PAPER *Special Section on Image Recognition and Understanding*

Optimizing a Triangular Mesh for Shape Reconstruction from Images

Atsutada NAKATSUJI[†], ***Student Member***, **Yasuyuki SUGAYA**[††a)], ***and*** **Kenichi KANATANI**[††], ***Members***

SUMMARY In reconstructing 3-D from images based on feature points, one usually defines a triangular mesh that has these feature points as vertices and displays the scene as a polyhedron. If the scene itself is a polyhedron, however, some of the displayed edges may be inconsistent with the true shape. This paper presents a new technique for automatically eliminating such inconsistencies by using a special template. We also present a technique for removing spurious occluding edges. All the procedures do not require any thresholds to be adjusted. Using real images, we demonstrate that our method has high capability to correct inconsistencies.

key words: *triangular mesh generation, Delaunay triangulation, polyhedral representation, mesh optimization, 3-D reconstruction*

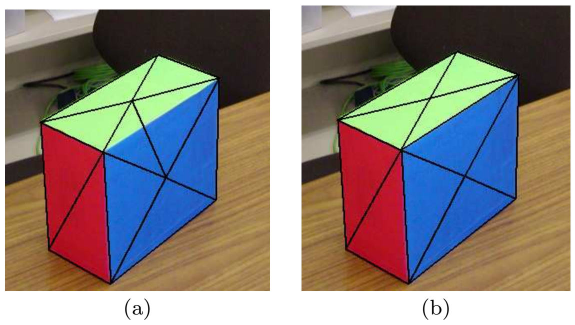
Fig. 1 (a) Triangulation inconsistent with the object shape. (b) Triangulation consistent with the object shape.

1. Introduction

One of the most important issues of 3-D reconstruction from images is how to represent the reconstructed shape. If we use stereo vision using calibrated cameras, we can obtain a dense depth map over all the pixels. By inter-pixel interpolation, we can display the scene as a curved surface. Alternatively, we can use a technique called *space carving* [5] and represent the scene as an aggregate of colored voxels. More sophisticated methods, called by such names as *plenoptic representation* [1], *light field rendering* [6], and *lumigraph* [2], are to register all the light rays in the scene for generating new views seen from an arbitrary viewpoint.

For images taken by uncalibrated cameras, on the other hand, we extract corresponding feature points from them and compute their 3-D coordinates, whether we deal with a continuous video stream using a method such as the *factorization* [10] or impose the *epipolar geometry* [3] on separate images. Then, we define a triangular mesh that has the feature points as vertices and display the scene as a texture-mapped polyhedron.

The triangular mesh is usually generated by *Delaunay triangulation* [9] of feature points in a specified frame. This produces triangles of balanced sizes and shapes, suitable for polyhedral representation of a curved surface. However, a serious problem occurs if the scene itself is a polyhedron. In man-made environments such as indoors and cities, most objects are polyhedra. If the vertices of polyhedral objects are chosen as feature points, some of the triangulation edges may not coincide with the physical edges. Then, the displayed 3-D shape may be inconsistent with the true polyhedral shape.

See Fig. 1, for example. The Delaunay triangulation in Fig. 1(a) does not correctly represent the object shape, but the triangulation in Fig. 1(b) correctly represents it. The aim of this paper is to present a technique for automatically transforming a given triangulation into a physically compatible one.

Sections 2 and 3 describe the principle of our method. The details of the procedure are described in Sect. 4 $\sim$ 7. In Sect. 8, we show real image examples to demonstrate that our method has high capability to correct inconsistencies. In Appendix, we present a technique for removing spurious occluding edges. The marked characteristic of our method is that no thresholds are required to be adjusted.

2. Compatibility of Triangulation

Many studies have been done in the past for generating optimal triangular meshes. They are roughly classified into two categories: one is to replace a dense mesh by a coarser one without impairing the faithfulness of the shape representation; the other is to upgrade the descriptiveness of the shape by adding vertices and edges. For the former, Vogiatzis et al. [11] introduced stochastic annealing coupled with Bayesian estimation on the assumption that the object mostly consists of planar faces. For the latter, Yu et al. [12] refined the mesh by iteratively estimating both the object shape and the surface reflectance map.

However, the only studies of optimizing edges for a given set of vertices are those of Morris and Kanade [7]

Manuscript received October 7, 2004.

[†]The author is with the Internet Terminal Division, NEC Engineering, Ltd., Yokohama-shi, 224-0053 Japan.

[††]The authors are with the Department of Computer Science, Okayama University, Okayama-shi, 700-8530 Japan.

a) E-mail: a-nakatsuji@pb.jp.nec.com

DOI: 10.1093/ietisy/e88-d.10.2269

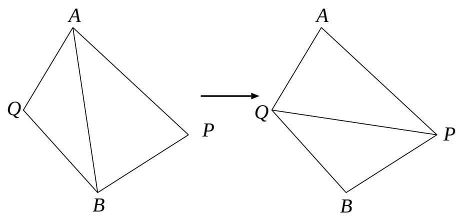

Fig. 2 Edge flipping.

and Perrier et al. [8]. The basic principle for their inconsistency detection is to compare the textures in corresponding triangular patches in different images. Suppose we have two images of a polyhedral object. If a triangular patch is defined on a planar surface, its texture in one image can be mapped onto the corresponding patch in the other by an affine transformation[†]. Hence, the intensity difference[††] after the mapping should be zero in that patch. If not, the patch is not on a planar surface, so we "flip"[†††] an appropriate edge into the diagonal position as illustrated in Fig. 2. Iterating this, we should end up with a triangular mesh compatible with the object shape [7].

In reality, however, the intensity difference is not exactly zero due to various disturbances such as inaccuracies of feature point matching, viewpoint dependent reflectance changes, and supposedly planar faces not being exactly planar. However, setting an appropriate threshold for judging planarity is very difficult. So, Morris and Kanade [7] and Perrier et al. [8] iteratively flipped edges so as to maximize the similarity (or minimize the dissimilarity) between the textures of corresponding patches.

As the texture (dis)similarity measure, Morris and Kanade [7] used the sum of square differences of the corresponding pixel values (to be minimized), while Perrier et al. [8] used the normalized correlation (to be maximized). In this paper, we show that the use of such a patch-based (dis)similarity measure is insufficient and present a more effective measure. Using real images, we demonstrate that our measure has higher capability to correct inconsistencies.

3. Principle of Inconsistency Detection

Given a triangular mesh over a polyhedral object scene, we hereafter say that an edge of the mesh is *correct* if it entirely lies on a planar face, and *incorrect* otherwise: an incorrect edge connects two points on different faces. We assume that the texture, color, or brightness of the object is different from face to face.

Figure 3 illustrates the principle of our inconsistency detection. Consider the 2-D quadrilateral $ABCD$ in the left of Fig. 3. Two possibilities exist for interpreting it as a 3-D polyhedron: one with faces $\triangle ABC$ and $\triangle CDA$; the other with faces $\triangle ABD$ and $\triangle BCD$. Suppose we view this object from a different angle. If the former interpretation is the case, the object should look as shown in the upper middle; if the latter is the case, it should look as shown in the lower middle. The right

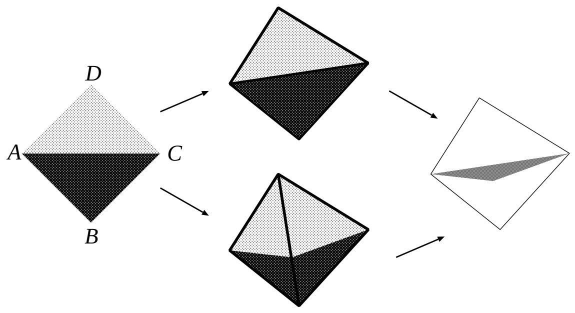

Fig. 3 Two 3-D interpretations of a quadrilateral $ABCD$ and the predicted intensity difference.

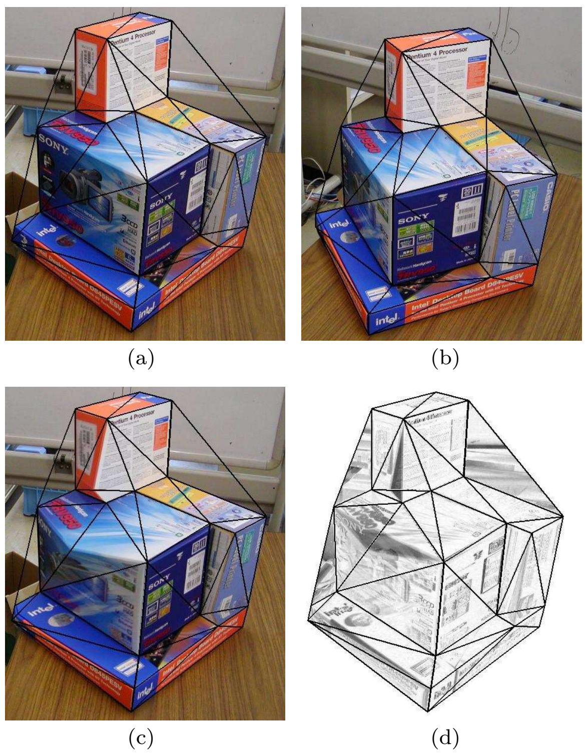

Fig. 4 (a),(b) Real images of a polyhedral object with a Delaunay triangulation (58 edges). (c) Texture mapping of (b) onto (a). (d) Intensity difference between (a) and (c). Darker tones correspond to larger values.

figure shows their intensity difference (darker tones correspond to larger values). Such intensity difference appears when the texture (including color and brightness) is different from face to face.

Figures 4(a),(b) are real images of a polyhedral

[†]Theoretically, corresponding patches in different views are related by a *homography* [3], but as far as individual patches are concerned, as opposed to a global planar scene, the mapping can be approximated by an affine transformation with negligible differences.

[††]In this paper, we consider color images and refer to the root-mean-square difference in the R, G, and B values simply as "intensity difference".

[†††]Morris and Kanade [7] used the term "swap", but since only one edge is involved, we use the more mathematically accepted term "flip" after Perrier et al. [8].

object, on which a Delaunay triangulation (based on (a)) is overlaid. Mapping the texture of Fig. 4(b) onto Fig. 4(a) patch by patch, we obtain Fig. 4(c). Figure 4(d) shows the intensity difference between Fig. 4(a) and Fig. 4(c). We observe narrow dark triangular regions that cross incorrect edges, because there are texture discontinuities across physical edges.

4. Inconsistency Detection Template

The above observation leads to the idea of detecting texture discontinuities by a template *specifically designed to detect them*. Figure 5(a) shows our template (lighter tones correspond to larger values). It is defined over a square region $ORST$ of size $l \times l$ with the following value:

$$T(x,y)=\begin{cases} e^{-\frac{(x+y-l)^2}{2\alpha^2(x-y-l)^2}} & x+y<l, x\geq y \\ T(y,x) & x+y\leq l, x<y \\ -T(l-y,l-x) & x+y>l \end{cases}. \quad (1)$$

The template value is symmetric with respect to the diagonal OS and anti-symmetric with respect to TR. The contour $T(x,y) =$ constant consists of two line segments starting from R and T and meeting on the diagonal OS. Figure 5(b) shows the cross section along the diagonal OS: the Gaussian function of mean $l/\sqrt{2}$ and standard deviation $\alpha l\sqrt{2}$ cut in the middle and placed upside down on the right side[†].

For a given edge, we map the intensity difference of the two triangles adjacent to it onto $\triangle OSR$ and $\triangle OST$ and compute the correlation (the sum of the product of corresponding pixel values) with this template. We set the template size l in such a way that the average area of the triangular patches in the input images is approximately $l^2/2$.

The reason why we use an anti-symmetric template is that we do not know a priori on which side the inconsistency region appears; it should lie only on one side of the diagonal of the surrounding quadrilateral (see Figs. 3 and 4(d)). Since the intensity difference is nearly zero on the other side, we can detect texture discontinuity by computing the absolute value of the correlation. It also has the advantage of canceling small fluctuations in the intensity difference caused by texture mapping inaccuracy, since such fluctuations are expected to spread randomly and evenly over the quadrilateral region.

In our experiment, we set the template value $T(x,y)$ to zero at the pixels on the diagonal TR and at the pixels within distance $0.02l$ pixels from the diagonal OS or from the boundary. This is for preventing texture mapping discrepancies caused by inaccuracies in locating feature points.

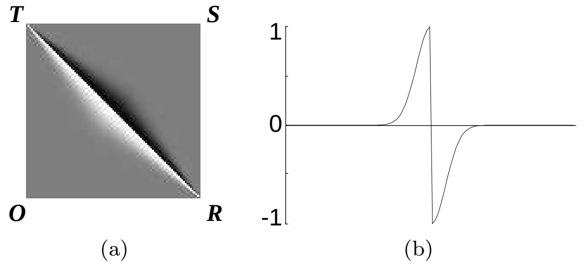

Fig. 5 (a) Inconsistency detection template. Lighter tones correspond to larger values. (b) Cross section along OS.

5. Evaluation of Edge Incorrectness

Given an initial triangulation over two corresponding images, we first measure the degree of incorrectness $w(AB)$ of each edge AB using the template $T(x,y)$ of Eq. (1). For this, we make the computation symmetric with respect to the two images: instead of mapping the texture from one image onto the other and computing the intensity difference there as described earlier, we directly map the texture onto the template region $ORST$ by a homography and compute the intensity difference there. The procedure is as follows:

1. If the edge AB has only one adjacent triangle, let $w(AB) = -1$, meaning that AB is a boundary edge.
2. Let $\triangle ABP$ and $\triangle ABQ$ be the adjacent triangles. Let $w(AB) = 0$ if the quadrilateral $APBQ$ is concave in either image.
3. Otherwise, map the texture in the quadrilateral $APBQ$ in the first image onto the template region $ORST$ by a *homography* and write down the intensity values there.
4. *Affinely* map the texture in $\triangle ABP$ and $\triangle ABQ$ in the second image onto $\triangle OSR$ and $\triangle OST$, respectively, and *subtract* the intensity values from the values written there.
5. Map the texture in the quadrilateral $APBQ$ in the second image onto the template region $ORST$ by a *homography* and *add* the intensity values to the values written there.
6. *Affinely* map the texture in $\triangle ABP$ and $\triangle ABQ$ in the first image onto $\triangle OSR$ and $\triangle OST$, respectively, and *subtract* the intensity values from the values written there.
7. Compute the correlation of the values written there with the template $T(x,y)$ of Eq. (1), and output its absolute value as $w(AB)$.

Here, we are assuming that at the time of generating the mesh each edge is classified either into a boundary edge with only one triangles on one side or into an internal edge with two triangles on both sides.

The concavity check in Step 2 is for preventing patch reversal as shown in Fig. 6. If $A : (a_1, a_2)$,

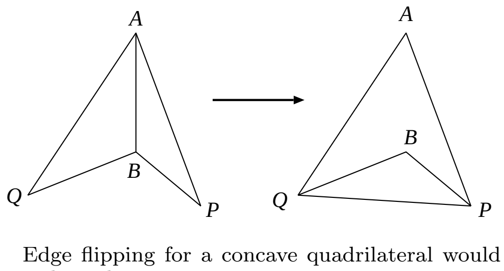

Fig. 6 Edge flipping for a concave quadrilateral would result in a reversed patch.

[†]We experimentally found that $\alpha = 0.1$ can produce a good result.

$B:(b_1,b_2)$, $P:(p_1,p_2)$, and $Q:(q_1,q_2)$ are the image coordinates of the four points A, B, P, and Q, respectively, the quadrilateral is concave if and only if

$$\begin{vmatrix} p_1-a_1 & q_1-a_1 \\ p_2-a_2 & q_2-a_2 \end{vmatrix} \cdot \begin{vmatrix} p_1-b_1 & q_1-b_1 \\ p_2-b_2 & q_2-b_2 \end{vmatrix} > 0. \quad (2)$$

6. Initial Mesh Generation

Given two images and corresponding feature points on them, we define a Delaunay triangulation using the feature points in the first image and isomorphically map it to the corresponding points in the the second image. Then, we compare the *signs* of corresponding triangular patches, where we define the sign of $\triangle ABC$ to be 1 if the order of A, B, and C is counterclockwise, -1 if clockwise, and 0 otherwise (i.e., degeneracy into a line segment). If their coordinates are $A:(a_1,a_2)$, $B:(b_1,b_2)$, and $C:(c_1,c_2)$, the sign of $\triangle ABC$ is given by

$$\mathrm{sgn}(\begin{vmatrix} b_1-a_1 & c_1-a_1 \\ b_2-a_2 & c_2-a_2 \end{vmatrix}), \quad (3)$$

where the symbol sgn() is the sign function, returning 1, -1, and 0 if the argument[†] is positive, negative, and zero, respectively.

If the signs are different between the two images, the triangle in the second image is reversed. We dissolve such reversals as follows. If one side of the reversed triangle is a boundary edge, we simply eliminate it (see Fig. 7). If the reversal occurs inside, we flip an appropriate side of the triangle, as discussed by Morris and Kanade [7] (see Fig. 8).

Once such patch reversals are resolved, no new reversals occur thereafter, since we check the concavity of the surrounding quadrilateral before flipping edges (Step 2 of the procedure in Sect. 5).

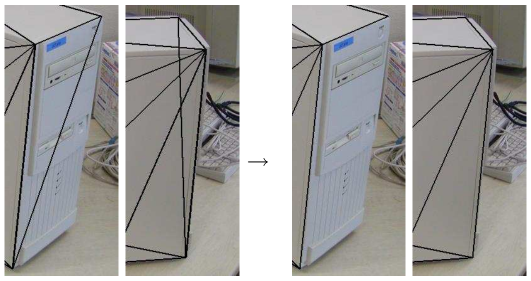

Fig. 7 If one side of the reversed triangle is a boundary edge, we eliminate it.

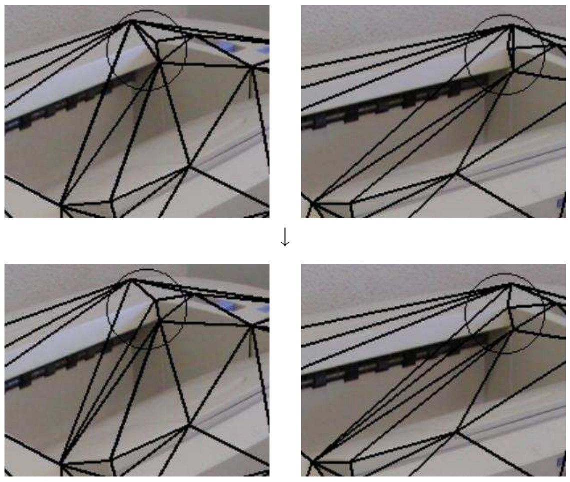

Fig. 8 If the reversal occurs inside, we flip an appropriate side of the triangle.

7. Procedure of Mesh Optimization

After resolving patch reversals, we do the following procedure:

1. Compute the incorrectness measure $w()$ for all the edges (Sect. 5).
2. Find the edge AB that has the largest value $w(AB)$.
3. Stop if $w(AB)=0$.
4. Flip the edge AB to PQ as shown in Fig. 2 and compute $w(PQ)$.
5. If $w(PQ)>w(AB)$, eliminate the edge PQ and restore the edge AB. Then, let $w(AB)=0$.
6. Otherwise, recompute $w()$ for edges PA, PB, QA, and QB, if $w()$ is not already 0, with respect to the new mesh configuration.
7. Go back to Step 2.

In this process, the value w itself has no absolute meaning; it is used only for comparison. Hence, no artificial thresholds need to be introduced. Since the largest value of $w()$ monotonically and strictly decreases at each flipping, and since edges once checked are not checked again, the above procedure terminates after all the edges are traversed once.

The above procedure can correct those incorrect edges that can be corrected by a single flipping operation. However, not all edges can be corrected that way, in particular when one physical edge is crossed by multiple mesh edges (Fig. 9; see also Fig. 12). So, we repeat the above procedure until the mesh configuration does not alter any further[††].

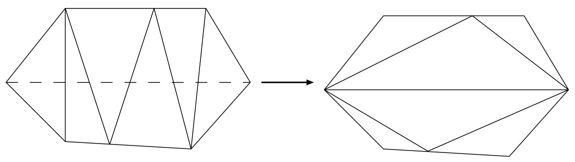

Fig. 9 If many mesh edges cross one physical edge (drawn in dashed lines here), multiple steps of flipping is necessary for resolving the inconsistency. See also Fig. 12.

[†]The argument is a signed area of the parallelogram defined by edges AB and AC.

[††]We record the history of the flipping and stop the computation if the same configuration appears twice, which occurs very rarely, though.

8. Experiments

Using real images, we compared the performance of our method with that for maximizing patch-similarity; we adopted the normalized correlation as Perrier et al. [8] for canceling view-dependent reflectance changes[†]. We randomly selected edges and flipped them if the dissimilarity decreases. We stopped this when no flipping occurred for consecutive E times, where E is the number of the mesh edges.

Applying our optimization to Figs. 4(a),(b), we obtained the mesh in Fig. 10(a). The iterations converged in two rounds of the procedure of Sect. 7. The correctness and the computation time are as written in the caption, where the correctness means (the number of correct edges)/(the number of non-boundary edges) in percentage. We used Pentium 4 3.2GHz for the CPU with 2GB main memory and Linux for the OS.

We compared this result with the patch-similarity maximization. Since the result changes at each trial because of the randomness of the search, we showed one typical result in Fig. 10(b). The correctness, the number of iterations, and the computation time written there are their averages over 10 trials.

Figures 11∼14 show other examples. Today, many algorithms are available for automatically extracting and matching feature points, e.g., [4], [13]. However, our concern here is not the accuracy of automatic matching but the performance of mesh optimization, so we selected matching points by hand. In all examples, figures (a),(b) are input images with the initial Delaunay triangulation overlaid; figures (c),(d) are the results corresponding to Figs. 10(a),(b), respectively.

.

9. Observations

We can see that our inconsistency detection template works very well. It is effective even when incorrect edges cannot be corrected by a single flipping operation (see Fig. 12).

We have also found that incorrect meshes obtained by patch-similarity maximization sometimes has higher similarity than the correct mesh obtained by our method, meaning that the patch-based correlation is not a good measure of shape consistency. In contrast, our method focuses specifically on regions where the texture discontinuity is most conspicuous.

For Fig. 13, however, patch-similarity maximization performed better than our method: one very short incorrect edge remained after our optimization. After careful investigations, we conclude that the triangles adjacent to it are too small to clearly detect the inconsistency region as compared with comparing the entire

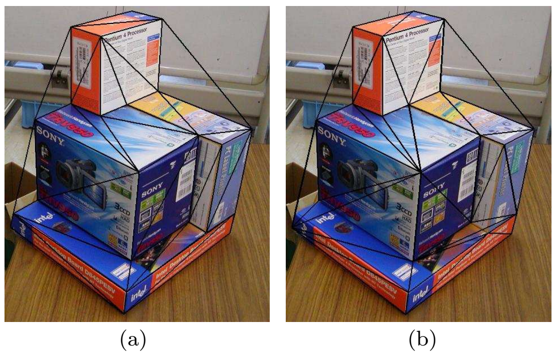
Fig. 10 Optimization of the mesh in Figs. 4(a),(b). (a) Our method (100% correct, 2 rounds, 3.43 sec). (b) Patch-similarity maximization (91.5% correct, 270 iterations, 9.05 sec).

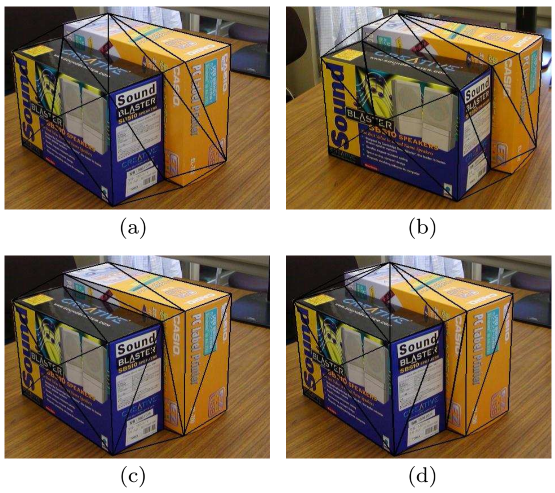
Fig. 11 (a),(b) Initial triangulation (31 edges). (c) Our method (100% correct, 3 rounds, 3.15 sec). (d) Patch-similarity maximization (75.2% correct, 159 iterations, 5.43 sec).

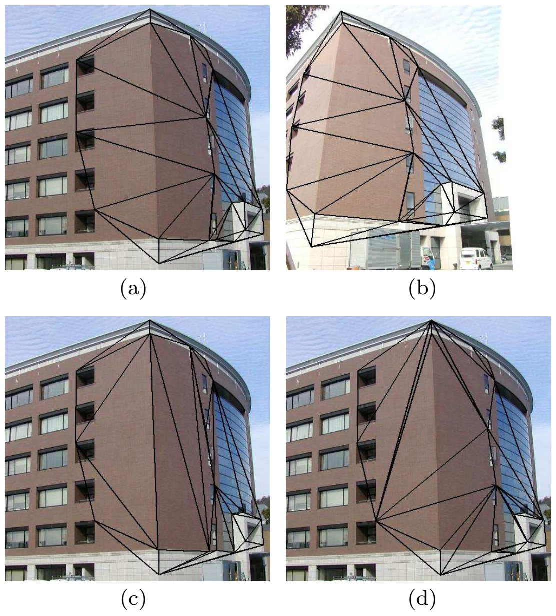
Fig. 12 (a),(b) Initial triangulation (47 edges). (c) Our method (100% correct, 3 rounds, 4.03 sec). (d) Patch-similarity maximization (96.2% correct, 313 iterations, 7.62 sec).

[†]In contrast, illumination changes are canceled when texture discontinuities are detected by our method.

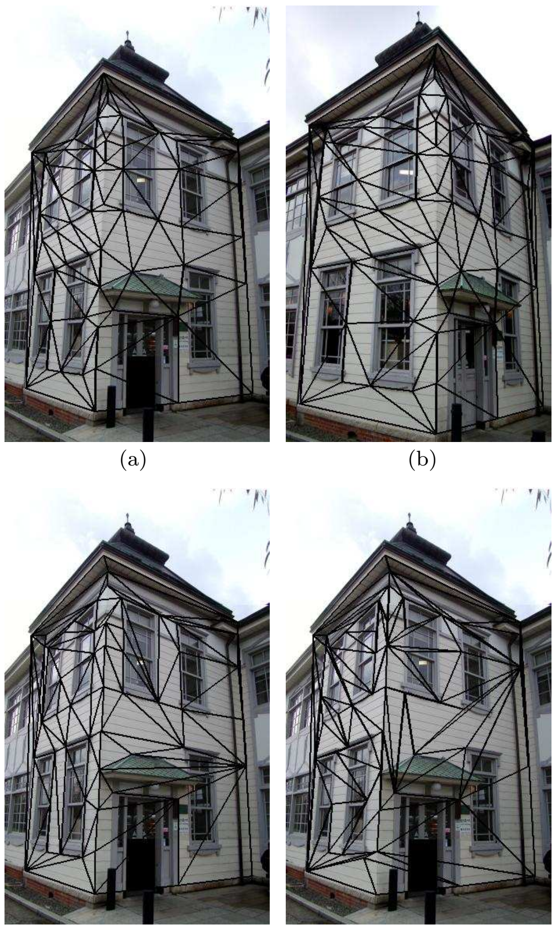

Fig. 13 (a),(b) Initial triangulation (157 edges). (c) Our method (98.7% correct, 7 rounds, 11.80 sec). (d) Patch-similarity maximization (99.3% correct, 2611 iterations, 19.60 sec).

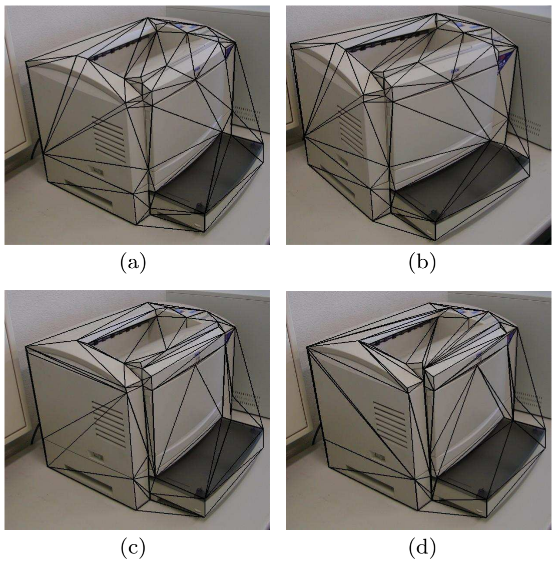

Fig. 14 (a),(b) Initial triangulation. (c) Our method (96.2% correct, 4 rounds, 11.96 sec). (d) Patch-similarity maximization (83.8%, 866 iterations, 19.70 sec).

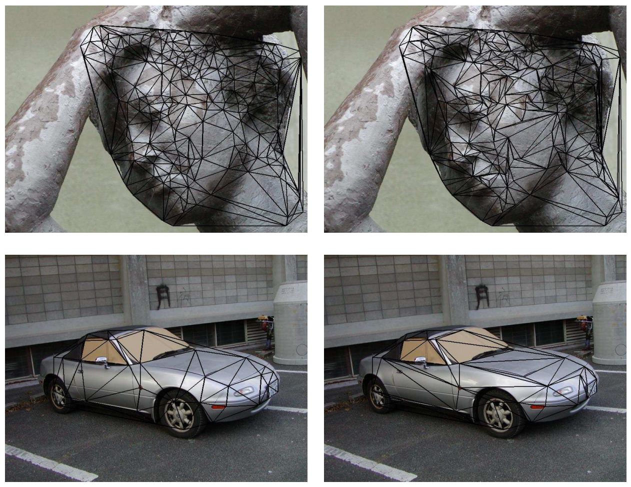

Fig. 15 Triangulation of curved surfaces. Left: Delaunay triangulation. Right: optimized triangulation.

patches.

We also tested our method using curved surfaces as well (Fig. 15). The feature points in the first example were generated by our automatic matching tool [4]. We see that our method yields better polyhedral approximations.

10. Concluding Remarks

We proposed a new technique for automatically transforming a triangular mesh so that it is compatible with the physical object shape. For this, we introduced a template that can sensitively detect texture discontinuities. Our procedure does not require any thresholds to be adjusted. Using real images, we demonstrated that our method has higher capability to correct inconsistencies than patch-similarity maximization.

In this paper, we used only pairs of images, but the principle can be straightforwardly extended to multiple images. Doing experiments (not shown here), however, we found that in most cases two separate views are sufficient.

We also studied another issue. The Delaunay triangulation defines a mesh over the convex hull of the given vertices. Hence, for objects having a concave boundary, some edges do not correspond to the object boundary, resulting in *spurious edges* (see Fig. 4). We can detect and remove them by checking image properties on both sides of each boundary edge. The details are described in Appendix.

One remaining issue is addition/deletion of vertices. Our method optimizes a given triangular mesh, but if some of the object corners are not chosen as mesh vertices, the resulting mesh does not represent a faithful 3-D shape however we optimize it. Also, too many feature points reduce efficiency. Perrier et al. [8] presented a scheme for removing points that produce extremely narrow triangles and adding new points inside large triangles. This type of process will be effective in practice.

Acknowledgments

This work was supported in part by the Ministry of Education, Culture, Sports, Science and Technology, Japan, under a Grant in Aid for Scientific Research C(2) (No. 17500112). The authors thank Daniel Morris of Northrop Grumman Corporation, U.S.A., for helpful comments and Masakazu Murata of Kumahira, Ltd. for helping the real image experiments.

References

[1] E.H. Adelson and J.R. Bergen, "The plenoptic function and the elements of early vision," in Computational Models of Visual Processing, ed. M. Landy and J.A. Movshon, pp.3–20, MIT Press, Cambridge, MA, U.S.A., Oct. 1991.

[2] S.J. Gortler, R. Gzreszczuk, R. Szeliski, and M.F. Cohen, "The lumigraph," Proc. SIGGRAPH, pp.43–54, New Orleans, LA, U.S.A., Aug. 1996.

[3] R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, Cambridge, U.K., 2000.

[4] Y. Kanazawa and K. Kanatani, "Robust image matching preserving global consistency," Proc. 6th Asian Conf. Comput. Vision, vol.2, pp.1128–1133, Jeju, Korea, Jan. 2004.

[5] K. Kutulakos and S. Seiz, "A theory of shape by space carving," Proc. Int. Conf. Comput. Vision, pp.307–314, Kerkyra, Greece, Sept. 1999.

[6] M. Levoy and P. Hanarahan, "Light field rendering," Proc. SIGGRAPH, pp.31–42, New Orleans, LA, U.S.A., Aug. 1996.

[7] D.D. Morris and T. Kanade, "Image-consistent surface triangulation," Proc. IEEE Conf. Comput. Vision Pattern Recog., vol.1, pp.332–338, Hilton Head, SC, U.S.A., June 2000.

[8] J. S. Perrier, G. Agin, and P. Cohen, "Image-based view synthesis for enhanced perception in teleoperation," in Enhanced and Synthetic Vision 2000, Proc. SPIE, ed. J. G. Verly, vol. 4023, June 2000.

[9] F. Preparata and M. Shamos, Computational Geometry, Springer, Berlin, Germany, 1985.

[10] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography—A factorization method," Int. J. Comput. Vis., vol.9, no.2, pp.137–154, Oct. 1992.

[11] G. Vogiatzis, P. Torr, and R. Cipolla, "Bayesian stochastic mesh optimization for 3D reconstruction," Proc. British Machine Vision Conf., vol.2, pp.711–718, Norwich, U.K., Sept. 2003.

[12] T. Yu, N. Xu, and N. Ahuja, "Shape and view independent reflectance map from multiple views," Proc. 8th Euro. Conf. Comput. Vision, vol.4, pp.602–615, Prague, Czech., May 2004,

[13] Z. Zhang, R. Deriche, O. Faugeras, and Q.-T. Luong, "A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry," Artif. Intell., vol.78, pp.87–119, 1995.

Appendix: Spurious Edge Removal

A naive idea for detecting spurious edges is to compare, over the two images, the texture inside the adjacent triangular patches. Since spurious edges define "transparent" patches, we should see through them different backgrounds over the two images. Hence, it appears that all we need to do is measure the intensity difference after texture mapping followed by thresholding. However, it is difficult to set an appropriate threshold a priori. Also, no intensity difference arises if the background is homogeneous, e.g. the sky or a white wall. So, we consider a method that does not require any thresholds.

Since occluding edges are on the object contours, they should lie on "intensity edges" if we generate an edge image (we use the term "intensity edges" to distinguish them from "mesh edges"). On the other hand, spurious edges may cross intensity edges of the background but rarely lie along them. So, we check if each boundary edge crosses intensity edges or lies along one[†].

Let $\boldsymbol{v}$ be the unit vector along the boundary edge under consideration. We measure the orientation of the intensity edge relative to $\boldsymbol{v}$ at each point by

$$E = 2(\boldsymbol{v}, \nabla I)^2 - \|\boldsymbol{v}\|^2\|\nabla I\|^2, \qquad (\mathrm{A}\cdot 1)$$

where $\nabla I = (I_x, I_y)^\top$ is the gradient of the image intensity I, and I_x and I_y are the derivatives of I with respect to x and y, respectively. We compute them using a 13×13 mask with Gaussian pre-smoothing of $\sigma = 3$ (pixels)[††]. The symbol $(\boldsymbol{v}, \nabla I)$ denotes the inner product of vectors $\boldsymbol{v}$ and ∇I, and $\|\nabla I\|$ denote the norm of ∇I.

The value E takes its maximum $\|\boldsymbol{v}\|^2\|\nabla I\|^2$ when ∇I is parallel to $\boldsymbol{v}$, its minimum $-\|\boldsymbol{v}\|^2\|\nabla I\|^2$ when when ∇I is orthogonal to $\boldsymbol{v}$, and 0 when $\boldsymbol{v}$ and ∇I makes 45° or 135°. The value of E does not depend on the sign of the vector $\boldsymbol{v}$.

We define the likeliness $\mathcal{E}$ of the edge in question by integrating[†††] Eq. (A· 1) along that edge and dividing it by its length. Then, we compare this $\mathcal{E}$ value with the $\mathcal{E}$ values of the adjacent inside edges. This does not require any thresholds to adjust.

At the time of generating the initial mesh, those edges adjacent to only one triangle on either side were labeled as *boundary edges*; this information was used in our mesh optimization procedure (Sect. 5). We traverse the list of boundary edges and do the following. Let AB be the edge we are currently visiting, and let $\triangle ABC$ be the triangle adjacent to it.

1. If AC or BC is a boundary edge (Fig. A· 1(a)), change the label of AB to *occluding edge* and go

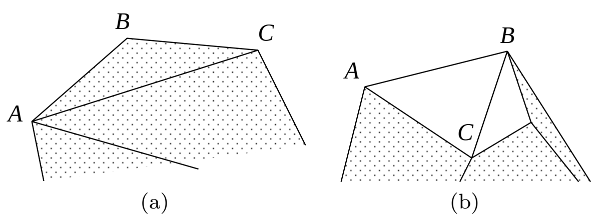

Fig. A· 1 (a) Edges AB and BC are judged to be occluding edges. (b) If edge AB is judged to be spurious, we check the edges AC and CB recursively.

[†]We tried to incorporate this information for internal edge correction, but this did not work: most detected intensity edges are due to the texture of planar surfaces.

[††]We apply this edge filter only to those pixels at which we need the value ∇I, rather than creating an edge image in advance.

[†††]We used the trapezoid rule with 100 subintervals for numerical integration.

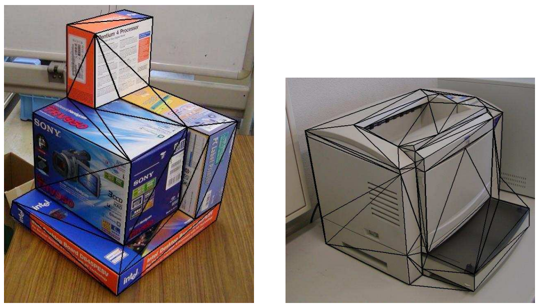
(a) (b)

Fig. A·2 Spurious edge removal. (a) The mesh resulting from Fig. 10(c). (b) The mesh resulting from Fig. 14(c).

to the next boundary edge.

2. Else, compute $\mathcal{E}(AB)$, $\mathcal{E}(AC)$, and $\mathcal{E}(BC)$.
3. If $\mathcal{E}(AB) \geq \mathcal{E}(AC)$ or $\mathcal{E}(AB) \geq \mathcal{E}(BC)$, change the label of AB to *occluding edge* and go to the next boundary edge.
4. Else, remove the edge AB, label the edges AC and BC as *boundary edges*, and recursively apply the same procedure to edges AC and BC.
5. Stop if no boundary edges remain to be visited.

The above procedure works only one image. If two images are available, we do Steps 2 and 3 for both images and change the label of AB to *occluding edge* if the condition is satisfied at least by one image. This prevents ambiguous edges from being removed. In fact, retaining some spurious edges does not do much harm, while the object shape cannot be described correctly if some occluding edges are inadvertently eliminated. If three more images are available, we can make the judgment more reliable by majority rule.

Figures A·2(a),(b) show the meshes obtained from those in Figs. 10(c) and Figs. 14(c). For Fig. A·2(a), in which the object color and reflectance are significantly different from the background, all spurious edges are correctly removed. For Fig. A·2(c), however, some spurious edges remain, because the object color and reflectance are very similar to the background, and removing them was judged to be dangerous.

Atsutada Nakatsuji received his B.E. and M.S. in mechanical engineering from the Osaka Institute of Technology, Japan, in 1999. Currently, he is at the Internet Termnal Division, NEC Engineering, Ltd. and is a Ph.D. candidate at the Department of Computer Science, Okayama University, Okayama, Japan. He is engaged in the research and development of 3-D sensing devices.

Yasuyuki Sugaya received his B.E., M.S., and Ph.D. in computer science from the University of Tsukuba, Ibaraki, Japan, in 1996, 1998, and 2001, respectively. He is currently Assistant Professor of computer science at Okayama University, Okayama, Japan. His research interests include image processing and computer vision. He received the IEICE best paper award in 2005.

Kenichi Kanatani received his B.E., M.S., and Ph.D. in applied mathematics from the University of Tokyo in 1972, 1974 and 1979, respectively. After serving as Professor of computer science at Gunma University, Gunma, Japan, he is currently Professor of computer science at Okayama University, Okayama, Japan. He is the author of many books on computer vision and received many awards including the best paper awards from IPSJ (1987) and IEICE (2005). He is an IEEE Fellow.