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. To solve the problem of constant reflectance over an object boundary, we split the user entered boundaries into n partitions, and assume that each of these partitions has constant reflectance. In our implementation, we use n = 8.

2.1. Finding the Normal Vectors

The reason we must estimate points that lie on a (occluding) boundary is that we are only able to estimate the normal vectors at the occluding boundary for a given object. We know that the z-component of the normal vector at the occluding boundary of an object is 0. This makes it much easier to estimate the x and y components. For each partition, we fit a quadratic curve using three points that the user enters. The user enters a fourth point for each patch as an indicator to the general direction of the normal vectors. We then use mathematical techniques to estimate the normal vector of the quadratic curve at the x coordinate of each point in the given partition.

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 3rd party systems or software was used. All components of our project including, the back end and front end GUI (graphic user interface), were implemented by us. We chose to implement our tool in matlab due to out familiarity with matlab and matlab's efficiency with images.

3.1. Back End

3.2. Front End GUI

Since many aspects of our tools 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.

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.