Homework Assignment 3

15-463/663/862, Computational Photography, Fall 2017
Carnegie Mellon University

Due Thursday, Oct. 12, at 11:59pm

The purpose of this assignment is to explore gradient-domain processing, and in particular Poisson blending [2]. As we discussed in class, this is a technique that allows seamlessly blending an object or texture from a source image into a target image. While the assignment and the class slides describe most of the steps you need to perform, it is highly recommended that you read the associated paper. There is a "Hints and Information" section at the end of this document that is likely to help.

1. Implement Poisson Blending (100 points)

As we discussed in class, the main insight behind Poisson blending is that people often care much more about the gradient of an image than the overall intensity. So, we can try to do blending by finding values for the area to be blended that maximally preserve the gradient of the source region without changing any of the background pixels of the target image. Note that we are making a deliberate decision here to ignore the overall intensity! So a green hat could turn red, but it will still look like a hat.

For an example of this, consider the picture of the bear and swimmers being pasted into a pool of water, shown in Figure 1. Let's ignore the bear for a moment (it's a non-threatening bear) and consider the swimmers. We will denote by s the source image, that is, the original image the swimmers were cut out of. That image isn't even shown, because we are only interested in the cut-out of the swimmers, which we denote as the region S. S includes the swimmers and a bit of light blue background. You can clearly see the region S in the left image, as the cutout blends very poorly into the pool of water. The pool of water, before things were rudely pasted into it, is the target image t.

How do we blend in the swimmers? We construct a new image v whose gradients inside the region S are similar to the gradients of the cutout we're trying to paste in (swimmers and a little bit of background). Outside S, v will match the pool. We won't even bother computing the gradients of the pool outside S; we'll just copy those pixels directly.

We can formulate the above as a least squares problem, as follows:

$$\boldsymbol{v} = \underset{\boldsymbol{v}}{\operatorname{argmin}} \sum_{i \in S,\ j \in N_i \cap S} \left((v_i - v_j) - (s_i - s_j)\right)^2 + \sum_{i \in S,\ j \in N_i \cap \neg S} \left((v_i - t_j) - (s_i - s_j)\right)^2. \tag{1}$$

In the first summation of Equation (1), we set the gradients of v inside S. We loop over all the pixels inside the region S, and require that the new image v have the same gradients as the swimmers. The summation is over every pixel i in S; j is the 4 neighbors of i (left, right, up, and down), giving us both horizontal and vertical gradients. The second summation takes care of the boundary around S. In this case, i is inside S, but j is outside. In this case we are not solving for a pixel intensity v_j, since j is not inside S. So we just insert the intensity value t_j from the target image, which we know since we are not modifying that part of the image. The gradients will not end up matching exactly: the least squares solver will take any hard edges of the cutout at the boundary and smooth them by spreading the error over the gradients inside S.

Toy Problem (20 points). The implementation for gradient domain processing is not complicated, but it is easy to make mistakes, so you will start with a toy example using the image `toy_problem.png` in the `./data` directory of the homework ZIP archive. In this example, you will compute the x and y gradients of an image s, then use these the gradients, plus one pixel intensity, to reconstruct an image v. If all is done correctly, the reconstructed image should match the original one.

In particular, denote the intensity of the source image at (x, y) as $s(x, y)$, and the values of the image to solve for as $v(x, y)$. For each pixel, then, we have two objectives:

1. Minimize the difference between the x-gradients of v and the x-gradients of s, that is,

$$\left((v(x+1, y) - v(x, y)) - (s(x+1, y) - s(x, y))\right)^2. \tag{2}$$

Figure 1: Pool example for Poisson blending.

2. Minimize the difference between the y-gradients of v and the y-gradients of s, that is,

$$((v(x, y+1) - v(x, y)) - (s(x, y+1) - s(x, y)))^2 . \tag{3}$$

Note that the above objectives could be solved while adding any constant value to v, so you will need to add one more objective:

3. Minimize the difference between the colors of the top left corners of the two images,

$$(v(1, 1) - s(1, 1))^2 . \tag{4}$$

Combine all of the above objectives into a single least-squares problem, expressed in matrix form:

$$(\boldsymbol{A}\boldsymbol{v} - \boldsymbol{b})^2 , \tag{5}$$

where $\boldsymbol{A}$ is a sparse matrix, $\boldsymbol{v}$ are the variables to be solved, and $\boldsymbol{b}$ is a known vector. Then, solve this problem using one of Matlab's solvers and compare the result to the original image. (See help for the \ operator)

Poisson blending (50 points.) It is now time to implement Poisson blending. For this, you will use the images `hiking.png`, `penguin.png`, and `penguin-chick.png` included in the `./data` directory of the homework ZIP archive. You will insert each of the two penguin images on top of the snow in the hiking image. You can select where exactly the insertion region will be. Given that the images are RGB, you should process each color channel separately. The file `starter.m`, provided in the `./src` directory of the homework ZIP archive, contains some starter code for this part. Its use is optional.

We can split implementation into three steps.

1. Select the boundaries of a region in the source image and specify a location in the target image where it should be blended. Then, transform (e.g., translate) the source image so that indices of pixels in the source and target regions match. For help with this, you can use the functions `getMask.m` and `alignSource.m` provided in the `./src` directory of the homework ZIP archive. You may want to augment the code to allow rotation or resizing into the target region. You can be a bit sloppy about selecting the source region: The penguin selection can be done very crudely, with lots of room around them—just make sure that you are kind to the penguins and you do not chop off parts of them.

2. Formulate and solve the blending constraints expressed in Equation (1).

3. Copy the solved values v_i into your target image.

Blending with mixed gradients (10 points.) Follow the same steps as Poisson blending, but use the gradient in source or target with the larger magnitude as the guide, rather than the source gradient. That

is, solve the least squares problem:

$$\boldsymbol{v} = \underset{\boldsymbol{v}}{\operatorname{argmin}} \sum_{i \in S,\ j \in N_i \cap S} ((v_i - v_j) - d_{ij})^2 + \sum_{i \in S,\ j \in N_i \cap \neg S} ((v_i - t_j) - d_{ij})^2 , \tag{6}$$

where d_{ij} is whichever of the source or the target gradients has the largest magnitude,

$$d_{ij} = \begin{cases} s_i - s_j & \text{if } |s_i - s_j| > |t_i - t_j| , \\ t_i - t_j, & \text{otherwise.} \end{cases} \tag{7}$$

Your own examples (20 points.) Apply Poisson blending and the mixed-gradients variant to your own examples. Show at least three results for each. Explain any failure cases (e.g., weird colors, blurred boundaries, etc.). We discussed some failure cases in class, so you can try to reproduce those. For mixed gradients, one possible example is blending a picture of writing on a plain background onto another image.

The total number of points you will get from this question will depend on how visually compelling your blending results are.

2. Bonus: Implement a different gradient-domain processing algorithm (up to 50 points)

Gradient-domain processing is a general image processing paradigm that can be used for a broad set of applications, including blending as above, but also tone-mapping, colorization, converting to grayscale, edge enhancement, image abstraction and non-photorealistic rendering (we saw a toy example of this when discussing median filtering), and so on.

GradientShop [1] is a paper and associated app that provide a unifying framework for expressing all these tasks as gradient-domain processing algorithms. Read this paper, and implement whichever of the applications described within appeal to you the most (the instructor's personal favorite is non-photorealistic rendering, but it won't reflect badly on your grade if you go for something else).

Your implementation does not need to follow the paper exactly. For instance, instead of implementing the "long-edge detector" described in the paper, you can come up with your own detector of continuous edges.

In your submitted report, you should make sure to describe in detail exactly what you did. This includes what task you chose to implement, where you followed and where you deviated from GradientShop, and in the latter case what you did differently. You should also show at least three examples showcasing the effects of your gradient-domain processing pipeline.

Deliverables

As described on the course website, solutions are submitted through Canvas. Your solution should be an archive (e.g., a ZIP file) that includes the following:

- A PDF report explaining what you did for each problem, including answers to all questions asked throughout Problem 1, as well as any of the bonus problems you choose to do. The report should include any figures and intermediate results that you think may help. Make sure to include explanations of any issues that you may have run into that prevented you from fully solving the assignment, as this will help us determine partial credit. The report should also explain any additional image files you include in your solution (see below).
- All of your Matlab code, including code for the bonus problems, as well as a `README` file explaining how to use the code.
- Image files showing the results of inserting the penguin images into the hiking image, using Poisson blending and mixed gradients, as well as the results of applying these techniques to your own chosen examples. In the latter case, make sure to include the original source and target images as well. You can also include additional image files for various experiments (e.g., using different crops and insertion regions) other than your final ones, if you think they show something important.

- If you do Bonus Problem 2: The original and corresponding processed image files you used.

Please organize your solution submission using the following file structure:

```
.zip
├── .pdf ..... The PDF report.
├── src/ ..... Contains all Matlab M-files and the README file explaining how to use the code.
└── data/ ..... Contains all image, video, and other data files.
```

Hints and Information

- To solve the toy problem, it is helpful to keep a matrix "im2var" that maps each pixel to a variable number, for example,

```
[imh, imw, nb] = size(im);
im2var = zeros(imh, imw);
im2var(1:imh*imw) = 1:imh*imw;
```

  If you find this Matlab trickery confusing, consider that you could have performed the mapping between pixel and variable number manually each time. For example, the pixel at $s(r, c)$, uses the variable number $(c-1) * imh + r$. While creating `im2var` may seem unnecessary for the toy problem, this trick will come in handy for the Poisson blending problem, where the mapping is from an arbitrarily-shaped block of pixels and will not be such a simple function. Therefore, it helps to understand the mappint trick now.

  Given this, you can create the rows of matrix $\boldsymbol{A}$ corresponding to the objective of Equation (2) as,

```
e=e+1;
A(e, im2var(y,x+1))=1;
A(e, im2var(y,x))=-1;
b(e) = s(y,x+1)-s(y,x);
```

  Here, `e` is used as an equation (matrix row) counter. Note that, confusingly, the y-coordinate is the first index in Matlab convention. The objectives of Equations (3) and (4) can be written similarly.

- As always when working with Matlab, it is important to initialize ("preallocate") large variables. In this case, you can use the following to initialize a sparse matrix with M equations (rows), N variables (columns), and at most $nzmax$ non-zero entries:

```
A = sparse([], [], [], M, N, nzmax);
```

Credits

Most of the write-up for this assignment came from previous offerings of this class.

References

[1] P. Bhat, C. L. Zitnick, M. Cohen, and B. Curless. Gradientshop: A gradient-domain optimization framework for image and video filtering. *ACM Transactions on Graphics (TOG)*, 29(2):10, 2010.

[2] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. In *ACM Transactions on graphics (TOG)*, volume 22, pages 313–318. ACM, 2003.