SIMON FRASER UNIVERSITY

Image Inpainting

Ali Mahdavi Amiri, Sepideh Sarajian

September 22, 2021

1 IMAGE RECONSTRUCTION (8 POINTS)

We already implemented image reconstruction from first-order derivatives by solving a least squares problem in the tutorial. This time we reconstruct the image by referring to its second-order derivatives. We denote the source image as S and the reconstructed image as V. Denote the intensity of the pixel (x, y) in image S as $s_{(x,y)}$ and the pixel in image V as $v_{(x,y)}$. Note that S is given so $s_{(x,y)}$ are constants, and $v_{(x,y)}$ are variables. To obtain variables $v_{(x,y)}$, we need to solve the equations (for each pixel):

$$V^g_{up} + V^g_{down} + V^g_{left} + V^g_{right} = S^g_{up} + S^g_{down} + S^g_{left} + S^g_{right} \tag{1.1}$$

Where $V^g_{left} = v_{(x,y)} - v_{(x-1,y)}$, $V^g_{right} = v_{(x,y)} - v_{(x+1,y)}$, $V^g_{up} = v_{(x,y)} - v_{(x,y-1)}$, $V^g_{down} = v_{(x,y)} - v_{(x,y+1)}$ are the gradients of $v_{(x,y)}$ in the four directions. The others are defined similarly, for example $S^g_{down} = s_{(x,y)} - s_{(x,y+1)}$. In order to compute it, we have:

$$4 \times v_{(x,y)} - v_{(x+1,y)} - v_{(x-1,y)} - v_{(x,y+1)} - v_{(x,y-1)} = S^g_{up} + S^g_{down} + S^g_{left} + S^g_{right} \tag{1.2}$$

Where the red part is made of variables, and the blue part is a constant. By representing the coefficients of V as a matrix A and the constants as b, we have $AV = b$. If the image has k pixels, V will be a $k \times 1$ matrix, A will be $k \times k$, and b will be $k \times 1$.

Note: If a pixel is an edge pixel, for example $(1,2)$, it does not have a left neighbor, therefore it does not have horizontal second-order derivative, so in the equation you only need to consider vertical gradients ($V^g_{up} + V^g_{down} = S^g_{up} + S^g_{down}$). If a pixel is a corner pixel, it has no second-order derivative ($0 = 0$). We have four corner pixels, so there will be four all-zero rows in A and the corresponding four zeros in b. This is why we cannot get the reconstructed image by solving the current equations. We need to provide four "control points" to make A has

Figure 1.1: Sample results for image reconstruction.

rank k so that solving $AV = b$ will give us a unique solution. The four "control points" control the global lightness and the global gradients. For this assignment, you need to choose the four corner points as the four "control points" for better control, i.e., $(1,1)$, $(1,n)$, $(m,1)$ and (m,n).

$$\begin{aligned} v_{(1,1)} &= s_{(1,1)} \\ v_{(1,n)} &= s_{(1,n)} \\ v_{(m,1)} &= s_{(m,1)} \\ v_{(m,n)} &= s_{(m,n)} \end{aligned} \tag{1.3}$$

Considering the four extra constraints, V will be a $k \times 1$ matrix, A will be $(k+4) \times k$, and b will be $(k+4) \times 1$. You may make A $k \times k$ by removing the four all-zero rows, but it is not necessary since in any case we can solve the equations.

You need to write down and solve the equations using $A = sparse(i, j, v)$ (6 points), and then copy those variable pixels to the appropriate positions in the output image (2 points). The framework is given (same to the one used in the tutorial).

Note: As a validation, print the error $sum(abs(AV - b))$ after you get the solution V. If your implementation is correct, this error should be very small (like $1e^{-11}$ or $1e^{-12}$), no matter what values you have assigned to the control points.

1.1 Report

Since you have 4 "control points" now, you can manipulate the ground truth values to control the global lightness and gradients. Show 5 reconstructed images in the report: one similar to the ground truth, one globally brighter, one brighter on the left side, one brighter on the bottom side, and one brighter on right bottom corner. Some examples are shown in Figure 1.1.

Figure 2.1: Poisson image blending sample from [1].

2 Poisson blending (12 Points)

In this section, you will implement Poisson blending [1]: given an image patch in the source image and its destination in the target image, Poisson blending can change the color of this patch but retain its content so that it can seamlessly blend into the target image.

The provided matlab code already gives you a framework that allows you to:

First, load a source image and select a region you want to copy. You can click multiple times to specify the boundaries of your patch, then press "Q" to get a closed patch.

Second, load a target image and place your patch onto the image. Press "W"/"S" to resize and "A"/"D" to rotate. Press "Q" to finish.

Tips: try to select the boundaries in the featureless regions for both images, e.g. sky.

You need to complete poisson_blend.m. In the code you are given the source image S, the target image T and the Mask M, where in M a pixel is 1 if it is in the selected region and 0 otherwise. Your task is to implement Poisson blending, which is to solve the equations (for each pixel in the selected region):

$$V^{tg}_{up} + V^{tg}_{down} + V^{tg}_{left} + V^{tg}_{right} = S^{g}_{up} + S^{g}_{down} + S^{g}_{left} + S^{g}_{right} \tag{2.1}$$

This time, we only consider the pixels in the selected region. Suppose we have k pixels in the selected region (that means mask M has k non-zero entries), In $AV = b$, V will be a $k \times 1$ matrix, A will be $k \times k$, and b will be $k \times 1$. No extra constraint is required.

V^{tg} is defined according to the mask. For (x, y), if $(x-1, y)$ is in the selected region, we have $V^{tg}_{left} = (v_{(x,y)} - v_{(x-1,y)})$; if $(x-1, y)$ is outside, $V^{tg}_{left} = (v_{(x,y)} - t_{(x-1,y)})$. The gradients for the other directions are defined in a similar way.

After implementing *Image reconstruction*, it should be straightforward to implement poisson blending. For each pixel in the selected region, consider its four neighbors. If a neighbor is inside the selected region, treat it as a variable; if it is outside, treat it as a constant defined by T.

You need to write down and solve the equations using $A = sparse(i, j, v)$ (8 points), and then copy those variable pixels to the appropriate positions in the output image to obtain the blended image (2 points). For RGB images, process each channel separately and combine the results together in the end. Write this in an recursive or iterative way so that poisson_blend can handle both grayscale and color images (2 points).

Note: As you can see, this time A again is a full rank matrix, therefore solving $AV = b$ will give us a unique solution. Print the error $sum(abs(AV - b))$ after you get the solution V. If your implementation is correct, this error should be very small.

2.1 Report

Show 3 pairs of blending results in the form of Figure 2.1. You can just blend two images instead of three. No need to draw the lines, just show the source, the target, the cloning and the blending result. Choose your images from the internet. Do not use the images I provided, be creative :)

References

[1] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on graphics (TOG)*, 22(3):313–318, 2003.