SIAM J. Imaging Sciences
Vol. 10, No. 4, pp. 2049–2090

Guidefill: GPU Accelerated, Artist Guided Geometric Inpainting for 3D Conversion of Film*

L. Robert Hocking†, Russell MacKenzie‡, and Carola-Bibiane Schönlieb†

Abstract. The conversion of traditional film into stereo 3D has become an important problem in the past decade. One of the main bottlenecks is a disocclusion step, which in commercial 3D conversion is usually done by teams of artists armed with a toolbox of inpainting algorithms. A current difficulty in this is that most available algorithms either are too slow for interactive use or provide no intuitive means for users to tweak the output. In this paper we present a new fast inpainting algorithm based on transporting along automatically detected splines, which the user may edit. Our algorithm is implemented on the GPU and fills the inpainting domain in successive shells that adapt their shape on the fly. In order to allocate GPU resources as efficiently as possible, we propose a parallel algorithm to track the inpainting interface as it evolves, ensuring that no resources are wasted on pixels that are not currently being worked on. Theoretical analyses of the time and processor complexity of our algorithm without and with tracking (as well as numerous numerical experiments) demonstrate the merits of the latter. Our transport mechanism is similar to the one used in coherence transport [F. Bornemann and T. März, *J. Math. Imaging Vision*, 28 (2007), pp. 259–278; T. März, *SIAM J. Imaging Sci.*, 4 (2011), pp. 981–1000] but improves upon it by correcting a "kinking" phenomenon whereby extrapolated isophotes may bend at the boundary of the inpainting domain. Theoretical results explaining this phenomenon and its resolution are presented. Although our method ignores texture, in many cases this is not a problem due to the thin inpainting domains in 3D conversion. Experimental results show that our method can achieve a visual quality that is competitive with the state of the art while maintaining interactive speeds and providing the user with an intuitive interface to tweak the results.

Key words. image processing, image inpainting, 3D conversion, PDEs, parallel algorithms, GPU

AMS subject classifications. 68U10, 68W10, 65M15

DOI. 10.1137/16M1103737

1. Introduction.

The increase in demand over the past decade for 3D content has resulted in the emergence of a multimillion 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 postproduction [40]. Examples of recent films converted in whole or in part include *Maleficent*, *Thor*, and *Guardians*

*Received by the editors November 15, 2016; accepted for publication (in revised form) August 17, 2017; published electronically November 21, 2017.

http://www.siam.org/journals/siims/10-4/M110373.html

Funding: The work of the first author was supported by the Cambridge Commonwealth Trust and the Cambridge Center for Analysis. The work of the third author was supported by the Leverhulme Trust project Breaking the Non-convexity Barrier, the EPSRC grants EP/M00483X/1 and EP/N014588/1, the Cantab Capital Institute for the Mathematics of Information, the CHiPS (Horizon 2020 RISE project grant), the Global Alliance project "Statistical and Mathematical Theory of Imaging," and the Alan Turing Institute.

†Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge CB2 1TN, UK (lrh30@cam.ac.uk, cbs31@cam.ac.uk).

‡Gener8 Media Corp, Vancouver V5T 1M6, BC, Canada (russell@gener8.com).

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).

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 [10] for an excellent recent overview. This pipeline has applications including 3D TV and free-viewpoint rendering [48, 18]. 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 [47, 40] for more details and discussion.

We focus in this paper 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 1(c). We go over this pipeline in detail in section 2.

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 [40]. However, while the disocclusion step arising in the first pipeline has received a lot of attention in the literature (see, for example, [10, 46, 45, 18, 48, 24, 34, 17, 26, 30] to name a few), the disocclusion step arising in the second pipeline relevant to film has received far less attention. The present paper, to the best of our knowledge, is the first paper to address it directly. While related, these two disocclusion problems have important differences. Most significantly, the facts that our pipeline comes with an explicit mask for every scene object—even occluded parts—and that we have a full 3D model instead of just a single depth map from a single viewpoint have two major consequences. First, 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 [10] or else do so based on rough heuristics [46, 45, 18, 48, 24, 34, 17, 26, 30], which, as noted in [10, sect. II.C.], tend to fail. Second, these masks give an explicit segmentation of the scene into relevant objects both in the old viewpoint and in 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 [10, sect. 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 of the user to influence the results of inpainting, which may not be a priority in the other pipeline, are important in ours.

Image and video inpainting. Image inpainting refers to the filling in of a region in an image called the inpainting domain in such a way that the result looks plausible to the human eye. Image inpainting methods can loosely be categorized as *exemplar-based* and *geometric*. The former generally operate based on some procedure for copying patches of the undamaged portion of the image into the inpainting domain, either in a single pass from the boundary inward as in Criminisi, Pérez, and Toyama [16], or iteratively as in Wexler, Shechtman, and Irani [44] and Arias et al. [3]. The choice of which patch or patches to copy into a given area of the inpainting domain is decided using a nearest neighbor search based on a patch similarity metric. Originally prohibitively expensive, a breakthrough was made in the PatchMatch algorithm [5], which provides a fast *approximate* nearest neighbor search. PatchMatch is used behind the scenes in Photoshop's famous *Content-Aware Fill* tool. On the other hand, geometric inpainting methods aim to smoothly extend image structure into the inpainting domain, typically using PDEs 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. [6], TV, TV-H^{-1}, Mumford–Shah, Cahn–Hilliard inpainting [14, 8], Euler's elastica [29, 13], as well as the joint interpolation of image values and a guiding vector field in Ballester et al. [4]. 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 [42] and coherence transport [7, 27] (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. Unlike their iterative counterparts, these two methods are both very fast, but may create "shocks"; see section 4.2. See also [39] for a comprehensive survey of geometric inpainting methods, as well as [20] for a recent survey of the field as a whole.

Geometric methods are designed to propagate structure, but fail to reproduce texture. Similarly, exemplar-based approaches excel at reproducing texture, but are limited in terms of their ability to propagate structure. A few attempts have been made at combining geometric and exemplar-based methods, such as Cao et al. [12], 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, [23]. 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 [31, 32], 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 (see section 6). Finally, the authors of [21] 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.

Related work. Our method is a (1st-order) transport-based inpainting method for the

disocclusion step in 3D conversion. Here we review the related work on both aspects.

I. *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 semiautomatic 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 work, 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 usable pixel [24], or taking a weighted average of the closest usable pixels along a small number (8–12) of fixed directions [48, 18]. Slightly more sophisticated is the approach in [34] which applies a depth-adapted variant of Telea's algorithm [42]. 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 [45, 46, 17, 26, 30, 10] (see also [10] 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 the best of our knowledge, [10] 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 [15], which copies spacetime patches into the inpainting domain via a process similar to Criminisi, Pérez, and Toyama.

II. *Inpainting based on* 1*st-order transport.* There are a small number of inpainting strategies in the literature based on the idea of 1*st-order* transport of image values along a vector field, which is either predetermined or else calculated concurrently with inpainting (the seminal work of Bertalmio et al. [6] was also based on transport, but in their case the equation was 3rd-order). While generally lower quality than their higher order counterparts, these methods have the potential to be extremely fast. The earliest of these to the best of our knowledge is Ballester et al. [4], which considers 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 note that their approach is equivalent to a 1st-order transport equation. This was the first time to the best of our knowledge that 1st-order transport was proposed as a strategy for inpainting. However, none of the approaches suggested in their paper are fast enough for our application.

Next, Telea [42] proposed filling the inpainting domain in successive shells from the boundary inward, visiting each pixel only once and assigning it a color equal to a weighted average of its already filled neighbors, resulting in a very fast algorithm. The connection to transport was not known until Bornemann and März showed that Telea's algorithm and their improvement thereof, which they called coherence transport [7, 27], both become 1st-order transport equations under a high-resolution and vanishing viscosity limit. In Telea's algorithm, the user has no control over the transport direction; it was shown in [7] to simply be equal to the local normal vector to the boundary of the inpainting domain. Coherence transport attempts to improve on this by either allowing the user to supply the desired transport direction manually

as a vector $\mathbf{g}$, or else finding "good" values for $\mathbf{g}(\mathbf{x})$ concurrently with inpainting. However, as we will see, the algorithm actually has challenges in both respects (see below). März went on in [27] to suggest improvements to coherence transport based on a carefully selected fill order, and then went on in [28] to explore in depth an issue first raised in [4]—how to make sense of the well-posedness of a 1st-order transport equation on a bounded domain with Dirichlet boundary conditions, where integral curves of the transport field may terminate at distinct points with incompatible image values.

Our contribution. Our contributions are multiple. First, while any of the disocclusion algorithms from the previous section could be adapted to our 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 [10], this is likely to lead to needless "bleeding" artifacts when pixels from the wrong object are used for inpainting. See Figure 2(c) as well as the discussion in [10, sect. II.C]. Our first contribution is to define an inpainting algorithm designed to take advantage of this extra information explicitly, which we do by making use of a set of "bystander pixels" not to be used for inpainting (section 2.1).

Second, even if the above methods were to be 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 1), there are many situations in which one can safely ignore texture.

Third, we acknowledge that in the movie industry inpainting is typically done by teams of artists who are happier if they have the ability to influence the results of inpainting, and thus we have designed our algorithm with this in mind.

Fourth, our method is a transport-based algorithm inspired by the coherence transport algorithm [7, 27], but improving upon it by correcting some of its shortcomings. Both methods proceed by measuring the orientation of image isophotes in the undamaged region near the inpainting domain and then extrapolating based on a transport mechanism. However, in the case of coherence transport both of these steps have problems. First, 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 5 as well as sections 3.2 and 3.2.1 for a discussion of this problem and our resolution. Second, once fed a desired transport direction $\mathbf{g}$ (which may or may not be accurate based on the last point), coherence transport instead transports 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 (see Figures 6, 7, and 9). This behavior, which the authors of [7, 27] appear unaware of (the theory in [7] does not account for it), is explored in section 3.3 and rigorously analyzed in section 4. We present an improved transport mechanism overcoming this problem, as well as a theoretical explanation of its origin and resolution; see Theorem 4.1. 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 can be significant for low-resolution images and wide inpainting domains, but otherwise it appears to be minimal; see section 4.1.2. Additional details on the similarities and differences between our method and coherence transport [7, 27] are presented in section 3.

In this paper we present a fast, geometric, user guided inpainting algorithm intended for use by artists for the hole-filling step of 3D conversion of film. We have designed our algorithm with two goals 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 Theorems 5.1 and 5.2. 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 those of Sun et al. [41] and Barnes et al. [5], 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 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.

Our method, which we call 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. While an extension that uses temporal information would be interesting (and is a direction we would like to explore in the future), it is outside of the scope of this paper.

Organization. In section 2 we go over a 3D conversion pipeline commonly used in film. Section 2.2 also goes over the alternative pipeline commonly appearing in the literature, highlighting some of its potential drawbacks. Next, in section 3 we present our proposed method as part of a broader class of shell-based algorithms, highlighting issues with earlier methods and how ours overcomes them. Sections 3.2.1 and 3.3 in particular focus on two kinking issues associated with coherence transport and how Guidefill overcomes them, in the latter case through the introduction of what we call "ghost pixels." Pixel ordering strategies for Guidefill are compared and contrasted with other strategies in the literature in section 3.4. Two separate GPU implementations are sketched in section 3.5. Section 4 is devoted to a

continuum analysis of our algorithm and others like it. It enables us to rigorously explain some of the strengths and shortcomings of both Guidefill and coherence transport. Our analysis is different from the analysis by Bornemann and März in [7]; we consider a different limit and uncover new behavior. In section 5 we analyze the time complexity and processor complexity of our method as a parallel algorithm. In section 6 we show the results of our method applied to a series of 3D conversion examples. Results are compared with competing methods both in terms of runtime and visual quality. At the same time, we also validate the complexity analysis of section 5. Finally, in section 7 we draw some conclusions.

Notation.

- $h =$ the width of one pixel.
- $\mathbb{Z}_h^2 := \{(nh, mh) : (n, m) \in \mathbb{Z}^2\}$.
- Given $\mathbf{x} \in \mathbb{R}^2$, we denote by $\theta(\mathbf{x}) \in [0, 2\pi)$ the counterclockwise angle $\mathbf{x}$ makes with the x-axis.
- $\Omega = [a, b] \times [c, d]$ 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 2.1) that are neither inpainted nor used for inpainting.
- $u_h : \Omega_h \backslash (D_h \cup B_h) \to \mathbb{R}^d$ is the given image (video frame).
- $D \subset \Omega := \{\mathbf{x} \in \Omega : \exists \mathbf{y} \in D_h \text{ s.t. } \|\mathbf{y} - \mathbf{x}\|_\infty < h\}$ 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.
- $\mathbf{g} : D_h \to \mathbb{R}^2$ is the guide field used to guide the inpainting.
- $A_{\epsilon,h}(\mathbf{x})$ denotes a generic discrete (but not necessarily lattice aligned) neighborhood of radius ϵ surrounding the pixel $\mathbf{x}$ and used for inpainting.[1]
- $B_{\epsilon,h}(\mathbf{x}) = \{\mathbf{y} \in \Omega_h : \|\mathbf{x} - \mathbf{y}\| \le \epsilon\}$, the choice of $A_{\epsilon,h}(\mathbf{x})$ used by coherence transport.
- $\tilde{B}_{\epsilon,h}(\mathbf{x}) = R(B_{\epsilon,h}(\mathbf{x}))$, 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.
- $\mathcal{N}(\mathbf{x}) = \{\mathbf{x} + \mathbf{y} : \mathbf{y} \in \{-h, 0, h\} \times \{-h, 0, h\}, \mathbf{y} \neq \mathbf{0}\}$ is the eight-point neighborhood of $\mathbf{x}$.
- Given $A_h \subset \mathbb{Z}_h^2$, we define the discrete (inner) boundary of A_h by

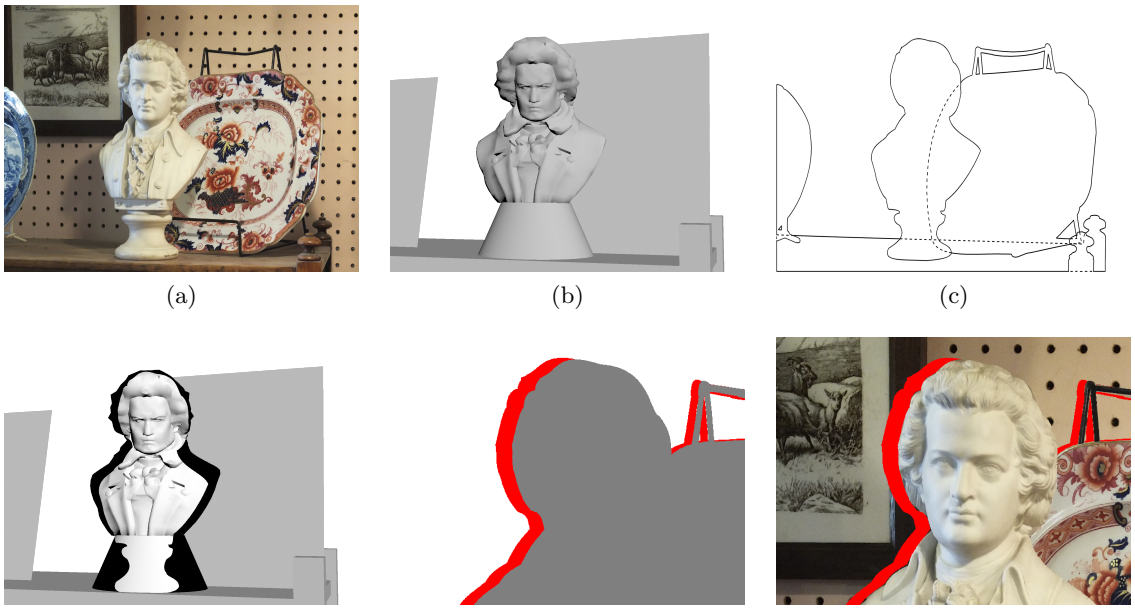
$$\partial A_h := \{\mathbf{x} \in A_h : \mathcal{N}(\mathbf{x}) \cap \mathbb{Z}_h^2 \backslash A_h \neq \emptyset\}.$$

  For convenience we typically drop the word "inner" and refer to ∂A_h as just the boundary of A_h.
- Given $A_h \subset \mathbb{Z}_h^2$, we define the discrete *outer* boundary of A_h by

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

- $\partial_{\text{active}} D_h^{(k)} \subseteq \partial D_h^{(k)}$ is the active portion of the boundary of the inpainting domain on

[1] That is, $A_{\epsilon,h}(\mathbf{x}) \subset \Omega$ is a finite set and $\|\mathbf{y} - \mathbf{x}\| \le \epsilon$ for all $\mathbf{y} \in A_{\epsilon,h}(\mathbf{x})$.



(a) (b) (c) (d) (e) (f)

Figure 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, gray are occluded in both views), and* (f) *the generated new view with red "cracks" requiring inpainting.*

step k of the algorithm. That is,

$$\partial_{\text{active}} D_h^{(k)} = \{\mathbf{x} \in \partial D_h^{(k)} : \mathcal{N}(\mathbf{x}) \cap (\Omega_h \backslash (D_h^{(k)} \cup B_h)) \neq \emptyset\}.$$

In other words, $\partial_{\text{active}} D_h^{(k)}$ excludes those pixels in $\partial D_h^{(k)}$ with no readable neighboring pixels.

- If $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$, then $\mathbf{x} \otimes \mathbf{y}$ denotes the $n \times m$ matrix *tensor product* of $\mathbf{x}$ and $\mathbf{y}$, defined by $(\mathbf{x} \otimes \mathbf{y})_{i,j} = x_i y_j$, where x_i is the ith component of $\mathbf{x}$ and y_j is the jth component of $\mathbf{y}$.

2. A 3D conversion pipeline for film. Here we briefly review a 3D conversion pipeline commonly used in film; see, for example, [47] 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. This process is often called "match-move" and is typically done with the aid of semiautomatic algorithms based on point tracking [38, 19].
2. Accurate masks must be generated for all objects and for every frame, including occluded areas (see Figure 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 [40].

3. A plausible 3D model of the scene must be generated (see Figure 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 1(d). For this reason 3D geometry is typically designed to be slightly larger than it would be in real life [47].
4. For each object, a multilabel mask must be generated assigning a label to each pixel in the new view as
   - belonging to the object and visible from the original viewpoint, or
   - 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 1(e) for an example where the four labels are colored white, red, gray, 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 1(f)). Inpainting then proceeds on an object by object basis, with each object inpainted separately.

2.1. 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 labeling 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 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."

2.2. 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 [10], 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

(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 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* [10]*, 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* [46, 45, 34]*) with the exception of* [10] *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)*.*

$\mathcal{W}$, 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 2(a), our pipeline also has access to the segmentation in Figure 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 and 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, [46, 45], is to divide the inpainting domain into horizontal scanlines. Each scanline is then filled with a constant depth value that may be that of the endpoint with the greater depth [45], 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 $\mathcal{W}$ [46]. In [34], 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 usable for inpainting, while discarding the other endpoint. These approaches will work

for most of the hole in Figure 2(a), but all of them will incorrectly cut off the vertical plate leg as in Figure 2(c). Another approach, used in, for example, [26], 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 that in [10], 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 2(b), produces the result in Figure 2(d).

3. Proposed approach. Guidefill is a member of an extremely simple class of inpainting algorithms which also contains coherence transport [7, 27] and Telea's algorithm [42]. These methods fill the inpainting domain in successive shells from the boundary inward, with the color of a given pixel due to be filled computed as a weighted average of its already filled neighbors. The averaging weights w_ϵ are nonnegative and are allowed to depend on $\mathbf{x}$ but must scale proportionally with the size of the neighborhood $A_{\epsilon,h}(\mathbf{x})$. That is,

$$w_\epsilon(\mathbf{x},\mathbf{y}) = \hat{w}\left(\mathbf{x}, \frac{\mathbf{y}-\mathbf{x}}{\epsilon}\right) \tag{3.1}$$

for some function $\hat{w}(\cdot,\cdot) : \Omega_h \times B_1(\mathbf{0}) \to [0,\infty]$. See (3.2) for the weights used by coherence transport and Guidefill, which are of the form (3.1). Note that we will sometimes write w_r or w_1 in place of w_ϵ; in this case we mean (3.1) with ϵ replaced by r or 1 on the left-hand side. As the algorithm proceeds, the inpainting domain shrinks, generating a sequence of inpainting domains $D_h = D_h^{(0)} \supset D_h^{(1)} \supset \cdots \supset D_h^{(K)} = \emptyset$. At iteration k, only pixels belonging to the current active boundary $\partial_{\text{active}} D_h^{(k)}$ are filled; however, $\partial_{\text{active}} D_h^{(k)}$ need not be filled in its entirety—certain pixels may be made to wait until certain conditions are satisfied before they are "ready" to be filled (see section 3.4 for discussion and (3.8) for a definition of "ready"). Algorithm 1 illustrates this with pseudocode. While basic, these methods have the advantage of being cheap and highly parallelizable. When implemented on the GPU, the entire active boundary of the inpainting domain can be filled in parallel. If done carefully, this yields a very fast algorithm suitable for very large images; see section 3.5.

Guidefill is inspired in part by coherence transport [7, 27]. Coherence transport operates by adapting its weights to extrapolate isophotes in the undamaged portion of the image when they are detected, and applying a smooth blur when they are not. While relatively fast and achieving good results in many cases, it has a number of drawbacks:

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 5 and section 3.2.1).
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 6, 7, and 9 and sections 3.3 and 4).
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 8. This is partially addressed in [27], where several methods are

Algorithm 1 Shell Based Geometric Inpainting

u_h = image/video frame.
$D_h^{(0)}$ = initial inpainting domain.
$\partial_{\text{active}} D_h^{(0)}$= initial active inpainting domain boundary.
B_h = bystander pixels.
for $k = 0, \ldots$ **do**
  if $D_h^{(k)} = \emptyset$ **then**
    break
  end if
  for $\mathbf{x} \in \partial_{\text{active}} D_h^{(k)}$ **do**
    compute $A_{\epsilon,h}(\mathbf{x})$ = neighborhood of $\mathbf{x}$.
    compute nonnegative weights $w_\epsilon(\mathbf{x}, \mathbf{y}) \geq 0$ for $A_{\epsilon,h}(\mathbf{x})$.
    if ready($\mathbf{x}$) **then**

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

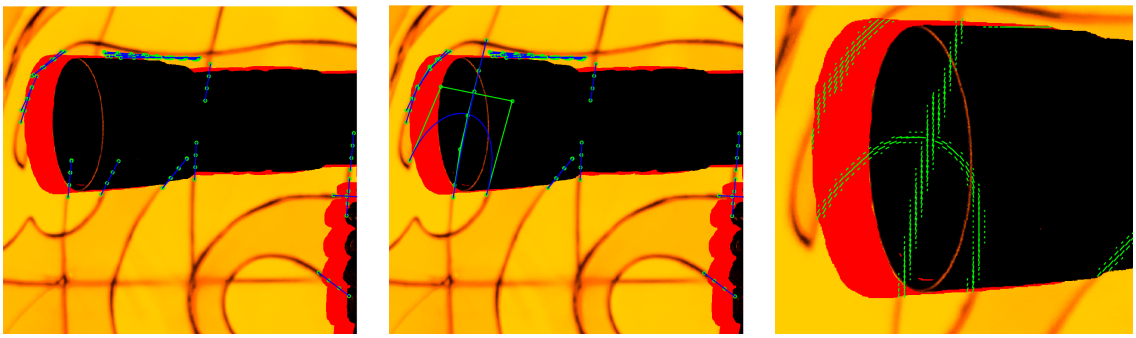
    end if
  end for
  $F = \{\mathbf{x} \in \partial_{\text{active}} D_h^{(k)} : \text{ready}(\mathbf{x})\}$.
  $D_h^{(k+1)} = D_h^{(k)} \setminus F$.
  $\partial_{\text{active}} D_h^{(k+1)} = \{\mathbf{x} \in \partial D_h^{(k+1)} : \mathcal{N}(\mathbf{x}) \cap (\Omega_h \setminus (D_h^{(k+1)} \cup B_h)) \neq \emptyset\}$.
end for

See (3.8) 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 4. They also both use the same weights (3.2). Note that in coherence transport, $\partial_{\text{active}} D_h^{(k)} = \partial D_h^{(k)}$ as there are no bystander pixels.

proposed for precomputing improved pixel orderings based on non-Euclidean distance functions. However, these methods all either require manual intervention or else have other disadvantages; see section 3.4.

Guidefill is aimed at overcoming these difficulties while providing an efficient GPU implementation (the implementation of coherence transport in [7, 27] 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.

3.1. 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 $\Omega_h \setminus (D_h \cup B_h)$ should be continued into D_h. These curves provide an intuitive mechanism by which the user can influence the results of inpainting (see Figure 3).



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

Figure 3. *Generating the guide field* $\mathbf{g}$ (c) *based on splines automatically generated by Guidefill* (a) *and edited by the user* (b).

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 8. 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 3.4.

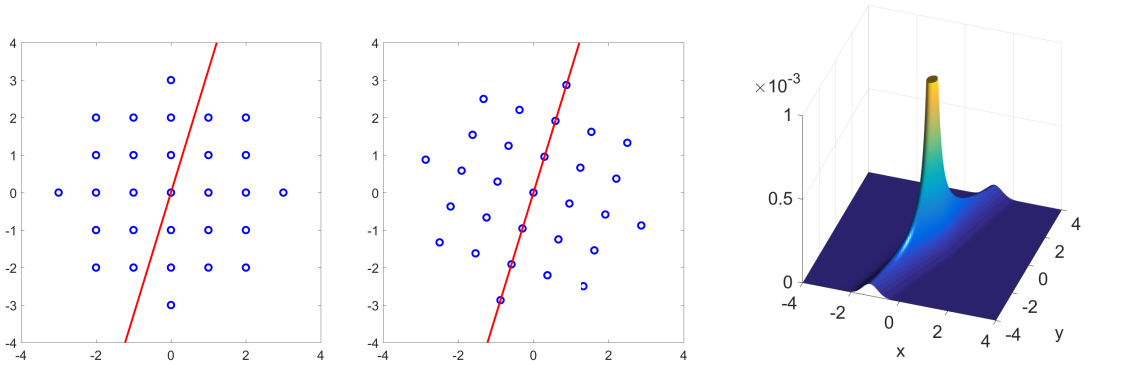
The method begins with the user either drawing the desired edges directly onto the image as Bézier splines using a GUI (Graphic User Interface), or else by having a set of splines automatically generated for them based on the output of a suitable edge detection algorithm run on $\Omega_h \backslash (D_h \cup B_h)$. 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 3.

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 3.2.

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

$$w_\epsilon(\mathbf{x}, \mathbf{y}) = \frac{1}{\|\mathbf{y} - \mathbf{x}\|} \exp\left(-\frac{\mu^2}{2\epsilon^2} (\mathbf{g}^\perp(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}))^2 \right), \tag{3.2}$$

(introduced in coherence transport [7]), 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



(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 (3.2) for $\mu = 10$.

Figure 4. *Illustration of the neighborhoods $A_{\epsilon,h}(\mathbf{x})$ and weights (3.2) used by coherence transport and Guidefill. In each case $\epsilon = 3px$ 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 (3.2) 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.*

guide direction $\mathbf{g}(\mathbf{x})$; see Figure 4 for an illustration. The color $u_h(\mathbf{x})$ is then computed using the formula in Algorithm 1, taking $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ and using weights (3.2). Coherence transport "fills" a pixel using exactly the same formula, except that now $A_{\epsilon,h}(\mathbf{x}) = B_{\epsilon,h}(\mathbf{x})$.

Unlike in coherence transport, however, our neighborhood $A_{\epsilon,h}(\mathbf{x}) = \tilde{B}_{\epsilon,h}(\mathbf{x})$ 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

$$u_h(\mathbf{x}) = \sum_{y \in \Omega_h} \Lambda_{\mathbf{y},h}(\mathbf{x}) u_h(\mathbf{y}) \quad \text{for all } \mathbf{x} \in \Omega \backslash (D^{(k)} \cup B), \tag{3.3}$$

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" phenomenon 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 sections 3.3 and 4. 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" phenomenon in the computation of $\mathbf{g}(\mathbf{x})$ itself.

Remark 3.1. *Note that although the weights (3.2) 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.*

3.2. Automatic spline detection and creation of the guide field. The goal of 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 [11] 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 labeled 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* [43]

$$\mathcal{J}_{\sigma,\rho} := g_\rho * (\nabla u_\sigma \otimes \nabla u_\sigma), \qquad \text{where } u_\sigma := g_\sigma * u_h \tag{3.4}$$

(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 that 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 [7, 27] (later adopted by Cao et al. [12]), 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 $\mathbf{g}(\mathbf{x}) = \mathbf{0}$.

3.2.1. 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"

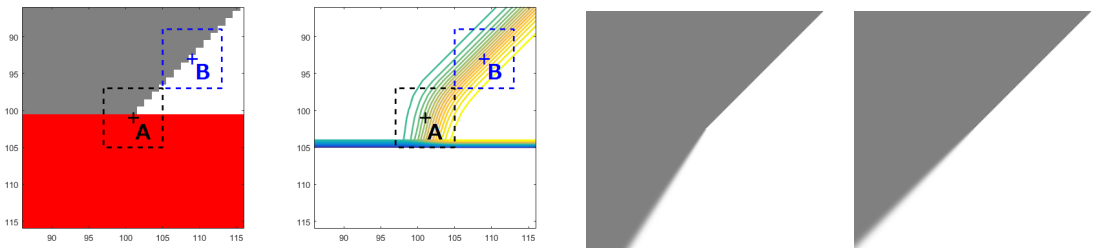
$\mathbf{g}(\mathbf{x})$ representing the orientation of isophotes in $\Omega_h \backslash (D_h \cup B_h)$ near $\mathbf{x}$. Inspired by the success of the *structure tensor* (3.4) as a robust descriptor of the local orientation of *complete* images, but also noting that $\mathcal{J}_{\sigma,\rho}(\mathbf{x})$ is undefined when $\mathbf{x} \in \partial D_h$, the authors proposed the *modified structure tensor*

(3.5)
$$\hat{\mathcal{J}}_{\sigma,\rho}(\mathbf{x}) := \frac{\left(g_\rho * \left(1_{\Omega_h \backslash (D_h^{(k)} \cup B_h)} \nabla v_\sigma \otimes \nabla v_\sigma\right)\right)(\mathbf{x})}{\left(g_\rho * 1_{\Omega_h \backslash (D_h^{(k)} \cup B_h)}\right)(\mathbf{x})}, \text{ where } v_\sigma := \frac{g_\sigma * \left(1_{\Omega_h \backslash (D_h^{(k)} \cup B_h)} u_h\right)}{g_\sigma * 1_{\Omega_h \backslash (D_h^{(k)} \cup B_h)}},$$

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 (3.5). This notation was introduced in [7] because u_σ is already defined in (3.4).) 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 gray above it ($B_h = \emptyset$ in this case). We take $\sigma = 2$, $\rho = 4$. This is presented in Figure 5(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) overlaps 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 gray and white. However, as the weights are radially symmetric and the "angular wedge" of the partial box centered at A contains far more gray pixels than it does white, at point A we end up with a color much closer to gray. This results in a curvature of the level curves of v_σ that can be seen in Figure 5(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°. Figures 5(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 3.2 we backed our splines up from the inpainting domain and computed their orientation using the structure tensor rather than the modified structure tensor.

Remark 3.2. *In some ways our spline-based approach resembles the earlier work by Masnou and Morel* [29] *and later Cao et al.* [12] *in which level lines are interpolated across the inpainting domain by joining pairs of "compatible T-junctions" (level lines with the same gray value intersecting the boundary with opposite orientations). This was done first as straight lines* [29]*, and later as Euler spirals* [12]*. An* $O(N^3)$ *algorithm was proposed in* [12] *for joining compatible T-junctions, where* N *is the number of such junctions. This could be beneficial in situations such as Figure* 3(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 usable pixels,



(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 (3.5) bend near point A.

(c) Inpainting using Guidefill with $\mathbf{g}$ calculated at point A using the modified structure tensor (3.5).

(d) Inpainting using Guidefill with $\mathbf{g}$ calculated at point B using the ordinary structure tensor (3.4).

Figure 5. *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* $\hat{\mathcal{J}}_{\sigma,\rho}$ (3.5) *(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* (3.5) *(as noted in the text, the notation* v_σ *was introduced in* [7] *because the ordinary structure tensor* (3.4) *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* (3.4) *at* B*.* (c)–(d) *show the results of inpainting using Guidefill with guidance directions* $\mathbf{g}_A$ *and* $\mathbf{g}_B$*, respectively.*

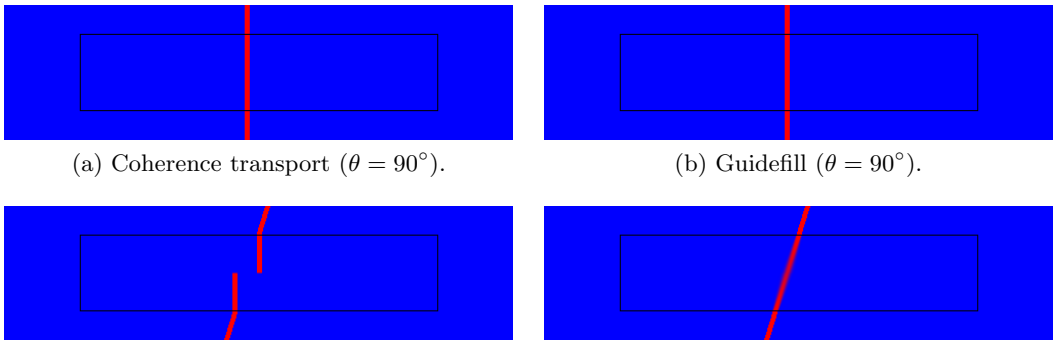
we now typically have D_h *with usable pixels on one side, and bystander pixels on the other (for example, pixels belonging to some foreground object as in Figure* 1(f)*). In some cases we might get around this by searching the perimeter of* $D_h \cup B_h$*, as opposed to just the perimeter of* D_h*, 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 does not make sense to talk about joining compatible T-junctions.*

Nevertheless, following the definition of "compatibility" given in [12]*, one way of incorporating this idea would be to declare two splines* S_1 *and* S_2 *based at* $\mathbf{x}_{base}^{(1)}$ *and* $\mathbf{x}_{base}^{(2)}$ *"compatible" if*

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

where u_σ *is given by* (3.4) *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 the scope of the present work.*

3.3. 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



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

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

Figure 6. *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 phenomenon is provided in Theorem 4.1 and illustrated in Figure 9.*

how Guidefill overcomes this as well. More details and a theoretical explanation are provided by our continuum analysis in section 4.

Figure 6 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 excluded as it has not been inpainted yet); see Figure 4(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 $n^2 + m^2 \leq 9$.

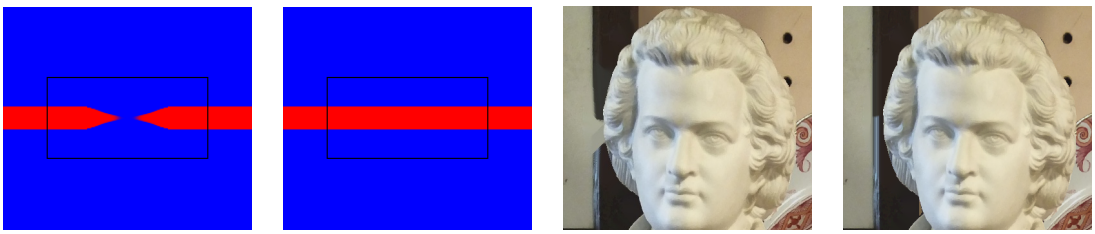
We resolve this problem by replacing $B_{\epsilon,h}(\mathbf{x})$ with the rotated ball of ghost pixels $\tilde{B}_{\epsilon,h}(\mathbf{x})$, which is constructed in order to contain at least one "pixel" on L besides $\mathbf{x}$, as illustrated in Figure 4(b).

In Figure 7 we also illustrate the importance of ghost pixels on the nonsynthetic example with a smoothly varying guide field shown in Figure 3. 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.

3.4. Automatic determination of a good pixel order (smart order). Figures 8(a) and 8(c) show 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 nonsynthetic

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

Figure 7. *The effect of ghost pixels on a nonsynthetic example ($\epsilon = 3px$, $\mu = 50$). When ghost pixels are disabled, the extrapolated isophotes are unable to smoothly curve as only finitely many transport directions are possible.*



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

Figure 8. *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 (3.8) avoids this by using information from the precomputed guide field to automatically decide when pixels should be filled.*

example. In these cases extrapolated lines are cut off due to certain pixels being filled too early. Figures 8(b) and 8(d) show the same examples using our improved fill order defined by the ready function (3.8).

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. [41] proposed having the user draw critical curves over top of the image, and then filling patches centered on those curves first. März [27] suggested calculating nonstandard distance from the boundary functions, and then filling pixels in an order based on those functions. Finally, Criminisi, Pérez, and Toyama [16] compute 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

$$C(\mathbf{p}) = \frac{\sum_{\mathbf{q}\in\Psi_{\mathbf{p}}\cap(\mathcal{I}\setminus\Omega)} C(\mathbf{q})}{|\Psi_{\mathbf{p}}|} \quad \text{and} \quad D(\mathbf{p}) = \frac{|\nabla I_{\mathbf{p}}^{\perp} \cdot \mathbf{n}_{\mathbf{p}}|}{\alpha}, \tag{3.6}$$

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 $\mathbf{n}_{\mathbf{p}}$ 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 changes, and certain patch priorities must be recomputed).

Our approach. The approach of März [27] 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, Pérez, and Toyama [16]. For each $\mathbf{x} \in \partial_{\text{active}} D_h$, we compute the ratio

$$C(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x}) \cap (\Omega \setminus (D^{(k)} \cup B))} w_\epsilon(\mathbf{x}, \mathbf{y})}{\sum_{\mathbf{y} \in \tilde{B}_{\epsilon,h}(\mathbf{x})} w_\epsilon(\mathbf{x}, \mathbf{y})}, \tag{3.7}$$

where $w_\epsilon(\mathbf{x}, \mathbf{y})$ is the weight function (3.2) 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 (3.6). 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 (3.6), which tries to ensure that the angle between $\nabla^{\perp} I_{\mathbf{p}}$ and $\mathbf{n}_{\mathbf{p}}$ is not too large. However, unlike Criminisi, Pérez, and Toyama [16], 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

$$\text{ready}(\mathbf{x}) = 1(C(\mathbf{x}) > c), \tag{3.8}$$

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

Possible extensions. Unlike [16], which assigns high priority to pixels with a large gradient, (3.8) 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. [41] 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 (3.8) as

$$\text{ready}(\mathbf{x}) = 1(D(\mathbf{x}) > c_2) 1(C(\mathbf{x}) > c_1), \tag{3.9}$$

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 (3.8).

3.5. GPU implementation. Here we sketch two GPU implementations of Guidefill, differing in how they assign GPU threads to pixels in Ω_h. In section 5 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.[2] Hence, it can be advantageous to ensure that GPU threads are only assigned to the subset of pixels currently being 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 are in the accompanying supplementary material (M110373_01.pdf [local/web 6.44MB]).

4. Continuum limit. Here we present a special case of the analysis in [22], which aims to provide a rigorous justification of the discussion in section 3.3. This is accomplished by considering the continuum limit of u_h as $h \to 0$ with $r := \epsilon/h \in \mathbb{N}$, the radius of the neighborhood $A_{\epsilon,h}(\mathbf{x})$ measured in pixels, fixed. Note that this is different from the limit considered in [7], where first $h \to 0$ and then $\epsilon \to 0$; see Remark 4.3. Our objective is to assume enough complexity to explain the phenomenon 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

$$\nabla u \cdot \mathbf{g}^* = 0, \tag{4.1}$$

where $\mathbf{g}^* \neq \mathbf{g}$ 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 (4.2), 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 a known issue of progressive blurring when bilinear interpolation is iterated [37, sect. 5].

Assumptions. We assume a constant guide direction

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

as this is all that is required to capture the phenomenon 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

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

[2] 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 [33, 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 $p \in [0, \infty]$:

$$\|f_h\|_p := \Big(\sum_{\mathbf{x} \in D_h} |f_h(\mathbf{x})|^p h^2 \Big)^{\frac{1}{p}}, \qquad \|f_h\|_\infty := \max_{\mathbf{x} \in D_h} |f_h(\mathbf{x})|. \tag{4.2}$$

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 [22] we 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 nonzero 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 (4.1) 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 $\mathbf{g}^*$ will also be constant (we will prove this), so this is simple. As long as $\mathbf{g}^* \cdot e_2 \neq 0$, we simply define the solution to the transport problem (4.1) with boundary conditions $u(x, 0) = u_0(x, 0)$, $u(0, y) = u(1, y)$ to be

$$u(x, y) = u_0(x - \cot(\theta^*) y \bmod 1, 0), \tag{4.3}$$

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

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

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

Theorem 4.1. *Let the image domain Ω, inpainting domain D, and 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 and 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 neighborhood

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

(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 (3.2) 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 consider $h \to 0$. Define the transport direction $\mathbf{g}^*$ by

$$\mathbf{g}^* = \frac{\sum_{\mathbf{y} \in A_r^-} w_r(\mathbf{0}, \mathbf{y}) \mathbf{y}}{\sum_{\mathbf{y} \in A_r^-} w_r(\mathbf{0}, \mathbf{y})}, \qquad A_r^- := \left\{ (y_1, y_2) \in \frac{1}{h} A_{\epsilon,h}(0) : y_2 \leq -1 \right\}. \tag{4.4}$$

Note that A_r^- depends only on r. Also note that we have written w_r to mean the weights (3.2) with ϵ replaced by r. Let $u : (0,1]^2 \to \mathbb{R}^d$ denote the weak solution (4.3) to the transport PDE (4.1) 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[3] of u, u_h we have the bound

$$\|u - u_h\|_p \le Kh^{\frac{1}{2p}} \tag{4.5}$$

if $\{\mathcal{C}_i\}$ is nonempty and

$$\|u - u_h\|_p \le Kh \tag{4.6}$$

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 4.1. *The transport direction $\mathbf{g}^*$ predicted by Theorem 4.1 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 (3.2). 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*

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

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(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

$$\mathbf{g}^*_{c.t.} = \frac{\sum_{\mathbf{y} \in b_r^-} w_r(\mathbf{0},\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y} \in b_r^-} w_r(\mathbf{0},\mathbf{y})} \quad \text{and} \quad \mathbf{g}^*_{g.f.} = \frac{\sum_{\mathbf{y} \in \tilde{b}_r^-} w_r(\mathbf{0},\mathbf{y})\mathbf{y}}{\sum_{\mathbf{y} \in \tilde{b}_r^-} w_r(\mathbf{0},\mathbf{y})}. \tag{4.7}$$

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 9.

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, $A_{\epsilon,h}(\mathbf{x}) \subset \mathbb{Z}_h^2$. For the case where $A_{\epsilon,h}(\mathbf{x})$ contains ghost pixels lying between pixel centers and for u_0 with lower regularity, we refer the reader to [22]. 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 the formula in Algorithm 1 as

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

[3] Remember that 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 (4.5), (4.6) to emphasize that it holds channelwise for each $i = 1, \ldots, d$, but we felt that this would lead to too much clutter.

Next we note that A_r^- is nonempty, since we assume $A_{\epsilon,h}(\mathbf{x}) \in \{B_{\epsilon,h}(\mathbf{x}), \tilde{B}_{\epsilon,h}(\mathbf{x})\}$ and $r \geq 2$ (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 $\mathbf{g}^*$ (4.4) is defined, and moreover $\mathbf{g}^* \cdot e_2 \neq 0$. 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

$$|e_h(\mathbf{x})| \leq Kh \tag{4.8}$$

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. $\|Hu\|_2 \leq D$ uniformly on $(0,1]^2$, where Hu denotes the Hessian of u and $\|\cdot\|_2$ is the 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

$$|e_h(ih, jh)| \leq jDr^2h^2, \tag{4.9}$$

from which (4.8) follows with $K = Dr^2$ since $j \leq N = 1/h$.

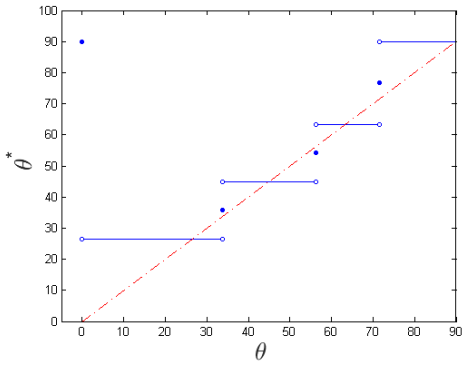
We proceed by induction, supposing that (4.9) holds for all $(i'h, j'h)$ with $1 \leq i' \leq N$ and $j' < j$ (the base case $j = 0$ is obvious). Applying our inductive hypothesis and expanding u to second order, we obtain

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

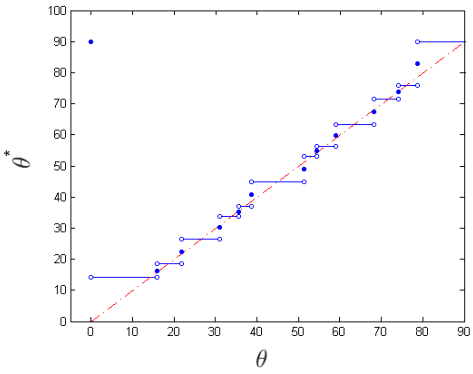
where we have used the fact that when $\mathbf{x} = (ih, jh)$ and $\mathbf{y} \in A_r^-$, then $(\mathbf{x}+\mathbf{y}h)$ is of necessary form $(i'h, j'h)$ with $1 \leq i' \leq N$ and $j' < j$ needed for our inductive hypothesis to hold. ∎

4.1. Consequences. Theorem 4.1 helps us to understand two important features of the class of algorithms under study. First, it helps us to understand a kinking phenomenon Guidefill aims to overcome. Second, it will help us to understand a new phenomenon Guidefill introduces (and, indeed, a limitation of the method)—the gradual degradation of the signal due to repeated bilinear interpolation.

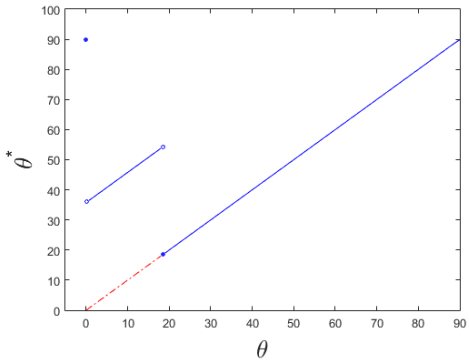
4.1.1. Kinking. Figure 9 illustrates the significance of Theorem 4.1 by plotting the phase $\theta(\mathbf{g}^*_{\text{c.t.}})$ and $\theta(\mathbf{g}^*_{\text{g.f.}})$ of the theoretical limiting transport directions of coherence transport and Guidefill, respectively, (4.7) as a function of the phase $\theta(\mathbf{g})$ of the guidance direction $\mathbf{g}$. The cases $\epsilon = 3h$ and $\epsilon = 5h$ are considered (coherence transport [7] recommends $\epsilon = 5h$ by default) with $\mu \to \infty$. For coherence transport we have $\theta(\mathbf{g}^*) \neq \theta(\mathbf{g})$ except for finitely many angles, explaining the kinking observed in practice. On the other hand, for Guidefill we have $\theta(\mathbf{g}^*) = \theta(\mathbf{g})$ (in other words, no kinking) for all angles greater than a minimum value. We refer the reader to [22] for additional details.



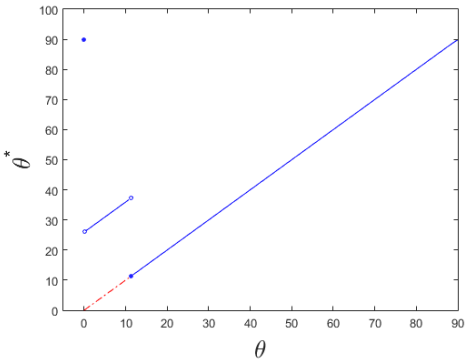
(a) Coherence transport, $r = 3$.



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



(c) Guidefill, $r = 3$.



(d) Guidefill, $r = 5$.

Figure 9. *The theoretical limiting curves $\theta^* = \theta(\mathbf{g}^*_{c.t.})$ (coherence transport* (a)–(b)*) and $\theta^* = \theta(\mathbf{g}^*_{g.f.})$ (Guidefill* (c)–(d)*) as a function of $\theta = \theta(\mathbf{g})$, with $\mathbf{g}^*_{c.t.}$ and $\mathbf{g}^*_{g.f.}$ given by* (4.7)*, and where $\mathbf{g}$ is the desired guidance direction fed into the weights* (3.2)*. We set $r := \epsilon/h = 3, 5$ and consider $\mu \to \infty$. The ideal curve $\theta^* = \theta$ is highlighted in red. The limiting guide directions $\mathbf{g}^*_{c.t.}$ and $\mathbf{g}^*_{g.f.}$ are related by* (4.4) *to the weights* (3.2) *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 $\theta^* \neq \theta$ 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* 4.2 *as well as* [22] *for more details.*

Remark 4.2. *In order to understand the kinking of Guidefill shown in Figure* 9(c)–(d) *at small angles for $\mathbf{g} \neq \mathbf{0}$ and $\mu \gg 1$, it is helpful to consider the decomposition*

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

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 4.1 *one may readily show that $\mathbf{g}^*_{g.f.}$ given by* (4.7) *obeys $\mathbf{g}^*_{g.f.} = \mathbf{g}$. The kinking observed for small angles in Figure* 9(c)–(d) *occurs when $\ell_{\epsilon,h}(\mathbf{x})$ contains no readable pixels, that is,*

$$\ell_{\epsilon,h}(\mathbf{x}) \cap (\Omega_h \backslash (D_h^{(k)} \cup B_h)) = \emptyset. \tag{4.10}$$

In practice, the smart order proposed in section 3.4 is likely to detect this situation. Since the weights (3.2) concentrate most of their mass in $\ell_{\epsilon,h}(\mathbf{x})$ when μ is large, in this case we can expect the confidence term (3.7) to be small and the smart order test (3.8) will tell the algorithm to delay the filling of $\mathbf{x}$. If at a later iteration (4.10) no longer holds, (3.8) should be satisfied and inpainting can resume with no kinking.

Remark 4.3. *The limiting transport direction $\mathbf{g}^*$ predicted by Theorem 4.1 is similar to the transport direction predicted by Bornemann and März in [7] (Theorem 1). The key difference is that while Bornemann and 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, which we argue in [22] is more relevant. The result is that whereas [7] obtains a formula for $\mathbf{g}^*$ as an integral over a (continuous) half-ball, our $\mathbf{g}^*$ 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 (note that we write w_r and w_1 to mean the weights (3.2) with ϵ replaced by r and 1, respectively):*

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

where

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

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

4.1.2. Signal degradation. Theorem 4.1 says that when u_0 has jump discontinuities, we can expect convergence in L^p for all $1 \le p < \infty$, but potentially with a gradually deteriorating rate and with no guarantee of convergence when $p = \infty$. This suggests that our method may have a tendency to gradually blur an initially sharp signal. Indeed, our method is based on bilinear interpolation, and a known property of the repeated application of bilinear interpolation is to do just that; see, for example, [37, sect. 5]. Moreover, this blurring is plainly visible in Figure 5(c)–(d). To explore this phenomenon, we considered the continuum problem of inpainting the line

$$\tan(73^\circ) - 0.1 \le y \le \tan(73^\circ) + 0.1$$

over the image domain $\Omega = [-1, 1] \times [-0.5, 0.5]$ and inpainting domain $D = [-0.8, 0.8] \times [-0.3, 0.3]$. We then used Guidefill to solve the discrete inpainting problem at four different image resolutions: 200×100px, 400×200px, 4000×2000px, and 8000×4000px. In each case, we examined a horizontal cross-section of the solution at three places: $y = 0.3$, the boundary of the inpainting domain where the signal is perfect, $y = 0.25$, a short distance inside the inpainting domain, and $y = 0$, the midpoint of the domain where we can expect maximal deterioration. The results are given in Figure 10.

There is indeed signal degradation, most significantly for low-resolution problems, and it does indeed get worse as we move further into the inpainting domain. This is a limitation of

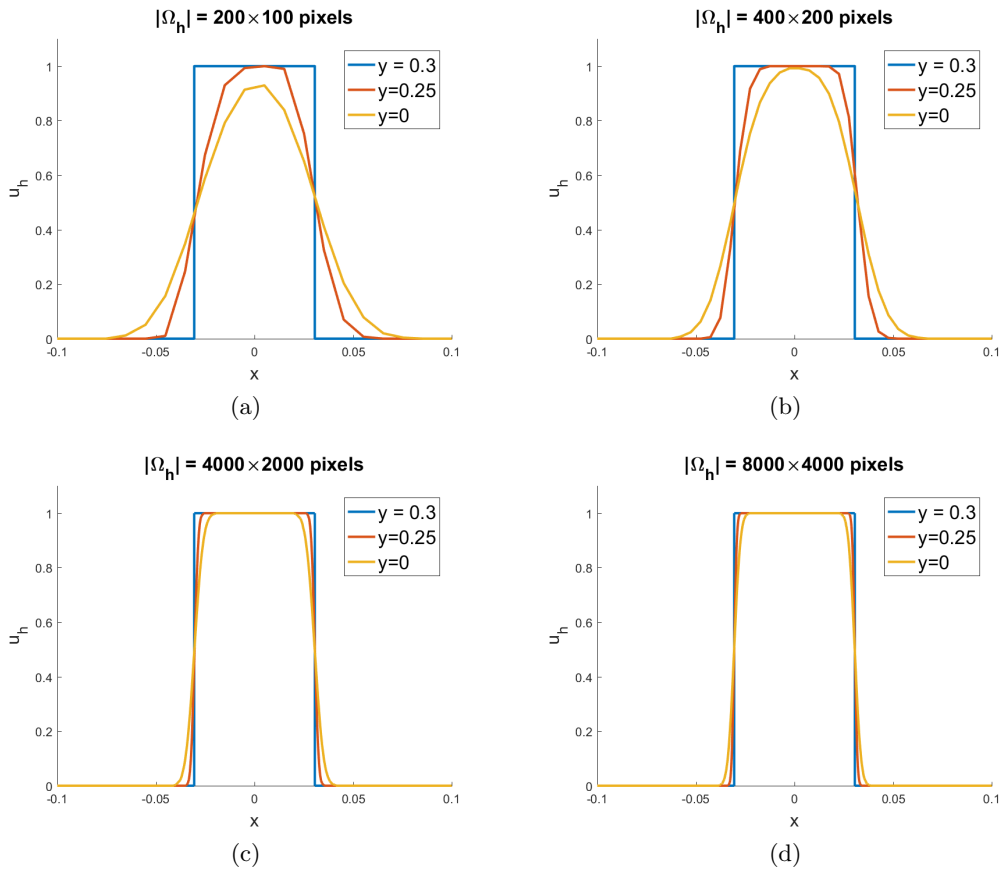


Figure 10. *Signal degradation and L^p convergence of Guidefill. The continuum problem of inpainting the line* $\tan(73°) - 0.1 \leq y \leq \tan(73°) + 0.1$ *with image domain* $\Omega = [-1, 1] \times [-0.5, 0.5]$ *and inpainting domain* $D = [-0.8, 0.8] \times [-0.3, 0.3]$ *is rendered at a variety 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. This deterioration is to be expected as our method is based on iterated bilinear interpolation, which is known to have this effect* [37, section 5]*. However, also note that, in accordance with Theorem* 4.1*, the signal is less degraded in higher resolution images, even though we have applied more bilinear interpolation operations.*

our method, especially for thin edges to be extrapolated across long distances. However, also note that, as predicted by Theorem 4.1, when we increase the image resolution, the degree of degradation drops significantly, even though we are applying many more bilinear interpolation operations.

4.2. Formation of shocks. A disadvantage of the shell based approach in Algorithm 1 is the potential to create a shock in the middle of D_h, where image values propagated from initially distant regions of ∂D_h meet; see Figure 11(a)–(b) for a simple example. One way of understanding this is based on the connection to 1st-order transport obtained in the continuum. This was anticipated by Ballester et al. [4], who noted that the boundary value

(a) An inpainting problem with incompatible boundary conditions. The inpainting domain D_h is in gray, 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 11. *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* [7, 28], *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.* [4] *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* [36] *to converge on a* $200\times200px$ *inpainting domain (Guidefill by contrast took only* $60ms$*).*

problem

$$\mathbf{c}(\mathbf{x})\cdot\nabla u = 0 \quad \text{in } D, \qquad u = \varphi \quad \text{on } \partial D \tag{4.11}$$

(obtained as a special case of their inpainting framework) 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 φ at those endpoints. To resolve this issue they suggested, among other things, adding a diffusive term $-\epsilon\Delta u$ to (4.11) to make it well posed, and then taking $\epsilon \to 0$. See Figure 11(c), where we solve the resulting nonsymmetric linear system with $\epsilon = 10^{-7}$ using GMRES (the Generalized Minimum RESidual method for nonsymmetric linear systems) [36]. In a series of papers [7, 27, 28] März took a different approach. First he showed that Algorithm 1 (with $B_h = \emptyset$ and under other assumptions) reduces to (4.11) with D replaced by $D\backslash\Sigma$ in a high-resolution and vanishing viscosity limit [7, Theorem 1]. The set Σ is a set of measure zero containing any potential shocks. Then he showed that (4.11) is well posed on $D\backslash\Sigma$. The set Σ is related to a distance map prescribing the order in which pixels are filled, so by choosing the distance map carefully one can in some cases eliminate shocks and in other cases at least have some control over where they appear [27].

In our case this issue is less significant as we only specify boundary data on $\partial_{\text{active}}D_h \subset \partial D_h$. Indeed, as long as the integral curves of $\mathbf{c}(\mathbf{x})$ do not cross and always have one endpoint on $\partial_{\text{active}}D_h$ and the other on $\partial D_h\backslash\partial_{\text{active}}D_h$, we avoid the issue altogether. However, there is nothing about our framework that explicitly prevents the formation of shocks, and indeed they do sometimes occur; see, for example, Figure 15(f).

5. Algorithmic complexity.

In this section we analyze the complexity of the two implementations of Guidefill sketched in section 3.5 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 $P(N, M)$

mean that the algorithm can be completed by $O(P(N,M))$ processors in $O(T(N,M))$ time per processor. See, for example, [35, Chap. 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, [33, Chap. 4] for a discussion of GPU architecture, and, for example, [25] for a more realistic theoretical model. We do not consider these additional complexities here.

In Theorem 5.1 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 5.2 we establish bounds on $K(N)$ under two simplifying assumptions. First, 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). Second, we assume that the smart ordering of section 3.4 is turned off. We also include a discussion in the accompanying supplementary material (M110373_01.pdf [local/web 6.44MB]) 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 5.1. *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

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

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

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

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

Guidefill with tracking allocates $O(|\partial D_h^{(k)}|)$ threads per iteration of Guidefill, each of which does $O(\log|\partial D_h^{(k)}|)$ work. This is because, as stated in section 3.5, 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 $O(\lceil\frac{|\partial D_h^{(k)}|}{\sqrt{N+M}}\rceil)$ 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 5.2. *If we make the same assumptions as in Theorem 5.1 and if we further suppose $(\partial_{outer}D_h)\cap B_h=\emptyset$ and that the smart order test from section 3.4 is turned off, then we additionally have*

$$K(N)=O(\sqrt{N}) \tag{5.1}$$

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 (3.8) 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 been 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 $K(N)=O(\sqrt{N})$. ∎

6. 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 5. 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.[4]

6.1. 3D conversion examples. We tested our method on a number of HD problems, including the four photographs shown in Figure 12 and the video illustrated in Figure 13. The photographs were converted into 3D by building rough 3D geometry and creating masks for each object, as outlined in section 2. For the movie, we used a computer generated model with existing 3D geometry and masks,[5] as generating these ourselves on a frame by frame basis would have been far too expensive (indeed, in the 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 13(k). Please see also the accompanying supplementary material, where our results can be viewed in video form (M110373_02.mp4 [local/web 973KB], M110373_03.mp4 [local/web 857KB], M110373_04.mp4 [local/web 833KB], M110373_05.mp4 [local/web 826KB], M110373_06.mp4 [local/web 817KB]) and in anaglyph 3D (anaglyph glasses required). Timings for Guidefill are given both with and without the boundary tracking as described in section 3.5.

As has been noted in the related work, the literature abounds with depth-guided variants of Criminisi's method [16] designed for the disocclusion step arising in 3D conversion using the depth-map based pipeline discussed in section 2.2 (see, for example, [45, 46, 17, 26, 30, 10]), 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 2.2, Figure 2, we have shown a simple example where with the exception of [10] these heuristics are likely to 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, Pérez, and Toyama from the point of view of running time; for example, [10] incorporates the running time improvements originally published in their earlier work [9]. We also could have based our "bystander-aware" Criminisi on the improvement in [9]; however, instead we note that the running time published in [10] is about 1500px/s, which is still much slower than Guidefill, especially for high-resolution problems (see Table 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 [7, 27], the variational exemplar-based methods nl-means and nl-Poisson from Arias et al. [3], and Photoshop's Content-Aware fill. For the movie, we compare with the exemplar-based video inpainting

[4]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.

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

(a) Wine.

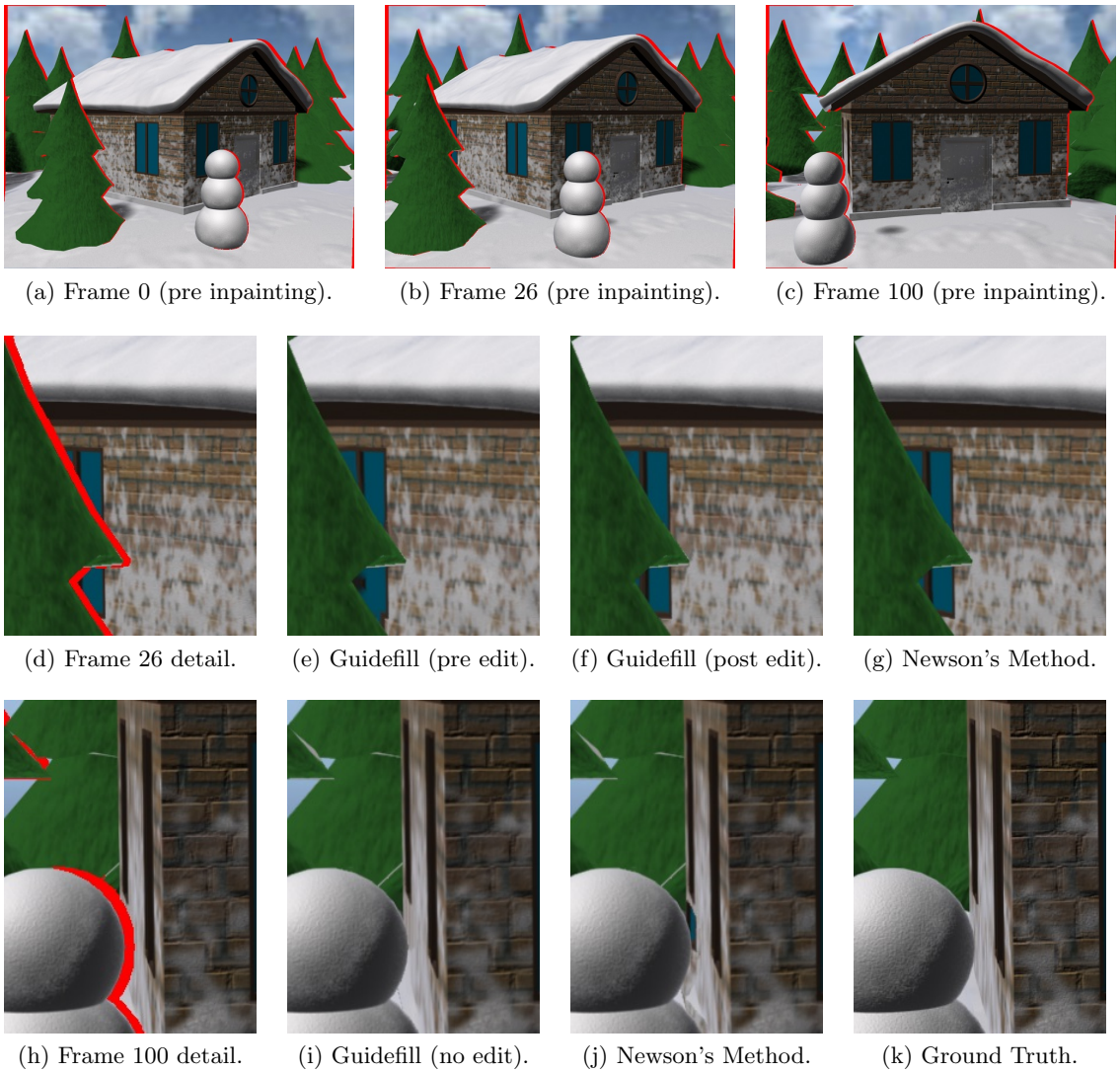
(b) Bust.

(c) Pumpkin.

(d) Planet.

Figure 12. *Example photographs used for* 3*D conversion of different sizes* (a) 528×960*px,* (b) 1500×1125*px,* (c) 4000 × 4000*px, and* (d) 5000 × 5000*px.*

method of Newson et al. [32, 31]. 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 in Figure 2(c), we first ran each method using $D_h \cup B_h$ as the in-



(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 13. *Comparison of Guidefill (*19*s with tracking,* 31*s without) and Newson's method (*5*hr*37*min) for inpainting the "cracks" (shown in red) arising in the* 3D *conversion of an HD video (*$1280px \times 960px \times 101fr$*). 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. The video is provided in the accompanying supplementary material* (M110373_02.mp4 [local/web 973KB], M110373_03.mp4 [local/web 857KB], M110373_04.mp4 [local/web 833KB], M110373_05.mp4 [local/web 826KB], M110373_06.mp4 [local/web 817KB]).

painting 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

Table 1

Timings of different inpainting algorithms used in the conversion of the four examples in Figure 12. 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**

are available for download online.[6] Figure 13 shows a few frames of a 1280px × 960px × 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 13(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 13(e). In this case the failure is because the one pixel wide ring described in section 3.2 fails to intersect certain edges we would like to extend. However, Guidefill requires only 19s (if boundary tracking is employed, and 31s if it is not) to inpaint the entire video, and these artifacts can be corrected as in Figure 13(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 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 14, 15, and 17. We do not show the output of every method and have included only the 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 16 shows the detected splines for the background object and illustrates the editing process. Results are shown in Figure 14 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 16. Guidefill is also the fastest method, although in this case the gains are not as large as those for bigger images.

[6]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) 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 14. *Comparison of different inpainting methods for the "Wine" example. Two challenging areas are shown.*

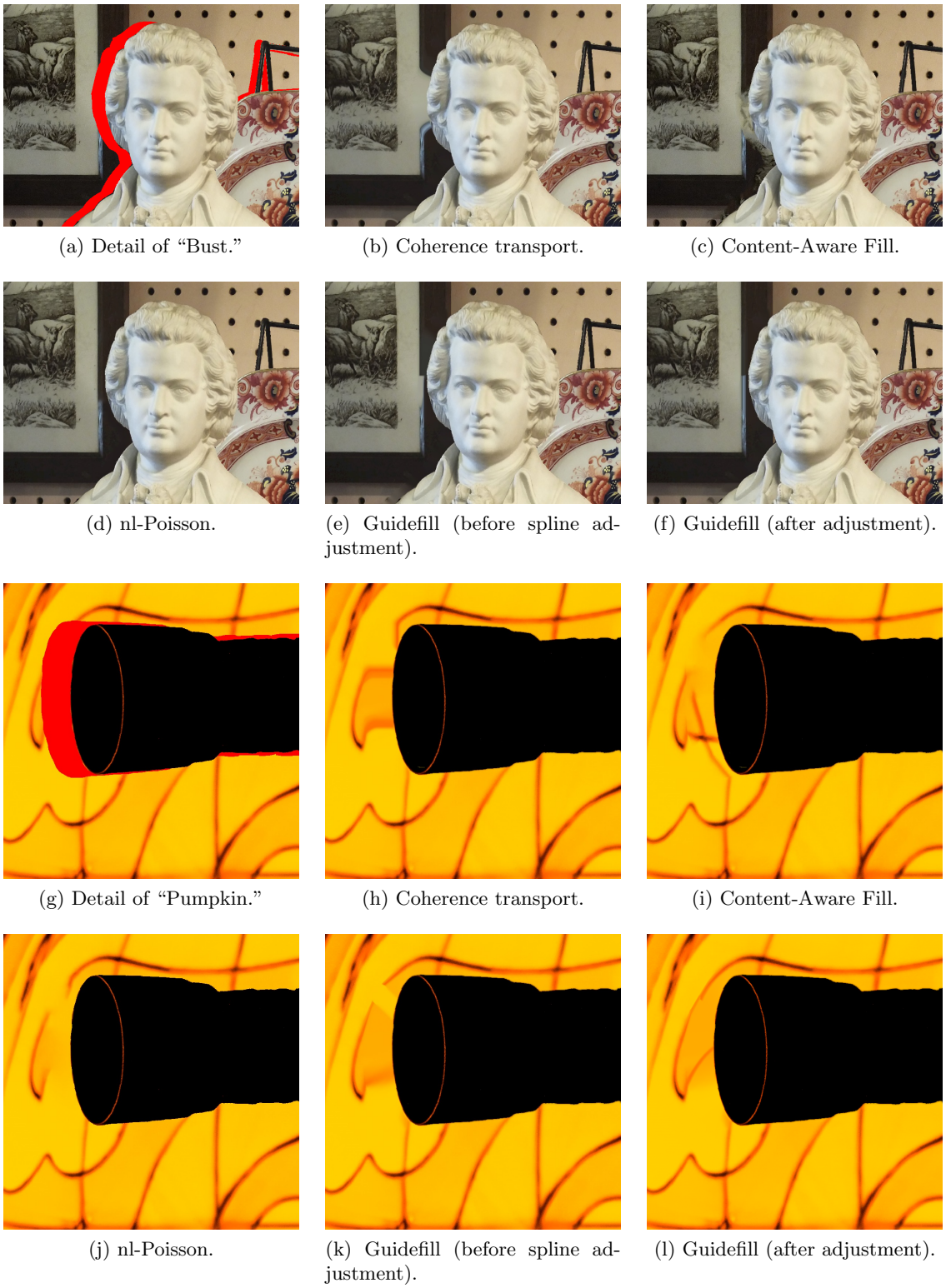
The second example, "Bust," is a 1500 × 1125px image. Timings are reported only for inpainting the background object, which has an inpainting domain containing 111277px, and results are shown in Figure 15(a)–(f). In this case we chose to edit the automatically detected splines, in particular rotating 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 15(f). As we noted in section 4.2, 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 15(g)–(l). We ran nl-Poisson on only the detail shown in Figure 15(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 3(a)–(b). In doing so we were 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 speedup 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 the accompanying supplementary material for the full 3D effect; here we show only a detail in Figure 17. 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.

6.2. Validation of complexity analysis. As stated in section 5, 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 the accompanying supplementary material (M110373_01.pdf [local/web 6.44MB]) 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 $0.45 \leq y \leq 0.55$ 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



(a) Detail of "Bust." (b) Coherence transport. (c) Content-Aware Fill.

(d) nl-Poisson. (e) Guidefill (before spline adjustment). (f) Guidefill (after adjustment).

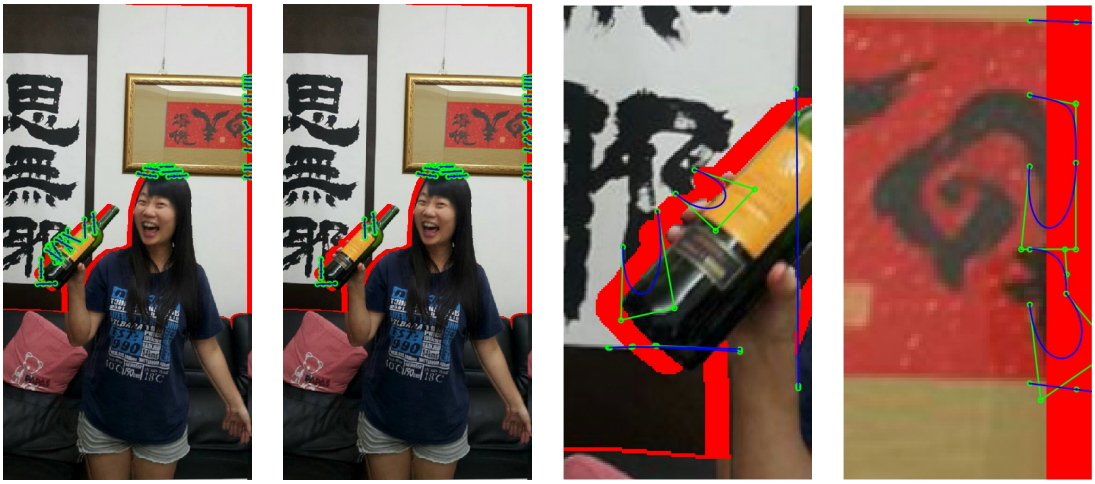
(g) Detail of "Pumpkin." (h) Coherence transport. (i) Content-Aware Fill.

(j) nl-Poisson. (k) Guidefill (before spline adjustment). (l) Guidefill (after adjustment).

Figure 15. *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* 4.2.

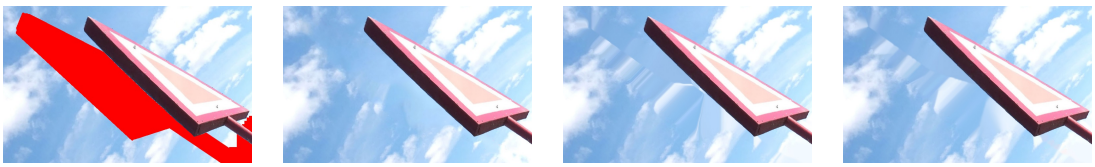
requested threads $P(N)$, with and without tracking. Results are shown in Figure 18; note the loglog scale. In Figure 18(b) we have also indicated the value of p for comparison; note that for Guidefill without tracking we have $P(N) \gg p$ for all but the smallest problems, but for Guidefill with tracking we have $P(N) < p$ up until $N \approx 2 \times 10^5$.

Based on Theorems 5.1 and 5.2 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 5 does a reasonable job of predicting the rough behavior of our method in practice.



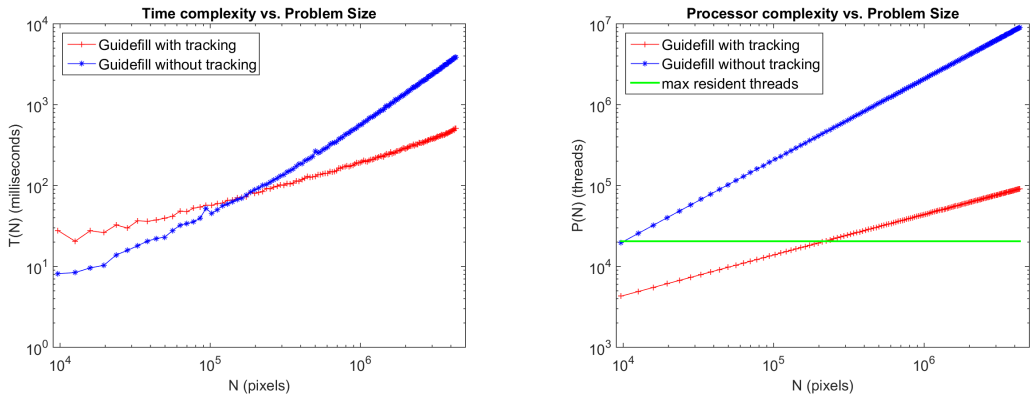
(a) (b) (c) (d)

Figure 16. *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 17. *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 18. *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 3.5 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).*

7. 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 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 [42] and coherence transport [7, 27]. 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 [7, 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 [7]. It is also able to explain how our ghost pixels are able to fix this problem, but also sheds 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 [22].

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 speedup 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, in order to keep execution times low, 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. However, these improvements are beyond the scope of the present work.

Acknowledgment. The authors are grateful to the insightful comments of three anonymous referees, which greatly improved the clarity of the text.

REFERENCES

[1] https://www.gener8.com/projects/.
[2] http://www.geforce.co.uk/whats-new/articles/nvidia-geforce-gtx-titan-x.
[3] P. Arias, G. Facciolo, V. Caselles, and G. Sapiro, *A variational framework for exemplar-based image inpainting*, Internat. J. Comput. Vision, 93 (2011), pp. 319–347, https://doi.org/10.1007/s11263-010-0418-7.
[4] C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera, *Filling-in by joint interpolation of vector fields and gray levels*, IEEE Trans. Image Process., 10 (2001), pp. 1200–1211, https://doi.org/10.1109/83.935036.
[5] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman, *PatchMatch: A randomized correspondence algorithm for structural image editing*, ACM Trans. Graphics (Proc. SIGGRAPH), 28 (2009), 24.

[6] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, *Image inpainting*, in Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press/Addison–Wesley, New York, 2000, pp. 417–424.
[7] F. Bornemann and T. März, *Fast image inpainting based on coherence transport*, J. Math. Imaging Vision, 28 (2007), pp. 259–278, https://doi.org/10.1007/s10851-007-0017-6.
[8] M. Burger, L. He, and C.-B. Schönlieb, *Cahn–Hilliard inpainting and a generalization for grayvalue images*, SIAM J. Imaging Sci., 2 (2009), pp. 1129–1167, https://doi.org/10.1137/080728548.
[9] P. Buyssens, M. Daisy, D. Tschumperlé, and O. Lézoray, *Exemplar-based inpainting: Technical review and new heuristics for better geometric reconstructions*, IEEE Trans. Image Process., 24 (2015), pp. 1809–1824, http://doi.org/10.1109/TIP.2015.2411437.
[10] P. Buyssens, O. Le Meur, M. Daisy, D. Tschumperlé, and O. Lézoray, *Depth-guided disocclusion inpainting of synthesized RGB-D images*, IEEE Trans. Image Process., 26 (2017), pp. 525–538, https://doi.org/10.1109/TIP.2016.2619263.
[11] J. Canny, *A computational approach to edge detection*, IEEE Trans. Pattern Anal. Machine Intell., 8 (1986), pp. 679–698.
[12] F. Cao, Y. Gousseau, S. Masnou, and P. Pérez, *Geometrically guided exemplar-based inpainting*, SIAM J. Imaging Sci., 4 (2011), pp. 1143–1179, https://doi.org/10.1137/110823572.
[13] T. Chan, S. Kang, and J. Shen, *Euler's elastica and curvature-based inpainting*, SIAM J. Appl. Math., 63 (2002), pp. 564–592, https://doi.org/10.1137/S0036139901390088.
[14] T. Chan and J. Shen, *Variational image inpainting*, Comm. Pure Appl. Math., 58 (2005), pp. 579–619.
[15] S. Choi, B. Ham, and K. Sohn, *Space-time hole filling with random walks in view extrapolation for 3D video*, IEEE Trans. Image Process., 22 (2013), pp. 2429–2441, https://doi.org/10.1109/TIP.2013.2251646.
[16] A. Criminisi, P. Pérez, and K. Toyama, *Region filling and object removal by exemplar-based image inpainting*, IEEE Trans. Image Process., 13 (2004), pp. 1200–1212.
[17] I. Daribo and B. Pesquet-Popescu, *Depth-aided image inpainting for novel view synthesis*, in 2010 IEEE International Workshop on Multimedia Signal Processing (Saint Malo, France), IEEE, Washington, DC, 2010, pp. 167–170, https://doi.org/10.1109/MMSP.2010.5662013.
[18] L. Do, G. Bravo, S. Zinger, and P. H. N. de With, *GPU-accelerated real-time free-viewpoint DIBR for 3DTV*, IEEE Trans. Consumer Electron., 58 (2012), pp. 633–640, https://doi.org/10.1109/TCE.2012.6227470.
[19] T. Dobbert, *Matchmoving: The Invisible Art of Camera Tracking*, Sybex, New York, 2005.
[20] C. Guillemot and O. Meur, *Image inpainting: Overview and recent advances*, IEEE Signal Process. Mag., 31 (2014), pp. 127–144, https://doi.org/10.1109/MSP.2013.2273004.
[21] J. Herling and W. Broll, *High-quality real-time video inpainting with PixMix*, IEEE Trans. Visualization Comput. Graphics, 20 (2014), pp. 866–879, https://doi.org/10.1109/TVCG.2014.2298016.
[22] L. Hocking, T. Holding, and C. Schönlieb, *Numerical Analysis of Shell-Based Geometric Image Inpainting Algorithms and Their Semi-Implicit Extension*, preprint, https://arxiv.org/abs/1707.09713, 2017.
[23] A. Kokaram, B. Collis, and S. Robinson, *Automated rig removal with Bayesian motion interpolation*, IEE Proc. Vision Image Signal Process., 152 (2005), pp. 407–414, https://doi.org/10.1049/ip-vis:20045152.
[24] Y. Lai, Y. Lai, and J. Lin, *High-quality view synthesis algorithm and architecture for 2D to 3D conversion*, in Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS 2012, Seoul, South Korea, 2012, pp. 373–376, https://doi.org/10.1109/ISCAS.2012.6272040.
[25] L. Ma, K. Agrawal, and R. Chamberlain, *A memory access model for highly-threaded many-core architectures*, Future Generation Comput. Syst., 30 (2014), pp. 202–215, https://doi.org/10.1016/j.future.2013.06.020.
[26] L. Ma, L. Do, and P. de With, *Depth-guided inpainting algorithm for Free-Viewpoint Video*, in Proceedings of the 19th IEEE International Conference on Image Processing (Orlando, FL), IEEE, Washington, DC, 2012, pp. 1721–1724, https://doi.org/10.1109/ICIP.2012.6467211.
[27] T. März, *Image inpainting based on coherence transport with adapted distance functions*, SIAM J. Imaging Sci., 4 (2011), pp. 981–1000, https://doi.org/10.1137/100807296.

[28] T. März, *A well-posedness framework for inpainting based on coherence transport*, Found. Comput. Math., 15 (2015), pp. 973–1033, https://doi.org/10.1007/s10208-014-9199-7.
[29] S. Masnou and J. Morel, *Level lines based disocclusion*, in Proceedings of the International Conference on Image Processing, ICIP 98, IEEE, Washington, DC, 1998, pp. 259–263.
[30] 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 Trans. Multimedia, 13 (2011), pp. 453–465, https://doi.org/10.1109/TMM.2011.2128862.
[31] 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, ACM, New York, 2013, 7, https://doi.org/10.1145/2534008.2534019.
[32] A. Newson, A. Almansa, M. Fradet, Y. Gousseau, and P. Pérez, *Video inpainting of complex scenes*, SIAM J. Imaging Sci., 7 (2014), pp. 1993–2019, https://doi.org/10.1137/140954933.
[33] NVIDIA, *CUDA C Programming Guide*, NVIDIA, Santa Clara, CA, 2015.
[34] 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 Picture Coding Symposium, PCS'09 (Chicago, IL), IEEE, Piscataway, NJ, 2009, pp. 233–236, https://doi.org/10.1109/PCS.2009.5167450.
[35] S. Roosta, *Parallel Processing and Parallel Algorithms: Theory and Computation*, Springer, New York, 2000, https://doi.org/10.1007/978-1-4612-1220-1.
[36] Y. Saad and M. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869, https://doi.org/10.1137/0907058.
[37] R. Sadek, G. Facciolo, P. Arias, and V. Caselles, *A variational model for gradient-based video editing*, Internat. J. Comput. Vision, 103 (2013), pp. 127–162, https://doi.org/10.1007/s11263-012-0597-5.
[38] 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 and Analysis of Visual Scenes, MVIEW '99 (Fort Collins, CO), IEEE Computer Society, Washington, DC, 1999, p. 21, https://doi.org/10.1109/MVIEW.1999.781079.
[39] C. Schönlieb, *Partial Differential Equation Methods for Image Inpainting*, Cambridge University Press, Cambridge, UK, 2015.
[40] M. Seymour, *Art of Stereo Conversion: 2D to 3D*, https://www.fxguide.com/featured/art-of-stereo-conversion-2d-to-3d-2012/, 2012.
[41] J. Sun, L. Yuan, J. Jia, and H. Shum, *Image completion with structure propagation*, ACM Trans. Graphics, 24 (2005), pp. 861–868, https://doi.org/10.1145/1073204.1073274.
[42] A. Telea, *An image inpainting technique based on the fast marching method*, J. Graphics Tools, 9 (2004), pp. 23–34.
[43] J. Weickert, *Anisotropic Diffusion in Image Processing*, ECMI Series, Teubner, Stuttgart, 1998, http://www.mia.uni-saarland.de/weickert/book.html.
[44] Y. Wexler, E. Shechtman, and M. Irani, *Space-time completion of video*, IEEE Trans. Pattern Anal. Mach. Intell., 29 (2007), pp. 463–476, https://doi.org/10.1109/TPAMI.2007.60.
[45] X. Xu, L. Po, C. Cheung, L. Feng, K. Ng, and K. Cheung, *Depth-aided exemplar-based hole filling for DIBR view synthesis*, in IEEE International Symposium on Circuits and Systems, ISCAS 2013 (Beijing, China), IEEE, Washington, DC, 2013, pp. 2840–2843, https://doi.org/10.1109/ISCAS.2013.6572470.
[46] S. Yoon, H. Sohn, Y. Jung, and Y. Ro, *Inter-view consistent hole filling in view extrapolation for multi-view image generation*, in IEEE International Conference on Image Processing, ICIP 2014 (Paris, France), IEEE, Washington, DC, 2014, pp. 2883–2887, https://doi.org/10.1109/ICIP.2014.7025583.
[47] J. Zhi, *A case of study for 3D stereoscopic conversion in visual effects industry*, Internat. J. Comput. Elect. Automat. Control Inform. Engrg., 7 (2013), pp. 53–58, http://iastem.com/Publications?p=73.
[48] S. Zinger, L. Do, and P. de With, *Free-viewpoint depth image based rendering*, J. Vis. Commun. Image Represent., 21 (2010), pp. 533–541, https://doi.org/10.1016/j.jvcir.2010.01.004.