i = 1, \ldots$ innerIt do
            compute $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ from $G(\mathbf{x})$.
            compute non-negative weights $w_\epsilon(\mathbf{x},\mathbf{y}) \geq 0$ for $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ using (7.3.2).
            if ready($\mathbf{x}$) then
                $$v(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in (\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x}) \backslash \{\mathbf{x}\}) \cap (\Omega \backslash D)} w_\epsilon(\mathbf{x},\mathbf{y}) v(\mathbf{y})}{\sum_{\mathbf{y} \in (\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x}) \backslash \{\mathbf{x}\}) \cap (\Omega \backslash D)} w_\epsilon(\mathbf{x},\mathbf{y})} \quad (7.3.6)$$
            end if
        end for
    end for
end function
```

7.4 Voxel Ordering Strategies

The implementation of spacetime transport presented in Algorithms 3 to 6 is based on two "ready" functions, namely "readyGuide" and "readyVid", intended for the guide field and video inpainting steps respectively. The latter of these, "readyVid", is defined in exactly the same way as the "ready" function for Guidefill (Section 4.5) and semi-implicit Guidefill (Section 5.2.2), but with $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ replacing $\tilde{B}_{\epsilon,h}(\mathbf{x})$, and requires no further discussion.

However, "readyVid" relies on the guide field being already known, which obviously is not the case in the guide field inpainting step, and hence something different must be done for "readyGuide". For now, consider only two simple options:

1. readyGuide $\equiv$ “true”,
2. $\text{readyGuide}(\mathbf{x}) = 1(\mathbf{x} + \boldsymbol{\Delta} \in \Omega_h \backslash \tilde{D}^{(k)})$ where $\boldsymbol{\Delta} \in \{\pm e_x, \pm e_y, \pm e_t\}$.

Choice one is the standard onion shell approach, while choice two forces the pixel ordering to follow a particular coordinate direction. In the future we would like to look into more sophisticated ordering strategies.

7.5 Implementation considerations

In its current form, spacetime transport is implemented sequentially in C++ with a MATLAB interface, and is not very efficient. It would be highly beneficial if it could be parallelized (at least in places) and run on the GPU. At a minimum, the computation of the structure tensor on the exterior of the extended inpainting domain is both expensive and completely parallelizable, so this part of the algorithm should be moved to the GPU in the future. However, it would be ideal to also do the inpainting itself on the GPU. This poses some challenges not present in image inpainting because of the large memory requirements of video. One approach, which would at the same time enforce a pixel ordering, would be to send the video to the GPU in packets of up to a few dozen “slices” at a time. These could be slices of constant t (that is frames) but could also be slices of constant x or y. One could then either pick a direction with which to pass through the video (for example forward or backwards through time, or from left to right through space), and send the frames to the GPU as they are needed. At any given time the GPU would need to have enough frames in the vicinity of the frame it is currently working on so that the solid ball $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ is always defined. Possibly multiple passes in different directions could be made, and the results fused together based on some criterion for selecting the “best” voxels from each pass. However, this is just one possible strategy - it is likely that several suitable strategies exist, each with their own tradeoffs.

7.6 Connection with Lucas–Kanade optical flow

We have already seen how the use of the 3D structure tensor in spacetime transport arises as a natural generalization of the ideas of coherence transport and Guidefill. Now we give another perspective based on generalizing the ideas of the Lucas-Kanada optical flow algorithm in a way that does not single out the time dimension as special.

As stated in Section 7.1, the optical flow equation (7.1.2) requires additional constraints in order to define a unique optical flow. One way of doing this is the Lucas–Kanade method [41] which is based on least-squares, or weighted least squares. In this approach, rather than trying to solve (7.1.2) at each pixel $\mathbf{x}$ exactly, one defines $\mathbf{v}(\mathbf{x}, t)$ to be the minimizer of the sum

$$\sum_{\mathbf{y} \in P_\delta^{(2)}(\mathbf{x})} w_{\mathbf{y}} \left(\nabla u(\mathbf{y}, t) \cdot \mathbf{v}(\mathbf{x}, t) - \frac{\partial u}{\partial t}(\mathbf{y}, t) \right)^2,$$

where $P_\delta^{(2)}(\mathbf{x})$ is the two dimensional image patch of size $(2\delta + 1)^2$ centered at $\mathbf{x}$, and $w_{\mathbf{y}}$ is the non-negative weight of the equation corresponding to pixel $\mathbf{y}$, typically chosen as $w_{\mathbf{y}} = f(\|\mathbf{x} - \mathbf{y}\|)$ for some decreasing non-negative function f. The solution is the least squares equation

$$\mathbf{v}(\mathbf{x}, t) = -\left[\sum_{\mathbf{y} \in P_\delta^{(2)}(\mathbf{x})} w_{\mathbf{y}} \nabla u(\mathbf{y}, t) \otimes \nabla u(\mathbf{y}, t) \right]^{-1} \sum_{\mathbf{y} \in P_\delta^{(2)}(\mathbf{x})} w_{\mathbf{y}} \frac{\partial u}{\partial t}(\mathbf{y}, t).$$

If one chooses as weights the Gaussian $w_{\mathbf{y}} = g_\rho(\|\mathbf{y}-\mathbf{x}\|)$, and replaces $\nabla u(\mathbf{y},t)$ with the pre-blurred gradient $\nabla u_\sigma(\mathbf{y},t)$ (4.3.1) used in Section 4.3 in the computation of the two dimensional image structure tensor $\mathcal{J}^{(2)}_{\sigma,\rho}(\mathbf{x})$, then this becomes:

$$\mathbf{v}(\mathbf{x},t) = -\left[\mathcal{J}^{(2)}_{\sigma,\rho}(\mathbf{x})\right]^{-1} \sum_{\mathbf{y}\in P^{(2)}_\delta(\mathbf{x})} w_{\mathbf{y}} \frac{\partial u}{\partial t}(\mathbf{y},t).$$

In other words, there is an explicit connection between the two dimensional structure tensor of images, and a particular way of computing optical flow.

To see how these ideas generalize to spacetime transport, let us again start from the constant brightness assumption (7.1.1), rephrased in terms of the guidance direction $\mathbf{g}(\mathbf{x}) \in \mathbb{R}^3$ as

$$u(\mathbf{x}+\mathbf{g}(\mathbf{x})) = u(\mathbf{x}),$$

where $\mathbf{x} \in \mathbb{R}^3$ now denotes a video voxel. Expanding to first order as before, we obtain the transport equation

$$\nabla u(\mathbf{x}) \cdot \mathbf{g}(\mathbf{x}) = 0.$$

Now let us apply the same ideas as before, but replacing the two dimensional image patch $P^{(2)}_\delta(\mathbf{x})$ with the three dimensional spacetime cube $P^{(3)}_\delta(\mathbf{x})$ of size $(2\delta+1)^3$ voxels centered at $\mathbf{x}$. That is, let us seek to minimize the sum

$$\sum_{\mathbf{y}\in P^{(3)}_\delta(\mathbf{x})} w_{\mathbf{y}} \left(\nabla u_\sigma(\mathbf{y}) \cdot \mathbf{g}(\mathbf{x})\right)^2, \tag{7.6.1}$$

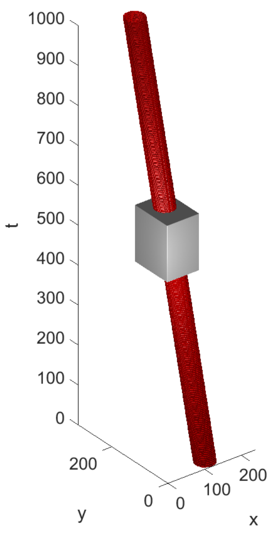
where $\mathbf{x} \in \mathbb{R}^3, \mathbf{y} \in \mathbb{R}^3$ are now voxels, $u_\sigma = g_\sigma * u$ as usual, and ∇u_σ is a vector in spacetime with three components. We add the constraint $\|\mathbf{g}(\mathbf{x})\| = 1$ in order to avoid the trivial solution, set $w_{\mathbf{y}} = g_\rho(\|\mathbf{y}-\mathbf{x}\|)$ as before, and as before we have replaced $\nabla u(\mathbf{y})$ with the pre-blurred gradient $\nabla u_\sigma(\mathbf{y})$. Then, applying the method of Lagrange multipliers yields the eigenvalue problem

$$\mathcal{J}^{(3)}_{\sigma,\rho}(\mathbf{x})\mathbf{g}(\mathbf{x}) = \lambda \mathbf{g}(\mathbf{x}),$$

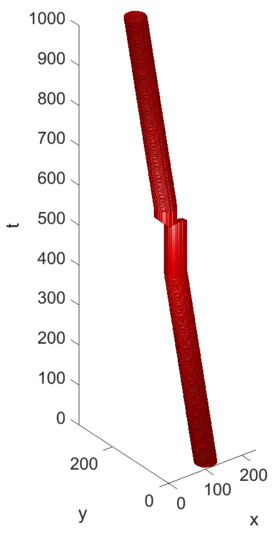
where $\mathcal{J}^{(3)}_{\sigma,\rho}(\mathbf{x})$ is exactly the three dimensional spacetime structure tensor introduced in Section 7.3. This is a necessary but not a sufficient condition - it is not necessarily the case that *all* of the eigenvectors of $\mathcal{J}^{(3)}_{\sigma,\rho}(\mathbf{x})$ minimize (7.6.1). In fact, substituting the above eigenvalue equation into (7.6.1) yields

$$\begin{aligned}
\sum_{\mathbf{y}\in P^{(3)}_\delta(\mathbf{x})} w_{\mathbf{y}} \left(\nabla u_\sigma(\mathbf{y}) \cdot \mathbf{g}(\mathbf{x})\right)^2 &= \sum_{\mathbf{y}\in P^{(3)}_\delta(\mathbf{x})} w_{\mathbf{y}} \left(\mathbf{g}(\mathbf{x}) \cdot \nabla u_\sigma(\mathbf{y})\right)\left(\nabla u_\sigma(\mathbf{y}) \cdot \mathbf{g}(\mathbf{x})\right) \\
&= \mathbf{g}(\mathbf{x}) \cdot \sum_{\mathbf{y}\in P^{(3)}_\delta(\mathbf{x})} w_{\mathbf{y}} \left(\nabla u_\sigma(\mathbf{y}) \otimes \nabla u_\sigma(\mathbf{y})\right) \mathbf{g}(\mathbf{x}) \\
&= \mathbf{g}(\mathbf{x}) \cdot \mathcal{J}^{(3)}_{\sigma,\rho}(\mathbf{x})\mathbf{g}(\mathbf{x}) \\
&= \lambda. \qquad\qquad \text{since } \|\mathbf{g}(\mathbf{x})\| = 1.
\end{aligned}$$

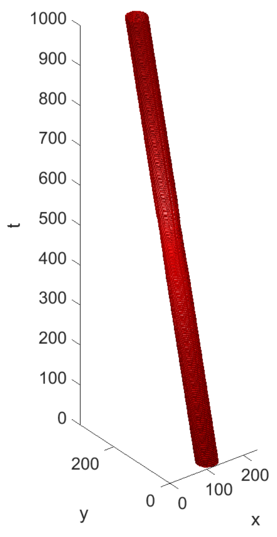
We see then that it is the minimal eigenvector $\mathbf{v}_3$ of $\mathcal{J}^{(3)}_{\sigma,\rho}(\mathbf{x})$ that satisfies the above minimization problem, and which can be thought of as a generalized optical flow direction.



(a) Inpainting problem in Example 1.



(b) Result using spacetime transport without ghost voxels.



(c) Result using spacetime transport with ghost voxels.

Figure 7.4: **Kinking in 3D:** Here we represent the inpainting problem presented in Example 1, as well as its solution using spacetime transport with and without ghost voxels, as 3D solids. In (a), the original video to be inpainted (consisting of a red ball moving against a white background) is represented by rendering the white pixels transparent, the red pixels as a red solid, and the inpainting domain as a grey solid. In (b) and (c) we see the results of inpainting without and with ghost voxels respectively, with $G = I - \mathbf{g} \otimes \mathbf{g}$ and $\mathbf{g} = \frac{1}{\sqrt{17}}(1,0,1)$ given (here $\mathbf{g}$ represents the velocity of the ball as a unit vector in spacetime). In this case $r = 3$ and $\tilde{\mu} = 40$. When ghost voxels are not used, the inpainted ball, represented here as a tilted cylinder, kinks to $90°$. However, when ghost voxels are used the inpainted cylinder maintains the correct slope. See also Figure 7.5.

7.7 Kinking and blur artifacts in 3D

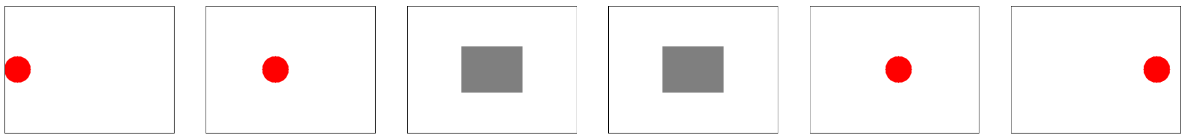
Just like shell based algorithms in 2D, spacetime transport suffers from kinking and blur artifacts as we illustrate here with a few examples. For simplicity, for now we assume that the guide field is constant and given, of the form $G = I - \mathbf{g} \otimes \mathbf{g}$ for some unit vector $\mathbf{g} \in \mathbb{R}^3$. This gives $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 0$, where $\mathbf{v}_3 = \mathbf{g}$ and $\mathbf{v}_1$, $\mathbf{v}_2$ are any orthonormal basis for the plane orthogonal to $\mathbf{g}$. This puts us into case 3 from Section 7.3.1, that is, weights are biased in favor of ghost voxels on the line parallel to $\mathbf{g}$. We have with $\mu_\Pi = 0$ (to machine precision), $\mu_L = \mu \tanh(1/\Lambda_L) := \tilde{\mu}$, so that

$$w_\epsilon(\mathbf{x}, \mathbf{y}) = \frac{\exp\left(-\frac{\tilde{\mu}^2}{2\epsilon^2} d_L(\mathbf{y} - \mathbf{x})^2\right)}{\|\mathbf{y} - \mathbf{x}\|^2}.$$

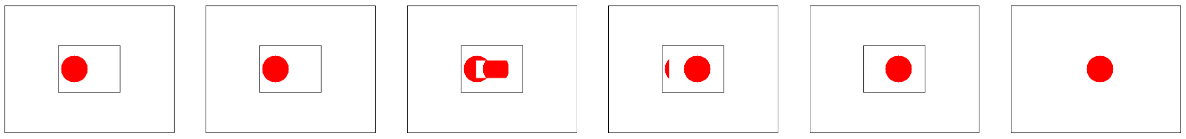
Rather than computing $\tilde{\mu}$ in terms of μ and Λ_L, for these examples it is simpler to specify it directly. In all our examples, $\mathbf{g}$ will represent either the velocity vector of a moving object or the optical flow induced by a moving camera. We focus on this simple setting first before moving onto the more complex scenario where G is computed based on inpainting the structure tensor using (7.3.4).

Example 1: A slow ball (320px ×240px ×1001fr)

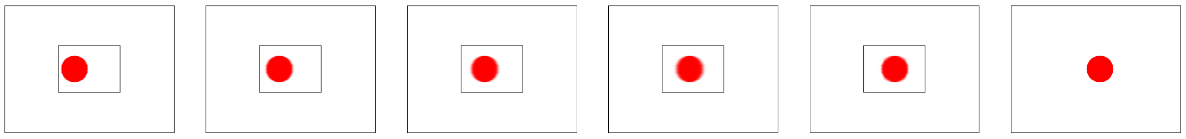
In this example, a red ball moves from left to right at a speed of one pixel every four frames for a total of 1001 frames. It is occluded by the square $[101, 215] \times [76, 162] \subset [1, 320] \times [1, 240]$ from frame 430 to 570. This is illustrated in Figure 7.4(a) where we represent the inpainting problem as a 3D solid, with the white background voxels rendered transparent, the inpainted domain represented by a grey solid, and



(a) Video to be inpainted: frame 1. (b) Video to be inpainted: frame 427. (c) Video to be inpainted: frame 428. (d) Video to be inpainted: frame 569. (e) Video to be inpainted: frame 570. (f) Video to be inpainted: frame 1001.



(g) Video to be inpainted: frame 428. (h) Inpainting without ghost voxels: frame 525. (i) Inpainting without ghost voxels: frame 543. (j) Inpainting without ghost voxels: frame 558. (k) Inpainting without ghost voxels: frame 566. (l) Inpainting without ghost voxels: frame 571.



(m) Inpainting with ghost voxels: frame 428. (n) Inpainting with ghost voxels: frame 456. (o) Inpainting with ghost voxels: frame 484. (p) Inpainting with ghost voxels: frame 514. (q) Inpainting with ghost voxels: frame 542. (r) Inpainting with ghost voxels: frame 570.

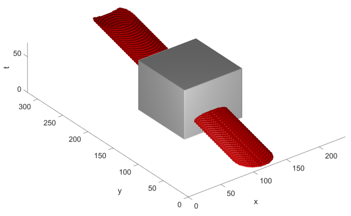
Figure 7.5: **Kinking viewed through time:** Here we explore again the inpainting problem presented in Example 1 (inpainting domain in grey), this time by presenting a few representative frames of the video to be inpainted (a)-(f), the result of inpainting using spacetime transport without ghost voxels (g)-(l), and the result with ghost voxels (m)-(r). All parameters are the same as in Figure 7.4. When viewed through time, what appeared in Figure 7.4 as the kinking of a slanted cylinder becomes a moving ball that abruptly stops, remains motionless from frames 427 to 525, disintegrates and teleports 36 pixels to the right over the next 40 frames, and then remains motionless again for a short period before continuing at its original speed. On the other hand, when ghost voxels are used the ball continues at the correct speed but now suffers from blur artifacts. For reference, we have outlined the inpainting domain in black.

the moving ball represented by a red slanted cylinder. It is further illustrated in Figure 7.5(a)-(f) where we present a few representative frames of the video to be inpainted in (a)-(f). The visualization using 3D solids was created using VoxelPlotter[1].

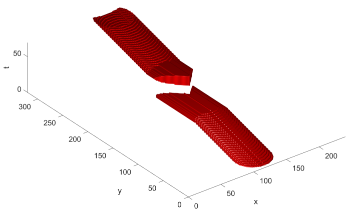
In this case $\mathbf{g} = \frac{1}{\sqrt{17}}(1, 0, 4)$, and we inpaint setting $\epsilon = 3h$ (that is, $r = 3$) and $\tilde{\mu} = 40$. The problem is essentially 2D dimensional (since the y component of $\mathbf{g}$ is zero) and since the boundary of the inpainting domain is locally plane shaped, it is reasonable to assume that the kinking behaviour will reduce down to the 2D case covered in Theorem 6.3.1 and Section 6.6. In order to save the reader the trouble of flipping back to Chapter 6, we have reprinted the relevant graphs in Figure 7.10.

In this case, the ball strikes the inpainting domain boundary in a location where it is locally a flat plane parallel to the xy-plane. The angle with the boundary is of $\arctan(4) \approx 76°$, which means that if we do not use ghost voxels then we expect to kink to a $90°$, as in Figure 7.10(a). On the other hand, with ghost voxels we expect to get the correct extrapolation. This is indeed exactly what happens, as is easiest to see in Figure 7.4(b) and Figure 7.4(c). In these examples we have converted the inpainted video, which contains not only red and white pixels but various shades of pink (due to blurring), into a solid. This is done in the same way as before but after first clamping shades of pink to either red or white, depending on which color they are closest to.

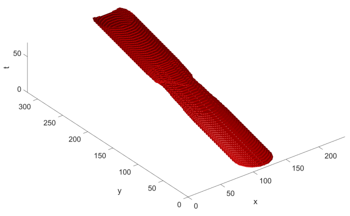
[1]Voxelplotter is a Matlab plugin available at `https://www.mathworks.com/matlabcentral/fileexchange/50802-voxelplotter`, used here in accordance with the terms of the license.



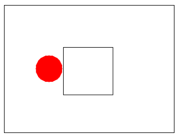
(a) Inpainting problem in Example 2, first inpainting domain.



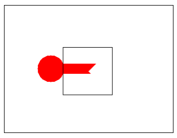
(b) Result of inpainting without ghost voxels.



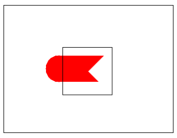
(c) Result of inpainting with ghost voxels.



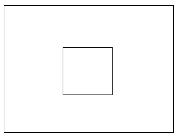
(d) Inpainting without ghost voxels: frame 15.



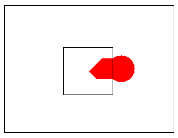
(e) Inpainting without ghost voxels: frame 16.



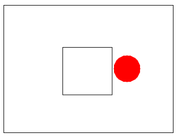
(f) Inpainting without ghost voxels: frame 20.



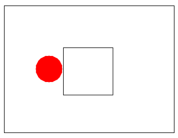
(g) Inpainting without ghost voxels: frame 36.



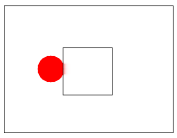
(h) Inpainting without ghost voxels: frame 49.



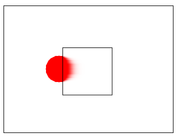
(i) Inpainting without ghost voxels: frame 52.



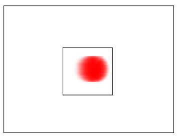
(j) Inpainting with ghost voxels: frame 15.



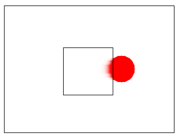
(k) Inpainting with ghost voxels: frame 16.



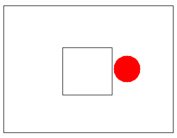
(l) Inpainting with ghost voxels: frame 20.



(m) Inpainting with ghost voxels: frame 36.



(n) Inpainting with ghost voxels: frame 49.



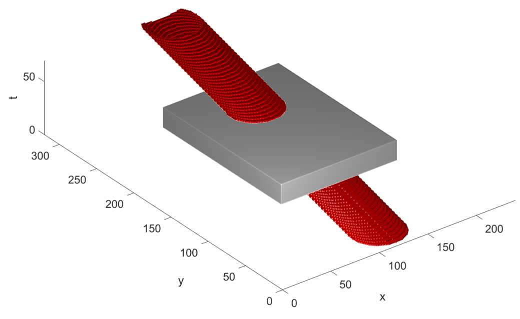
(o) Inpainting with ghost voxels: frame 52.

Figure 7.6: **Horizontal kinking:** Here we illustrate the inpainting problem presented in Example 2 with the first inpainting domain, which consists of an occluding square present in all frames. This is represented in (a) as a grey solid. In this case the ball, which appears as a strongly slanted cylinder in 3D, is bent into a horizontal bar when ghost voxels are not used, as in (b). In (c) we see the result using ghost voxels, where a successful connection is made. We further illustrate this in (d)-(i) and (j)-(o), which show a few representative frames of the result without ghost voxels (a)-(i) and with ghost voxels (j)-(o). The frames (j)-(o) illustrate an issue with blur in the solution using ghost voxels that is not evident from the 3D visualization. Once again the inpainting domain is outlined for reference.

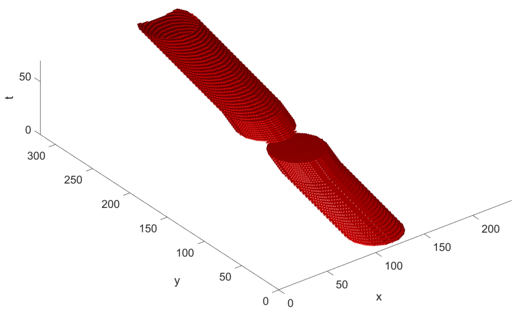
This is also illustrated temporally in Figure 7.5(g)-(l) (no ghost voxels) and Figure 7.5(m)-(r) (ghost voxels) for a few representative frames. When viewed through time, the "kinking" behaviour of the solution without ghost voxels manifests itself as an abrupt change in the speed of the ball, combined with a disintegration and teleportation effect. In this case the ball stops - between frames 428 and 525, as well as between frames 566 and 571, the ball does not move. At the same time between frames 543 and 558 the ball seems to disintegrate and reconstruct itself 36 pixels to the right. On the other hand, with ghost voxels we see that the ball continues at the correct speed but becomes progressively blurrier as we near the center of the inpainting domain before becoming gradually less blurry as we near the opposite side.

Example 2: A fast ball (320px ×240px ×71fr)

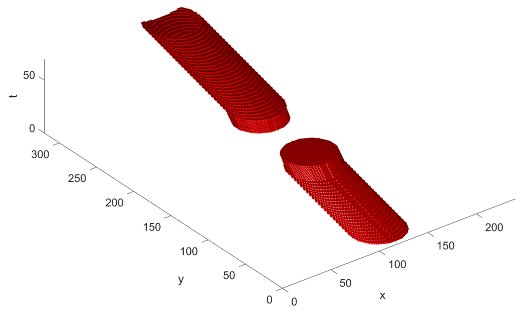
In this example, the same red ball moves left to right, but with a speed of 4 pixels per frame, and now the video is only 71 frames long. We consider two inpainting domains. The first, illustrated in Figure 7.6(a), is the square $[111, 202] \times [79, 166] \subset [1, 320] \times [1, 240]$ appearing in every frame of the video. The second, illustrated in 7.7(a), is the square $[79, 237] \times [59, 177] \subset [1, 320] \times [1, 240]$ which appears from frame 21 to frame 39. The guidance direction in this case is $\mathbf{g} = \frac{1}{\sqrt{17}}(4, 0, 1)$, and $\epsilon = 3h$ (that is, $r = 3$) and $\tilde{\mu} = 40$ as before. In the first case the ball strikes the boundary of the inpainting domain in a place where it is parallel to the yt-plane, making an angle of $\arctan(4) \approx 76°$ with the plane. Once again this results in kinking to 90° relative to the boundary, which manifests itself as a stretching of the ball into a



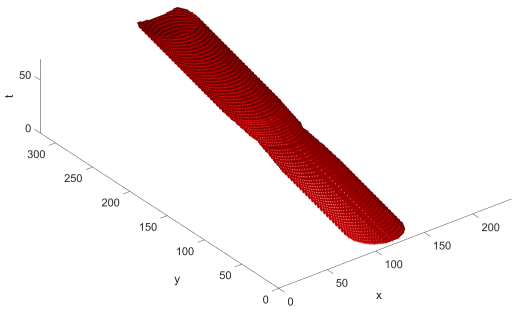
(a) Inpainting problem in Example 2, second inpainting domain.



(b) Result of inpainting without ghost voxels.



(c) Result of inpainting with ghost voxels.

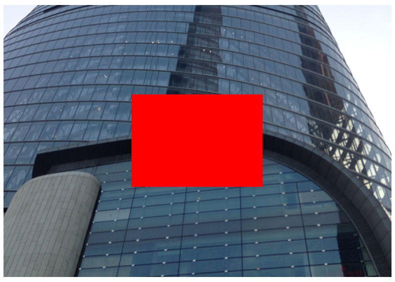


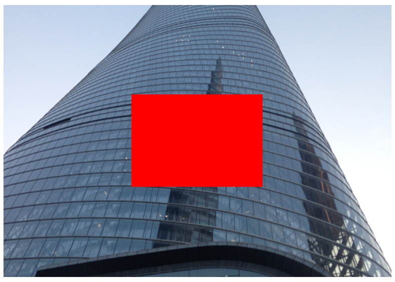
(d) Result of inpainting with ghost voxels and semi-implicit extension.

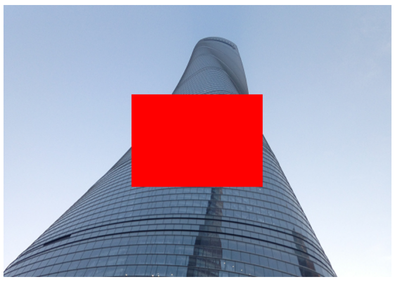
Figure 7.7: **The semi-implicit extension in 3D:** Here we illustrate an example where ghost voxels on their own are not enough to resolve kinking artifacts. This time we illustrate in (a) the inpainting problem from Example 2 with the second inpainting domain, where an occluding rectangle appears for 18 frames before vanishing. In this case the ball makes a very shallow angle with the inpainting domain at its point of entry, so that the direct form of spacetime transport, both without (b) and with ghost voxels (c), fail to produce to the correct extrapolation. Only (d), which uses ghost voxels together with 5 iterations of SOR per shell to solve the linear system arising in the semi-implicit form of spacetime transport is able to produce a satisfactory result.

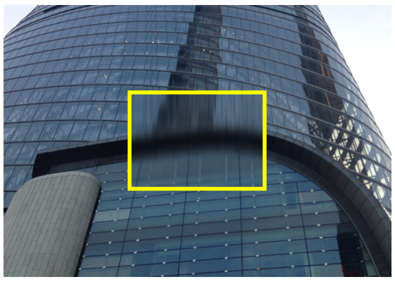
horizontal bar, as illustrated in 3D in Figure 7.6(b), and through time in Figure 7.6(d)-(i). When viewed through time, the ball first stretches into a bar, then disappears (frame 36), before re-emerging as a bar that pinches off into a ball that then continues at its original speed. Once again, the use of ghost voxels alleviates this problem but creates some blur, as shown in 7.6(c) and Figure 7.6(j)-(o).

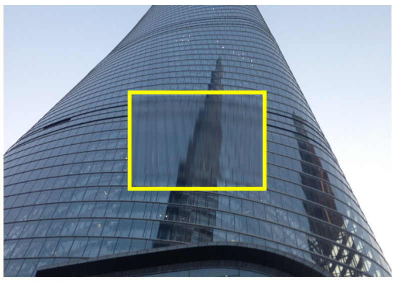
In the second case, the ball now strikes the inpainting domain in a location where it is locally parallel to the xy-plane. This time the angle relative to the plane is $\arctan(\frac{1}{4}) \approx 14°$. Since this angle is smaller than the critical angle $\Delta\theta_3 \approx 35.8°$ at which Guidefill fails for $r = 3$ (Section 6.6.2 and Figure 7.10(b)), in this case we expect the *direct* form of spacetime transport to kink both with and without ghost voxels. Indeed, this is exactly what is observed in Figure 7.7(b) and Figure 7.7(c) - in fact, the result with ghost voxels is worse. However, as in the 2D case, the situation is resolved using the semi-implicit extension. Figure 7.7(d) illustrates the semi-implicit form of spacetime transport using 5 iterations of SOR to approximate the solution to the resulting linear system. As expected, in this case a successful connection is made.

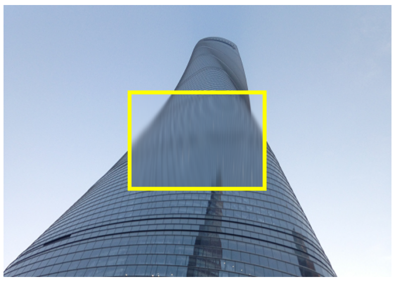

(a) Original video with inpainting domain in red: frame 1.

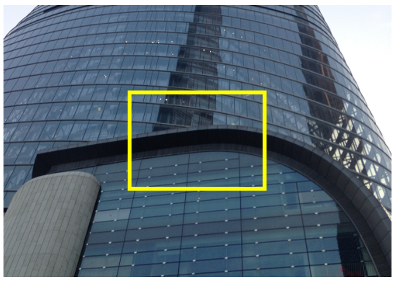

(b) Original video with inpainting domain in red: frame 51.

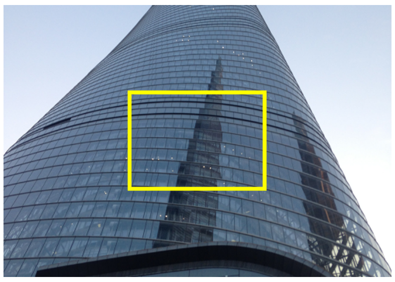

(c) Original video with inpainting domain in red: frame 101.


(d) Inpainted video with $r = 3$ and ghost voxels on: frame 1.


(e) Inpainted video with $r = 3$ and ghost voxels on: frame 51.


(f) Inpainted video with $r = 3$ and ghost voxels on: frame 101.


(g) Inpainted video with $r = 4$ and ghost voxels off: frame 1.


(h) Inpainted video with $r = 4$ and ghost voxels off: frame 51.

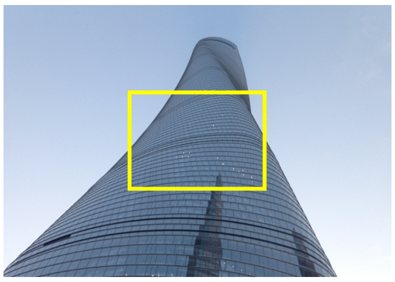
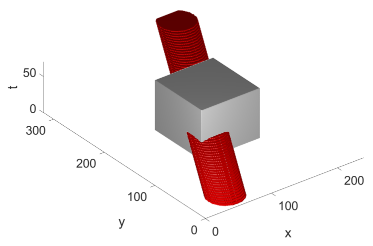

(i) Inpainted video with $r = 4$ and ghost voxels off: frame 101.

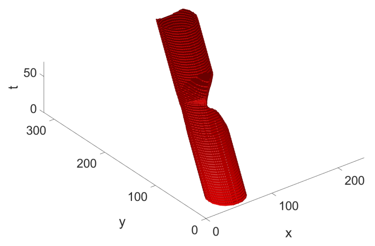
Figure 7.8: **A blurry Shanghai tower:** In this example we illustrate the problem of inpainting a stationary rectangle blocking a video where the camera pans up the Shanghai tower. In (a)-(c), we see the original inpainting problem, with the occluding rectangle. The camera moves up 3 pixels per frame, so $\mathbf{g} = \frac{1}{\sqrt{10}}(0, -3, 1)$. In (d)-(f) we see the results of inpainting using spacetime transport with $G = I - \mathbf{g} \otimes \mathbf{g}$ given, $\tilde{\mu} = 40$, $r = 3$, and ghost voxels turned on. The result is recognizable but suffers from heavy blur. In (g)-(i), the radius is increased to $r = 4$ and ghost voxels are turned off. Since $\mathbf{g}$ (after rescaling by $\sqrt{10}$) belongs to the discrete integer ball of radius 4, in this case we have no kinking artifacts and also no blur. For reference, the inpainting domain is outlined in yellow.

Example 3: The Shanghai tower (500px $\times$350px $\times$101fr)

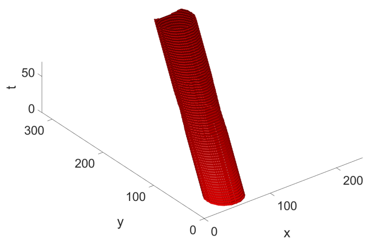
Our third example illustrates not a moving object but a moving camera and is meant to highlight blur artifacts rather than kinking artifacts. In this case the camera pans up the Shanghai tower at a rate of three pixels per frame, so that $\mathbf{g} = \frac{1}{\sqrt{10}}(0, -3, 1)$. The inpainting domain, illustrated in red in Figure 7.8(a)-(c), is a stationary rectangle blocking the middle of video. The result of inpainting with spacetime transport with $r = 3$, $\tilde{\mu} = 40$, and ghost voxels turned on is illustrated in Figure 7.8(d)-(f). In this case the result appears correct but suffers from significant and highly noticeable blur that destroys the texture of the video. In 7.8(g)-(i) we are able to resolve this issue in this case by setting $r = 4$ and turning ghost voxels off. Since $(0, -3, 1) = \sqrt{10}\mathbf{g}$ belongs to the discrete ball of radius 4, the problem is resolved in this case. However, this fix is unlikely to generalize.



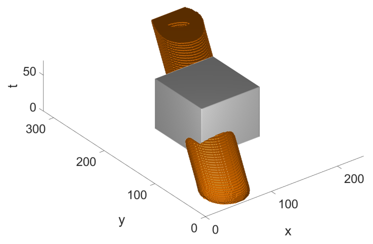
(a) Original video inpainting problem.



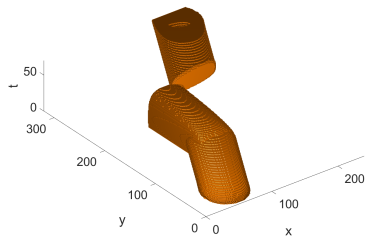
(b) Video inpainting using the inpainted guide field (e).



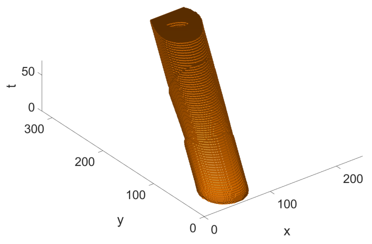
(c) Video inpainting using the inpainted guide field (f).



(d) Auxillary guide inpainting problem.



(e) Inpainting the guide field using one iteration of (7.3.5).



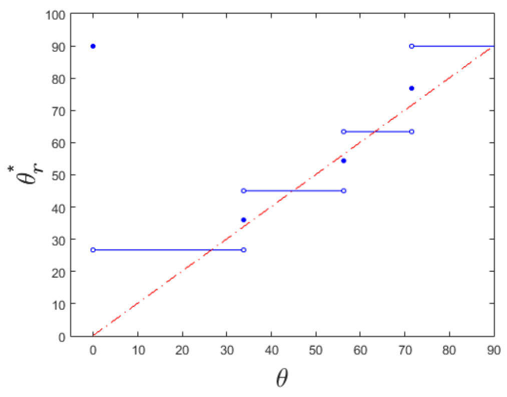
(f) Inpainting the guide field using two iterations of (7.3.5).

Figure 7.9: **Video and guide field inpainting:** In this example, a ball moves diagonally through a video with a square shaped inpainting domain, as shown in (a). Inpainting then proceeds in three steps. First, we compute the exhilary inpainting problem in (d), where we used the undamaged video data in (a) to construct the guidance tensor field on the exterior of the extended inpainting domain (7.3.3) (to create this visualization, the guide field at a voxel $\mathbf{x}$ is first visualized as an RGB triplet as in Figure 7.11 - low intensity voxels are clamped down to black, which is rendered transparent, while high intensity voxels are rendered as a solid). Next, we inpaint the guide field. In (e), we see the result of inpainting the guide field using a single iteration of (7.3.5). In this case the guide field, which should follow the intended direction of motion of the ball, instead make a sharp turn. The third step is the use the inpainted guide field to inpaint the video. When the inpainted guide field (e) is used for video inpainting as in (b), the result is that the inpainted ball has a large "bite" taken out of it. In (f), we now inpaint the guide field using two iterations of (7.3.5). This is a big improvement, and leads in (c) to a much better reconstruction of the video. See also Figure 7.11.

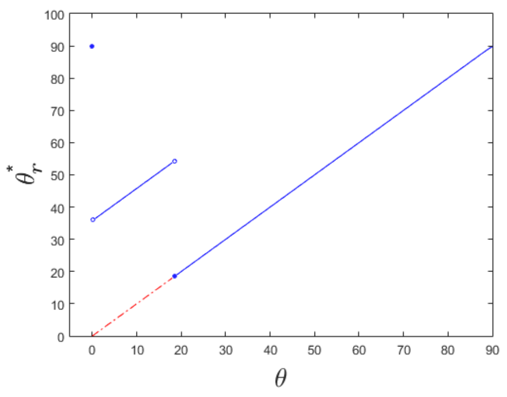
Remark 7.7.1. *The issue of blur experienced by spacetime transport is not unlike that discussed in Section 7.1.2 for optical flow based video inpainting. Since those authors had noted that bilinear interpolation leads to blur, and ghost voxels are based on trilinear interpolation, it does not come as a surprise that spacetime transport suffers from blur. In the future, it would be interesting to investigate alternative definitions of ghost pixels and ghost voxels based on other forms of interpolation. However, not just any interpolation scheme will do - in order to simultaneously ensure that kinking artifacts remain resolved, we need to consider schemes that lead to the same fixed ratio continuum limit as bilinear (trilinear) interpolation. One way of doing this is to consider only interpolation schemes that satisfy the properties of ghost pixels listed in Section 5.1 (or, in the 3D case, the generalization of these properties).*

7.8 Examples including guide field computation

In the previous section we went over kinking and blur artifacts that can occur within spacetime transport even when the method is given ground truth values for the guide field. Having covered that, we now explore the computation of the guide field itself. As discussed in Section 7.3.2, the guide field is computed by first calculating the structure tensor $\mathcal{J}_{\sigma,\rho}$ on the exterior of the extended inpainting domain (7.3.3), and then inpainting using (7.3.5). All examples compute the structure tensor using $\sigma = \rho = 2$px, and fix $r = 4$.



(a) Coherence transport, theoretical curve, $r = 3$.



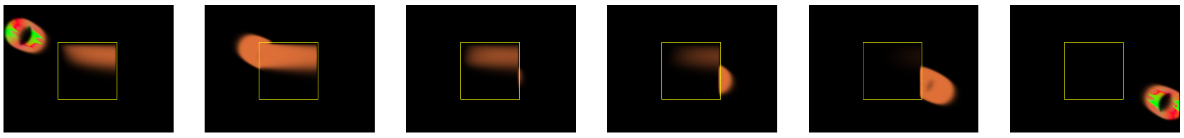
(b) Guidefill, theoretical curve, $r = 3$.

Figure 7.10: **Reprint of Theoretical Curves for coherence transport and Guidefill:** Reprint of the theoretical kinking curves for coherence transport and Guidefill with $r = 3$. Section 7.7 contains various kinking examples for spacetime transport which, although we do not prove this, appear to be governed by this same curves. We place them here again for the reader's convenience.

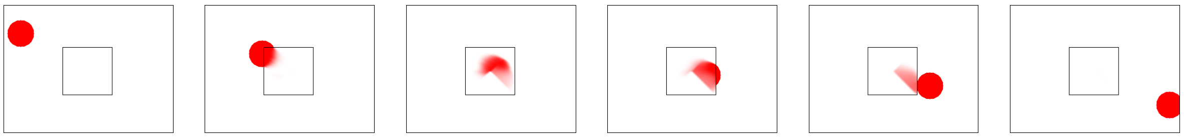
In Figure 7.9 and Figure 7.11, we return to the fast moving ball from Example 2 in the last section. The ball now moves 4px right and 2px down every frame, that is $\mathbf{g} = \frac{1}{\sqrt{17}}(4, -2, 1)$, however, now the guide field is no longer a specified constant, rather it is computed by the fixed point iteration (7.3.5). The convergence of this iteration is illustrated in 7.9(d)-(f) (the guide field is visualized as a solid in this case), where we see respectively the guide field to be inpainted including the extended inpainting domain (7.3.3) in grey, the result of inpainting using one iteration of (7.3.5) (which is shown to be inadequate), and the result of two iterations, which already yields a satisfactory result. This is also illustrated in rows one and three Figure 7.11, where we see representative frames of the inpainted guide field using one and two iterations respectively of (7.3.5). The guide field is color coded so that the RGB triplet of a given voxel $\mathbf{x}$ is parallel to the componentwise absolute value of the minimal eigenvector of $G(\mathbf{x})$, while the magnitude $\|(R, G, B)\|$ is proportional to $\tanh(|\lambda_1 - \lambda_3|/\Lambda_{1,3})$. The corresponding inpainted videos with one and two iterations are shown in rows two and four of Figure 7.11 respectively, as well as Figure 7.9(b)-(c) where they are represented as a solid. When only one iteration of 7.3.5 is used, the inpainted guide field makes a sharp turn immediately upon entering the inpainting domain, resulting in poor reconstruction of the moving ball - see Figure 7.9(b) and 7.11)(g)-(l). However, just one additional iteration yields a dramatic improvement and the resulting inpainted video 7.11)(s)-(x) is already about as good as Example 2 in the last section, where ground truth values for the guide field were provided.

Remark 7.8.1. *In Section 4.3.1, we discussed inaccuracies in the structure tensor that arise when it is computed too close to the inpainting domain, leading us to define the extended inpainting domain. Similar inaccuracies arise when the structure tensor is computed within* $\max(4\rho, 4\sigma)$ *voxels of the boundary of the video domain - this is visible in Figure 7.11(a) and Figure 7.11(f), as well as 7.11(m) and Figure 7.11(r). Really, the extended inpainting domain should also include all voxels within distance* $\max(4\sigma, 4\rho)$ *of the video domain boundary - if the guide field is needed in these areas, it should be inpainted. This issue is more significant in the next example.*

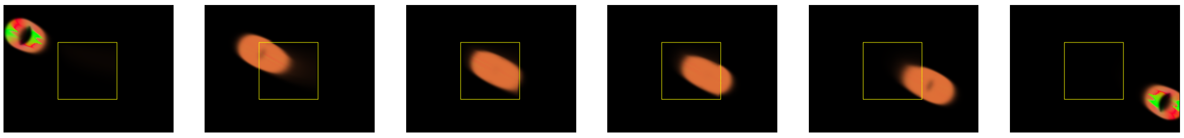
Next, we return to the example of the Shanghai tower, first with the original rectangular inpainting domain appearing in every frame (Figure 7.12), then later with a modified version that only appears



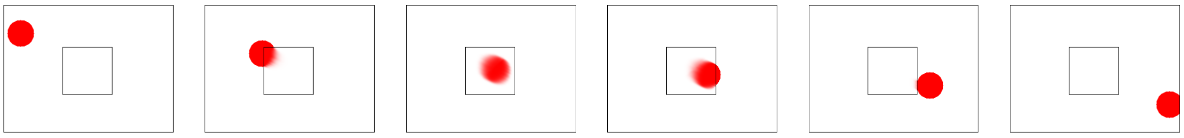
(a) Inpainted guide field ($k = 1$): Frame 2. (b) Inpainted guide field ($k = 1$): Frame 21. (c) Inpainted guide field ($k = 1$): Frame 36. (d) Inpainted guide field ($k = 1$): Frame 41. (e) Inpainted guide field ($k = 1$): Frame 51. (f) Inpainted guide field ($k = 1$): Frame 69.



(g) Inpainted video ($k = 1$): Frame 2. (h) Inpainted video ($k = 1$): Frame 21. (i) Inpainted video ($k = 1$): Frame 36. (j) Inpainted video ($k = 1$): Frame 41. (k) Inpainted video ($k = 1$): Frame 51. (l) Inpainted video ($k = 1$): Frame 69.



(m) Inpainted guide field ($k = 2$): Frame 2. (n) Inpainted guide field ($k = 2$): Frame 21. (o) Inpainted guide field ($k = 2$): Frame 36. (p) Inpainted guide field ($k = 2$): Frame 41. (q) Inpainted guide field ($k = 2$): Frame 51. (r) Inpainted guide field ($k = 2$): Frame 69.



(s) Inpainted video ($k = 2$): Frame 2. (t) Inpainted video ($k = 2$): Frame 21. (u) Inpainted video ($k = 2$): Frame 36. (v) Inpainted video ($k = 2$): Frame 41. (w) Inpainted video ($k = 2$): Frame 51. (x) Inpainted video ($k = 2$): Frame 69.

Figure 7.11: **Video and guide inpainting viewed temporally:** Here we provide further illustration of the example in Figure 7.9 with a selection of frames from both the inpainted guide field and the inpainted video, using $k = 1$ and $k = 2$ iterations of (7.3.5). The guide field is color coded so that the RGB triplet of a given voxel $\mathbf{x}$ is parallel to the componentwise absolute value of the minimal eigenvector of $G(\mathbf{x})$, while the magnitude $\|(R, G, B)\|$ is proportional to $\tanh(|\lambda_1 - \lambda_3|/\Lambda_{1,3})$. The first and second rows show the inpainted guide field and video respectively when one iteration of (7.3.5) is used in the construction of the guide field. The third and fourth rows do the same but now two iterations of (7.3.5) are used. For reference, the inpainting domain is outlined in black in the video examples, and the extended inpainting domain is outlined in yellow in the guide field inpainting examples.

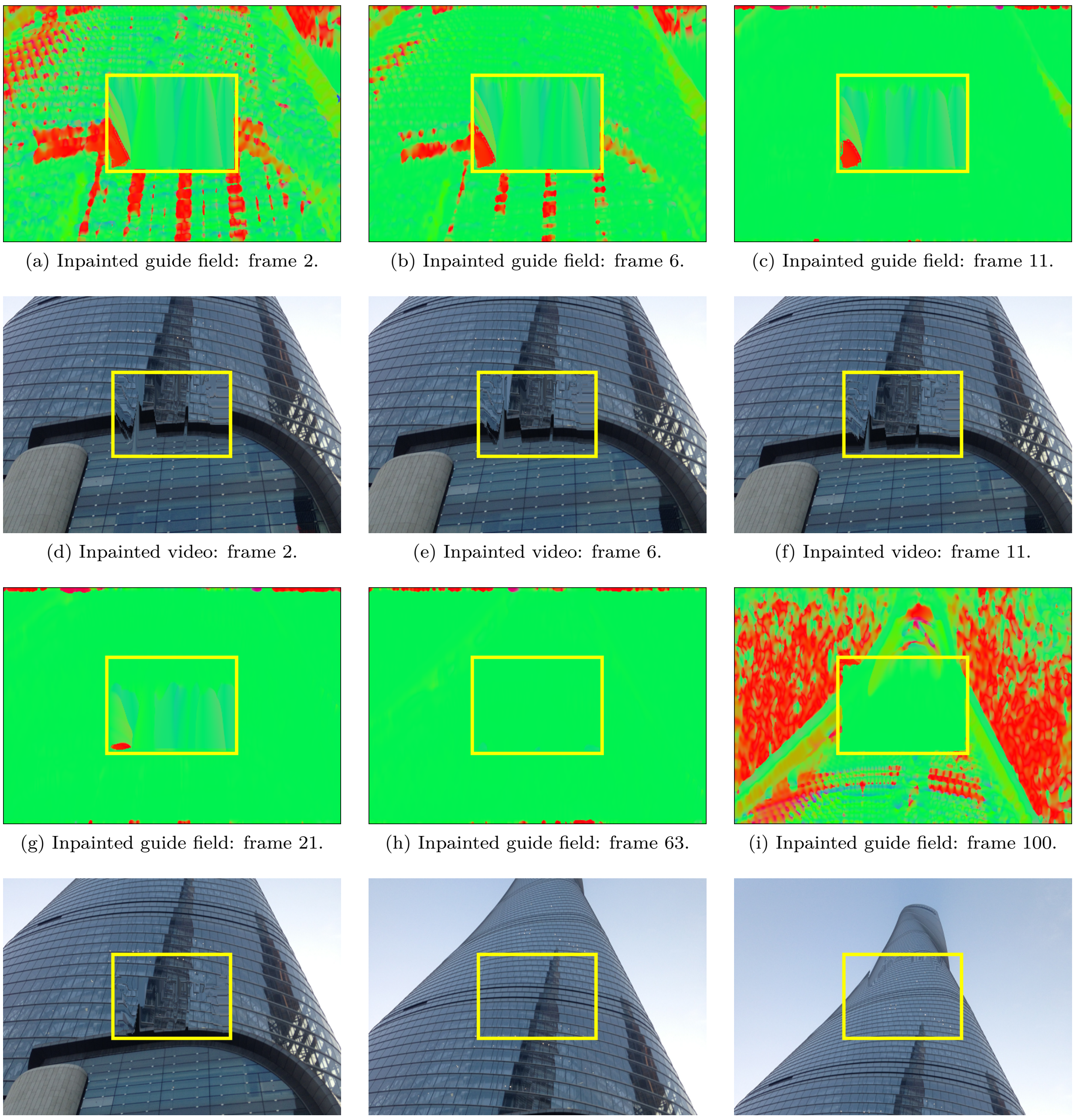
from frame twenty onwards (Figure 7.13). In this case we illustrate the guide field by representing the normalized minimal eigenvector of the guidance tensor as an RBG triplet (taking its componentwise absolute value as negative color values are not defined). We know the ground truth motion camera motion (down by three pixels per frame), and hence the correct answer is a bluish green with three parts green to one part blue. However, as noted in Remark 7.8.1, when the structure tensor is computed fewer than $\max(4\sigma, 4\rho)$ voxels from $\partial\Omega_h$, the structure tensor is inaccurate. These inaccuracies in the guide field are clearly visible in 7.12(a). On the exterior of the inpainting domain these have settled down by about frame eleven, where we see a nearly pure shade of bluish green with (R,G,B) = (0,242,80), which is what we want. However, in the first example where the inpainting domain stretches all the way down to frame one, these first eleven frames contain inaccuracies that are then inpainted, meaning that the first eleven frames of the inpainting domain are full of garbage values. These garbage values are then propagated upwards through the inpainting domain, not fully disappearing until frame 61. Thus, in this case, the inpainted solution contains artifacts all the way up until frame 61. Beyond that, the result appears the same as in Figure 7.8(g)-(i), at least up until the final frames where we once again encounter inaccuracies in the guide field because we are within nine voxels of $\partial\Omega_h$. As illustrated in Figure 7.13, in the second case where the first twenty frames are unoccluded, this problem is largely alleviated. The exception is the final few frames, for the reasons we have just discussed. To avoid blur artifacts, in this example we used ghost voxels when inpainting the guide field but disabled them when inpainting the video itself.

7.9 Conclusions

This chapter introduced spacetime transport, a natural generalization of the ideas of coherence transport and Guidefill to three dimensions (two space plus one time). Spacetime transport is a kind of hybrid between shell-based image inpainting and a form of optical flow based video inpainting, possessing features of both classes of algorithms. For example, we have seen experimentally in Section 7.7 that kinking artifacts present in shell-based image inpainting, as well as their resolution based on ghost pixels and the semi-implicit extension, carry over into three dimensions essentially unchanged (although we have not proved this). Blur artifacts also carry over, but can now be seen as fitting into a bigger picture as a known problem with optical flow based inpainting (Section 7.1.2). At the same time, when the minimal eigenvector $\mathbf{v}_3$ of the guidance tensor has a nonzero timelike component, spacetime transport is essentially equivalent to a form of optical flow based video inpainting. However, unlike optical flow based inpainting, $\mathbf{v}_3$ may be entirely spatial, and this means that spacetime transport can be thought of as inpainting based on a type of generalized optical flow that doesn't distinguish between time and space.

However, spacetime transport is still very much a work in progress. In its present form, it suffers from serious blur artifacts that render traditional optical flow based algorithms a better choice. This needs to be fixed, and we have already mentioned one possible strategy in Remark 7.7.1. It is also unclear if the current method for inpainting the guide field is the best one. The alternative that we mentioned in passing based on animated splines might be better. But even if we do stick with this strategy, more work needs to be done, especially in terms of voxel ordering strategies. Another issue that remains undecided is the best way to parallelize the algorithm (in whole or in part) as discussed in Section 7.5.

It is unclear whether the way we adapt our weights in the case $\lambda_1 \gg \lambda_2 \approx \lambda_3$ has any benefit. In this case it seems evident that no preference should be given to the direction $\mathbf{v}_2$ over the direction $\mathbf{v}_3$, and so we bias weights in favor of the plane $\Pi_{\mathbf{v}_2,\mathbf{v}_3}$ as opposed to the line $L_{\mathbf{v}_3}$. However, despite exploring relevant examples such as a line or square in uniform motion (where the isosurfaces of u_h are planes), we have yet to see any significant impact of this feature. In the future it may be scrapped.



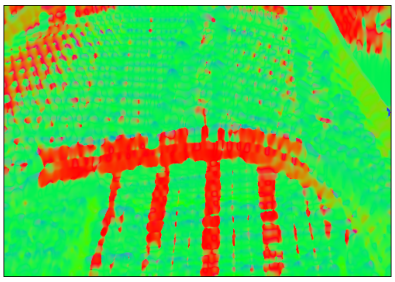
(a) Inpainted guide field: frame 2. (b) Inpainted guide field: frame 6. (c) Inpainted guide field: frame 11.

(d) Inpainted video: frame 2. (e) Inpainted video: frame 6. (f) Inpainted video: frame 11.

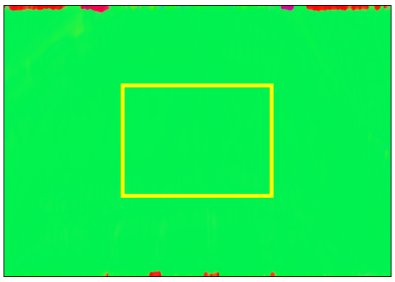
(g) Inpainted guide field: frame 21. (h) Inpainted guide field: frame 63. (i) Inpainted guide field: frame 100.

(j) Inpainted video: frame 21. (k) Inpainted video: frame 63. (l) Inpainted video: frame 100.

Figure 7.12: **Boundary issues with the structure tensor:** Here we redo the inpainting problem from Example 3 of Section 7.7 (illustrated in Figure 7.8), but calculate the guide field numerically rather than using a provided ground truth value. The guide field at a given pixel $\mathbf{x}$ in this case is visualized as an RBG triplet equal to the componentwise absolute value of the normalized minimal eigenvector of $G(\mathbf{x})$. The inpainted guide field is presented for a selection of frames in rows one and three, while the corresponding inpainted video is presented in rows two and four. In this case we expect the guide field to appear pure bluish green - three parts green to one part blue, as the camera moves down by three pixels per frame. This is more or less true on the exterior of the extended inpainting domain (highlighted in yellow) by about frame ten, but the earlier frames contain inaccuracies due to their proximity to $\partial\Omega_h$, as discussed in Remark 7.8.1. In this example the inaccuracies in the early frames are inpainted and then propagated into later frames, only fully disappearing by frame 63. The result is artifacts in the inpainted video (inpainting domain highlighted in yellow) prior to frame 63. These artifacts disappear after frame 63 but reemerge in the final few frames as we are once again too close to $\partial\Omega_h$ for the structure tensor to be computed accurately. To avoid blur artifacts we have used ghost voxels when inpainting the guide field but disabled them when inpainting the video itself.

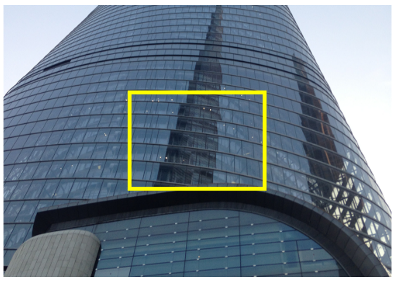

(a) Inpainted guide field: frame 2.

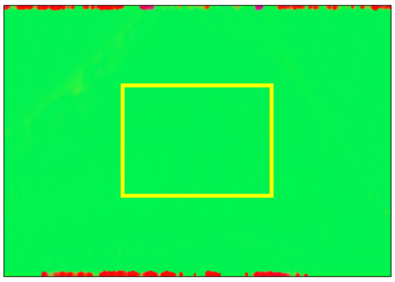
(b) Inpainted guide field: frame 11.

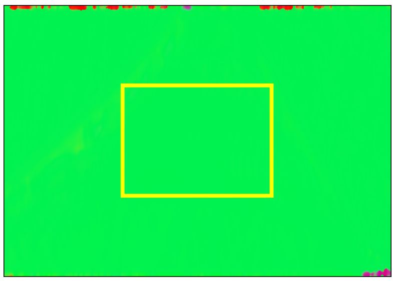

(c) Inpainted guide field: frame 26.

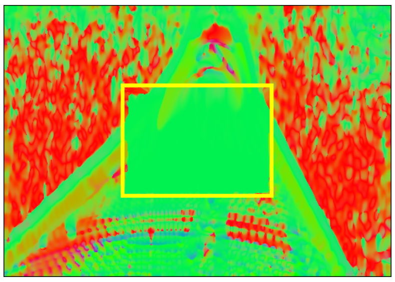
(d) Inpainted video: frame 2.

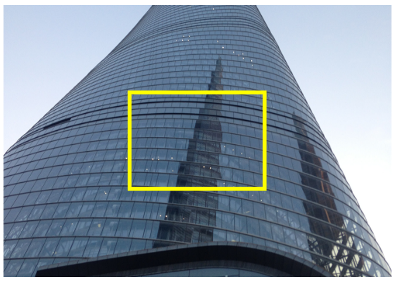
(e) Inpainted video: frame 11.

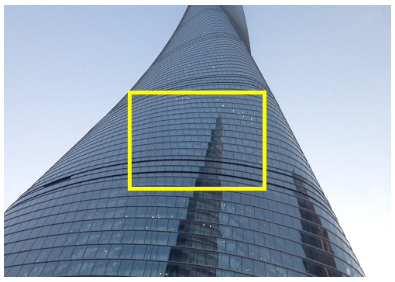

(f) Inpainted video: frame 26.


(g) Inpainted guide field: frame 51.


(h) Inpainted guide field: frame 76.


(i) Inpainted guide field: frame 100.


(j) Inpainted video: frame 51.


(k) Inpainted video: frame 76.

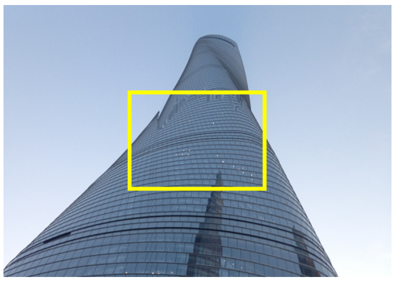

(l) Inpainted video: frame 100.

Figure 7.13: **Alleviating boundary issues with a new inpainting domain:** Here we repeat the experiment in Figure 7.12, but modify the inpainting domain so that it doesn't appear until frame 20. This prevents the propagation of garbage values present in the guide field in earlier frames from propagating into later frames as we saw in Figure 7.12. In this case the inpainted solution is indistinguishable from the one in Figure 7.8, where ground truth values for $G(\mathbf{x})$ were provided, at least up until the final few frames where we once again encounter problems. In practice, one does not get to choose the inpainting domain, so a better solution is to avoid calculating the structure tensor close to $\partial\Omega_h$ and inpaint this area instead. As in the last example, to avoid blur artifacts we have used ghost voxels when inpainting the guide field but disabled them when inpainting the video itself.

Finally, there is significant analytical work left to do regarding spacetime transport. First, the analysis of Chapter 6 should be generalized to this setting. This is unlikely to be difficult but it is worth doing. More interesting, and potentially more challenging, would be to explore analytically the effect of different forms of interpolation in the definition of ghost voxels, and to analytically prove the convergence properties of our fixed point iteration for computing the guide field.

Chapter 8

Summary and Outlook

This thesis has been about two things - the design of specific inpainting algorithms within a framework, and the analysis of the framework itself. On the design side of things, we have operated both in two dimensions with Guidefill in Chapter 4, and three dimensions with spacetime transport in Chapter 7. Analytically, we have limited ourselves to the 2D case (Chapter 6), but we have provided evidence that much of our analysis carries over into three dimensions (Section 7.7).

In this final chapter, we will begin in Section 8.1 with a review of the three main parts of the thesis and give a brief description of potential future research directions. Then in Section 8.2 we will single out what we consider to be the three most interesting directions for future research and discuss them in more detail.

8.1 Thesis review and directions for future research

Part I: The first part of this thesis, after introducing the problem in Chapter 2, was concerned with Guidefill. Guidefill, as we have seen, is an algorithm for solving the inpainting problem arising in the 3D conversion of film. It was designed for a specific 3D conversion pipeline used in film, which we introduced in Chapter 3. I designed it while working at Gener8, a company specializing in the 3D conversion of Hollywood films. Guidefill is influenced by coherence transport [12, 44], but improves upon it while at the same time adapting to the unique needs of inpainting for 3D conversion. Guidefill has a number of strengths. First, it corrects two types of kinking artifacts present in coherence transport. In one case this is accomplished through the introduction of ghost pixels, in the other, by eliminating the use of the modified structure tensor (Section 4.3.1). Second, it puts the artist in the loop with adjustable splines controlling the output of the algorithm. Third, it has fast execution times enabled by parallelizing the algorithm on the GPU and using a novel boundary tracking algorithm to efficiently allocate GPU processors to pixels. However, Guidefill also has a serious drawback from the point of view of temporal stability, since video frames are inpainted independently. This flaw was the motivation behind spacetime transport, which was discussed near the end of the thesis in Chapter 7. Spacetime transport (which is still a work in progress) has the potential to improve upon Guidefill in terms of temporal coherence and also has a natural connection to optical flow. This connection enables it to take advantage of situations such as inpainting a moving object where the information occluded in one frame is available in another. However, as discussed in Section 7.5, it comes with its own challenges in terms of an efficient GPU implementation.

In the future, it would be interesting to explore a “temporally smoothed” Guidefill that is a compromise between Guidefill and spacetime transport. In this algorithm, frames would still be inpainted one at a time, but they would be sent to the GPU in packets in such a way that the current frame always has a "cushion" of frames on either side. This cushion would need to be big enough that the 3D solid ball $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ is always "full" for each pixel $\mathbf{x}$ in the current frame. That is, the cushion needs to be wide enough that the color of every voxel in $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ is defined. The current frame would be inpainted in exactly the same way as in Guidefill, with the guide field still computed using only information present in the current frame, but the averaging would now be done over the 3D ball $\tilde{\mathbf{B}}_{\epsilon,h}(\mathbf{x})$ that includes pixel values from neighboring frames, rather than the 2D ball $\tilde{B}_{\epsilon,h}(\mathbf{x})$, that only includes values from the current frame. Such an algorithm would not have the connection to optical flow that spacetime transport enjoys, but would gain improved temporal coherence at minimal additional cost. This avoids some of the implementation challenges of spacetime transport and may be faster and good enough for certain cases. Further improvements to temporal stability could be gained via some procedure for enforcing temporal continuity into the guide field itself (as opposed to just the video derived from it). This could be as simple as replacing the guide field in a given frame with a weighted average of it plus the guide field in previous frames, but there are many possibilities to explore. Finally, another idea would be to combine one or both of these with a procedure whereby instead of computing splines independently for every frame, they are instead calculated for a small number of user specified frames and then propagated to neighboring frames using optical flow or some other method. As we have already stated in Chapter 7 it would also be interesting to explore something like this in the context of spacetime transport.

Part II: The second part of this thesis was concerned with analysis. This began as an attempt to understand the origins of the kinking behaviour of coherence transport, and why the use of ghost pixels in Guidefill was able largely to correct it. This initial goal was accomplished by analyzing the framework considered in this thesis using the fixed ratio continuum limit proposed in Section 2.4, as opposed to Bornemann and März's high-resolution vanishing viscosity limit, as had been done previously. Later it was noticed that Guidefill also produced kinking artifacts when the angle between the guide direction $\mathbf{g}$ and the inpainting domain boundary is small. The semi-implicit extension was introduced to correct this, and the fixed ratio limit was able to explain the difference in behaviour between the direct and semi-implicit forms of different algorithms. In particular, it was proven in Section 6.6.1 that the direct form of *any* method within the inpainting framework considered must exhibit kinking artifacts. At the same time, it was proven that the semi-implicit form of Guidefill only kinks if the guidance direction $\mathbf{g}$ is exactly parallel to the boundary of the inpainting domain (at least under the simplifying assumptions under which our analysis operates). Interestingly, this prediction is exactly the same as the one that the high-resolution vanishing viscosity limit makes for coherence transport.

The analysis of the fixed ratio limit was originally done assuming smooth boundary data, as we have done in Theorem 4.7.1. However, as this assumption is far from realistic when it comes to images, this analysis was later generalized in Theorem 6.3.1 to include boundary data with very low regularity, such as boundary data with jump discontinuities or boundary data that is nowhere differentiable. As a result, Theorem 6.3.1 is by far the most technically challenging piece of the thesis, with a lengthy proof broken up into several sections. Since Taylor series (the main tool in the proof of Theorem 4.7.1) were not available, the proof was instead accomplished by exploiting a connection to stopped random walks. Theorem 6.3.1 not only proves convergence but also gives convergence rates, which depend on both the regularity of the boundary data and the choice of L^p norm used to measure to convergence. This result is interesting in and of itself, and it also hints at the existence of blur artifacts, which are the other major topic of our analysis. However, to analyze blur artifacts fully, we need to use the asymptotic limit, which also comes from the connection to stopped random walks. Unfortunately, for technical reasons at present, we are unable to prove convergence to the asymptotic limit. Although convergence is supported by strong numerical evidence in Section 6.7, this is left as a conjecture (Conjecture 6.7.1) for now. Proving this conjecture is at the top of the agenda for future research, however, there are also a number of other questions we would like to explore:

1. Does there exist an algorithm within the framework we have described that avoids blur artifacts and kinking artifacts at the same time? If not, is it at least possible to design an algorithm that, like semi-implicit Guidefill, avoids kinking artifacts so long as the guidance direction $\mathbf{g} = (\cos\theta, \sin\theta)$ obeys $\theta \neq 0$, but for which the severity of blur as a function of θ remains bounded? Could this be done, for example, by replacing the bilinear interpolation used to define ghost pixels with a more sophisticated interpolation scheme?

2. While we have already shown in Proposition 6.2.1 that SOR (with an appropriate ordering of unknowns) is particularly effective for solving the linear system arising in semi-implicit Guidefill, is this generically true regarding the semi-implicit extension of *any* inpainting method within the class under investigation? Moreover, SOR is not ideal because it is sequential whereas Guidefill was designed to be a parallel algorithm. Does another method exist which maintains the fast convergence rate of SOR, but can be implemented in parallel? One possibility is the scheduled Jacobi method [4].

3. Is our semi-implicit extension of Algorithm 1 the best way of avoiding kinking artifacts? In our analysis, we have assumed that the radius of our averaging neighborhood remains fixed. Another possibility, based on the observation that as r increases, (direct) Guidefill can successfully extrapolate shallower and shallower angles (Section 6.6.2), is to use the direct form of Guidefill but dynamically resize $\tilde{B}_{\epsilon,h}(\mathbf{x})$ as needed.

4. What happens if the semi-implicit version of Algorithm 1 is generalized to a fully-implicit extension in which pixel colors are computed as a weighted average not only of their (known) already filled neighbors in $A_{\epsilon,h}(\mathbf{x}) \cap (\Omega \backslash D^{(k)})$ and (unknown) neighbors within the same shell $\partial D_h^{(k)}$, but of *all* neighbors in $A_{\epsilon,h}(\mathbf{x})$? Are there additional benefits in terms of artifact reduction and, if so, can the resulting linear system be solved efficiently enough to make this worthwhile?

Part III: The last part of the thesis is concerned with spacetime transport, a 3D generalization of Guidefill. Spacetime transport is based on filling the inpainting domain in concentric 3D shells of voxels, assigning colors to voxels based on a weighted average over a 3D rotated ball of ghost voxels. The weights are calculated based on a guide field of 3×3 symmetric positive semi-definite (s.p.s.d.) matrices that are computed by first computing the 3D structure tensor (which is s.p.s.d.) outside of the extended inpainting domain (7.3.3), and then extending it inside by inpainting (using an inpainting method that preserves the s.p.s.d. property). The behaviour of the weights at a given voxel depends on the spectrum of the guidance tensor at that voxel. When all eigenvalues are of a similar size, the weights are radially symmetric. If there is one large eigenvalue and two smaller, nearly equal eigenvalues, then the weights are biased in favor of the plane spanned by the eigenvectors corresponding to these smaller eigenvalues. Finally, if there is a unique minimal eigenvalue much smaller than the other two, then the weights are biased in favor of the line defined by the minimal eigenvector. It is in this last case that spacetime transport is connected to optical flow, since as we show in Section 7.6, the minimal eigenvector of the 3D structure tensor is related to a kind of generalized Lucas–Kanade optical flow. However, we also argue in Section 7.1.1 that shell-based image inpainting and optical flow based video inpainting are connected as well.

In the future it would be interesting to explore the connection between shell-based inpainting and optical flow based video inpainting more deeply. In particular, it would be interesting to see whether or not an exchange of ideas could be fruitful. For example, in the context of optical flow based video inpainting, it is known that interpolation schemes that are more sophisticated than bilinear interpolation leads to a reduction in blur artifacts. Could this idea be applied to ghost pixels, to reduce the blur not only in spacetime transport, but also in Guidefill? Going in the other direction, it would be interesting to see whether or not the analytical framework we have developed in Chapter 6 for shell-based image inpainting could be generalized to incorporate optical flow based video inpainting, and whether this could lead to new insights. At the same time, it would be interesting to apply our procedure for inpainting the guide tensor field based on a fixed point iteration to the inpainting of optical flow fields and to see if there are any advantages to this. In particular, it would be interesting to see how it compares to approaches such as the one in [62] which inpaints the optical flow using Telea's algorithm [64].

8.2 Details on the most promising future research directions

We would like to conclude this thesis with a list of what we consider to be the three most promising directions for future research, including a sketch of an action plan for each. These are listed in order according to how important we perceive them to be, and according to the order in which we plan to attack them.

1. Develop the asymptotic limit further and use it to create a unified framework for analyzing blur: First, Conjecture 6.7.1 should be proven (or disproven, as the case may be). Assuming it is true, after it has been proven Conjecture 6.7.1 should be generalized to three dimensions so that it can be applied to spacetime transport and other 3D shell-based inpainting algorithms. Ideally, it should also be applicable to some of the schemes used in optical flow based video inpainting. Once this is done, the asymptotic limit should be used to systematically search for an interpolation strategy that minimizes blur. In the context of Guidefill and spacetime transport, this would amount to seeking alternative definitions of ghost pixels (voxels) based on other interpolation strategies. Hopefully, a similar approach could be used to find good interpolants for optical flow based inpainting. However, as many higher order interpolation strategies use negative weights, it would be ideal if a method could be found for getting past the current restriction of the asymptotic limit to non-negative weights summing to 1. However, since eliminating this restriction would break the connection between Algorithm 1 and stopped random walks upon which the asymptotic limit is based, it is not immediately clear how to go about this.

2. Complete spacetime transport: This action item is listed second because I believe the previous item has to be completed first (at least in part). This is because spacetime transport in its current form suffers from serious blur artifacts that must be corrected before it can be a viable method. Aside from this, the fixed ratio limit should be generalized to 3D so that it can be used to study spacetime transport from a theoretical point of view. A theoretical justification should also be given for the fixed point iteration used to inpaint the guide field, and alternative inpainting strategies should be explored. Experimentation needs to be done regarding the best GPU implementation strategy. Besides the strategy sketched in Section 7.5 based on sweeping through the video (potentially multiiple times) along the different coordinate directions, another strategy would be to send the video to the GPU shell by shell, with only the current shell and those it depends on inside the GPU at any given time. Due to the potentially irregular shape of the shells, some preprocessing might need to be done before they are sent to the GPU, such as first "flattening" each shell into an image. Spacetime transport should be developed in parallel with related strategies such as "temporally smoothed Guidefill" discussed above in Section 8.1, and published together.

3. Develop and study a fully implicit form of Algorithm 1: Currently, our framework computes the color of a given pixel (voxel) $\mathbf{x}$ on the inpainting domain boundary as a weighted average of either its already filled neighbors within $A_{\epsilon,h}(\mathbf{x})$ (or $\mathbf{A}_{\epsilon,h}(\mathbf{x})$ in 3D), as in the direct form of the method, or these neighbors plus additional unknown neighbors within the same shell in the semi-implicit form. A third option should be added to this in which the color of $\mathbf{x}$ is computed as a weighted average of *all* neighbors in $A_{\epsilon,h}(\mathbf{x})$ (or $\mathbf{A}_{\epsilon,h}(\mathbf{x})$). This option would be called the fully implicit form and would involve a linear system coupling together not just those pixels (voxels) within the current shell, but all pixels (voxels) within the inpainting domain. The fixed ratio continuum limit of this new form of the algorithm should be derived, and its properties studied. Since the coupling structure of the unknowns now allows information to travel in all directions, rather than just from the boundary inwards, a possible advantage of this approach is the elimination of shock artifacts. Numerical experiments should be done to see whether or not this is true, and if other advantages exist. If advantages exist and are sufficiently compelling, an efficient numerical solution should be sought. Whether or not the gains are ultimately worth the additional computational cost, this last piece of the puzzle is needed to complete the story of what is possible within this framework.

Bibliography

[1] URL https://www.gener8.com/projects/.

[2] URL http://www.geforce.co.uk/whats-new/articles/nvidia-geforce-gtx-titan-x.

[3] A.R. Abraham, A.K. Prabhavathy, and J.D. Shree. Article: A survey on video inpainting. *International Journal of Computer Applications*, 56(9):43–47, October 2012.

[4] J.E. Adsuara, I. Cordero-Carrión, P. Cerdá-Durán, and M.A. Aloy. Scheduled relaxation jacobi method: Improvements and applications. *Journal of Computational Physics*, 321:369 – 413, 2016. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/j.jcp.2016.05.053. URL http://www.sciencedirect.com/science/article/pii/S002199911630198X.

[5] P. Arias, G. Facciolo, V. Caselles, and G. Sapiro. A variational framework for exemplar-based image inpainting. *Int. J. Comput. Vision*, 93(3):319–347, July 2011. ISSN 0920-5691. doi: 10.1007/s11263-010-0418-7. URL http://dx.doi.org/10.1007/s11263-010-0418-7.

[6] C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera. Filling-in by joint interpolation of vector fields and gray levels. *IEEE Transactions on Image Processing*, 10(8):1200–1211, Aug 2001. ISSN 1057-7149. doi: 10.1109/83.935036.

[7] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.

[8] S.S. Beauchemin and J.L. Barron. The computation of optical flow. *ACM Comput. Surv.*, 27(3): 433–466, September 1995. ISSN 0360-0300. doi: 10.1145/212094.212141. URL http://doi.acm.org/10.1145/212094.212141.

[9] J.V. Beck. The mollification method and the numerical solution of ill-posed problems (diego a. murio). *SIAM Review*, 36(3):502–503, 1994. doi: 10.1137/1036117. URL https://doi.org/10.1137/1036117.

[10] M. Bertalmío. *Image Processing for Cinema*. Chapman & Hall/CRC Mathematical and Computational Imaging Sciences Series. CRC Press - Taylor & Francis, 1st edition, 2014. URL http://www.amazon.com/Processing-Chapman-Mathematical-Computational-Sciences/dp/1439899274/.

[11] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424. ACM Press/Addison-Wesley Publishing Co., 2000.

[12] F. Bornemann and T. März. Fast image inpainting based on coherence transport. *Journal of Mathematical Imaging and Vision*, 28(3):259–278, 2007. ISSN 0924-9907. doi: 10.1007/s10851-007-0017-6. URL http://dx.doi.org/10.1007/s10851-007-0017-6.

[13] M. Burger, L. He, and C. Schönlieb. Cahn-hilliard inpainting and a generalization for grayvalue images. *SIAM J. Imaging Sci.*, 2(4):1129–1167, 2009.

[14] P. Buyssens, M. Daisy, D. Tschumperlé, and O. Lézoray. Exemplar-based inpainting: Technical review and new heuristics for better geometric reconstructions. *IEEE Trans. Image Processing*, 24(6):1809–1824, 2015. URL `http://dblp.uni-trier.de/db/journals/tip/tip24.html#BuyssensDTL15`.

[15] P. Buyssens, O.L. Meur, M. Daisy, D. Tschumperlé, and O. Lézoray. Depth-guided disocclusion inpainting of synthesized rgb-d images. *IEEE Transactions on Image Processing*, 26(2):525–538, Feb 2017. ISSN 1057-7149. doi: 10.1109/TIP.2016.2619263.

[16] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986.

[17] F. Cao, Y. Gousseau, S. Masnou, and P. Pérez. Geometrically guided exemplar-based inpainting. *SIAM J. Img. Sci.*, 4(4):1143–1179, December 2011. ISSN 1936-4954. doi: 10.1137/110823572.

[18] T. Chan and J. Shen. Variational image inpainting. *Communications on pure and applied mathematics*, 58(5):579–619, 2005.

[19] T. Chan, S. Kang, and J. Shen. Euler's elastica and curvature-based inpainting. *SIAM Journal on Applied Mathematics*, pages 564–592, 2002.

[20] S. Choi, B. Ham, and K. Sohn. Space-Time Hole Filling With Random Walks in View Extrapolation for 3D Video. *IEEE Transactions on Image Processing*, 22:2429–2441, June 2013. doi: 10.1109/TIP.2013.2251646.

[21] J. Cilleruelo. The distribution of the lattice points on circles. *Journal of Number Theory*, 43(2): 198 – 202, 1993. ISSN 0022-314X. doi: http://dx.doi.org/10.1006/jnth.1993.1017. URL `http://www.sciencedirect.com/science/article/pii/S0022314X83710176`.

[22] A. Criminisi, P. Pérez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 13:1200–1212, 2004.

[23] I. Daribo and B. Pesquet-Popescu. Depth-aided image inpainting for novel view synthesis. In *2010 IEEE International Workshop on Multimedia Signal Processing*, pages 167–170, Oct 2010. doi: 10.1109/MMSP.2010.5662013.

[24] V. Derr and D. Kinzebulatov. On the extension of schwartz distributions to the space of discontinuous test functions of several variables. *Rocky Mountain J. Math.*, 39(4):1173–1193, 08 2009. doi: 10.1216/RMJ-2009-39-4-1173. URL `http://dx.doi.org/10.1216/RMJ-2009-39-4-1173`.

[25] T. Dobbert. *Matchmoving: The Invisible Art of Camera Tracking.* Sybex, February 2005. ISBN 0782144039. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0782144039`.

[26] domotorp (http://mathoverflow.net/users/955/domotorp). Conjecture regarding closest point inside a discrete ball to a line. MathOverflow. URL `http://mathoverflow.net/q/187830`. URL:http://mathoverflow.net/q/187830 (version: 2014-11-22).

[27] P. Erdös and R.R. Hall. On the angular distribution of gaussian integers with fixed norm. *Discrete Mathematics*, 200(1–3):87 – 94, 1999. ISSN 0012-365X. doi: http://dx.doi.org/10.1016/S0012-365X(98)00329-X. URL `http://www.sciencedirect.com/science/article/pii/S0012365X9800329X`.

[28] S. Zinger ; P. De With G. Bravo. Gpu-accelerated real-time free-viewpoint dibr for 3dtv. *IEEE Transactions on Consumer Electronics*, 58(2):633 – 640, May 2012. ISSN 0098-3063. doi: 10.1109/TCE.2012.6227470.

[29] C. Guillemot and O.L. Meur. Image inpainting : Overview and recent advances. *IEEE Signal Processing Magazine*, 31(1):127–144, Jan 2014. ISSN 1053-5888. doi: 10.1109/MSP.2013.2273004.

[30] A. Gut. On the moments of some first passage times for sums of dependent random variables. *Stochastic Processes and their Applications*, 2(1):115 – 126, 1974. ISSN 0304-4149. doi: http://dx.doi.org/10.1016/0304-4149(74)90015-5. URL `http://www.sciencedirect.com/science/article/pii/0304414974900155`.

[31] A. Gut. On the moments and limit distributions of some first passage times. *Ann. Probab.*, 2(2):277–308, 04 1974. doi: 10.1214/aop/1176996709. URL `http://dx.doi.org/10.1214/aop/1176996709`.

[32] A. Gut. *Stopped Random Walks: Limit Theorems and Applications*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2009. ISBN 9780387878355. URL `https://books.google.co.uk/books?id=tWfBs5G7jHIC`.

[33] A. Gut and S. Janson. The limiting behaviour of certain stopped sums and some applications. *Scandinavian Journal of Statistics*, 10(4):281–292, 1983. ISSN 03036898, 14679469. URL `http://www.jstor.org/stable/4615930`.

[34] G.H. Hardy. Weierstrass's non-differentiable function. *Transactions of the American Mathematical Society*, 17(3):301–325, 1916. ISSN 00029947. URL `http://www.jstor.org/stable/1989005`.

[35] J. Herling and W. Broll. High-quality real-time video inpainting with pixmix. *IEEE Transactions on Visualization and Computer Graphics*, 20(6):866–879, 2014. ISSN 1077-2626. doi: doi.ieeecomputersociety.org/10.1109/TVCG.2014.2298016.

[36] L.R. Hocking, T. Holding, and C. Schönlieb. Numerical analysis of shell-based geometric image inpainting algorithms and their semi-implicit extension. . URL `https://arxiv.org/abs/1707.09713`.

[37] L.R. Hocking, R. MacKenzie, and C. Schönlieb. Guidefill: Gpu accelerated, artist guided geometric inpainting for 3d conversion of film. . URL `http://arxiv.org/abs/1611.05319`.

[38] A. Kokaram, B. Collis, and S. Robinson. Automated rig removal with Bayesian motion interpolation. *Vision, Image and Signal Processing, IEE Proceedings* -, 152(4):407–414, August 2005. ISSN 1350-245X. doi: 10.1049/ip-vis:20045152. URL `http://dx.doi.org/10.1049/ip-vis:20045152`.

[39] Y.K. Lai, Y.F. Lai, and J. Lin. High-quality view synthesis algorithm and architecture for 2d to 3d conversion. In *2012 IEEE International Symposium on Circuits and Systems, ISCAS 2012, Seoul, Korea (South), May 20-23, 2012*, pages 373–376, 2012. doi: 10.1109/ISCAS.2012.6272040. URL `http://dx.doi.org/10.1109/ISCAS.2012.6272040`.

[40] T.T. Le, A. Almansa, Y. Gousseau, and S. Masnou. MOTION-CONSISTENT VIDEO INPAINTING. In *ICIP 2017: IEEE International Conference on Image Processing*, ICIP 2017: IEEE International Conference on Image Processing, Beijing, China, September 2017. URL `https://hal.archives-ouvertes.fr/hal-01492536`.

[41] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. pages 674–679, 1981.

[42] L. Ma, L. Do, and P.H.N. de With. Depth-guided inpainting algorithm for free-viewpoint video. In *2012 19th IEEE International Conference on Image Processing*, pages 1721–1724, Sept 2012. doi: 10.1109/ICIP.2012.6467211.

[43] L. Ma, K. Agrawal, and R.D. Chamberlain. A memory access model for highly-threaded many-core architectures. *Future Generation Computer Systems*, 30:202 – 215, 2014. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2013.06.020. URL `http://www.sciencedirect.com/science/article/pii/S0167739X13001349`.

[44] T. März. Image inpainting based on coherence transport with adapted distance functions. *SIAM J. Img. Sci.*, 4(4):981–1000, October 2011. ISSN 1936-4954. doi: 10.1137/100807296. URL `http://dx.doi.org/10.1137/100807296`.

[45] T. März. A well-posedness framework for inpainting based on coherence transport. *Foundations of Computational Mathematics*, 15(4):973–1033, 2015. ISSN 1615-3383. doi: 10.1007/s10208-014-9199-7. URL `http://dx.doi.org/10.1007/s10208-014-9199-7`.

[46] S. Masnou and J. Morel. Level lines based disocclusion. In *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, pages 259–263. IEEE, 1998.

[47] D.A. Murio, C.E. Mejía, and S. Zhan. Discrete mollification and automatic numerical differentiation. *Computers & Mathematics with Applications*, 35(5):1 – 16, 1998. ISSN 0898-1221. doi: http://dx.doi.org/10.1016/S0898-1221(98)00001-7. URL `http://www.sciencedirect.com/science/article/pii/S0898122198000017`.

[48] P. Ndjiki-Nya, M. Koppel, D. Doshkov, H. Lakshman, P. Merkle, K. Muller, and T. Wiegand. Depth image-based rendering with advanced texture synthesis for 3-d video. *IEEE Transactions on Multimedia*, 13(3):453–465, June 2011. ISSN 1520-9210. doi: 10.1109/TMM.2011.2128862.

[49] A. Newson, A. Almansa, M. Fradet, Y. Gousseau, and P. Pérez. Towards fast, generic video inpainting. In *Proceedings of the 10th European Conference on Visual Media Production*, CVMP '13, pages 7:1–7:8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2589-9. doi: 10.1145/2534008.2534019. URL `http://doi.acm.org/10.1145/2534008.2534019`.

[50] A. Newson, A. Almansa, M. Fradet, Y. Gousseau, and P. Pérez. Video inpainting of complex scenes. *SIAM Journal on Imaging Sciences*, 7(4):1993–2019, October 2014. ISSN 1936-4954. doi: 10.1137/140954933.

[51] E. Di Nezza, G. Palatucci, and E. Valdinoci. Hitchhiker's guide to the fractional sobolev spaces. *Bulletin des Sciences Mathématiques*, 136(5):521 – 573, 2012. ISSN 0007-4497. doi: http://dx.doi.org/10.1016/j.bulsci.2011.12.004. URL `http://www.sciencedirect.com/science/article/pii/S0007449711001254`.

[52] NVIDIA. *CUDA C Programming Guide*. 2015.

[53] K. Oh, S. Yea, and Y. Ho. Hole filling method using depth based in-painting for view synthesis in free viewpoint television and 3-d video. In *Proceedings of the 27th Conference on Picture Coding Symposium*, PCS'09, pages 233–236, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-4593-6. URL `http://dl.acm.org/citation.cfm?id=1690059.1690118`.

[54] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005. ISBN 0321335597.

[55] S.H. Roosta. *Parallel processing and parallel algorithms : theory and computation*. Springer, New York, 2000. ISBN 0-387-98716-9. URL `http://opac.inria.fr/record=b1096260`.

[56] Y. Saad and M.H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. doi: 10.1137/0907058. URL `http://dx.doi.org/10.1137/0907058`.

[57] R. Sadek, G. Facciolo, P. Arias, and V. Caselles. A variational model for gradient-based video editing. *International Journal of Computer Vision*, 103(1):127–162, 2013. ISSN 1573-1405. doi: 10.1007/s11263-012-0597-5. URL `http://dx.doi.org/10.1007/s11263-012-0597-5`.

[58] H. Sawhney, Y. Guo, J. Asmuth, and R. Kumar. Multi-view 3d estimation and applications to match move. In *Proceedings of the IEEE Workshop on Multi-View Modeling & Analysis of Visual Scenes*, MVIEW '99, pages 21–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0110-9. URL `http://dl.acm.org/citation.cfm?id=519933.825159`.

[59] C. Schönlieb. *Partial Differential Equation Methods for Image Inpainting*. Cambridge University Press, 2015.

[60] M. Seymour. Art of stereo conversion: 2d to 3d., May 2012. URL `https://www.fxguide.com/featured/art-of-stereo-conversion-2d-to-3d-2012/`.

[61] T. Shiratori, Y. Matsushita, Y. Xiaoou Tang, and Y. Sing Bing Kang. Video completion by motion field transfer. volume 1, pages 411–418. IEEE, 2006. ISBN 0-7695-2597-0.

[62] M. Strobel, J. Diebold, and D. Cremers. Flow and color inpainting for video completion. In *German Conference on Pattern Recognition (GCPR)*, Münster, Germany, September 2014. doi: 10.1007/978-3-319-11752-2_23.

[63] J. Sun, L. Yuan, J. Jia, and H. Shum. Image completion with structure propagation. *ACM Trans. Graph.*, 24(3):861–868, July 2005. ISSN 0730-0301. doi: 10.1145/1073204.1073274. URL `http://doi.acm.org/10.1145/1073204.1073274`.

[64] A. Telea. An image inpainting technique based on the fast marching method. *Journal of Graphics Tools*, 9(1):23–34, 2004.

[65] A. Tucker. *Computer Science Handbook, Second Edition*. Chapman & Hall/CRC, 2004. ISBN 158488360X.

[66] J.M. Varah. A lower bound for the smallest singular value of a matrix. *Linear Algebra and its Applications*, 11(1):3 – 5, 1975. ISSN 0024-3795. doi: http://dx.doi.org/10.1016/0024-3795(75)90112-3. URL `http://www.sciencedirect.com/science/article/pii/0024379575901123`.

[67] R.S. Varga. *Matrix iterative analysis.* Prentice-Hall series in automatic computation. Prentice-Hall, 1962. URL `https://books.google.co.uk/books?id=PFYWAAAAIAAJ`.

[68] J. Weickert. *Anisotropic Diffusion in Image Processing.* ECMI Series. Teubner, Stuttgart, 1998. URL `http://www.mia.uni-saarland.de/weickert/book.html`.

[69] Y. Wexler, E. Shechtman, and M. Irani. Space-time completion of video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):463–476, March 2007. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.60.

[70] D. Williams. *Probability with Martingales.* Cambridge mathematical textbooks. Cambridge University Press, 1991. ISBN 9780521406055. URL `https://books.google.co.uk/books?id=RnOJeRpk0SEC`.

[71] X. Xu, L. Po, C. Cheung, L. Feng, K.H. Ng, and K.W. Cheung. Depth-aided exemplar-based hole filling for DIBR view synthesis. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013), Beijing, China, May 19-23, 2013*, pages 2840–2843, 2013. doi: 10.1109/ISCAS.2013.6572470. URL `http://dx.doi.org/10.1109/ISCAS.2013.6572470`.

[72] S.S. Yoon, H. Sohn, Y.J. Jung, and Y.M. Ro. Inter-view consistent hole filling in view extrapolation for multi-view image generation. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2883–2887, Oct 2014. doi: 10.1109/ICIP.2014.7025583.

[73] J. Zhi. A case of study for 3d stereoscopic conversion in visual effects industry. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 7(1):53 – 58, 2013. ISSN 1307-6892. URL `http://iastem.com/Publications?p=73`.

[74] S. Zinger, L. Do, and P.D. With. Free-viewpoint depth image based rendering. *J. Vis. Comun. Image Represent.*, 21(5-6):533–541, July 2010. ISSN 1047-3203. doi: 10.1016/j.jvcir.2010.01.004. URL `http://dx.doi.org/10.1016/j.jvcir.2010.01.004`.

Appendix A

Appendix

A.1 Punctured sums

Here we provide further justification for our exclusion of the point $\mathbf{x}$ from the update formula (2.0.3) in Algorithm 1 as mentioned in Remark 2.5.1. As we mentioned there, this makes no difference to the direct form of Algorithm 1, because the subroutine FillRow only involves sums taken over $A_{\epsilon,h}(\mathbf{x}) \cap (\Omega \backslash D^{(k)})$, which never contains $\mathbf{x}$. However, the semi-implicit extension of Algorithm 1 expresses $u_h(\mathbf{x})$ as a sum of $u_h(\mathbf{y})$ over a set of points that might include $\mathbf{x}$. The reason we deliberately exclude $\mathbf{x}$ is because, as the following proposition shows, if $w_\epsilon(\mathbf{x},\mathbf{x}) < \infty$, it doesn't matter, but if $w_\epsilon(\mathbf{x},\mathbf{x}) = \infty$, it wreaks havoc. Moreover, the weights (2.5.2) which we wish to study *do* have the property that $w_\epsilon(\mathbf{x},\mathbf{x}) = \infty$.

Proposition A.1.1. *Suppose $\mathbf{x} \in B$ for some finite set $B \subset \mathbb{R}^2$, and suppose there exist non negative weights $w : B \times B \to [0,\infty]$, finite everywhere except possibly at $(\mathbf{x},\mathbf{x})$. Then if $w(\mathbf{x},\mathbf{x}) < \infty$, we have*

$$u_h(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in B} w(\mathbf{x},\mathbf{y}) u_h(\mathbf{y})}{\sum_{\mathbf{y} \in B} w(\mathbf{x},\mathbf{y})} = \frac{\sum_{\mathbf{y} \in B \backslash \{\mathbf{x}\}} w(\mathbf{x},\mathbf{y}) u_h(\mathbf{y})}{\sum_{\mathbf{y} \in B \backslash \{\mathbf{x}\}} w(\mathbf{x},\mathbf{y})}.$$

On the other hand, if $w(\mathbf{x},\mathbf{x}) = \infty$, we have an indeterminate expression

$$u_h(\mathbf{x}) = \frac{\infty}{\infty}.$$

Proof. The proof is an exercise in cancellation and left to the reader. □

A.2 Details of the GPU Implementation

Some care is required in order to obtain a maximally efficient GPU implementation. In particular, some thought needs to go into how to assign threads to pixels, as well as how to cut down on memory accesses when working with ghost pixels. Here we go over two optimizations which together can lead to a speedup by an order of magnitude or more.

Efficient Implementation of Ghost Pixels. The core of our algorithm is the computation of $u_h(\mathbf{x})$ as a weighted sum of $u_h(\mathbf{y})$ over all "available" ghost pixels $\mathbf{y}$ in the rotated ball $\tilde{B}_{\epsilon,h}(\mathbf{x})$ in Figure 2.9(b). Since ghost pixels are computed based on bilinear interpolation (4.2.2) of their (up to) four nearest neighbors[1] in Ω_h, being available means that all of the relevant neighbors have already been "filled". The

[1] If a ghost pixel happens to lie exactly on the vertical or horizontal line joining two pixel centers, then only two neighbors are required.

naive approach to checking availability, therefore, requires up to four memory accesses per ghost pixel.

We reduce this to one (effective) memory access by taking advantage of the ability of most graphics cards to perform bilinear interpolation in hardware[2]. Specifically, we flag pixels as "filled" by writing a zero into their alpha channel. If a pixel is not filled, then we ensure that the alpha channel is at least 1. Since the bilinear interpolant of a non-negative function is zero at a point if and only if it is also zero at all neighboring (contributing) lattice points, we can determine if a ghost pixel $\mathbf{y}$ is available simply by evaluating the alpha channel at $\mathbf{y}$ using (hardware based) interpolation. If the result is zero, $\mathbf{y}$ is available - otherwise, it is not. Experiments indicate that this small change leads to a speed up by a factor of roughly four.

Boundary Tracking for Efficient Assigment of Threads to Pixels.

While early GPU programming languages such as GLSL leave programmers writing image processing code with little explicit control over the correspondence between threads and pixels, modern languages like CUDA and OpenCL assume no a priori connection between the two. These languages simply assign each thread a unique index (either an integer or tuple of integers) and leave it up to the programmer to decide how these indices map to pixels. This flexibility may be exploited for additional performance gains.

A naive approach is to assign one thread per pixel. However this is inefficient as only the boundary pixels are actually being updated. Since a typical HD image contains millions of pixels but the maximum number of concurrent threads in a typical GPU is in the tens of thousands[3], this approach wastes a great deal of time in which the GPU is busy processing pixels that do not actually change.

A better approach is to assign threads only to pixels on the boundary of the inpainting domain, by maintaining a list of the coordinates of all boundary pixels. This list is updated every iteration by a parallel algorithm that requires each of $O(|\partial D_h|)$ threads to do logarithmic work.

Our algorithm is based on the observation that every boundary pixel at step k of Guidefill either was also a boundary pixel at step $k-1$, or else has an immediate neighbor that was. More precisely, if $D_h^{(k-1)}$ and $D_h^{(k)}$ denote the inpainting domain at steps $k-1$ and k respectively, then we can construct $\partial D_h^{(k)}$ as follows: For each $\mathbf{x} \in \partial D_h^{(k-1)}$, if $\mathbf{x} \in D_h^{(k)}$ then we insert $\mathbf{x}$ into $\partial D_h^{(k)}$. On the other hand, if $\mathbf{x} \notin D_h^{(k)}$, then we iterate through the eight immediate neighbors in $\mathcal{N}(\mathbf{x})$ and add to $\partial D_h^{(k)}$ all neighbors which belong to $D_h^{(k)}$. This will result in a list which contains all pixels in the new boundary, but additionally contains some duplicates as a given pixel in $\partial D_h^{(k)}$ will in general be the neighbor of more than one pixel in $\partial D_h^{(k-1)}$. These duplicates may then be eliminated using a standard parallel algorithm such as the one presented in [65, Ch. 10].

Rather than eliminate duplicates in post processing, we use a slightly more complex variant of the above approach that prevents duplicates before they occur. This is explained in the next short Section.

Details of the boundary update step. For the boundary update step we use an algorithm like the one described above but designed to eliminate duplicates before they occur. Specifically, every time we encounter a pixel $\mathbf{x} \in \partial D_h^{(k-1)} \backslash \partial D_h^{(k)}$ with a neighbor $\mathbf{y} \in \mathcal{N}(\mathbf{x}) \cap D_h^{(k)}$, before inserting $\mathbf{y}$ into $\partial D_h^{(k)}$ we iterate over $\mathcal{N}(\mathbf{y})$ looking for a third pixel $\mathbf{x}' \in \partial D_h^{(k-1)} \backslash \partial D_h^{(k)}$ such that $\mathbf{x}' \neq \mathbf{x}$. In this case we know that both $\mathbf{x}$ and $\mathbf{x}'$ will try to insert $\mathbf{y}$ into $\partial D_h^{(k)}$. To prevent duplicates we only allow whichever of the two comes first in the lexicographic ordering of pixels from left to right and top to bottom to perform the insertion.

[2]Experiments indicate that bilinear interpolation in hardware (implemented in CUDA as a single texture lookup at a non-pixel center) is about four times fasters than making four separate texture lookups at each of the nearest pixel centers and then performing the interpolation in software. However, we have been unable to find a reference explaining why this is so.

[3]For example, the GeForce GTX Titan X is a flagship NVIDIA GPU at the time of writing and has a total of 24 multiprocessors [2] each with a maximum of 2048 resident threads [52, Appendix G.1].

For the insertion itself, we initialize $\partial D_h^{(k)}$ as an empty array with space for at least eight times the number of pixels in $\partial D_h^{(k-1)}$. Each pixel $\mathbf{x} \in \partial D_h^{(k-1)}$ is allocated eight "slots" in $\partial D_h^{(k)}$ - one for each of its neighbors in $\mathcal{N}(\mathbf{x})$. Each time $\mathbf{x}$ inserts a $\mathbf{y} \in \mathcal{N}(\mathbf{x})$ into $\partial D_h^{(k)}$, it places it into one of these eight slots - all unused slots are filled with null entrees. These may be removed at the end using $O(|\partial D_h^{(k)}|)$ processors and $O(\log |\partial D_h^{(k)}|)$ operations per processor using a standard algorithm such as [54, Chapter 36].

Termination conditions for the main loop. When Guidefill is run without boundary tracking (i.e. by allocating one thread per pixel in the image), the number of unfilled pixels is not automatically known and must be periodically computed - we do this every ten iterations using the "count_ if" function that comes with the CUDA Thrust library. To prevent the algorithm possibly getting trapped in an infinite loop, if two successive computations of $|D_h|$ return the same number, the "smart order" condition (4.5.3) is turned off for one iteration. The number of unfilled pixels $|D_h|$ is then computed again - if it has now decreased, execution continues as usual, otherwise the program terminates. Termination also occurs if $|D_h| = 0$. Termination conditions for Guidefill with boundary tracking are the same but with $|D_h|$ replaced by $|\partial D_h|$. In this case there is no need for a separate mechanism to compute $|\partial D_h|$ as it is already computed every iteration concurrently with the boundary update.

Algorithms SM1 and SM2 provide pseudocode for Guidefill with and without boundary tracking.

Algorithm 7 Guidefill Without Boundary Tracking

```
$u_h$ = image/video frame.
$B$ = bounding box of inpainting domain.
$M$ = countUnfilledPixels($B$).
$k = 1$.
while $M > 0$ do
    threads ← Allocate a thread for each pixel in $B$.
    for thread in threads do
        $\mathbf{x}$ = pixel pointed to by thread.
        if not boundaryPixel($\mathbf{x}$) then
            return
        end if
        if not readyToBeFilled($\mathbf{x}$) and smart order on then
            return
        end if
        Fill($\mathbf{x}$) and flag $\mathbf{x}$ as filled.
    end for
    if Mod($k$,10)== 0 or smart order is off then
        $M$ = countUnfilledPixels($B$).
    end if
    if $M$ has not changed and smart order is turned on then
        turn off smart order for one iteration.
    else if $M$ has not changed and smart order is off then
        exit
    end if
    $k \leftarrow k + 1$.
end while
```

Algorithm SM3 provides pseudocode for the boundary update step.

Algorithm 8 Guidefill With Boundary Tracking

```
$u_h$ = image/video frame.
$\Gamma$ = findInpaintingDomainBoundary($u_h$).
while $\Gamma \neq \emptyset$ do
    threads $\leftarrow$ Allocate a thread for each pixel in $\Gamma$.
    for thread in threads do
        $\mathbf{x}$ = pixel pointed to by thread.
        if not readyToBeFilled($\mathbf{x}$) and smart order on then
            return
        end if
        Fill($\mathbf{x}$) and flag $\mathbf{x}$ as filled.
    end for
    update($\Gamma$) and count unfilled pixels
    if no pixels were filled and smart order is turned on then
        turn off smart order for one iteration.
    else if no pixels were filled and smart order is off then
        exit
    end if
end while
```

Algorithm 9 Boundary Update

```
Require: $|\Gamma^{(k+1)}| \geq 8|\Gamma^{(k)}|$.
  $\Gamma^{(k)}$ = current boundary.
  $\Gamma^{(k+1)}$ = new boundary.
  threads $\leftarrow$ allocate one thread per pixel in $\Gamma^{(k)}$.
  for thread in threads do
      $i \leftarrow$ Index(thread).
      $\mathbf{x} = \Gamma^{(k)}[i]$.
      if $\mathbf{x} \in D^{(k+1)}$ then
          $\Gamma^{(k+1)}[8 * i] = \mathbf{x}$.
          for $j = 1$ to 7 do
              $\Gamma^{(k+1)}[8 * i + j]$=NULL.
          end for
      else
          for $\mathbf{y}_j \in \mathcal{N}(\mathbf{x})\backslash\{\mathbf{x}\}$ do
              if $\mathbf{y}_j \notin D_h^{(k+1)}\backslash\Gamma^{(k)}$ then
                  $\Gamma^{(k+1)}[8 * i + j]$=NULL.
              else
                  canAdd = true.
                  for $\mathbf{z}_\ell \in \mathcal{N}(\mathbf{y}_j)\backslash\{\mathbf{y}_j\}$ do
                      if $\mathbf{z}_\ell \in \Gamma^{(k)}$ and Order($\mathbf{z}_\ell$) < Order($\mathbf{x}$) then
                          canAdd = false.
                      end if
                  end for
                  if canAdd then
                      $\Gamma^{(k+1)}[8 * i + j] = \mathbf{y}_j$.
                  else
                      $\Gamma^{(k+1)}[8 * i + j]$=NULL.
                  end if
              end if
          end for
      end if
  end for
  $\Gamma^{(k+1)} \leftarrow$removeNullEntrees($\Gamma^{(k+1)}$).
```

A.3 Algorithmic Complexity in the General case

This short section is meant to complement the complexity analysis in Section 4.8, by discussing what happens when we relax some of the assumptions of Theorem 4.8.2. If we relax anything, then Theorem 4.8.2 ceases to be true in general. For example, if we eliminate the assumption that B_h is surrounded by readable pixels then $K(N)$ is no longer bounded above by $\lceil\sqrt{N}/2\rceil$ and is instead given by

$$K(N) = \max_{\mathbf{x}\in D_h} \min_{\mathbf{y}\in\Omega_h\backslash(D_h\cup B_h)} d_{\mathcal{G}}(\mathbf{x},\mathbf{y}),$$

where $d_{\mathcal{G}}(\mathbf{x},\mathbf{y})$ is equal to the length of the shortest path in the graph $\mathcal{G}$ with vertex set $\mathcal{V} = \Omega_h\backslash B_h$ and where each vertex $\mathbf{x}\in\mathcal{V}$ is connected to each of its (up to eight) neighbors in $\mathcal{N}(\mathbf{x})\cap\mathcal{V}$. This can be as bad as $O(N)$ - consider, for example, an inpainting domain that is one pixel tall and N pixels wide, with just a single readable boundary pixel on one side and all other pixels within an annulus of width ϵ pixels surrounding the inpainting domain non-readable. If we take that one pixel away, then the runtime becomes infinite[4]. On the other hand, if we retain the assumption $(\partial_{\text{outer}}D_h)\cap B_h = \emptyset$ but turn on the smart order, we run into similar problems. The case where both assumptions are relaxed is even more complicated, but is related to the case of relaxing only one or the other by the following proposition:

Proposition A.3.1. *Let $K(N)$ denote the number of iterations required for Guidefill to terminate, and let $K_{smart}(N)\le K(N)$ denote the last iteration on which the smart order test* (4.5.3) *causes the filling of a pixel to be delayed. Let $K^*(N)$ denote the number of iterations for Guidefill to terminate if the smart order test is disabled. Then*

$$K(N)\le K_{smart}(N)+K^*(N) \tag{A.3.1}$$

Proof. After $K_{\text{smart}}(N)$ iterations, we will have filled a subset $\mathcal{V}^*\subseteq D_h$ of pixels and the smart order test (4.5.3) will no longer have any effect. The number of remaining iterations is therefore given by

$$\begin{aligned} K_{\text{remaining}}(N) &= \max_{\mathbf{x}\in D_h\backslash\mathcal{V}^*} \min_{\mathbf{y}\in(\Omega_h\backslash(D_h\cup B_h))\cup\mathcal{V}^*} d_{\mathcal{G}}(\mathbf{x},\mathbf{y}) \\ &\le \max_{\mathbf{x}\in D_h} \min_{\mathbf{y}\in\Omega_h\backslash(D_h\cup B_h)} d_{\mathcal{G}}(\mathbf{x},\mathbf{y}) \\ &= K^*(N). \end{aligned}$$

□

Although each of the quantities on the RHS of (A.3.1) is difficult to bound in general, we can obtain rough estimates under modest assumptions. In particular, as long as the splines are laid out in a reasonable way with one endpoint in the readable portion of the image $\Omega_h\backslash(D_h\cup B_h)$ and the other in D_h, such that none of the splines overlap the non-readable pixels in B_h and they are all spaced far enough apart that they do not cross or otherwise interact, then we expect that

$$K_{\text{smart}}(N)\lesssim \text{length in pixels of the longest spline.}$$

At the same time, we expect that the inpainting domains arising in 3D conversion will typically consist *mostly* of horizontal line segments with a readable pixel at one end and a non-readable pixel at the other. Exceptions can and do occur, and it is also possible to have "cracks" with unreadable pixels on either

[4]In practice we implement checks to detect this case and terminate the algorithm.

side, however, so long as these cracks are not too deep we can expect

$$K^*(N) \lesssim \text{width in pixels of } D_h \text{ at its widest point.}$$

Excluding situations such as a very long and thin inpainting domain with a spline running down the entirety of its length, we typically expect both of these quantities to be much smaller than N, so that

$$K(N) \leq K_{\text{smart}}(N) + K^*(N) \ll N$$

in most cases in practice.

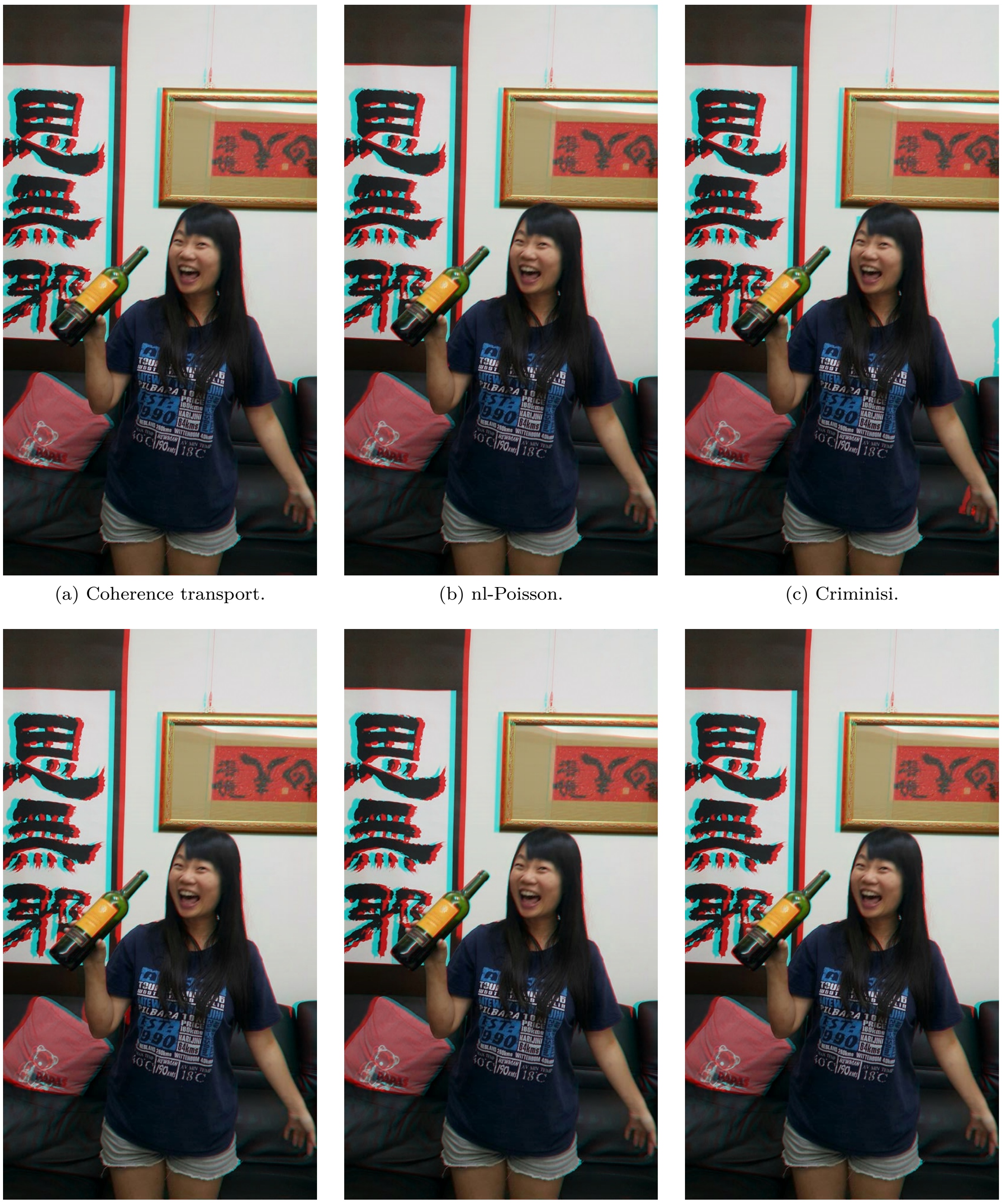
A.4 Results in Anaglyph 3D

In the Chapter 4 Section 4.9, we showed some examples of the left or right eye view produced by our method. Here we show in Figures A.1, A.2, and A.3 both eyes together in anaglyph 3D (note that anaglyph 3D glasses are required). Since the inpainted area is only visible in one eye, small artifacts may be less noticeable. See also the anaglyph videos included in the supplementary material. Unlike before we do not crop our output.

A.5 Brief discussion of GPU Architechure

Here we briefly give more justification of our choice $p = 20480$, the maximum number of resident threads, in Section 4.9.2. NVIDIA GPUs contain a small number of multiprocessors, each of which issue one or two instructions to each of a small number of *warps* of 32 threads every clock cycle. For our particular GPU, this limits the number of threads that can receive instructions in a given clock cycle to 1280, [5] in comparison with a maximum of 20480 threads that can be resident in the GPU at any given time[6]. However, because instructions take anywhere from tens to hundreds of clock cycles to execute, a GPU can keep a much larger number of threads "active" by issuing instructions to other warps in the intervening time. For simplicity, we assume in our analysis that p is equal to the maximum number of resident threads, that is $p = 20480$.

[5]10 multiprocessors times 4 warp schedulers per multiprocessor times 32 threads per warp.
[6]10 multiprocessors times 2048 maximum resident threads per multiprocessor.



(a) Coherence transport. (b) nl-Poisson. (c) Criminisi.

(d) Content-Aware Fill. (e) Guidefill (pre spline adjustment). (f) Guidefill (post spline adjustment).

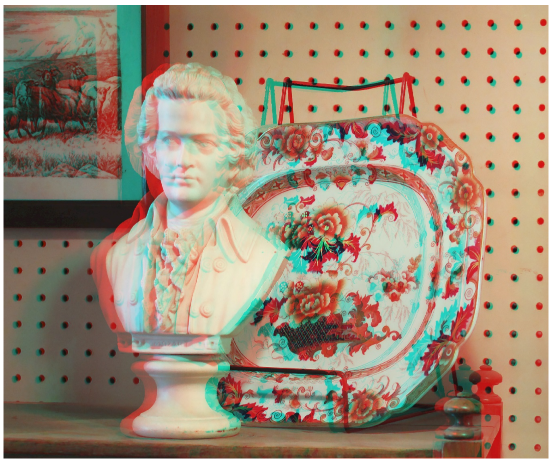
Figure A.1: Anaglyph 3D output of the “Wine” 3D conversion example, using various methods for the inpainting step. Since the inpainted area is only visible in one eye, artifacts may be less noticeable.

(a) Coherence transport.

(b) nl-Poisson.

(c) Content-Aware Fill.

(d) Criminisi.


(e) Guidefill (pre spline adjustment).

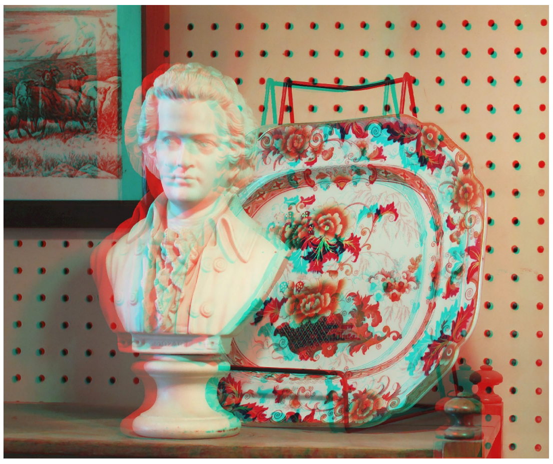

(f) Guidefill (post spline adjustment).

Figure A.2: Anaglyph 3D output of the “Bust” 3D conversion example, using various methods for the inpainting step. Since the inpainted area is only visible in one eye, artifacts may be less noticeable.

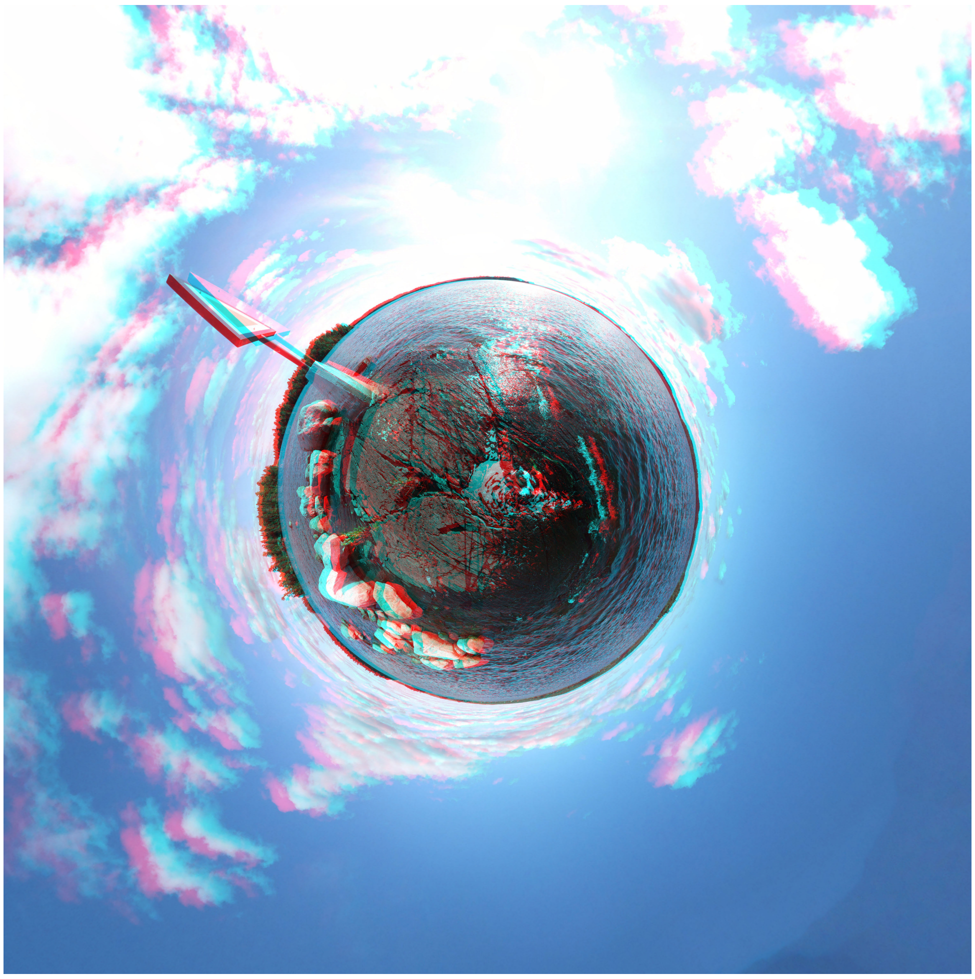


Figure A.3: 3D conversion to create an illusion. Anaglyph 3D output of the “Planet” 3D conversion example, using fake spherical geometry to enhance the illusion of a tiny planet floating in space. Inpainting is done with Photoshop’s Content-Aware Fill in this case.

A.6 Properties of Ghost Pixels and Equivalent Weights

In this appendix we prove the six properties of ghost pixels and equivalent weights listed in Section 5.1. These properties all follow from the definition of $u_h(\mathbf{y})$ when $\mathbf{y}$ is a ghost pixel, namely

$$u_h(\mathbf{y}) = \sum_{\mathbf{z}\in\mathrm{Supp}(\{\mathbf{y}\})} \Lambda_{\mathbf{z},h}(\mathbf{y})u_h(\mathbf{z}),$$

where $\{\Lambda_{\mathbf{z},h}\}_{\mathbf{z}\in\mathbb{Z}_h^2}$ denote the basis functions of bilinear interpolation associated with the lattice $\mathbb{Z}_h^2$, and where $\mathrm{Supp}(A)$ denotes the set of real pixels needed to define a set A of ghost pixels using bilinear interpolation. Note that if $\mathbf{y}\in A$, then

$$\Lambda_{\mathbf{z},h}(\mathbf{y}) = 0 \quad \text{for all } \mathbf{z}\notin \mathrm{Supp}(A). \tag{A.6.1}$$

The following explicit formula for $\mathrm{Supp}(\{(x,y)\})$ (which comes from the definition of bilinear interpolation) will occasionally be useful.

$$\mathrm{Supp}(\{(x,y)\}) = \left\{\left(\left\lfloor\frac{x}{h}\right\rfloor h, \left\lfloor\frac{y}{h}\right\rfloor h\right), \left(\left\lceil\frac{x}{h}\right\rceil h, \left\lfloor\frac{y}{h}\right\rfloor h\right), \left(\left\lfloor\frac{x}{h}\right\rfloor h, \left\lceil\frac{y}{h}\right\rceil h\right), \left(\left\lceil\frac{x}{h}\right\rceil h, \left\lceil\frac{y}{h}\right\rceil h\right)\right\}. \tag{A.6.2}$$

First we prove property one.

1. Explicit formula:

$$\tilde{w}(\mathbf{x},\mathbf{z}) = \sum_{\mathbf{y}\in A(\mathbf{x})} \Lambda_{\mathbf{y},h}(\mathbf{z})w(\mathbf{x},\mathbf{y})$$

Proof. This follows straightforwardly from the definition of ghost pixels, the property (A.6.1), and a few exchanges of finite sums.

$$\begin{aligned}
u_h(\mathbf{x}) &= \sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y})u_h(\mathbf{y}) \\
&= \sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y}) \sum_{\mathbf{z}\in\mathrm{Supp}(\{\mathbf{y}\})} \Lambda_{\mathbf{z},h}(\mathbf{y})u_h(\mathbf{z}) \\
&= \sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y}) \sum_{\mathbf{z}\in\mathrm{Supp}(A(\mathbf{x}))} \Lambda_{\mathbf{z},h}(\mathbf{y})u_h(\mathbf{z}) \\
&= \sum_{\mathbf{z}\in\mathrm{Supp}(A(\mathbf{x}))} \underbrace{\left\{\sum_{\mathbf{y}\in A(\mathbf{x})} \Lambda_{\mathbf{z},h}(\mathbf{y})w(\mathbf{x},\mathbf{y})\right\}}_{:=\tilde{w}(\mathbf{x},\mathbf{z})} u_h(\mathbf{z})
\end{aligned}$$

□

Next, instead of proving properties two and three, we prove a stronger result from which they both follow.

1.(a) Preservation of degree 1 polynomials: Suppose $f(\mathbf{y})$ is a (scalar valued) polynomial of degree at most 1, that is $f(\mathbf{y}) = a + \mathbf{b}\cdot\mathbf{y}$ for some $a\in\mathbb{R}$ and $\mathbf{b}\in\mathbb{R}^2$. Then

$$\sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y})f(\mathbf{y}) = \sum_{\mathbf{y}\in\mathrm{Supp}(A(\mathbf{x}))} \tilde{w}(\mathbf{x},\mathbf{y})f(\mathbf{y}).$$

Proof. This follows from the fact that the bilinear interpolant of a polynomial of degree at most 1 is just the polynomial back. That is,

$$\sum_{\mathbf{z}\in \mathrm{Supp}(\{\mathbf{y}\})} \Lambda_{\mathbf{z},h}(\mathbf{y})f(\mathbf{z}) = f(\mathbf{y}).$$

This is obvious and we do not prove it. We will also use (A.6.1) once.

$$\begin{aligned}
\sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y})f(\mathbf{y}) &= \sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y}) \left\{ \sum_{\mathbf{z}\in \mathrm{Supp}(\{\mathbf{y}\})} \Lambda_{\mathbf{z},h}(\mathbf{y})f(\mathbf{z}) \right\} \\
&= \sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y}) \left\{ \sum_{\mathbf{z}\in \mathrm{Supp}(A(x))} \Lambda_{\mathbf{z},h}(\mathbf{y})f(\mathbf{z}) \right\} \\
&= \sum_{\mathbf{z}\in \mathrm{Supp}(A(x))} \left\{ \sum_{\mathbf{y}\in A(\mathbf{x})} \Lambda_{\mathbf{z},h}(\mathbf{y})w(\mathbf{x},\mathbf{y}) \right\} f(\mathbf{z}) \\
&= \sum_{\mathbf{z}\in \mathrm{Supp}(A(x))} \tilde{w}(\mathbf{x},\mathbf{z})f(\mathbf{z}).
\end{aligned}$$

□

2. Preservation of total mass.

$$\sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y}) = \sum_{\mathbf{y}\in \mathrm{Supp}(A(\mathbf{x}))} \tilde{w}(\mathbf{x},\mathbf{y}).$$

Proof. Special case of 1.(a), $p(\mathbf{y}) \equiv 1$. □

3. Preservation of center of mass (or first moment).

$$\sum_{\mathbf{y}\in A(\mathbf{x})} w(\mathbf{x},\mathbf{y})\mathbf{y} = \sum_{\mathbf{y}\in \mathrm{Supp}(A(\mathbf{x}))} \tilde{w}(\mathbf{x},\mathbf{y})\mathbf{y}.$$

Proof. Apply 1.(a) to each component of $f(\mathbf{y}) = \mathbf{y}$. □

4. Inheritance of non-negativity:

$$\tilde{w}_\epsilon(\mathbf{x},\mathbf{z}) \geq 0 \quad \textit{for all } \mathbf{z} \in \textit{Supp}(A_{\epsilon,h}(\mathbf{x})).$$

Proof. This is immediate from the non-negativity of the original weights $\{w_\epsilon\}$, the non-negativity of the basis functions $\{\Lambda_{\mathbf{y},h}\}$, and the explicit formula (5.1.1). □

5. Inheritance of non-degeneracy condition (2.0.2).

$$\sum_{\mathbf{y}\in \textit{Supp}(A_{\epsilon,h}(\mathbf{x})\cap(\Omega\setminus D^{(k)}))} \tilde{w}_\epsilon(\mathbf{x},\mathbf{y}) > 0.$$

Proof. Apply preservation of total mass to (2.0.2). □

6. Universal Support. *For any $n \in \mathbb{Z}$, we have*

$$\textit{Supp}(A_{\epsilon,h}(\mathbf{x}) \cap \{y \leq nh\}) \subseteq D_h(B_{\epsilon,h}(\mathbf{x})) \cap \{y \leq nh\} \subseteq B_{\epsilon+2h,h}(\mathbf{x}) \cap \{y \leq nh\}.$$

where $\{y \le nh\} := \{(x,y) \in \mathbb{R}^2 : y \le nh\}$, *and where* D_h *is the dilation operator defined in our section on notation.*

Proof. Let $(x,y) \in A_{\epsilon,h}(\mathbf{x}) \cap \{y \le nh\}$. Then $x^2 + y^2 \le \epsilon^2$ and $y \le nh$. Hence $\left(\left\lfloor \frac{x}{h} \right\rfloor h, \left\lfloor \frac{y}{h} \right\rfloor h\right) \in B_{\epsilon,h}(\mathbf{x})$, and by (A.6.2) we have

$$\operatorname{Supp}(\{(x,y)\}) \subseteq \mathcal{N}\left(\left(\left\lfloor \frac{x}{h} \right\rfloor h, \left\lfloor \frac{y}{h} \right\rfloor h\right)\right) \subseteq D_h(B_{\epsilon,h}(\mathbf{x})),$$

Where $\mathcal{N}(\mathbf{x})$ denotes the nine-point neighborhood of $\mathbf{x} \in \mathbb{Z}_h^2$ defined in the notation section. But since we also know $y \le nh$, we have $\left\lceil \frac{y}{h} \right\rceil h \le \left\lceil \frac{nh}{h} \right\rceil h \le nh$, and hence applying (A.6.2) again we conclude

$$\operatorname{Supp}(\{(x,y)\}) \subseteq \{y \le nh\}$$

as well. Since $(x,y) \in A_{\epsilon,h}(\mathbf{x}) \cap \{y \le nh\}$ was arbitrary, the first inclusion follows. For the second inclusion, note that every element of $D_h(B_{\epsilon,h}(\mathbf{x})) \cap \{y \le nh\}$ is of the form $(x,y) = \mathbf{y} + \Delta\mathbf{y}$ where $\mathbf{y} \in B_{\epsilon,h}(\mathbf{x})$ and $\Delta\mathbf{y} \in \{-h,0,h\} \times \{-h,0,h\}$. Hence

$$\|(x,y)\| = \|\mathbf{y} + \Delta\mathbf{y}\| \le \|\mathbf{y}\| + \|\Delta\mathbf{y}\| \le \epsilon + \sqrt{2}h < \epsilon + 2h.$$

At the same time we have $y \le nh$, so $(x,y) \in B_{\epsilon+2h,h}(\mathbf{x}) \cap \{y \le nh\}$ as claimed. □

A.7 Proof of Theorem 6.5.1

We begin by breaking the error up into pieces as

$$\|u_h - u_{m\ddot{a}rz}\|_p \le \|u_h - u\|_p + \|u - u_{m\ddot{a}rz}\|_p,$$

where u is our proposed limit (6.3.3)*. We will then separately prove*

$$\begin{aligned} \|u_h - u\|_p &\le K_1 \cdot (rh)^{(\frac{s'}{2} + \frac{1}{2p}) \wedge \frac{s}{2} \wedge 1} \\ \|u - u_{m\ddot{a}rz}\|_p &\le K_2 r^{-q\left\{s \wedge \left(s' + \frac{1}{p}\right) \wedge 1\right\}} \end{aligned}$$

for constants $K_1 > 0$, $K_2 > 0$ with the claimed properties. Taken together, these two inequalities prove the first claim (6.5.2)*. For the second claim* (6.5.3)*, just the second is enough.*

Step 1: Bounding $\|u_h - u\|_p$: *This step is straightforward. By Theorem 6.3.1 we know*

$$\|u_h - u\|_p \le K \cdot (rh)^{(\frac{s'}{2} + \frac{1}{2p}) \wedge \frac{s}{2} \wedge 1}$$

where $K(\theta_r^*, \frac{r}{\mathbf{g}_r^* \cdot e_2}) > 0$ depends on u_0, $\mathcal{U}$, and $\{U_i\}_{i=1}^M$ (which are fixed) and continuously on $\frac{r}{\mathbf{g}_r^* \cdot e_2}$ and θ_r^* (which are not fixed in this case). By assumption, $\frac{\mathbf{g}_r^*}{r} \to \mathbf{g}^*$ as $r \to \infty$. Moreover, by continuity we know there is a $\eta > 0$ s.t. $|\frac{\mathbf{g}_r^* \cdot e_2}{r} - \mathbf{g} \cdot e_2| < \eta$ and $|\theta_r^* - \theta^*| < \eta$ implies

$$K\left(\theta_r^*, \frac{r}{\mathbf{g}_r^* \cdot e_2}\right) \le K\left(\theta^*, \frac{1}{\mathbf{g}^* \cdot e_2}\right) + 1.$$

We simply apply our assumption $\frac{\mathbf{g}_r^*}{r} \to \mathbf{g}^*$ to find an R such that $r > R$ implies that the bounds involving

η are satisfied, and then define

$$K_1 = \max\left\{\max_{r=1}^{R}\left\{K\left(\theta_r^*, \frac{r}{\mathbf{g}_r^*\cdot e_2}\right)\right\}, K\left(\theta^*, \frac{1}{\mathbf{g}^*\cdot e_2}\right)+1\right\}.$$

All the claimed dependency properties of K_1 follow from the analogous properties of K.

Step 2: Bounding $\|u - u_{\textbf{märz}}\|_p$: *This step is a little more work, but also straightforward. We begin by relating $|u - u_{märz}|$ to $|\cot\theta - \cot\theta_r|$, which in turn is related to $\|\frac{\mathbf{g}_r^*}{r} - \mathbf{g}^*\|$. Indeed, it is straightforward to show*

$$|\cot\theta - \cot\theta_r| \le \frac{r}{|\mathbf{g}_r^*\cdot e_2|}\left\|\frac{\mathbf{g}_r^*}{r} - \mathbf{g}^*\right\| \le Cr^{-q},$$

where

$$C = \max\left\{\max_{r=1}^{R'}\left\{\frac{r}{\mathbf{g}_r^*\cdot e_2}\right\}, \frac{1}{\mathbf{g}^*\cdot e_2}+1\right\}\cdot D,$$

and where D is the hidden constant in our assumption of the $O(r^{-q})$ convergence of $\frac{\mathbf{g}_r^*}{r}$ to $\mathbf{g}^*$, and where $r > R'$ implies $\frac{r}{\mathbf{g}_r^*\cdot e_2} < \frac{1}{\mathbf{g}^*\cdot e_2}+1$ (similarly to before, R' exists since we assume $\frac{\mathbf{g}_r^*}{r} \to \mathbf{g}^*$). Note that $\frac{1}{\mathbf{g}^*\cdot e_2} \to \infty$ as $\theta^* \to 0$ or $\theta^* \to \pi$, and so C does as well. Next, first note that

$$|u(\mathbf{x}) - u_{märz}(\mathbf{x})| = |u_0(\Pi_{\theta_r^*}(\mathbf{x})) - u_0(\Pi_{\theta^*}(\mathbf{x}))| \text{ for all } \mathbf{x}\in D.$$

where $\Pi_{\theta_r^*}$, Π_{θ^*} are the usual transport operators defined by (6.3.3)*. Then note that we have the bound*

$$|\Pi_{\theta_r^*}(\mathbf{x}) - \Pi_{\theta^*}(\mathbf{x})| \le |\cot\theta_r^* - \cot\theta^*| \le Cr^{-q}.$$

This estimate will play the same role as our estimate on the moments of $\mathbf{X}_\tau$ in the proof of Theorem 6.3.1.

Next, similarly to the proof of Theorem 6.3.1, we divide D into bands. This time, however, we have two separate sets of bands $\{B_i\}_{i=1}^M$ and $\{\hat{B}_i\}_{i=1}^M$, each of which individually partition D. The bands B_i are the same as in the proof of theorem 6.3.1, while the bands $\hat{B}_i$ are the same as B_i except that the transport operator $\Pi_{\theta_r^*}$ has been replaced with Π_{θ^*}. That is,

$$B_i := \Pi_{\theta_r^*}^{-1}((x_i, x_{i+1}]) \quad \text{and} \quad \hat{B}_i := \Pi_{\theta^*}^{-1}((x_i, x_{i+1}]).$$

As usual we define $B_{i,h} := B_i \cap \Omega_h$ and $\hat{B}_{i,h} := \hat{B}_i \cap \Omega_h$. It will be convenient in a moment to have an estimate of $\|\hat{B}_{i,h}\backslash B_{i,h}\|_p$ (for $\|\hat{B}_{i,h}\cap B_{i,h}\|_p$, the trivial bound $\|\hat{B}_{i,h}\cap B_{i,h}\|_p \le 1$ suffices). To that end, note that the set difference $\hat{B}_i\backslash B_i$ is a triangle with a base of length $|\cot\theta^* - \cot\theta_r^*|$ and a height of 1. At the same time $\|\hat{B}_{i,h}\backslash B_{i,h}\|_p^p = |\hat{B}_{i,h}\backslash B_{i,h}|h^2$ can be thought of as a Riemann sum approximating Area$(\hat{B}_i\backslash B_i)$. It is elementary to show that this Riemann sum overestimates the area by at most $2h$. Hence

$$\|\hat{B}_{i,h}\backslash B_{i,h}\|_p^p \le \text{Area}(\hat{B}_i\backslash B_i) + 2h = \frac{1}{2}|\cot\theta^* - \cot\theta_r^*| + 2h \le \left(\frac{C}{2}+2\right)r^{-q},$$

and therefore

$$\|\hat{B}_{i,h}\backslash B_{i,h}\|_p \le \left(\frac{C}{2}+2\right)^{\frac{1}{p}} r^{-\frac{q}{p}} \le \left(\frac{C}{2}+2\right) r^{-\frac{q}{p}},$$

since we have assumed $h \leq r^{-q}$. Proceeding now as in the proof of Theorem 6.3.1, we note that

$$\begin{aligned}
\|u - u_{\textit{märz}}\|_p &\leq \sum_{i=1}^{M} \left\{ \|(u - u_{\textit{märz}}) 1_{\hat{B}_{i,h} \setminus B_{i,h}}\|_p + \|(u - u_{\textit{märz}}) 1_{\hat{B}_{i,h} \cap B_{i,h}}\|_p \right\} \\
&\leq C_{u_0, \mathcal{U}, \{U_i\}} \sum_{i=1}^{M} \left\{ |\cot\theta^* - \cot\theta_r^*|^{s' \wedge 1} \|1_{\hat{B}_{i,h} \setminus B_{i,h}}\|_p \right. \\
&+ \left. |\cot\theta^* - \cot\theta_r^*|^{s \wedge 1} \|1_{\hat{B}_{i,h} \cap B_{i,h}}\|_p \right\} \\
&\leq C_{u_0, \mathcal{U}, \{U_i\}} \sum_{i=1}^{M} \left\{ C r^{-q(s' \wedge 1)} \cdot \|1_{\hat{B}_{i,h} \setminus B_{i,h}}\|_p + C r^{-q(s \wedge 1)} \cdot \|1_{\hat{B}_{i,h} \cap B_{i,h}}\|_p \right\} \\
&\leq C_{u_0, \mathcal{U}, \{U_i\}} \sum_{i=1}^{M} \left\{ C \left(\frac{C}{2} + 1 \right) r^{-q\left(s' + \frac{1}{p}\right) \wedge \left(1 + \frac{1}{p}\right)} + C r^{-q(s \wedge 1)} \cdot 1 \right\} \\
&\leq M C_{u_0, \mathcal{U}, \{U_i\}} C \left(\frac{C}{2} + 1 \right) \left(r^{-q\left\{\left(s' + \frac{1}{p}\right) \wedge \left(1 + \frac{1}{p}\right)\right\}} + r^{-q(s \wedge 1)} \right) \\
&\leq 2M C_{u_0, \mathcal{U}, \{U_i\}} C \left(\frac{C}{2} + 1 \right) r^{-q\left\{\left(s' + \frac{1}{p}\right) \wedge s \wedge 1\right\}} \\
&:= K_2 r^{-q\left\{\left(s' + \frac{1}{p}\right) \wedge s \wedge 1\right\}}.
\end{aligned}$$

That K_2 also satisfies the claimed properties follows from its relationship to $C_{u_0, \mathcal{U}, \{U_i\}}$ and C. □

A.8 Additional details on coherence transport and the angular spectrum

In Section 6.6.2 we related the limiting transport direction $\mathbf{g}_r^* = \lim_{\mu \to \infty} \mathbf{g}_{\mu,r}^*$ of coherence transport to the angular spectrum $\Theta(b_r^-)$ of b_r^- defined by (6.6.7). More precisely, first we connected $\mathbf{g}_r^*$ to the set of minimizers Ψ within b_r^- of the orthogonal distance to the line $L_{\mathbf{g}} = \{\lambda \mathbf{g} : \lambda \in \mathbb{R}\}$, where $\mathbf{g}$ is the guidance direction of coherence transport. Then we claimed that Ψ and $\Theta(b_r^-)$ are related. Now is the time to prove that claim. We will do this in Proposition A.8.3 not just for b_r^-, but for a general finite subset $A \subseteq \mathbb{Z}^2 \cap \{y \leq -1\}$. To do this, however, first we generalize the concept of angular spectrum to a general subset $A \subseteq \mathbb{Z}^2$.

Definition A.8.1. *Given $A \subseteq \mathbb{Z}^2$ we define the angular spectrum of A by*

$$\Theta(A) = \{\theta \in [0, \pi) : \theta = \theta(\mathbf{y}) \text{ for some } \mathbf{y} \in A \backslash \{\mathbf{0}\}\} \tag{A.8.1}$$

If A is finite it follows that $\Theta(A)$ is as well, and we write

$$\Theta(A) = \{0 \leq \theta_1 < \theta_2 < \ldots < \theta_n < \pi\}.$$

See Figure A.4(b) for an illustration of $\Theta(A)$ in the case $A = b_r^-$.

Once again we have defined $\Theta(A)$ modulo π to reflect the fact that $\mathbf{g}_r^*$ and $-\mathbf{g}_r^*$ define the same transport equation. The characterization of $\Theta(A)$ is of interest in and of itself and has been studied for $A = b_r$ by the likes of Erdös [27] and many others, see for example [21] (they, however, do not define it modulo π).

Remark A.8.2. *The point of generalizing the concept of angular spectrum and generalizing Proposition A.8.3 from b_r^- to a general $A \subseteq \mathbb{Z}^2 \cap \{y \leq -1\}$ is so that we can show (Corollary A.8.6) that our kinking results for coherence transport from Section 6.6.2 continue to hold, essentially unchanged, if coherence transport is modified to sum over a discrete square, for example, rather than a discrete ball.*

Proposition A.8.3. *Let $A \subseteq \mathbb{Z}^2 \cap \{y \leq -1\}$ obey $|A| < \infty$, and let $\Theta(A) = \{\theta_1, \theta_2, \ldots, \theta_n\}$ denote the angular spectrum of A, and assume $n = |\Theta(A)| \geq 2$ in order to avoid degenerate cases (that is, the elements of A do not all sit on a single line through the origin). Let $\mathbf{g}_\theta = (\cos\theta, \sin\theta)$ and denote by Ψ_θ the set of minimizers of $|\mathbf{g}_\theta^\perp \cdot \mathbf{y}|$ over $\mathbf{y} \in A$ (that is, the point(s) in A minimizing the orthogonal distance to the line $L_{\mathbf{g}_\theta}$. Given $\mathbf{y} \in A$, we say that $\mathbf{y}$ is a singleton minimizer if there is some $\theta \in [0, \pi)$ for which $\Psi_\theta = \{\mathbf{y}\}$. Let $Y := \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{n'}\}$ denote the set of all singleton minimizers, ordered so that $\theta(\mathbf{y}_i) \leq \theta(\mathbf{y}_{i+1})$ for all $i = 1, \ldots n' - 1$. Then $n' = n$,*

$$\Theta(A) = \{\theta(\mathbf{y}_1), \theta(\mathbf{y}_2), \ldots, \theta(\mathbf{y}_n)\},$$

and moreover $\theta_i = \theta(\mathbf{y}_i)$ for all $i = 1, \ldots, n$. Finally, each singleton minimizer $\mathbf{y}_i$ is the shortest *vector in A such that $\theta(\mathbf{y}) = \theta_i$, that is for every $\mathbf{y} \in A$ such that $\theta(\mathbf{y}) = \theta_i$, we have $\|\mathbf{y}\| \geq \|\mathbf{y}_i\|$.*

Proof. Let $\theta_i \in \Theta(A)$. Our main objective is to show that $\theta_i = \theta(\mathbf{y}_i)$. To achieve that, it suffices to show that the *sets* $\Theta(A)$ and $\{\theta(\mathbf{y}_1), \theta(\mathbf{y}_2), \ldots, \theta(\mathbf{y}_{n'})\}$ are equal, since from here it follows that $n' = n$, and then the desired identity follows from the ordering property $\theta(\mathbf{y}_i) \leq \theta(\mathbf{y}_{i+1})$ for all $i = 1, \ldots, n-1$. Our secondary objective, to show that $\mathbf{y}_i$ is the shortest vector in A obeying $\theta(\mathbf{y}) = \theta_i$ is something that will be proved along the way.

For the first step, the inclusion $\Theta(A) \supseteq \{\theta(\mathbf{y}_1), \theta(\mathbf{y}_2), \ldots, \theta(\mathbf{y}_{n'})\}$ is obvious and follows from the definition of $\Theta(A)$. Hence it suffices to prove

$$\Theta(A) \subseteq \{\theta(\mathbf{y}_1), \theta(\mathbf{y}_2), \ldots, \theta(\mathbf{y}_{n'})\}. \tag{A.8.2}$$

Still fixing $\theta_i \in \Theta(A)$, by definition we have $\theta_i = \theta(\mathbf{y})$ for some $\mathbf{y} \in A$. In fact, we have $\theta_i = \theta(\mathbf{y})$ for all $\mathbf{y} \in \Psi_{\theta_i}$, which in this case is a set of vectors that are all scalar multiples of $\mathbf{g}_{\theta_i}$ and hence all of which obey $|\mathbf{g}_{\theta_i}^\perp \cdot \mathbf{y}| = 0$. Define the functions $\Delta(\theta)$ and $\delta(\theta)$ by

$$\delta(\theta) := \max_{\mathbf{y} \in \Psi_\theta} |\mathbf{g}_\theta^\perp \cdot \mathbf{y}|$$

$$\Delta(\theta) := \begin{cases} \min_{\mathbf{y} \in A \backslash \Psi_\theta} |\mathbf{g}_\theta^\perp \cdot \mathbf{y}|. & \text{if } A \backslash \Psi_\theta \neq \emptyset \\ \delta(\theta) & \text{otherwise.} \end{cases}$$

Then $\delta(\theta)$ and $\Delta(\theta)$ each depend continuously on θ. Moreover, it is straightforward to show that $\Delta(\theta_i) > \delta(\theta_i) = 0$, since we have assumed $|\Theta(A)| \geq 2$ (this condition could only ever be violated if all elements of A were scalar multiples of one another). By continuity, it follows that for $|\theta - \theta_i|$ sufficiently small we have $\delta(\theta) < \Delta(\theta)$, which means that $\Psi_\theta \subseteq \Psi_{\theta_i}$. But for $|\theta - \theta_i| \leq \frac{\pi}{2}$ and for $\mathbf{y} \in \Psi_{\theta_i}$, we have the explicit formula

$$|\mathbf{g}_\theta^\perp \cdot \mathbf{y}| = \|\mathbf{y}\| \sin|\theta - \theta_i|.$$

This is obviously minimized by whichever $\mathbf{y}^* \in \Psi_{\theta_i}$ is shortest - i.e. $\|\mathbf{y}^*\| \leq \|\mathbf{y}\|$ for all $\mathbf{y} \in \Psi_{\theta_i}$. Moreover, since A is contained in the lower half plane we know this minimizer is unique. Hence $\Psi_\theta = \{\mathbf{y}^*\}$, which makes $\mathbf{y}^*$ a singleton minimizer. Since $\theta_i = \theta(\mathbf{y}^*)$, this proves the desired inclusion (A.8.2), and we have already proven our secondary claim on the length of $\mathbf{y}^*$ being minimal. □

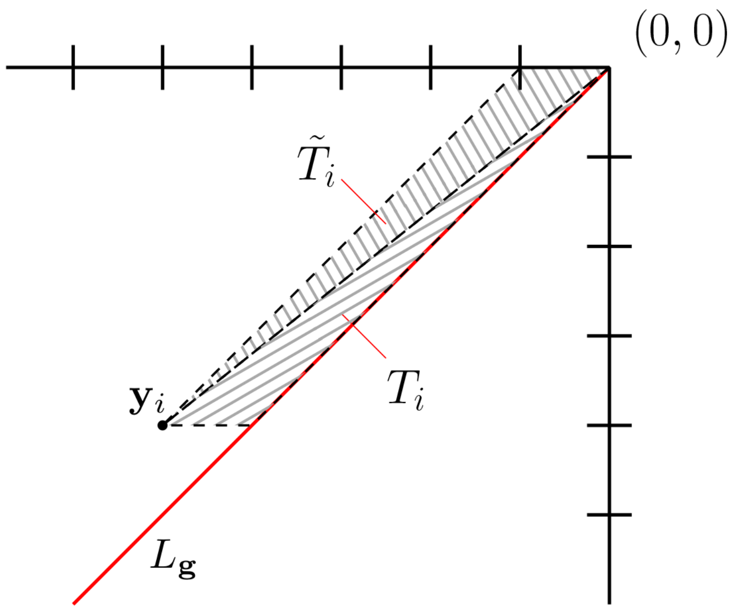


Figure A.4: **Proving that the point casting the shallowest angle on $L_{\mathbf{g}}$ from above is also the point minimizing the orthogonal distance from above:** Given $\mathbf{y}_i \in A \subseteq \mathbb{Z}^2 \cap \{y \leq -1\}$ and line $L_{\mathbf{g}} = \{\lambda \mathbf{g} : \lambda \in \mathbb{R}\}$, $\mathbf{g} = (\cos\theta, \sin\theta)$ passing through the origin, we define the (open) triangles T_i, $\tilde{T}_i$ to be the unique pair of open triangles with one side parallel to $L_{\mathbf{g}}$, another side horizontal, and a third side equal to the ray from the origin to $\mathbf{y}_i$. A symmetry-based argument in Proposition A.8.4 shows that, under modest hypotheses on A, the triangle T_i contains a lattice point (element of $\mathbb{Z}^2$) if and only if $\tilde{T}_i$ does.

Our next claim was a formula for Ψ valid when $\theta_i < \theta < \theta_{i+1}$ for two consecutive members $\theta_i, \theta_{i+1} \in \Theta(A)$, when $0 := \theta_{0,1} < \theta < \theta_1$, and when $\theta_n < \theta < \theta_{n,n+1} := \pi$. Proposition A.8.4 derives this formula, under the assumption that A can be described a union of discrete rectangles on or below the line $y = -1$ and straddling the line $x = 0$. This includes the case $A = b_r^-$, but also covers a number of other cases, as mentioned in Remark A.8.2. Credit for this proposition goes to Dömötör Pálvölgyi, who had the critical idea of using a symmetry based argument [26].

Proposition A.8.4. *Let $A \subseteq \mathbb{Z}^2 \cap \{y \leq -1\}$ be a finite union of discrete rectangles of the form*

$$A = \bigcup_{k=1}^{K} [a_k, b_k] \times [c_k, d_k] \cap \mathbb{Z}^2$$

where $-\infty < a_k \leq 0 \leq b_k < \infty$, $-\infty < c_k \leq d_k \leq -1$ for all k. Let

$$\Theta(A) := \{\theta_1, \theta_2, \ldots, \theta_n\}$$

denote the angular spectrum of A, let $\mathbf{g} = (\cos\theta, \sin\theta)$, and let $Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$ be the set of singleton minimizers defined in Proposition A.8.3 of $|\mathbf{g}^\perp \cdot \mathbf{y}|$ over A as θ varies from 0 to π. For each $1 \leq i \leq n-1$, define the transition angle $\theta_{i,i+1} \in (\theta_i, \theta_{i+1})$ by

$$\theta_{i,i+1} = \theta(\mathbf{y}_i + \mathbf{y}_{i+1}).$$

Define also $\theta_{0,1} := 0$ *and* $\theta_{n,n+1} = \pi$ *for convenience. Then*

$$\Psi = \begin{cases} \{\mathbf{y}_i\} & \text{if } \theta_i < \theta < \theta_{i,i+1} \quad \text{for some } i = 1, \dots, n \\ \{\mathbf{y}_i, \mathbf{y}_{i+1}\} & \text{if } \theta = \theta_{i,i+1} \quad \text{for some } i = 1, \dots, n-1 \\ \{\mathbf{y}_{i+1}\} & \text{if } \theta_{i,i+1} < \theta < \theta_{i+1} \quad \text{for some } i = 0, \dots, n-1 \end{cases}$$

Proof. The bulk of the work is to prove that if $\theta_i < \theta < \theta_{i+1}$, then

$$\Psi_\theta := \operatorname{argmin}_{\mathbf{y}\in A} |\mathbf{g}^\perp \cdot \mathbf{y}| \subseteq \{\mathbf{y}_i, \mathbf{y}_{i+1}\}.$$

Once this is established, since we evidently have $\Psi_\theta = \{\mathbf{y}_i\}$, $\Psi_\theta = \{\mathbf{y}_{i+1}\}$ for θ sufficiently close to θ_i and θ_{i+1} respectively, it follows that that

$$\Psi_\theta = \begin{cases} \{\mathbf{y}_i\} & \text{if } \theta < \theta_c \\ \{\mathbf{y}_i, \mathbf{y}_{i+1}\} & \text{if } \theta = \theta_c \\ \{\mathbf{y}_{i+1}\} & \text{if } \theta > \theta_c. \end{cases}$$

where θ_c is defined by $|\mathbf{g}^\perp \cdot \mathbf{y}_i| = |\mathbf{g}^\perp \cdot \mathbf{y}_{i+1}|$. One can readily show this is equivalent to $\theta_c = \theta(\mathbf{y}_i + \mathbf{y}_{i+1})$.

To prove that $\Psi_\theta := \operatorname{argmin}_{\mathbf{y}\in A} |\mathbf{g}^\perp \cdot \mathbf{y}| \subseteq \{\mathbf{y}_i, \mathbf{y}_{i+1}\}$ as claimed, consider the open triangles T_i, $\tilde{T}_i$ defined in terms of $\mathbf{y}_i$ as shown in Figure A.4, as well as open triangles T_{i+1}, $\tilde{T}_{i+1}$ defined in the same way in terms of $\mathbf{y}_{i+1}$. The triangles T_i and T_{i+1} each have empty intersection with A, as $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$ are the elements of A that cast the shallowest angles on $L_\mathbf{g}$ from above and below. To prove $\Psi_\theta \subseteq \{\mathbf{y}_i, \mathbf{y}_{i+1}\}$, we need to show that $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$ are also the two *closest* points in A to $L_\mathbf{g}$. This amounts to proving that the triangles $\tilde{T}_i$ and $\tilde{T}_{i+1}$ have empty intersection with A as well.

To show this, first note that our assumptions on A imply that the intersection of each of our four triangles with A is equal to their intersection with $\mathbb{Z}^2$ as a whole (because A has no "holes"). Second, note that the map

$$\mathbf{F}(\mathbf{x}) = \mathbf{y}_i - \mathbf{x}$$

is a bijection of the plane taking T_i to $\tilde{T}_i$ such that $\mathbf{F}(\mathbb{Z}^2) = \mathbb{Z}^2$. Hence $\tilde{T}_i$ contains a lattice point if and only if T_i does. But

$$T_i \cap \mathbb{Z}^2 = T_i \cap A = \emptyset$$

by assumption, and so

$$\tilde{T}_i \cap \mathbb{Z}^2 = \tilde{T}_i \cap A = \emptyset$$

as well. This proves the claim for $\tilde{T}_i$ and the proof for $\tilde{T}_{i+1}$ is analogous. The remaining cases $0 := \theta_{0,1} < \theta < \theta_1$ and $\theta_n < \theta < \theta_{n,n+1} := \pi$ are straightforward and left as an exercise. □

Proposition A.8.4 has a couple of straightforward corollaries. The first is our claim from 6.6.2 that Ψ *is a singleton set for all but finitely many* θ*. This is obvious from the statement of the proposition (which gives an expression for* Ψ *for all but finitely many* θ*) and requires no proof. The second corollary generalizes our formula for* θ_r^* *from Section 6.6.2, and uses the following observation, which we also used in Section 6.6.2 and owe a proof of.*

Observation A.8.5. *Let* $\mathbf{v}$, $\mathbf{w}$ *be unit vectors in* S^1*. Then*

$$\theta(\mathbf{v} + \mathbf{w}) = \frac{\theta(\mathbf{v}) + \theta(\mathbf{w})}{2}.$$

Proof. This is simplest if we work in complex arithmetic, that is, we write $\mathbf{v} = e^{i\psi}$ and $\mathbf{w} = e^{i\phi}$ for some $\psi, \phi \in [0, 2\pi)$. However, by symmetry we may assume $\mathbf{v} = 1$ (otherwise rotate the plane). Hence, it suffices to prove

$$\frac{1 + e^{i\phi}}{|1 + e^{i\phi}|} = e^{i\frac{\phi}{2}}.$$

But this follows from the following simple manipulation:

$$1 + e^{i\phi} = e^{i\frac{\phi}{2}}(e^{-i\frac{\phi}{2}} + e^{i\frac{\phi}{2}}) = 2\cos\left(\frac{\phi}{2}\right) e^{i\frac{\phi}{2}}.$$

□

The following corollary shows that coherence transport-like algorithms, which use the same weights but replace $B_{\epsilon,h}(\mathbf{x})$ with a different set of pixels (a finite union of discrete rectangles) exhibit similar kinking behaviour in the limit $\mu \to \infty$.

Corollary A.8.6. *Suppose we inpaint $D = (0, 1]^2$ using Algorithm 1 with boundary data $u_0 : \mathcal{U} \to \mathbb{R}^d$ and suppose the symmetry assumptions of Section 6.1 hold as usual. Assume our stencil a_r^* is of the form*

$$a_r^* = \bigcup_{k=1}^{K} [a_k, b_k] \times [c_k, d_k] \cap \mathbb{Z}^2$$

where $-\infty < a_k \leq 0 \leq b_k < \infty$, $c_k \leq d_k \leq -1$ for all k. Let

$$\Theta(a_r^*) := \{\theta_1, \theta_2, \ldots, \theta_n\}$$

denote the angular spectrum of a_r^*, let $\mathbf{g} = (\cos\theta, \sin\theta)$, assume we use as stencil weights the weights of März (2.5.2), and denote by $\mathbf{g}_{\mu,r}^*$ the limiting transport direction from Theorem 6.3.1. Let $\mathbf{g}_r^* = \lim_{\mu\to\infty} \mathbf{g}_{\mu,r}^*$ and define $\theta_r^* := \theta(\mathbf{g}_r^*)$. Then

$$\theta_r^* = \begin{cases} \frac{\pi}{2} & \text{if } \theta = 0 \\ \theta_i & \text{if } \theta_{i-1,i} < \theta < \theta_{i,i+1} \text{ for some } i = 1, \ldots n \\ \frac{\theta_i + \theta_{i+1}}{2} & \text{if } \theta = \theta_{i,i+1} \text{ for some } i = 1, \ldots n \end{cases} \tag{A.8.3}$$

Proof. This follows from Proposition A.8.4 and observation A.8.5 in exactly the same way as when showed this for the special case $a_r^* = b_r^-$ in Section 6.6.2. □

We conclude this appendix with a remark on a practical algorithm for computing the angular spectrum $\Theta(A)$ given $A \subseteq \mathbb{Z}^2$ satisfying the hypotheses of Proposition A.8.3. This algorithm was used to generate the theoretical limiting curves for θ_r^* for coherence transport in Section 6.6.2.

Remark A.8.7. *Given $A \subseteq \mathbb{Z}^2$ satisfying the hypotheses of Proposition A.8.3, a simple algorithm for computing the angular spectrum $\Theta(A)$ and singleton minimizers Y is as follows:*

1. *Starting with $Y^* = \emptyset$, go through each $\mathbf{y} \in A$ and find the unique $\mathbf{y}' \in A$ such that $\theta(\mathbf{y}) = \theta(\mathbf{y}')$ and $\mathbf{y}'$ is of minimal length. If $\mathbf{y}'$ is not already in Y^*, add it.*

2. *For each $\mathbf{y} \in Y^*$, compute $\theta(\mathbf{y})$. Sort Y^* according to $\theta(\mathbf{y})$. The sorted list Y^* is now equal to Y, and the sorted list of angles is $\Theta(A)$.*

A.9 List of Examples

In this appendix we describe in detail the examples used in some of the numerical experiments in Section 6.8.

Boundary Data. *Our first objective is to build boundary data satisfying the assumptions listed in Section 6.3 and illustrated in Figure 6.4. To that end, we let* $\mathcal{U} = (0,1] \times (-\delta, 0]$ *as usual and define the sets* $\{U_i\}_{i=1}^2$ *as*

$$U_1 := [0, 0.25) \times (-\delta, 0] \cup (0.75, 1] \times (-\delta, 0] \quad U_2 := (0.25, 0.75) \times (-\delta, 0]$$

for some $\delta > (r+2)h$*. Next, to test that our bounds in Theorem 6.3.1 are sharp, what we would like to do is build a family* $\mathcal{F}$ *of boundary data* u_0*, parameterized by* $0 < s$, $0 \le s' \le s$*, such that* $u_0 \in W^{\nu,\infty}(U_i)$ *for* $i = 1, 2$ *iff* $\nu \in [0, s]$ *and* $u_0 \in W^{\nu',\infty}(\mathcal{U})$ *iff* $\nu' \in [0, s']$*. We will almost, but not quite be able to do this. Specifically, if* $s \in \mathbb{N}$*, we will instead have* $u_0 \in W^{\nu,\infty}(U_i)$ *for* $i = 1, 2$ *iff* $\nu \in [0, s)$*. However, we do not expect this detail to have a serious impact on our convergence rates (indeed, our numerical experiments in Appendix A.10 - with a few possible exceptions - suggest that it doesn't matter). To accomplish this we set*

$$u_0(x, y) = w_s(x) + H_{s'}(x) \tag{A.9.1}$$

where w_s *is Fourier series*

$$w_s(x) = \sum_{n=1}^{\infty} 2^{-sn} \cos(2^n \pi x), \tag{A.9.2}$$

which for $0 < s \le 1$ *is an example of a nowhere-differentiable* Weierstrass function *[34]. The other component* $H_{s'}$ *is the (possibly smooth) step function*

$$H_{s'}(x) := 1 - 1\left(x \le \frac{1}{4}\right) \tanh\left(20\left[\frac{1}{4} - x\right]\right)^{s'} - 1\left(x \ge \frac{3}{4}\right) \tanh\left(20\left[x - \frac{3}{4}\right]\right)^{s'}.$$

See Figure A.5 for an illustration of w_s *and* $H_{s'}$ *for various values of* s *and* s'*. The following Lemma establishes that* u_0 *given by* (A.9.1) *has the claimed regularity properties.*

Lemma A.9.1. *Let* $0 < s$, $0 \le s' \le s$*. Let* $u_0 := w_s + H_{s'}$ *be defined as in* (A.9.1) *and illustrated in Figure A.5. Then* $u_0 \in W^{\nu,\infty}(U_i)$ *for* $i = 1, 2$ *iff* $\nu \in [0, s]$ *if* $s \notin \mathbb{N}$*, and iff* $\nu \in [0, s)$ *if* $s \in \mathbb{N}$*. At the same time,* $u_0 \in W^{\nu',\infty}(\mathcal{U})$ *iff* $\nu' \in [0, s']$*.*

Proof. It suffices to prove that $w_s \in W^{\nu,\infty}(\mathcal{U})$ for $i = 1, 2$ iff $\nu \in [0, s]$ if $s \notin \mathbb{N}$, and iff $\nu \in [0, s)$ if $s \in \mathbb{N}$, while $H_{s'} \in W^{\nu',\infty}$ iff $\nu' \in [0, s']$. The claims involving $H_{s'}$ follow from the fact that this function is C^∞ everywhere except $x = 0.25$ and $x = 0.75$, along with the identity $\tanh(x) \le x^\alpha$ for all $x \ge 0$ iff $0 \le \alpha \le 1$. The details are left as an exercise.

On the other hand, the regularity of w_s has been studied for $0 < s \le 1$ by a number of authors. In particular, our claim for $0 < s < 1$ follows from [34, Theorem 1.31], while for $s = 1$ it is an easy corollary of a statement on [34, Pg. 311]. The case $s > 1$ is not typically considered, but it is a simple exercise to show that in this case $w_s \in W^{s,\infty}((0,1])$ whenever $s \notin \mathbb{N}$, and $w_s \in W^{s-\epsilon,\infty}((0,1])$ for all $\epsilon > 0$ if $s \in \mathbb{N}$. To see this, note that if $s \notin \mathbb{N}$ and $\lfloor s \rfloor = n \in \mathbb{N}$, then the nth derivative of the partial sums in (A.9.2) converges uniformly by the Weierstrass M-test. Thus the nth derivative of w_s exists - moreover, it is related in a simple way to the Weirstrass function $w_{s-\lfloor s \rfloor}$ (in some cases with sines replacing the cosines, but this makes no difference), and hence $w_s^{(n)} \in W^{s-\lfloor s \rfloor,\infty}((0,1]))$. The argument for $s \in \mathbb{N}$ is similar. □

Neighborhood and weights. *We consider three separate pairings of neighborhoods* $A_{\epsilon,h}$ *and weights*

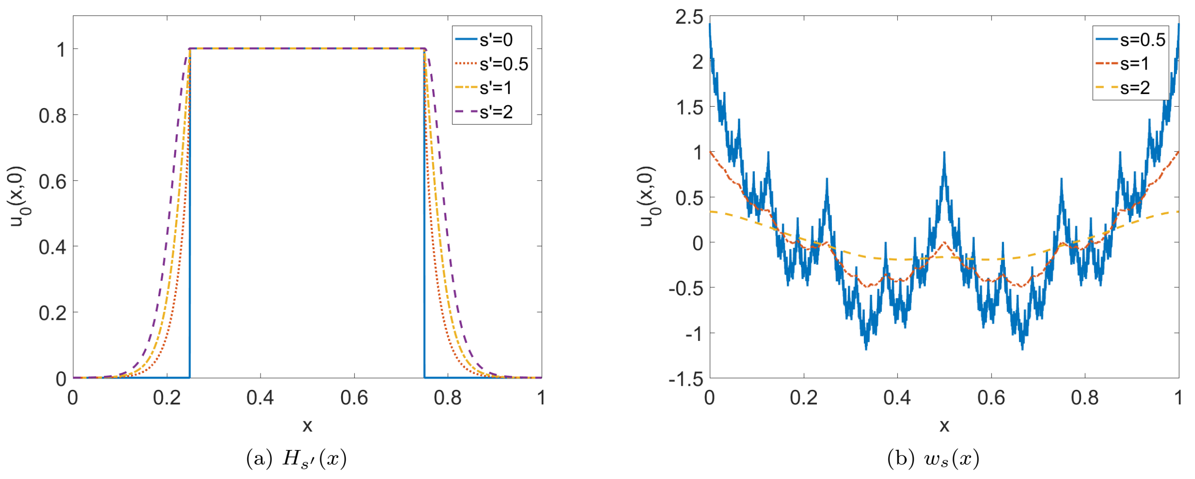


Figure A.5: **Building boundary data with desired smoothness properties:** Our boundary data $u_0(x,y) = w_s(x) + H_{s'}(x)$ is a sum of two components. The first component $H_{s'}$ is smooth everywhere with the exception of two isolated points, while the second component w_s has uniform (but potentially very low) regularity. When $s' = 0$, the first component $H_{s'}$ is a step function. When $0 < s \leq 1$, the second component w_s is an example of a nowhere differentiable Weierstrass function [34]. See Lemma A.9.1.

$w_\epsilon(\cdot,\cdot)$.

Example 1: Coherence Transport with constant transport direction

In this case we choose as our weights $w_\epsilon(\mathbf{x},\mathbf{y})$ *März's weights* (2.5.2) *and neighborhood* $\mathcal{V}_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$*, that is, we use the same neighborhood and weights as in coherence transport. We fix* $\mathbf{g} = (\cos(20°), \sin(20°))$ *and test two different values of* μ*, namely* $\mu = 10$ *and* $\mu = 50$.

Example 2: Gaussian weights with a box neighbourhood

In this example for our weights we choose the offset Gaussian

$$w_\epsilon(\mathbf{x},\mathbf{y}) = \exp\left(-5\left\|\frac{\mathbf{x}-\mathbf{y}}{\epsilon} + \left(\frac{1}{2},\frac{1}{2}\right)\right\|^2\right). \tag{A.9.3}$$

and as a neighborhood $A_{\epsilon,h}(\mathbf{x})$ *we use the discrete square*

$$\mathcal{A}_{\epsilon,h}(\mathbf{x}) = \mathbf{x} + \{-rh, (-r+1)h \ldots (r-1)h, rh)\}^2.$$

Example 3: Guidefill with a smoothly varying Transport Field

In this example we consider Guidefill - that is $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ *and* $w_\epsilon(\cdot,\cdot)$ *given by* (2.5.2) *- with the spatially varying transport field*

$$\mathbf{g}(x,y) = \left(\frac{4xy}{1+2y^2}, 1\right). \tag{A.9.4}$$

We fix $r = 3$ *and* $\mu = 50$*. Assuming our results continue to hold in this case, from the discussion in Section 6.6.2 for* μ *sufficiently large we expect*

$$\mathbf{g}^*_r(x,y) = \mathbf{g}(x,y)$$

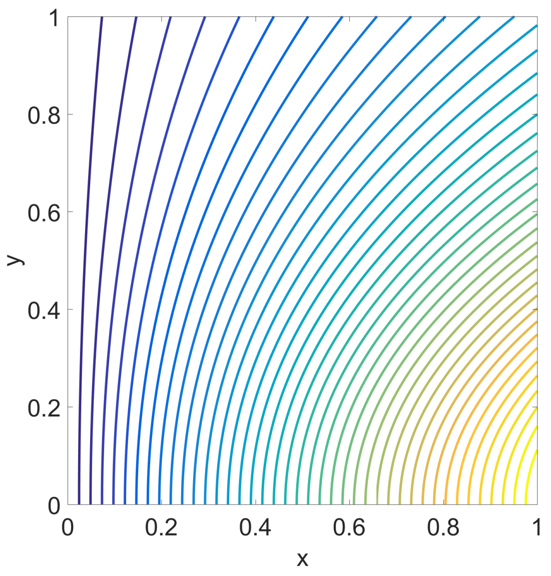


Figure A.6: **A smoothly varying guideance field:** Characteristic curves of the smoothly varying transport field $\mathbf{g}(x, y)$ given by (A.9.4). The transport operator $\Pi(\mathbf{x})$ follows the characteristic passing through $\mathbf{x}$ backwards to image data located on the boundary of the inpainting domain at $(0, 1] \times \{0\}$.

to machine precision provided $\mathbf{g}(x, y)$ never makes an angle shallower than $\theta_c = \arctan(\frac{1}{\sqrt{10}})$ with the x-axis. Indeed, in this case we can easily check that

$$\inf_{(x,y)\in(0,1]^2} \theta(\mathbf{g}(x, y)) = \arctan\left(\frac{5}{4\sqrt{2}}\right) > \theta_c.$$

In this case we know that the characteristics of $\mathbf{g}(x, y) = \mathbf{g}_r^*(x, y)$ are the level curves of the function $f(x, y) = x/(1 + 2y^2)$ (See Figure A.6). This allows us to write the transport operator in this case explicitly as

$$\Pi(x, y) = \left(\frac{x}{1 + 2y^2}, 0\right).$$

The continuum limit is then given in terms of our boundary data u_0 by $u(\mathbf{x}) = u_0(\Pi(\mathbf{x}))$ as usual.

A.10 Experiment III : Verifying Theorem 6.3.1

Our final experiment aims to verify the correctness of bounds in Theorem 6.3.1, and to check if they are tight.

For each of the examples listed above we first derive the continuum limit u predicted Theorem 6.3.1, then compute u_h for a sequence of grids with resolution $h = 2^{-n}$ with $r = 3$ fixed. We then look at the ratio

$$R_h := \frac{-1}{\log 2} \log\left(\frac{\|u - u_{2h}\|_p}{\|u - u_h\|_p}\right). \tag{A.10.1}$$

Assuming the bounds in Theorem 6.3.1 are asympototically tight, we expect

$$R_h \xrightarrow[h\to 0]{} \alpha \quad \textit{where } \alpha := \frac{s}{2} \wedge \left(\frac{s'}{2} + \frac{1}{2p}\right) \wedge 1 \tag{A.10.2}$$

for stencil weights that are non-degenerate (assign mass to more than one $\mathbf{y} \in a_r^*$*), and*

$$\alpha = s \wedge \left(s' + \frac{1}{p}\right) \wedge 1 \tag{A.10.3}$$

otherwise (recall (6.7.2) *and the discussion at the start of Section 6.7).*

Examples 1 to 3 all use non-degerate stencil weights, with the exception of example 1 with $\mu = 50$*, where the weights are degenerate to machine precision. We therefore expect convergence rates given by* (A.10.3) *in this case and by* (A.10.2) *in every other case. Although Theorem 6.3.1 is not technically applicable to Example 3 due to the presence of spatially varying weights, we include this example as a test to see if our bounds continue to hold in this case. It appears that they do.*

Table A.1 provides a comparison of R_h *with the expected convergence rate* α *for a variety of choices of* s*,* s' *and* p*, and for neighborhoods and neighborhood weights given by Examples 1 and 3. Results for Example 2 and for additional values of* s *and* s' *are analogous but omitted.*

Table A.1 suggests that our bounds are tight asymptotically *as* $h \to 0$*, with the resolution required to enter the asymptotic regime apparently dependent on the regularity of* u_0*. In particular, when* u_0 *is very rough (e.g.* $s = 0.5$*,* $s' = 0$*) we need to use a grid with* $|\Omega_h|$ *greater than a billion before* R_h *and* α *agree to a one decimal place. At the opposite extreme, when* u_0 *very smooth (e.g.* $s = s' = 2$*) we enter the asymptotic regime much earlier. In nearly every case the experimental rate* R_h*, possibly after some initial oscillations, monotically approaches the predicted value* α *(albeit rather slowly in some cases). There are, however, two exceptions, namely the case of coherence transport with* $\mu = 10$ *and* $\mathbf{g} = (\cos 20°, \sin 20°)$ *applied to the boundary data* (A.9.1) *parameterized by* $s = 1$ *and* $s' = 0.5$*, with* $p \in \{2, \infty\}$*. In these two cases we start above the expected* α *and then shoot past it. This may, we concede, have something to do with the exceptional nature of the case* $s \in \mathbb{N}$ *pointed out in Lemma A.9.1. Or it may be that in these cases we need* h *to be extremely small before we enter the asymptotic regime. In any event, the overall message appears to be that the rates from Theorem 6.3.1 correspond well to experimental rates measured in practice. Also note that our results do not appear to depend on whether or not we use the constant weights as in Example 1 or the smoothly varying weights from Example 3. This suggests that the hypotheses of Theorem 6.3.1 can be weakened, which is something we aim to do in the future.*

Table A.1: Comparison of experimental convergence rate R_h (A.10.1) with the theoretical rate predicted by Theorem 6.3.1, for a variety of choices of neighborhoods $A_{\epsilon,h}(\mathbf{x})$, weights $w_\epsilon(\cdot,\cdot)$, boundary data u_0, and norms $\|\cdot\|_p$. In each case our boundary data is the function (A.9.1) parametrized by s and s'. For weights we use März's weights (2.5.2), with either $\mathbf{g} = (\cos(20°), \sin(20°))$ fixed or $\mathbf{g}(x, y)$ given by the smoothly varying transport field (A.9.4). We consider $\mu = 10$ and $\mu = 50$ - the latter case leads to degenerate stencil weights and a bigger α (notice, for example, $s = s' = 0.5$ for $\mu = 10$ vs $\mu = 50$). As a neighborhood we use either the discrete ball $B_{\epsilon,h}(\mathbf{x})$ used in coherence transport or the rotated ball $\tilde{B}_{\epsilon,h}(\mathbf{x})$ used by guidefill.

$A_{\epsilon,h}(\mathbf{x})$	$w(\cdot,\cdot)$	u_0	$\|\cdot\|_p$	$R_{2^{-9}}$	$R_{2^{-11}}$	$R_{2^{-13}}$	$R_{2^{-15}}$	$R_{2^{-17}}$	α
$B_{\epsilon,h}(\mathbf{x})$	$\mu = 10$, $\mathbf{g}$ constant	$s = 0.5$ $s' = 0$	$p = 1$	0.408	0.344	0.309	0.289	0.278	0.25
			$p = 2$	0.392	0.318	0.281	0.265	0.257	0.25
			$p = \infty$	0.126	0.07	0.036	0.019	0.009	0
		$s = 0.5$ $s' = 0.5$	$p = 1$	0.408	0.336	0.299	0.28	0.27	0.25
			$p = 2$	0.402	0.323	0.284	0.266	0.257	0.25
			$p = \infty$	0.253	0.181	0.228	0.235	0.239	0.25
		$s = 1.0$ $s' = 0.5$	$p = 1$	0.735	0.606	0.554	0.53	0.521	0.5
			$p = 2$	0.701	0.574	0.509	0.492	0.49	0.5
			$p = \infty$	0.437	0.308	0.255	0.237	0.232	0.25
		$s = 2$ $s' = 1$	$p = 1$	0.932	0.95	0.968	0.98	0.988	1
			$p = 2$	0.825	0.831	0.816	0.794	0.775	0.75
			$p = \infty$	0.57	0.573	0.551	0.528	0.514	0.5
	$\mu = 50$, $\mathbf{g}$ constant	$s = 0.5$ $s' = 0.5$	$p = 1$	0.497	0.494	0.497	0.499	0.5	0.5
			$p = 2$	0.5	0.493	0.496	0.498	0.5	0.5
			$p = \infty$	0.477	0.472	0.494	0.494	0.499	0.5
		$s = 1$ $s' = 0$	$p = 1$	0.92	0.937	0.947	0.954	0.96	1
			$p = 2$	0.564	0.522	0.507	0.502	0.501	0.5
			$p = \infty$	0.072	0.021	0.005	0.001	0	0
		$s = 1$ $s' = 1$	$p = 1$	0.946	0.949	0.953	0.958	0.963	1
			$p = 2$	0.94	0.946	0.952	0.958	0.962	1
			$p = \infty$	0.882	0.864	0.907	0.916	0.928	1
		$s = 2$ $s' = 0$	$p = 1$	1.003	1.001	1	1	1	1
			$p = 2$	0.517	0.504	0.501	0.5	0.5	0.5
			$p = \infty$	0.019	0.005	0.001	0	0	0
$\tilde{B}_{\epsilon,h}(\mathbf{x})$	$\mu = 50$, $\mathbf{g}$ smooth	$s = 0.5$ $s' = 0$	$p = 1$	0.366	0.349	0.329	0.308	0.29	0.25
			$p = 2$	0.357	0.323	0.298	0.277	0.263	0.25
			$p = \infty$	0.164	0.102	0.092	0.07	0.054	0
		$s = 1$ $s' = 0$	$p = 1$	0.58	0.529	0.507	0.501	0.5	0.5
			$p = 2$	0.345	0.278	0.258	0.253	0.251	0.25
			$p = \infty$	0.134	0.034	0.014	0.012	0.006	0
		$s = 1$ $s' = 1$	$p = 1$	0.724	0.6	0.533	0.509	0.503	0.5
			$p = 2$	0.726	0.586	0.522	0.503	0.499	0.5
			$p = \infty$	0.687	0.476	0.485	0.493	0.496	0.5
		$s = 2$ $s' = 2$	$p = 1$	1.001	0.997	0.997	0.997	0.998	1
			$p = 2$	0.964	0.984	0.993	0.997	0.998	1
			$p = \infty$	0.955	0.99	0.995	0.994	0.994	1