The 8th France-Japan and
6th Europe-Asia Congress on Mechatronics
November 22-24, 2010, Yokohama, Japan

Pyramidal Normal Map Integration for Real-time Photometric Stereo

Vincent Nozick
Gaspard Monge Institute, UMR 8049
Paris-Est Marne-la-Vallee University, France
Email: vnozick@univ-mlv.fr

***Abstract*—This paper presents a real-time photometric stereo method designed for online 3D telecommunications. This method requires only consumer grade hardware such as a webcam and a GPU. The lighting conditions are controlled from the computer screen and the photometric reconstruction is performed on the GPU. Our method reaches real-time rendering thanks to a pyramidal integration of the normal maps based on an iterative scheme. Moreover, we propose a method to relax the constraint on the synchronization between the light source controller and the camera.**

I. Introduction

The development of 3D-TV and autostereoscopic displays is a promising source of entertainment. This new technologies will probably make 3D telecommunications be a reality at home in a near future. In that context, we present a method to recover the 3D structure of an human face by photometric means. Our method requires only common consumer grade hardware and is designed to make users communicate with a 3D perception in real-time. Indeed, our method uses the light of the computer screen to control the lighting environment. The illuminated scene is captures from a webcam in order to perform a 3D reconstruction of the scene with a photometric technique computed on the GPU, as show on Figure 1.

II. 3D reconstruction

There exist various techniques that perform 3D reconstruction from videos in real-time. Most of them require several calibrated cameras, like Depth from stereo methods [1] that compute disparity maps using pixel correspondences from two views. If the cameras are fully calibrated, the link between the disparity map and the 3D reconstruction is straightforward. The extension of depth from stereo to more than two cameras, known as multi-view stereo [2], is ill suited to run in real-time. The visual hull [3], also known as shape from silhouettes, extracts the silhouette of a considered subject from every input image. Then, the 3D shape of this object is approximated by the union of the projected silhouettes. The plane-sweep algorithm [4] also requires a set of calibrated cameras, the 3D reconstruction of the scene can be computed in real-time using a discretization of the scene with parallel planes.

Some other methods can compute a 3D reconstruction from a single video stream. Optical flow methods [5] compute a vector field that describe the apparent motion of objects in the image. These methods can lead to a 3D reconstruction but fail if the scene is static.

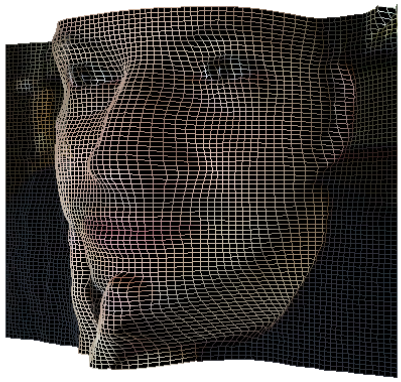


Fig. 1. 3D photometric reconstruction computed in real-time.

Structured light consists in projecting a known light pattern on the subject to be reconstructed. A recent state of the art can be found on [6]. These methods perform accurate 3D reconstruction however they are ill suited to run in real-time.

Finally, photometric techniques can recover the 3D shape of a scene from the image intensity (brightness). The photometric problem can be considered by two approaches: shape from shading techniques that use only one image (as presented in [7], [8]) and photometric stereo techniques that require several images of the same scene under different lighting condition. Since our method belongs to the latter family, we expose the related works on the next section.

III. Photometric reconstruction : Overview

First introduced by Woodham [9], the purpose of photometric stereo is to estimate the surface normals of objects by observing them under different lighting conditions. The surface normal is then integrated to generate a 3D reconstruction.

As an improvement, Mallick et al. [10] presented a photometric stereo method that handles specular reflectance. To overcome the problem of the lighting conditions information that are commonly required, Hallinan [11] or Basri et al. [12] propose a low-dimensional illumination representation of the scene under arbitrary light conditions. Woodham [13] introduces one of the first method to reach real-time photometric reconstruction using three spectrally different light sources from three distinct positions. In the same way, Hernandez and al. [14] use spatially separated red, green and blue light sources

to estimate a dense depth map from a untextured non-rigid surface. By merging geometric and colorimetric calibration data, they provide a mapping from color intensity to normal map. Then, a depth map is obtained by integrating the normal map.

Malzbender et al. [15] reach real-time photometric reconstruction using a GPU implementation of a program that controls a lighting device synchronized with a high speed video camera. The main contribution of this work is a normal computation that enhance the shape details.

According to our purpose, i.e. real-time photometric stereo using consumer grade hardware, we can observe that all these methods are too dependent to a specific acquisition device or lack in the speed computation.

IV. Photometric stereo from screen light

We present a GPU based method to recover the 3D structure of an human face by photometric stereo means. This method uses a standard web camera mounted on the computer screen. The light of the screen is used as the main light source and will determine the illumination conditions. The camera captures consecutive images under these different lighting conditions in order to compute a normal map of the scene that can be integrated to a depth map and thus a 3D reconstruction.

This paper is an extension of Nozick et al. [16] designed to run in real-time with only consumer grade hardware. Our main contribution are:

- a fast, robust and automatic method to retrieve the lighting conditions from imperfect illumination data. More precisely, our method is robust to camera-screen synchronization issues.
- a consequent speed up for the normal map integration using an iterative pyramidal approach. This method is especially well suited for an implementation on the GPU.

The rest of the paper is organized as follows: the next sections detail each step of our method. Section VIII describes the real-time implementation of the method on GPU. Finally some results are presented in Section IX and the conclusion is discussed in Section X.

V. Lighting from screen

The computer screen is used as a large programmable light source area and provides various lighting conditions for the photometric reconstruction (see Figure 2 as an example of four consecutive lighting setups). The illuminated scene is captured by a camera attached on the screen and a reconstruction is performed for every new captured frame. Indeed, the new acquired image is transfered to the system and inserted to a "captured images set". This new image will replace the oldest image of the set and thus, the system do not have to wait for a new set of new views between two consecutive reconstructions, but just update the current set. To optimize the quality of the reconstruction, every image of the set should be taken under different lighting condition. A set should contain at least three images but using more images (ideally, as much as possible) improves the reconstruction quality. However, using too much images will increase the computation time and prevent from real-time performance. Considering these constraints, using four input images seems to be a good compromise. Figure 3 depicts a set of four captured images under four lighting conditions. To ensure enough lighting, the light source should not be punctual. A lighting condition possibility can be to sequentially illuminate every top, right, bottom and left half part of the screen. As we will discuss in section VI-B, any other light configuration (different from one image to the other) is acceptable if the camera and the screen are roughly synchronized. Finally, the ambient light of the scene should be reduced to the minimum to maximize the screen light contribution.

Fig. 2. A sample of different input images sequentially displayed on the screen to produce varying illumination conditions.

Fig. 3. Four images captured under four illumination conditions.

VI. Normal map computation

A. Eigen problem

Consider a set of N images of dimension $width \times height$ representing the same scene under various illumination. The images are converted to grayscale images and considered as one dimensional arrays. A matrix $\mathbf{A}$ with N rows and $width \times height$ columns is defined such that every row of $\mathbf{A}$ corresponds to an input image.

As presented by Hayakawa [17], the Singular Value Decomposition (SVD) of $\mathbf{A}$ can be used yo compute a surface normal map. Yuille et al. [18] improved this method by computing the SVD on the covariance matrix $\mathbf{A}\mathbf{A}^\top$ rather than on $\mathbf{A}$. According to the big size of $\mathbf{A}\mathbf{A}^\top$, the computation time required for the SVD may prevent from real-time rendering. An alternative approach is to compute the SVD of $\mathbf{A}^\top\mathbf{A}$ which is a $N \times N$ matrix. This approach is much faster however the eigenvector information is common for every pixel of the image.

As for a principal component analysis, the SVD of $\mathbf{A}^\top\mathbf{A}$ provides a set of N right eigenvectors associated to a set of non-negative singular values sorted in decreasing order. As mentioned in [16], these eigenvectors represent the photometric principal axis in the image referential. The geometric inter-

pretation of this eigen-system is to consider the eigenvectors as the light direction components in each input image. The three first eigenvectors associated to the three biggest singular values can be considered as 3D geometric coordinates. In practical terms, let the SVD of $\mathbf{A}^\top\mathbf{A}$ give the decomposition $\mathbf{A}^\top\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ and let $\mathbf{B}$ be the matrix composed by the 3 first predominant eigen-vectors (i.e. the 3 first rows of $\mathbf{V}^\top$). Let also define the image 3D referential where x-axis and y-axis define the image plane and the z-axis corresponds to a depth component.

$$\mathbf{V}^\top = \left[\begin{array}{c}\mathbf{B}\\ \hline \vdots\end{array}\right] \quad \mathbf{B} = \overbrace{\left[\begin{array}{cccc}\bullet & \bullet & \cdots & \bullet\\ \bullet & \bullet & \cdots & \bullet\\ \bullet & \bullet & \cdots & \bullet\end{array}\right]}^{N\ light\ sources}\left.\begin{array}{l} z\\ x\ or\ y\\ y\ or\ x\end{array}\right.$$

The i^{th} column of $\mathbf{B}$ can be interpreted as the light direction on the i^{th} image. The first raw of $\mathbf{B}$, corresponding to the biggest singular value, represents the z-axis that is always predominant. The last two rows correspond to the x and y axis but their associated singular value are usually similar and thus we can not distinguish which is x-axis and which is y-axis by just using the SVD. Moreover, the sign of the eigen-vectors is also undetermined.

B. Light source identification

In this section, we propose a method to solve the SVD $x-y$-axis ambiguity using the light source condition. Indeed, since the images displayed on the screen and the images acquired by the camera are approximatively synchronized, a comparison between the light direction proposed by $\mathbf{B}$ and the light direction evaluated on the displayed image can lead to an axis identification.

Let's consider for each displayed image the average 2D light position in the canonical image referential (i.e. every image coordinate component ranges is $[-1;1]$). Then we can define the matrix $\mathbf{L}$ which columns are the N 2D average direction of the light contribution in the N displayed images, in the same order than for $\mathbf{A}$.

$$\mathbf{L} = \overbrace{\left[\begin{array}{cccc}\bullet & \bullet & \cdots & \bullet\\ \bullet & \bullet & \cdots & \bullet\end{array}\right]}^{N\ displayed\ images}\left.\begin{array}{l} x\\ y\end{array}\right.$$

Now let's consider the matrix $\mathbf{B}^\blacktriangledown$ corresponding to the two last rows of $\mathbf{B}$, i.e. the undetermined x and y-axis.

$$\mathbf{B} = \left[\begin{array}{c}\cdots\\ \hline \mathbf{B}^\blacktriangledown\end{array}\right] \quad \mathbf{B}^\blacktriangledown = \overbrace{\left[\begin{array}{cccc}\bullet & \bullet & \cdots & \bullet\\ \bullet & \bullet & \cdots & \bullet\end{array}\right]}^{N\ light\ sources}\left.\begin{array}{l} x\ or\ y\\ y\ or\ x\end{array}\right.$$

If each row of $\mathbf{L}$ is normalized, one can expect that $\mathbf{B}^\blacktriangledown$ is a row (and possibly sign) permutation of $\mathbf{L}$ since both $\mathbf{L}$ and $\mathbf{B}^\blacktriangledown$ respectively represent the light orientation in the displayed image and in the captured image. In that perspective, the matrix $\mathbf{P} = (\mathbf{L}^\top\mathbf{B}^\blacktriangledown)^{-\top}$ should be a permutation of identity $\mathbf{Id}_2$, including the sign variation. In practice, this will not be completely the case since the display and the camera are not perfectly synchronized, but the nearby integer matrix of $\mathbf{P}$, noted $\overline{\mathbf{P}}$, will be the nearest permutation matrix from $\mathbf{P}$. Finally, the permutation matrix $\mathbf{M}$ that identifies the x, y and z axis, including the sign correction, can be defined as follows:

$$\mathbf{M} = \left[\begin{array}{cc}\mathbf{0}_2 & \overline{\mathbf{P}}\\ 1 & \mathbf{0}_2^\top\end{array}\right]$$

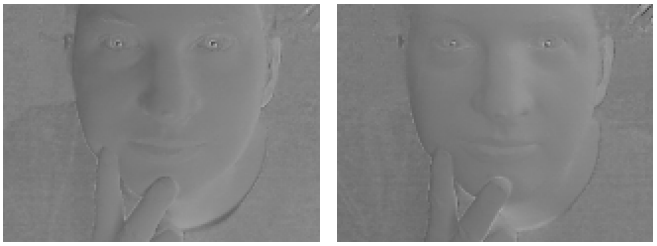
Hence, the matrix $\mathbf{MB}$ will represent the identified and signed-corrected eigen-vectors of $\mathbf{A}^\top\mathbf{A}$.

C. From Images to Normals

Once the x-y-z-axis are identified, the next step is to compute the normal map from the grayscale color of the N captured images and the matrix $\mathbf{MB}$. Let $\mathbf{p}_i = \mathbf{A}_i$ (the i^{st} column of $\mathbf{A}$) be the photometric contribution of the pixel i from the N grayscale input images. Then the normal $\mathbf{n}_i = (n_x, n_y, n_z)^\top$ of the pixel i can computed by:

$$\mathbf{n}_i = \mathbf{MB}\mathbf{p}_i$$

Then, like in any standard photometric stereo method, the norm of the normal $\mathbf{n}_i$ at the pixel i corresponds to the albedo $\rho_i = \|\mathbf{n}_i\|$ of this pixel. In some cases, it can happen that $\|\mathbf{n}_i\| = 0$. This corresponds to a lack of photometric information, the normal is then set to $(0,0,1)^\top$ and albedo to 0. For the following parts, we assume that for any pixel i, we have $\|\mathbf{n}_i\| = 1$. Figure 4 shows a sample of x-y-normal maps.



(a) X component (b) Y component

Fig. 4. Example of normal surface map.

VII. Depth map computation

A. Normal map integration

A perfect normal map should fit the integrability constraint or the zero-curl constraint. In practice, the normal map generated in section VI-C is noisy and the zero-curl constraint is usually not satisfied. Petrovic et al. [19] propose a method to enforce the integrability of the normal map. Another solution is to integrate the normal map with an iterative method that conserves the integrability constraint, as proposed by Basri and Jacobs [12]. Like in Nozick et al. [16], we compute the depth value $z = z_{i,j}$ of an object at the pixel position (i,j) using an iterative Gauss-Seidel scheme under K iterations $k = 1...K$ from the following equation:

$$\begin{array}{rcl} z_{i,j}^{k+1} = & \frac{1}{4} & [z_{i+1,j}^k + z_{i-1,j}^k + z_{i,j+1}^k + z_{i,j-1}^k \\ & & +n_{i-1,j}^x - n_{i,j}^x + n_{i,j-1}^y - n_{i,j}^y] \end{array}$$

To ensure a constant reference depth for all the reconstructions, the relaxation process is not applied on the corners of the depth map. The final reconstruction will be defined up to scale factor, as defined in Belhumeur et al. [20].

B. A Pyramidal computation

The depth map computation is a key part to reach real-time rendering. To speed up the normal map integration, we propose a pyramidal computation of the iterative process. Indeed, we apply the first iterations on a depth map image with a very low resolution in order to obtain a fast estimation of the depth map. Then, this depth map is refined with a higher resolution for several iterations. This process is repeated until the maximum resolution is reached and computed. The iterative equation of part VII-A becomes :

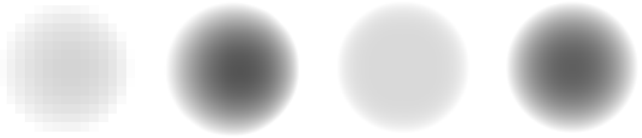
$$z_{i,j}^{k+1} = \frac{1}{4} [z_{i+1,j}^{k} + z_{i-1,j}^{k} + z_{i,j+1}^{k} + z_{i,j-1}^{k} + \Delta x(n_{i-1,j}^{x} - n_{i,j}^{x}) + \Delta y(n_{i,j-1}^{y} - n_{i,j}^{y})]$$

where Δx and Δy are the numerical steps representing the pyramid level scale. For a mipmap level l, we get:

$$\Delta x = \Delta y = 2^l$$

i.e. big for the first iterations and equal to 1 for the last pyramid level.

This optimization speeds up the integration for two reasons. First, the convergence speed increases because the first iterations are not accurate, but very fast since Δx and Δy are bigger. Thus, this approach requires much less iterations than the standard method to reach the same result, as shown in Figure 5.



(a) 20 iter. (b) 70 iter. (c) 500 iter. (d) 2600 iter.

Fig. 5. Depth map obtained during an iterative normal map integration of an hemisphere. Our method after 20 iterations (a) and 70 iterations (b). To reach the same result, the standard method requires respectively 500 iterations (c) and 2600 iterations (d).

Moreover, the first iterations on small images are much faster to perform since very few pixels are concerned. Indeed, even if we compute the same number of iterations as with the standard method, the pyramidal approach computation time on an image with the resolution $(n \times n)$ is in $\mathcal{O}(Kn.ln(n))$ when the standard one is $\mathcal{O}(Kn^2)$. We can note that this approach involves the computation of all levels of mipmap pyramid, or at least minifying interpolation. However, our application is designed to be implemented on the GPU, hence the mipmapping or the minifying interpolation computation is extremely fast.

Figure 6 presents the depth map computed from the normal map shown on Figure 4.

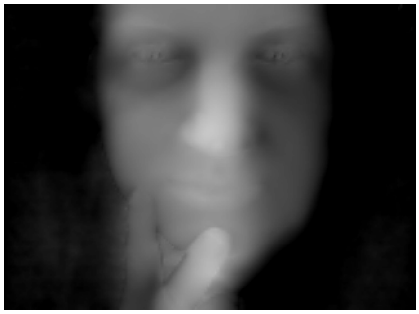


Fig. 6. Depth map computed from a normal maps computation.

VIII. Implementation

In our method, the first issue concerns the image to be displayed on the screen. To ensure correct photometric light conditions, we display a white area on the border of a black screen, rotating around the center of the image. The rotation speed is adapted according to the camera frame rate acquisition to make a turn in approximatively four acquired frames, since we decided to use four input images for the reconstruction. The camera and the screen should be as synchronized as possible, but inaccuracies are corrected by the light identification process presented in section VI-B.

Then, for each new reconstruction, the latest grabbed image is converted in grayscale and inserted in the matrix $\mathbf{A}$. The $\mathbf{A}^\top\mathbf{A}$ matrix is updated, but not with a standard matrix multiplication. Indeed, only one row and one column should be updated and since $\mathbf{A}^\top\mathbf{A}$ is symmetric, these row and column can be updated simultaneously. Then the SVD of $\mathbf{A}^\top\mathbf{A}$ and the eigen-vectors identification are computed on CPU. Finally, the x-y-z normal maps and the albedo image are computed on the GPU and stored as float textures.

The next step concerns the depth map computation. The Gauss-Seidel relaxation is very well suited to be implemented on GPU since every pixel can be processed simultaneously. Every iteration is performed off-screen by the GPU using frame buffer objects (FBO). Two textures are used alternatively to contain the depth map of the previous iteration or to be attached to the FBO for the current iteration rendering. The resolution of the FBO varies according to the level of pyramid. The relaxation process uses the equation presented on section VII-B. Concerning the normal map textures, we can note that mipmapping is not required if the interpolation minify filter is enabled during the texture access.

Finally, a dense flat mesh is projected on the screen (Figure 7). Every vertex depth is computed from the depth map using a vertex program. Then, the meshes are textured with the albedo image using a fragment program. This rendering method does not require any transfer of the depth map between the GPU and the main memory.

IX. Results

We have implemented our system on a PC Intel core duo 3.16 GHz with an nVidia Quadro FX 1700. Our program is implemented in C++, OpenGL and GLSL. The video

Fig. 7. Textured mesh.

Fig. 8. Real-time 3D reconstruction.

acquisition is performed by one USB Logitech fusion web camera connected to the computer. With a 320×240 resolution, the acquisition frame rate reaches 20 frames per second. All the computations are make within the image size of 320×240 pixels. Figure 8 presents a 3D reconstruction performed in real-time.

The depth map computation computed with the GPU parallelism combined to our pyramidal method usually requires 70 iterations to obtain a convergent solution. During our tests, the main bottleneck is the camera frame rate limitations.

X. Conclusion

In this paper we present a real time implementation on GPU of a 3D facial reconstruction designed for consumer grade application by simply using a standard web camera and the computer screen.

The proposed method is robust to an approximative knowledge of the light conditions and can reconstruct the 3D geometrical surface with only one webcam.

Thanks to our GPU implementation of the normal map relaxation, this method can reach real-time rendering.

However a limitation of this method concerns the screen light contribution that must be predominant over the ambient light.

References

[1] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, pp. 7–42, 2002. 1

[2] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 519–528. 1

[3] M. A. Magnor, *Video-based Rendering*, A. K. P. Ltd, Ed. A K Peters Ltd, 2005. 1

[4] V. Nozick and H. Saito, "On-line free-viewpoint video : From single to multiple view rendering," *Journal of Automation and Computing (IJAC)*, vol. 5, no. 3, pp. 257–267, 2008. 1

[5] S. Baker, S. Roth, D. Scharstein, M. Black, J. Lewis, and R. Szeliski, "A database and evaluation methodology for optical flow," in *ICCV07*, 2007, pp. 1–8. 1

[6] T. Weise, B. Leibe, and L. V. Gool, "Fast 3d scanning with automatic motion compensation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'07)*, June 2007. 1

[7] L. L. Kontsevich, A. P. Petrov, and I. S. Vergelskaya, "Reconstruction of shape from shading in color images," *Journal of Optic Society, A*, vol. 11, no. 3, pp. 1047–1052, 1994. 1

[8] R. Zhang, P. Tsai, J. Cryer, and M. Shah, "Shape from shading: A survey," *PAMI*, vol. 21, no. 8, pp. 690–706, August 1999. 1

[9] R. J. Woodham, *Photometric method for determining surface orientation from multiple images*. Cambridge, MA, USA: MIT Press, 1989. 1

[10] S. P. Mallick, T. E. Zickler, D. J. Kriegman, and P. N. Belhumeur, "Beyond lambert: Reconstructing specular surfaces using color," *Computer Vision and Pattern Recognition, 2005. CVPR 2005*, vol. 2, pp. 619–626, 2005. 1

[11] P. Hallinan, "A low-dimensional representation of human faces for arbitrary lighting conditions," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '94)*, pp. 995–999, Jun. 1994. 1

[12] R. Basri, D. Jacobs, and I. Kemelmacher, "Photometric stereo with general, unknown lighting." *International Journal of Computer Vision*, vol. 72, no. 3, pp. 239–257, 2007. 1, 3

[13] R. J. Woodham, "Gradient and curvature from the photometric stereo method, including local confidence estimation," *Journal of the Optical Society of America, A*, vol. 11, no. 11, pp. 3050–3068, 1994. 1

[14] C. Hernandez, G. Vogiatzis, G. Brostow, B. Stenger, and R. Cipolla, "Non-rigid photometric stereo with colored lights," in *Proc. IEEE 11th International Conference on Computer Vision ICCV 2007*, 2007, pp. 1–8. 1

[15] T. Malzbender, B. Wilburn, D. Gelb, and B. Ambrisco, "Surface enhancement using real-time photometric stereo and reflectance transformation," in *Eurographics Symposium on Rendering 2006, Nicosia, Cyprus*, 2006. 2

[16] V. Nozick, I. Daribo, and H. Saito, "Gpu-based photometric reconstruction from screen light," in *8th Annual International Conference on Artificial reality and Telexistence (ICAT2008)*, dec 2008, pp. 242–245. 2, 3

[17] H. Hideki, "Photometric stereo under a light-source with arbitrary motion," *JOSA-A*, vol. 11, no. 11, pp. 3079–3089, November 1994. 2

[18] A. Yuille, D. Snow, R. Epstein, and P. Belhumeur, "Determining generative models of objects under varying illumination: Shape and albedo from multiple images using SVD and integrability," *International Journal of Computer Vision*, vol. 35, no. 3, pp. 203–222, 1999. 2

[19] N. Petrovic, I. Cohen, B. J. Frey, R. Koetter, and T. S. Huang, "Enforcing integrability for surface reconstruction algorithms using belief propagation in graphical models," in *In: Proc. Conf. Computer Vision and Pattern Recognition*, 2001, pp. 743–748. 3

[20] P. N. Belhumeur, D. J. Kriegman, and A. L. Yuille, "The bas-relief ambiguity," in *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 1060–1072. 4