EECS 442 Computer Vision: Final Project Report

Nathan Immerman
College of Engineering, University of Michigan
Ann Arbor, Michigan
immerman@umich.edu

Alexander Chocron
College of Engineering, University of Michigan
Ann Arbor, Michigan
achocron@umich.edu

1. Introduction

With the advent of software tools such as Photoshop and Gimp, it is becoming increasingly simple to doctor and create fake images. One method of doctoring images is to create a composite image out of existing source images. Since composite images are difficult to detect for the common person, we hope to create a tool that enables users to determine composite images.

It is often difficult to make the light direction throughout the entirety of composite images consistent. This fact can be leveraged to detect whether even well-stitched images are fake. By analyzing the light direction of different surfaces within the image, one can detect whether or not the two surfaces came from the same original image. An algorithm for detecting such inconsistencies has been outlined in Johnson and Farid's paper, *Exposing digital forgeries by detecting inconsistencies in lighting*. We shall attempt our own implementation of this algorithm and measure our implementation based on accuracy.

2. Approach

The equation that we are making use of to estimate the light directions is

$$I(x, y) = R \times (\vec{N}(x, y) \cdot \vec{L}) + A$$

where $I(x, y)$ is the intensity at the point (x, y), R is the reflectance term (a constant), and $N(x, y)$ is the unit vector normal to the boundary at the point (x, y). This equation assumes an infinite light source.

A limitation of this equation is that it requires that the reflectance of the surface is known and that it is constant. By assuming that the reflectance equals 1, a vector for the light direction can be obtained with an unknown scale factor. However, this still requires that the object has constant reflectance. As the paper suggests, we are able to relax this constraint by partitioning the boundary into n patches, and assume that each of these patches has constant reflectance. The underlying assumption here is that more smaller, local patches of an object are likely to be most uniform, and therefore are likely to have a more constant reflectance than the entire object. In our implementation, we use n = 8, which the authors of the paper recommend.

2.1. Finding the Normal Vectors

The algorithm assumes that the boundaries are given, but the equation we are using requires that the vectors normal to the object boundaries are calculated. The reason we do not consider an arbitrary point on an object is that we cannot easily estimate the normal vectors at these points. We already know that the z-component of the vector is equal to 0 on the object boundary, and the x and y components point in the same direction in both the scene and image coordinates at these points, making them very easy to estimate with no object-level knowledge or 3D information. We estimate a normal vector for each point given in every partition. But the boundary is given as a set of points, and it is clearly impossible to estimate the normal vectors of discrete, non-continuous function. To solve this, we begin by fitting a quadratic curve $y = p(x)$. The curve is fit to three user-selected points that lie near the boundary partition. We then observe each point (a, b) in the partition, and find a unit-vector perpendicular to the quadratic curve at the point $(a, p(a))$.

At each partition, the user must indicate a fourth point that lies on the object to indicate which direction the normal vectors should point. If the normal vector that we calculate points away from this point, we simply scale it by a factor of -1.

2.2. Estimating intensities at the occluding boundary

We need to estimate the intensities of the pixels at the occluding boundary because the pixels are not in the image. For a single point along the boundary, this is accomplished by sampling n pixels along the direction opposite of the normal vector and fitting an exponential curve to these values, namely

$$I(x) = ax^b$$

where x is the position along the negative normal direction and $I(0)$ is the intensity at the occluding boundary. We used $n = 15$.

We use least-squares estimation on $\log(I(x))$ to determine the parameters a and b. The intensity at the occluding boundary is then calculated by evaluating $I(0)$.

2.3. Least Squares Solution

We cannot take a direct approach to solving for the light directions; we have an unknown ambient term and several points in each partition segment. To overcome this, the paper reformulates the problem in such a way that it is possible to apply least squares estimation to solve for the unknowns. After this is complete, we estimate the overall light direction on the object by averaging the light direction of each segment in the partition.

The paper reformulates the problem by first organizing the data:

$(x_j^{(i)}, y_j^{(i)})$ is the jth point of the ith partition for a particular boundary. There are n partitions, each of which contains p points.

$$D_k = \begin{pmatrix} N_x(x_1^{(1)}, y_1^{(1)}) & N_y(x_1^{(1)}, y_1^{(1)}) \\ \vdots & \vdots \\ N_x(x_p^{(1)}, y_p^{(1)}) & N_y(x_p^{(1)}, y_p^{(1)}) \end{pmatrix},$$

$$M = \begin{pmatrix} D_1 & 0 & \cdots & 0 & 1 \\ 0 & D_2 & \cdots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & D_n & 1 \end{pmatrix},$$

$$C = \begin{pmatrix} -1 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & \cdots & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & \ddots & & & & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 0 & 1 & 0 \end{pmatrix},$$

$$\vec{b} = \begin{pmatrix} I(x_1^{(1)}, y_1^{(1)}) \\ \vdots \\ I(x_p^{(1)}, y_p^{(1)}) \\ \vdots \\ I(x_1^{(n)}, y_1^{(n)}) \\ \vdots \\ I(x_p^{(n)}, y_p^{(n)}) \end{pmatrix}, \vec{v} = \begin{pmatrix} L_x^{(1)} \\ L_y^{(1)} \\ \vdots \\ L_x^{(n)} \\ L_y^{(n)} \\ A \end{pmatrix}$$

$\vec{v}$ is the quantity we are trying to find. To do so, we use this closed form of the least squares optimization:

$$\vec{v} = (M^T M + \lambda C^T C)^+ M^T \vec{b}$$

where A^+ would denote the pseudo-inverse of A. This is the closed form equation for finding the least-squares estimate of the light directions.

3. Implementation

No third-party software or systems were used. All components of our project, including the back end and front end GUI (graphic user interface), were implemented from scratch (other than the built-in functions that MATLAB provides). We chose to implement our tool in MATLAB due to our familiarity with the software and its built-in linear algebra and image manipulation functionality.

3.1. Back End

3.2. Front End GUI

Since many aspects of our tool requires user input, we decided to make a MATLAB GUI to provided a pleasant experience for the user. Below are the steps the user must take to enter the boundary data:

- The user opens the tool, entering the image path as a command-line argument.
- The user draws the occluding boundaries at which he or she wishes to determine the light direction. For best results, we recommend using the magnifying glass to zoom in on boundary locations before drawing. The drawings do not need to be perfect, but they should be quite close for best results. To initiate the drawing tool, the user must select *Add Boundary*. This process must be repeated for each boundary desired. There is no limit to the number of boundaries that can be drawn.
- Once the user is done adding all desired boundaries, he or she must select *Finish Boundaries*. In this phase of the tool, the user will be entering points that are used to estimate a quadric curve to fit to the boundaries that they user added. The GUI zooms in on small patches of a boundary, where the user selects three points that might approximate the small patch. After each set of three points, the user must select a point on the object itself. Once these four points are entered, the GUI zooms in on the next patch. This is repeated for all patches of all boundaries.
- The GUI zooms out to reveal the entire boundary and the user selects on the object where the arrow indicating the light direction will be plotted. This process repeats for each boundary that the user added.
- The GUI plots the estimated light directions for each boundary at the locations that the user specified

The user also has the ability to view the entered boundaries and the estimated normal vectors by toggling the buttons *Toggle Boundaries* and *Toggle Normal Vectors* respectively. Figure 1.

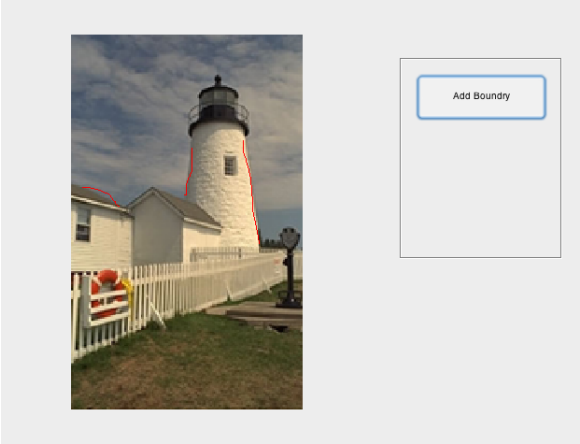

Figure 1. Current State of our GUI, user selected boundaries in red

4. Experiments

We performed a number of experiments to test the performance of our tool. These experiments included both testing the light direction of calculated in images for which a ground truth has been obtained, and testing the tool on known composits, with an emphasis on the former.

Above are examples of images that we tested our tool on. The ground truth for the light direction was estimated by holding a board with a protruding rod parallel to the image plane and measuring the direction of the shadow cast by the sun. This was then compared to the output of our tool. A brief summary of our results:

Image	Est. Direction	Ground Truth	Diff.
(a)	121°	113°	8°
(b)	122°	128°	6°
(c)	115°	120°	5°
(d)	133°	120°	13°
(e)	95°	89°	6°
(f)	126°	119°	7°

Angles are measured South of West.

The averge error for these images is 7.5°. We find that different kinds of surfaces yield different results. Typically, shirt sleeves, spherical objects, and fairly solid-colored objects give the best results. The algorithm makes a Lambertian assumption, so the less spectral reflection the better the tool performs (but sometimes surprisingly good results are obtained with shiny objects).

Image (g) is an example of a composite image that our tool performs quite well with. An image of a golf ball has been placed on the bulb on the left, and the tool calculates a light direction discrepancy of 69°, strongly (and correctly) indicating a forgery.ADD IMAGE (G) (should be close up of golf ball lamp with arrows that we used in the poster)

(a), (b)

(c),(d)

(e),(f)

4.1. Error Analysis

5. Conclusion

Light directions are often a reliable way to detect forgeries, however it is somewhat limited. Many images are not well suited for this kind of detection, whether they lack suitable surfaces or the user simply does not know what he or she is looking for. In order to makes use of the tool, the user must select boundaries on portions of the image that are suspected of being stitched together.

6. References

[1] M. K. Johnson and H. Farid. Exposing digital forgeries by detecting inconsistencies in lighting. *In Proceedings of the 7th workshop on Multimedia and security*, pages 1-10, 2005.