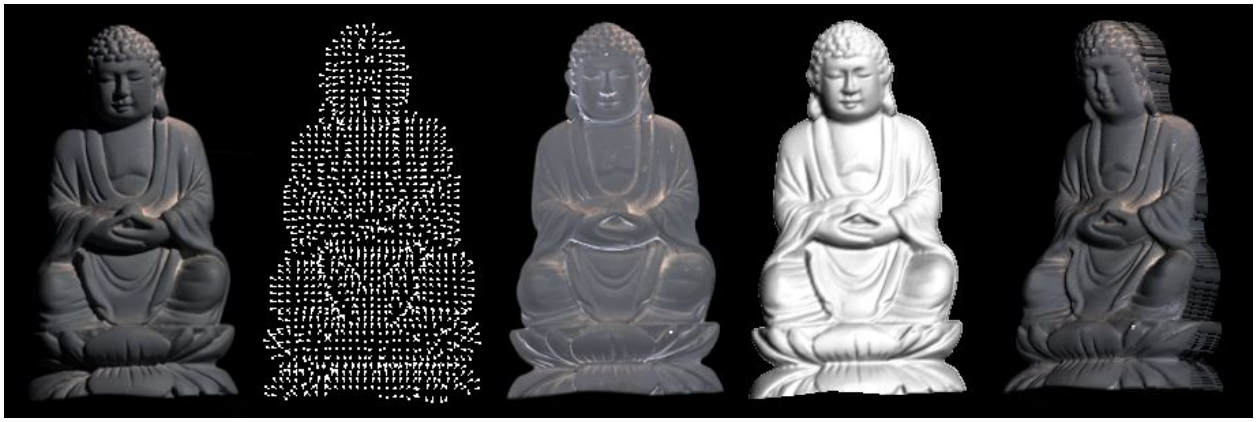
Project: 3D Reconstruction from Photometric Stereo

Announcement: 14 March 2017

Submission Deadline: 11 April 2017



Example of reconstruction using "Buddha" image set. From left to right: source image (1 of 12), normal vectors, albedo map, depth map, and a reconstructed view.

DESCRIPTION

In this project, you will implement an algorithm to construct a height field from a series of 2D images. These images are taken of a (mostly) diffuse object under a series of point light sources. Your software will be able to calibrate the lighting directions, find the best fit normal and albedo at each pixel, then find a surface which best matches the solved normals.

To start your project, you will be supplied with some test images.

Calibration

Before we can extract normals from images, we have to calibrate our capture setup. This includes determining the lighting intensity and direction, as well as the camera response function. For this project, we have already taken care of two of these tasks: First, we have linearized the camera response function, so you can treat pixel values as intensities. Second, we have balanced all the light sources to each other, so that they all appear to have the same brightness. You'll be solving for albedos relative to this brightness, which you can just assume is 1 in some arbitrary units. In other words, you don't need to think too much about the intensity of the light sources.

The one remaining calibration step we have left for you is calibrating lighting directions. One method of determining the direction of point light sources is to photograph a shiny chrome sphere in the same location as all the other objects. Since we know the shape of this object, we can determine the normal at any given point on its surface, and therefore we can also compute the reflection direction for the brightest spot on the surface.

Normals from images

The appearance of diffuse objects can be modeled as $I = k_d \mathbf{L}\mathbf{n}^T$, where I is the pixel intensity, k_d is the albedo, and L is the lighting direction (a unit vector), and n is the unit surface normal. (Since our images are already balanced as described above, we can assume the incoming radiance from each light is 1.) Assuming a single color channel, we can rewrite this as $I = \mathbf{L}(k_d\mathbf{n}^T)$, so the unknowns are together. With three or more different image samples under different lighting, we can solve for the product by solving a linear least squares problem. The objective function is:

$$Q = \sum_i (I_i - \mathbf{L}_i\mathbf{g}^T)^2$$

To help deal with shadows and noise in dark pixels, it is helpful to weigh the solution by the pixel intensity: in other words, multiply by I_i:

$$Q = \sum_i (I_i^2 - I_i\mathbf{L}_i\mathbf{g}^T)^2$$

The objective Q is then minimized with respect to g. Once we have the vector $g = k_d * n^T$, the length of the vector is k_d and the normalized direction gives n.

Solving for color albedo

This gives a way to get the normal and albedo for one color channel. Once we have a normal n for each pixel, we can solve for the albedos by another least squares solution. The objective function is:

$$Q = \sum_i (I_i - k_d\mathbf{L}_i\mathbf{n}^T)^2$$

To minimize it, differentiate with respect to k_d, and set to zero:

$$\frac{\partial Q}{\partial k_d} = \sum_i 2(I_i - k_d\mathbf{L}_i\mathbf{n}^T)(\mathbf{L}_i\mathbf{n}^T) = 0$$

$$k_d = \sum_i I_i\mathbf{L}_i\mathbf{n}^T / \sum_i (\mathbf{L}_i\mathbf{n}^T)^2$$

Writing $J_i = L_i \cdot n$, we can also write this more concisely as a ratio of dot products: $k_d = I \cdot J / J \cdot J$. This can be done for each channel independently to obtain a per-channel albedo.

Least-squares surface fitting

Next we'll have to find a surface which has these normals, assuming such a surface exists. We will again use a least-squares technique to find the surface that best fits these normals. Here's one way of posing this problem as a least squares optimization.

If the normals are perpendicular to the surface, then they'll be perpendicular to any vector on the surface. We can construct vectors on the surface using the edges that will be formed by neighbouring pixels in the height map. Consider a pixel (i,j) and its neighbour to the right. They will have an edge with direction:

$$(i+1, j, z_{(i+1,j)}) - (i, j, z_{(i,j)}) = (1, 0, z_{(i+1,j)} - z_{(i,j)})$$

This vector is perpendicular to the normal n, which means its dot product with n will be zero:

$$(1, 0, z_{(i+1,j)} - z_{(i,j)}) \cdot n = 0$$

$$n_x + n_z(z(i+1,j) - z(i,j)) = 0$$

Similarly, in the vertical direction:

$$n_y + n_z(z(i,j+1) - z(i,j)) = 0$$

We can construct similar constraints for all of the pixels which have neighbours, which gives us roughly twice as many constraints as unknowns (the z values). These can be written as the matrix equation Mz = v. The least squares solution solves the equation $\mathbf{M}^T\mathbf{M}\mathbf{z} = \mathbf{M}^T\mathbf{v}$. However, the matrix $\mathbf{M}^T\mathbf{M}$ will still be very big! It will have as many rows and columns as there are pixels in your image. Even for a small image of 100x100 pixels, the matrix will have 10^8 entries!

Fortunately, most of the entries are zero, and there are some clever ways of representing such matrices and solving linear systems with them. We are providing you with code to do this, all you have to do is fill in the nonzero entries of the matrix M and the vector v.

IMPLEMENTATION

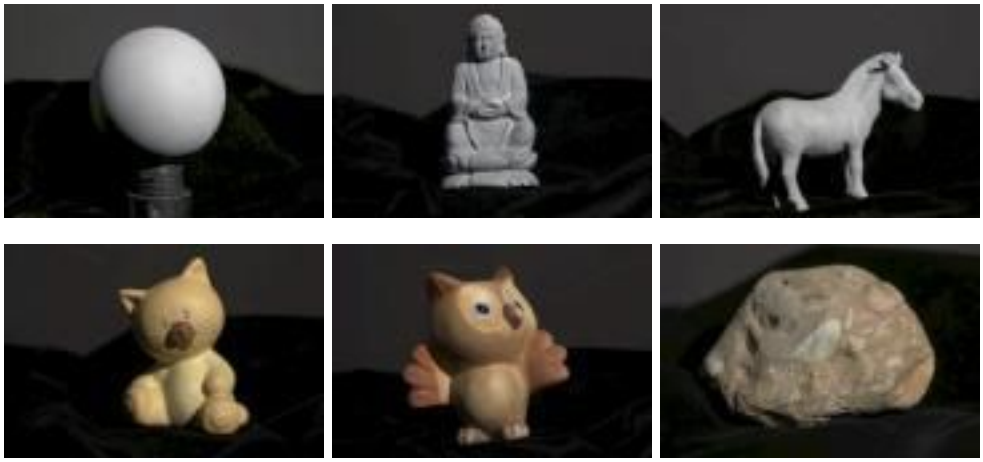
Compute lighting directions: read in the file chrome.txt and compute an estimate of the reflection directions for the lights visible in these images. All the images in this data set were captured using the same set of lighting directions, so after you've computed this once you can save this out to a file.

Solve for normals: read in a set of images, for example buddha.txt and using the lighting directions estimate a per-pixel normal.

Now that the normals have been computed, we can solve for the color albedos. You can save the albedos into tga files.

Solve for depths: this computation will take longer than the others. Like normals, you can save these to a binary data file. A software will be provided for visualizing the final 3D model.

We have provided six different datasets of diffuse (or nearly diffuse) objects for you to try out:

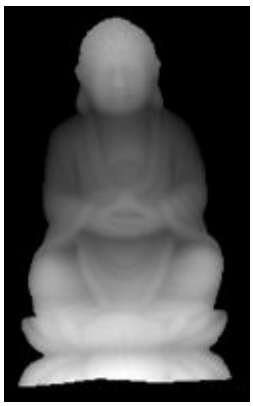


Top row: gray, buddha, horse

Bottom row: cat, owl, rock

In the sample images directory, we have provided a few files computed by the sample solution, for comparison with your own solution and also so that you can work on these tasks independently. The files are:

- lights.txt (lighting directions for all images)
- buddha/buddha.normals.dat
- buddha/buddha.albedo.tga
- buddha/buddha.depths.dat



If you want to open the depth map elsewhere:

- The file contains a series of 32-bit floats.
- The first two values are a prefix identifier and the remaining values are depth values.
- The 2D resolution will be the same as the 2D source images, e.g. it will be 512x340 for the Buddha set.
- With appropriate scaling and resampling, it will give results like the one shown at right.

DELIVERABLES

Your write-up should include results for at least three of the example data sets (excluding the "chrome" calibration set). Each data set should include the following:

- RGB-encoded normals
- albedo map
- two or more views of the reconstructed surface without albedos
- two or more views of the reconstructed surface with albedos

In the report you should include a discussion of your results. What worked and what didn't? Which reconstructions were successful and which were not? What are some problems with this method and how might you improve it?

EXTRA CREDIT

1. Do 3D reconstruction using facial images, either from the internet or from your own collection.
2. Use 3D printing to make a physical model from one of your 3D reconstructions. If you're interested in this option, talk to the professor - the department and university have 3D printers available, and we can steer you to the appropriate resources. By the due date, turn in the model file you intend to print. Your write-up should include documentation of the steps up to this point, as well as images of your print-ready model from at least three descriptive views. You then have until the last day of lecture to bring in a printed model to show the class.
3. Be creative! Come up with a project extension we didn't think of.

Submission (electronic submission through EAS only)

Please create a zip file containing your C/C++ code and a readme text file (.txt).
In the readme file document the features and functionality you have implemented, and anything else you want the grader to know i.e. control keys, keyboard/mouse shortcuts, etc.

Additional Information

- The assignment's images can be found here

Credits

Based on the assignment developed Steve Seitz, James Gray, and Avanish Kushal.