Published in Image Processing On Line on YYYY–MM–DD.
Submitted on YYYY–MM–DD, accepted on YYYY–MM–DD.
ISSN 2105–1232 © YYYY IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
http://dx.doi.org/10.5201/ipol

A Simple Poisson Solver for Image Processing

Enric Meinhardt-Llopis[1], Gabriele Facciolo[2]

[1] CMLA, Ecole Normale Suprieure de Cachan, France [1] CMLA, Ecole Normale Suprieure de Cachan, France

PREPRINT September 14, 2015

Abstract

We present a naïve algorithm for solving the discrete Poisson equation on a domain which is an arbitrary finite subset of a regular grid. The boundary condition for our algorithm must be Dirichlet, there are no restrictions on the shape of the domain, and the running time is proportional to the size of the domain. As an illustration, we present several applications of this algorithm to image processing: inpainting by smooth functions, Poisson editing, and the periodic + smooth image decomposition.

Source Code

The reviewed source code and documentation of a C implementation for the Poisson solver, and of the described applications are available from the web page of this article[1]. Usage instructions are included in the `README.txt` file of the archive.

Keywords: Poisson equation, Laplace equation, Poisson editing, inpainting, interpolation, periodic component

1 Introduction

The celebrated *Poisson Editing* paper by Pérez–Gangnet–Blake contains the following sentences:

> Equations (7) form a classical, sparse (banded), symmetric, positive-definite system. Because of the arbitrary shape of boundary $\partial\Omega$, we must use well-known iterative solvers. Results shown in this paper have been computed using either Gauss-Seidel iteration with successive overrelaxation or V-cycle multigrid. Both methods are fast enough for interactive editing of medium size color image regions, e.g., 0.4 s. per system on a Pentium 4 for a disk-shaped region of 60,000 pixels.

Our goal is to describe explicitly the methods mentioned in this paragraph. We show that, in practice, a parallel CPU multigrid algorithm is capable of achieving realtime performance for images of arbitrary size.

Poisson equation is probably the easiest elliptic PDE, where it is used as an extremely well-behaved model for other, more difficult equations. In numerical analysis, it is used as a toy example to illustrate the power of the most advanced methods. For example, it is one of the few equations where quadratic superconvergence of the multigrid is observing (thus the time to compute a solution is proportional to the size of the problem).

[1] http://dx.doi.org/10.5201/ipol

2 Results on a continuous setting.

Before introducing the discrete algorithms on Section 3 let us recall the basic constructions on a continuous setting.

Poisson equation on a continuous domain. Let Ω be a domain of the euclidean plane $\mathbf{R}^2$, and let $f : \Omega \to \mathbf{R}$ and $g : \partial\Omega \to \mathbf{R}$ be two functions. Poisson equation is

$$\begin{cases} \Delta u = f & \text{on } \Omega \\ u = g & \text{on } \partial\Omega \end{cases} \tag{1}$$

Here the function $u : \Omega \to \mathbf{R}$ is the unknown to be found, the function f is called the datum and g is called the Dirichlet boundary condition. The differential operator $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is called the Laplace operator, and it is arguably the simplest nontrivial second order operator. The existence and uniqueness of solutions to equation (1), provided the data are regular enough, is a standard result in linear PDE theory [9].

Laplace equation. Laplace equation is the particular case of Poisson equation when $f = 0$.

$$\begin{cases} \Delta u = 0 & \text{on } \Omega \\ u = g & \text{on } \partial\Omega \end{cases} \tag{2}$$

Solutions of Laplace equation are called harmonic functions. Thus, an harmonic function is uniquely determined by its values on the boundary of the domain. A very important property of harmonic functions is the *maximum principle*, that states that the minimum and maximum values of u are attained on $\partial\Omega$. In other words, the solution u of (2) is bounded by g:

$$\|u\|_{L^\infty(\Omega)} \leq \|g\|_{L^\infty(\partial\Omega)}$$

Thanks to these properties, we see that Laplace equation can be used to *interpolate* [5] the values g given on the curve $\partial\Omega$ to the whole domain Ω.

Relationship with heat equation. The heat equation with source $-f$ and Dirichlet boundary condition g is

$$\begin{cases} u_t = \Delta u - f & \text{on } \Omega \times [0, \infty] \\ u = g & \text{on } \partial\Omega \times [0, \infty] \end{cases} \tag{3}$$

Here we assume that f and g do not depend on t. If the solution u tends to a limit state u^∞ as $t \to \infty$ then u^∞ is a solution of the corresponding Poisson equation. This idea is used in some numerical methods to solve Poisson equation: we solve the corresponding heat equation (which is easier) and we let $t \to \infty$.

Variational interpretation. Laplace equation is associated to the following energy functional

$$E_l(u) = \int_\Omega \|\nabla u\|^2 \tag{4}$$

Solving Laplace equation is equivalent [9] to minimizing $E_l(u)$ among all the functions u such that $u|_{\partial\Omega} = g$. This variational interpretation is important because it says that solutions of Laplace

equation are very smooth (so that they can be very well approximated at a lower resolution). Furthermore, the gradient descent of the functional $E_l(u)$ is the heat equation. The following functional

$$E_p(u) = \int_\Omega fu + \|\nabla u\|^2 \tag{5}$$

is likewise associated to Poisson equation. Notice that the functional is of the form $E_p(u) = a(u,u) + l(u)$, where a is a coercive bilinear map and l is a linear form. Thus, existence and uniqueness of the minimum is directly assured by Lax-Milgram theorem [9].

Another lagrangian: $E_q(u) = \int_\Omega \|\nabla u - \vec{v}\|^2$. This is associated to the Poisson equation $\Delta u = \operatorname{div}(\vec{v})$. This model is useful when we want to recover a function u from its gradient $\vec{v}$. Note that this has a solution even if $\operatorname{rot}(\vec{v}) \neq 0$.

Relationship to Fourier analysis. Let us assume now that $\Omega = \mathbf{R}^2$. Now, Poisson equation reduces to $\Delta u = f$ because there is no boundary. This linear equation can be solved directly by taking Fourier transforms:

$$-(\xi^2 + \eta^2)\hat{u}(\xi,\eta) = \hat{f}(\xi,\eta) \tag{6}$$

so that

$$\hat{u}(\xi,\eta) = \frac{-1}{\xi^2+\eta^2}\hat{f}(\xi,\eta) \tag{7}$$

with $1/0$ being an arbitrary value fixed by convention. Notice that the solution is not unique, being defined up to the addition of an arbitrary harmonic function. The Fourier transforms above are necessarily performed in the sense of distributions. An interesting particular case is when f is a band-limited $[0,2\pi]^2$-periodic function. This means that f is a linear combination of functions $e_{p,q}(x,y) = e^{i(px+qy)}$ with $(p,q) \in \mathbf{Z}^2$, whose anti-Laplacian is readily computed. This particular case is commonly used in image processing for solving Poisson equations on rectangular domains, using the discrete Fourier transform.

3 The discrete Poisson, Laplace and Heat equations

To solve a Partial Differential Equation using a computer, it must be modelled as a finite problem. There are several such discretization strategies. Conceptually, the simplest one is the finite element method [14], where we solve the same differential equation, but restricted to a finite-dimensional subspace of functions. A different strategy, better suited to image processing, is to use a finite-difference approximation [15] of the derivatives appearing in the equation. Thus, the functions appearing on the differential equation are modelled by digital images, and the differential equation is reduced to an algebraic equation involving the pixel values of these images. This is the strategy that we will use here.

The whole domain of a digital image is a finite rectangular grid $D = \{1,\dots,W\}\times\{1,\dots,H\} \subset \mathbf{Z}^2$. We work with a region of interest Ω which is an arbitrary subset of D. The boundary of Ω is defined as the points of $D \setminus \Omega$ that are 4-connected to Ω. This definition of boundary is well-suited for the discrete Laplacian introduced below, but for other operators a different boundary may be needed. A digital image is a function $I : D \to \mathbf{R}$. The discrete Laplacian is defined by the following five-point scheme:

$$\Delta u(i,j) = -4u(i,j) + u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1) \qquad \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \tag{8}$$

Notice that to compute Δu at the boundaries of D we need to give values to u outside D. This is done by copying the value of u from the nearest point inside D. This is an arbitrary choice

Algorithm 1: `evaluate-image-at`

Input : Image $I : \{1, \dots, W\} \times \{1, \dots, H\} \to \mathbf{R}$
Input : Position $(i, j) \in \mathbf{Z}^2$
Output: Value of $I(i, j)$

```
if i < 1 then
    i ← 1

if i > W then
    i ← W

if j < 1 then
    j ← 1

if j > H then
    j ← H

return I(i, j)
```

that corresponds to a vanishing Neumann condition on the boundary of D. It is also equivalent to the *graph Laplacian* [3] on the graph whose vertices are the points of D and whose edges are given by 4-connectivity. More formally:

Definition 1 (Discrete Laplace Operator)**.** *Let W and H be positive integers and let $D = \{1, \dots, W\} \times \{1, \dots, H\}$. A digital image of size $W \times H$ is a function $u : D \to \mathbf{R}$. The* discrete Laplace operator *on D is the linear endomorphism $u \mapsto \Delta u$ defined by*

$$\Delta u(p) = \sum_{p' \in D \text{ is a 4-neighbor of } p} \Big(u(p') - u(p) \Big) \tag{9}$$

Note that Definition 1 is equivalent to the five-point scheme (8). A vanishing discrete Laplacian has an intuitive interpretation: if we regard the values of u as measuring some quantity, the fact that $\Delta u(p) = 0$ says that the quantity at point p equals the average quantity on the neighbors of p (thus, $u(p)$ can be recovered by averaging the values of u on the neighbors of p). If $\Delta u(p) = 0$ for every p, this means that the quantity is *in equilibrium*.

With these definitions we can introduce the discrete Poisson equation using the same notation as in the continuous case.

Definition 2 (Discrete Poisson Equation)**.** *Let Ω be a subset of $D = \{1, \dots, W\} \times \{1, \dots, H\}$, let $f : \Omega \to \mathbf{R}$ and let $g : D \setminus \Omega \to \mathbf{R}$. The* Discrete Poisson Equation *on Ω with data f and boundary condition g is*

$$\begin{cases} \Delta u(i, j) = f(i, j) & (i, j) \in \Omega \\ u(i, j) = g(i, j) & (i, j) \in \partial\Omega \end{cases} \tag{10}$$

There are two interesting particular cases of Definition 2. When $f = 0$, the equation is called the *Discrete Laplace Equation* on the domain Ω. When $\Omega = D$, the boundary $\partial\Omega$ is empty and the solution is determined only up to the addition of functions h such that $\Delta h = 0$.

Algorithm 2: `evaluate-laplacian-at` (implements equation 8)

Input : Image $I : D \to \mathbf{R}$
Input : Position $(i, j) \in D$
Output: Value of $\Delta I(i, j)$

$x \leftarrow$ `evaluate-image-at`(I, i, j)
$a \leftarrow$ `evaluate-image-at`$(I, i + 1, j)$
$b \leftarrow$ `evaluate-image-at`$(I, i - 1, j)$
$c \leftarrow$ `evaluate-image-at`$(I, i, j + 1)$
$d \leftarrow$ `evaluate-image-at`$(I, i, j - 1)$
return $a + b + c + d - 4x$

Proposition 1. *The discrete Poisson equation (10) is a square linear system of size n, where n is the cardinal of Ω. If Ω is a proper subset of D then the discrete Poisson equation is a nonsingular linear system with a unique solution. If $\Omega = D$ then the system is singular with a one-dimensional kernel formed by constant functions.*

This proposition assures that the discrete Poisson equation is well-posed, and can be solved by standard linear solvers. Note that, with empty boundary conditions, the only harmonic functions are the constants. An analogous result in the continuous domain states that the only bounded harmonic functions on the whole plane are the constants.

The discrete analogous of Heat equation describes the evolution of a function $u(t, i, j)$ where (i, j) are points on a rectangular grid and t are integer multiples of a time step τ:

Definition 3 (Discrete Heat Equation)**.** *Let Ω be a subset of $D = \{1, \dots, W\} \times \{1, \dots, H\}$, let $f : \tau\mathbf{Z}^+ \times \Omega \to \mathbf{R}$, let $g : \tau\mathbf{Z}^+ \times D \setminus \Omega \to \mathbf{R}$ and let $h : \Omega \to \mathbf{R}$. The* Discrete Heat Equation *on Ω with source f, boundary condition g, initial condition h, and time step τ is*

$$\begin{cases} \frac{u(t+\tau,i,j)-u(t,i,j)}{\tau} = \Delta u(t, i, j) - f(t, i, j) & (i, j) \in \Omega,\ t \in \tau\mathbf{Z}^+ \\ u(t, i, j) = g(t, i, j) & (i, j) \in \partial\Omega,\ t \in \tau\mathbf{Z}^+ \\ u(0, i, j) = h(i, j) & (i, j) \in \Omega \end{cases} \tag{11}$$

Gradient discretization issues. The the guidance field in the discrete domain is depends on the choice of the gradient discetization. This issue is highlighted in Figure **??** *(NO FIGURE YET) where 4 discretization schemes for computing the gradient yield different results. The solution proposed in [1] consists in averaging the solution of the four schemes (CHECK IF CORRECT!).*

4 Applications

Below we display several applications of the discrete Laplace and Poisson equations to image processing. Note that in most of these applications, the equation is used as a heuristic model and its exact solution is not necessarily sought for. Thus, we can solve them satisfactorily by running our algorithm with very few iterations (e.g. 10), which gives approximate solutions with the visually desirable properties, and runs extremely fast.

All the examples that we show here have been computed with the provided code with default parameters. The input images for each experiment are included in the source code distribution, so that these experiments can be reproduced exactly.

4.1 Smooth inpainting

The simplest application of Laplace equation is smooth inpainting (Figure 1). This is the task of filling-in the missing values of an image, like holes or scratches. We define Ω as the set of unknown pixels, and the known values of u outside Ω define the boundary condition. The solution of Laplace equation on this data provides a smooth interpolation of the data.

This technique does not have any of the desirable properties of "real" inpainting methods [2, 6, 7, 8]: it does not recover the texture of the inpainted areas, and it fails even to continue edges that traverse Ω. On the other hand, it may still be useful to complete very small holes (like letters) or thin scratches. It is extremely fast to compute and the solutions satisfy the maximum principle (all the new values are bounded by the values at the boundary of the hole). In some sense, it is the simplest possible inpainting method. Accordingly [11], we name this algorithm *Simplest Inpainting*, as the baseline inpainting method compared to which any other method will fare better.

Input: an image with holes Output: inpainting result

Figure 1: An example of smooth inpainting. The region Ω is defined as the bright green pixels. To assign values to these pixels, we solve Laplace equation on Ω with the boundary condition taken from the known pixels.

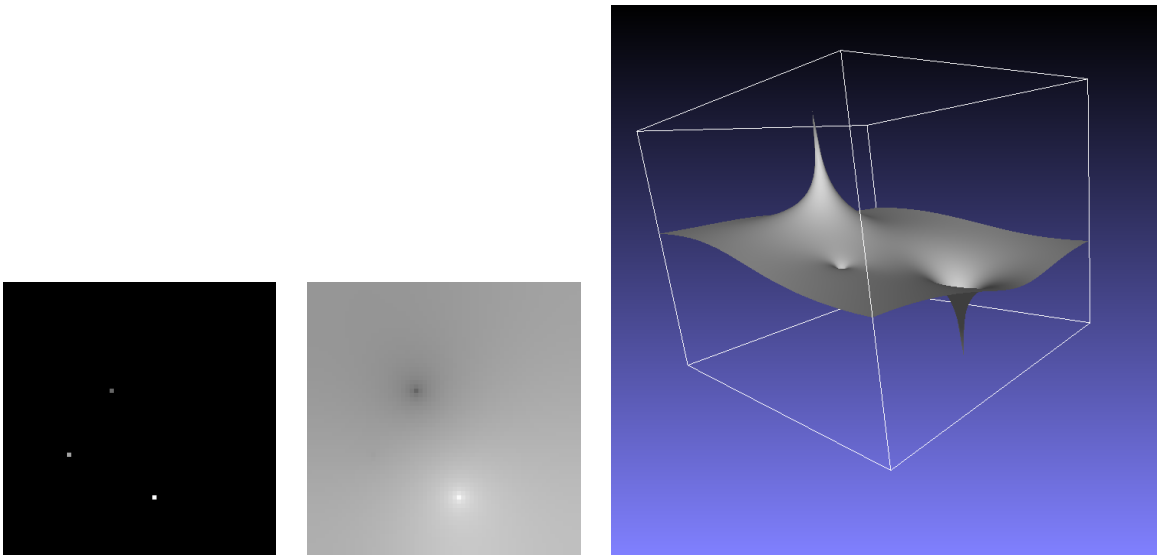
4.2 Interpolation from scattered data points

Interpolation is the extreme case of inpainting where the inpainting region is the whole image except a few scattered points. Since our formulation (10) of the discrete Poisson equation is valid for an

arbitrary domain Ω, the same algorithm can be used. Note that this is not possible in the continuous formulation (1), where $\partial\Omega$ must consist of a set of curves, not isolated points.

The simplest example is when Ω consists of a single pixel. In that case, the value of this pixel is copied everywhere, resulting in a constant image. This happens even with zero iterations, due to the multi-scale scheme.

The next simplest example is three different data points. In that case, we would like to recover an affine image that interpolates the three values and has vanishing Laplacian. However, an affine image does not satisfy the boundary condition at the borders of the image domain. Instead, this results in a surface with cusps around the data points (Figure 2).

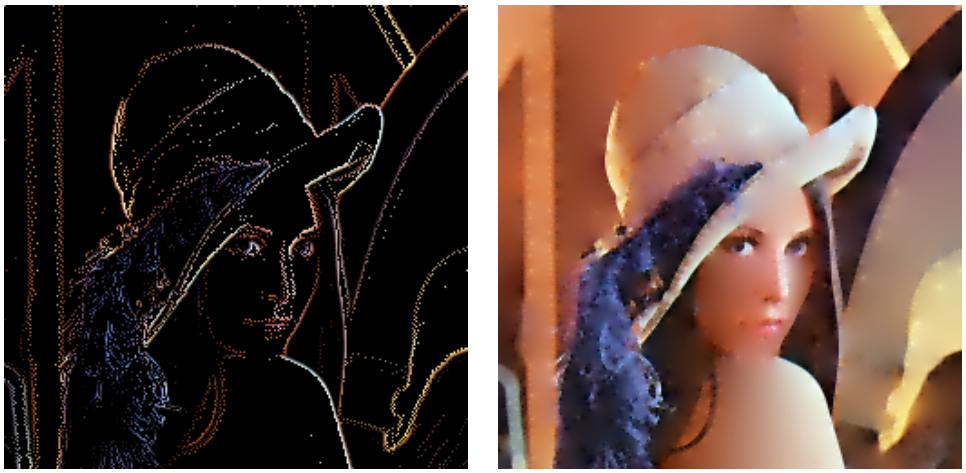


Input Output 3D visualization

Figure 2: Laplace interpolation of three data points.

4.3 Image approximation

It is tempting to feel disheartened by the cusps on Figure 2, and disregard Laplace equation as a sensible tool for interpolation. Yet, Laplace interpolation has been used successfully [10] to approximate images to a very good precision for the purpose of lossy compression. The trick lies in choosing the optimal points to represent the image data. It turns out that the best points are those located in the parts of the image with high gradient, thus a Floyd-Sternberg dithering of the gradient norm gives a near-optimal choice of points. The results are shown on 3.



Input (9% of the original pixels) Output (Laplace interpolation)

Figure 3: Image approximation by Laplace interpolation of carefully selected data points. Here Ω is the black region on the input image, the boundary condition is given by the colored pixels, and we solve Laplace equation on this data.

4.4 Periodic plus smooth image decomposition

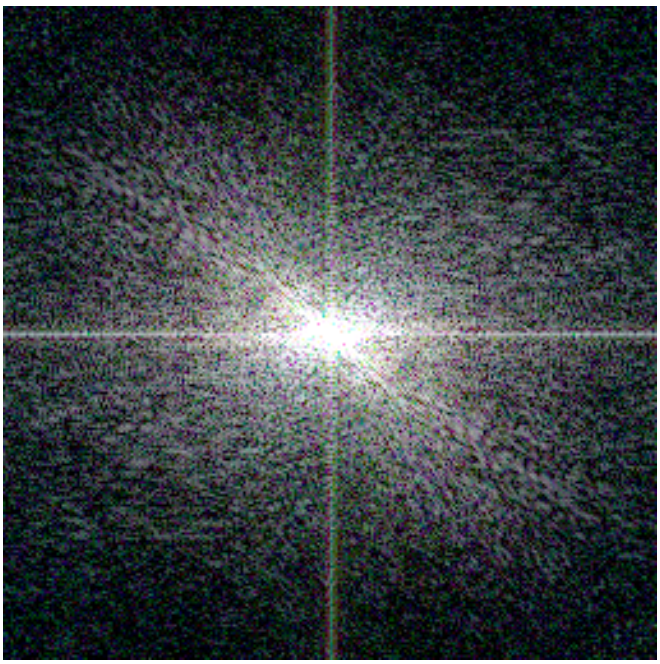
An arbitrary image is typically not tile-able, because the colors at each side of the image do not match, creating sharp discontinuities between tile boundaries (see Figure 5). These discontinuities are horizontal and vertical, resulting in a cross-shaped pattern when one regards the power spectrum of the original image.

A common trick [12] to obtain tile-able textures (and nicer power spectra) is to decompose an image into its "periodic and smooth" components: $I = P + S$, where P is a tile-able image and S is a smooth function. Since S is smooth, the image contents are mostly contained in P. If we define "smooth" as $\Delta S = 0$ and enforce tile-ability by fixing P on its border, we can obtain this decomposition by Laplace equation. The resulting images P are tileable and their power spectrum does not have a cross-pattern (Figure 4).

I P S

Figure 4: Decomposition of an image I into its periodic and smooth components, $I = P + S$. The image S has positive and negative values and it has been re-scaled to $[0, 255]$ for display.

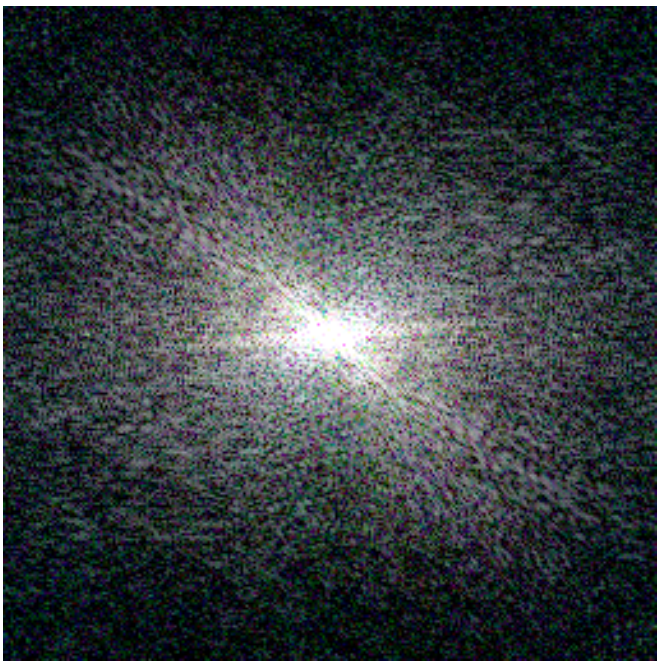
Four copies of an image I



Power spectrum of I

Figure 5: A typical image is not smoothly tile-able. Discontinuities in the replicated image appear as a cross pattern in the power spectrum.

Four copies of P



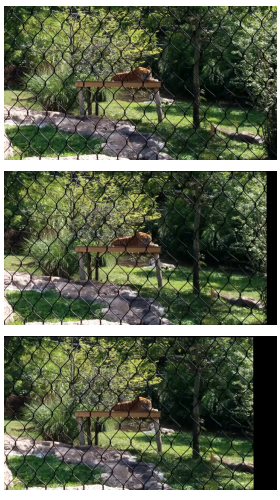
Power spectrum of P

Figure 6: The image P of Figure 4 is tile-able. Note that there are no discontinuities at the boundaries, and that the power spectrum has no cross pattern.

4.5 Recover an image from its gradient

Poisson equation allows us to recover (up to an additive constant) an image I from the vector field ∇I. For that, we compute the scalar field $f = \operatorname{div}(\nabla I)$, and then we solve Poisson equation $\Delta u = f$. Since the domain is the whole image, this can be solved in the frequency domain multiplying by the filter $\frac{-1}{\xi^2+\eta^2}$, where the indeterminacy $1/0$ is assigned an arbitrary value 0. The resulting image u has the required gradient and zero average. We cannot use directly the discrete Poisson equation (10) because it requires a Dirichlet boundary condition on at least one point. Thus, we fix the value of u at a single pixel, and define the domain Ω as the complement of that pixel. The solution of the discrete Poisson equation with this data coincides exactly with I when this single pixel is set to its correct value. In practice, this algorithm is an order of magnitude slower than using Fourier transforms.

Of course, you never need to compute an image from its own gradient (because you already have the image!). In computational photography, however, you often build the gradient of an image that you do not see, and then you need to recover this image. For example, to remove occlusions by fusion of several images (see Figure 7), it is convenient to combine the gradients of several images [4]. Then, a clean image is recovered from this gradient by solving Poisson equation. The Fourier technique is Fast, but often it introduces artifacts due to the discontinuities at the boundary of the registered images. Solving the discrete Poisson equation by the techniques allows a finer control of the desired boundary conditions; in particular, the domain need not be rectangular (see Figure 8).



...

Input images

Average of input images

Combined Laplacian

Figure 7: An image fusion technique [4] computes the pointwise vector median of the gradients of several registered images. Then the image without occlusions is recovered by solving Poisson equation on the combined Laplacian.

Ω = Whole image, using Fourier

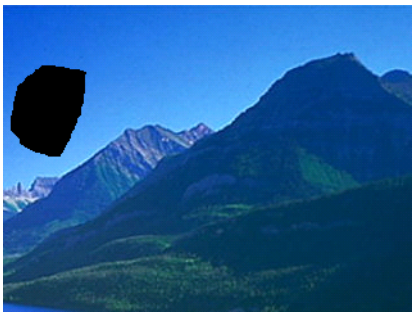
Ω = Central ellipse, using Gauss-Seidel

Figure 8: Notice that the result with Fourier is locally correct (the occluding fence has disappeared), but there are low-frequency color halos covering the whole image. These halos are due to the boundary conditions implied by the discrete Fourier transform. Solving Poisson equation by iterative techniques allows finer control of the boundary conditions. Here, we have solved Poisson with Ω equal to an elliptical region in the image domain. The boundary condition has been taken from the first image of the input sequence. Note that there are no low-frequency artifacts due to bad boundary conditions.

4.6 Poisson editing

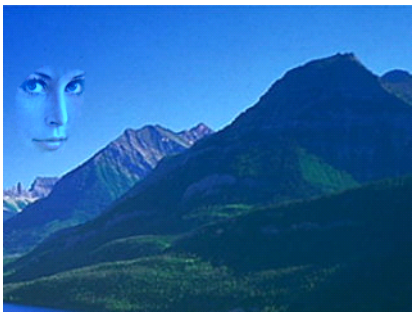
The image fusion technique described above is just an example of *Poisson Editing* [1]. We can obtain a seamless cloning by copying the gradients from one image into another, and then solving a Poisson equation (see Figure 9).

[1] M. W. Tao, M. K. Johnson, and S. Paris, "Error-tolerant image compositing," in Proceedings of the 11th European conference on Computer vision: Part I, 2010, pp. 31–44.



Ω and g

Data term f



Poisson solution u

Figure 9: An example of Poisson editing: how to draw the face of Lena on the sky. For this experiment, we have cut the face of the Lena image and computed its Laplacian f. Then we have imposed this Laplacian on part Ω of another image g, by computing the solution of Poisson equation $\Delta u = f$ on Ω, with Dirichlet boundary condition g. Note that the colors at the boundary are taken from the landscape image, and they are filled using the texture of the face.

4.7 Poisson matting

[1] J. Sun, J. Jia, C. K. Tang, and H. Y. Shum, "Poisson matting," ACM Trans. Graph., vol. 23, no. 3, pp. 315–321, Aug. 2004.

From the article

In image composition, an image I is divided in a background image B and a foreground image F with its alpha matte α by the matting equation:

$$I = \alpha F + (1 - \alpha)B.$$

In order to get an approximate gradient field of matte, we take the partial derivatives on both sides of the matting equation:

$$\nabla I = (F - B)\nabla\alpha + \alpha\nabla F + (1 - \alpha)\nabla B.$$

This is the differential form of the matting equation, for R, G and B channels individually. In situations in which foreground F and background B are smooth, i.e., $\alpha\nabla F + (1 - \alpha)\nabla B$ is relatively small with respect to $(F - B)\nabla\alpha$, we can get an approximate matte gradient field as follows:

$$\nabla\alpha \approx \frac{1}{F - B}\nabla I.$$

It means that the matte gradient is proportional to the image gradient.

5 The Gauss-Seidel and conjugate gradient algorithms

The discrete Heat equation is not an equation that must be "solved" in any way. The values $u(t, i, j)$ at any $t = k\tau$ can be computed directly by an iterated evaluation of the definition.

Thus, a simple method to solve the discrete Poisson equation is to iterate the discrete Heat equation until convergence is observed. The following proposition states that convergence happens if the time step τ is small enough.

Proposition 2. *In the discrete Heat equation (11), assume that $\tau < 1/2$ and that f and g do not depend on t. Then the succession of images $u^k(i, j) = u(k\tau, i, j)$ is convergent and the limit u^∞ is a solution of the discrete Poisson equation with data f and boundary condition g.*

Proof. See for example [13]. It suffices to show that the evolution of the discrete heat equation is described by the iterated multiplication by a matrix whose singular values are all strictly smaller than 1. This condition is assured by the inequality $\tau < 1/2$. □

Remark 1. *The convergence described in proposition 2 holds true for any initial condition h.*

The previous proposition provides an algorithm to obtain arbitrarily good approximations for the solution of the discrete Poisson equation equation:

1. Chose a time step $\tau \in (0, 1/2)$
2. Chose an initialization $h(i, j)$
3. Chose a number of iterations N
4. Compute $u(N\tau, i, j)$ using equations (11).

Standard numerical linear algebra gives optimal answers for these three choices (see chapter 5.3 of Strang's book [13] for a fascinating account on this subject). Here we are interested in the best answers for image processing applications, were exact convergence is not always needed. For example, for interpolation, or for Poisson editing, a visually acceptable approximation is extremely fast to compute (say, in 3 iterations), but it can be very far from convergence.

Let us recall four standard iterative methods for solving a linear system $Ax = b$, given an initial guess x^0. Applied to the discrete Poisson equation, the methods are correspond to the following:

1. **Jacobi Method.** Iterate the discrete heat equation with $\tau = 0.25$.
2. **Gauss-Seidel Method.** Iterate the discrete heat equation *in-place* with $\tau = 0.25$.
3. **Successive Over-Relaxation.** Iterate the discrete heat equation *in-place* withp $\tau = 0.48$.
4. **Conjugate gradient.** Unrelated to heat equation.

What does it mean "in-place"? Well, note that to implement Jacobi method, we need to store two images in memory, one for the current iteration, and different one for the next iteration. These images are swapped upon each iteration. In Gauss-Seidel method, we use only one image, and perform all the computations in-place. This is obviously incorrect (with regards to heat equation) because when we compute the five-point scheme on a new position, we are mixing values of the previous and the current iteration. However, this "incorrect" method turns out to converge faster to the correct solution. In Successive Over-Relaxation, we use a time step which is almost twice as large as that allowed by Jacobi's method, which converges even faster. Note that the ordering of the pixels is inconsequential for Jacobi method, but a different ordering of pixels gives different results for the Gauss-Seidel iterations.**Nota: aquí se podrían poner los experimentos sobre el orden de recorrido de Ω: lexicographic, forth-back, 4-ways, random, onion-peeling...**.

Algorithm 3: `gauss-seidel-iteration`

Input : Image $I : D \to \mathbf{R}$ // image with a combination of u and g
Input : Image $f : \Omega \to \mathbf{R}$ // data term
Input : Number $\tau > 0$ // time step
Output: (no output, the data of I is updated in-place)

1 **for** $(i, j) \in \Omega$ **do**
2 $\quad \ell \leftarrow$ `evaluate-laplacian-at`(I, i, j)
3 $\quad I(i, j) \leftarrow I(i, j) + \tau(\ell - f(i, j))$

Algorithm 4: `gauss-seidel-solver`

Input : Image $f : \Omega \to \mathbf{R}$ // data
Input : Image $g : D \setminus \Omega \to \mathbf{R}$ // boundary condition
Input : Image $h : \Omega \to \mathbf{R}$ // initialization
Input : Number $\tau > 0$ // time step
Input : Number $N \in \mathbf{Z}^+$ // number of iterations
Output: Image $u : \Omega \to \mathbf{R}$ // computed approximation

1 **for** $(i, j) \in \Omega$ **do** // fill-in $I(\Omega)$ with the initialization h
2 $\quad I(i, j) \leftarrow h(i, j)$
3 **for** $(i, j) \in D \setminus \Omega$ **do** // fill-in $I(\partial\Omega)$ with the boundary condition g
4 $\quad I(i, j) \leftarrow g(i, j)$
5 **for** $k \in 1 \dots N$ **do** // run N iterations
6 $\quad$ `gauss-seidel-iteration`(I, f, τ)
7 **for** $(i, j) \in \Omega$ **do** // fill-in $u(\Omega)$ with the result
8 $\quad u(i, j) \leftarrow I(i, j)$

6 The Multi-scale strategy

As it is shown on Section 7, the convergence of Gauss-Seidel and conjugate gradient methods towards the solution of Poisson equation is rather slow. However, looking at the intermediate images on each iteration we see that the small details are solved on the first few iterations, while the global trends in the image require thousands of iterations. The main insight of multi-scale methods is that *any object is small enough when seen from far enough.*

Taking this insight into account, we see that the main tools of the multi-scale methods will be the operations of `zoom-in` and `zoom-out`.

Algorithm 5: `zoom-out-by-factor-two`

Input : Image $f : \{1, \dots, W\} \times \{1, \dots, H\} \to \mathbf{R}$
Output: Image $g : \{1, \dots, W/2\} \times \{1, \dots, H/2\} \to \mathbf{R}$

for $(i, j) \in \{1, \dots, W/2\} \times \{1, \dots, H/2\}$ **do** // traverse the output image
  $a_1 \leftarrow f(2i + 0, 2j + 0)$ // values of the four corresponding points
  $a_2 \leftarrow f(2i + 1, 2j + 0)$ // on the input image
  $a_3 \leftarrow f(2i + 0, 2j + 1)$
  $a_4 \leftarrow f(2i + 1, 2j + 1)$
  $m \leftarrow 0$ // cumulative sum
  $c \leftarrow 0$ // counter of finite values
  for $k \in \{1, \dots, 4\}$ **do** // compute sum and amount of finite values
    if `is-finite`(a_k) **then**
      $m \leftarrow m + a_k$
      $c \leftarrow c + 1$
  if $c > 0$ **then**
    $g(i, j) \leftarrow m/c$ // average of all finite values, if there were any
  else
    $g(i, j) \leftarrow NaN$ // NaN otherwise

Algorithm 6: `zoom-in-by-factor-two`

Input : Image $f : \{1, \dots, W/2\} \times \{1, \dots, H/2\} \to \mathbf{R}$
Output: Image $g : \{1, \dots, W\} \times \{1, \dots, H\} \to \mathbf{R}$

for $(i, j) \in \{1, \dots, W\} \times \{1, \dots, H\}$ **do** // traverse the output image
  $x \leftarrow (i - 0.5)/2$
  $y \leftarrow (j - 0.5)/2$
  $g(i, j) =$ `bilinear-interpolation`(f, x, y)

Algorithm 7: `bilinear-interpolation`

Input : Image $f : D \to \mathbf{R}$
Input : Number $x \in \mathbf{R}$
Input : Number $y \in \mathbf{R}$
Output: Number $f(x, y)$

$i \leftarrow \lfloor x \rfloor$
$j \leftarrow \lfloor y \rfloor$
$a \leftarrow$ `evaluate-image-at`$(f, i + 0, j + 0)$
$b \leftarrow$ `evaluate-image-at`$(f, i + 1, j + 0)$
$c \leftarrow$ `evaluate-image-at`$(f, i + 0, j + 1)$
$d \leftarrow$ `evaluate-image-at`$(f, i + 1, j + 1)$
$r \leftarrow$ `evaluate-bilinear-cell`$(a, b, c, d, x - i, y - j)$
return r

Algorithm 8: `evaluate-bilinear-cell`

Input : Numbers $a, b, c, d \in \mathbf{R}$ // values on the four cell corners
Input : Numbers $x, y \in \mathbf{R}$ // position inside the cell
Output: Number r // bilinear interpolation of the four values

$r \leftarrow a(1-x)(1-y) + bx(1-y) + c(1-x)y + dxy$
return r

Algorithm 9: `pyramidal-laplace-solver`

Input : Image $g : D \setminus \Omega \to \mathbf{R}$ // boundary condition
Input : Number $\tau > 0$ // time step
Input : Number $N \in \mathbf{Z}^+$ // number of iterations
Input : Number $S \in \mathbf{Z}^+$ // number of scales
Output: Image $u : D \to \mathbf{R}$ // computed approximation

if $S > 1$ **then** // compute initialization Y recursively
    $g' \leftarrow$ `zoom-out-by-factor-two`(g)
    $u' \leftarrow$ `pyramidal-laplace-solver`$(g', \tau, N, S-1)$
    $Y \leftarrow$ `zoom-in-by-factor-two`(u')
else // on last scale, initialize Y to zero
    for $p \in D$ **do**
        $Y(p) \leftarrow 0$
$u \leftarrow$ `gauss-seidel-solver`$(0, g, Y, \tau, N)$// run the solver initialized by Y
return u

Algorithm 10: `pyramidal-poisson-solver`

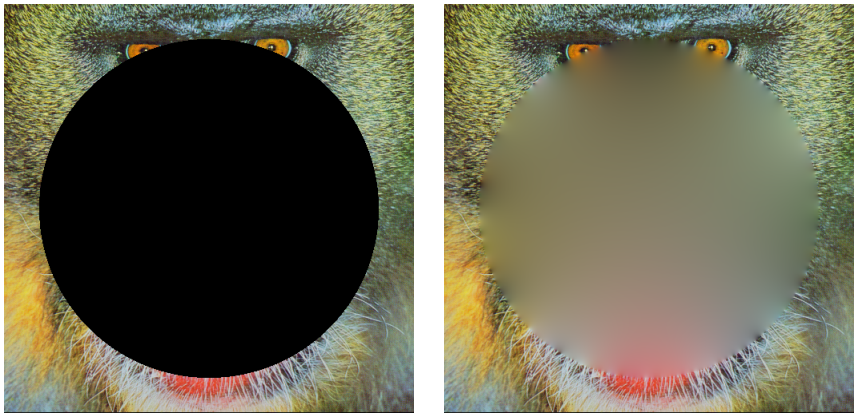
Input : Image $f : \Omega \to \mathbf{R}$ // data term
Input : Image $g : D \setminus \Omega \to \mathbf{R}$ // boundary condition
Input : Number $\tau > 0$ // time step
Input : Number $N \in \mathbf{Z}^+$ // number of iterations
Input : Number $S \in \mathbf{Z}^+$ // number of scales
Output: Image $u : D \to \mathbf{R}$ // computed approximation

if $S > 1$ **then** // compute initialization Y recursively
    $g' \leftarrow$ `zoom-out-by-factor-two`(g)
    $f' \leftarrow$ `zoom-out-by-factor-two`(f)
    for $p \in D$ **do**
        $f'(p) \leftarrow 3f'(p)$// appropriate re-scaling of the data term

    $u' \leftarrow$ `pyramidal-poisson-solver`$(g', f', \tau, N, S-1)$
    $Y \leftarrow$ `zoom-in-by-factor-two`(u')
else // on last scale, initialize Y to zero
    for $p \in D$ **do**
        $Y(p) \leftarrow 0$
$u \leftarrow$ `gauss-seidel-solver`$(0, g, Y, \tau, N)$// run the solver initialized by Y
return u

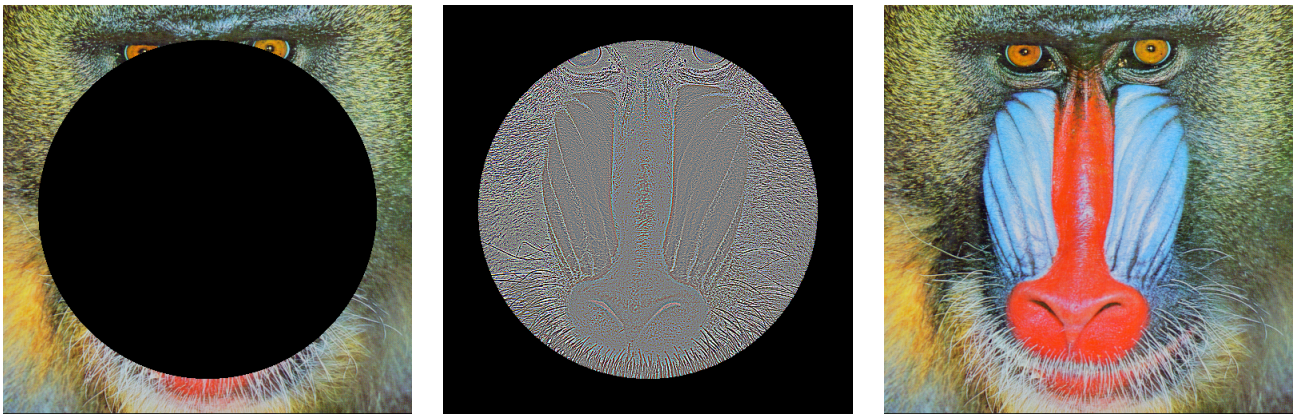
7 Experiments

In this section we illustrate the effect of each step of the proposed algorithm over two simple examples, one for Laplace equation and one for Poisson equation. We show the solution of each case in Figures 10 and 11.



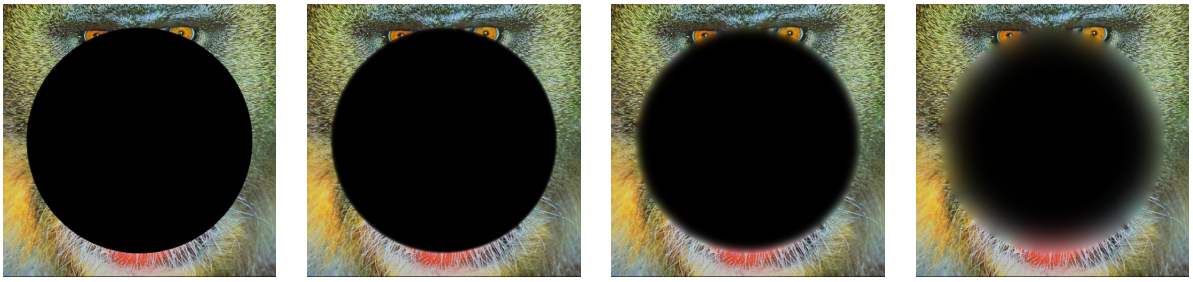
Input (mask and boundary) Output

Figure 10: An example for Laplace equation. The input data is shown on the first image. The region Ω is the set of black pixels, and the boundary condition is taken from the other pixels. The output image is the exact solution obtained by running the method described in this paper with 500 iterations (error less than 10^{-6}).



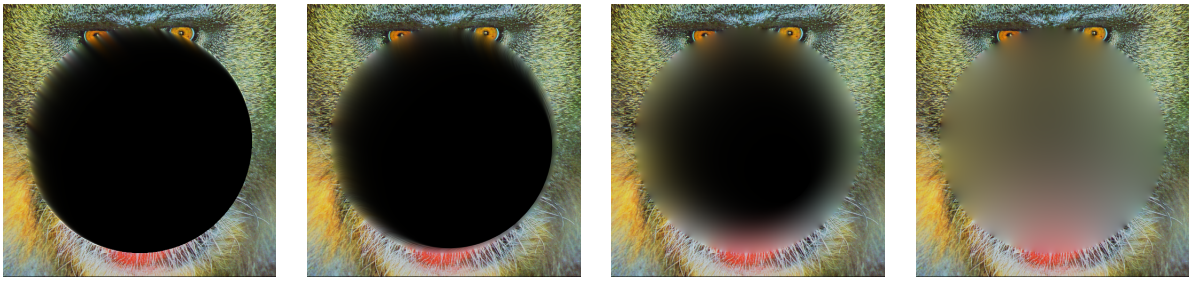
Input (mask and boundary) Input (data term) Output

Figure 11: An example for Poisson equation. The input data is shown on the first two images. The region Ω is the set of black pixels, and the boundary condition is taken from the other pixels. The output image is the exact solution obtained by running the method described in this paper with 500. The output coincides exactly with the original "baboon" image.



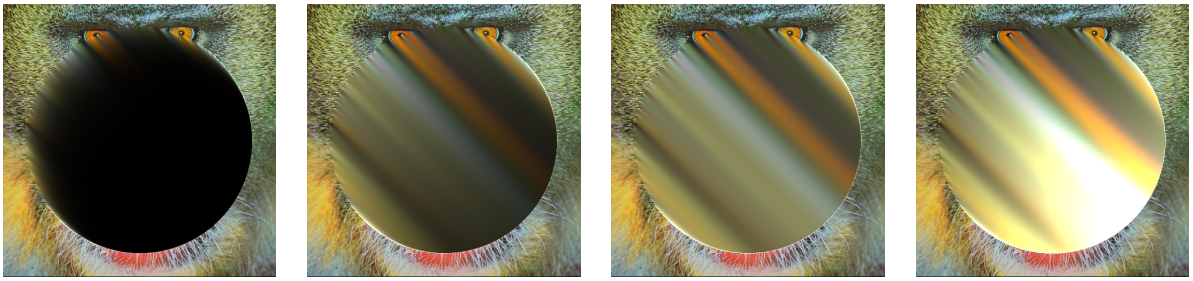
$n = 1$ $n = 10$ $n = 100$ $n = 1000$

Figure 12: Laplace equation. First Gauss-Seidel iterations using the natural time step $\tau = 0.25$. The iterations have been initialized with $u = 0$ inside Ω (the black pixels). Note that the convergence is very slow, and even after 1000 iterations the central part of Ω is still black.



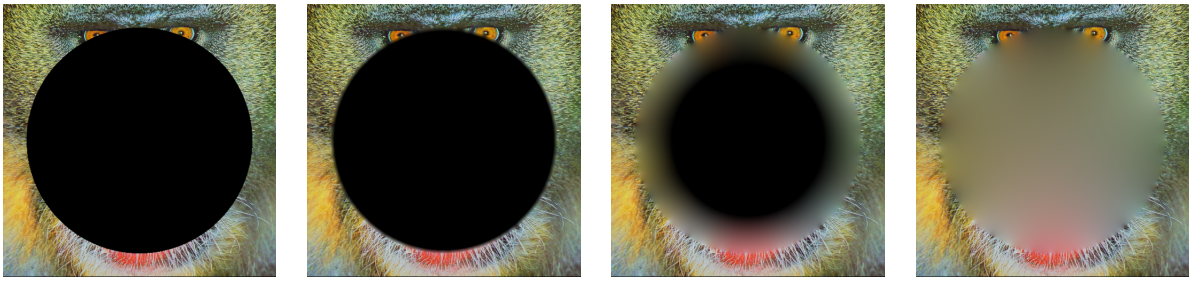
$n = 1$ $n = 10$ $n = 100$ $n = 1000$

Figure 13: Laplace equation. First Gauss-Seidel iterations using an *overshooting* time step $\tau = 0.48$. Note that the convergence is much faster than for $\tau = 0.25$.



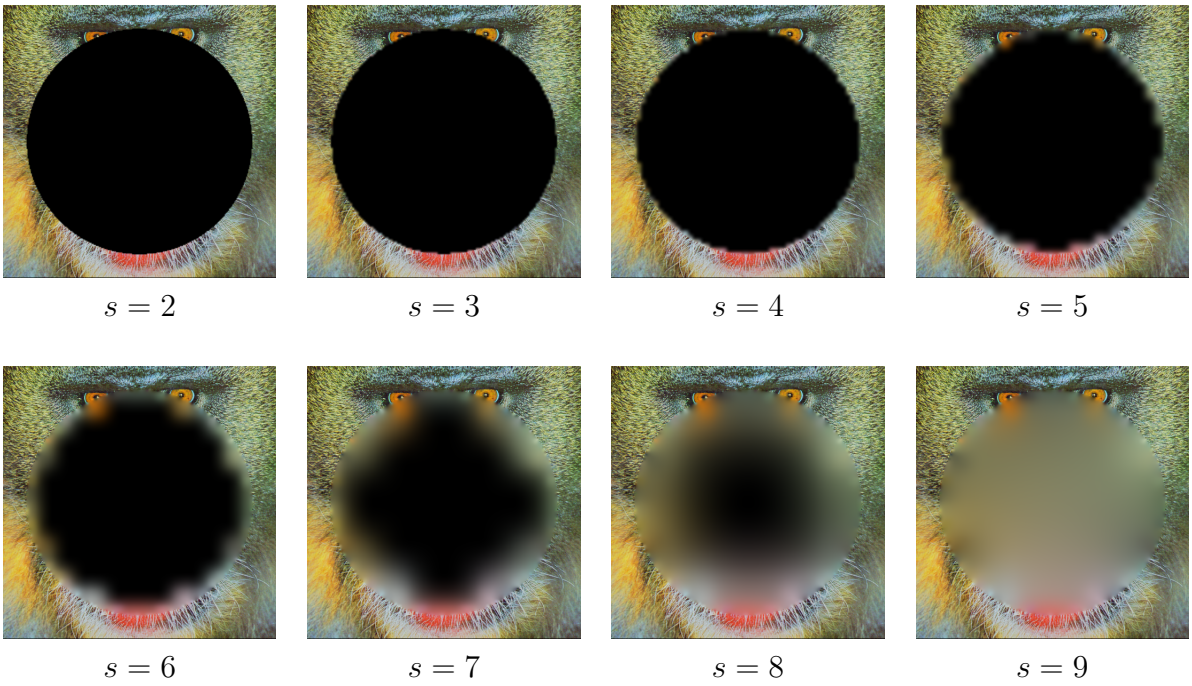
$\tau = 0.49$ $\tau = 0.499$ $\tau = 0.5$ $\tau = 0.501$

Figure 14: The first Gauss-Seidel iteration, using different overshot parameters. If τ is larger, it fills the domain with good values faster, but for $\tau \geq 0.5$ it becomes numerically unstable. For each domain Ω there is an optimal parameter in the interval $(0.48, 0.5)$.



$c = 1$ $c = 10$ $c = 100$ $c = 1000$

Figure 15: Laplace Equation. First Conjugate Gradient iterations. The convergence rate is similar to Gauss-Seidel with the optimal overshot parameter. However, the error for the first hundreds of iterations is much larger.



$s = 2$ $s = 3$ $s = 4$ $s = 5$

$s = 6$ $s = 7$ $s = 8$ $s = 9$

Figure 16: Laplace Equation. Effect of the multi-scale initialization. In these experiments, we perform *zero* iterations of Gauss-Seidel and Conjugate Gradient. Note that with 9 multi-scale levels, we obtain a visually satisfying approximation of the correct solution, even with zero iterations.

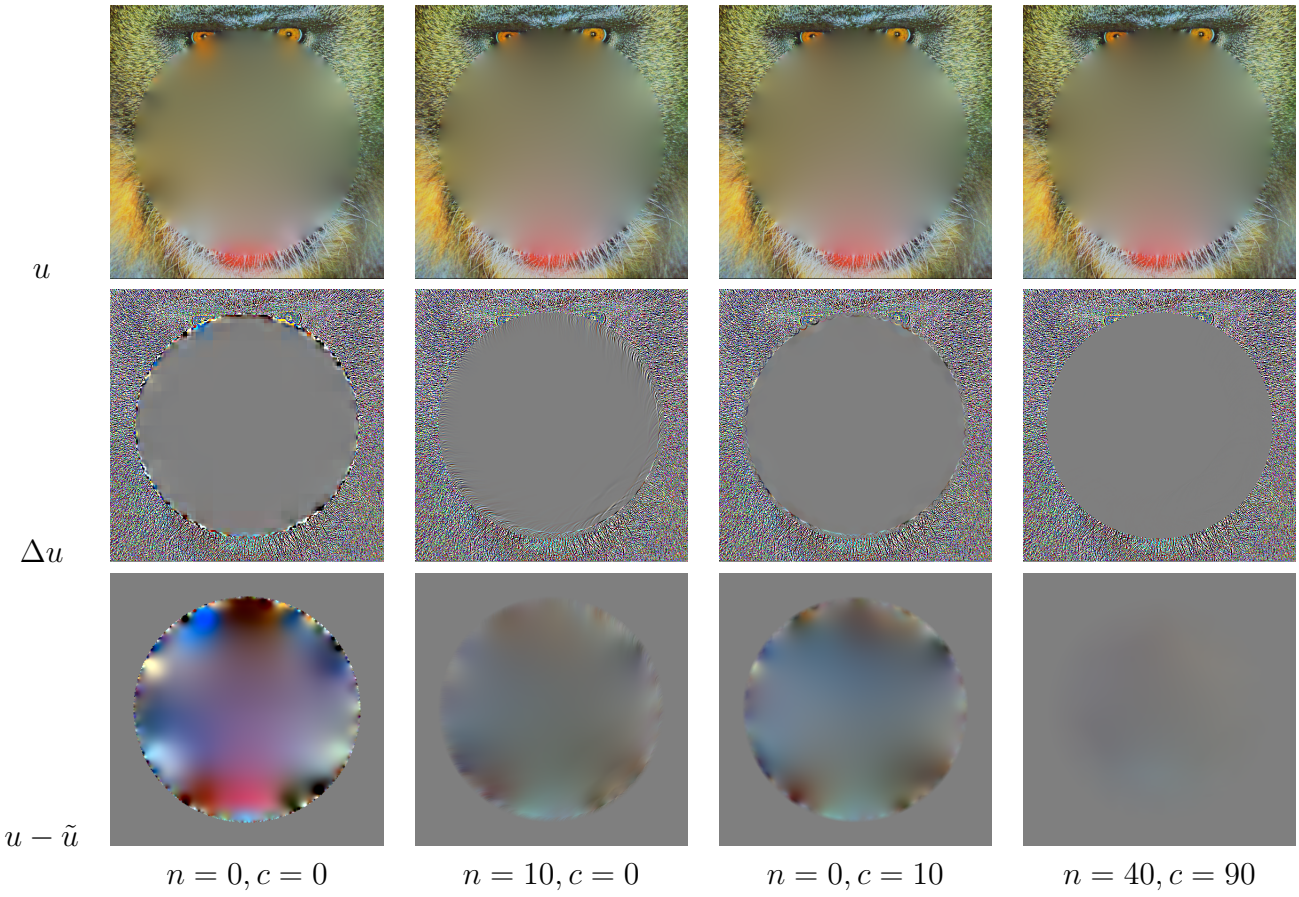
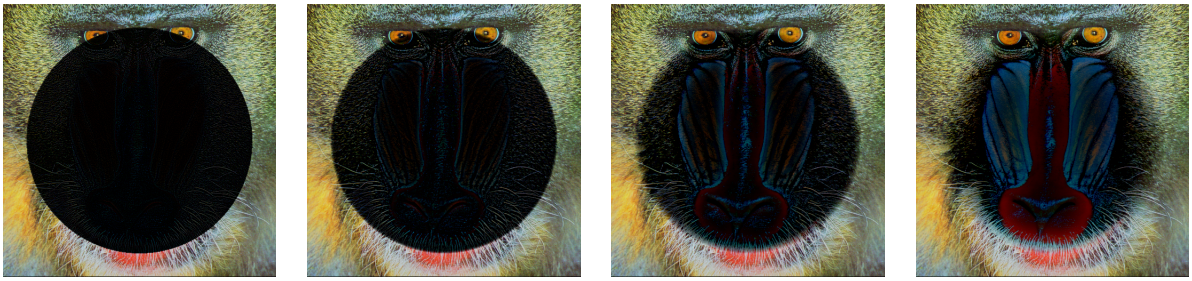


Figure 17: Laplace Equation. Effect of the some parameters of the approximate solutions. Here n = number of Gauss-Seidel iterations and c = number of Conjugate Gradient iterations. For each experience we display the image u, its laplacian Δu with colors on the interval $[-1, 1]$, and the difference with the exact solution $\tilde{u}$, with colors on the interval $[-20, 20]$. In all these experiments we have used $\tau = 0.48$ and a number of scales $s = 9$. Note that the Laplacian is always very small far from the boundary of the domain, even before starting the iterations. When running a few iterations of each method, we obtain and very good approximation. To attain numerically exact convergence for this example we need bout 800 Conjugate gradient iterations.

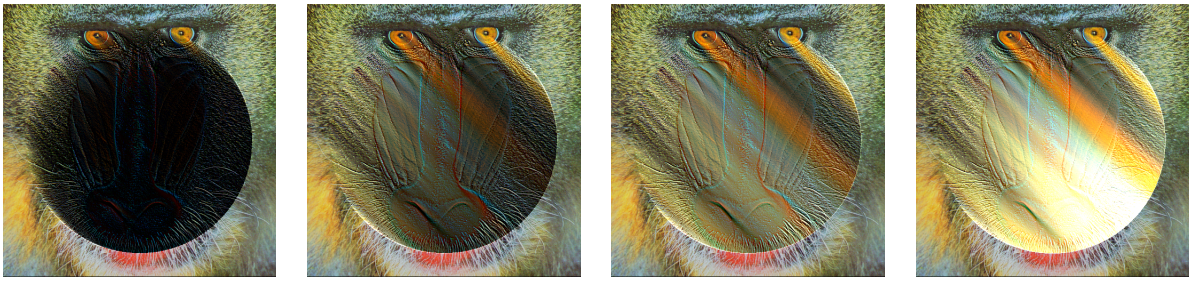


$n = 1$ $n = 10$ $n = 100$ $n = 1000$

Figure 18: Poisson equation. First Gauss-Seidel iterations using the natural time step $\tau = 0.25$. The iterations have been initialized with $u = 0$ inside Ω (the black pixels). Note that the convergence is very slow, and even after 1000 iterations the central part of Ω is still dark. There is an important observation, that can be repeated elsewhere regarding iterative methods for Poisson equation: *the convergence of the high frequencies is fast, and the convergence of the low frequencies is slow.*

$n = 1$ $n = 10$ $n = 100$ $n = 1000$

Figure 19: Poisson equation. First Gauss-Seidel iterations using an *overshooting* time step $\tau = 0.48$. Note that the convergence is much faster than for $\tau = 0.25$.

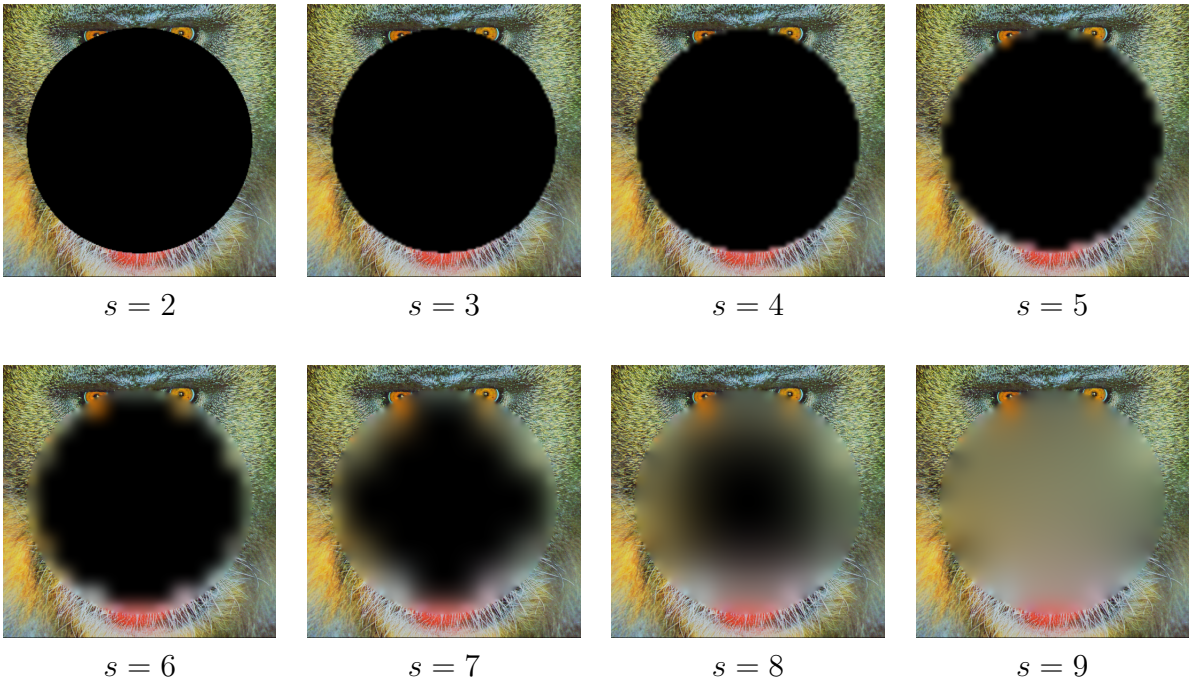


$\tau = 0.49$ $\tau = 0.499$ $\tau = 0.5$ $\tau = 0.501$

Figure 20: The first Gauss-Seidel iteration, using different overshot parameters. If τ is larger, it fills the domain with good values faster, but for $\tau \geq 0.5$ it becomes numerically unstable. For each domain Ω there is an optimal parameter in the interval $(0.48, 0.5)$.

$c = 1$ $c = 10$ $c = 100$ $c = 1000$

Figure 21: Poisson Equation. First Conjugate Gradient iterations. The convergence rate is similar to Gauss-Seidel with the optimal overshot parameter, and much faster than Gauss-Seidel with any other parameter.



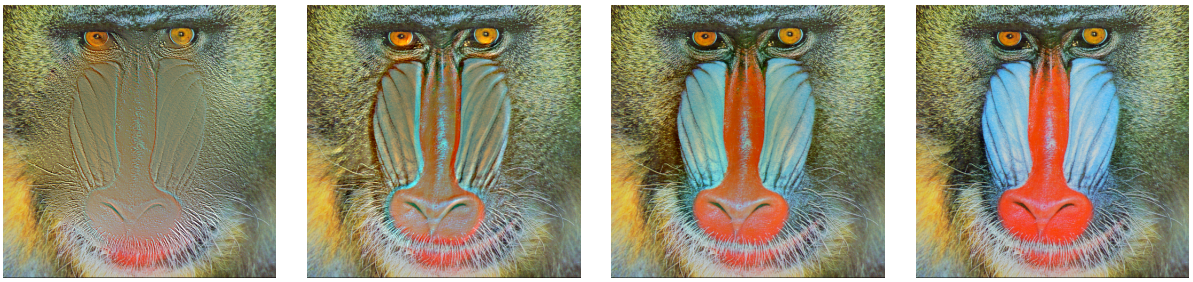
$s = 2$ $s = 3$ $s = 4$ $s = 5$

$s = 6$ $s = 7$ $s = 8$ $s = 9$

Figure 22: Poisson Equation. Effect of the multi-scale initialization. In these experiments, we perform *zero* iterations of Gauss-Seidel and Conjugate Gradient. Note that the results are identical as those for Laplace equation (Figure 16), since we are never using the data term f.

$n = 1$ $n = 10$ $n = 100$ $n = 1000$

Figure 23: Poisson Equation. First Gauss-Seidel iterations with natural timestep $\tau = 0.25$, initialized with the multi-scale solution $s = 9$ from Figure 22. Important observation: the iterations have only been run in the last, full-resolution scale level. Note that the texture is correct but the colors are washed out. This means that the high frequencies are correct and the low frequencies have still not converged.

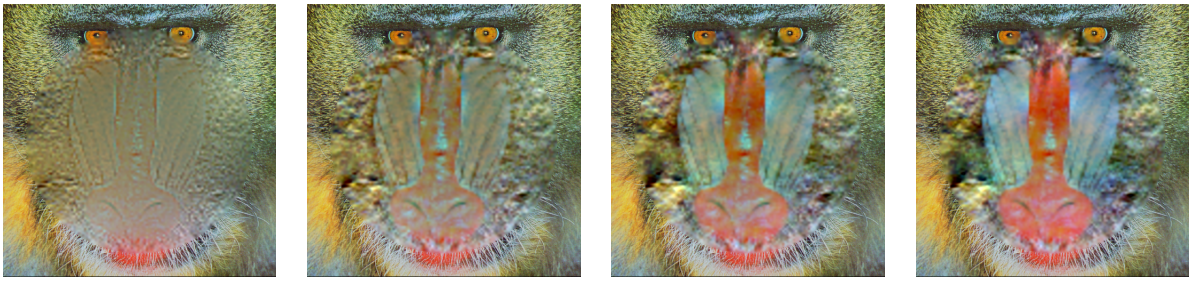


$n = 1$ $n = 10$ $n = 100$ $n = 1000$

Figure 24: Poisson Equation. First Gauss-Seidel iterations with overshooting timestep $\tau = 0.48$, initialized with the multi-scale solution $s = 9$ from Figure 22.

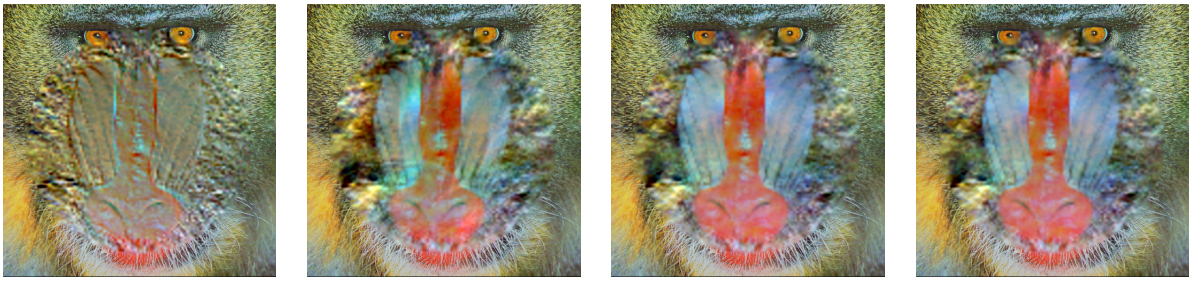
$c = 1$ $c = 10$ $c = 100$ $c = 1000$

Figure 25: Poisson Equation. First Conjugate Gradient iterations, initialized with the multi-scale solution $s = 9$ from Figure 22. The last image coincides exactly (up to the RGB quantization error), with the original baboon image, meaning that the method has converged before 1000 iterations.



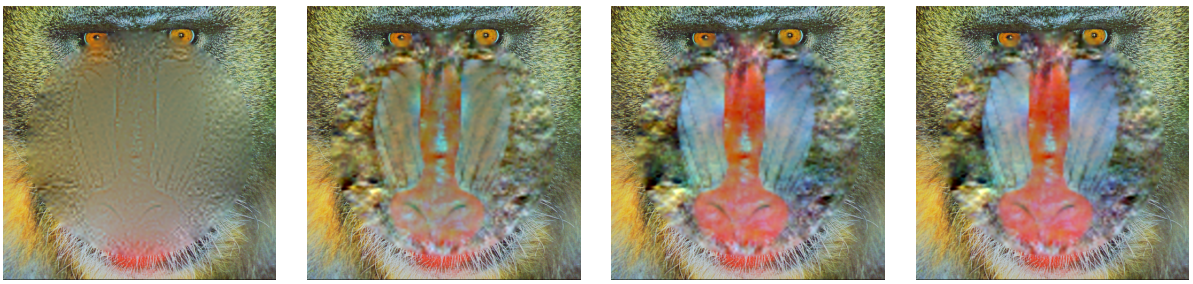
$n = 1$ $n = 10$ $n = 100$ $n = 1000$

Figure 26: Poisson Equation. Gauss-Seidel iterations run only at the scale 1/4, with natural timestep $\tau = 0.25$, There are no iteration at the other scales.



$n = 1$ $n = 10$ $n = 100$ $n = 1000$

Figure 27: Poisson Equation. Gauss-Seidel iterations run only at the scale 1/4, with overshooting timestep $\tau = 0.48$. There are no iteration at the other scales.



$c = 1$ $c = 10$ $c = 100$ $c = 1000$

Figure 28: Poisson Equation. Conjugate Gradient iterations run only at the scale 1/4, with overshooting timestep $\tau = 0.48$. There are no iteration at the other scales.

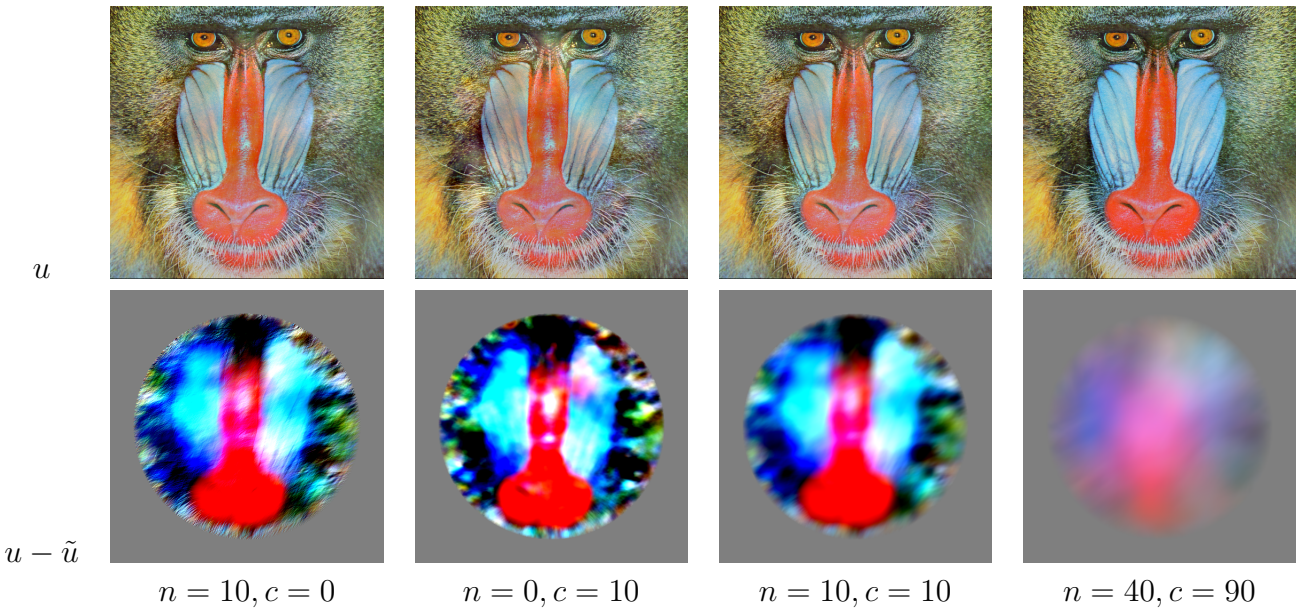


Figure 29: Poisson Equation. Solutions with the same amount of iterations run at every scale level. Here n = number of Gauss-Seidel iterations and c = number of Conjugate Gradient iterations. For each experience we display the image u and the difference with the exact solution $\tilde{u}$, with colors on the interval $[-20, 20]$. In all these experiments we have used $\tau = 0.48$ and a number of scales $s = 9$. Note that, as opposed to Laplace Equation, here we need to run a much larger amount of iterations to obtain a visually acceptable result.

8 Conclusion and future work

Possible extensions: nonzero neumann boundary conditions, riemannian metrics, linear shape from shading, p-laplacian, amle, double laplacian (thin plates), etc.

References

[1] 5, 12

[2] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, *Image inpainting*, in Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 417–424. 6

[3] J. E. Boillat, *Load balancing and poisson equation in a graph*, Concurrency: Practice and Experience, 2 (1990), pp. 289–313. 4

[4] A. Buades, G. Haro, and E. Meinhardt-Llopis, *Obtaining High Quality Photographs of Paintings by Image Fusion*, Image Processing On Line, 5 (2015), pp. 159–175. 10

[5] V. Caselles, J.-M. Morel, and C. Sbert, *An axiomatic approach to image interpolation*, Image Processing, IEEE Transactions on, 7 (1998), pp. 376–386. 2

[6] T. Chan and J. Shen, *Local inpainting models and tv inpainting*, SIAM J. Appl. Math, 62 (2001), pp. 1019–1043. 6

[7] A. Criminisi, P. Pérez, and K. Toyama, *Region filling and object removal by exemplar-based image inpainting*, Image Processing, IEEE Transactions on, 13 (2004), pp. 1200–1212. 6

[8] A. Efros, T. K. Leung, et al., *Texture synthesis by non-parametric sampling*, in Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, vol. 2, IEEE, 1999, pp. 1033–1038. 6

[9] L. C. Evans, *Partial differential equations*, Graduate studies in mathematics, American Mathematical Society, Providence (R.I.), 1998. 2, 3

[10] L. Hoeltgen, S. Setzer, and J. Weickert, *An optimal control approach to find sparse data for laplace interpolation*, in Energy Minimization Methods in Computer Vision and Pattern Recognition, Springer, 2013, pp. 151–164. 7

[11] N. Limare, J.-L. Lisani, J.-M. Morel, A. B. Petro, and C. Sbert, *Simplest color balance*, Image Processing On Line, 1 (2011). 6

[12] L. Moisan, *Periodic plus smooth image decomposition*, Journal of Mathematical Imaging and Vision, 39 (2011), pp. 161–179. 8

[13] G. Strang and K. Aarikka, *Introduction to applied mathematics*, vol. 16, Wellesley-Cambridge Press Wellesley, MA, 1986. 13

[14] G. Strang and G. J. Fix, *An analysis of the finite element method*, vol. 212, Prentice-Hall Englewood Cliffs, NJ, 1973. 3

[15] A. Weiser and M. F. Wheeler, *On convergence of block-centered finite differences for elliptic problems*, SIAM Journal on Numerical Analysis, 25 (1988), pp. 351–375. 3