Photometric Stereo

Gustav Nilsson Gisleskog, Niklas Sandén, Supervisor: Carl Olsson

July 10, 2024

1 Introduction

In this project, we have implemented a system for performing photometric stereo. The goal of photometric stereo is to extract the 3D surface of an object given multiple images of it from the same camera setup, but under different lighting conditions. Two examples of such images can be seen in Figure 1. No additional data about the locations and directions of the light sources is necessary, because under a simplified light model, the image intensities alone allow us to find the normals of the surface, which can be integrated to reconstruct the surface. However, there are multiple challenges when implementing this in practice, such as how to deal with the ambiguities that arise because the system is underdetermined, and how to deal with pixels that are in shadow. We outline our attempts to solve these problems, along with our results.

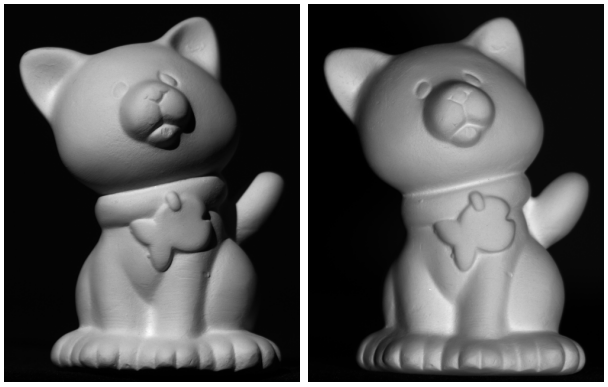

Figure 1: Two of the images from the cat dataset

2 Background

2.1 Lambertian lighting

In this project, we assume that the surface of the object is a Lambertian surface. This means that the luminance at a point doesn't depend on the viewing direction, but only on the albedo, light intensity and angle between the surface normal and light direction. This is in contrast to e.g. a mirror, where the viewer's position and angle may influence the perceived luminance.

Consider a point where we represent the surface normal and light direction by the normalized vectors $\mathbf{n}$ and $\mathbf{m}$, and the reflectance and light intensity by the numbers r and t. Then the luminance in that point will be $rt\mathbf{n}^T\mathbf{m} = r\mathbf{n}^T\mathbf{m}t$. If the only light source in a scene is far away, all light rays will be approximately parallel, meaning that the light intensity and direction will be (almost) the same in all points that are not shadowed. This situation is visualized in Figure 2. If an image is taken of such a scene, and we define functions $r(x, y)$ and $\mathbf{n}(x, y)$ to represent the reflectance and normal in pixel (x, y), then we can directly express the luminance in pixel (x, y) as $r(x, y)\mathbf{n}(x, y)^T\mathbf{m}t$.

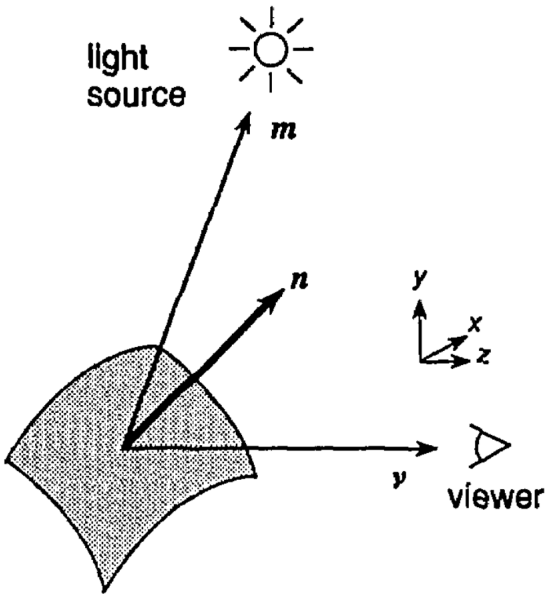

Figure 2: This figure is taken from Hayakawa's paper [1]. It illustrates the geometry of the Lambertian model. The luminance in a point does not depend on the viewer direction $\mathbf{v}$, but rather on the normal $\mathbf{n}$, light source direction $\mathbf{m}$ as well as the albedo r and light source intensity t. However, in this figure r and t were not included.

2.2 Basic solution

In the paper [1] by Hayakawa, he introduces a method for photometric stereo with knowledge of neither the surface normals nor the light source directions. We assume that the object's surface is Lambertian, that each image only has a single light source which is placed far away and that the camera and object never change their position.

Assume that there are f frames captured, where the image intensity was recorded in the same p pixels. In order to keep track of where in the image the p recorder pixels are, we keep a *mask* that allows us to directly map between image pixel position and index among the p pixels. We can place the image intensity data in a $p \times f$ matrix:

$$I = \begin{pmatrix} i_{11} & \dots & i_{1f} \\ \vdots & \ddots & \vdots \\ i_{p1} & \dots & \imath_{pf} \end{pmatrix}.$$

Here $i_{ab} = r_a \mathbf{n}_a^T \mathbf{m}_b t_b$ is the image intensity in pixel a, frame b. r_a and $\mathbf{n}_a$ represent the surface reflectance and normal in pixel a, and $\mathbf{m}_b$ and t_b represent the light direction and intensity in frame b. Create the matrices R and N to represent the surface, and M and T to represent the light sources:

$$R = \begin{pmatrix} r_1 & & 0 \\ & \ddots & \\ 0 & & r_p \end{pmatrix}, \quad N = \begin{pmatrix} \mathbf{n}_1 & \dots & \mathbf{n}_p \end{pmatrix}^T, \quad M = \begin{pmatrix} \mathbf{m}_1 & \dots & \mathbf{m}_f \end{pmatrix}, \quad T = \begin{pmatrix} t_1 & & 0 \\ & \ddots & \\ 0 & & t_p \end{pmatrix}.$$

Then I can be factored as $I = RNMT$. From here, we can create the surface matrix S and the light source matrix L:

$$S = RN =: \begin{pmatrix} \mathbf{s}_1 & \dots & \mathbf{s}_p \end{pmatrix}^T, \quad L = MT =: \begin{pmatrix} \mathbf{l}_1 & \dots & \mathbf{l}_f \end{pmatrix}.$$

The problem now is to actually find matrices S and L to factorize I. Because S is $p \times 3$ and L is $3 \times f$, I can at most have rank 3. In practice, because of noise the measured I will most likely have a higher rank than that, so we would like to find matrices $\hat{S}$ and $\hat{L}$ such that $\hat{I} = \hat{S}\hat{L}$ is as close as

possible to I. If we want to approximate I in a least squares sense, i.e. minimize $\|I - \hat{S}\hat{L}\|_F$, then we can use the singular value decomposition and the Eckart-Young-Mirsky theorem to solve this.

Let $I = U\Sigma V$ be the singular value decomposition of I. Here $U \in \mathbb{R}^{p\times p}$, $\Sigma \in \mathbb{R}^{p\times f}$ and $V \in \mathbb{R}^{f\times f}$. U and V are orthogonal matrices, and Σ contains 0 everywhere outside the diagonal, and the diagonal contains non-negative numbers sorted in descending order. Create the matrices U', Σ' and V', where U' is the first 3 columns of U, Σ' is the top left 3×3 submatrix of Σ and V' is the first 3 rows of V. Then by the Eckart-Young-Mirsky theorem, $\hat{I} = U'\Sigma'V'$ is the minimizer of $\|I - X\|_F$ over rank 3 matrices X. Note that for computational purposes, we only compute the *thin SVD*, where $U \in \mathbb{R}^{p\times f}$, $\Sigma \in \mathbb{R}^{f\times f}$. But performing the same steps with the matrices from the thin SVD still gives us the best rank 3 approximation of I.

So, in conclusion we can set $\hat{S}\hat{L} = U'\Sigma'V'$. One way to do this is by setting:

$$\hat{S} = U'\Sigma'^{\frac{1}{2}}, \quad \hat{L} = \Sigma'^{\frac{1}{2}}V'.$$

Note that the vectors in these matrices are written in an arbitrary coordinate system, and not necessarily the one aligned with the viewer direction which we in practice are interested in. For example, if we let the matrix W be invertible, then $\hat{I} = \hat{S}\hat{L} = \hat{S}WW^{-1}\hat{L}$ and $S' = \hat{S}W$, $L' = W^{-1}\hat{L}$ could also be seen as a valid factorization of $\hat{I}$. So, we would like to find a suitable W that will transform the vectors to have the properties we want, i.e. so that it fulfills

$$S = \hat{S}W, \quad L = W^{-1}\hat{L}.$$

We will start with this process in this section, and continue with it in the next section.

From the definition of S, we can say that the albedo in a pixel k corresponds to the length of the row vector $\hat{s}_k$ in $\hat{S}$. In this project we make the assumption that the surface has constant albedo, which can be modeled as $\|\hat{s}_k\|_2 = 1 \ \forall \ k$. Right now we cannot expect this to hold for $\hat{S}$, but we can try making a 3×3 matrix A (like the W above) to find a solution that is as close as possible to this. If we replace $\hat{S}$ by $\hat{S}A$, then we instead want to have

$$\|\hat{s}_k^T A\|_2 = \hat{s}_k^T AA^T \hat{s}_k = 1. \tag{1}$$

Here we can first solve for $B = AA^T$ and then extract A from it. The equation becomes $\hat{s}_k^T B\hat{s}_k = 1$. B is symmetric and 3×3, so it has 6 unknowns. So, we get a linear system of equations of 6 unknowns and p equations, which we solve with least squares. Then because B is symmetric it will have a singular value decomposition on the form $Q\Pi Q^T$, so we can set $A = Q\Pi^{\frac{1}{2}}$. Then we replace $\hat{S}$ by $\hat{S}A$ and $\hat{L}$ by $A^{-1}\hat{L}$. Note that we could also set $A = -Q\Pi^{\frac{1}{2}}$, which will switch the vectors between a right handed and left handed coordinate system. We will touch more on this later.

3 Problems/challenges

3.1 Normals coordinate system

The base solution as described in the paper by Hayakawa [1] gives normals in an arbitrary coordinate system. This is because the A matrix from Equation 1 was arbitrarily set to $Q\Pi^{\frac{1}{2}}$, when there are an infinite number of choices. In fact, any orthogonal matrix R could be used to transform the normals (and light directions) and still satisfy our equations because $\hat{s}_k^T AA^T \hat{s}_k = \hat{s}_k^T AIA^T \hat{s}_k = \hat{s}_k^T AR^T RA^T \hat{s}_k$. Therefore, a natural next step is to align the coordinate system of the normals with the camera. The paper mentions that this can be done if you know the exact directions for three of the normals in the camera's coordinate system. The method we opted for is to show the object to the user, and let the user select three points (see Figure 3). The first one should correspond to a normal that points directly towards the camera. This is typically the easiest one to locate, since the object is rotated towards us. The second point should have a normal pointing along the positive x-axis (to the right in the image), and the final one should point along the negative y-axis (down in the image). Our algorithm then does the following:

1. Rotate the normals so that the first normal selected has values $(0, 0, -1)$, i.e. points towards the camera.

2. Rotate the normals around z-axis so that the second normal's x and y coordinates are $(1, 0)$.

Figure 3: Example of images shown to the user when they should pick three points. You can pick points in any of the three images. The reason we show three images is in case a region is shadowed and difficult to select in one of the images.

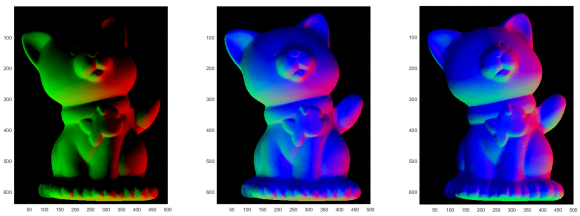

Figure 4: This image shows how the normals are rotated after picking the three points. The normals' xyz-coordinates are directly mapped to the RGB channels, although the sign of the z-channel is flipped so that normals pointing towards the camera are blue. The first image shows the normals in the initial coordinate system. The second one is after we have rotated so that the z-direction points away from us. The third image is after we have rotated (and potentially flipped) the normals around the z-axis. The expected result is that red corresponds to normals pointing to the right, green to normals pointing down, and blue to normals pointing towards the camera.

3. If the final normal points up, mirror the normals along the xz-plane.

The benefit of doing it iteratively like this is that only the first and easiest normal selected needs to be accurate. The second one only needs to be accurate after projecting in the xy-plane, and the final one only needs to point *more* down than up in the image. The results of the different rotations can be seen in Figure 4.

3.2 Integration

The second problem to tackle is that of integration. The method presented thus far only concretely describes how to approximate the normals of the surface, but to get the surface itself, we need to get the depth at each point. This is done by *integrating* the normals. There are multiple ways to try to do this, but we did a simple solution that sets up a system of linear equations where the depth of each point depends on the depth values of the four neighboring points and their normals. Our solution ignores the fact that there could be discontinuities in the surface, but this could be an interesting problem for future work.

For a given pixel (x, y) with unknown depth $d(x, y)$ and normal n (aligned with the camera's coordinate system), the plane that tangents the surface at point (x, y) can be written as

$$\pi : (x, y, d(x, y)) \cdot n + c = 0 \implies d(x, y) = -\frac{xn_x + yn_y + c}{n_z}. \tag{2}$$

Taking the derivative with respect to x and y yields

$$\frac{d}{dx}d(x, y) = -\frac{n_x}{n_z} \qquad \text{and} \qquad \frac{d}{dy}d(x, y) = -\frac{n_y}{n_z}. \tag{3}$$

Given $d(x, y)$, we can use this gradient to approximate the depth values of a neighboring pixel (a, b) as:

$$d(a, b) \approx d(x, y) + (a - x, b - y)\nabla_{xy}d(x, y) \implies d(a, b) - d(x, y) \approx (a - x, b - y)\nabla_{xy}d(x, y). \tag{4}$$

This gives us four equations for each center pixel, assuming we only look at the four closest neighbors, three for each edge pixel, and two for each corner pixel. We can express this system of linear equations as $Ad = b$, where d is a vector containing all the depth values, b contains the known constants in the right-hand side of Equation 4. Each row of A would correspond to a single equation, and have a 1 for the destination depth, a -1 for the starting depth, and zeros everywhere else. This matrix is very large, but the problem is easily solvable if one uses the fact that A is exceptionally sparse. However, there is one remaining issue, the system is underdefined. If d is a solution, then any constant added to all elements of d is also a solution. To handle this, we add a final simple equation that just says that one arbitrary pixel should have depth 0. After a solution has been found, we remove all values outside our mask and offset all the depth values by the smallest one, so that all depth values are non-negative.

The way we do integration now also adds equations for the background pixels, forcing the entire image to be continuous in depth before then removing the background. Another option would be to only add the equations for the pixels in the mask. If this is done, then the mask either has to have a single connected component, or you would need to add the extra condition specifying the depth for an arbitrary pixel for each extra component as well. Regardless, there is no way to tell the relative depths between the different components.

3.3 Shadows

One problem with getting the initial $\hat{S}$ and $\hat{L}$ in the way described in Subsection 2.2 is that the object can cast shadows onto itself if part of it is in the way of the light. When this happens it breaks our light model and the assumption on which we base the entire method does not hold. However, this happens rarely enough that it works in practice anyway. Nevertheless, we would still like to use a better method that does take the shadowed regions into account.

In the paper [1] where Hayakawa described the original method, he also proposes an algorithm to better handle shadows. The idea is to first filter out all pixels that are shadowed in any image, leaving only the pixels that are always illuminated. This can be done with a simple thresholding. Then, perform the same SVD strategy as in Subsection 2.2 to get an initial $\hat{S}$ and $\hat{L}$, but where $\hat{S}$ so far only contains the normals at pixels that are always illuminated. After that, the remaining normals can be decided by least squares. Again, we use the light model $i_{ab} = \hat{s}_a^T \hat{l}_b$, i.e. the intensity in pixel a, frame b is the dot product of initial normal $\hat{s}_a$ and light source $\hat{l}_b$. For pixels a that are shadowed in at least one frame, we can set up a system of equations using the previously mentioned equation, for all frames where a was illuminated. Because the light sources $\hat{l}$ are known, there are 3 unknowns (the 3 numbers in $\hat{s}_a$), and provided that a was illuminated in at least 3 frames, we would expect to find a unique solution for $\hat{s}_a$.

Sadly, in practice this usually performed worse than the initial algorithm.

4 Implementation

We implemented and ran our experiments in MATLAB. The steps of the algorithm are summarised below:

1. Extract illuminated pixels from the different frames (mask).
    - To get the mask, we simply applied a threshold on the intensities (≥ 40 worked well for the sample dataset we had). The mask contains the pixels where this threshold is exceeded in any of the input images. For more challenging data, a more sophisticated method is required.
2. Use singular value decomposition to get an initial solution $\hat{S}$, $\hat{L}$.
3. Upgrade the solution, with the matrix A, assuming that the surface has constant albedo.
4. Rotate normals so that they are aligned with the camera coordinate system.
5. Calculate the gradients from the normals (Equation 3).
    - Note that the division can create massive gradients in magnitude if the z-component is close to zero. For this reason, we clamp the gradients between -10 and 10 to guard us from this.

6. Integrate the gradients to get the depth values.

    - As previously noted, using sparse matrices is necessary to integrate using our method on any but the smallest of images.

5 Results

We ran the described algorithm on a number of different datasets. In Figure 5, we show the objects along with their estimated depth maps and 3D reconstructions, with the method that does not take shadows into account. Figure 6 shows the corresponding plots for the cat and the scholar, but with the method that tries to handle shadows.

We can note that the latter method seems much less stable.

6 Discussion & future work

The system, as it is currently defined, works mostly well. However, there are some details that cause inaccuracies in the 3D reconstruction, or inconveniences for the user. Below we discuss some of them.

6.1 Shadows

The method of handling shadows may intuitively seem like it should work. Moreover, when computing the RMSE of a solution by comparing the predicted image intensities from $\hat{S}$ and $\hat{L}$ with the actual image intensities, but only considering the illuminated data from I, this method outperforms the algorithm that does not handle shadows. So, it achieves lower error, which should indicate that the acquired normals would be more accurate. However, in practice it still does not work very well.

Our best guess as to why this is the case relates to the fact that we have to manually specify points where the normals should go in a certain direction. For example, when specifying normals that should point to the right, the easiest way to do it is to pick a point very close to an edge. However, one can imagine that on the edge, shadows are more common, giving us less data in the least squares to calculate the normal. This in turn might make it less accurate. We also want to note that for the cat, there were some pixels that were illuminated in less than 3 frames, giving us an underdetermined system for the corresponding normals.

6.2 Gradients

One observation we made is that the 3D reconstructions (without handling shadows) are almost always more extruded than they are supposed to be. For example, Figure 7 shows the cat reconstruction viewed from another angle. We believe the reason for this is that we don't handle shadows in these results. If a pixel is shadowed in an image, then that image's contribution to the estimate of the normal will guide the normal towards being orthogonal to the light source (since this gives a black pixel). The solution in the end is the least squares solution, so shadows typically guide normals to be more horizontal. This in turn causes the gradients to be larger because of the division by the z-component (see Equation 3), which causes the surface to extrude more.

One bit of evidence that could support this idea is that the left side of scholar in Figure 5 was shadowed in a lot of the images, and since it is far to the left, this will often cause the normals there to go towards the left too. Therefore, we get large gradients, which gives the left side a much higher depth than it should have. This is in contrast to Figure 6, where that problem is not present.

6.3 Manually specify normals

One limiting factor of the algorithm presented here is that the user has to manually select three points on the input image, as described in Section 3.1. This is the only step that requires user interaction, so if this step is automated, then the entire process will be automatic. One idea we tried was to assume that the normals along the silhouette were in the image plane. Using this assumption, we can find the rotation matrix that minimizes the absolute value of the z-component of the normals around the silhouette using SVD. This aligns the z-axis with camera view direction (potentially with the wrong

sign). Then, to rotate the normals around the z-axis, we looked at the normals along the points furthest to the right, with the assumption that they should now point to the right in the image. However, this method proved unreliable in most cases. A more sophisticated way to automate this would be an interesting improvement that we leave for future work.

References

[1] Hideki Hayakawa. "Photometric stereo under a light source with arbitrary motion". In: *Journal of the Optical Society of America* 11.11 (1994), pp. 3079–3089.

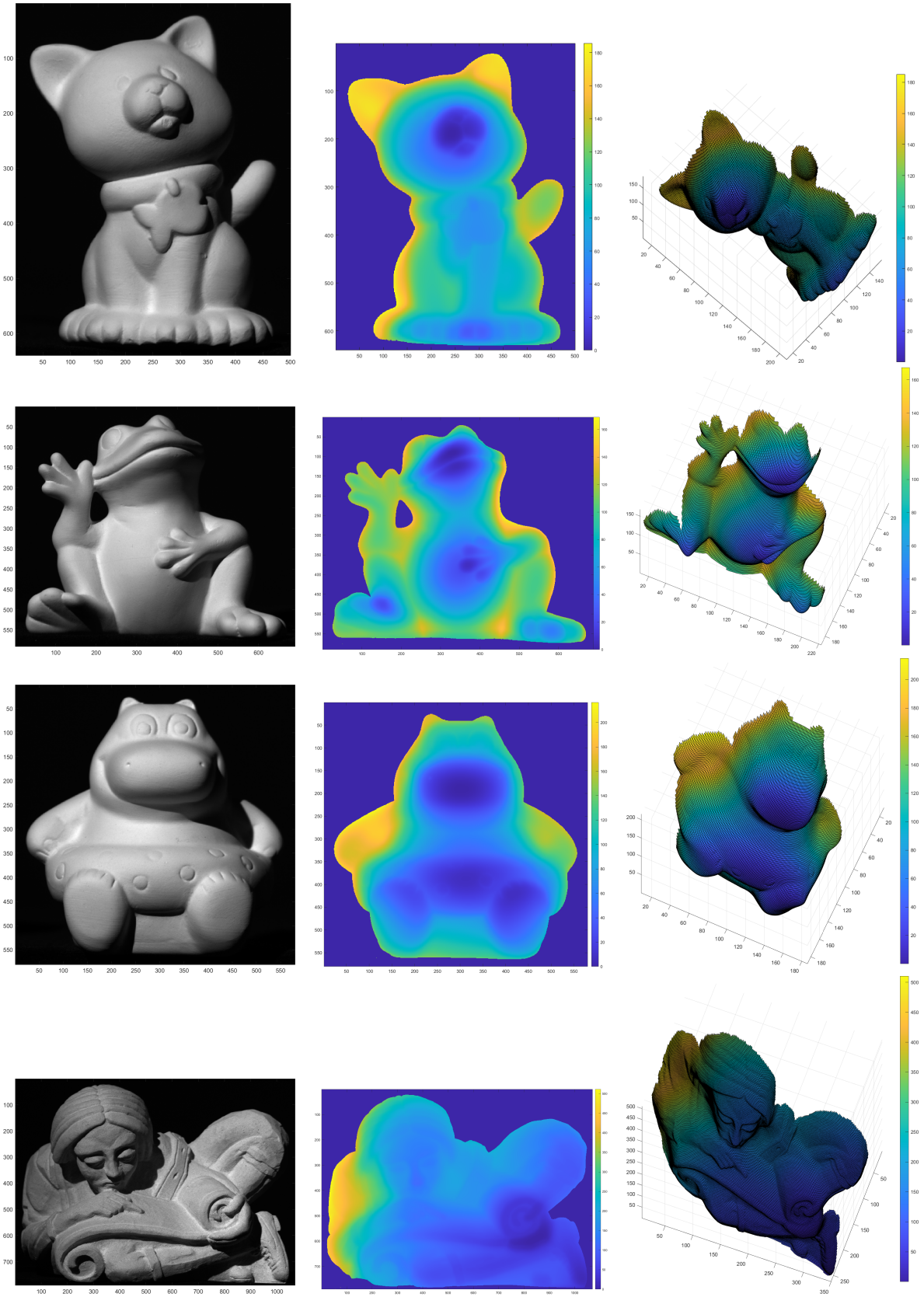

Figure 5: Our results without accounting for shadows for 4 different models (cat, frog, hippo & scholar). One of the input images (out of 20) is shown first, then the estimated depth map, and finally the 3D surface after masking out the background.

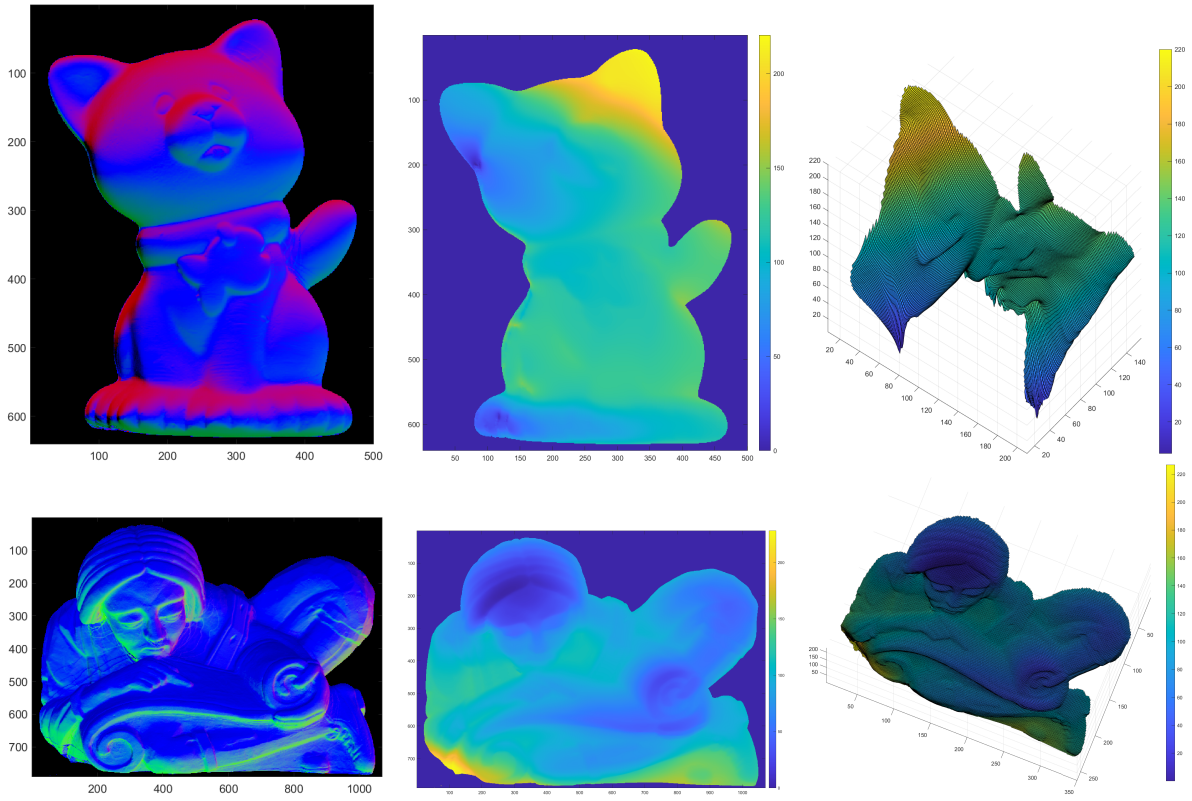

Figure 6: This shows our results when trying to account for shadows (on the cat and scholar models). The first image is our estimated normals after we tried to fix them. The second is the estimated depth map and the right one is the 3D surface. While it performs reasonably well on the scholar model, it completely fails on the cat.

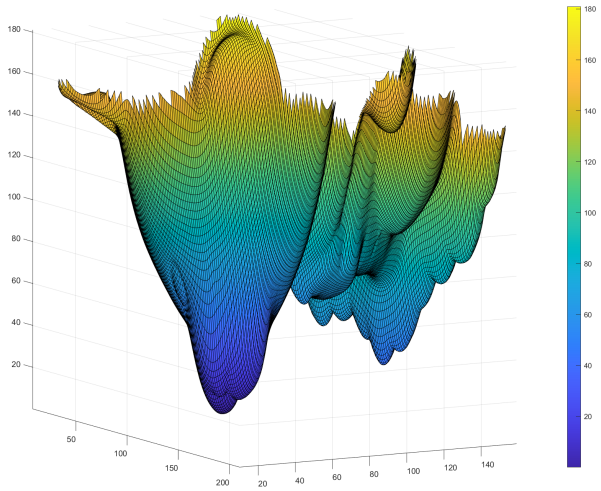

Figure 7: The reconstructed 3D surface for the cat model viewed from a different angle compared to Figure 5. The model is more extruded than it is supposed to be.