Interactive Material Replacement in Photographs

Steve Zelinka Hui Fang Michael Garland John C. Hart

Department of Computer Science
University of Illinois at Urbana–Champaign

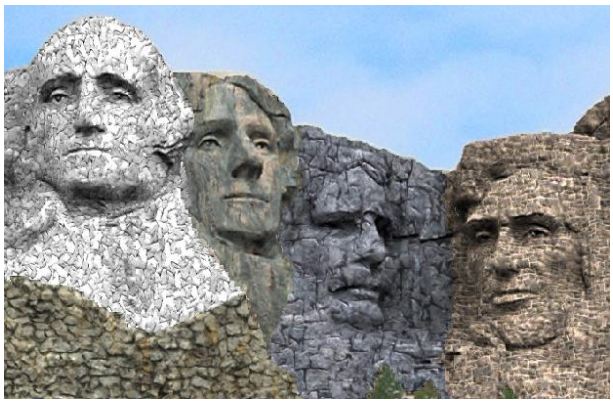

Figure 1: Replaced materials on Mount Rushmore. Total time, including user interaction, was under one minute.

Abstract

Material replacement has wide application throughout the entertainment industry, particularly for post-production make-up application or wardrobe adjustment. More generally, any low-cost mock-up object can be processed to have the appearance of expensive, high-quality materials. We demonstrate a new system that allows fast, intuitive material replacement in photographs. We extend recent work in object selection and fast texture synthesis, as well as develop a novel approach to shape-from-shading capable of handling objects with albedo changes. Each component of our system runs with interactive speed, allowing for easy experimentation and refinement of results.

Key words: material replacement, texture synthesis, image segmentation, jump maps, shape from shading

1 Introduction

In its simplest form, material replacement has been widely used for decades. Chroma-keying, or blue-screen matting [19], replaces objects of a particular colour (typically blue or green) with new objects or materials, and is a staple of movie and television production. While chroma-keying is most often used to replace the background, placing actors in new virtual sets, it is increasingly used in special effects production to replace simple mock-up objects with computer-generated imagery. The advantages to this approach over simply directly inserting the computer-generated imagery is that actors may naturally interact with the mock-up object, and the mock-up also produces lighting effects that are difficult to achieve with computer-generated imagery, such as casting shadows into the rest of the scene.

A key limitation of chroma-key systems is that one must know before-hand which objects and materials one wishes to replace in the final output, since only objects of a particular distinctive colour may be replaced. In this paper, we address a more general material replacement problem. Given any diffuse, possibly textured object within a photograph, and an image swatch of a desired material, we replace the apparent material of the object with the desired material. Thus, the makeup of an actor may be easily adjusted after a photo shoot if errors are discovered or a different look is desired. Logos or advertisements on clothing may be removed, or new colours and patterns applied. Internet-based vendors can show people how user-supplied objects would look using their fabrics, patterns, or upholstery. Since editing is fast and intuitive, there is no user penalty for experimenting with replacing different objects or using different textures, so the system is useable even by novices for personal use.

Textureshop [6] pioneered the idea of using a very simple but fast approach to shape-from-shading to recover a rough set of normals for an object in a photograph, and using these normals to distort texture synthesis over the object. They note that to some extent, texture variations mask the inaccuracy of the recovered normals, and so despite the low quality of the normals, they are still able to obtain visually pleasing results. However, Textureshop can be cumbersome to use, since object selection is based on manual outlining, and Graphcut Textures [9] is used as the texture synthesis engine. Manually outlining objects from photographs can be tedious, even when aided with tools such as intelligent scissors [14]. As Graphcut Textures are relatively slow to synthesize, experimentation with synthesis parameters is discouraged. Furthermore, the normal recovery algorithm used in Textureshop is highly inaccurate for all but uniformly coloured, dif-

fuse objects, often requiring a non-trivial amount of user interaction to fix normal recovery errors.

Our system makes three important, practical contributions over the state of the art:

- Integration of advanced object selection [11] with a Gaussian mixture model [18] to allow simple, fast, and intuitive object selection in images.
- A novel shape-from-shading algorithm making explicit use of the Gaussian mixture model of the object selection phase. The most reliable colour cluster drives the normal recovery, producing higher quality normals and allowing for *textured* input surfaces.
- The use of jump map-based texture synthesis, which, while lower quality than other algorithms, runs in interactive time, allowing for easy experimentation with different textures and texturing parameters, such as scale and orientation.

Each step of the resulting system – selection, normal recovery, and texture synthesis – runs in interactive time, allowing for more intensive experimentation and refinement of parameters. Users can update the selection area, adjust recovered normals, and change textures or texturing parameters, and view the effects of their changes interactively. Previous methods for material replacement require a non-trivial amount of computation time. This lag time, especially during texture synthesis, is a severe detriment to experimentation. It is also important to note that once a user is happy with the parameters of the algorithm, better quality texturing and rendering may be used to produce a final high-quality result.

2 Related Work

Recently, graph cut-based techniques [2, 11, 18] have been very successfully applied in selecting objects from images. Previous methods [14, 15, 7] require the user to trace the boundary of the object directly, which can often be tedious and error-prone. In contrast, graph cut-based approaches use strokes within or outside the object, and compute the actual boundary by formulating an energy minimization problem to be solved by computing a minimum cost graph cut. Grabcut [18] additionally uses Gaussian mixtures to statistically model each region, and requires only a box to be placed around the object. Lazy snapping [11] instead pre-segments the image to reduce the size of the graph cut problem and achieve real-time updates after each brush stroke. While the focus of these methods is on hard segmentation, high-quality mattes can be extracted with both Grabcut or Poisson matting [21].

There are a variety of methods for computing the shape of a 3D object from its shading in a 2D image [8]. Bichsel and Pentland [1] propagate depth from the brightest points to the rest of the surface using a minimum downhill principle. Most similar to ours, Lee and Rosenfeld [10] assume local surface regions are spherical patches, yielding fast but poor results. More recently, Cho and Chow [3] iteratively combine surfaces independently recovered from each colour channel of an image. Oh *et al.* [16] extract lighting from texture via bilateral filtering. Zhang *et al.* [27] survey and compare several shape-from-shading methods, discovering generally poor quality and limited generality. However, our results reinforce the conclusion of Fang and Hart [6], that high quality normals are not required to produce good synthesized results. For objects with regular or near-regular textures, Liu *et al.* [12] demonstrate excellent results by manually fitting a regular lattice to the texture.

Recent texture synthesis methods can be classified into two approaches. Pixel-based techniques [5] iteratively copy pixels by searching for the best match for the current output neighbourhood within the input image. Patch-based methods [4, 9] instead place entire patches of texture while attempting to find a good boundary between the existing output and the new patch. Methods to texture 3D surfaces have been demonstrated by generalizing both pixel-based [23, 22, 26, 24] and patch-based techniques [17, 20, 13]. For material replacement, we do not necessarily have a coherent 3D surface, so the more local pixel-based methods are a more natural solution. Texton maps [26], utilize a coarse map of prominent texture features to yield the highest quality results for most textures. Jump map-based synthesis [24] generally produces the lowest-quality but fastest results, and is the only technique to produce results in interactive time.

3 System Overview

Our material replacement system is composed of three main components. First, the user *selects* an object from the input image. Once satisfied with the selection, *normal recovery* with our novel colour cluster-based shape-from-shading algorithm approximates the surface normal at each pixel of the object. Finally, these normals are used to produce a set of distortions to be applied during jump map-based *texture synthesis*. In the following sections, we describe each of these steps in detail.

4 Object Selection

Given a photograph containing objects needing material replacement, the first task is to identify the objects from the background. In our system, we use Lazy Snapping [11] as the basic interaction mechanism, as it offers the highest interactivity and natural refinement. Lazy Snapping is a graph cut-based approach, posing the im-

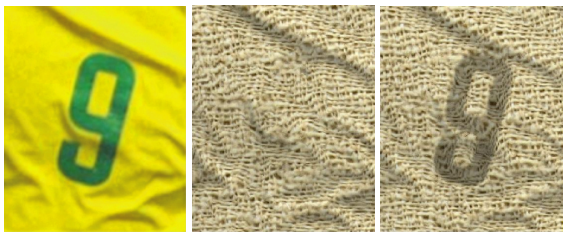

Figure 2: From left to right: original image; textured result with our texture-sensitive procedure; result without texture-awareness.

age segmentation problem as energy minimization solved via minimum cost graph cut. The energy is a sum of two terms: a smoothness term, which penalizes segmentation boundaries cutting smooth areas of the image, and an approximation error term. The system statistically approximates both the foreground and background, and the approximation error term ensures pixels more likely to be fit by one model over the other are penalized for assuming contradicting labels. The key idea of Lazy Snapping over alternate approaches is to pre-segment areas of the image unlikely to be cut in object selection. The graph problem uses the resulting segments, rather than individual pixels, as nodes, so its size (and solution time) is vastly reduced.

To augment Lazy Snapping for material replacement, we first note that as we are concerned with textured, diffuse objects, it is especially important that colour images are handled well. Lazy Snapping's approximation model is a *k*-means clustering of the pixels that are selected by the user as definite-foreground or definite-background. We instead use a limited Gaussian mixture. Under a normal Gaussian mixture, each pixel is a weighted combination of the colours in the mixture, each of which is modelled with a Gaussian distribution. Following GrabCut [18], we assume each pixel is drawn from only one colour of the mixture. In comparison to *k*-means clustering, the limited Gaussian mixtures include the covariance of each colour cluster, allowing for a slightly more accurate approximation error term during segmentation. For efficiency, we do not iterate our algorithm, but once segmentation is complete, we update the foreground mixture according to the pixels actually selected. As discussed below, this mixture is used for better normal recovery.

5 Normal Recovery

Once an object is identified, the next phase of material replacement is to recover its normals. We generalize Textureshop-style normal recovery to textured objects. For simplicity of description, we assume that the light source is at the viewer; the extension to other positions is straightforward, though we require the user to specify the light position if not at the viewer. The basic idea is to assume a Lambertian reflection model, so that at the brightest point in the image, the surface normal points to the light source, while the darkest point of the image lies on a silhouette and its normal aligns with the reverse image gradient. For other pixels, the *x* and *y* components of the normal are given by the reverse image gradient, while the *z* component is linearly mapped from its luminance, with the mapping fixed by the brightest and darkest points (if l_p is the luminance of pixel p, and l_{min} and l_{max} are the minimum and maximum observed luminances, then $z_p = (l_p - l_{min})/(l_{max} - l_{min})$). In general, the resulting normals are very approximate, but after some smoothing are quite useable for our application.

We improve on this approach by making use of the Gaussian mixture model for the selected object. Each pixel has an approximating colour cluster from the Gaussian mixture. We first recover the normals over pixels of the most *reliable* colour cluster in the mixture using the above approach. Then, normals are propagated across the image-space boundaries of that cluster to seed the normal recovery of each subsequent colour cluster. After all clusters are processed, some smoothing of the recovered normals is typically required.

The reliability r_c of each colour cluster c is based on its area a_c (number of pixels) and dynamic range v_c (luminance variation): $r_c = \alpha * a_c + (1-\alpha) * v_c$, where α is a user parameter (for all images in this paper, we used $\alpha = 0.75$). Intuitively, we desire the largest cluster as it is most visually important, while we desire the cluster of largest dynamic range as it likely contains a wide normal variation in accordance with our assumptions above.

To propagate normals across cluster boundaries, we assume the change in colour is entirely due to albedo change, and the underlying surface remains smooth, allowing us to copy normals across boundaries. We then iteratively choose the cluster with the longest boundary with already-processed clusters. The *x* and *y* components of the normals in this new cluster remain the reverse image gradient, and the *z* components are still derived from a linear mapping of luminance. However, each pixel on the boundary of the cluster is a sample of this mapping (its luminance and normal being copied from the neighbouring cluster). We thus use these samples to compute a least-squares fit of the luminance-to-*z* mapping.

In some cases, pixels on the boundary between two colour clusters may be blends of the cluster means. Here, boundary propagation will not work since the boundary pixels will differ significantly from the rest of the colour cluster, and the computed luminance-to-*z* mapping will be highly inaccurate. We instead first erode the boundaries and extrapolate from each cluster to a sharp boundary. Generally, we erode until the pixels are sufficiently

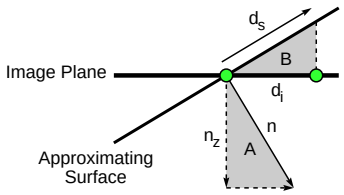

Figure 3: The image plane is viewed from above, and two neighbouring pixels are in green. The surface between the pixels is approximated by a plane whose normal is $\mathbf{n}$*, the average of the recovered normals at each pixel. The texture-space distance between the two pixels,* $||\mathbf{d}_s||$*, is derived from the similar triangles A and B.*

close to the mean of the cluster, or until a user-specified maximum erosion is reached.

As can be seen in Figure 2, our results are much better than simply recovering the normals over the entire object. Note that we modulate the textured output according to the luminance of the object in order to preserve its shading. With albedo changes, the means of various colour clusters may have different luminances, and thus the original luminances are not adequate for this task (this is the main reason the "9" is very clearly visible in the rightmost image of Figure 2). Since the luminance is directly proportional to the z component of the surface normal, we have effectively recovered consistent luminance levels for the object as well, and we use these to modulate the textured output.

6 Texture Synthesis

Once normals for the image have been recovered, the final step for material replacement is to synthesize new texture. We use texture synthesis based on *jump maps* [25]. While jump map-based texture synthesis produces lower quality textures than alternate techniques, it is the fastest known method for synthesizing textures on surfaces. Thus, users can interactively experiment with different textures, or texture orientations or scales.

A jump map is a parallel image to a texture which records for each pixel a small set of similar pixels within the image. Texture is produced from a seed point by iteratively copying pixels from the input to the output. Occasionally, or near input image boundaries, a jump is made using the jump map, and copying proceeds instead from the jump destination. On surfaces, the method is similar, walking across vertices instead of pixels. To accomodate the irregular topology and sampling of surfaces, each edge is individually mapped into a 2D texture space offset according to a supplied orientation field and texture scale. Instead of moving one pixel in the input, moving across vertices moves the corresponding offset.

We adapt jump map-based texture synthesis for material replacement in images by treating the image selection as a surface with varying edge lengths between pixels. Since the synthesis procedure considers only the immediate neighbours when synthesizing a given pixel, we need not have a physically-realizable surface; this is a distinct advantage over Textureshop, whose Graphcut-based approach required the recovery of consistent surface patches. The remaining question is how to compute texture space offsets between neighbouring pixels.

We assume that at each pixel the user has specified a scale, s_p, as the number of texture pixels per target image pixel, and an angle θ_p, which specifies the orientation of the texture with respect to the target image. These are typically specified globally, but may be gradually varied across the image as well [26] if desired. From the previous step, we have a recovered normal $\mathbf{n}_p$ at each pixel as well. On each edge between neighbouring pixels, we first compute average normals, scales, and orientations, $\mathbf{n}$, s, θ. We approximate the local surface by a plane whose normal is $\mathbf{n}$. As shown in Figure 3, the length of $\mathbf{d}_s$, the distance on the surface (equivalently, in texture space) between these two pixels, can be derived via similar triangles. The direction of $\mathbf{d}_s$ is given by projecting the image-space offset between the pixels ($\mathbf{d}_i$) onto the approximating plane. Thus, we have:

$$\mathbf{d}_s = \frac{s||\mathbf{d}_i||}{||\mathbf{n}_z||} \frac{\mathbf{d}_i - (\mathbf{n}\mathbf{d}_i)\mathbf{n}}{||\mathbf{d}_i - (\mathbf{n}\mathbf{d}_i)\mathbf{n}||} \tag{1}$$

Finally, we construct a basis for texture space by rotating the image axes by θ (the user-specified orientation), and projecting them onto the approximating plane. The 3D vector $\mathbf{d}_s$ is finally projected into this basis to get the texture space offset. The resulting graph of offsets is not necessarily coherent, but as shown in the next section, it is sufficient to give good synthesis results.

Texture synthesis produces texture coordinates at each pixel of the object. We render a final result by splatting a small texture patch at each pixel, both filtering the texture and obscuring discontinuities. As in Textureshop, normals may also be recovered for the source texture, and splatted together with the recovered object normals for bump mapping. Finally, the result is modulated by the recovered luminance to preserve the object's shading.

7 Results and Discussion

All of the results presented in this paper took about a minute or two to generate, mostly for user interaction. Selection occurs in real-time with each brush stroke, normal recovery takes up to five seconds, and synthesis and rendering takes up to two seconds; both operations scale linearly with the number of pixels, while normal recovery

Figure 4: Selecting and retexturing multiple objects is fast and efficient with our system.

Figure 5: Jump map-based texture synthesis allows for easy experimentation with different textures, texture scales, and texture orientations. As shown in the video, each image is produced with virtually no lag time.

also depends on the number of clusters in the Gaussian mixture. These timings, as well as the real-time captures in the video, are on a 1.8 GHz G5 machine. As shown in Figures 1, 4 and 5, as well as the accompanying video, the speed of the system easily allows for selecting multiple objects from a scene, using multiple textures, and varying texturing parameters. Updates to the result occur interactively, usually under a second.

We demonstrate results of material replacement on objects with albedo changes in Figures 2 and 6. Note that in the dancing image (Figure 6, bottom), the high-frequency colour changes in the shirt of the left dancer are inaccurately treated as a single colour cluster by our system, producing some brightness artifacts, but the result is still quite convincing. With Textureshop, the recovered luminances are incorrect (Figure 7).

The main limitations of our system are shown in Figure 8. Clustering can succeed even on high frequency albedo changes, as with the patterned shirt example, but the resulting clusters are not coherent enough to provide good shape-from-shading results. Thus, the luminance

Figure 6: Material replacement with albedo changes.

Figure 7: Results using Textureshop-style recovery. The textures are severely distorted and the luminance incorrect in areas with different albedo.

Figure 8: Material replacement failures. Left to right: original image, colour clusters, recovered brightnesses.

recovery is incorrect, revealing the original pattern instead of removing it.

A more subtle problem can cause the clustering itself to fail. In the dish cloth example, which uses 3 clusters, the shading variation outweighs the albedo changes, so pixels with different albedo are incorrectly clustered together, causing later steps in the algorithm to fail. Clustering in different colour spaces can help, but no one colour space is good over a wide variety of images. We expect a more fruitful approach is for the user to provide some sort of extra information to make the recovery more accurate. From our results, simply specifying the expected number of colour clusters is not sufficient (as using too many clusters can lead to the same problem, we leave the number of clusters as a user parameter).

Another limitation of our approach is that since normals are examined locally, two parts of the surface at varying depth but with similar normals will be given texture at the same scale, and may appear to be at the same depth after material replacement. This can be fixed by manually altering the scale of the texture appropriately, but a more automatic solution would be preferable.

8 Conclusion

We have presented a new system for interactive material replacement. Our novel normal recovery procedure reliably estimates the normals of diffuse objects even with albedo changes, broadly extending the utility of material replacement techniques. Integrating Lazy Snapping into the system vastly reduces the tedium of object selection. Jump map-based texture synthesis produces slightly lower-quality textured results, but produces results interactively, allowing for real-time texture brushing and very quick overall texturing. By achieving interactive speed for all elements of the system, we remove the perceived penalty for experimentation with different selections, textures, and rendering options.

Acknowledgements

We would like to thank the anonymous reviewers for all of their helpful comments. This research was supported in part under grants from the NSF (CCR-0086084 and CCF-0121288).

References

[1] M. Bichsel and A. P. Pentland. A simple algorithm for shape from shading. *Computer Vision and Pattern Recognition*, pages 459–463, 1992.

[2] Yuri Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *Proceedings of the Eighth IEEE International Conference on Computer Vision*, volume 1, pages 105–112, 2001.

[3] S. Y. Cho and T. W. S. Chow. A new colour 3d sfs methodology using neural based colour reflectance models and iterative recursive method. *Neural Computation*, 14(11):2751–2789, 2002.

[4] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, pages 341–346, Los Angeles, CA, August 2001. ACM SIGGRAPH.

[5] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece, September 1999. IEEE Computer Society.

[6] Hui Fang and John C. Hart. Textureshop: texture synthesis as a photograph editing tool. *ACM Transactions on Graphics*, 23(3):354–359, 2004.

[7] Michael Gleicher. Image snapping. In *Proceedings of SIGGRAPH 95*, pages 183–190, 1995.

[8] Berthold K.P. Horn. Height and gradient from shading. *International Journal of Computer Vision*, 5(1):37–75, 1990.

[9] Vivek Kwatra, Arno Schödl, Irfan Essa, Grek Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(2):277–286, July 2003.

[10] C.H. Lee and A. Rosenfeld. Improved methods of estimating shape from shading using the light source coordinate system. *Artificial Intelligence*, 26(2):125–143, 1985.

[11] Yin Li, Jian Sun, and Chi-Keung Tang abd Heung-Yeung Shum. Lazy snapping. *ACM Transactions on Graphics*, 23(3):303–308, 2004.

[12] Yanxi Liu, Wen-Chieh Lin, and James Hays. Near regular texture analysis and manipulation. *ACM Transactions on Graphics*, 23(3):368–376, 2004.

[13] Sebastian Magda and David Kriegman. Fast texture synthesis on arbitrary meshes. In *Proceedings of the Eurographics Symposium on Rendering 2003*, pages 82–89, Leuven, Belgium, June 2003. Eurographics Association.

[14] E. Mortensen and W. Barrett. Intelligent scissors for image composition. In *Proceedings of SIGGRAPH 95*, pages 191–198, 1995.

[15] E. Mortensen and W. Barrett. Tobogan-based intelligent scissors with a four parameter edge model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 452–458, 1999.

[16] Byong Mok Oh, Max Chen, Julie Dorsey, and Fredo Durand. Image-based modeling and photo editing. In *Proceedings of SIGGRAPH 2001*, pages 433–442, Los Angeles, CA, August 2001. ACM SIGGRAPH.

[17] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of SIGGRAPH 2000*, pages 465–470, New Orleans, LA, July 2000. ACM SIGGRAPH.

[18] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3):309–314, 2004.

[19] A. R. Smith and J. F. Blinn. Blue screen matting. In *Proceedings of SIGGRAPH 96*, pages 259–268, 1996.

[20] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. In *Proceedings of SIGGRAPH 2002*, pages 673–680, San Antonio, TX, July 2002. ACM SIGGRAPH.

[21] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. *ACM Transactions on Graphics*, 23(3):315–321, 2004.

[22] Greg Turk. Texture synthesis on surfaces. In *Proceedings of SIGGRAPH 2001*, pages 347–354, Los Angeles, CA, August 2001. ACM SIGGRAPH.

[23] Li-Yi Wei and Mark Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of SIGGRAPH 2001*, pages 355–360, Los Angeles, CA, August 2001. ACM SIGGRAPH.

[24] Steve Zelinka and Michael Garland. Interactive texture synthesis on surfaces using jump maps. In *Proceedings of the Eurographics Symposium on Rendering 2003*, pages 90–96, Leuven, Belgium, June 2003. Eurographics Association.

[25] Steve Zelinka and Michael Garland. Jump map-based interactive texture synthesis. *ACM Transactions on Graphics*, 23(4):930–962, 2004.

[26] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively variant texture on arbitrary surfaces. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(2):295–302, July 2003.

[27] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Shape from shading: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.

[28] Qinfen Zheng and Rama Chellappa. Estimation of illuminant direction, albedo and shape from shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:680–702, 1991.