SIAM J. IMAGING SCIENCES
Vol. 4, No. 4, pp. 981–1000

Image Inpainting Based on Coherence Transport with Adapted Distance Functions*

Thomas März†

Abstract. We discuss an extension of our method *image inpainting based on coherence transport*. For the latter method the pixels of the inpainting domain have to be serialized into an ordered list. Until now, to induce the serialization we have used the distance to boundary map. But there are inpainting problems where the distance to boundary serialization causes unsatisfactory inpainting results. In the present work we demonstrate cases where we can resolve the difficulties by employing other distance functions which better suit the problem at hand.

Key words. image processing, image inpainting, distance functions

AMS subject classifications. 65D18, 51K99

DOI. 10.1137/100807296

1. Introduction.

Nontexture image inpainting, also termed image interpolation, is the task of determining the values of a digital image for a destroyed, or consciously masked, subregion of the image domain.

The simple idea of the generic single pass method—which forms the basis for our method *image inpainting based on coherence transport* published in [5]—is to fill the inpainting domain by traversing its pixels in an onion peeling fashion from the boundary inward and thereby setting new image values as weighted means of given or already calculated ones.

Telea has been the first to use such an algorithm in [13]: the pixels are serialized according to their Euclidean distance to the boundary of the inpainting domain, and the weight is such that image values are propagated mainly along the gradient of the distance map. By his choices of the weight and the pixel serialization, the method of Telea is not adapted to the image.

In [5], we addressed the adaption of the weight: our method uses an image dependent weight such that image values are propagated along the estimated tangents of color lines which have been interrupted by the inpainting domain. In that way we could improve the quality of the inpainting results compared to Telea (see Figure 1). Beyond that, we illustrated in [5] that our method matches the high level of quality of the methods in [4], [10], [9], and [14] while being considerably faster.

However, serializing the pixels by their distance to the boundary, as Telea [13] and we [5] did, is not always a good idea, as Figure 2 illustrates. The image in the middle shows the

*Received by the editors September 1, 2010; accepted for publication (in revised form) July 26, 2011; published electronically October 20, 2011. This work was supported in part by the Graduiertenkolleg Angewandte Algorithmische Mathematik (GKAAM) funded by the Deutsche Forschungsgemeinschaft (DFG) at the Technische Universität München (TUM), and by award KUK-C1-013-04 made by King Abdullah University of Science and Technology (KAUST).
http://www.siam.org/journals/siims/4-4/80729.html
†Mathematical Institute, University of Oxford, Oxford OX1 3LB, UK (maerz@maths.ox.ac.uk).

Figure 1. *Scratch removal. Left: vandalized image; the white scratches are the inpainting domain. Image from* [13, Figure 8.i] *(Copyright* 2004. *From "An image inpainting technique based on the fast marching method," by A. Telea. Reproduced by permission of Taylor & Francis Group, LLC,* http://www.taylorandfrancis.com*). Middle: result of Telea's method. Right: result of our method.*

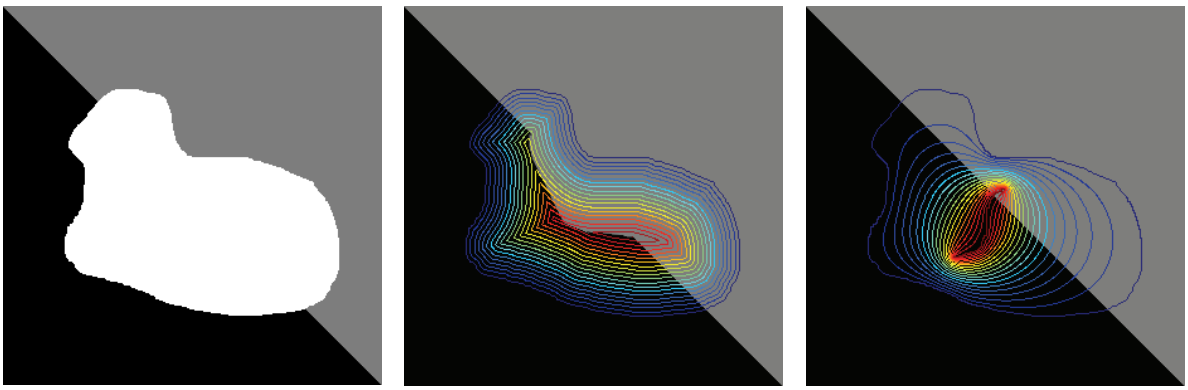


Figure 2. *A broken diagonal? Left: data image with inpainting domain in white. Middle: result of our method, pixel serialization by distance to boundary; levels of the distance to boundary map are overlaid. Right: result of our method, pixel serialization by another distance function; see section* 4.1.

result which we obtain by using our method with distance to boundary serialization. The diagonal is not restored, and the contours of the distance to boundary map indicate that this is due to the poor location of the skeleton which consists of the ridges of the distance map. Before reaching the skeleton, however, the performance is good and the diagonal is continued tangentially. The right image of Figure 2 shows the result which we get if we use another distance function (for pixel serialization) that better suits the problem at hand (see section 4.1, Figure 3).

The way of serializing the pixels is an important degree of freedom and has already been addressed in patch-based inpainting methods; see, e.g., the articles [6], [12], and [3]. For the methods of the latter two papers the user can provide hand-driven geometric constraints and thereby control the filling process.

Similarly, the pixel serializations that we discuss here are all induced by a more general type of distance function which respects geometric constraints specified by the user. Our method is still a PDE-based geometric inpainting technique, and the new distance functions help to resolve some obstructions which the Euclidean distance to boundary serialization entails.

Outline of the paper. In section 2 we summarize the existing results and give a complete description of the basic algorithm. Section 3 describes the interface of the algorithm. In section 4 we turn to concrete distance functions which are used to serialize the pixels of the inpainting domain. We describe three approaches: distance by harmonic interpolation (section 4.1), modified distance to boundary (section 4.2), and distance to skeleton (section 4.3). The experiments here are restricted to synthetic examples. Section 5 discusses some natural examples.

2. Summary of existing results. Our starting point is the generic algorithm for gray tone images. We assume that all gray tone images, seen as functions, take values in the real interval $[0, 255]$, and we assume that gray tones are mapped onto $[0, 255]$ such that the natural order on the interval reflects the order of the shades of gray by their brightness from black ($= 0$) to white ($= 255$). Moreover, we distinguish between discrete (digital) notions by using the index h and continuous (analog) notions where we omit the index. Continuous notions are thought of as the high-resolution limit of the corresponding discrete notions. Finally, we identify pixels with their midpoints.

Notation.

(a) $\Omega_{0,h}$ is the image domain, the matrix of pixels for the final, restored image $u_h : \Omega_{0,h} \to [0, 255]$.
(b) $\Omega_h \subset \Omega_{0,h}$ is the inpainting domain whose values of u_h have to be determined.
(c) $\Omega_{0,h} \backslash \Omega_h$ is the data domain whose values of u_h are given as $u_h|_{\Omega_{0,h}\backslash\Omega_h} = u_{0,h}$.
(d) $\partial\Omega_h \subset \Omega_h$ is the discrete boundary, i.e., the set of inpainting pixels that have at least one neighbor in the data domain.

Continuous quantities are defined correspondingly. Finally, we define discrete and continuous ε-neighborhoods by

$$B_{\varepsilon,h}(x) := \{y \in \Omega_{0,h} : |y - x| \le \varepsilon\} , \quad B_\varepsilon(x) := \{y \in \Omega_0 : |y - x| \le \varepsilon\} .$$

Generic algorithm. The basic idea is to fill the inpainting domain from its boundary inward by using weighted means of given or already calculated image values. We assume that the pixels which are to be inpainted have already been serialized, i.e., $\Omega_h = (x_1, x_2, \dots, x_N)$ is an ordered list. The set

$$B^<_{\varepsilon,h}(x_k) := B_{\varepsilon,h}(x_k) \backslash \{x_k, \dots, x_N\} , \quad k = 1, \dots, N,$$

denotes the neighborhood of the pixel x_k consisting only of known or already inpainted pixels. Then, the algorithm reads as follows:

$$u_h|_{\Omega_{0,h}\backslash\Omega_h} = u_{0,h} ,$$

$$u_h(x_k) = \frac{\displaystyle\sum_{y \in B^<_{\varepsilon,h}(x_k)} w(x_k, y) u_h(y)}{\displaystyle\sum_{y \in B^<_{\varepsilon,h}(x_k)} w(x_k, y)} , \quad k = 1, \dots, N. \tag{2.1}$$

Here, $w(x,y) \geq 0$ are called the weights of the algorithm, and we assume that

$$\sum_{y \in B^<_{\varepsilon,h}(x)} w(x,y) > 0 \,, \quad x \in \Omega_h \,.$$

Pixel serialization. In the generic algorithm, any serialization of the pixels can be used which geometrically goes from the boundary inward in an onion peeling fashion. Now, let $T : \Omega \to [0, T_{\max}]$ be a map with $T|_\Gamma = 0$, $\Gamma \subset \partial\Omega$, and which strictly grows into the interior of Ω. We define the onion peels to be the level lines of T. Using such a map T or rather a discretized version $T_h : \Omega_h \to [0, T_{\max}]$, we serialize the pixels of Ω_h by

$$T_h(x_j) < T_h(x_k) \quad \Rightarrow \quad j < k \tag{2.2}$$

into an ordered list $\Omega_h = (x_1, x_2, \ldots, x_N)$. Unfortunately, the serialization rule (2.2) cannot order pixels which belong to the same onion peel where the T_h-values are all equal. Now, in the case of $T_h(x_k) = T_h(x_j)$ we force the computations of the values $u_h(x_k)$ and $u_h(x_j)$ to be independent of each other by redefining the pixel neighborhood in terms of T_h as

$$B^<_{\varepsilon,h}(x_k) := \{x \in B_{\varepsilon,h}(x_k) : T_h(x) < T_h(x_k) \text{ if } x \in \Omega_h\} \,.$$

Hence we can process along an onion peel of T_h in arbitrary order.

Clearly, there are different possible choices of a suitable T. In the article [5] this degree of freedom was fixed by setting $T = d$, where d is the Euclidean distance to the boundary $\partial\Omega$,

$$d(x) = \operatorname{dist}(x, \partial\Omega) \,, \quad x \in \Omega \,.$$

As pointed out in the introduction $T = d$ is not always a good choice, and in the next section we will use generalized distances T.

Weights. Here again, different choices for the weights are possible. We consider weights of the form

$$w(x,y) = \frac{1}{|x-y|}\, k\left(x, (x-y)\cdot\varepsilon^{-1}\right) \tag{2.3}$$

with a kernel $k(x,\eta)$.

(a) *Normal transport* (Telea's method). Telea, in [13], uses the kernel

$$k(x,\eta) = \frac{|\,\langle N(x), \eta\rangle\,|}{|\eta|} \,, \quad \text{with} \quad N(x) = \frac{\nabla T(x)}{|\nabla T(x)|} \,.$$

Though Telea has taken $T = d$, the kernel carries over to more general T.

(b) *Coherence transport* (our method of [5]). For a given guidance field called g we use the kernel

$$k_\mu(x,\eta) = \sqrt{\frac{\pi}{2}}\mu \exp\left(-\frac{\mu^2}{2}\left\langle g^\perp(x), \eta\right\rangle^2\right) \,. \tag{2.4}$$

The guidance vector $g(x)$, supposed to be an approximate tangent, is computed by employing structure tensor analysis. The set-up of the structure tensor $S_{\sigma,\rho}(x)$ is as follows:

$$\alpha_\sigma(y,x) = \int_{\Omega(x)} K_\sigma(y-h)\,dh,$$

$$v_\sigma(y,x) = \frac{1}{\alpha_\sigma(y,x)} \cdot \int_{\Omega(x)} K_\sigma(y-h)\cdot u(h)\,dh,$$

$$S_{\sigma,\rho}(x) = \frac{1}{\alpha_\rho(x,x)} \cdot \int_{\Omega(x)} K_\rho(x-y)\cdot \nabla_y v_\sigma(y,x)\cdot \nabla_y v_\sigma^T(y,x)\,dy,$$

where $\Omega(x) := \{y \in \Omega : T(y) < T(x)\} \cup \Omega_0\backslash\Omega$ and K_σ is a Gaussian kernel. The coherence vector $g(x)$ which enters k_μ as guidance is the eigenvector of $S_{\sigma,\rho}(x)$ w.r.t. the minimal eigenvalue. Since g depends on the image u, the weight does here, too. For more details on the usage of the structure tensor w.r.t. coherence transport inpainting, see [5] and [8]. For other applications of structure tensor analysis and the related scale space theory, see, e.g., [1], [15], and [2].

A note on the theory behind our method. Because continuous theory is not an issue of this paper, we want at least make some comments. For weights of the form (2.3) we have shown in [5] and [8] that the high-resolution ($h \to 0$) vanishing-viscosity ($\varepsilon \to 0$) limit yields transport/advection PDEs of the following forms:

(a) *Normal transport*:

$$\langle N(x), \nabla u(x)\rangle = 0 \quad \text{in } \Omega\backslash\Sigma\,, \qquad u|_{\partial\Omega} = u_0|_{\partial\Omega}\,.$$

(b) *Coherence transport*:

$$\langle c_\mu(x), \nabla u(x)\rangle = 0 \quad \text{in } \Omega\backslash\Sigma\,, \qquad u|_{\partial\Omega} = u_0|_{\partial\Omega}\,, \qquad \langle c_\mu(x), N(x)\rangle \geq \beta_\mu > 0\,. \tag{2.5}$$

In both cases the set Σ denotes the set where $N(x) = \nabla T(x)/|\nabla T(x)|$ is singular. The side condition $\langle c_\mu(x), N(x)\rangle \geq \beta_\mu$ is the continuous counterpart of the pixel serialization: here the function T plays the role of a Lyapunov function, and this condition says that the characteristics of the advection PDE evolve strictly into the interior of Ω and stop at Σ. The general existence and well-posedness theory for such type of equation and its application to the analog inpainting model above can be found in [8].

Finally, in order to better understand the role of the parameter μ of the kernel k_μ, we cite two results of [5]. First, according to [5, Theorem 1], every weight w of the form (2.3) corresponds to a transport field c. Second, in [5, Theorem 2], when using the coherence transport kernel k_μ, we have proved an asymptotic expansion of $c_\mu(x)$, w.r.t. $\mu \to \infty$, which implies the following limit behavior:

$$\lim_{\mu\to\infty} c_\mu(x) = \begin{cases} g(x), & \langle g(x), N(x)\rangle > 0, \\ -g(x), & \langle g(x), N(x)\rangle < 0, \\ N(x), & \langle g(x), N(x)\rangle = 0. \end{cases} \tag{2.6}$$

That means the coherence vector $g(x)$ in fact guides the coherence transport (2.5) and μ is the strength of this guidance.

3. Interface of the algorithm. In the next section the algorithm will be performed only in its coherence transport version. That is, the weight function w has the form of (2.3) with the kernel k_μ given by (2.4). The execution of the coherence transport algorithm depends on the choice of four parameters:

- ε, the averaging radius,
- μ, the guidance strength of k_μ,
- σ and ρ, the scale parameters of the smoothing operations in the structure tensor $S_{\sigma,\rho}$.

Finally, the algorithm is supplied with the data image u_0 and a sorted list of the pixels which are to be inpainted. Any item of this list is a triple

$$[i \quad j \quad T_h(i,j)] \,,$$

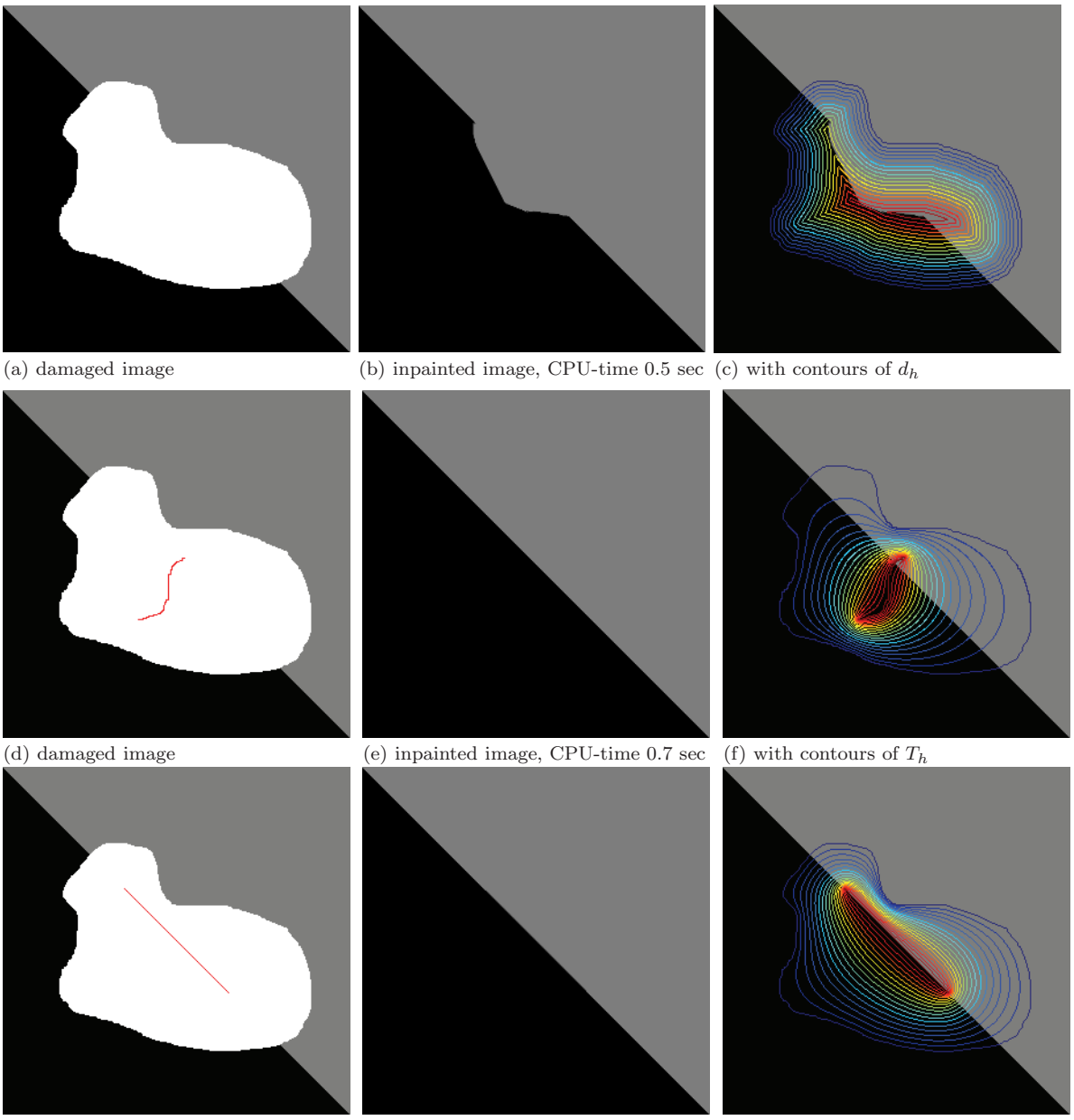
where (i,j) are the pixel coordinates and the list is sorted in ascending order of the distance values $T_h(i,j)$. Hence, pixel serialization, which means setting up this list, is a preprocessing step.

For comparing CPU-times. The experiments in the following sections have been performed using MATLAB 7.6 (R2008a) on a MacBook with a 2.33 GHz Intel Core 2 Duo processor and 2 GB of RAM. The codes of the inpainter and that of the fast marching–based pixel serializers are written in C with a MEX interface to MATLAB. For each inpainting experiment we state the overall CPU-time, i.e., serialization plus inpainting.

4. Concrete distance functions. The generic algorithm of section 2 depends on a prescribed pixel serialization, which orders the pixels from the boundary inward. In all previous experiments of [5] the pixels were serialized by their Euclidean distance to boundary. The advantage in practice is that for all types of domains the approximate distance to boundary map is easy to generate by the fast marching method (see [11] and [7]). But the disadvantage is that it is not always the best choice if one wants to get a nice looking inpainting result. Here, we present three other ways of setting up a generalized discrete distance function T_h. We show a few synthetic examples where these methods yield better inpaintings than the distance to boundary.

4.1. Distance by harmonic interpolation. The broken diagonal is our first example. Figure 3(b) shows the result when using the distance to boundary d_h. A restored diagonal would be a nicer result, but only parts of the diagonal have been continued correctly. Figure 3(c)—with the contours of d_h overlaid—demonstrates that the undesired effect is due to the *wrong* location of the stop set. Figures 3(e) and 3(h) show the result when using T_h which we obtain by prescribing a better located stop set (red lines in Figures 3(d) and 3(g)) and harmonic interpolation. Here, the algorithm is able to restore the diagonal. For all three cases we have used the same set of parameters $[\varepsilon, \mu, \sigma, \rho] = [3, 50, 0.5, 5]$.

Now, we describe the construction of the discrete distance function T_h. Here, we can prescribe the location of the stop set arcs and their distance values.

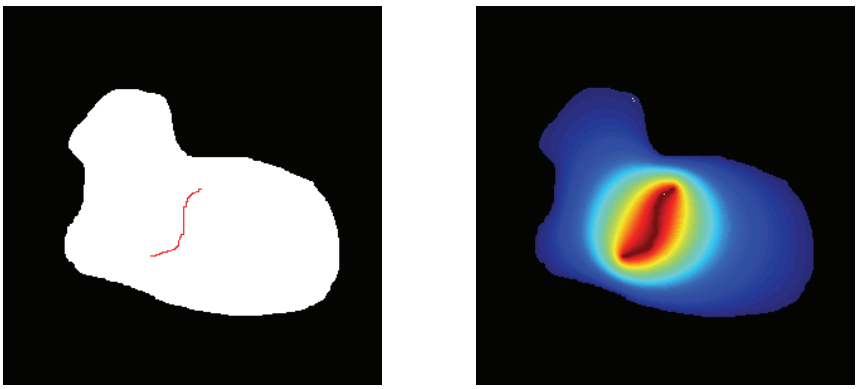


(a) damaged image (b) inpainted image, CPU-time 0.5 sec (c) with contours of d_h

(d) damaged image (e) inpainted image, CPU-time 0.7 sec (f) with contours of T_h

(g) damaged image (h) inpainted image, CPU-time 0.7 sec (i) with contours of T_h

Figure 3. *Broken diagonal. In the first row we have* (a) *the damaged image with the inpainting domain* Ω_h *in white,* (b) *the result inpainted using Euclidean distance to boundary* d_h*, and* (c) *the result with contours of* d_h*. In the second and third rows, we have* (d), (g) *the same damaged image with a specified stop set* Γ_h *in red;* (e), (h) *the result inpainted using distance by harmonic interpolation* T_h*; and* (f), (i) *the result with contours of* T_h*. (Image size:* 301×301*, damage:* 29%*.)*



(a) white: Ω_h, red: Γ_h^1 with $t_1 = 127$ (b) contours of T_h

Figure 4. *Single Γ_h yields a valid T_h.*

Because the boundary is the start set, the discrete distance function T_h shall equal zero on $\partial\Omega_h$. In addition, we take at least one or more discrete curves Γ_h^k, $k \in \{1, \dots, n\}$, which are contained in Ω_h and should belong to the stop set Σ_h. Moreover, for every Γ_h^k we specify a distance value $t_k > 0$. The remainder of T_h is then calculated by harmonic interpolation. That is, we solve the discrete Laplace equation

$$\begin{aligned} \Delta_h T_h = 0 \quad \text{in} \quad \Omega_h \backslash \Sigma_h \,, \qquad & \Sigma_h = \bigcup_{k=1}^{n} \Gamma_h^k \,, \\ T_h = 0 \quad \text{on} \quad \partial\Omega_h \,, \qquad & T_h = t_k \quad \text{on} \quad \Gamma_h^k \,, \quad k = 1, \dots, n \,. \end{aligned}$$

Hereby, the discretization Δ_h of the Laplacian is due to the five-point-stencil

$$\Delta_h T_h(i,j) = T_h(i-1,j) + T_h(i,j-1) - 4T_h(i,j) + T_h(i,j+1) + T_h(i+1,j) \,.$$

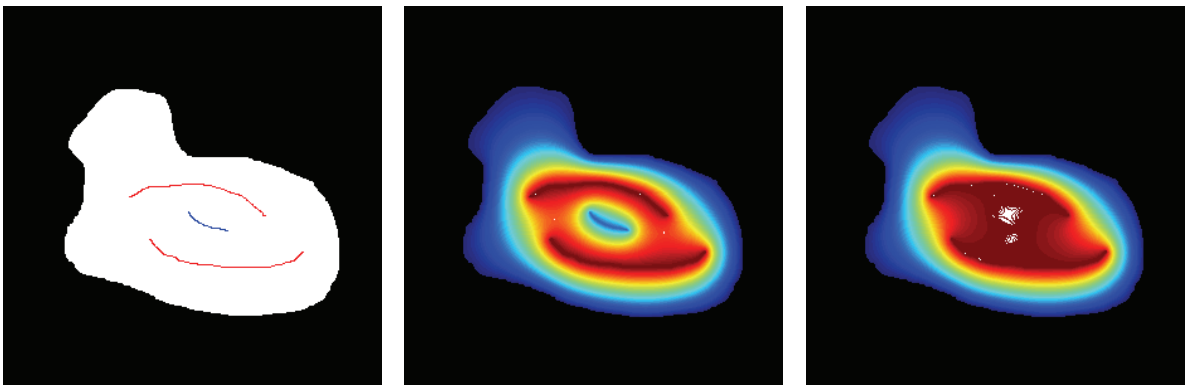
The resulting linear system is symmetric, banded, and sparse: we solve it using the MATLAB backslash-operator for sparse matrices.

Since harmonic interpolation provides a minimum and maximum principle, our construction of T_h can be imagined as setting up a tent roof over the domain Ω_h, where Γ_h^k are the locations of the tent poles and every tent pole of Γ_h^k has the length t_k.

In the case of a single curve Γ_h, the resulting T_h is always a valid distance because of the minimum and maximum principle; see Figure 4. Moreover, the choice of the prescribed distance value $t > 0$ does not matter in this case: we will of course get different distance functions T_h, but they all induce the same pixel serialization.

Unfortunately, not every choice of curves Γ_h^k, with time values t_k, results in a valid distance function: T_h might have local minima and so would not strictly increase into the interior of Ω_h. If there are two or more curves Γ_h^k, the question of whether the resulting T_h is valid or not depends on the location of the curves in relation to each other and the differences in their prescribed values t_k . Figure 5(b) shows an example with three curves Γ_h^1, Γ_h^2, and Γ_h^3 with

$$t_1 = t_2 = 250 \quad > \quad t_3 = 50 \,.$$



(a) white Ω_h (b) nonvalid T_h (c) valid T_h

Figure 5. (a) *shows the domain* Ω_h *and the desired stop arcs, red:* Γ_h^1 *and* Γ_h^2*, blue:* Γ_h^3*. For* (b) *the distances on* Γ_h^k *have been set to* $t_1 = t_2 = 250$*,* $t_3 = 50$*, and we can see that the resulting* T_h *has* Γ_h^3 *as local minimum and thus is not valid. For* (c) *the distances on* Γ_h^k *have been set to* $t_1 = t_2 = 250$*,* $t_3 = 249$*, and the resulting* T_h *is free of minima.*

Here, all points of Γ_h^3 are local minima of T_h. But, if we keep the geometry of Γ_h^1, Γ_h^2, and Γ_h^3, and change the prescribed distances to

$$t_1 = t_2 = 250 \quad > \quad t_3 = 249\ ,$$

then T_h does not have any minima (see Figure 5(c)). Generally speaking, if we have two or more curves Γ_h^k and if the values t_k are chosen unfavorably, then the resulting T_h might possess local minima on some of the Γ_h^k. Luckily, again owing to the minimum and maximum principle, there is always a set of t_k-values such that the resulting T_h is free of local minima.

Let us review our first example. Figure 3(e) illustrates that our T_h—which is that shown in Figure 4(b)—works fine. More generally, if we think of Σ_h as an initial scratch, which has been dilated to all of Ω_h over the time T_h, then the backward filling-in process, if Σ_h is well located, makes the matching opposite sides come together. Figures 3(g)–3(i) demonstrate that it is also possible to put Σ_h on the diagonal to be restored. Clearly, if we deliberately place Σ_h badly (for this example that means Σ_h is not on the diagonal and prevents the parts of the diagonal from coming together), then the method cannot produce the diagonal (see Figure 6, where the parameters are $[\varepsilon, \mu, \sigma, \rho] = [3, 50, 0.5, 5]$, as before). This might also be seen as an alternative if a user does not want a restored diagonal: choose a Σ_h that prevents the edges from coming together but which has a nice shape.

Now, we discuss two further examples. Figure 7(a) shows two broken diagonals with the same inpainting domain as in Figure 3(a). We emphasize here that the appearance of an undesired effect depends on how the edge that needs to be continued is located in relation to the inpainting domain. In Figure 7(b)—which is inpainted using distance to boundary—the bottom-left-to-top-right diagonal is continued as desired, while the continuation of the top-left-to-bottom-right diagonal suffers from a badly located stop set. In Figures 7(d) and 7(g) we have the damaged image with a single curve Γ_h, $t_1 = 127$, shown in red (for the latter (g) Γ_h is just a point). Figures 7(e) and 7(h) each show the results inpainted using the corresponding

(a) damaged image (b) inpainted image, CPU-time 0.7 sec (c) with contours of T_h

Figure 6. *Broken diagonal. The method must fail if the stop set is deliberately placed badly.* (a) *the damaged image with the inpainting domain Ω_h in white with a single stop arc Γ_h in red,* (b) *the result inpainted using distance by harmonic interpolation T_h, and* (e) *the result with contours of T_h. (Image size:* 301×301*, damage:* 29%*.)*

distance by harmonic interpolation T_h. Again, a good location of Σ_h makes for a good result. For all three cases we have used the same set of parameters $[\varepsilon, \mu, \sigma, \rho] = [3, 50, 0.5, 5]$.

Figure 8(a) shows a damaged cross junction. A cross junction would, in any case, geometrically be the simplest object for completion. Using the distance to boundary, we obtain the result shown in Figure 8(b). Here, the stop set which is the central arc of the skeleton (see Figure 8(c)), has an unfavorable location because the bar coming from the right-hand side can never reach its counterpart. Setting Γ_h as the center of the cross junction (see Figure 8(d)) and using the distance by harmonic interpolation, we are able to restore the cross junction (see Figure 8(e)). Which of the bars is closed in the end depends on the coherence strength. The brighter bar has the higher contrast w.r.t. the black background and is thus of stronger coherence. This is the reason why this bar is closed. For both cases we have used the same set of parameters $[\varepsilon, \mu, \sigma, \rho] = [5, 100, 0.5, 10]$.

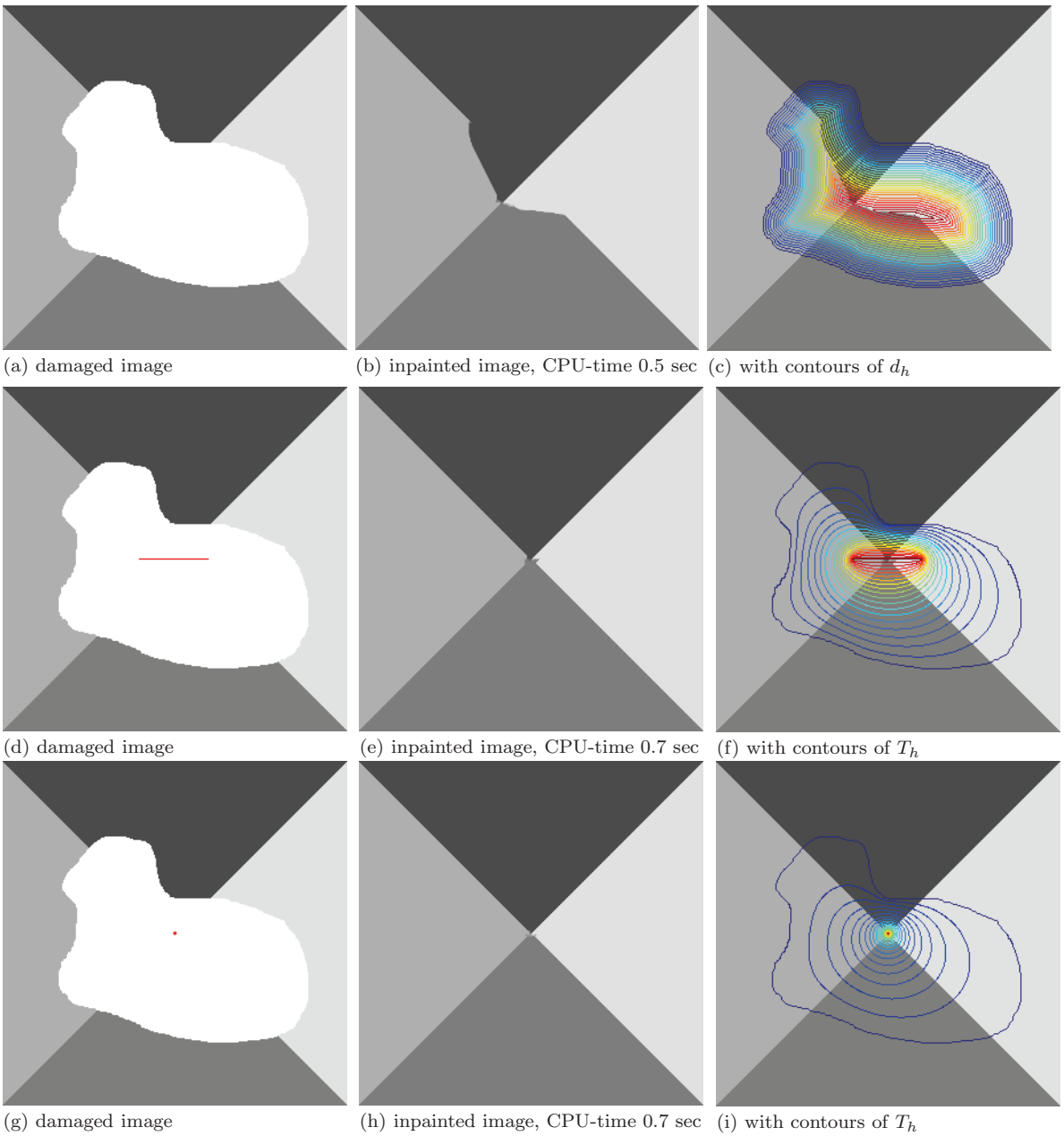
4.2. Modified distance to boundary. Figure 9(a) shows a damaged stripe pattern. Performing the algorithm with distance to boundary and the set of parameters $[\varepsilon, \mu, \sigma, \rho] = [5, 100, 0.5, 10]$ yields the result shown in Figure 9(b). The difficulty is that the coherence vector which is tangent to the edges is orthogonal to the lower left and the upper right segment of $\partial\Omega_h$. Thus, the transport vector c switches to the normal N as in the exceptional case of (2.6). Figure 9(c) confirms the switch of the transport to N.

To combat this we suggest a modification of the distance map set-up. The Euclidean distance to boundary map d is the viscosity solution of

$$|\nabla d| = 1 \quad \text{in} \quad \Omega\,, \qquad\qquad d|_{\partial\Omega} = 0\,.$$

The modification is that we search for the Euclidean distance d_* to a subset Γ of the boundary $\partial\Omega$, i.e.,

$$|\nabla d_*| = 1 \quad \text{in} \quad \Omega\,, \qquad\qquad d_*|_{\Gamma} = 0\,. \tag{4.1}$$



(a) damaged image (b) inpainted image, CPU-time 0.5 sec (c) with contours of d_h

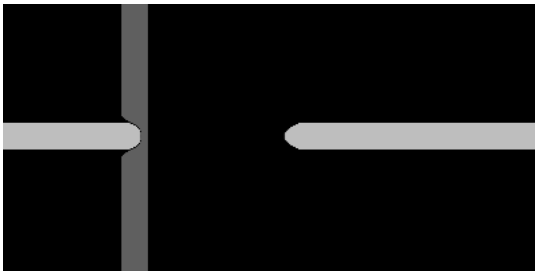
(d) damaged image (e) inpainted image, CPU-time 0.7 sec (f) with contours of T_h

(g) damaged image (h) inpainted image, CPU-time 0.7 sec (i) with contours of T_h

Figure 7. *Two broken diagonals. In the first row we have* (a) *the damaged image with the inpainting domain* Ω_h *in white,* (b) *the result inpainted using Euclidean distance to boundary* d_h*, and* (c) *the result with contours of* d_h*. The result in* (b) *demonstrates that the undesired effect depends on how edges are located in relation to the inpainting domain. In the second and third rows we have* (d), (g) *the same damaged image with a specified stop set* Γ_h *in red;* (e), (h) *the result inpainted using distance by harmonic interpolation* T_h*; and* (f), (i) *the result with contours of* T_h*. (Image size:* 301×301*, damage:* 29%*.)*

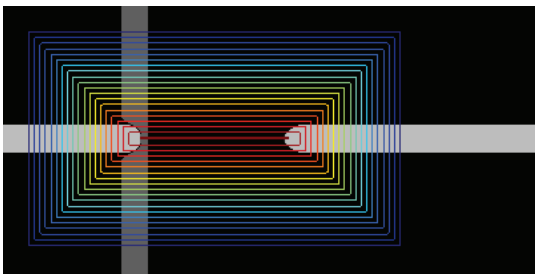
(a) damaged image

(d) damaged image, red Γ_h

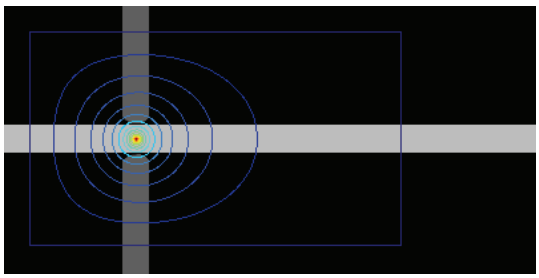


(b) inpainted image, CPU-time 0.7 sec

(e) inpainted image, CPU-time 1.3 sec



(c) with contours of d_h



(f) with contours of T_h

Figure 8. *Broken cross junction. In the first column we have* (a) *the damaged image with the inpainting domain* Ω_h *in white,* (b) *the result inpainted using Euclidean distance to boundary* d_h*, and* (c) *the result with contours of* d_h*. In the second column we have* (d) *the same damaged image with the stop set* Γ_h *being a single point in red (at the center of the cross junction),* (e) *the result inpainted using distance by harmonic interpolation* T_h*, and* (f) *the result with contours of* T_h*. (Image size:* 200×400*, damage:* 56%*.)*

We classify the points which shall not belong to Γ. Assume $x \in \partial\Omega$ satisfies $d_*(x) = 0$; then the boundary normal is given by $N(x) = \nabla d_*(x)$. Now, if we have at x

$$\langle g(x), N(x)\rangle^2 = 0 \ ,$$

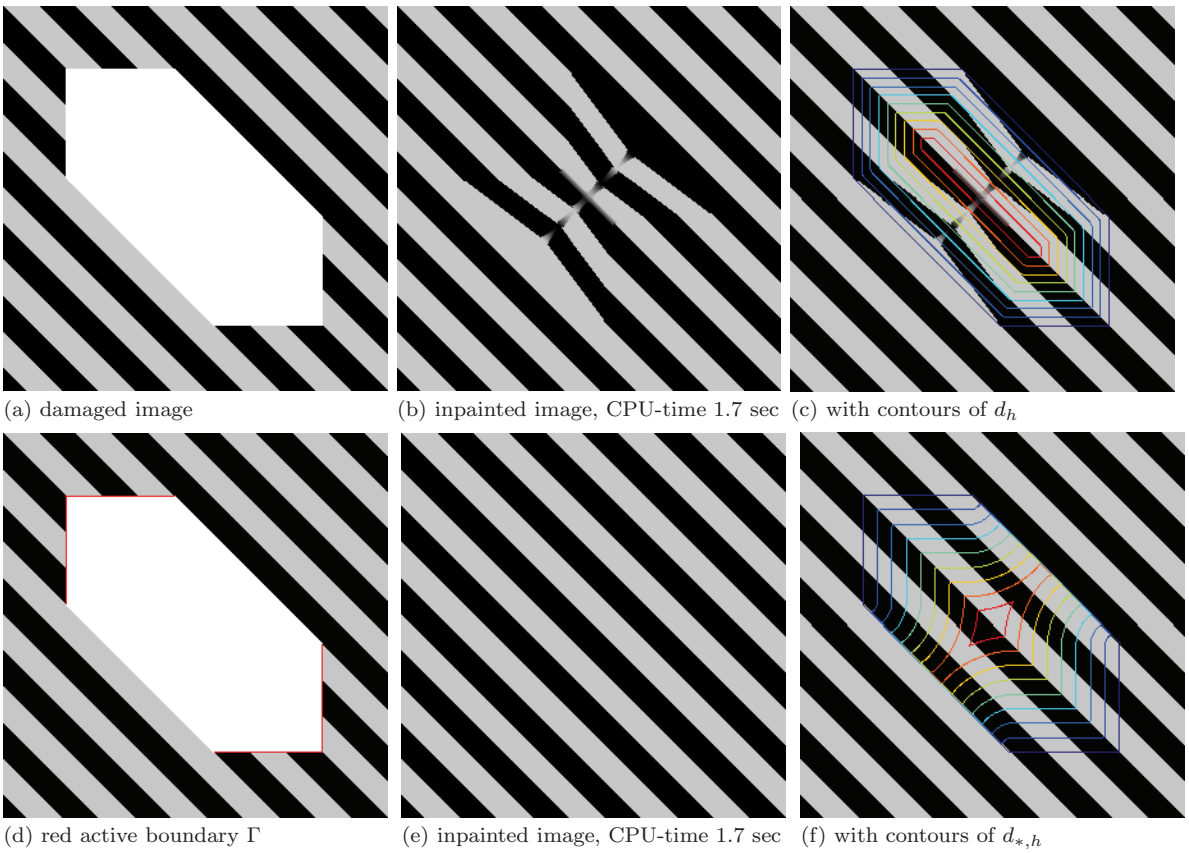
where g is the guidance vector, then either there is no guidance ($|g(x)| = 0$) or the guidance vector does not point inward. Such a boundary point shall not belong to Γ. In fact, we use the stronger condition

$$0 \le \langle g(x), N(x)\rangle^2 \le \gamma$$

with a threshold parameter $0 < \gamma \le 1$. Complementarily, the set of active boundary points Γ is given by

$$\Gamma = \{x \in \partial\Omega : \langle g(x), N(x)\rangle^2 > \gamma\} \ .$$

Of course, the new parameter γ must be chosen such that Γ is not empty.



(a) damaged image (b) inpainted image, CPU-time 1.7 sec (c) with contours of d_h

(d) red active boundary Γ (e) inpainted image, CPU-time 1.7 sec (f) with contours of $d_{*,h}$

Figure 9. *Stripe pattern. The difficulty is that the damage is aligned with the pattern. In the first row we have* (a) *the damaged image with the inpainting domain Ω_h in white,* (b) *the result inpainted using Euclidean distance to boundary d_h, and* (c) *the result with contours of d_h. In the second row we have* (d) *the same damaged image with the active part Γ of the boundary in red,* (e) *the result inpainted using $d_{*,h}$, which is the Euclidean distance to Γ, and* (f) *the result with contours of $d_{*,h}$. (Image size:* 301×301*, damage:* 30%*.)*

After having found the active boundary Γ, we use the fast marching method to solve (4.1) for $d_{*,h}$.

We have applied this idea to the stripe pattern: the red lines in Figure 9(d) are the active boundary points, and the result is shown in Figure 9(e). Our standard parameters are $[\varepsilon, \mu, \sigma, \rho] = [5, 100, 0.5, 10]$, while the additional parameter is set to $\gamma = 0.1$. The overlaid contour plot of $d_{*,h}$ in Figure 9(f) shows that the guidance vector always points inward.

4.3. Distance to skeleton. The third approach to obtaining a serialization is to use the distance to a prescribed stop part of the skeleton. Let $\mathcal{S}^k$, $k \in \{1, \dots, n\}$, be curves in the image domain Ω_0. Those curves $\mathcal{S}^k$ which are contained in Ω will later belong to the skeleton $\mathcal{S}$. Then, let T_* be the viscosity solution of

$$|\nabla T_*| = 1 \quad \text{in} \quad \Omega_0 \,, \qquad T_* = 0 \quad \text{on} \quad \mathcal{S}^k \,, \quad k \in \{1, \dots, n\} \,,$$

and let

$$T_{*,\max} = \max_{x \in \Omega} T_*(x) \,.$$

(a) damaged image (b) inpainted image, CPU-time 0.8 sec (c) with contours of T_h

Figure 10. *Broken diagonal.* (a) *shows the damaged image, white* Ω_h*, with the prescribed skeleton arc* $\mathcal{S}_h^1$ *in red.* (b) *is the result inpainted using distance to skeleton* T_h*.* (c) *is the result with contours of* T_h*. (Image size:* 301×301*, damage:* 29%*.)*

The desired distance function is defined by

$$T(x) = T_{*,\max} - T_*(x) \,, \quad x \in \Omega \,.$$

Here again, we use the fast marching method to get the approximation $T_{*,h}$.

Warning: as in the case of harmonic interpolation (see section 4.1) one must check whether T is admissible, i.e., whether T is free of local minima.

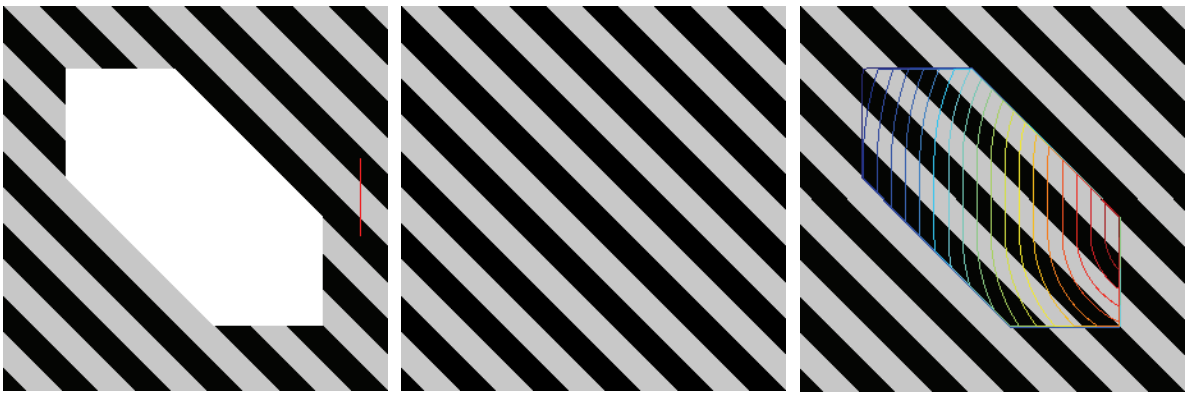
Figure 10 shows the result for the example of the broken diagonal. The red curve in Figure 10(a) defines $\mathcal{S}_h^1$ (discrete) which shall belong to the skeleton. The set of parameters is $[\varepsilon, \mu, \sigma, \rho] = [3, 50, 0.5, 5]$, the same as we used for Figure 3(e). The inpainted result here (Figure 10(b)) is the same as in Figure 3(e), but the distance function has changed (compare Figures 10(c) and 3(f)). If there is only one curve $\mathcal{S}_h^1$ which is completely contained in Ω, then distance by harmonic interpolation (with $\Gamma_h^1 = \mathcal{S}_h^1$) and distance to skeleton will yield very similar results. If there are two or more curves, then distance by harmonic interpolation allows for different distance values on Γ_h^k, while distance to skeleton has exactly one distance value on all of the curves $\mathcal{S}_h^k$. In contrast to distance to skeleton, the distance by harmonic interpolation method requires $T_h|_{\partial\Omega_h} = 0$.

Moreover, since T_* is defined on Ω_0 (the full image domain), we can place $\mathcal{S}^k$ outside of the inpainting domain Ω. We use this possibility to restore the stripe pattern in Figure 11. The parameters are $[\varepsilon, \mu, \sigma, \rho] = [5, 100, 0.5, 10]$, the same as we used for Figure 9. It is obvious from the level lines of T_h (see Figure 11(c)) that the guidance vector always points inward. Thus, we get the desired result again here.

Remark. The construction here, as well as that of section 4.2, produces distance functions where not all of the boundary belongs to the start set. In addition, it is possible that parts of the boundary in fact belong to the stop set.

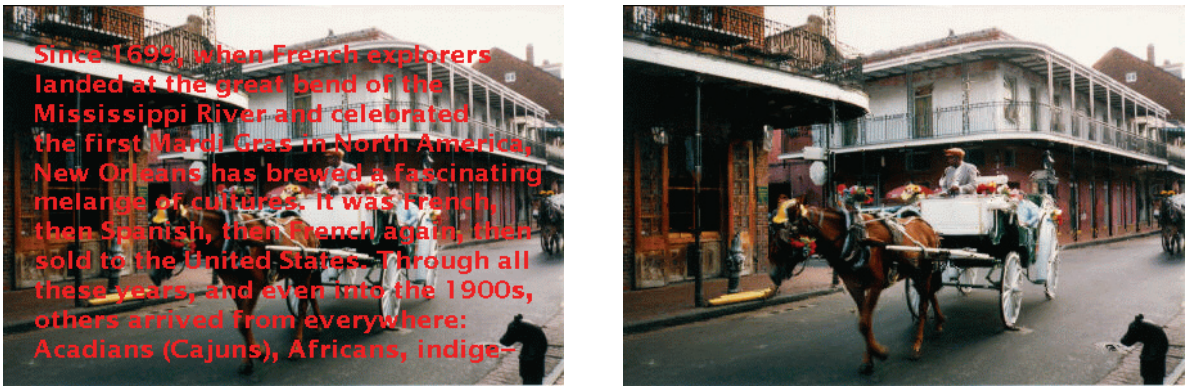
5. Natural inpainting examples.

In the previous section we have demonstrated that there are inpainting problems where a distance to boundary induced serialization of the pixels is not



(a) damaged image (b) inpainted image, CPU-time 2.0 sec (c) with contours of T_h

Figure 11. *Stripe pattern.* (a) *shows the damaged image, white* Ω_h*, with the red arc* $\mathcal{S}_h^1$ *outside* Ω_h*.* (b) *is the result inpainted using* T_h*, the distance to "skeleton"* $\mathcal{S}_h^1$*.* (c) *is the result with contours of* T_h*. (Image size:* 301×301*, damage:* 30%*.)*



(a) original (courtesy of Bertalmio et al. [4, Figure 6]) (b) inpainted: $[\varepsilon, \mu, \sigma, \rho] = [4, 25, 2, 3]$, CPU-time 0.9 sec

Figure 12. *Removal of superimposed text. The letters in* (a) *are the inpainting domain* Ω_h*.* (b) *inpainted using Euclidean distance to boundary. (Image size:* 296×437*, damage:* 16%*.) A comparison to the result of Bertalmio et al.* [4] *can be found in* [5].

favorable, and we have constructed other distances which are able to resolve the difficulties which appear.

So far we have considered only synthetic images, because their image geometry is easy to understand. Thus, we were able to construct distances which are adapted to the image or rather to an expected result.

Of course, there are inpainting problems, as shown in Figures 1 (left) and 12(a), where it is not as easy to set up an adapted distance function. This is because

- the geometry of the image is harder to understand,
- the damaged region is complicated.

Moreover, if the damaged region Ω consists of many connected components and one wants to use our distance by harmonic interpolation technique (section 4.1), one must prescribe arcs Γ_h^k for every single component of connectivity; for the example of Figure 12(a) this means for

(a) original image (courtesy of Sun et al. [12, Figure 1], from http://www.pbase.com)

(b) masked image

(c) with contours of d_h

(d) inpainted image, CPU-time 0.3 sec

(e) masked image, red Γ_h

(f) with contours of T_h

(g) inpainted image, CPU-time 0.4 sec

Figure 13. *Remove the pumpkin from the original image* (a). *In the second row we have* (b) *the masked image with the inpainting domain* Ω_h *in white,* (c) *the masked image with contours of the Euclidean distance to boundary map* d_h*, and* (d) *the result inpainted using* d_h*. In the third row we have* (e) *the same masked image with a specified stop set* Γ_h *in red,* (f) *the masked image with contours of the distance by harmonic interpolation* T_h*, and* (g) *the result inpainted using* T_h*. Parameters in both cases:* $[\varepsilon, \mu, \sigma, \rho] = [4, 15, 0.7, 4]$*. (Image size:* 334×473*, damage:* 4%*.)*

every single letter. This can be tedious.

In contrast, the distance to boundary map can be computed quickly and easily for every type of inpainting domain. And, if the damage is such that color lines have been broken by scratches (by scratches we mean rather narrow and lengthy damages), the skeleton of Ω_h, being a simplified version of the scratch, is well placed. Thus, the filling-in process using distance to boundary makes the matching opposite sides come together and produces results of high quality (see Figures 1 (right) and 12(b)).

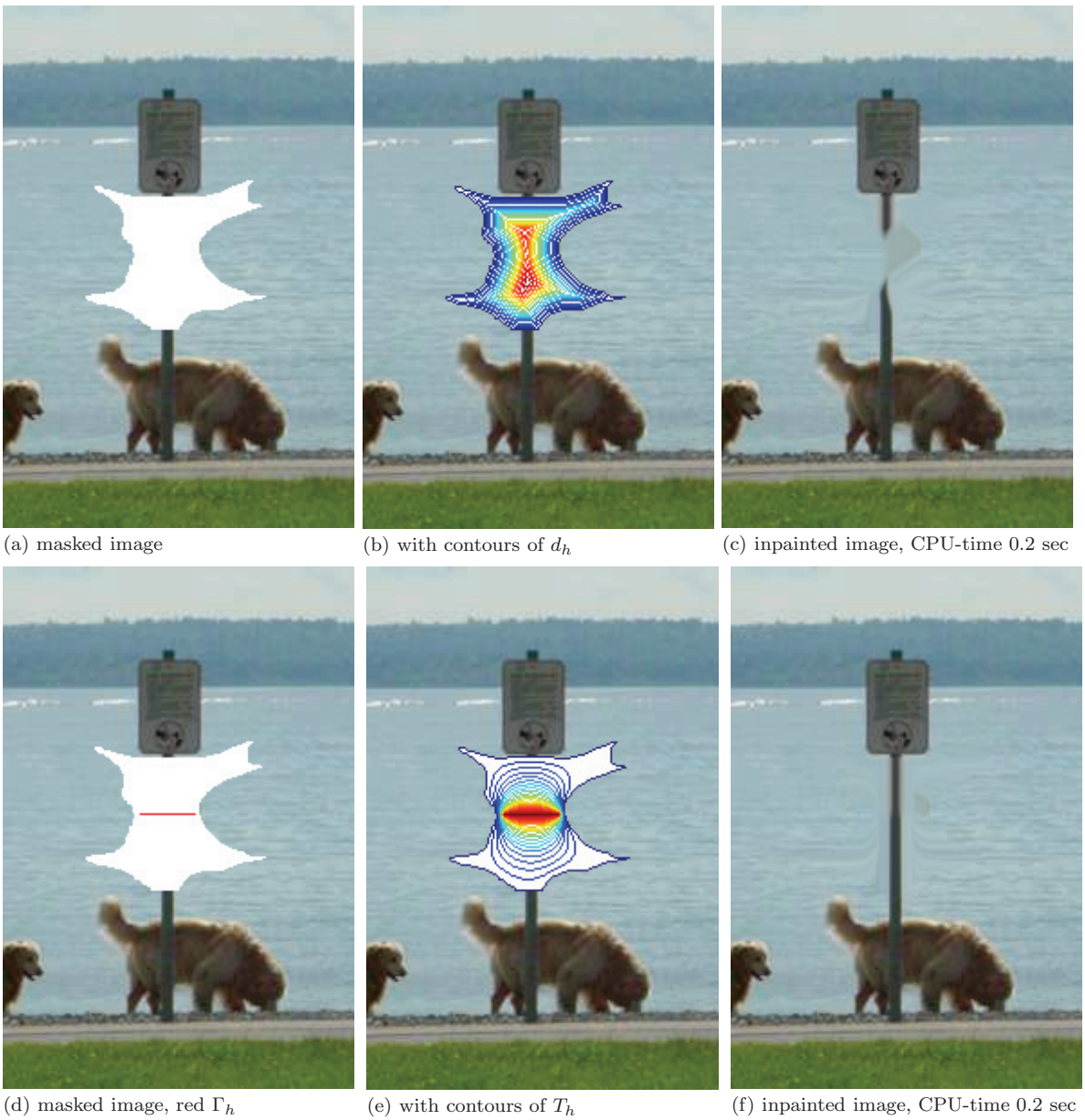
There are, however, also natural inpainting examples with a simple inpainting domain where we can use the ideas presented in the previous section. Figure 13 (taken from [12]) and Figure 14 (taken from [6]) show two such examples.

(a) original image

(b) masked image

Figure 14. *Task: remove the lower sign from the original image. Both images from* [6, Figure 11f] (© 2004 *IEEE. Reprinted, with permission, from 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)*. Results shown in Figures* 15–17*. (Image size:* 290 × 195*, damage:* 7%*.)*

Our method is PDE-based and geometric, and thus it does not perform very well on textures in contrast to the patch-based methods of Sun et al. [12] and Criminisi, Pérez, and Toyama [6]. Therefore, we have chosen examples where the image data is geometry dominated. The second row of Figure 13 shows the unfavorable result of the distance to boundary serialization, while in the third row we have a nice result from using distance by harmonic interpolation for serializing the pixels. The special choice of the stop set helps to restore the window cross. Similarly (for example, Figure 14), we can help the inpainter to restore the pole, again employing the distance by harmonic interpolation serialization; see Figure 15 (second row) and Figure 16. Criminisi, Pérez, and Toyama [6] have already reported on the problem of the distance to boundary map where only the shape of the inpainting domain determines the pixel serialization. In this case our method as well as Criminisi's cannot restore the pole; see the first row of Figure 15 and compare to [6, Figure 11f]. Criminisi, Pérez, and Toyama manage this problem with a "structure-first" fill order. Our modified distance to boundary approach (section 4.2) is in a similar vein: the structure of the surrounding data image determines the active boundary. Figure 17 shows the result of using this technique.



(a) masked image (b) with contours of d_h (c) inpainted image, CPU-time 0.2 sec

(d) masked image, red Γ_h (e) with contours of T_h (f) inpainted image, CPU-time 0.2 sec

Figure 15. *Remove the lower sign from the original image shown in Figure* 14(a)*. In the first row we have* (a) *the masked image with the inpainting domain* Ω_h *in white,* (b) *the masked image with contours of the Euclidean distance to boundary map* d_h*, and* (c) *the result inpainted using* d_h*. In the second row we have* (d) *the same masked image with a horizontal straight line as stop set* Γ_h *in red,* (e) *the masked image with contours of the distance by harmonic interpolation* T_h*, and* (f) *the result inpainted using* T_h*. Parameters in both cases:* $[\varepsilon, \mu, \sigma, \rho] = [7, 25, 0.7, 3]$*. (Image size:* 290×195*, damage:* 7%*.)*

(a) masked image, red Γ_h (b) with contours of T_h (c) inpainted image, CPU-time 0.2 sec

Figure 16. *Remove the lower sign from the original image shown in Figure* 14(a). (a) *the masked image with the inpainting domain* Ω_h *in white and with a vertical straight line in the upper part as stop set* Γ_h *in red,* (b) *the masked image with contours of the distance by harmonic interpolation* T_h*, and* (c) *the result inpainted using* T_h*. Parameters in all three cases:* $[\varepsilon, \mu, \sigma, \rho] = [7, 25, 0.7, 3]$*. (Image size:* 290×195*, damage:* 7%*.)*

(a) masked image (b) with contours of d_h (c) inpainted image, CPU-time 0.2 sec

Figure 17. *Remove the lower sign from the original image shown in Figure* 14(a). (a) *the masked image with the inpainting domain* Ω_h *in white and the active part of the boundary in red (short straight line at the bottom of* Ω_h*),* (b) *the masked image with contours of the modified distance to boundary map* $d_{*,h}$*, and* (c) *the result inpainted using* $d_{*,h}$*. Parameters:* $[\varepsilon, \mu, \sigma, \rho, \gamma] = [7, 25, 1.25, 5, 0.9]$*. (Image size:* 290×195*, damage:* 7%*.)*

Acknowledgment. The author would like to thank Folkmar Bornemann for his advice and the inspiring discussions.

REFERENCES

[1] T. Aach, C. Mota, I. Stuke, M. Mühlich, and E. Barth, *Analysis of superimposed oriented patterns*, IEEE Trans. Image Process., 15 (2006), pp. 3690–3700.
[2] L. Alvarez, F. Guichard, P.-L. Lions, and J.-M. Morel, *Axioms and fundamental equations of image processing*, Arch. Rational Mech. Anal., 123 (1993), pp. 199–257.
[3] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, *PatchMatch: A randomized correspondence algorithm for structural image editing*, ACM Trans. Graph., 28 (2009), 24.
[4] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, *Image inpainting*, in Proceedings of the 27th Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00), New Orleans, LA, ACM Press/Addison-Wesley, New York, 2000, pp. 417–424.
[5] F. Bornemann and T. März, *Fast image inpainting based on coherence transport*, J. Math. Imaging Vision, 28 (2007), pp. 259–278.
[6] 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.
[7] R. Kimmel, *Numerical Geometry of Images*, Springer-Verlag, New York, 2004.
[8] T. März, *First Order Quasi-Linear PDEs with BV Boundary Data and Applications to Image Inpainting*, Logos Verlag, Berlin, 2010.
[9] S. Masnou, *Disocclusion: A variational approach using level lines*, IEEE Trans. Image Process., 11 (2002), pp. 68–76.
[10] S. Masnou and J. Morel, *Level lines based disocclusion*, in Proceedings of the 1998 International Conference on Image Processing (ICIP '98), Chicago, IL, IEEE Press, New York, 1998, pp. 259–263.
[11] J. Sethian, *Level Set Methods and Fast Marching Methods*, 2nd ed., Cambridge University Press, Cambridge, UK, 1999.
[12] J. Sun, L. Yuan, J. Jia, and H.-Y. Shum, *Image completion with structure propagation*, ACM Trans. Graph., 24 (2005), pp. 861–868.
[13] A. Telea, *An image inpainting technique based on the fast marching method*, J. Graphics Tools, 9 (2004), pp. 23–34.
[14] D. Tschumperle, *Fast anisotropic smoothing of multi-valued images using curvature-preserving PDE's*, Int. J. Comput. Vision, 68 (2006), pp. 65–82.
[15] J. Weickert, *Anisotropic Diffusion in Image Processing*, B. G. Teubner, Stuttgart, Germany, 1998.