IMPERIAL COLLEGE LONDON, DEPARTMENT OF MATHEMATICS

Scientific Computing (M3SC)

Peter J. Schmid

March 14, 2017

1 INPAINTING

Images are ubiquitous in the scientific and artistic area and are some of the oldest means of expressing information, recording historic or cultural events or displaying creative energy. The restoration of artwork is an ancient practice and has been used and perfected extensively over centuries. With the rise of digital photography and image processsing, ancient and modern techniques for graphics manipulation and repair have been encoded in particular algorithms. In this section, we will concentrate on image inpainting, a technique that attempts to restore and fill in damaged and otherwise missing parts of an image; removal of scratches, cracks, noise, stains are applications of inpainting, as are removal of stamped dates, watermarks, reflections, glare, red-eye effects or other artifacts.

Digital image inpainting requires the specification of an area of undesirable features which are to be removed and subsequently filled in based on the information available in the surrounding area of the image. This latter information is propagated (or diffused) into the removed part (referred to as the mask) until the gap is filled and the full image restored. Different algorithms and techniques are available depending on the type of information that is propagated or preserved; structure, texture, sharp gradients are among the image features that can be conserved or approximated during the process. The propagation of image information classifies the algorithms into PDE-based, convolution-based, texture-based, transform-based, etc. A broader classification rests on diffusion-based and non-diffusion-based methods. In the section below, we will explore two diffusion-based techniques: (i) isotropic diffusion and (ii) anisotropic diffusion. Both techniques will be implemented as convolution filters. The first technique applies an isotropic smoothing filter to fill in the

missing information, while the second technique, based on bilateral filters, takes into account sharp gradients and edges.
Digital image reconstruction started with the solution of partial differential equation, propagating image information into the damaged or missing area. Various types to image information are used, the most common one being the image Laplacian which represents a measure of image smoothness; the propagation direction is chosen along so-called isophotes which are perpendicular to contours of image gradients. Starting from the edge of the inpainting area, a front is then propagated into the missing domain, until the image is properly restored.

The earliest PDE-based approach is based on the heat equation

$$\frac{\partial I}{\partial t} = D\nabla^2 I, \qquad \mathbf{x} \in \Omega \tag{1.1}$$

which diffuses image information into the damaged area Ω. In this expression, $I(\mathbf{x})$ denotes the pixel value at location $\mathbf{x}$, and D stands for the diffusion coefficient (degree of smoothing). While simple to implement and sufficiently accurate for small cracks or smooth images, it has quickly been recognized that a more sophisticated approach is necessary. While clinging to the heat equation, an anisotropic diffusivity has been introduced which replaces the isotropic (direction-independent) propagation of information with a directed (anisotropic) propagation of information.

$$\frac{\partial I}{\partial t} = \nabla \cdot (a(I)\nabla I), \qquad \mathbf{x} \in \Omega \tag{1.2}$$

In this case, the image information does not diffuse isotropically or omnidirectionally into the damaged area; rather, the diffusivity coefficient $a(I)$ guides the diffusion in a manner that locally depends on the image information. Various functions for $a(I)$ are available, the most known is linked to the famous Perona-Malik equation with

$$a(I) = g\left(|\nabla I|^2\right) \qquad g(s) = \frac{1}{1+s}. \tag{1.3}$$

The local absolute image gradient thus controls the amount of diffusion into Ω. For numerical stability, the gradient has to be presmoothed, for example by a Gaussian filter. Additional constraints can be added to the propagation of image information, such as the preservation of total variation, the preservation of isophotes or identified edges.

Starting from the realization that a diffusive step can be modelled as a convolution with a (Gaussian) heat-kernel, the concept of inpainting by filtering has been born. In this approach, repeated application of a compactly supported filter window (for example, 3×3 pixels) over the missing area will diffuse information from the edge of the inpainting region to the missing pixels. These techniques are particularly attractive as they eliminate the need for the tedious (and slow) solution of complex partial differential equations; the gain in speed, however, has to be balanced by the fact that isophote directions are not preserved and high-contrast areas are excessively smoothed. Special conditions have to be implemented to account for edges, high-gradient areas, small details or particular image texture.

Independent of the applied algorithm, the image to be processed is taken as a two-dimensional numerical array of gray values, ranging from 0 (black) to 255 (white). When processing color images, the algorithm has to be applied to the R (red), G (green) and B (blue) subimages separately and then reassembled again, even though a more appropriate decomposition into luminance and two chromas is more advantageous and avoids color artifacts near edges. The damaged area to be filled is referred to as Ω and is a (small) subset of the full two-dimensional array. This subset Ω is defined by the user as a mask; the image outside this mask is left untouched by the algorithm, but will provide the starting values for filling in the damaged parts.

For large areas Ω and for images that contain a significant amount of complex texture, sophisticated algorithms have to brought to bear. For locally smaller (and thinner) regions Ω, however, simpler models can be used to restore image information from neighboring pixels.

2 Inpainting by isotropic filtering

For locally small areas Ω, we can fill the missing information, starting from the boundary $\partial\Omega$ of Ω, by an isotropic diffusion process that propagates pixel values from the edge into the area to be repaired. The diffusion process is equivalent to an iterative filtering process whereby the cleared pixels in Ω are updated using a nearest-neighbor stencil; this is equivalent to a localized convolution with a diffusion kernel. The repeated application of this convolution to all pixels in Ω results a steady state solution of an isotropic diffusion based on Dirichlet data on $\partial\Omega$.

For our application, we choose a 3×3-neighborhood and update the pixel at the center by isotropically averaging over its eight neighbors. This is equivalent to a convolution with a local-area filter; the corresponding stencil is sketched in figure 3.1(a). Repeated application of the filter, until a steady state is reached, will restore the missing area of the image.

A minor modification of the stencil, accounting for the varying distance of the neighboring pixels from the central pixel yields slightly different filter weights (see figure 3.1(b)). This modification removes artificial grid effects and more accurately represents a radially symmetric propagation/diffusion of information.

3 Inpainting by anisotropic bilateral filtering

The diffusion of image information by the above filters is highly dissipative, in the sense that sharp edges are increasingly smoothed out and eventually lost. This effect is the consequence of the filter's concentration on spatial information (while neglecting pixel information). In other words, only the distance to the central pixel sets the weight of the filter; the pixel value itself does not enter the final filter weight. It thus should not come as a surprise, that edges (i.e., large jumps in pixel values) are ignored and ultimately dissipated. To

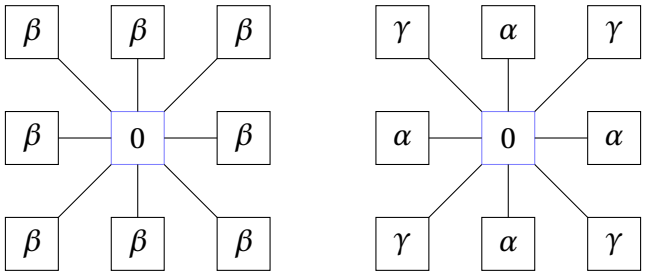

Figure 3.1: Filter weights for isotropic and anisotropic convolution. For the isotropic case (a), we have $\beta = 1/8$; for the anisotropic case (b), we have $\alpha = 0.176765$ and $\gamma = 0.073235$.

counterbalance this tendency, a bilateral filter is introduced: it consists of two parts, (i) a smoothing spatial filter (similar to the one above) and (ii) an edge-preserving filter.

In its simplest representation we take two Gaussian filters for each of the components and write

$$w_1(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{2}\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma_s^2}\right), \tag{3.1a}$$

$$w_2(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{2}\frac{|I(\mathbf{x})-I(\mathbf{y})|^2}{\sigma_r^2}\right). \tag{3.1b}$$

The first weight function w_1 depends on the spatial distance of the pixels and is responsible for pure diffusion. The second weight function w_2 takes into consideration the pixel values I between the two considered points $\mathbf{x}$ (the central position) and $\mathbf{y}$ (the neighborhood position). For a large jump in I between $\mathbf{x}$ and $\mathbf{y}$, the Gaussian in w_2 will be exceedingly small. The two standard deviations σ_s and σ_r will determine the filter width of each component.

The full convolutional filtering then reads

$$I(\mathbf{x}) \leftarrow \frac{1}{w(\mathbf{x},\mathbf{y})}\sum_{\mathbf{y}\in\mathcal{N}} w_1(\mathbf{x},\mathbf{y})\,w_2(\mathbf{x},\mathbf{y})\,I(\mathbf{y}), \tag{3.2a}$$

$$w(\mathbf{x},\mathbf{y}) = \sum_{\mathbf{y}\in\mathcal{N}} w_1(\mathbf{x},\mathbf{y})\,w_2(\mathbf{x},\mathbf{y}). \tag{3.2b}$$

In the above expression, $\mathcal{N}$ is the neighborhood considered in the filter windows. For thin cracks, the window size typically contains the crack and connects known information from both sides of the crack.

4 Implementation

The code below implements the inpainting algorithm above. We import an image that has been vandalized by graffiti. We wish to restore the vandalized image to its original glory. We

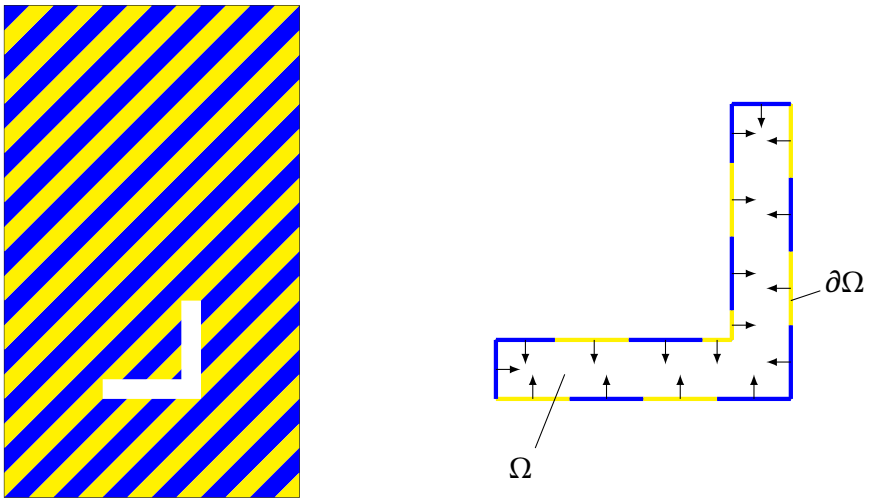

Figure 3.2: Concept of inpainting. A damaged or missing area is identified using a (user-defined) mask (a), after which the information on the edge is propagated into the interior of the domain (b). The propagation ranges from simple diffusion to complex information-propagation along preferred direction. Additional constraints for the propagation, to preserve image features and texture, can be imposed as well.

define a mask that identifies the part of the vandalized image that we want to remove. Once this part is flagged, the pixels in this region are set to 255 (white) and subsequently filled in from the known edges. When applying the convolutional, bilateral filter it is important to consider efficiency issues. While it may be tempting to loop over all pixels in the damaged region, it is far more efficient (you are encouraged to explore the difference) to first identify the indices of the missing pixels and then process them as vectors. This vectorized version of processing the damaged region is far superior in speed than a for-loop. This kind of consideration should be made not only in this example, but in general.

```
1  from PIL import Image
2  from scipy.misc import imsave
3  import numpy as np
4
5  def inpaint(imarr,mask,niter):
6      # eliminate the part of the image covered by the mask
7      irow,jcol = np.where(mask==1)
8      imarr[irow,jcol] = 255
9
10     # embed in a bigger frame
11     n,m = imarr.shape
12     imARR = np.zeros((n+2,m+2))
13     imARR[1:-1,1:-1] = imarr
14
```

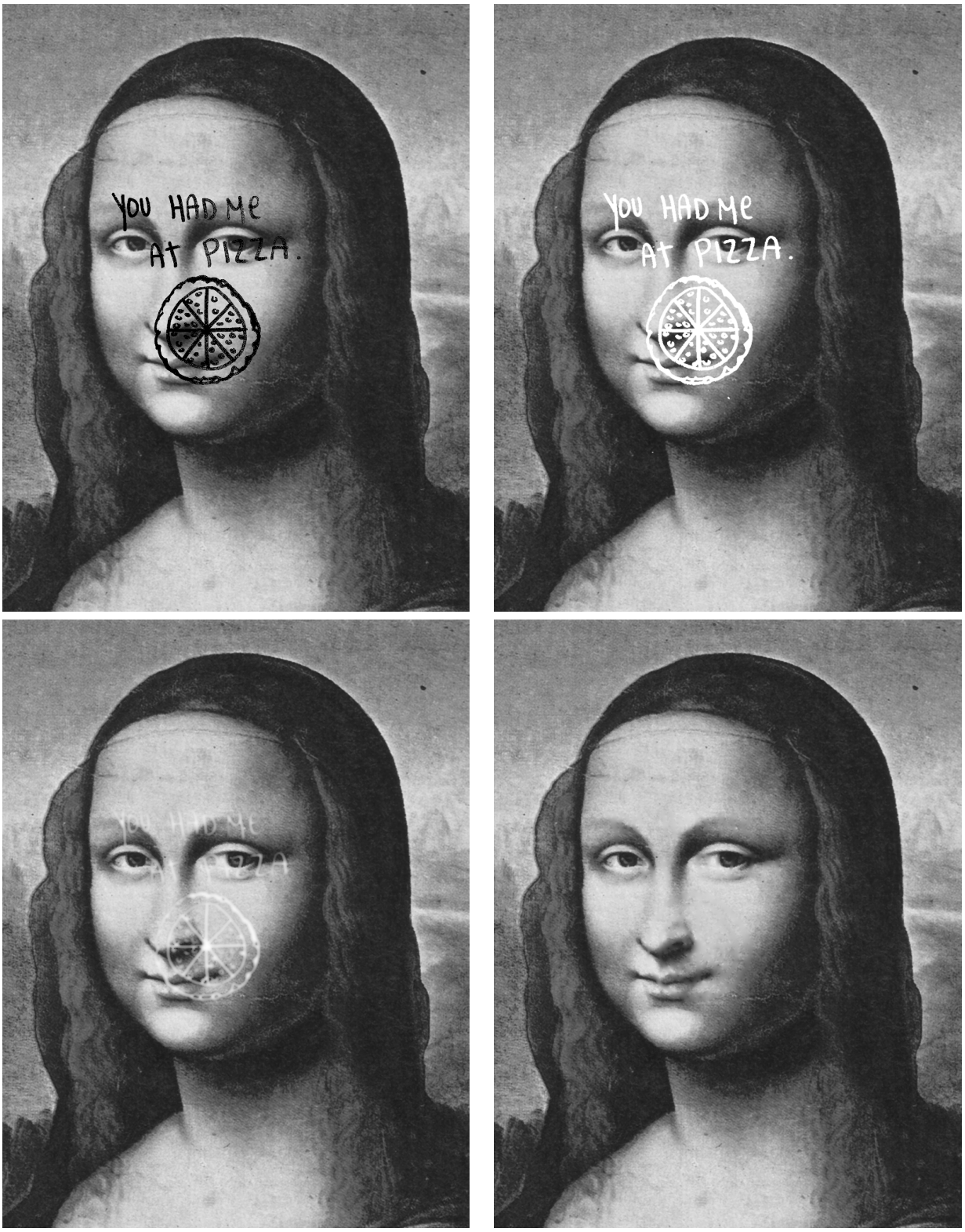

Figure 3.3: Restoration of a vandalized painting by image inpainting. (upper left) vandalized image, with black graffiti; (upper right) the part to be eliminated has been flagged (in white) and defined as a mask Ω; (lower left) work in progress, after 30 iterations of the inpainting algorithm; (lower right) restored painting, after 300 iterations of the inpainting algorithm.

```
    # Neumann extensions
    imARR[0,1:-1]  = imarr[0,:]
    imARR[-1,1:-1] = imarr[-1,:]
    imARR[1:-1,0]  = imarr[:,0]
    imARR[1:-1,-1] = imarr[:,-1]

    # adjust indices from mask (due to embedding)
    ir = irow+1
    jc = jcol+1

    # set the parameters
    iwid   = 1
    sigma2 = 1.
    alpha  = 250.

    # start the inpainting loop
    for i in range(niter):
        if (i%100 == 0):
            print 'iteration = ',i,' out of ',niter
        dARR = np.zeros_like(imARR)
        ww   = np.zeros(len(ir))
        for id in range(-iwid,iwid+1):
            for jd in range(-iwid,iwid+1):
                if ((id==0) and (jd==0)):
                    ee = np.zeros(len(ir))
                    EE = np.zeros(len(ir))
                else:
                    # bilateral filter (Gaussian)
                    dd = np.sqrt(id*id + jd*jd)
                    ee = np.exp(-dd*dd/sigma2)
                    # bilateral filter (gradients)
                    xk = (imARR[ir+id,jc+jd]-imARR[ir,jc])/dd
                    EE = np.exp(-xk*xk/alpha/alpha)
                # combine the two filters
                dARR[ir,jc] += ee*EE*imARR[ir+id,jc+jd]
                ww          += ee*EE
        imARR[ir,jc] = dARR[ir,jc]/ww

    # take off boundary rows
    imarr = imARR[1:-1,1:-1]

    # convert back to image and return
    im_inp = Image.fromarray(imarr.astype(np.uint8))
    return im_inp
```

```
59
60 def convert2BW(img_in):
61     # open the image file
62     im_file = Image.open(img_in)
63
64     # convert image to monochrome
65     im_BW = im_file.convert('L')
66
67     # convert to num-array
68     im_array = np.array(im_BW)
69
70     # convert back to image (for later use)
71     im_file2 = Image.fromarray(im_array)
72
73     # show image (for later use)
74     # im_file2.show()
75     return im_array
```

The main code is given below. It imports the image and the graffiti, overlays the two images, then defines a mask and calls the inpainting algorithm to restore the vandalized image.

```
1  if __name__ == '__main__':
2
3      # convert an image to an "black-and-white" array
4      aa = convert2BW('MonaLisa2.jpg')
5      a  = aa[0:1300,500:1500]
6
7      # convert overlay to an "black-and-white" array
8      bb    = convert2BW('Pizza.jpg')
9      nb,mb = bb.shape
10     for i in range(0,nb):
11         for j in range(0,mb):
12             if (bb[i,j]<100):
13                 bb[i,j] = 0
14     b       = bb.copy()
15     c       = 255*np.ones_like(a)
16     (nb,mb) = b.shape
17     dx      = 210
18     dy      = 180
19     c[dx:dx+nb,dy:dy+mb] = b
20     f     = np.minimum(a,c)
21     ma    = (c<=220)
22     mask  = ma.astype(np.int)
23
```

```
24      niter = 300
25      # convert back to image (for later use)
26      im_f  = Image.fromarray(f.astype(np.uint8))
27      im_f.show()
28      im_f.save('vandalized.png')
29
30      # inpainting
31      img_pp = inpaint(f,mask,niter)
32
33      # display image
34      img_pp.show()
35      img_pp.save('inpainted.png')
```

After 300 iterations of the convolutional, bilateral filter, the image is sufficiently restored (see figure 3.3). While this image has not seriously challenged our bilateral filter in terms of preservation of gradients and edges, it has shown efficiency and efficacy in recovering a moderately damaged painting by inpainting.