AM205 Final Paper

Justin L. Xie, Max Guo

December 2020

Abstract

Inpainting is the process of filling in an occlusion in an image. In this paper, we examine the efficacy of three different methods for inpainting: heat equation based diffusion, anisotropic diffusion, and isophote propagation. We examine their efficacy under homogenous backgrounds and under harsh borders; we also benchmark our results against *cv2*, an existing Python library implementing inpainting via Navier Stokes and the Fast Marching Method. We discuss a few other attempted implementations and discuss their shortcomings. Ideally, an inpainting algorithm would smoothly reconstruct the occlusion under homogenous backgrounds and preserve borders when those are present. We find that anisotropic diffusion described by Perona-Malik performs best under harsh border conditions, with all three methods performing nearly identically under homogenous background conditions. However, we examine efficacy by evaluating MSE, but MSE based evaluations may not translate into the "best" visual reconstruction. In the future, we hope to find a better metric for evaluating the efficacy of our implementations, potentially utilizing broad surveys of individuals who rank the reconstructions from best to worst. We also wish to generalize our methods and potentially find a better composite method for inpainting.

1 Introduction

For centuries, art restorers have dedicated their lives to the inpainting problem. Today, computational strategies to execute this inpainting task has become a blossoming area of academic study. Image inpainting is the problem of filling in a portion of an image or a video using the surrounding pixels. This is often done to reconstruct a damaged photo or video, or it can be used to remove an occluding object from a scene. This is a traditionally classified as a branch within computer vision that has been studied through a multitude of different approaches. Image inpainting is still an active area of research today, especially due to recently developed methods that involve deep learning and generative models [1].

Formally, the problem can be formulated as follows. Given some region D representing an image, there exists an occlusion, an area of damage, $\Omega \subset D$ such that $D \setminus \Omega$ has pixels of known values. In particular, we have a matrix I representing our color intensities for the image across the region D. Using the information in $D \setminus \Omega$, we seek to propagate information into the occlusion Ω, generally starting from the $\partial\Omega$, the border. Ideally, we can inpaint Ω with a solution I^* such that the I^* is visually indistinguishable from the rest of I. Finding the best I^* is the inpainting problem [2]

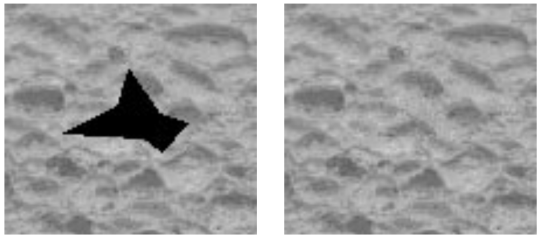
We note that there is a slight difference between image inpainting and image restoration. The latter may involve denoising or uncompressing an entire image, whereas the former only seeks to reconstruct a portion of the image. Moreover, in image inpainting we do not gain any information from the initial region Ω that is to be reconstructed, only from the surrounding neighborhood of the region. For all intents and purposes, we may assume that each pixel in the inpainted region is initially blank. However, in image restoration the initial value of each pixel to be restored may play a crucial role in reconstructing the image.

In this paper, we seek to conduct a survey of image inpainting methods inspired by the AM205 material. We implemented three inpainting methods, a Heat Equation/diffusion-based method, an anisotropic diffusion method that preserves borders, and an isophote propagation method (a name that we imposed onto Bertalmio's method). We begin with a discussion of existing literature and classes of inpainting methods.

Traditional Inpainting can largely be classified into two distinct classes of methods: Geometric Methods and Texture Synthesis Methods. Geometric methods take heavy inspiration from fluid dynamics and usually involve a diffusion or PDE approach, relying on convergence to a steady state. As a result, geometric methods are iterative in nature and can be time consuming as a result. These methods focus on inpainting structural properties of the image rather than more granular texture effects. In contrast, texture analysis methods involve using local texture properties to restore/fill a hole in textured images. Texture analysis is not as effective in the reconstruction/inpainting of structured backgrounds in natural images that are not texture based. Recent literature has analyzed the usage of both texture and geometric methods in conjunction for image inpainting [5]. We focus our discussion and methods on Geometric methods for inpainting natural images rather than texture synthesis for the remainder of the paper.

1.1 Geometric Methods

Geometric methods focus on image inpainting under any backgrounds; generally, they rely on information from neighboring pixels, propagating information into the hole/area of damage. Geometric methods



(a) Texture Synthesis Inpainting [3]

(b) Geometric Inpainting [4]

for image inpainting are largely iterative in nature, and PDEs are a primary source of algorithms for this technique. Notably, geometric methods oftentimes lose texture level granularity in favor of effectively preserving high level structural qualities.

1.1.1 Bertalmio et al [6]

A seminal paper in the field is *Image Inpainting* by Bertalmio et al. (2000), and this paper introduced one of the earlier PDE solutions to inpainting. The algorithm attempted to recreate the process taken by manual painters who repaired old paintings: start from the border and work inwards, using the non-damaged area to inform where lines should be drawn. Bertalmio et al.'s algorithm attempts to preserve continuity and second derivative continuity of the spatial image in its restoration. We'll use I^n to denote the color function at iteration n, and I_t^n denotes the change for a given unit of time such that $I^{n+1} = I^n + \Delta t I_t^{n+1}$. Δt is chosen experimentally; a value too large and the algorithm may "overshoot" the convergent solution. The algorithm works iteratively, pixel by pixel, performing these computations in each iteration as follows:

1. First, compute smoothness of $I^n(i,j)$, utilizing a discrete Laplacian. I_{xx}^n and I_{yy}^n can be computed using a central difference approximation:

$$L^n(i,j) := I_{xx}^n + I_{yy}^n$$

2. Compute smoothness of the previous iteration $I^{n-1}(i,j)$, utilizing a discrete Laplacian again. Once again, utilize central difference approximations to discretize the second derivatives:

$$L^{n-1}(i,j) := I_{xx}^{n-1} + I_{yy}^{n-1}$$

3. Calculate the variation $I_t^{n+1} = L^n(i,j) - L^{n-1}(i,j)$ in the $-\vec{N}$ direction.

$\vec{N}$ is the vector dictating our direction of propagation for each point (i,j). Intuitively, $\vec{N}$ being normal to our boundary is a natural choice. However, propagating in this way does not preserve borders, and we would be implicitly giving a "shape" to the propagation because we propagate towards the center of the occlusion for every point. Ideally, we would like to diffuse/propagate into the direction of isophotes, the direction of minimum variation. This will help preserve and extend borders because the algorithm does not diffuse in a direction with large color changes, which is where we expect these borders. Therefore, our choice of $\vec{N}$ should be:

$$\vec{N} = \nabla^{\perp} I^n(i,j)$$

where $\nabla I^n(i,j)$ is the gradient, or direction of maximal change, so $\nabla^{\perp} I^n(i,j)$ (the perpendicular vector in either direction) is the direction of least change. We choose $\vec{N}$ such that $-\vec{N}$ has a component directed into the occlusion. Therefore, our desired propagation/change I_t is:

$$I_t = -\vec{N} \cdot \Delta\nabla I = -\vec{N} \cdot \nabla\Delta I\,, \quad \vec{N} = \nabla^{\perp} I^n(i,j) \tag{1}$$

with the central difference discretization. We note that $\nabla\Delta I = \Delta\nabla I$ for C^3 functions. Given that we only have discrete pixel values, we can think about these as smooth functions: we can take an infinite number of derivatives.

One small caveat to this algorithm is that the propagation follows level curves: moving in the direction of least change essentially enforces this behavior. However, there are cases where one might need to propagate across a level curve in order to fully inpaint the occlusion. For example, a level curve that exists entirely within the occlusion would not be inpainted by this algorithm as described. To remedy this issue, Bertalmio et al. suggests utilizing intermediate diffusion steps to allow for some propagation across level curves. The propagation ensures everything is inpainted, and the isophote propagation enforces the borders that we desire.

1.1.2 Level Lines [7]

Another geometric approach to inpainting is via level lines. This approach falls under a different class of geometric methods focused on minimizing energy functions, specifically the Euler elastica energy function. We do not implement any of these methods, but these methods constitute a large class of iterative non-PDE geometric methods that are widely studied. We felt it important to highlight this class of iterative method that contrasts with the aforementioned PDE approach described in section 1.1.1.

This discussion will focus on gray scaled images, but in recent years, this method has been generalized to fully colored images. First, we define $u(x)$ as the grayness of an image u at point x. Next, we define the level lines $X_t u$ as

$$X_t u = \{x \mid u(x) \geq t\}$$

Notably, this definition is such that $u(x) = \sup\{t \mid u(x) \in X_t u\}$. Effectively, these level lines $X_t u$ are defined by the lower value bound t. Now, let Ω be the area which we want to inpaint. Then, given u, we want to propagate the level lines into the occlusion. To properly propagate these level lines, we define a term T-Junction. A T-Junction is a point at the intersection of $\partial\Omega$ and a level line. In particular, we can assign an orientation to each level line entering Ω by considering whether ∇u along the level line is oriented into the occlusion or not. For every positively oriented level line, there is a negatively oriented level

line; a line "going into" the occlusion must also exit the occlusion. Then, the problem reduces to optimizing a set of curves $(\Gamma_j^t)_{j\in J(t)}$ where $J(t)$ are the various level lines. Note that $|J(t)| = \frac{|\{\text{T-JunctionPoints}\}|}{2}$ because a T-Junction exists at every level line intersection with the border, and each level line intersects $\partial\Omega$ at two points. Then, $(\Gamma_j^t)_{j\in J(t)}$ is optimized to reduce an energy equation:

$$E = \int_{-\infty}^{\infty} \sum_{j\in J(t)} \left(\int_{\Gamma_j^t} (\alpha + \beta|\kappa|^p) ds + \text{angles} \right)$$

α, β are positive constants and $|\kappa|$ is the curvature exponent from Euler's elastica energy.

1.2 Existing Python Implementations

Following a discussion of important methods for geometric inpainting, we turn to two methods for image inpainting that are implemented in OpenCV's *cv2* Python package. We used Navier Stokes and the Fast Marching Method as baseline comparisons for our methods.

1.2.1 Navier Stokes [2]

The Navier stokes equations are a set of equations in fluid dynamics:

$$v_t + v \cdot v = -\nabla p + \nu \Delta v$$
$$\nabla \cdot v = 0$$

where v is the velocity, v_t is change in velocity, v is the viscosity, and p is the pressure [2]. Transitioning to a two dimensional Navier Stokes, we introduce a stream function Ψ where $\nabla^\perp \Psi = v$. If we let the vorticity $\omega = \Delta\Psi = \nabla \times v$ (the amount of "rotation" in our fluid), then the new 2D Navier Stokes is:

$$\omega_t + v \cdot \nabla \omega = \nu \Delta \omega$$

where w_t is the change in vorticity at any given point. Converting to our image inpainting nomenclature, we have that Ψ is equivalent to the image intensity I, fluid velocity $v = \nabla^\perp \Psi$ is equivalent to isophote direction $\nabla^\perp I$, vorticity $\omega = \Delta\Psi$ is equivalent to smoothness ΔI. Lastly, viscosity ν is equivalent to the anisotropic diffusion coefficient c. Rewriting our Navier stokes equation in terms of image inpainting terms, we have the equations below with the g accounting for anisotropic diffusion, that is diffusion that preserves edges.

$$w_t + v \cdot \nabla w = \nu \Delta w$$

$$I_t + \nabla^\perp I \cdot \nabla \Delta I = c \nabla \cdot (g(|\nabla \Delta I|) \nabla \Delta I) \tag{2}$$

c, g are determined experimentally. We note that when $c = 0$ (i.e. when we expect no anisotropic diffusion), we have that

$$I_t + \nabla^\perp I \cdot \nabla \Delta I = 0$$
$$I_t = -\nabla^\perp I \cdot \nabla \Delta I$$

We note that this looks exactly like equation 1! The non-anisotropic case leads to precisely the iterative method described by Bertalmio et al. Navier stokes simply adds an anisotropic diffusion term that improves on the edge preservation from section 1.1.1.

1.2.2 Fast Marching Method (FMM) [4]

The Fast Marching method is loosely based on section 1.1.1. Given an occlusion $\Omega \subset D$, we want to inpaint the image starting at the points closest to $\partial\Omega$. To start, take $p \in \partial\Omega$. We consider $B_\varepsilon(p)$, a ball of radius ε for sufficiently small ε around p. Given that $p \in \partial\Omega$, $B_\varepsilon(p) \cap D \neq \emptyset$. All the pixel values of q are in the known region. Then, given the $q \in B_\varepsilon(p) \cap D$, we can compute the value of p to be:

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

This is essentially a linear interpolation in the direction of $\nabla I(q)$. Then, we take a weighted average of $I_q(p)$ for all $q \in B_\varepsilon(p) \cap D$ to determine the propagated $I(p)$. In particular, given a normalized weighting function $w(p, q)$ ($\sum_{q \in B_\varepsilon(p)} I_q(p) = 1$):

$$I(p) = \frac{\sum_{q \in B_\varepsilon(p)} w(p, q) I_q(p)}{\sum_{q \in B_\varepsilon(p)} w(p, q)} \tag{3}$$

To inpaint, equation 3 is used repeatedly. We first apply it to $p \in \partial\Omega$ and then we apply it to the next closest set of points (those one pixel away). This process is repeated until the entire Ω has been filled. This is the Fast Marching Method: start from the boundary and move inwards as we repeatedly apply equation 3.

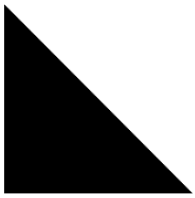
Lastly, it remains to discuss our choice of weighting function using this method.

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

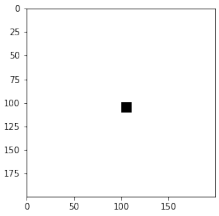
$dir(p, q)$ weights points lying close to the line L where p lies on L and $L \perp \partial\Omega$ at p. $dst(p, q)$ weights points close to p; d_0 is the distance between pixels. $lev(p, q)$ weights the p, q based on their distance to the boundary $T(p)$, $T(q)$; T_0 is the distance between pixels. The set of $w(p, q)$ may require normalization after being calculated. In practice, $T_0 = d_0 = 1$ is commonly used.

2 Survey of Implemented Methods

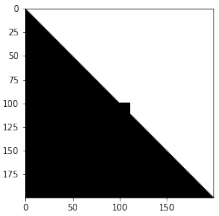
Next, we discuss a few of our image inpainting implementations. For each of the following methods in this section, we assumed that we are given two inputs: an RGB image (with each color channel assumed to have values of type float in the range $[0, 1]$) that has a region which is blank (zeroed out), and a mask that indicates with 0's and 1's whether or not a given pixel is part of an image. The problem would greatly increase in difficulty without the mask, as the region to be inpainted would then have to be identified. For each of the methods, we used the images and masks in Figure 2 for testing. We will refer to the images as Diagonal and Orange. Diagonal is an extreme test of whether or not the inpainting preserves hard edges, while Orange tests the method's general ability to inpaint relatively homogeneous regions. For each method we may assume that we are working with greyscale images: we simply apply the method on each color channel and then combine the results to obtain the final image. For the reconstructed images, we let IR stand for the inpainted region.



(a) Black/White Diagonal, Original

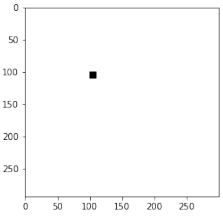


(b) Mask for Diagonal

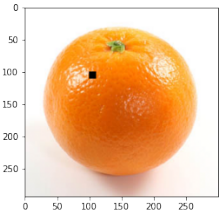


(c) Diagonal with Region Removed

(d) Orange, Original



(e) Mask for Orange



(f) Orange with Region Removed

Figure 2: Our two test images, labeled Diagonal (2a) and Orange (2d). 2b and 2e are the masks for Diagonal and Orange, respectively, and 2c and 2f are Diagonal and Orange with the mask regions removed. The region removed for Diagonal contains a sharp edge, while the region removed for Orange is more homogeneous. We will apply our various implementations against these two images and masks.

2.1 Heat Equation and Diffusion[8]

The first method we implemented utilized the heat equation in two dimensions. Let $u_{j,k}$ be the value of the image (on one color channel) at the point (j, k). Let u_t represent the partial derivative of u with respect to time, and let u_x and u_y represent the partial derivatives of u with respect to the x and y directions, respectively. The heat equation then diffuses the pixels bordering the hole into the hole by the following equation:

$$u_t = u_{xx} + u_{yy} \tag{4}$$

To implement this in code, we discretized the equation using central difference approximations and the forward Euler method to obtain the following iterative formula:

$$\frac{u_{j,k}^{n+1} - u_{j,k}^{n}}{\Delta t} = \frac{u_{j-1,k}^{n} - 2u_{j,k} + u_{j+1,k}^{n}}{h^2} + \frac{u_{j,k-1}^{n} - 2u_{j,k} + u_{j,k+1}^{n}}{h^2} \tag{5}$$

$$= \frac{u_{j,k-1}^{n} + u_{j,k+1}^{n} + u_{j-1,k}^{n} + u_{j+1,k}^{n} - 4u_{j,k}}{h^2}, \tag{6}$$

where h is the distance between pixels, set to 1. We now have an iterative formula from which we can diffuse the values of our pixels into any region to be inpainted. We successfully implemented this method, and the results are shown in Figures 3i, 3j, 3k, and 3l (labeled Forward Euler). We also attempted another implementation based on the steady state solution to the discretized heat equation. The steady state

solution for the heat equation is given by:

$$u_{j,k} = \frac{u_{j,k-1} + u_{j,k+1} + u_{j-1,k} + u_{j+1,k}}{4} \tag{7}$$

In this implementation, we diffuse towards the center by consistently averaging all neighboring pixels to obtain the value of a given pixel following the steady state equation 7:

$$u^{n+1}_{j,k} = \frac{u^n_{j,k-1} + u^n_{j,k+1} + u^n_{j-1,k} + u^n_{j+1,k}}{4} \tag{8}$$

The results of this algorithm (Standard Diffusion) as applied to test image Diagonal and Orange are shown in Figures 3a, 3b, 3e, 3f. Note the smoothness in the inpainted regions in both this method and the previous Forward Euler one. While reasonable for the homogeneous case of the orange, this is undesirable in the case of the Diagonal because diffusion takes place across borders.

We further increased the convergence rate of the last method by sampling beyond the nearest neighbors of a pixel in Randomized Diffusion. This used the equation:

$$u^{n+1}_{j,k} = \frac{u^n_{j,k-r_1} + u^n_{j,k+r_2} + u^n_{j-r_3,k} + u^n_{j+r_4,k}}{4} \tag{9}$$

where r_1, r_2, r_3, r_4 were randomly chosen in a suitable interval ([1, 8] in our implementation) for each pixel for every time step, accounting for edge cases appropriately. This allowed the propagation into the region of interest to occur at a more rapid pace, as seen in Figures 7a and 7b. The results are shown in Figures 3c, 3d, 3g, 3h. Note that the tradeoff for this method is that we obtain less homogeneous diffusion and a little bit of blockiness, as evident in both the orange and diagonal cases. In addition, we never converge on a true steady state because the randomness will continue to lead to micro changes to the inpainted area. This also does not fix the issue of blurring the edge in Diagonal.

To remedy the issue of inpainting homogenously without regard for borders, we turn to the Perona-Malik method, which better respects edges.

2.2 Perona Malik [9]

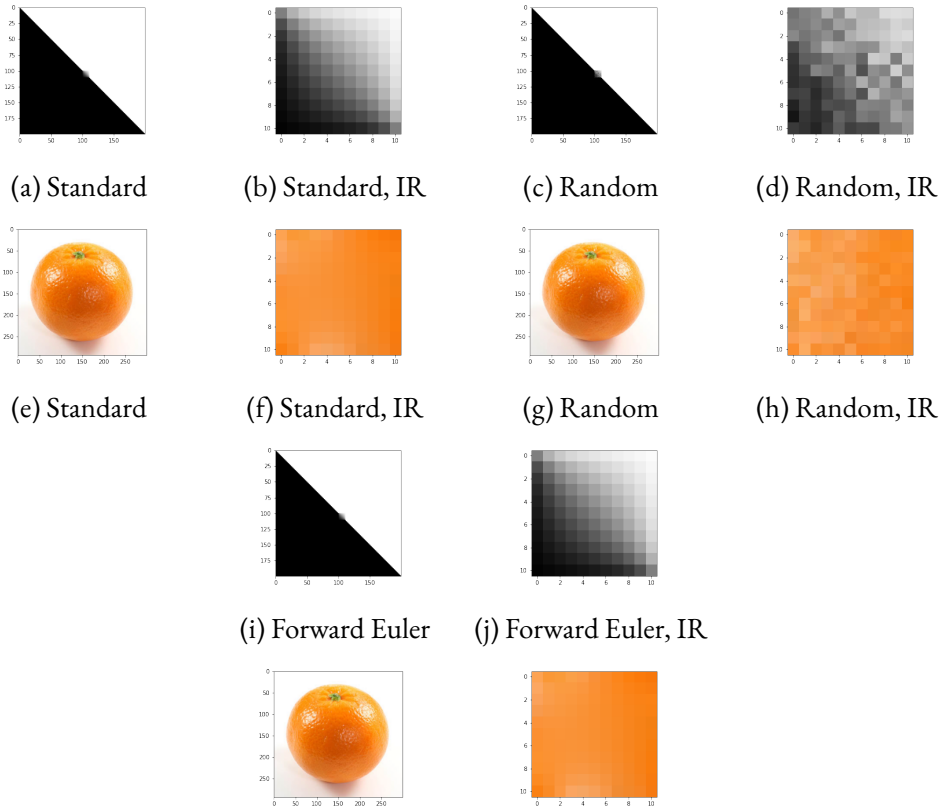
The Perona Malik equation, introduced by Perona and Malik in 1990 is designed to address the issues with the diffusion equation. Namely, in the diffusion equation, the diffusion constant is independent of position, leading to a homogenous inpainting of images. In particular, this in effect blurs the image, not respecting the borders of objects within the image. Instead, Perona Malik turns toward the Anisotropic Diffusion Equation ([9]):

$$u_t = \operatorname{div}(c(x, y, t)\nabla u) = c(x, y, t)\Delta u + \nabla c \cdot \nabla u \tag{10}$$

We note that if the diffusion coefficient $c(x, y, t)$ is a constant, then this equation is equivalent to the prior heat equation:

$$u_t = \operatorname{div}(c\nabla u) = c(u_{xx} + u_{yy})$$

Ideally, to preserve the border and boundary of objects, if we have an object A in our image, we would want $c(x, y, t) = 1$ if $(x, y) \in \text{interior}(A)$ and $c(x, y, t) = 0$ if $(x, y) \in \text{boundary}(A)$. Then, we would not



(a) Standard (b) Standard, IR (c) Random (d) Random, IR

(e) Standard (f) Standard, IR (g) Random (h) Random, IR

(i) Forward Euler (j) Forward Euler, IR

(k) Forward Euler (l) Forward Euler, IR

Figure 3: The results of three implementations of diffusion on Orange and Diagonal. The images labeled IR zoom in on the inpainted region of the previous image. Note the general smoothness of the inpainted regions, especially in the cases of the Standard and Forward Euler implementations. The diffusion using neighbors of random distances apart is slightly less homogenous but is not better at detecting edges.

have diffusion across boundaries. In practice, we cannot know where the interior and boundary of an object is located. However, $||\nabla u(x, y, t)||$ gives us an estimate for how different a pixel and its neighbors are, and Perona and Malik assert that simple functions that are a function of $||\nabla u(x, y, t)||$ do surprisingly well in practice. Hence, we can take $c(x, y, t) : \mathbb{R}^2 \times \mathbb{R}^+ \to [0, 1]$ to be a non-negative monotonically decreasing function in $||\nabla u(x, y, t)||$. Intuitively, if $||\nabla u(x, y, t)|| = 0$, the object is in an interior, so we want $c(x, y, t) \approx 1$. If $||\nabla u(x, y, t)|| = 1$, then we want to minimize diffusion, so we take $c(x, y, t) \approx 0$. Recall that our pixel intensities for each color channel lie in the interval $[0, 1]$, so $0 \leq ||\nabla u(x, y, t)|| \leq 1$. In our implementation, we consider two choices for c:

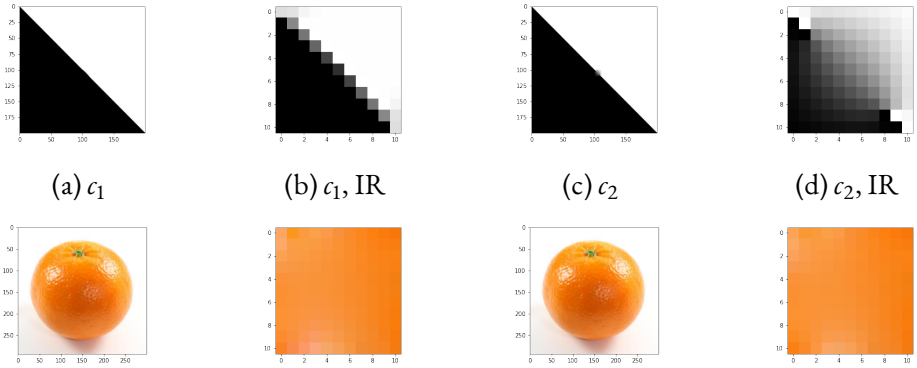
$$c_1(x, y, t) = \frac{1}{1 + \left(\frac{||\nabla u||}{k_1}\right)^2} \tag{11}$$

$$c_2(x, y, t) = e^{-\left(\frac{||\nabla u||}{k_2}\right)^2} \tag{12}$$

where k_1, k_2 are experimentally chosen constants. Both satisfy the non-negative monotonically decreasing condition on the interval $0 \leq ||\nabla u(x, y, t)|| \leq 1$.

Now, given that the Anisotropic diffusion equation is designed for continuous functions c, u, we can implement them in practice by discretizing them using central difference approximations. The procedure for doing so is covered in the Appendix.

The results of our Perona-Malik equation as applied to Orange is shown in Figure 4. We see that, in the almost-homogenous case, Perona-Malik behaves very similarly to the diffusion method. This is expected, since when $||\nabla u(x, y, t)|| \approx 0$ everywhere in our inpainted region, $c \approx 1$ (a constant), and we noted the reduction of Equation 10 to the Heat Equation 4. For Diagonal, we note that the function choice c_1 resulted in a qualitatively clearer distinction for the edge, whereas with c_2 the method inpainted very similarly to the diffusion method.



(a) c_1 (b) c_1, IR (c) c_2 (d) c_2, IR

(e) c_1 (f) c_1, IR (g) c_2 (h) c_2, IR

Figure 4: The results of two different implementations of Perona-Malik on Orange and Diagonal. The images labeled IR zoom in on the inpainted region of the previous image. 4a, 4b, 4e, 4f, are the results of the implementation of Perona-Malik that utilizes c_1 as defined in Equation 11, while 4c, 4d, 4g, 4h, are the results of the implementation of Perona-Malik that utilizes c_2 as defined in Equation 12.

2.3 Isophote Propagation [6]

Finally, we implemented the method as suggested in Bertalmio, et. al [6] that propagates pixels in the directions of least change, following the level lines. As discussed before, Bertalmio utilizes an iterative method with a similar discretized pixel update formula as noted in the previous two methods:

$$u^{n+1}(i,j) = u^n(i,j) + \Delta t u_t^n(i,j), \forall (i,j) \in \Omega$$

where Ω is the region to be inpainted. The update follows the relation

$$u_t^n(i,j) = \left(\delta \vec{L}^n(i,j) \cdot \frac{\vec{N}(i,j,n)}{|\vec{N}(i,j,n)|} \right) |\nabla u^n(i,j)|$$

where

$$\frac{\vec{N}(i,j,n)}{|\vec{N}(i,j,n)|} = \frac{(-u_y^n(i,j), u_x^n(i,j))}{\sqrt{(u_y^n(i,j))^2 + (u_x^n(i,j))^2 + \varepsilon}}$$

is the direction of least change (i.e. perpendicular to the gradient), and

$$\delta\vec{L}^n(i,j) = (L^n(i+1,j) - L^n(i-1,j), L^n(i,j+1) - L^n(i,j-1)),$$
$$L^n(i,j) = u^n_{xx}(i,j) + u^n_{yy}(i,j)$$

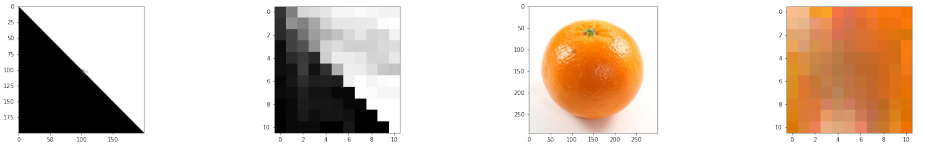
roughly approximates the change in smoothness. The ε is present in order to ensure the equations are well defined in the case when there is a zero gradient. To compute the value of $|\nabla u^n(i,j)|$, the authors use the following approximation:

$$\beta^n(i,j) = \delta\vec{L}^n(i,j) \cdot \frac{\vec{N}(i,j,n)}{|\vec{N}(i,j,n)|}$$

$$|\nabla u^n(i,j)| = \begin{cases} \sqrt{(u^n_{xbm})^2 + (u^n_{xfM})^2 + (u^n_{ybm})^2 + (u^n_{yfM})^2} & \text{when } \beta^n > 0 \\ \sqrt{(u^n_{xbM})^2 + (u^n_{xfm})^2 + (u^n_{ybM})^2 + (u^n_{yfm})^2} & \text{when } \beta^n < 0 \end{cases}$$

where b and f denote backwards and forwards differences, and M and m denote maximum or minimum of a quantity with zero.

In our implementation, we found that the method worked best when we ran a simple diffusion before applying this iteration, which is also motivated by the authors' suggestion of running diffusion interspersed with the isophote propagation. The results show that the method preserves edges and diffuses homogeneous regions to an extent, as shown in Figure 5.



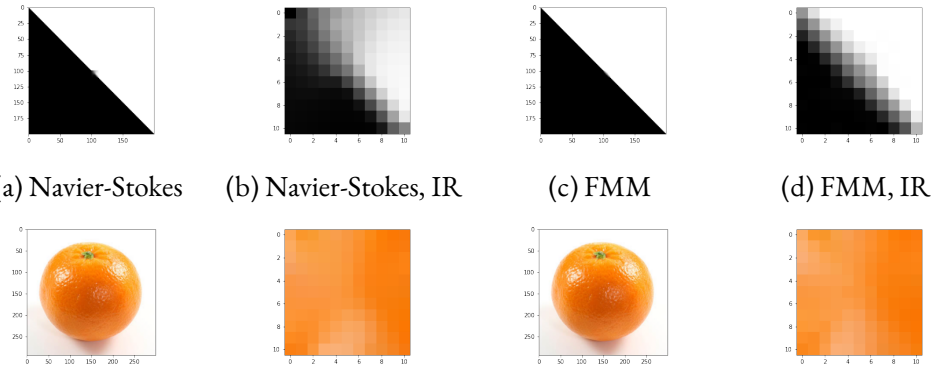
(a) Diagonal (b) Diagonal, IR (c) Orange (d) Orange, IR

Figure 5: The results of the isophote propagation inpainting method as suggested by Bertalmio. The images labeled IR zoom in on the inpainted region of the previous image. Note the edge preservation in (b). However, colors appear in (d) that appear less homogeneous than other methods.

We note that the colors in the inpainted region of the orange are less ideal. We suspect that the same aspect of Bertalmio causes this issue and the appearance of gray cells in the white region of the Diagonal. The ε term in calculating the direction of the level curve ensures that there is no division by zero-gradient error, but it may affect the value of the pixels. One future test to qualify the scope of this issue would be to try images that have small but nonzero gradients (rather than complete white as we did in the Diagonal) and test if edge-preservation would occur in these images.

2.4 OpenCV Methods

As a baseline, we also tested the previously discussed inpainting methods that are implemented in the cv2 library: Navier Stokes and the Fast Marching Method. The results are shown in Figure 6, and the MSEs are included in our evaluations in Table 1.



(a) Navier-Stokes (b) Navier-Stokes, IR (c) FMM (d) FMM, IR

(e) Navier-Stokes (f) Navier-Stokes, IR (g) FMM (h) FMM, IR

Figure 6: The results of cv2 Methods on the Orange and Diagonal. Both do well qualitatively on the Orange, and it appears as though the Fast Marching Method does slightly better than Navier-Stokes in inpainting the hard edge in Diagonal.

2.5 Evaluating our Methods

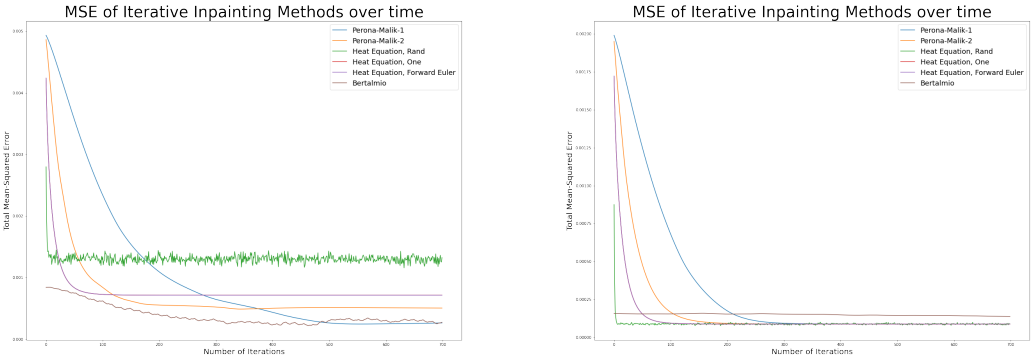
We evaluated our methods using the mean-squared-error (MSE) metric between the restored image and the original image (or, effectively, the inpainted region with the original subportion of the image in that region). The MSE is summed over all three color channels. This was done in each iteration for our implemented methods, as shown in Figure 7, to find the convergence rates of each of our methods. Moreover, we recorded each final MSE in Table 1 after observing, qualitatively, the convergence of each of the methods. The following paragraph discusses our observations.

First, among the heat-equation based diffusion methods, the Forward Euler and distance-one steady-state (equation 8) implementations are nearly identical for both the Diagonal and the Orange and almost overlap completely in the final graphs. The implementation utilizing random neighbors, however, fails to achieve the best results in the case of the Orange because it doesn't diffuse smoothly, and its MSE is higher. All of these diffusion methods (especially the random implementation) do poorly on Diagonal, achieving MSEs larger than $7 \cdot 10^{-4}$.

Other than the Bertalmio method (whose issue was addressed earlier), most of the other MSE's for the Orange are similar, so we focus our discussion on the Diagonal. Perona-Malik with coefficient equation c_1 does extremely well in preserving the edge, outdoing both of OpenCV's inpainting methods. Bertalmio is not far behind. These match up with our predictions from the qualitative observations. However, both Bertalmio and Perona-Malik with c_1 take a while to converge compared to simple diffusion methods, as evident in Figure 7a. This may require more fine-tuning of the hyperparameters, or perhaps it suggests a composite method in which we should only apply these more edge-preserving methods when we suspect edges are present in the regions to be inpainted.

2.6 Other Methods

We tried a number of other methods that did not end up achieving results that we hoped for. First, we naively attempted to do interpolation over the image, but the image simply lied in too high-dimensional



(a) Diagonal (b) Orange

Figure 7: Graphs of Mean-Squared Error per iteration for various implemented methods for Diagonal (left) and Orange (right). The names Perona-Malik-1 and Perona-Malik-2 correspond to our implementations of Perona-Malik with choices of c_1 and c_2. Heat Equation, (Rand, One, Forward Euler) refer to the simple diffusion implementations utilizing equations 9, 8, and 5, respectively. Forward Euler and the Steady-State implementation using its direct neighbors are almost identical in convergence rate. Note that each method has qualitatively converged in the given number of iterations for both Diagonal and Orange.

of a space to accurately interpolate over.

Secondly, we attempted to try some newer deep-learning models, but those proved too complex to understand intuitively and to implement in the given timeframe.

Finally, we attempted to use Singular Value Decomposition to extract a low dimensional approximation over a series of images, each containing a region to be inpainted. However, we realized that the SVD could not inpaint well if the regions to be inpainted among the series of images overlapped. In other words, SVD could only faithfully reconstruct the region to be inpainted if there existed two images such that their missing regions were disjoint. However, we reasoned that the best solution in this scenario would simply be to fill in the missing region of one image using information from the other, so SVD

Method	Diagonal	Orange
Diffusion, One	7.15E-04	8.77E-05
Diffusion, Random	8.52E-04	9.72E-05
Diffusion, Forward Euler	7.15E-04	8.77E-05
Perona-Malik, c_1	2.65E-04	8.79E-05
Perona-Malik, c_2	5.06E-04	8.76E-05
cv2, Navier-Stokes	6.84E-04	8.10E-05
cv2, FMM	2.72E-04	8.30E-05
Bertalmio	2.74E-04	1.45E-04

Table 1: Final MSE values for each of the methods implemented as recorded at the end of each run in 7.

does not contribute much in this inpainting process. If the backgrounds of the images differed slightly so that SVD would be necessary, then we recognized this problem to be one of reconstruction and outside the scope of our analysis of image inpainting techniques.

Thus, we deliberately chose to show the three implemented methods of diffusion, anisotropic diffusion, and isophote propagation because of the intuition backing their methods and the results they provided on our test images.

3 Conclusion

In conclusion, we implemented three methods of heat equation diffusion, anisotropic diffusion that maintains object border integrity, and isophote propagation, propagating along level lines and directions of least change. We found that, for the Orange, they all perform to within the same order of magnitude (from an MSE perspective), except Bertalmio, which performed poorly on our homogenous test case. However, in the case of the the Diagonal, the Python package *cv2*'s implementation of the Fast Marching Method, Perona Malik with c_1, and Bertalmio's inpainting algorithm performed the best from an MSE perspective. However, visually, Perona Malik with c_1 (see section 4.1 for details) preserved the edge best.

In the future, we hope to better identify a mechanism for evaluating accuracy. MSE does not necessarily capture the best visual result. Quantitatively assessing our results may not be meaningful, and ultimately, we may need to turn towards surveys conducted on real people, who rank the quality of the inpaintings.

Moreover, composite methods that interpret based on the surroundings whether or not a region to be inpainted has an edge or not and then applies either anisotropic diffusion or heat equation diffusion may combine the strengths of both methods. Given time, we would investigate how exactly to determine whether or not a region should be homogeneous. Also, given that our implementations were focused on geometric inpainting, we lose a lot of textural properties of the Orange. We hope to combine textural and geometric techniques in the future to continue to preserve borders, while also maintaining the texture of the object that we inpaint.

Lastly, we hope to apply these inpainting algorithms to a more diverse array of masks, enabling us to inpaint non-rectangular, more natural regions. In naturally damaged paintings, rarely do we get a perfect rectangle cut out of a region; normally, damage is accumulated over time, where we often get an accumulation of many small pieces of damage on a painting or image. Therefore, we believe it is important for our implementation to be capable of handling any arbitrary mask.

Finally, thank you to Chris and the AM205 teaching staff for teaching a great class and being so responsive on Slack. We greatly appreciate the help that you have provided in the past semester and the fact that you have made it this far in the paper. Happy holidays!

4 Appendix

4.1 Perona Malik Discretization

In this section we derive the discretized version of Perona-Malik for completeness. Recall that the Anisotropic Diffusion Equation (Equation 10) is given by the following:

$$u_t = \text{div}(c(x, y, t)\nabla u) = c(x, y, t)\Delta u + \nabla c \cdot \nabla u$$

The left hand can be discretized into the following:

$$\frac{\partial u}{\partial t} \approx \frac{u_{jk}^{n+1} - u_{jk}^{n}}{\Delta t}$$

For the right hand side, we show derivations for both c_1 (Equation 11) and c_2 (Equation 12):

$$\begin{aligned}
c_1(x, y, t) &= \frac{k_1^2}{k_1^2 + \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2} \\
\implies \frac{\partial c_1}{\partial x} &= -\frac{k_1^2}{\left(k_1^2 + \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2\right)^2} \cdot \left(2\left(\frac{\partial u}{\partial x}\right) \cdot \left(\frac{\partial^2 u}{\partial x^2}\right) + 2\left(\frac{\partial u}{\partial y}\right) \cdot \left(\frac{\partial^2 u}{\partial y \partial x}\right)\right) \\
\implies \frac{\partial c_1}{\partial y} &= -\frac{k_1^2}{\left(k_1^2 + \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2\right)^2} \cdot \left(2\left(\frac{\partial u}{\partial y}\right) \cdot \left(\frac{\partial^2 u}{\partial y^2}\right) + 2\left(\frac{\partial u}{\partial x}\right) \cdot \left(\frac{\partial^2 u}{\partial x \partial y}\right)\right) \\
c_2(x, y, t) &= e^{-\left(\frac{||\nabla u||}{k_2}\right)^2} \\
\implies \frac{\partial c_2}{\partial x} &= -\frac{e^{-\left(\frac{||\nabla u||}{k_2}\right)^2}}{k_2^2} \cdot \left(2\left(\frac{\partial u}{\partial x}\right) \cdot \left(\frac{\partial^2 u}{\partial x^2}\right) + 2\left(\frac{\partial u}{\partial y}\right) \cdot \left(\frac{\partial^2 u}{\partial y \partial x}\right)\right) \\
\implies \frac{\partial c_2}{\partial y} &= -\frac{e^{-\left(\frac{||\nabla u||}{k_2}\right)^2}}{k_2^2} \cdot \left(2\left(\frac{\partial u}{\partial y}\right) \cdot \left(\frac{\partial^2 u}{\partial y^2}\right) + 2\left(\frac{\partial u}{\partial x}\right) \cdot \left(\frac{\partial^2 u}{\partial x \partial y}\right)\right)
\end{aligned}$$

We now provide central finite difference approximations for the relevant partial differential equations:

$$\begin{aligned}
\frac{\partial^2 u}{\partial y \partial x} &\approx \frac{u_{j+1,k+1}^n - u_{j+1,k-1}^n - (u_{j-1,k+1}^n - u_{j-1,k-1}^n)}{4h^2} \\
\frac{\partial^2 u}{\partial x \partial y} &\approx \frac{u_{j+1,k+1}^n - u_{j-1,k+1}^n - (u_{j+1,k-1}^n - u_{j-1,k-1}^n)}{4h^2}
\end{aligned}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u^n_{j-1,k} - 2u_{j,k} + u^n_{j+1,k}}{h^2}$$
$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u^n_{j,k-1} - 2u_{j,k} + u^n_{j,k+1}}{h^2}$$

$$\frac{\partial u}{\partial y} \approx \frac{u^n_{j+1,k} - u^n_{j-1,k}}{2h}$$
$$\frac{\partial u}{\partial x} \approx \frac{u^n_{j,k+1} - u^n_{j,k-1}}{2h}$$

We note that, if we take $h = 1$ to be the distance between pixels, then $||\nabla u|| \leq 1$ by our image constraint that $0 \leq u \leq 1$. Substituting the approximations into the original equation, we obtain the iterative method we used for Perona-Malik.

References

[1] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative Image Inpainting With Contextual Attention," p. 10.

[2] W. Au, "Image Inpainting with the Navier-Stokes Equations," p. 15.

[3] A. A. Efros and T. K. Leung, "Texture Synthesis by Non-parametric Sampling," p. 6.

[4] A. Telea, "An Image Inpainting Technique Based on the Fast Marching Method," *Journal of Graphics Tools*, vol. 9, pp. 23–34, Jan. 2004.

[5] A. Bugeau and M. Bertalmio, "Combining Texture Synthesis and Diffusion for Image Inpainting.," in *VISAPP 2009 - Proceedings of the Fourth International Conference on Computer Vision Theory and Applications*, (Portugal), pp. 26–33, 2009.

[6] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image Inpainting," p. 12, 2000.

[7] S. Masnou, "Disocclusion: a variational approach using level lines," *IEEE Transactions on Image Processing*, vol. 11, pp. 68–76, Feb. 2002. Conference Name: IEEE Transactions on Image Processing.

[8] "Applying Modern PDE Techniques to Digital Image Restoration."

[9] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 629–639, July 1990. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.