Final report - Photometric Stereo

COMPUTER VISION

2024 - SAITO HIDEO

Louis Peyrondet

July 2024

1 Project Description

Photometric stereo is a powerful technique in computer vision and computer graphics used to estimate the 3D shape and surface properties of objects. By capturing multiple images of an object under varying lighting conditions, photometric stereo analyzes the way light interacts with the surface to recover information about its geometry and reflectance. This method provides a way to accurately reconstruct surface details and texture, making it useful for applications in fields such as robotics, animation, and visual inspection.

Given these features, I chose to make my final report on this algorithm as I think that it was impressive to be able to compute the normals and albedos from just a varying light source and I wanted to understand how this could be done.

For this work I went and got a dataset of pictures from the paper "From Shading to Local Shape"[1], available on this website[1]. I didn't take my own pictures as I firstly do not have the equipment necessary such as the sphere object for calibration, directional light or a stable camera. Additionally, I learned from the previous projects and classes that often a computer vision algorithm depends on the quality of the input images, and poor-quality images can result in degraded and sometime incorrect output after the processing. Therefore using a curated dataset was the best solution for me and would also allow me to compare my results to those of the paper.

2 Reflectance map

Before diving in the actual implementation of the Photometric Stereo algorithm, I will quickly explain the role of the Reflectance map and how the algorithm uses it to estimate the normals values.

The image intensity I can be expressed using the following expression :

$$I = c\frac{\rho}{\pi}\frac{J}{r^2}\cos\theta_i = c\frac{\rho}{\pi}k(\mathbf{n}\cdot\mathbf{s})$$

where k is the source brightness, ρ is the reflectance/albedo, c the camera gain, $\mathbf{n}$ the surface normal and $\mathbf{s}$ the source light direction vector. Then if we assume $c\frac{\rho}{\pi}k = 1$ we have

$$I = \cos\theta_i = \mathbf{n}\cdot\mathbf{s}$$

Therefore, if we know the light source vector direction vector with the intensity value and that we assume that the surface is a Lambertian surface, which is a type of surface that reflects light uniformly in all directions so that the viewing angle doesn't have any effect, we can get the surface normal. As an example, clay is a material considered to be a Lambertian surface.

This equation lets us create the Reflectance map of a Lambertian surface for a given light source direction. It represents the light intensity at the point for the values x, y of the surface normal. This means that for a given intensity value we can plot a iso-brightness contour line that represent all of the possible surface normals that would result in this intensity. Here is an example :

[1] http://vision.seas.harvard.edu/qsfs/Data.html

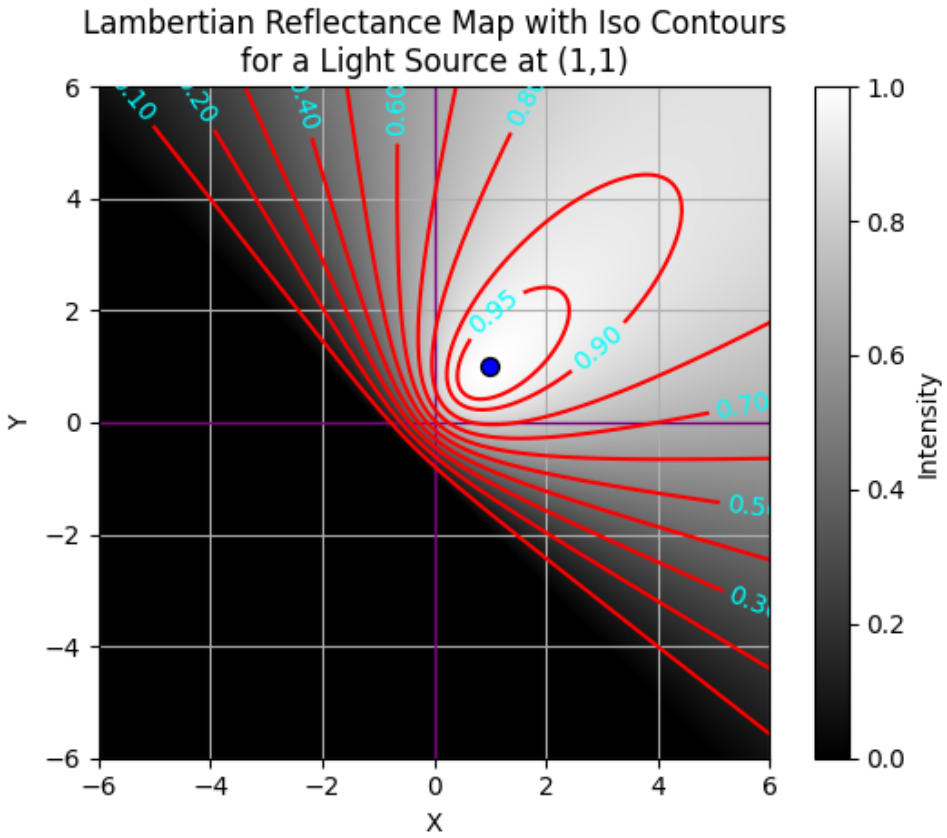


Figure 1: Reflectance map of a Lambertian surface with a light at (1,1,1). The highlighted blue point represents the brightest spot, which is directly under the light source, and the lines represents the iso-brightness contours from 0 to 1 in steps of 0.1, in addition to the level 0.95

Therefore if we add the iso-brightness contours from another light source we have intersections between the lines. Given that each one would have a measured intensity we just have to follow the iso-brightness contour representing the said value for each image and their intersections should show us the normal value. Of course we saw on Figure 1 that with 1 iso-brightness contour we have an infinity of possible solution, then with 2 contours we should have 2 possible results and with 3 contours we can identify the unique vector value for the surface normal. This is one of the requirement of the Photometric Stereo algorithm : You need at least 3 images to estimate the normals. However, as we are making assumption and that different light orientation result in different possible information, we usually use more than 3 images to get better results. In our case the example used for this report have 20 images.

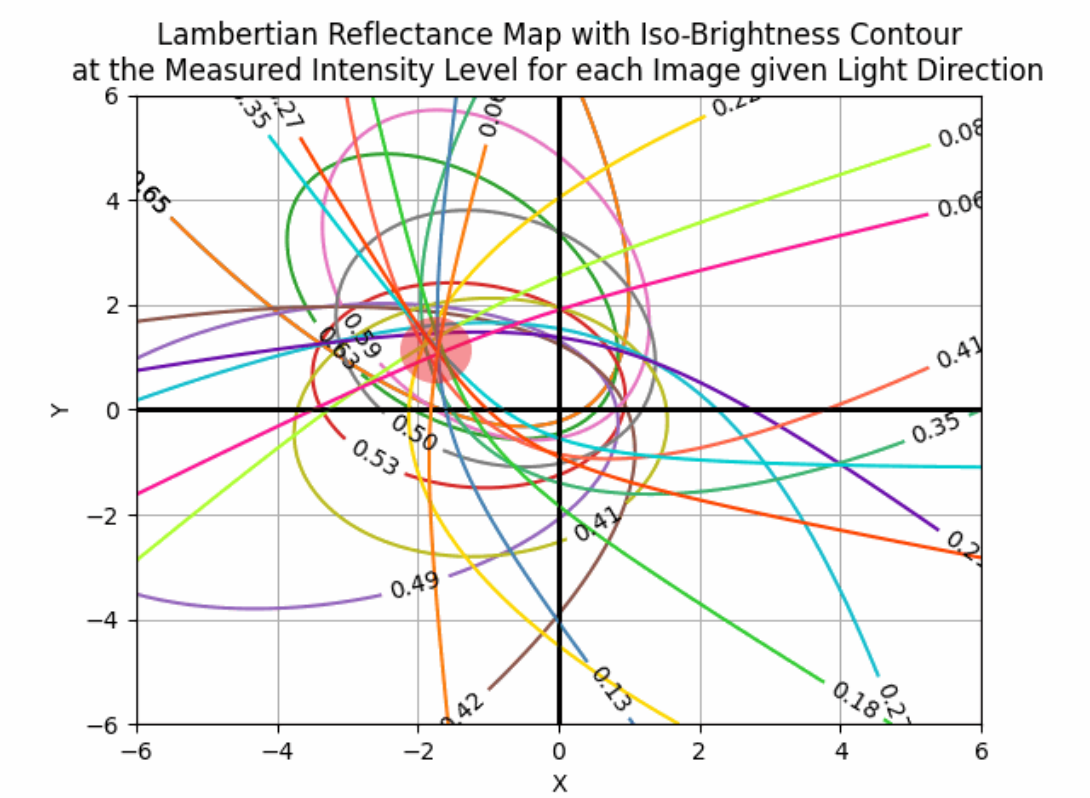


Figure 2: Iso-brightness contours from each of the 20 images each at the level of their respective measured light intensities. The red circle is the area where the density and intersection are the more frequent.

Ideally, all of the iso-brightness contours should intersect at the same value point, but given the assumptions and input images inaccuracies, we can instead observe a region where the iso-contours density and overlaps are more frequent. This region is the area giving us an approximation of the normal vector value. For example in Figure 2, if we sample a value in the red region we have $x = -1.75$ and $y = 1.50$ and we know that the z value is 1. Normalizing this vector results in the vector which gives an approximation of the surface normal. We can compare it to the actual value computed using our Photometric Stereo program to show that it is accurate :

$$n_{sampled} = [-1.75, 1.50, 1]$$
$$n_{approximated} = ||n_{sampled}|| = [-0.696, 0.597, 0.398]$$
$$n_{computed} = [-0.680, 0.571, 0.458]$$

To get a better understanding of it I made an animation[2] showing the iso-brightness lines overlapping and how each new image reduces the possible set of values to estimate the normal. Especially in the video, you can see how the last images information allow the iso lines to clearly determine that a good estimation is in the -2,2 area instead of the 1,1 area. I believe that this is the heart of the Photometric Stereo algorithm and that the animation does a good job of showcasing it.

[2]https://www.youtube.com/watch?v=4jEdv5bEzTo

3 Photometric Stereo

Now that we have an understanding of how we're estimating the normal values we can describe the steps of our Photometric Stereo algorithm.

3.1 Calibration

The first step is to estimate, for each image, the position of the light source and get the source light vector. To do this, we have to use an object that have a known geometry. In our case a sphere is used, because with this geometry we can easily compute the light direction vector.

Figure 3: An example of one of the calibration image.

$$L = 2(N.R)N - R$$

with

$$Nx = Px - Cx$$

$$Ny = Py - Cy$$

$$Nz = \sqrt{(R^2 - N_x^2 - N_y^2}$$

where P is the location of the brightest spot and C is the center of the sphere.

By repeating this for every input image, we now have all of the source light direction vectors, l_i and we can make the matrix L of light source direction vectors

$$L = \begin{pmatrix} l_1^T \\ l_2^T \\ \vdots \\ l_n^T \end{pmatrix}$$

3.2 Normals and Albedos

As we saw the normal of a surface gives us its orientation and the albedo represents how much light is reflected by the surface. For n images, we can model each pixel intensity I_i as :

$$\begin{pmatrix} I_1 \\ I_2 \\ \vdots \\ I_n \end{pmatrix} = \rho \begin{pmatrix} l_1^T \\ l_2^T \\ \vdots \\ l_n^T \end{pmatrix} \mathbf{n}$$

simplified as

$$I = \rho L \mathbf{n}$$

with ρ being the albedo of the surface at that point and $\mathbf{n}$ the normal vector of the surface at that point. This assumes that the albedo of the surface does not change between the images, which makes the estimations of a working LCD (Liquid Crystal Displays) surface impossible with this model.

Afterward we can use the least squares method to solve the equation where $g = \rho\mathbf{n}$:

$$g = (L^T L)^{-1} L^T I$$

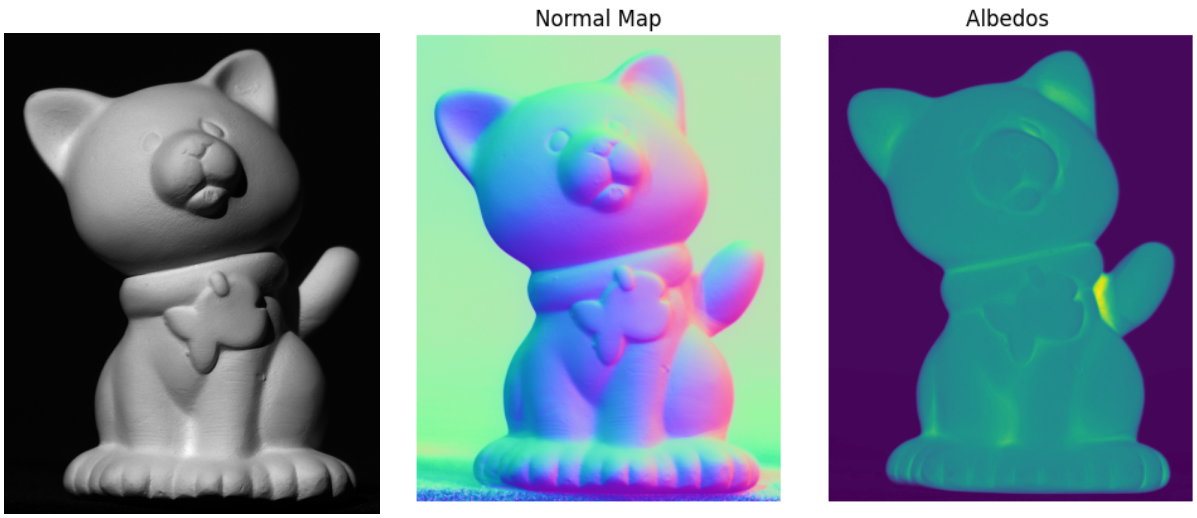
Then we can extract the albedo and normal from g :

$$\rho = ||g||$$

$$\mathbf{n} = \frac{g}{||g||}$$

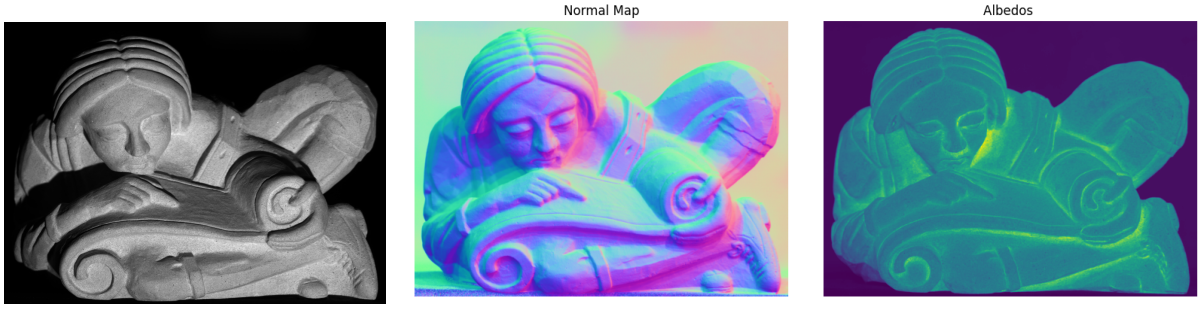
where $||g||$ is the magnitude of g, giving the albedo ρ, and normalizing g gives the unit normal vector $\mathbf{n}$.

The compact Python implementation of this equation is available in the Annex as Figure 9.



(a) One of the sample image (b) Estimated normals (c) Estimated albedos

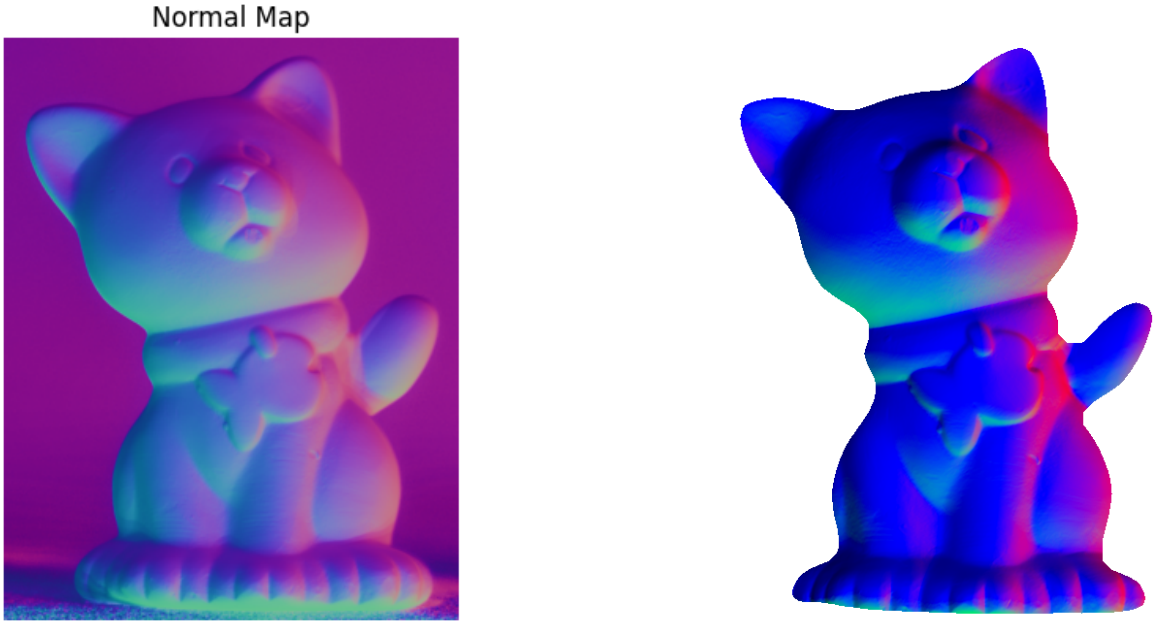
Figure 4: Surface normals and albedos estimation using the Photometric Stereo algorithm for the cat object



(a) One of the sample image (b) Estimated normals (c) Estimated albedos

Figure 5: Surface normals and albedos estimation using the Photometric Stereo algorithm for the scholar object

We can now also compare our result for the normals to the one showcased in the paper in Figure 6. Note that the results in the paper used a different kind of normals visualization that I tried to match, close to the differences between OpenGL and DirectX normal map visualization, so that is why the shades are not exactly the same but the variations are the important aspect and in this aspect our result closely matches the paper's.

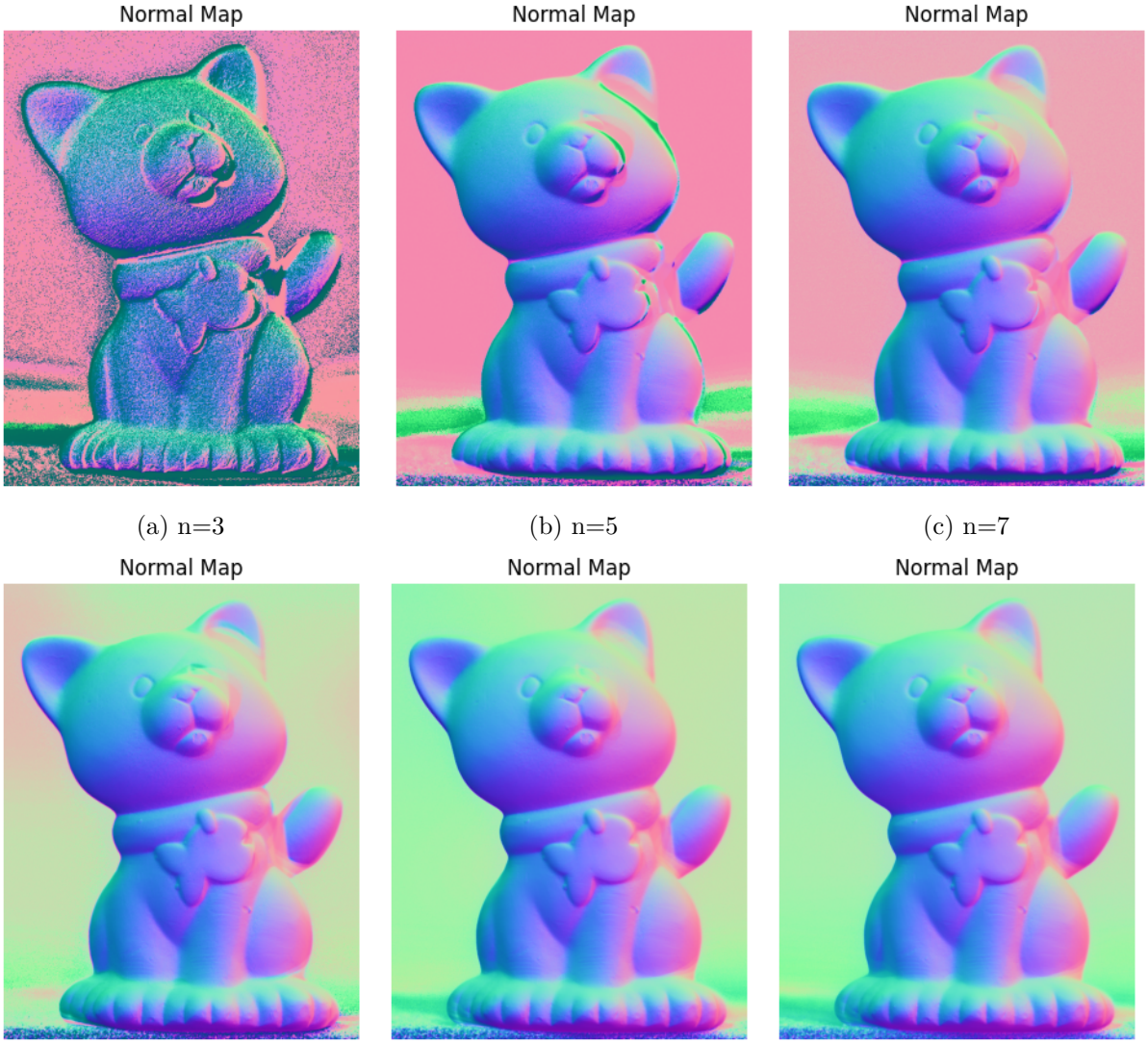


(a) Algorithm Results (b) Paper results

Figure 6: Comparison between our result to the paper result

4 Removing images

As it was explained in the Reflectance map section, having multiple image allow the algorithm to make better predictions. Let's try to remove some image and see how it will affect the results. Of course, it should be noted that each image does not provide the same amount of information, so I first removed the images that were the most similar in order to give the maximum amount of data to the program.

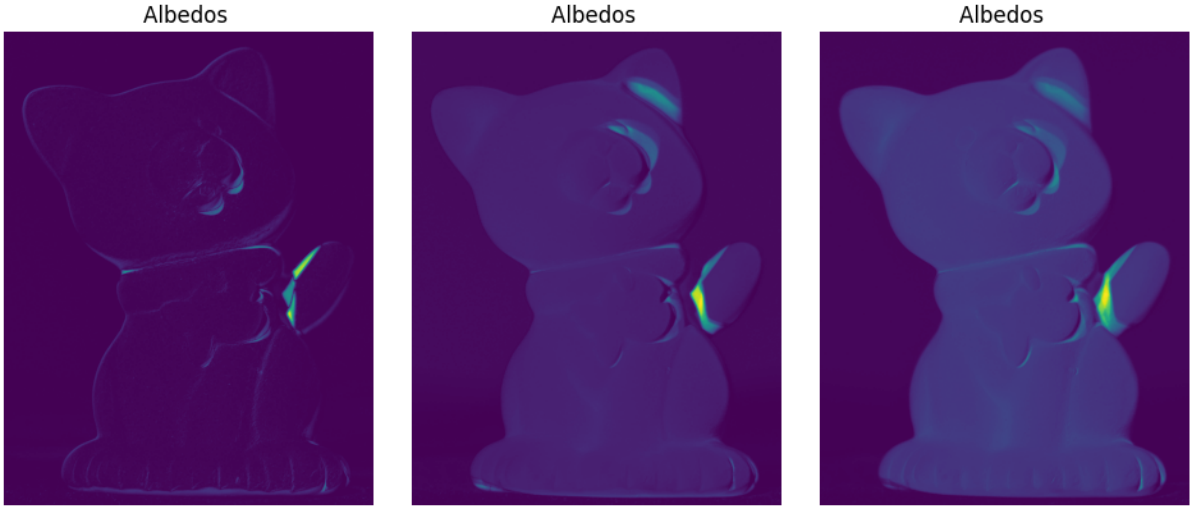


(a) n=3 (b) n=5 (c) n=7

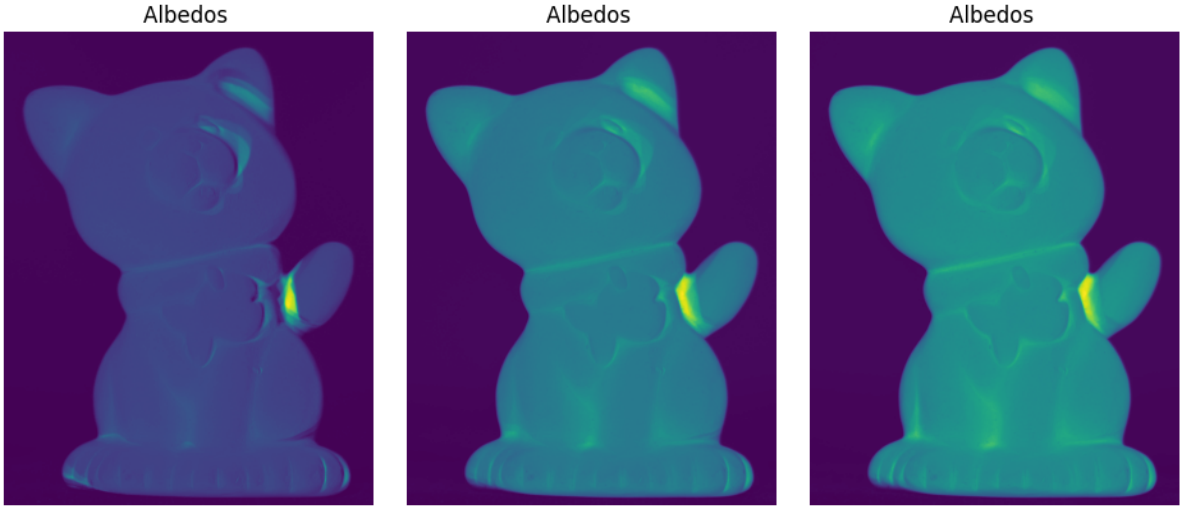
(d) n=10 (e) n=15 (f) n=20

Figure 7: Effect of reducing the amount of input images on the surface normals estimations

We can observe that with only 3 input images, the normals are very noisy and inaccurate. Jumping to 5 images already removes most of the noise but the algorithm is missing some information about the right side of the image and cannot estimate the normal there. By adding 2 more images and going to 7 images, we added an image containing lighting information about the right side which solves the previous problem.



(a) n=3 (b) n=5 (c) n=7



(d) n=10 (e) n=15 (f) n=20

Figure 8: Effect of reducing the amount of input images on the albedos estimations

For the albedos, whereas the normals didn't improve much from 15 to 20, here we can observe that the albedos still get more refined with this increase, especially around the cat's ears area.

5 Conclusion

This report demonstrated the effectiveness of photometric stereo in estimating 3D surface details using images taken under varying lighting conditions. By analyzing the reflectance map and iso-brightness contours, I showed how multiple images improve the accuracy of surface normal estimation.

A possible addition to this work would be to generate the depth map from the surface normals. I didn't make it as the report was already long enough, and I believe that Photometric Stereo is a middle point. From there, you can either explore higher level concepts such as depth mapping or instead, as I chose, go deeper and showcase the reflectance map inner workings.

I personally believe that the Reflectance map with iso-brightness contours animation

really showcases the core idea behind the Photometric Stereo algorithm and is the most valuable part of this report.

6 Annex

```
249    L = np.asmatrix(light_sources)
250
251    Lt = L.transpose()
252    L_gram = np.matmul(Lt, L)
253    L_gram_inverse = np.linalg.inv(L_gram)
254    intermediate_g = np.matmul(L_gram_inverse, Lt)
255
256    for y in range(images_height):
257        for x in range(images_width):
258            g = np.matmul(intermediate_g, intensity_vectors[y][x])
259            albedos[y][x] = np.linalg.norm(g)
260            if albedos[y][x] != 0:
261                normals[y][x] = (g / albedos[y][x])[0]
262            else:
263                normals[y][x] = np.zeros_like(g)
```

Figure 9: Compact Python implementation of the least square method used to solve the equation

```
395  def compute_reflectance_at(x, y, light_source):
396      n = np.array([x,y, 1])
397      normed = np.linalg.norm(n)
398      if normed != 0 :
399          n /= normed
400      return np.dot(n, light_source)
401
402  def lambertian_reflectance_map(light_direction, resolution=500):
403      norm = np.linalg.norm(light_direction)
404      if norm != 0 :
405          light_direction /= norm
406
407      lookup = np.linspace(-plot_size, plot_size, resolution)
408      reflectance_map = []
409      for y in range(resolution):
410          reflectance_map.append([])
411          for x in range(resolution):
412              reflectance_map[y].append(compute_reflectance_at(lookup[x], lookup[y], light_direction))
413
414      reflectance_map = np.clip(reflectance_map, 0, 1)
415
416      return reflectance_map
```

Figure 10: Implementation for the reflectance map computation

```
100  def create_animation(light_sources, intensities, output_filename='animation.gif'):
101      fig, ax = plt.subplots()
102      ax.set_title('Lambertian Reflectance Map with Iso-Brightness Contour\nat the Measured Intensity Level for each Image given Light Direction')
103      ax.set_xlabel('X')
104      ax.set_ylabel('Y')
105      ax.axhline(0, color='black', linewidth=1)
106      ax.axvline(0, color='black', linewidth=1)
107      ax.grid(True)
108
109      def plot_reflectance_map(reflectance_map, level_val):
110          global color_index
111          X, Y = np.meshgrid(np.linspace(-plot_size, plot_size, len(reflectance_map[0])),
112                             np.linspace(-plot_size, plot_size, len(reflectance_map[0])))
113          c = ax.contour(X, Y, reflectance_map, colors=[colors[color_index]], levels=[level_val])
114          plt.clabel(c, inline=True, fontsize=10, fmt='%1.2f', colors='black')
115          color_index = color_index + 1
116
117      def animate(i):
118          ax.set_title('Lambertian Reflectance Map with Iso-Brightness Contour\nat the Measured Intensity Level for each Image given Light Direction')
119          ax.set_xlabel('X')
120          ax.set_ylabel('Y')
121          ax.axhline(0, color='black', linewidth=1)
122          ax.axvline(0, color='black', linewidth=1)
123          ax.grid(True)
124
125          light_direction = light_sources[i]
126          reflectance_map = lambertian_reflectance_map(light_direction)
127          plot_reflectance_map(reflectance_map, intensities[i]/255)
128          return ax
129
130      anim = animation.FuncAnimation(fig, animate, frames=len(light_sources), repeat=False)
131      anim.save(output_filename, writer='imagemagick', fps=2)
132
133  create_animation(light_sources, intensities)
```

Figure 11: Implementation to create the animation for the reflectance map

References

[1] Y. Xiong, A. Chakrabarti, R. Basri, S. J. Gortler, D. W. Jacobs, and T. Zickler, "From shading to local shape," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 1, pp. 67–79, 2015.