

An Image Inpainting Technique Based on the Fast Marching Method

Alexandru Telea
Eindhoven University of Technology

Abstract. Digital inpainting provides a means for reconstruction of small damaged portions of an image. Although the inpainting basics are straightforward, most inpainting techniques published in the literature are complex to understand and implement. We present here a new algorithm for digital inpainting based on the fast marching method for level set applications. Our algorithm is very simple to implement, fast, and produces nearly identical results to more complex, and usually slower, known methods. Source code is available online.

1. Introduction

Digital inpainting, the technique of reconstructing small damaged portions of an image, has received considerable attention in recent years. Digital inpainting serves a wide range of applications, such as removing text and logos from still images or videos, reconstructing scans of deteriorated images by removing scratches or stains, or creating artistic effects. Most inpainting methods work as follows. First, the image regions to be inpainted are selected, usually manually. Next, color information is propagated inward from the region boundaries, i.e., the known image information is used to fill in the missing areas. In order to produce a perceptually plausible reconstruction, an inpainting technique should attempt to continue the *isophotes* (lines of equal gray value) as smoothly as possible inside the reconstructed region. In other words, the missing region should be inpainted so that the inpainted gray value and gradient extrapolate the gray value and gradient outside this region.

Several inpainting methods are based on the above ideas. In [Bertalmio 00, Bertalmio 01], the image smoothness information, estimated by the image Laplacian, is propagated along the isophotes directions, estimated by the image gradient rotated 90 degrees. The *Total Variational* (TV) model [Chan and Shen 00a] uses an Euler-Lagrange equation coupled with anisotropic diffusion to maintain the isophotes' directions. The *Curvature-Driven Diffusion* (CCD) model [Chan and Shen 00b] enhances the TV method to drive diffusion along the isophotes' directions and thus allows inpainting of thicker regions. All above methods essentially solve a Partial Differential Equation (PDE) that describes the color propagation inside the missing region, subject to various heuristics that attempt to preserve the isophotes' directions. Preserving isophotes is, however desirable, never perfectly attained in practice. The main problem is that both isophote estimation and information propagation are subject to numerical diffusion. Diffusion is desirable as it stabilizes the PDEs to be solved, but leads inevitably to a certain amount of blurring of the inpainted area.

A second type of methods [Oliveira 01] repeatedly convolves a simple 3×3 filter over the missing regions to diffuse known image information to the missing pixels.

However impressive, the above methods have several drawbacks that preclude their use in practice. The PDE-based methods require implementing nontrivial iterative numerical methods and techniques, such as anisotropic diffusion and multiresolution schemes [Bertalmio 00]. Little or no information is given on practical implementation details such as various thresholds or discretization methods, although some steps are mentioned as numerically unstable. Moreover, such methods are quite slow, e.g., a few minutes for the relatively small inpainting region shown in Figure 1. In contrast, the convolution-based method described in [Oliveira 01] is fast and simple to implement. However, this method has no provisions for preserving the isophotes' directions. High-gradient image areas must be selected manually before inpainting and treated separately so as not to be blurred.

We propose a new inpainting algorithm based on propagating an image smoothness estimator along the image gradient, similar to [Bertalmio 00]. We estimate the image smoothness as a weighted average over a known image neighborhood of the pixel to inpaint. We treat the missing regions as level sets and use the fast marching method (FMM) described in [Sethian 96] to propagate the image information. Our approach has several advantages:

- it is very simple to implement (the complete pseudocode is given here);
- it is considerably faster than other inpainting methods—processing an 800×600 image (Figure 1) takes under three seconds on a 800 MHz PC;
- it produces very similar results as compared to the other methods;



Figure 1. An 800×600 image inpainted in less than three seconds.

- it can easily be customized to use different local inpainting strategies.

In Section 2, we describe our method. Section 3 presents several results, details our method’s advantages and limitations in comparison to other methods, and discusses possible enhancements. Source code of a sample method implementation is available online at the address listed at the end of the paper.

2. Our Method

This section describes our inpainting method. First, we introduce the mathematical model on which we base our inpainting (Section 2.1). Next, we describe how the missing regions are inpainted using the FMM (Section 2.2). Finally, we detail the implementation of inpainting one point on the missing region’s boundary (Section 2.3).

2.1. Mathematical Model

To explain our method, consider Figure 2, in which one must inpaint the point p situated on the boundary $\partial\Omega$ of the region to inpaint Ω . Take a small neighborhood $B_\varepsilon(p)$ of size ε of the known image around p (Figure 2(a)). As described in [Bertalmio 00, Oliveira 01, Chan and Shen 00a], the inpainting of p should be determined by the values of the known image points close to p , i.e., in $B_\varepsilon(p)$. We first consider gray value images, color images being a natural extension (see Section 2.4). For ε small enough, we consider a first order approximation $I_q(p)$ of the image in point p , given the image $I(q)$ and gradient $\nabla I(q)$ values of point q (Figure 2(b)):

$$I_q(p) = I(q) + \nabla I(q)(p - q). \quad (1)$$

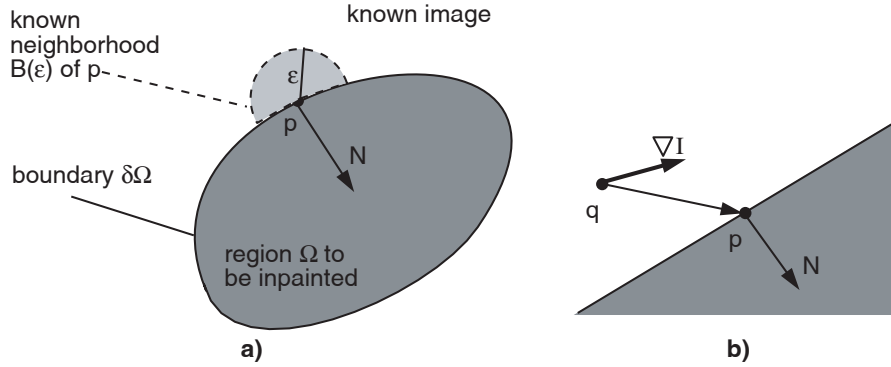


Figure 2. The inpainting principle.

Next, we inpaint point p as a function of all points q in $B_\epsilon(p)$ by summing the estimates of all points q , weighted by a normalized weighting function $w(p, q)$:

$$I(p) = \frac{\sum_{q \in B_\epsilon(p)} w(p, q) [I(q) + \nabla I(q)(p - q)]}{\sum_{q \in B_\epsilon(p)} w(p, q)}. \quad (2)$$

The weighting function $w(p, q)$, detailed in Section 2.3, is designed such that the inpainting of p propagates the gray value as well as the sharp details of the image over $B_\epsilon(p)$.

2.2. Adding Inpainting to the FMM

Section 2.1 explained how to inpaint a point on the unknown region's boundary $\partial\Omega$ as a function of known image pixels only. To inpaint the whole Ω , we iteratively apply Equation 2 to all the discrete pixels of $\partial\Omega$, in increasing distance from $\partial\Omega$'s initial position $\partial\Omega_i$, and advance the boundary inside Ω until the whole region has been inpainted (see pseudocode in Figure 3). Inpainting points in increasing distance order from $\partial\Omega_i$ ensures that areas closest to known image points are filled in first, thus mimicking manual inpainting techniques [Bertalmio 00, Bertalmio 01].

Implementing the above requires a method that propagates $\partial\Omega$ into Ω by advancing the pixels of $\partial\Omega$ in order of their distance to the initial boundary $\partial\Omega_i$. For this, we use the fast marching method. In brief, the FMM is an algorithm that solves the Eikonal equation:

$$|\nabla T| = 1 \quad \text{on } \Omega, \quad \text{with } T = 0 \text{ on } \partial\Omega. \quad (3)$$

The solution T of Equation 3 is the distance map of the Ω pixels to the boundary $\partial\Omega$. The level sets, or isolines, of T are exactly the successive

```

 $\delta\Omega_i$  = boundary of region to inpaint
 $\delta\Omega = \delta\Omega_i$ 
while ( $\delta\Omega$  not empty)
{
  p = pixel of  $\delta\Omega$  closest to  $\delta\Omega_i$ 
  inpaint p using Eqn.2
  advance  $\delta\Omega$  into  $\Omega$ 
}

```

Figure 3. Inpainting algorithm.

boundaries $\partial\Omega$ of the shrinking Ω that we need for inpainting. The normal N to $\partial\Omega$, also needed for inpainting, is exactly ∇T . The FMM guarantees that pixels of $\partial\Omega$ are always processed in increasing order of their distance-to-boundary T [Sethian 99], i.e., that we always inpaint the closest pixels to the known image area first.

We prefer the FMM over other Distance Transform (DT) methods that compute the distance map T to a boundary $\partial\Omega$ (e.g., [Borgefors 84, Borgefors 86, Meijster et al. 00]). The FMM's main advantage is that it *explicitly* maintains the narrow band that separates the known from the unknown image area *and* specifies which is the next pixel to inpaint. Other DT methods compute the distance map T but do not maintain an explicit narrow band. Adding a narrow band structure to these methods would complicate their implementation, whereas the FMM provides this structure by default.

To explain our use of the FMM in detail—and since the FMM is not straightforward to implement from the reference literature [Sethian 96, Sethian 99]—we provide next its complete pseudocode. The FMM maintains a so-called *narrow band* of pixels, which is exactly our inpainting boundary $\partial\Omega$. For every image pixel, we store its value T , its image gray value I (both represented as floating-point values), and a flag f that may have three values:

- *BAND*: the pixel belongs to the narrow band. Its T value undergoes update.
- *KNOWN*: the pixel is outside $\partial\Omega$, in the known image area. Its T and I values are known.
- *INSIDE*: the pixel is inside $\partial\Omega$, in the region to inpaint. Its T and I values are not yet known.

The FMM has an initialization and propagation phase as follows. First, we set T to zero on and outside the boundary $\partial\Omega$ of the region to inpaint and to some large value (in practice 10^6) inside, and initialize f over the whole image as explained above. All *BAND* points are inserted in a heap

```

while (NarrowBand not empty)
{
  extract P(i,j) = head(NarrowBand);      /* STEP 1 */
  f(i,j) = KNOWN;
  for (k,l) in (i1,j),(i,j1),(i+1,j),(i,j+1)
  if (f(k,l)!=KNOWN)
  {
    if (f(k,l)==INSIDE)
    {
      f(k,l)=BAND;                        /* STEP 2 */
      inpaint(k,l);                       /* STEP 3 */
    }
    T(k,l) = min(solve(k1,l,k,l1),        /* STEP 4 */
                 solve(k+1,l,k,l1),
                 solve(k1,l,k,l+1),
                 solve(k+1,l,k,l+1));
    insert(k,l) in NarrowBand;           /* STEP 5 */
  }
}

float solve(int i1,int j1,int i2,int j2)
{
  float sol = 1.0e6;
  if (f(i1,j1)==KNOWN)
    if (f(i2,j2)==KNOWN)
    {
      float r = sqrt(2(T(i1,j1)T(i2,j2))*(T(i1,j1)T(i2,j2)));
      float s = (T(i1,j1)+T(i2,j2)r)/2;
      if (s>=T(i1,j1) && s>=T(i2,j2)) sol = s;
      else
        { s += r; if (s>=T(i1,j1) && s>=T(i2,j2)) sol = s; }
    }
  else sol = 1+T(i1,j1));
  else if (f(i2,j2)==KNOWN) sol = 1+T(i1,j2));
  return sol;
}

```

Figure 4. Fast marching method used for inpainting.

NarrowBand sorted in ascending order of their T values. Next, we propagate the T , f , and I values using the code shown in Figure 4. Step 1 extracts the $BAND$ point with the smallest T . Step 2 marches the boundary inward by adding new points to it. Step 3 performs the inpainting (see Section 2.3). Step 4 propagates the value T of point (i, j) to its neighbors (k, l) by solving

the finite difference discretization of Equation 3 given by

$$\max(D^{-x}T, -D^{+x}T, 0)^2 + \max(D^{-y}T, -D^{+y}T, 0)^2 = 1, \quad (4)$$

where $D^{-x}T(i, j) = T(i, j) - T(i - 1, j)$ and $D^{+x}T(i, j) = T(i + 1, j) - T(i, j)$ and similarly for y . Following the upwind idea of Sethian [Sethian 96], we solve Equation 4 for (k, l) 's four quadrants and retain the smallest solution. Finally, Step 5 (re)inserts (k, l) with its new T in the heap.

2.3. Inpainting One Point

We consider now how to inpaint a newly discovered point (k, l) , as a function of the *KNOWN* points around it, following the idea described in Section 2.1. (Step 3 in Figure 4, detailed in Figure 5). We iterate over the *KNOWN* points in the neighborhood B_ε of the current point (i, j) and compute $I(i, j)$ following Equation 2. The image gradient ∇I (`gradI` in the code) is estimated by central differences. As stated in Section 2.1, the design of the weighting function $w(p, q)$ is crucial to propagate the sharp image details and the smooth zones as such into the inpainted zone. We design $w(p, q) = \text{dir}(p, q) \cdot \text{dst}(p, q) \cdot \text{lev}(p, q)$

```
void inpaint(int i,int j)
{
  for (all (k,l) in Bε(i,j) such that f(k,l)!=OUTSIDE)
  {
    r = vector from (i,j) to (k,l);
    dir = r * gradT(i,j)/length(r);
    dst = 1/(length(r)*length(r));
    lev = 1/(1+fabs(T(k,l)-T(i,j)));
    w = dir*dst*lev;
    if (f(k+1,l)!=OUTSIDE && f(k-1,l)!=OUTSIDE &&
        f(k,l+1)!=OUTSIDE && f(k,l-1)!=OUTSIDE)
      gradI = (I(k+1,l)-I(k-1,l), I(k,l+1)-I(k,l-1));
    Ia += w * (I(k,l) + gradI * r);
    s += w;
  }
  I(i,j) = Ia/s;
}
```

Figure 5. Inpainting one point.

as a product of three factors:

$$\begin{aligned} dir(p, q) &= \frac{p - q}{\|p - q\|} \cdot N(p) \\ dst(p, q) &= \frac{d_0^2}{\|p - q\|^2} \\ lev(p, q) &= \frac{T_0}{1 + |T(p) - T(q)|}. \end{aligned}$$

The *directional* component $dir(p, q)$ ensures that the contribution of the pixels close to the normal direction $N = \nabla T$ (`gradT` in the code), i.e., close to the FMM's information propagation direction, is higher than for those farther from N . The *geometric distance* component $dst(p, q)$ decreases the contribution of the pixels geometrically farther from p . The *level set distance* component $lev(p, q)$ ensures that pixels close to the contour through p contribute more than farther pixels. Both dst and lev are relative with respect to the reference distances d_0 and T_0 . In practice, we set d_0 and T_0 to the interpixel distance, i.e., to 1. Overall, the above factors model the manual inpainting heuristics [Bertalmio 00] that describe how to paint a point by strokes bringing color from a small region around it.

For ε up to about six pixels, i.e., when inpainting thin regions, dst and lev have a weak effect. For thicker regions to inpaint, such as Figure 8(d), where we used an ε of 12 pixels, using dst and lev provides better results than using dir alone. The above is clearly visible in Figure 6, on a test image taken from [Bertalmio 00], where the missing ring-shaped region is more than 30 pixels thick. Figure 6(c) shows, on an image detail, the effect of dir alone. The results are somewhat less blurry when dir and dst (Figure 6(d)) or dir and lev (Figure 6(e)) are used together. The inpainting is the best visually when all three components are used (Figure 6(f)).

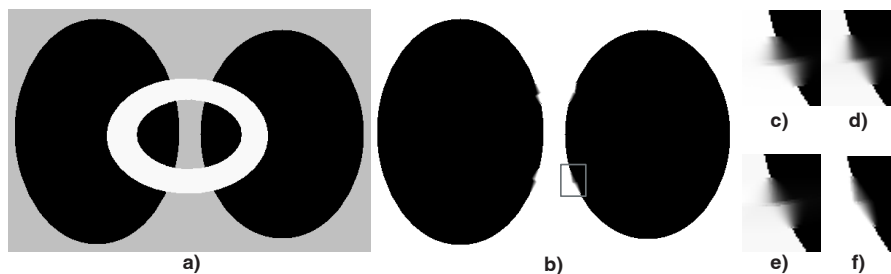


Figure 6. Thick region to inpaint (a) and result (b). Effect of weighting functions: direction (c), direction and geometric distance (d), direction and level set distance (e), direction, geometric, and level set distance (f).

2.4. Implementation Details

Several implementation details are important. First, we compute the boundary normal $N = \nabla T$ by numerical derivation of the field T computed by the FMM. Derivating T on the fly as it is computed by the FMM is unstable, since we are not guaranteed that a large enough neighborhood around the current point contains only *KNOWN* points. We first run the FMM outside the initial inpainting boundary $\partial\Omega$ and obtain the distance field T_{out} . Since we use only those points closer to $\partial\Omega$ than ε , we run the FMM outside $\partial\Omega$ only until we reach $T > \varepsilon$. This restricts the FMM computations to a band of thickness ε around $\partial\Omega$, thus speeding up the process. Next, we run the FMM inside $\partial\Omega$ and obtain T_{in} . The field T over the whole image is given by

$$T(p) = \begin{cases} T_{in}(p) & \text{if } p \in \Omega \\ -T_{out}(p) & \text{if } p \notin \Omega. \end{cases} \quad (5)$$

Next, we smooth T by a 3×3 tent filter, and then compute ∇T by central differences.

The value of ε giving the size of B usually ranges from three to ten pixels. This corresponds with the “thickness” of the regions to inpaint, which is usually less than 15 pixels. Higher values blur the sharp details to be reconstructed by inpainting, although they are useful when inpainting thicker regions.

The test `f(k,1)!=OUTSIDE` in Figure 5 that restricts B_ε to the known image points can be changed to `f(k,1)==KNOWN`. The results are visually identical, as B_ε contains very few *BAND* pixels. However, one would use the second test if the initial $\partial\Omega$ corresponds to unknown image pixels.

The `NarrowBand` sorted heap (Section 2.2) is straightforwardly implemented using the C++ STL `multimap` container [Musser and Saini 96]. Finally, for color (RGB) images, we apply the presented method separately for each color channel.

3. Discussion

We have compared our inpainting method with the methods presented by Bertalmio et al. in [Bertalmio 00] and Oliveira et al. in [Oliveira 01], further denoted by BSCB and OBMC, by running it on the same input images (see Figures 7 and 8(a)–(c)). For BSCB, we used the implementation publicly available at [Yung and Shankar ??], whereas we reimplemented OBMC ourselves. Our method produced visually nearly identical results with BSCB. Compare, for example, the inpaintings in Figure 7(c), (d) (Figure 7(e), (f) in detail) and Figure 8(b), (c) (Figure 8(g), (h) in detail). In contrast, the

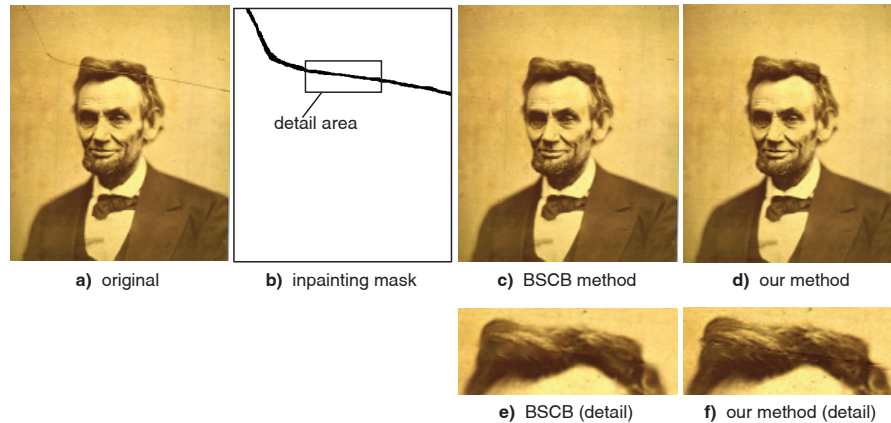


Figure 7. Lincoln “cracked photo” inpainting.

results of OBMC (shown in [Oliveira 01]) were visibly more blurry for regions thicker than six pixels. The runtime for our method was in all cases much shorter than for BSCB. Our C++ implementation took less than 3 seconds on an 800 MHz PC for a 800×600 color image with about 15% pixels to inpaint (Figure 1). On the same input, the original BSCB method, published in [Bertalmio 00], is reported to take less than 5 minutes on a 300 MHz PC. The BSCB implementation we used [Yung and Shankar ??], which is mentioned to be unoptimized by its authors, took between 2.5 and 3 minutes on the 800 MHz PC, depending on its various parameter settings. In contrast, OBMC takes in all cases about the same time as our method. The above matches the fact that both OBMC and our method are linear in the inpainted region’s size.

The main limitation of our method (applicable to the BSCB and OBMC methods mentioned here) is the blurring produced when inpainting regions thicker than 10–15 pixels, especially visible when sharp isophotes intersect the region’s boundary almost tangentially. See, for example, the inpainting in Figure 8(d), (e) (in detail in Figure 8(i), (j)). The above is caused by the linear and local character of our method. Techniques using an explicit nonlinear and/or global image model, such as the TV [Chan and Shen 00a] and CDD [Chan and Shen 00b] methods, achieve better results, at the cost of considerably more complex implementations.

Overall, the presented inpainting method is simple to implement (our complete C++ code is about 500 lines), fast, and easy to customize for different inpainting strategies. We plan to extend the method by developing new inpainting functions that are better able to preserve the isophotes’ directions. One such way is to integrate anisotropic diffusion, e.g., following [Bertalmio 00], in the FMM boundary evolution in order to reduce the blurring for inpainting

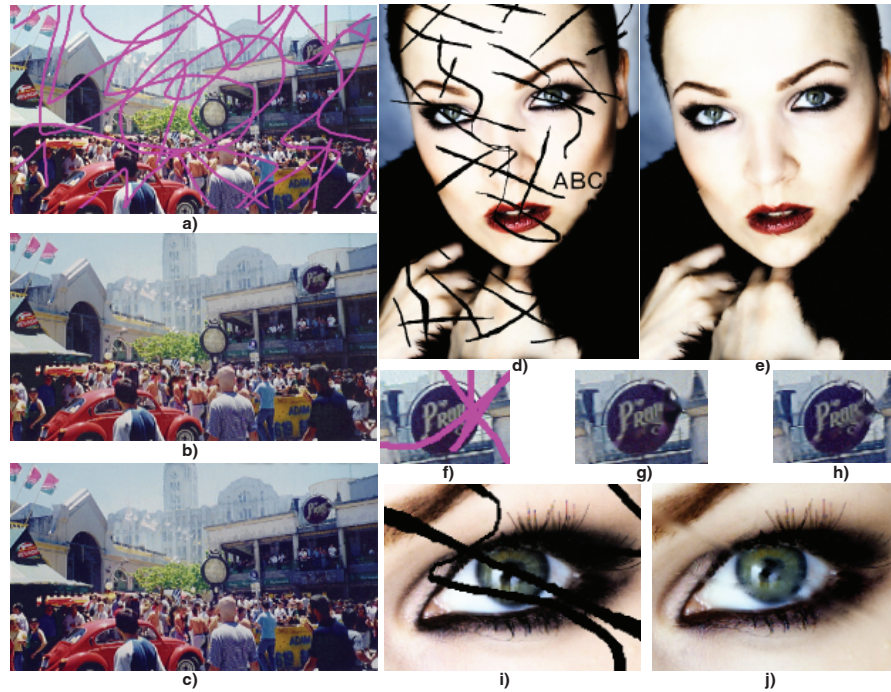


Figure 8. Inpainting examples. Damaged photo (a), inpainting by method BSCB (b), our method (c), and close-ups (f, g, h). Damaged photo (d), distance-weighted inpainting (e), and close-ups (i, j)

thick regions. A second extension would be to modulate the evolution speed of the FMM, now equal to 1, by the image anisotropy, i.e., let inpainting “work more” on the high detail areas than on the smooth regions.

Acknowledgments. We are indebted to Professor J. J. van Wijk from the Department of Mathematics and Computer Science of the Eindhoven University of Technology for his numerous suggestions for improving this paper.

References

- [Bertalmio 00] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. “Image Inpainting.” In *Proceedings SIGGRAPH 2000, Computer Graphics Proceedings, Annual Conference Series*, edited by Kurt Akeley, pp. 417–424, Reading, MA: Addison-Wesley, 2000.

- [Bertalmio 01] M. Bertalmio, A. L. Bertozzi, and G. Sapiro. “Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting.” In *Proc. ICCV 2001*, pp. 1335–1362, IEEE CS Press 1. [CITY]: [PUB], 2001.
- [Borgefors 84] G. Borgefors. “Distance Transformations in Arbitrary Images.” *Comp. Vision, Graphics, and Image Proc.* 27:3 (1984), 321–345.
- [Borgefors 86] G. Borgefors. “Distance Transformations in Digital Images.” *Comp. Vision, Graphics, and Image Proc.* 34:3 (1986), 344–371.
- [Oliveira 01] M. Oliveira, B. Bowen, R. McKenna, and Y. -S. Chang. “Fast Digital Image Inpainting.” In *Proc. VIIP 2001*, pp. 261–266, [CITY]: [PUB], 2001.
- [Chan and Shen 00a] T. Chan and J. Shen. “Mathematical Models for Local Deterministic Inpaintings.” Technical Report CAM 00-11, Image Processing Research Group, UCLA, 2000.
- [Chan and Shen 00b] T. Chan and J. Shen. “Non-Texture Inpainting by Curvature-Driven Diffusions (CDD).” Technical Report CAM 00-35, Image Processing Research Group, UCLA, 2000.
- [Meijster et al. 00] A. Meijster, J. Roerdink, and W. Hesselink. “A General Algorithm for Computing Distance Transforms in Linear Time.” In *Math. Morph. and Its Appls. to Image and Signal Proc.*, pp. 331–340, [CITY]: Kluwer, 2000.
- [Sethian 99] J. A. Sethian. *Level Set Methods and Fast Marching Methods*, Second edition. Cambridge, UK: Cambridge Univ. Press, 1999.
- [Sethian 96] J. A. Sethian. “A Fast Marching Level Set Method for Monotonically Advancing Fronts.” *Proc. Nat. Acad. Sci.* 93:4 (1996), 1591–1595.
- [Yung and Shankar ??] W. Yung and A. J. Shankar. “Image Inpainting Implementation Software.” Available from World Wide Web (<http://www.bantha.org/~aj/inpainting>), [YEAR].
- [Musser and Saini 96] D. R. Musser and A. Saini. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison-Wesley Professional Computing Series. Reading, MA: Addison-Wesley, 1996.

Web Information

Source code of a sample C++ implementation of the inpainting method described here is available at <http://www.acm.org/jgt/papers/Telea03.html>.

Alexandru Telea, Department of Mathematics and Computer Science, Eindhoven University of Technology, Den Dolech 2, Eindhoven 5600 MB, The Netherlands (alex@win.tue.nl, <http://www.win.tue.nl/~alex>)

Received October 24, 2002; accepted in revised form May 21, 2003.