Closed-Form Matting and Astrophotography

Byron Xu

1 Introduction

One of the classes I took this semester was 12.409 Hands-On Astronomy, which introduced me to astrophotography. I learned that by using long camera exposures, we can detect night sky objects much fainter than the eye can see. Furthermore, by stacking several exposures, we can increase the signal-to-noise ratio.

However, there are a few challenges. Without a powerful (and expensive) telescope, pictures of almost any particular object will look like an unrecognizable dot. The most interesting pictures that we can take are wide-angle photos of lots of stars in the night sky. Moreover, stars appear to move across the sky due to the Earth's rotation. If we take too long of an exposure, stars will start forming streaks instead of dots. This forces us to take many short exposures, align, and stack them.

Good photos of the night sky usually include some of the ground in addition to the stars. However, the ground doesn't move with the stars. If we stack all images with the ground in the same place, the stars form streaks. If we align the stars, the ground becomes blurry. In order to get around this problem, I used image matting to separate the ground (foreground) from the night sky (background).

2 Background

3 papers that were related to my work:

- A. Levin, D. Lischinski and Y. Weiss, "A Closed-Form Solution to Natural Image Matting," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 228-242, Feb. 2008. doi: 10.1109/TPAMI.2007.1177

  Levin et. al. was the source of the matting algorithm that makes up the primary part of my project. Using assumptions that pixels within the foreground or background vary smoothly, the authors create a quadratic cost function for an image matte, which can be minimized by setting the derivative equal to zero. The authors give a closed-form matrix expression for this, which can be solved as a sparse system of linear equations.

- Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A Bayesian Approach to Digital Matting. In *Proceedings of IEEE Computer Vision and Pattern Recognition (CVPR 2001)*, Vol. II, 264-271, December 2001

  The work of Chuang et. al. precedes the previous paper. The authors create a matting technique using Bayesian inference. Color distributions of known foreground and background regions are computed, and a maximum-likelihood estimate is made for the foreground opacity at each point based on these color distributions.

- Hroch, Filip. (2000). The robust detection of stars on CCD images. *Experimental Astronomy*. 9. 10.1023/A:1008195518637.

  Hroch's paper describes techniques for detecting stars in telescope images. Detecting stars is harder than simply taking local maxima because of factors like noise, hot pixels, and cosmic ray events. Using a *sharp* parameter, describing a distribution in intensity, and a *shape* parameter, describing a spread in space, we can more accurately distinguish stars from other phenomena in astronomical photos.

3 Closed-Form Matting

The image matting problem, given an input image I, consists of estimating a foreground image F, background image B, and *matte* α. The α is a parameter of opacity for each pixel, where $\alpha = 1$ if the pixel is entirely foreground, $\alpha = 0$ if the pixel is entirely background, and $0 < \alpha < 1$ for pixels in between. The *matting equation* that we aim to satisfy is

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

given as Equation (1) in Levin et. al. We can think of α as telling us how much to "blend" between the foreground and background.

In order to estimate a good α, we make the assumption that pixels within the foreground and pixels within the background are locally smooth (for some definition of smoothness). A sharp transition in color most likely occurs because we're at a discontinuity between foreground and background. From this assumption, we can approximate α as a linear function of the input image for most pixels.

More specifically, we let α be a $N \times 1$ vector, where N is the number of pixels in the image. The closed-form matting algorithm works by deriving and minimizing a cost function in terms of α. This cost function J is equal to

$$J(\alpha) = \alpha^T L \alpha,$$

where L is a $N \times N$ *matting Laplacian* matrix. The paper gives a (fairly complicated) formula for L in Equation (12).

When solving the matting problem, we need to begin with a set of known pixels to be provided by the user of the algorithm. We let D_S be a $N \times N$ diagonal matrix, where an entry is 1 if we are given the pixel value and 0 otherwise. We let b_S be a $N \times 1$ vector that contains the known value for all known pixels and zero otherwise. We let λ be a parameter controlling how much to weigh these known values against potentially better solutions. Finding the optimal α is equivalent to solving

$$(L + \lambda D_S)\alpha = b_S.$$

This is Equation (14) in the paper.

3.1 Implementation

In my code, the function `matting_laplacian` computes the L matrix. The function `cfm_with_trimap` solves for α given an image and a trimap indicating known values.

Since N is the number of pixels in our image (likely in the millions), a $N \times N$ matrix can be very large. Fortunately, $L + \lambda D_S$ is very sparse, and Eigen's sparse matrix solvers can handle it just fine for moderately sized images.

- `SparseLU` is a direct solver that seems to work very well. This was the solver that I decided to use.
- `ConjugateGradient` is an iterative solver that's fast when it works, but sometimes fails to converge.
- `BiCGSTAB` with the `IncompleteLUT` preconditioner is an iterative solver that always works, but seems to sometimes be less accurate than the direct `SparseLU`.

For larger images, the paper describes an iterative technique. We scale down the image, solve α for the smaller image, and scale back up. We threshold this resulting α, interpreting all values sufficiently close to zero or one as known values. By making these values known, we reduce the number of pixels in the larger image that we need to solve for, which makes the matrix L more sparse.

I implemented a function `iterative_cfm` that recursively decimates the image, solves for α, and scales back up. The iterative version is roughly twice as fast, taking around 15 seconds as opposed to 30 seconds on the dandelion image. One result of the thresholding is that we lose some detail in values very close to zero or one.

Matting is quite slow, but it is fast enough to be practically used. Solving for a 16MP image with `iterative_cfm` takes roughly 2.5 minutes and 10 gigabytes of RAM.

4 Star Finding and Alignment

In a wide-angle photo of the night sky, stars can be approximated by points that have a roughly Gaussian distribution in intensity. An approximate way to find stars that I implemented was to correlate the image with a Gaussian distribution and find local maxima. Correlating with a Gaussian is the same as convolving with the Gaussian kernel in the `gaussianBlur` functions that we have already implemented.

Afterwards, local maxima are checked to remove most non-stars. Since stars make up a very small amount of the image, the average color should be close to the color of non-star regions of the sky. If the correlation of a point is insufficiently greater than this average color, we reject the point. This allows for a lot of local maxima resulting from noise to be suppressed.

With the locations of stars found, we want to align two different images such that corresponding stars line up. This is quite similar to the panorama assignment, except that we first filter Harris corners for only those near a star. When finding correspondences, we consider distance in addition to similarity. Stars do not move much between images, so we can assume that correspondences should be near each other. On the other hand, the L2 similarity metric used for panoramas is not very effective because most stars look really similar.

4.1 Implementation

The functions `find_stars_matte` and `find_stars_color` are optimized for finding stars in alpha mattes and color images, respectively. The latter is used in `HarrisCorners_stars`, a modification of the panorama version that checks for proximity to a star. Similarly, the function `findCorrespondences_stars` is modified to consider spatial distance as previously described. The process of finding features, computing homographies, and aligning images takes a similar amount of time as computing the matte.

Another change from the panorama assignment was that I implemented solving for the homography matrix H with SVD, using Eigen's `JacobiSVD`. The motion of stars in the sky should be a rotation, which suggests that the entry in H assumed to be 1 would have to be 0. SVD overcomes this issue.

5 Stacking

After all images are aligned, we can stack them by applying a function to the set of pixels at each location in the stack. The ones I implemented were

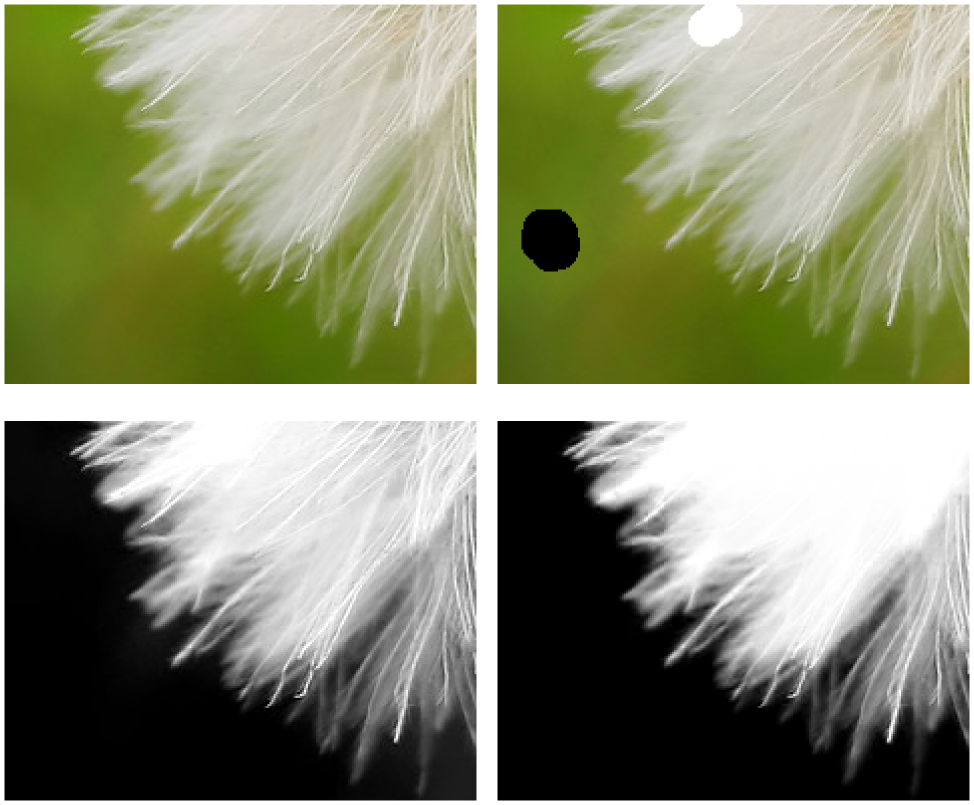
- `stack_mean`, which takes the average value. This is good for increasing the signal-to-noise ratio, but it leaves some "halo" along the border with the ground.
- `stack_max`, which takes the maximum value. When stacking unaligned images, this is good for visualizing star trails.
- `stack_mean_remove_outliers`, which computes the mean and variance for each channel, discards pixels that deviate too much from the mean, and averages the remaining pixels. In theory, this should perform better than `stack_mean`, but I never got very good results out of this function.

After stacking, the resulting picture of stars can be blended with a picture of the ground to create the final image.

6 Further Work

If I had more time, one idea that I would like to attempt is to create "synthetic" star trail images. Because the homography matrices describe the motion of stars over time, we can repeatedly apply these matrices to trace out the path that the stars would take over the equivalent of a very long exposure.

Example of input image, a "scribble" indicating known colors, α matte solved in one step, and α matte solved with the iterative method. Original image was from the Levin et. al. paper.



Stacking without alignment (star trails visible) and stacking with matting and alignment.