

Image-Based Photorealistic 3D Reconstruction Using Hexagonal Representation

HIDENORI SATO,[†] HIROTO MATSUOKA,[†] HITOSHI KITAZAWA^{††}
and AKIRA ONOZAWA[†]

An algorithm for reconstructing photorealistic 3D model from multiple-view images is proposed. The idea is based on the surface light field approach. In the algorithm, a geometric model is reconstructed as a visual hull using an image-based multi-pass algorithm we have developed, then the hull is represented as a quadrilateral-meshed surface. Next, colors of input images are assigned onto each vertex according to viewing directions using a new data structure we have developed. The structure is a hexagonal tessellation based on expansion and replacement of a buckyball. Finally, the hexagonal tessellation is represented as a hexagonal image whose pixels represent colors of corresponding input images. Experimental results for real objects show that 3D data can be successfully generated automatically in a short time and that photorealistic data can be viewed from arbitrary viewpoints even for objects with reflective or translucent surfaces.

1. Introduction

With the recent advances in computer vision (CV) and computer graphics (CG) research, many *image-based modeling* and *image-based rendering* techniques have succeeded in producing photorealistic synthetic views from real objects.

Image-based modeling^{1),2)} reconstructs a texture-mapped three-dimensional (3D) model once, and then renders it from arbitrary viewpoints. On the other hand, image-based rendering^{3),4)} directly generates new views viewed from various viewpoints. In addition, hybrid approaches that combine both a geometric model and image-based rendering have been proposed^{5)~9)}. Among them, the surface light field is a representative approach. Its data include both geometric and color data assigned to every small region or polygon on the surface. The assigned color data is a collection of rays leaving the region in every direction. For rendering, color from a viewing angle is selected and rendered. Therefore, the surface light fields can reproduce a photorealistic scene from arbitrary viewpoints with geometric data. Surface light field approaches have successfully visualized various types of real objects, including reflective or translucent ones^{7)~9)}.

In this paper, motivated by advances in surface light field approaches, we have developed

an algorithm to produce surface light field data from multiple-view images. The algorithm focuses on robustness and speed because they are important for practical use. To achieve our goal, we have developed 1) a fast multi-pass algorithm to generate a geometric model based on the shape-from-silhouette approach, and 2) a new data structure based on a hexagonal tessellation for capturing light field data. In our algorithm, light field data are represented as a hexagonal image and finally stored as a 2D normal image after a simple transformation so that conventional fast image compression algorithms can be applied. Experimental results show the algorithm can generate various types of data, including reflective or translucent objects, in a short time.

The remainder of this paper is organized as follows. Section 2 describes related work in more detail. Section 3 describes the overall algorithm and the image acquisition method. Section 4 describes the multi-pass geometry reconstruction algorithm we have developed. Section 5 discusses the hexagonal representation and describes the light field data assignment algorithm. Section 6 describes the interpolation algorithm. Section 7 presents some experimental results to show the effectiveness of the proposed algorithm, and Section 8 concludes the paper.

2. Related Work

Image-based modeling reconstructs 3D shapes from 2D multiple-view images. Such meth-

[†] NTT Microsystem Integration Laboratories

^{††} Tokyo University of Agriculture and Technology

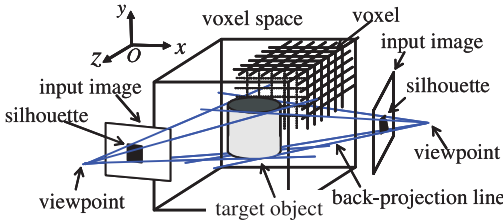


Fig. 1 Shape-from-silhouette approach.

ods are commonly known as *shape from X* methods, where X represents silhouette, shading, stereo, and so on. Among them, the silhouette-based methods are practically usable because of their robustness and low calculation cost^{1),2),10)~12)}. The approach first extracts an object region as a silhouette from each image, and then back-projects the silhouettes onto a voxel space (Fig. 1). The obtained intersection volume is the reconstructed shape, which is called a visual hull¹⁰⁾. Finally, texture is assigned to every mesh (polygon) on the surface. Texture assignment usually uses energy minimization schemes. For example, Matsumoto, et al. used an energy in terms of the size of polygons and viewpoints of images to assign texture onto adjacent polygons¹⁾. Although this approach can reconstruct a geometry quickly by using octree structure representation of a voxel space¹²⁾, it can not reconstruct concave surfaces. Therefore, it is actually impossible to assign a correct texture to a polygon on a concave surface. In addition, rendered views sometimes look odd even for convex surfaces because textures of adjacent polygons are assigned from different images.

On the other hand, image-based rendering is based on the idea that the appearance of an object from an arbitrary viewpoint can be completely presented by a collection of rays through the 3D space. A representative image-based rendering technique is the light field one proposed by Levoy, et al.³⁾, which assumes two planes pass through rays from a viewpoint. All rays are represented as a function of coordinates of the two planes. Gortler proposed a lumigraph⁴⁾ based on a similar idea. The quality of rendered views is good in practice. However, because the views don't use any geometric information, producing a photorealistic scene requires lots of images. In addition, it's difficult to handle light fields in a virtual space.

More recently, Matusik, et al. have developed a kind of hybrid approach⁶⁾. Like the light

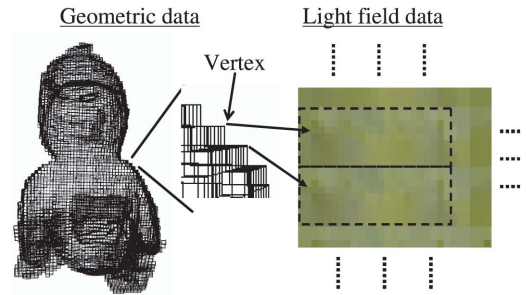


Fig. 2 Image of generated data.

fields, this method directly render views from multiple-view images. However, it directly estimates a viewed shape from a viewpoint based on the visual hull scheme and mixes it with a traditional image-based rendering algorithm to obtain more accurate results.

Surface light field approaches have also been proposed^{7)~9)}. The surface light fields can reproduce a photorealistic scene from arbitrary viewpoints using geometric data. Wood, et al.⁸⁾ construct geometric models using a laser digitizer, which makes it quite difficult to match the image data and geometric data. Effective representation of light field data is another issue in surface light field approaches. Capturing surface light field data involves associating the color of a radiant ray from a surface point with the surface position and direction of radiance. In practical cases, when multiple-view images and the geometric model are given, capturing surface light fields involves assigning the color (RGB value) of appearances (rays) of all images to every small region or polygon of the surface. The color is represented as a function of the viewing angles of images. Wood, et al. used an s-times-subdivided regular octahedron to assign rays⁸⁾. Compression of light field data is also an issue. Conventional methods use vector quantization or eigenspace, which can achieve high quality and compression ratio^{7),8)}. However, this requires many input images and much execution time for actual use.

3. Proposed Algorithm

3.1 Overview

The goal of our work is to develop an algorithm that can automatically generate a photorealistic 3D model from multiple-view images of real objects.

Figure 2 shows an example of data generated using our algorithm. The data comprise geometric data and image data. The geomet-

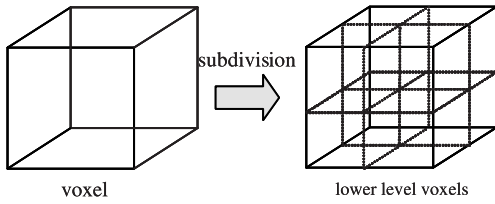


Fig. 3 Voxel subdivision.

ric data are ordinary polygonal models and the image data are collections of light field data of all vertices. In the image data in Fig. 2, a rectangular region represented by dotted lines indicates a light field data for a corresponding vertex. And pixel values in each rectangular region represent colors of different viewing angles. For rendering, colors of pixels corresponding to the nearest direction of the viewing angle are selected. The correspondence between a pixel and a viewing angle is determined according to the subdivision level.

The 3D reconstruction algorithm we developed is basically as follows.

- Step1.** Geometry reconstruction.
- Step2.** Light field data assignment.
- Step3.** Interpolation.

Step1 reconstructs a geometric model of an object based on the shape-from-silhouette approach. It is an octree-based hierarchical-subdivision similar to Szeliski's¹²⁾. The basic idea of subdivision is as follows. For each hierarchy, voxels are classified through an intersection test, which estimates whether they are projected on the border of the silhouette or not. Then, only voxels correspond to the border are further subdivided into eight equal-sized voxels (**Fig. 3**). The voxel subdivision is iterated until voxel size is as small as the specified size. In this paper, we call this voxel subdivision procedure from coarse to the finest resolution a *pass*. Our algorithm is an iteration of a pass using a simple and quick intersection test. Finally, a quadrilateral-polygonal surface of the object is obtained by connecting central points of neighboring voxels. The polygonal surface is a geometric data.

Step2 assigns colors of images onto all vertices of the reconstructed surface using the hexagonal image representation discussed in Section 5. Because the assignment uses the same-size hexagonal image and a common look-up table determined once, the calculation cost

is much reduced compared to calculating vertex by vertex. This also reduces rendering costs. Then, colors of all uncolored hexagons are interpolated in **Step3**. Next, all images are merged into one image and stored as normal image data. The image is a light field data.

3.2 Image Acquisition

To acquire multiple-view images, we used the *3D capturing system*¹³⁾. The system has a PC-controlled robot arm and a turntable, and it can capture images from arbitrary viewing points of almost all of the upper half of a target object placed on a turntable. Although we don't describe the system in detail, we mention that the system can capture multiple-view images that involve a 256-scaled multi-level silhouette so that it indicates an opaque region of an object.

To extract a silhouette, we set a threshold to the multi-level silhouette. Although this thresholding yields some segmentation errors in practice, the reconstructed geometries look correct as shown in Fig. 9. We think this is because the shape-from-silhouette approach regards an intersection volume of back-projection lines as a reconstructed shape; the intersection voxel was carved unless all images have common errors.

4. Geometry Reconstruction

This section describes the detail of the geometry reconstruction algorithm.

4.1 Intersection Test for Voxel Classification

First, we discuss our intersection test. Voxels in the same hierarchy are classified into three types based on the intersection test results:

- white:** a voxel whose vertices are projected on the silhouettes of all input images.
- black:** a voxel whose vertices are projected on the backgrounds of all input images.
- gray:** other voxels.

That is, a **gray** is a voxel whose vertices are projected on the borders of silhouettes (**Fig. 4**). In our algorithm, only **grays** are subdivided into lower levels.

The intersection test is simpler and faster than Szeliski's, which makes an approximate bounding box of intersection points. Ours uses only positions of intersection points. Moreover, just when two kinds of vertices, those lying on a silhouette and on a background, are found, the calculation of intersection points is stopped

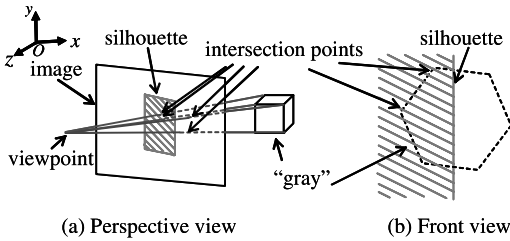


Fig. 4 Image of intersection test detected as **gray**.

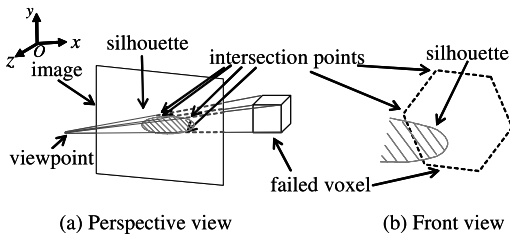


Fig. 5 Example of failed voxel.

and the voxel is classified as a **gray**.

4.2 Multi-Pass Subdivision Algorithm

The above intersection test fails in some actual cases because of its simplicity. For example, if a voxel is projected on a narrow tip of a silhouette so that avoiding all intersection points, it would be classified as a **black** even though it is actually a **gray** (Fig. 5). To overcome this problem, we have developed the multi-pass subdivision algorithm. The algorithm iterates the following procedure as long as failed voxels are found.

- Step1.1** Subdivide an initial voxel space or failed voxels into a lower level by force.
- Step1.2** Apply above voxels for a pass algorithm.
- Step1.3** Detect failed voxels.

The algorithm is based on the idea that the generated surface must be closed. In **Step1.1**, initial voxels are subdivided into eight voxels in the first pass. After subsequent passes, remaining failed voxels are subdivided. **Step1.2** divide above voxels recursively with the single-pass procedure using the intersection test.

In **Step1.3**, **grays** of the finest resolution are found based on the octree-structure based search¹⁴⁾ and adjacent **grays** are connected. Here, detected **grays** shape the surface of a reconstructed geometry. In addition, if there is a hole on the reconstructed surface, the coarse voxel on the corresponding coordinates will be searched. This is a failed voxel.

Finally, a closed surface constructed from only **grays** can be obtained, although it requires multiple passes and the octree structure must be kept to find failed voxels. In addition, it is not guaranteed that our algorithm is theoretically faster than Szeliski's. However, experimental results show that more than around 90% of voxels are generated in the first pass and the first pass accounts for most of the total execution time. Therefore, we believe our algorithm is effective for almost all real data.

5. Hexagonal Image Representation

In this section, we first show that a hexagonal tessellation is effective to capture the color of rays from multiple-view images by applying a simple transformation. Then, we describe **Step2** in Section 3.

5.1 Generation of Hexagonal Tessellation from Buckyball

To capture data from multiple-view images, we use a buckyball. The buckyball is a truncated icosahedron. It consists of 12 regular pentagons and 20 regular hexagons so that each pentagon adjoins 5 hexagons and each hexagon adjoins three hexagons and three pentagons [Fig. 6 (a)]. It is inscribed in a sphere, and it can also represent every direction as surface positions because omni-directional rays incident to its center can be mapped onto the surface positions one to one. In addition, the buckyball can be expanded on a 2D plane without overlaps [Fig. 6 (b)]. A hexagonal tessellation is obtained by replacing all pentagons with hexagons. Thus, every direction can be represented as 2D positions on the hexagonal tessellation shown in Fig. 6 (c) without loss of connectivity. The hexagonal tessellation can be regarded as a hexagonal image having 32 hexagonal pixels if one color is assigned to each hexagon. This means if one of the rays intersecting each polygon on a buckyball surface is selected and if its color is assigned to the polygon, the ray can be represented as a color of a hexagonal pixel.

On the other hand, it is impossible to represent a sphere approximately as a normal square organized image (square image) without overlaps of adjacent pixels. Therefore, assigning each color of a ray to only one squared pixel is difficult.

In addition, it is well known that a hexagonal tessellation has higher symmetry and connectivity than a square tessellation. Actually,

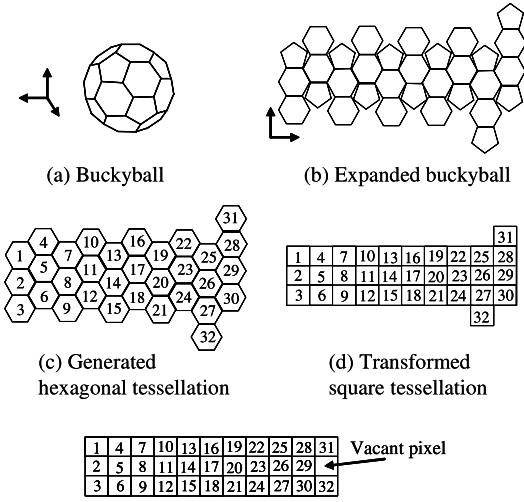


Fig. 6 Buckyball expansion.

a hexagonally organized image (hexagonal image) has been compared to the square image in image processing research¹⁵⁾. Snyder proposed a coordinate transformation for representing a general pattern of a hexagonal tessellation in a 2D Cartesian coordinate system¹⁶⁾.

To store the regular pattern of the hexagonal tessellation as a hexagonal image, we have developed a method for transformation from (c) to (d) in Fig. 6. In the figure, hexagons in (c) and squares in (d) with the same number correspond to each other. The method is based on transformation of each column of the hexagonal tessellation so that adjacent columns will be side by side in the square tessellation.

Note that, like the coordinate system transformation in Ref. 16), the transformation is performed without loss of connectivity. When storing an image, we moved hexagons 31 and 32 to the upper right and lower right pixels respectively to prevent the image from generating too many vacant pixels. At the same time, one vacant pixel is generated between them [Fig. 6 (e)]. Finally, we can store the omnidirectional light field data as a 2D image by applying conventional fast compression algorithm.

5.2 Light Field Data Assignment

Step2 uses the hexagonal representation discussed in the previous section and it consists of two consecutive sub-steps, **Step2.1** and **Step2.2**.

In **Step2.1**, a buckyball is placed at the center of the reconstructed model first. Then, the data structure of the hexagonal image is de-

termined as described later, and a look-up table is generated simultaneously using the algorithm described in the next section. The look-up table shows the correspondence between input images and hexagonal pixels and is used as a common table in **Step2.2**.

Step2.2 allocates hexagonal images for all vertices first. Then, using the look-up table, the RGB value of the intersection point of each image and its line of sight is assigned to a corresponding hexagonal pixel as light field data. For each vertex, only the color of input images that are visible from the vertex are mapped on pixels using the z-buffer algorithm.

5.3 Extension to Multiple Views

Although producing a photorealistic view requires lots of images, the hexagonal image generated from a buckyball can capture rays from at most 32 images. Regarding the hexagonal tessellation as level 1, we have developed a hierarchical subdivision algorithm that can be applied for making the look-up table in **Step2.1**. The algorithm is as follows.

- Step2.1.0.** Position a buckyball inscribed in a unit sphere.
- Step2.1.1.** Calculate all intersection points of the buckyball surface and lines of sight of input images in a 3D coordinate system.
- Step2.1.2.** Expand the buckyball with all intersection points.
- Step2.1.3.** Convert all coordinates to a 2D coordinate system.
- Step2.1.4.** Iterate the following sub-steps for subdivision until the specified level is achieved (**Fig. 7**).
 - Step2.1.4.1.** Subdivide every hexagon so that a quarter-size regular hexagon will be surrounded by six equal-size trapezoids.
 - Step2.1.4.2.** Merge two adjacent trapezoids whose bases are common so that they will be a quarter-size regular hexagon.
- Step2.1.5.** For every hexagon, choose the intersection point nearest the center of the hexagon if it contains more than two intersection points.

The subdivision doesn't rotate any hexagons around the symmetry axis level by level, and also doesn't create any spaces or overlaps between hexagons on the same level. Thus, the hexagonal tessellation of each level contains all

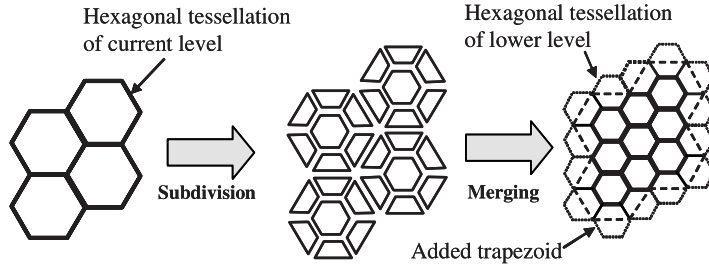


Fig. 7 Subdivision of hexagonal tessellation.

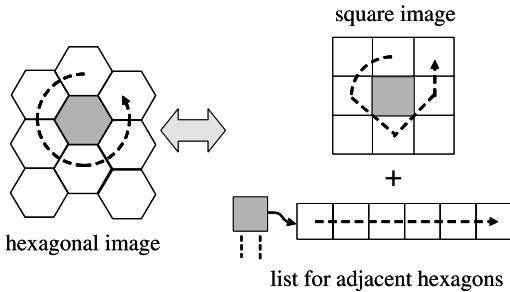


Fig. 8 Data structure for hexagonal image.

intersection points in it. In addition, we have to consider hexagons on borders for subdivision. We solved this problem by adjoining virtual trapezoids adaptively as shown in Fig. 7.

On the other hand, we represented a hexagonal image using two types of data structures. One is a square image whose pixels represent the color of hexagonal pixels. The other is a list that represents six adjacent hexagonal pixels for every hexagonal pixel. An image of our data structure is shown in Fig. 8. In the figure, a gray pixel indicates a target pixel and pixels through which the broken line passes indicate pixels adjacent to the target pixel. The list is used to find adjacent pixels for interpolation.

In the actual implementation, we prepared the square image shown Fig. 6 (e) and a corresponding list for adjacency for level 1 as initial data. Then, the data for the specified level were generated by subdividing the structure by maintaining the adjacency level by level. Here, we set the adjacent pixels for a vacant pixel as the adjacent pixels in a corresponding square image. The structure requires more memory and calculation in order to generate and maintain both a look-up table and hexagonal images. However, calculation cost for adjacent pixel search is almost the same as that for a square image.

Finally, every resulting hexagon has at most one intersection point because of calculating in

the 2D coordinate system. And we can also generate a look-up table that associates hexagons with input images one by one because one intersection point corresponds to one input image. As for storing, the size of an image will be 7×24 , including nine vacant pixels, for level 2, while it will be 3×11 , including one vacant pixel, for level 1.

6. Interpolation

In Step2, some pixels might be uncolored because of a lack of images and visibility. In Step3, colors of uncolored pixels are determined by a linear interpolation approach. The algorithm for one hexagonal image is as follows.

- Step3.1 Produce a queue of uncolored pixels according to the number of adjacent colored pixels.
- Step3.2 Calculate the color of the pixel at the top of the queue as a mean of the colors of adjoining colored pixels.
- Step3.3 Mark the pixel as a colored pixel.
- Step3.4 If uncolored pixels remain, go to Step3.1.

In Step3, we take the connectivity of pixels on borders into account by virtually adjoining pixels. For example, at level 1 hexagon 22 and hexagon 28 in Fig. 6 (c) are treated as adjoining in Step3.1 and Step3.2.

7. Results

We implemented the proposed algorithm in VC++ on a PC/AT computer (Windows XP Xeon, 1.7 GHz with 4GBytes of memory).

Before capturing images, we calibrated the 3D capture system from the same positions as those we took using a calibration pattern. Then, we obtained 83 images of each of ten real objects, which included objects with reflective and translucent surfaces under normal fluorescent light. For each set of data, viewing angles

of the images were assigned uniformly to almost the entire upper half of the object. We generated reconstructed data as geometric data of 1-*mm* resolution so that the geometric data would be almost the same as that of input images and light field data with rays of 159 directions for level 2. Finally, we successfully obtained 3D data without manual adjustment or improvement of input data.

Table 1 shows results of geometry reconstruction for a “shoe”, a “PET bottle” and a “teapot”. In the table, “total” in execution time means the time to execute all passes, and “failed” means the time to execute the second and later passes. The results show that most of the shape was determined after the first pass.

Table 2 also shows the total execution times, including geometry reconstruction, and the properties of the reconstructed data. As for the light field data, we show the amount of data in PNG-compressed and JPG-compressed formats. The maximum memory consumption during execution was about 500 MBytes for both sets of data. As shown in Table 2, the data were generated around 4 minutes or less.

The rendered views of generated data are shown in **Fig. 9** and **Fig. 10**. In Fig. 9, input images, rendered results of reconstructed wire frames, and rendered results for PNG-compressed and JPG-compressed light field data from the same viewing points are shown for each data. The rendered results for an interpolated-angle view are shown in Fig. 10. Here, note that PNG compression is lossless. For both sets of data, photorealistic rendered views were obtained, although there is some blurring of images obtained for the interpolated-angle view and the JPG compressed data. Especially, noteworthy is that the

lining of the “shoe” can be viewed, although the lining is on a concave surface. In addition, the reflective or transparent part of the objects are also well reconstructed.

8. Discussions and Conclusions

We have developed an algorithm for generating a surface-light-field data from multiple-view images. In the algorithm, geometric data are reconstructed as a visual hull using the fast multi-pass algorithm we have developed. In addition, we also have developed a new data structure based on a hexagonal tessellation generated from a buckyball for capturing light field data. The experimental results show the algorithm yields the surface light field data in a short time for various types of real objects and that photorealistic data can be viewed from arbitrary viewpoints.

We believe our algorithm is very practical for its speed, quality, and wide applicability for various types of objects. However, there are some remaining problems. One is the frame rate for rendering. The actual frame rate was about 8 ~ 12 fps with the NVIDIA GeForce4 video-card, because of the large scale geometric models and rendering method. In our current rendering program, we use vertex coloring instead of texture mapping because our algorithm assigns colors vertex by vertex. We think rendering speed will be improved by combining our data with conventional polygonal simplification algorithm so that the simplified polygon has multiple-view textures that are collection of colors of original vertices like in Wood’s rendering algorithm⁸⁾.

Another problem is the interpolation. As shown in Fig. 10, the interpolated views we obtained were blurred. To improve the quality, we have to make input images denser or use more correct and stricter calibration and segmentation results, but this would reduce the robustness of the algorithm. So we plan to develop different algorithms to improve the quality of interpolation views. Another problem concerns the lighting condition and an effective compress-

Table 1 Results of geometry reconstruction.

Data	execution time (s)		# of generated voxels	
	total	failed	total	failed
“shoe”	17.4	0.3	100,418	1,288
“PET bottle”	22.1	2.1	116,553	13,207
“teapot”	16.6	2.8	62,745	4,193

Table 2 Final results.

Data	Execution time	geometric data		light fields data	
		total amount (Mbyte)	# of vertices	PNG (Mbyte)	JPG (Mbyte)
“shoe”	4 m 00 s	1.1	98,102	44.3	8.5
“PET bottle”	4 m 01 s	0.9	77,443	34.0	5.6
“teapot”	2 m 35 s	0.7	60,259	25.2	4.5






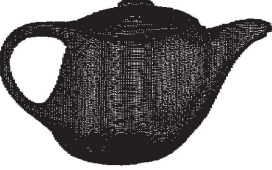






		“shoe”	“PET bottle”	“teapot”
Input Images				
	wire frame			
Rendered Views	PNG Compressed			
	JPG Compressed			

Fig. 9 Rendered views for “shoe”, “PET bottle” and “teapot”.

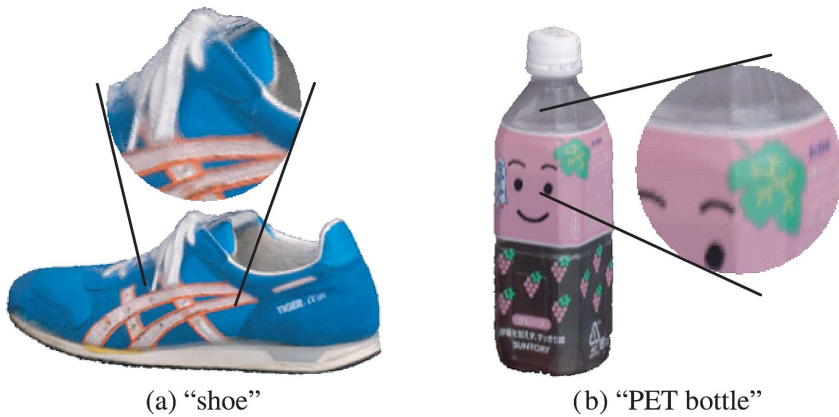


Fig. 10 Interpolated views for “shoe” and “PET bottle”.

sion algorithm. Our current algorithm doesn't take lighting conditions into account. Basically, it regenerates views of input images with reconstructed geometry. The simplest and most intuitive way to consider the lighting conditions is to obtain input images under various lighting conditions and store all of them, extending the idea of the hexagonal representation. However, it would be better to develop compression algorithm that makes more effective use of the hexagonal image representation.

Acknowledgments The authors would like to acknowledge to members of the Home Communication Research Group at NTT for helpful discussions.

References

- 1) Matsumoto, Y., Terasaki, H., Sugimoto, K. and Arakawa, T.: A Portable Three-dimensional Digitizer, *Proc. Recent Advances in 3-D Digital Imaging and Modeling*, pp.197–204 (1997).
- 2) Niem, W. and Wingbermühle, J.: Automatic Reconstruction of 3D Objects Using a Mobile Monoscopic Camera, *Proc. Recent Advances in 3-D Digital Imaging and Modeling*, pp.173–180 (1997).
- 3) Levoy, M. and Hanrahan, P.: Light Field Rendering, *SIGGRAPH '96 Proceedings*, pp.31–42 (1996).
- 4) Gortler, S., Grzeszczuk, R. and Szeliski, R.: The Lumigraph, *SIGGRAPH '96 Proceedings*, pp.43–54 (1996).
- 5) Debevec, P.E., T. C.J. and Malik, K.: Modelling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach, *SIGGRAPH '96 Proceedings*, pp.11–20 (1996).
- 6) Matusik, W., Buehler, C., Rasker, R., Gortler, S. and McMillan, L.: Image-based Visual Hulls, *SIGGRAPH '00 Proceedings*, pp.369–374 (2000).
- 7) Nishino, K., Sato, Y. and Ikeuchi, K.: Eigen-Texture Method: Appearance compression and synthesis based on 3D model, *Proc. CVPR '99*, Vol.1, pp.618–624 (1999).
- 8) Wood, D.N., Azuma, D.I. and Aldinger, K.: Surface Light Fields for 3D Photography, *SIGGRAPH '00 Proceedings*, pp.287–296 (2000).
- 9) Chen, W.-C., Bouguet, J.-Y., Chu, M.H. and Grzeszczuk, R.: Light field mapping: efficient representation and hardware rendering of surface light fields, *SIGGRAPH '02 Proceedings*, pp.447–456 (2002).
- 10) Laurentini, A.: The Visual Hull Concept for Silhouette-Based Image Understanding, *IEEE PAMI*, Vol.16, No.2, pp.150–162 (1994).
- 11) Wingbermühle, G.E.J. and Niem, W.: Shape Refinement for Reconstructing 3D-Objects Using Analysis-Synthesis Approach, *Proc. ICIP 2001*, Vol.3, pp.903–906 (2001).
- 12) Szeliski, R.: Rapid Octree Reconstruction From Image Sequences, *CVGIP: Image Understanding*, Vol.58, No.1, pp.23–32 (1993).
- 13) Matsuoka, H., Onozawa, A., Sato, H. and Nojima, H.: Regeneration of Real Objects in the Real World, *SIGGRAPH '02*, Sketch and Applications (2002).
- 14) Chen, H.H. and Huang, T.S.: A Survey of Construction and Manipulation of Octrees, *Computer Vision, Graphics, and Image Processing*, Vol.43, pp.409–431 (1988).
- 15) Kamgar-Parsi, B., Kamgar-Parsi, B. and Sander, III, W.A.: Quantization Error in Spatial Sampling: Comparison between Square and Hexagonal Pixels, *Proc. CVPR '89*, pp.604–611 (1989).
- 16) Snyder, W.E., Qi, H. and Sander, W.: A coordinate system for hexagonal pixels, *Proc. SPIE Medical Imaging 1999*, Vol.3661, SPIE, pp.716–727 (1999).

(Received April 19, 2004)

(Accepted November 1, 2004)

(Online version of this article can be found in the IPSJ Digital Courier, Vol.1, pp.36–45.)



Hidenori Sato Senior Research Engineer, Ubiquitous Interface Laboratory, NTT Microsystem Integration Laboratories, NTT Corporation. He received the B.S. and M.S. degrees in physics from Tohoku University, Sendai, Japan, in 1987 and 1989, respectively. In 1989, he joined NTT LSI Laboratories, Kanagawa, Japan, where he worked on LSI CAD. His current research interests are in computer vision, computer graphics, and human computer interaction system. Mr. Sato is a member of IEICE, IPSJ, ITE, and IEEE.



Hiroto Matsuoka Research Engineer, Ubiquitous Interface Laboratory, NTT Microsystem Integration Laboratories, NTT Corporation. He received the B.S. degree in electrical and electronic engineering from

Ritsumeikan University, Kyoto, Japan in 1989. From 1989 to 1992, he had worked at NEC Kansai. He received the M.S. degree from Graduate School of Information Science of Nara Institute of Science and Technology, Nara, Japan, in 1995. In 1995, he joined NTT, where he researched circuit designs for microwave equipment. His current research interest is three-dimensional computational graphics, especially for modeling and rendering algorithms. Mr. Matsuoka is a member of IPSJ and IEICE.



Hitoshi Kitazawa received his B.S., M.S. and Ph.D. degrees in electronic engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1974, 1976 and 1979, respectively. He joined the Electrical Communication Laboratories, Nippon Telegraph and Telephone Corporation (NTT) in 1979. Since 2002, he is a professor at Tokyo University of Agriculture and Technology. His research interests are in VLSI CAD algorithm, Computer Graphics and Image Processing. Dr. Kitazawa is a member of IPSJ, IEICE and IEEE.



Akira Onozawa Senior Research Engineer, Supervisor, Ubiquitous Interface Laboratory, NTT Microsystem Integration Laboratories, NTT Corporation. He received his B.E. in 1983 and M.E. in 1985 both in Electronic Communication Engineering and Ph.D. in 2002 in Information and Computer Science all from Waseda University, Tokyo, Japan. He joined NTT in 1985 where since then he had been working for the research and development of LSI CAD algorithms and systems. His current interests include HCI, CV and CG. Dr. Onozawa is a member of ACM, IEEE, IEICE and IPSJ.