Kartikesh Mishra December 15 2022

6.8370 Final Project : Deblurring, Composition and Inpainting

1 Introduction

This project is the implementation of the deblurring algorithm presented on 6.815/6.865 Fall 2013 Assignment 8 and using those algorithms to solve inpainting, I see a good practical application of inpainting to computationally protect copyrighted places from getting captured on camera by accident. We can use some QR codes for the camera to sense and mask the copyrighted region and digitally inpaint over it. Also we can remove unwanted reflections and repair corrupted images. Deblurring can be used almost always to refine the image quality. This project achieves deblurring by solving deconvulation equation using conjugate gradient descent method. The same method is used to solve poisson equations for composition and inpainting.

The poisson equations are taken from [1] Perez et Al. which describes generic image interpolation techniques based on solving Poisson equations and presents tools for seamless importation of both opaque and transparent source image regions into a destination region and allows the user to modify the appearance of the image seamlessly, within a selected region. For inpainting, there can be another method using fast marching method presented in [4] Telea et. al. which is also used in a very popular image editing library openCV as inpainting method. Also, in [6] Facciolo et. al., poisson equation with drichilet boundary condition is solved for image editing applications in inpainting by smooth function and poisson editing.

2 Mathematical Explanation

2.1 Deblurring

We know blurring is a convolution of the image with a blur kernel. Let Y be blur image, X be original image and K be blur kernel.

$$Y = X \otimes K \tag{1}$$

Which can also be represented by matrix multiplication. i.e.for some matrix M depending on K.

$$Y = MX \tag{2}$$

Given Y, there can be two different kind of deblurring. One to estimate a conjugate kernel $K^{'}$ or M^{-1} such that $X = M^{-1}Y = Y \otimes K^{'}$ and the other is non-blind deblurring, which is what I am going to discuss, and implement in this project where we know the kernel K and we solve for X directly. In either case we need a energy function to detrmine how close we are to the solution. I present here an energy functions for non-bind deblurring.

$$G(X) = ||MX - Y||^2 \implies X = argmin_X G(X) \tag{3}$$

$$\frac{d}{dX}(G(X)) = M^T MX - M^T Y = -r(X) \tag{4}$$

To find such X where residual r(X) is almost 0 i.e. setting the first derivative of convex quadratic energy function to 0 to minimize it, it can be approached by using conjugate gradient descent method (faster than gradient descent) which is by taking an initial assumption of X and updating it in each iteration.

2.2 Gradient Descent

Start with $X_0 = 0$ and use iteration to find X which minimizes G. It can be done for a fixed number of iteration or until $r(X)$ is very close to 0.

$$X_{i+1} = X_i + \alpha_i r_i \tag{5}$$

Here, α_i is a scalar step size chosen such that $\frac{d}{d\alpha}(G(X_{i+1})) = 0$ i.e. minimizes energy function with iteration.

$$\alpha_i = \frac{r_i^T r_i}{r_i^T M^T M r_i} \tag{6}$$

2.3 Conjugate Gradient Descent

In gradient descent most of times we repeat the direction and thus, slow convergence, in this method, we choose our update direction orthogonal to previous ones. i.e.

$$X_{i+1} = X_i + \alpha_i d_i \implies r_{i+1} = r_i - \alpha_i M^T M d_i \tag{7}$$

$$d_0 = r_0, d_{i+1} = r_{i+1} + \beta_i d_i \tag{8}$$

$$\alpha_i = \frac{r_i^T r_i}{d_i^T M^T M d_i} \tag{9}$$

$$\beta_i = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i} \tag{10}$$

Here α is computed to minimize energy function over iteration and β using orthogonality constraints.

2.4 Regularized Conjugate Gradient Descent

To reduce the noise while deconvolutionizing, we can regularize the noise using laplacian kernel, let L be matrix representing it then we update energy function by.

$$G(X) = ||MX - Y||^2 + \lambda ||LX||^2 \tag{11}$$

Equations (7) and (9) should be updated accordingly.

2.5 Compositing

Here, we have a source image S, a destination image D and a region $\Omega \subset \mathbf{R}^2$ and we have to seamless composite S on Ω of D which is also not easy task to achieve so instead we can work on minimize the energy function clearly being sum of square differences in gradient of S and D over Ω. Let X be the desired D over Ω and Y be S over Ω. Also let G be matrix representing gradient operator.

$$E(X) = ||GX - GY||^2 \tag{12}$$

$$\frac{d}{dX}(E(X)) = G^T GX - G^T GY = -r(X) \tag{13}$$

Since G represents gradient, $G^T G$ represents laplacian and thus $G^T G = L$ Now we can use conjugate gradient descent to find X so that E(X) is minimized.

2.6 Inpainting

In this problem, we have an image I and a mask region $\Omega \subset \mathbf{R}^2$ where I should be interpolated from outside Ω. It can be achieved by composition using I as D and a flat surface 0 as S. Another way can be to use gradients in both x and y direction of I to guide the interpolation which can be done either solving another poisson equation or using fast marching method or using anisotropic diffusion. Poisson equations are similar to composition.

3 Algorithm Implementation

3.1 Conjugate Gradient Descent

```
def dotIm(im1, im2):
  return np.sum(im1*im2,axis=0)
def applyA(x, kernel):
  return applyConjugatedKernel(applyKernel(x,kernel),kernel)
def conjugateGD(y, kernel, niter, applyfunc =applyA, mask =1, bg=None):
  x = (1-mask)*y
  r = applyConjugatedKernel(y)
  d = r
  for _ in range(niter):
    Ad = applyfunc(d,kernel)
    r_dot = dotIm(r,r)
    alpha = r_dot/dotIm(d,Ad) #stepSize
    x = x + alpha*d
    r = r - alpha*Ad
    beta = dotIm(r,r)/r_dot
    d = r + beta*d
  return x
```

3.2 Deblurring

```
def deBlur(im_blur, kernel):
  return conjugateGD(im_blur, kernel, 100)
#for regularized deblurring in conjugateGD, set
#    applyfunc = applyRegularizedOperator
def applyRegularizedOperator(x, kernel, lam):
  return applyA(x,kernel) + lamb*applyKernel(x,laplacianKernel)
```

3.3 Compositing

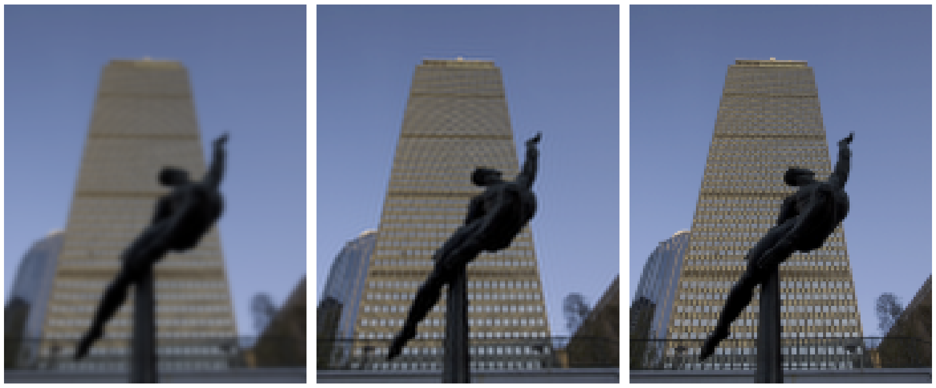
```
#in conjugateGD, replace
#    x =(1-mask)*y by  x =(1-mask)*bg
def composite(bg, fg, mask):
    return conjugateGD(fg, laplacianKernel, 100, applyKernel, mask, bg)
```

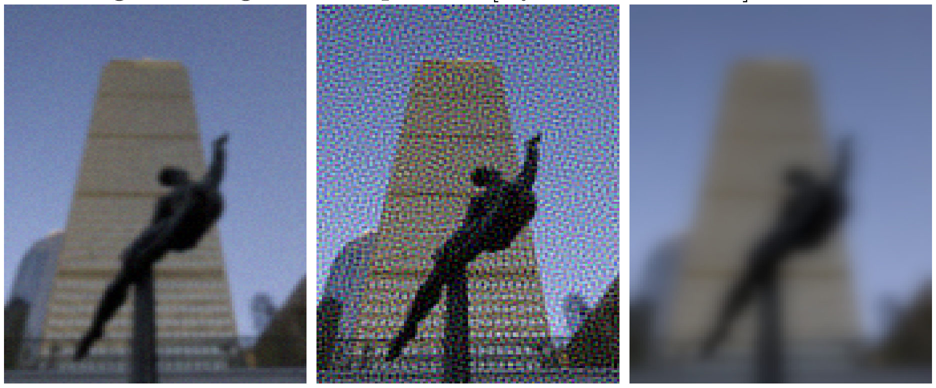
3.4 Inpainting

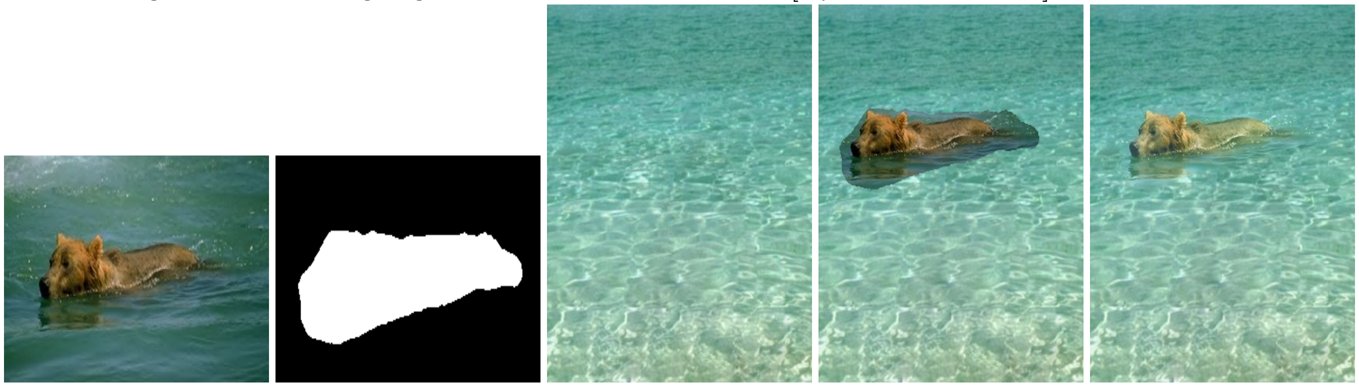
```
def inpaint_poisson(im, mask):
    flat = zeros_like(im);
    return composite(im,flat ,mask)

def inpaint_structure(im, mask):
    flat = zeros_like(im);
    lum, chrom = lumChrom(im)
    tensor = computeTensor(lum)
    tensor_new = composite(tensor ,flat ,mask)
    return color_interpolation(tensor_new , im, mask)

def inpaint_l2l(im, mask):
    w, h, c = im.shape
    m = flatten(mask)
    band = []
    masked = (1−mask)∗im
    Ix = gradient(masked)
    Iy = gradient(masked)
    for (x in range(w∗h∗c)):
        if (m[x]==0 and (m[x−1]==1 or m[x+1]==1 or m[x−w] ==1 or m[x+w]==1):
            band.append(x)
            updateGradient(m,x,Ix,Iy, w, h)
    while band:
        p = band.pop()
        for ((k,l,u,z) in getNeighbours(p,1)):
            if (m[u]==1):
                updateGradient(m,k,l,Ix,Iy, w, h)
                inpaint_point(k,l,z, masked, Ix, Iy)
                band.append(u)
```

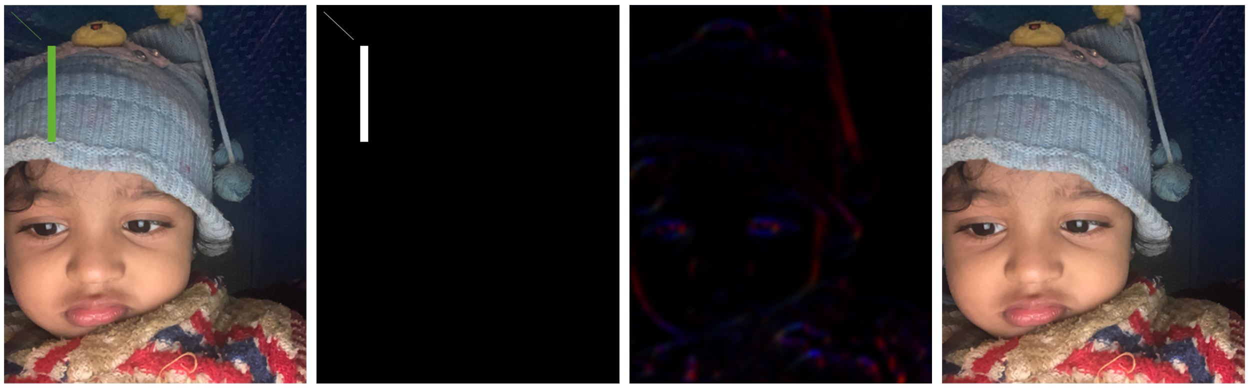
While implementing be clear on dotIm and other helper functions. I misimplemented dotIm and to find the bug, it took me a lot of time, which was most difficult part in this project.

4 Intermediate Results


Blurred image followed by deblurred using 20 iteration of conjugated gradient method and then original image for comparison [Python code used]


Blurred image+noise followed by deblurred using 20 iteration of conjugated gradient method without reg and then using reguarization for 20 iteration [Python code used]


An image of bear in water followed by mask, waterpool, naive composition [python], poisson composition using 100 iteration of conjugated gradient descent [C++]



Above pictures are a masked picture followed by mask, structure tensor and result obtained after doing inpainting.

Refrences

1 Patrick Perez, Michel Gangnet, Andrew Bake,Poisson Image Editing, https://www.cs.jhu.edu/ misha/ Fall07/Papers/Perez03.pdf

2 https://stellar.mit.edu/S/course/6/fa13/6.815/homework/assignment10/handout/a8.pdf

3 http://en.wikipedia.org/wiki/Inpainting

4 Alexandru Telea, An Image Inpainting Technique Based on the Fast Marching Method, http://www.olivier-augereau.com/docs/2004JGraphToolsTelea.pdf

5 https://en.wikipedia.org/wiki/Anisotropic_diffusion

6 Enric Meinhardt-Llopis, Gabriele Facciolo, A Simple Poisson Solver for Image Processing, http://dev.ipol.im/ coco/website/docs/simpois.pdf