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.

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. 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. The following are the steps that the user would take when using our tool.

First, start the tool with the desired image as a command line argument.

Second, add the occluding boundaries that the user wishes to determine the light direction of. To do this first we recommend zooming in on the occluding boundary by using the magnifying glass from the tool bar. Then click the button *Add Boundary* and draw as close as possible to the occluding boundary while remaining on the actual object. The user should use the button *Add Boundary* for each boundary that they wish to add.

Third, once the user is done adding boundaries, the user should click the button *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 will zoom in on small patches of a boundary, the user should then enter three points that approximate the small patch. Then the user should enter an additional point that is on the object while remaining close to the patch. Once these four points are entered the GUI will zoom in on the next patch and the user should repeat the process until the user has entered four points for each patch on a boundary.

Then the GUI will zoom out to reveal the entire boundary and the user should enter one point on the object where the user would like the estimated light direction to be plotted. This process will repeat for each boundary that the user added.

Once the user has completed entering all of these points, the GUI will plot the estimate 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.

4. Experiments

5. Conclusion

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.