Real-time dynamic 3-D object shape reconstruction, and high-fidelity texture mapping for 3-D video

Matsuyama, T; Wu, XJ; Takai, T; Wada, T

Real-Time Dynamic 3-D Object Shape Reconstruction and High-Fidelity Texture Mapping for 3-D Video

Takashi Matsuyama, Xiaojun Wu, Takeshi Takai, and Toshikazu Wada, *Member, IEEE*

***Abstract*—Three-dimensional (3-D) video is a real 3-D movie recording the object's full 3-D shape, motion, and precise surface texture. This paper first proposes a parallel pipeline processing method for reconstructing a dynamic 3-D object shape from multiview video images, by which a temporal series of full 3-D voxel representations of the object behavior can be obtained in real time. To realize the real-time processing, we first introduce a plane-based volume intersection algorithm: first represent an observable 3-D space by a group of parallel plane slices, then back-project observed multiview object silhouettes onto each slice, and finally apply two-dimensional silhouette intersection on each slice. Then, we propose a method to parallelize this algorithm using a PC cluster, where we employ five-stage pipeline processing in each PC as well as slice-by-slice parallel silhouette intersection. Several results of the quantitative performance evaluation are given to demonstrate the effectiveness of the proposed methods. In the latter half of the paper, we present an algorithm of generating video texture on the reconstructed dynamic 3-D object surface. We first describe a naive view-independent rendering method and show its problems. Then, we improve the method by introducing image-based rendering techniques. Experimental results demonstrate the effectiveness of the improved method in generating high fidelity object images from arbitrary viewpoints.**

***Index Terms*—Parallel volume intersection, PC cluster, real-time three-dimensional (3-D) volume reconstruction, three-dimensional (3-D) video, video texture mapping.**

I. Introduction

THREE-DIMENSIONAL (3-D) video [3] is a real movie recording dynamic visual events in the real world as they are: time-varying 3-D object shapes with a high-fidelity surface texture. Its applications cover wide varieties of personal and social human activities: entertainment (e.g., 3-D games and 3-D TV), education (e.g., 3-D animal picture books), sports (e.g., sport performance analysis), medicine (e.g., 3-D surgery monitoring), and culture (e.g., 3-D archive of traditional dance).

In recent years, several research groups developed real-time full 3-D shape[1] reconstruction systems for 3-D video [3]–[7]. All of these systems focus on capturing human body actions and share a group of distributed video cameras for real-time synchronized multiviewpoint action observation. While the real-time quality of the earlier systems [3], [5] was confined to the synchronized multiviewpoint video observation alone, the parallel volume intersection on a PC cluster has enabled the real-time full 3-D shape reconstruction [4], [6], [7].

Note that, even if its accuracy is limited, the real-time dynamic full 3-D object shape reconstruction has many applications such as human behavior analysis in sports (e.g., golf swing) and medical rehabilitations, on-site clothes fitting, motion capture for making animated films, and perceptual user interface systems [15] as well as generation of 3-D video.

To cultivate the 3-D video world and make it usable in everyday life, we have to solve the following technical problems:

- *Computation speed:* we have to develop both faster machines and algorithms, because near-frame-rate 3-D shape reconstruction has been attained only in coarse resolution.
- *High fidelity:* to obtain high-fidelity 3-D video of the same quality as ordinary video images, we have to develop high-fidelity texture-mapping methods as well as increase the resolution.
- *Wide-area observation:* To capture high-resolution 3-D video, the systems developed so far restricted their 3-D observable spaces to rather small areas, especially those for teleconference systems [12], [14], in which the space is limited to the upper half of a human body. To extend the 3-D observable space while maintaining the resolution, for example, to capture dancing people, we have to introduce an active object tracking capability [16] and/or increase the number of cameras drastically.
- *Data compression:* since naive representation of 3-D video results in huge amounts of data, effective compression methods are required to store and transmit 3-D video data [17].
- *Editing and visualization:* since editing and visualization of 3-D video are conducted in the four-dimensional (4-D) space (3-D geometric + one-dimensional (1-D) temporal), we have to develop human-friendly 3-D video editors and visualizers that help a user to understand dynamic events in the 4-D space.

This paper first describes an overall process of generating 3-D video and then proposes a plane-based volume intersection method followed by its parallel pipeline implementation using a PC cluster.[2] With this method, a temporal series of full 3-D voxel representations of the object behavior can be obtained in real time. Several results of its quantitative performance evaluation are given to demonstrate its effectiveness. In the latter

Manuscript received December 19, 2002; revised September 30, 2003. This work was supported by a Grant-in-Aid for Scientific Research (A) 13308017.

T. Matsuyama, X. Wu and T. Takai are with the Graduate School of Informatics, Kyoto University, Kyoto, 606–8501, Japan (e-mail: tm@vision.kuee.kyoto-u.ac.jp; wxj@vision.kuee.kyoto-u.ac.jp; takesi-t@vision.kuee.kyoto-u.ac.jp).

T. Wada is with the Faculty of Systems Engineering, Wakayama University, Wakayama, 640-8510, Japan (e-mail: twada@sys.wakayama-u.ac.jp).

Digital Object Identifier 10.1109/TCSVT.2004.823396

[1] While many real-time stereo systems (e.g., [8]–[12]) and image-based visual hull systems [13], [14] have been developed, they can reconstruct only a partial 2.5-D shape.

[2] An earlier version was published in [1].

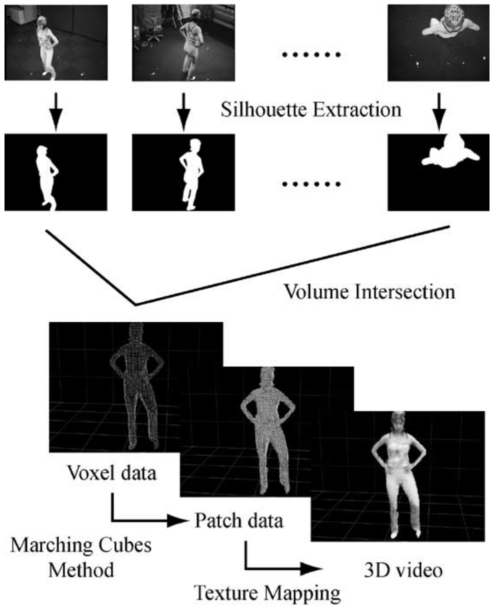


Fig. 1. 3-D video generation process.

half of this paper, we will present an algorithm of generating video texture on the reconstructed dynamic 3-D object surfaces. We first describe a naive view-independent rendering method and show its problems. Then, we improve the method by introducing image-based rendering techniques. Experimental results demonstrate the effectiveness of the improved method in generating high0fidelity object images from arbitrary viewpoints.[3]

II. Basic Method of 3-D Video Generation

Fig. 1 illustrates the basic process of generating a 3-D video frame in our system, described here.

1) **Synchronized Multiview Image Acquisition:** A set of multiview object images are taken simultaneously by a group of distributed video cameras (see the top row of Fig. 1).
2) **Silhouette Extraction:** Background subtraction is applied to each captured image to generate a set of multiview object silhouettes (see the second row from the top in Fig. 1).
3) **Silhouette Volume Intersection:** Each silhouette is back-projected into the common 3-D space to generate a visual cone encasing the 3-D object. Then, such 3-D cones are intersected with each other to generate the voxel representation of the object shape (see the third row from the bottom row of Fig. 1).
4) **Surface Shape Computation:** The discrete marching cubes method [18] is applied to convert the voxel representation to the surface patch representation, and then the surface patch is deformed to increase the accuracy of the reconstructed 3-D shape [19] (see the second row from the bottom of Fig. 1).
5) **Texture Mapping:** Color and texture on each patch are computed from the observed multiview images (see the bottom row of Fig. 1).

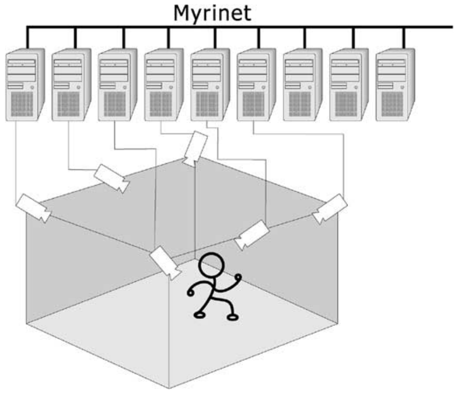


Fig. 2. PC cluster for our real-time active dynamic 3-D object shape reconstruction system.

By repeating the above process for each video frame, we can create a live 3-D motion picture. Note that, in the current implementation, since the surface patch deformation requires large computation time (approximately a few minutes per frame) due to its naive iterative optimization process, the entire process above cannot run in real time, while the 3-D shape reconstruction and the texture mapping run in near video-rate, respectively.

In the following sections, we describe the technical details of our real-time 3-D shape reconstruction system and high-fidelity video texture-mapping algorithm. As for the surface mesh deformation to increase the 3-D shape accuracy, refer to [19].

III. Real-Time Dynamic 3-D Object Shape Reconstruction System

A. System Organization

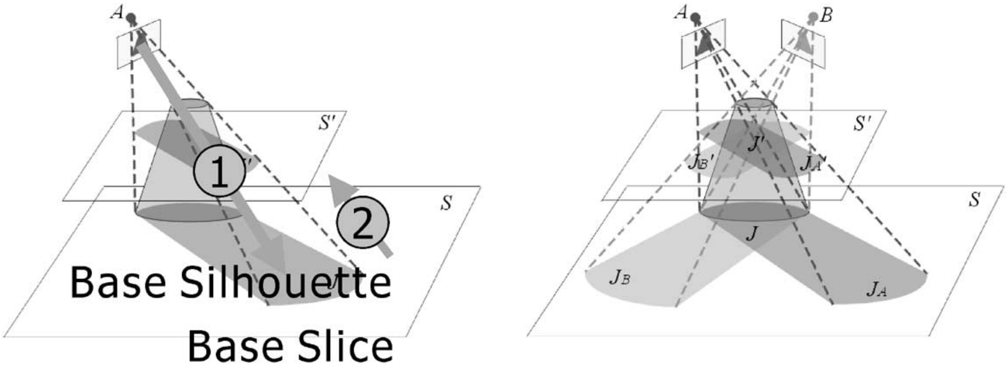
Fig. 2 illustrates the hardware organization of our real-time *active* dynamic 3-D object shape reconstruction system. It consists of:

- PC cluster: 30 node PCs (dual Pentium III 1 GHz) are connected through Myrinet, an ultrahigh-speed network (full duplex 1.28 Gb/s). A PM library for Myrinet PC clusters [20] allows very low latency and high-speed data transfer, based on which we can implement efficient parallel processing on the PC cluster.
- Distributed active video cameras: Among 30 PCs, 25 have calibrated fixed-viewpoint pan-tilt (FV-PT) cameras [21] for active object tracking and image capturing. In the FV-PT camera, the projection center stays fixed irrespective of any camera rotations, which greatly facilitates real-time active object tracking and its 3-D shape reconstruction in a widespread area [16].

We employ volume intersection [23]–[28] as a basic computational algorithm to obtain the 3-D shape of the object. This is because it can compute the full 3-D object shape by well-defined geometric computations, while stereo methods involve difficult matching processes as well as can generate only a partial 2.5-D shape.

As is well known, however, the 3-D shape reconstructed by the volume intersection is just an approximation. To increase the accuracy of the reconstructed 3-D shape, the authors of [29] proposed the space carving method, where photometric infor-

[3] An earlier version was published in [2].



(a)

(b)

Fig. 3. Plane-based volume intersection method.

mation as well as multiview silhouettes are employed. In [19], we proposed a deformable 3-D mesh model to reconstruct an accurate 3-D object shape by integrating object silhouettes, photometric properties, and 3-D motion flows computed from multiview video data.

Here we confine ourselves to the problem of how to realize real-time volume intersection.

Since the volume intersection involves a considerable amount of arithmetic operations, many methods for its efficient computation have been proposed. The work in [23], [28], and [7] employed octree-based volume intersection methods. The authors of [13], on the other hand, proposed an image-based volume intersection, where a 2.5-D depth map from an arbitrary viewpoint is generated by projecting and intersecting multiview object silhouettes on the image plane corresponding to that viewpoint.

To realize efficient volume intersection, we first developed the plane-based volume intersection method, where the 3-D voxel space is partitioned into a group of parallel planes and the cross section of the 3-D object volume on each plane is reconstructed. Second, we devised the plane-to-plane perspective projection (PPPP) algorithm to realize efficient plane-to-plane projection computation. Third, to realize real-time processing, we implemented parallel pipeline processing on a PC cluster system. In what follows, we describe these methods in detail.

B. Plane-Based Volume Intersection Method

Fig. 3 illustrates the plane-based volume intersection method. For each camera, an object silhouette is first projected onto a common base plane to generate a base silhouette, which then is mapped onto the other planes [Fig. 3(a)]. The cross section of the object on each plane can be obtained by calculating the 2-D intersection among the projected silhouettes [Fig. 3(b)]. This plane-to-plane back-projection (*homography*) [30] is computationally less expensive than general 3-D perspective projection as we demonstrate here:

- General perspective projection from 3-D point (X, Y, Z) to 2-D point $(x_1/x_3,\ x_2/x_3)$ can be represented by the following equation:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{1}$$

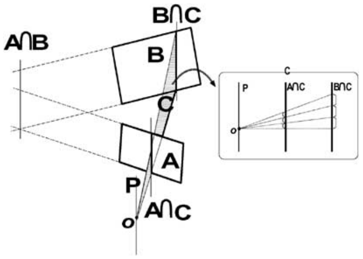


Fig. 4. Linear PPPP algorithm.

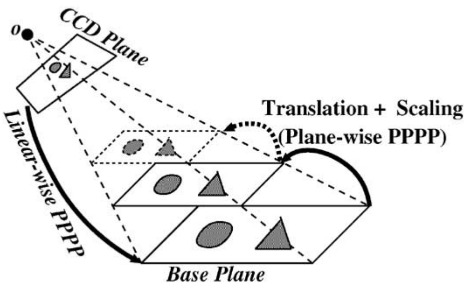


Fig. 5. Plane-wise PPPP.

This transformation requires nine additions, nine multiplications, and two divisions.

- In the case of the plane-to-plane projection, where the source 3-D point is constrained on a 2-D plane, the projection equation is simplified to the following equation:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}. \tag{2}$$

This requires six additions, six multiplications, and two divisions per point.

To accelerate the plane-to-plane projection computation further, we developed the following algorithm.

C. Accelerated PPPP Algorithm

Based on the geometric relations between a pair of planes involved in the projection, the acceleration of the PPPP algorithm can be achieved in the following two ways.

1) For planes which are not parallel, we devised the linear PPPP algorithm (see Fig. 4).
2) For parallel planes, we apply the plane-wise PPPP algorithm (see Fig. 5).

As will be described below, both algorithms consist of simple linear computations, which can be executed efficiently by popular graphics hardware.

Linear PPPP: In Fig. 4, we want to map a silhouette on plane A onto B, where A and B are not parallel. $A \bigcap B$ denotes the intersection line between the planes and O is the center of the perspective projection. Let P denote the line that is parallel to $A \bigcap B$ and passing O. Then, take any plane including P (C in Fig. 4), the image data on the intersection line $A \bigcap C$ is projected onto $B \bigcap C$. As shown in the right part of Fig. 4, this linear (i.e., line-based) perspective projection can be computed by a scaling operation, since $A \bigcap C$ and $B \bigcap C$ are parallel to each other. By rotating plane C around line P, we can map the entire 2-D image that is on A onto B.

In [6], we analyzed the computational complexity of this linear PPPP method and showed its computational efficiency. The work in [13] employed a similar plane-based projection method to map a silhouette from one image plane to another and proved its computational efficiency. The differences between their method and ours are: 1) we reconstruct a full 3-D shape while [13] generates a 2.5-D depth map for a specified viewpoint and 2) we employ the intersecting line $A \bigcap B$ to linearize the plane-to-plane projection while [13] used a group of epipolar lines.

Plane-Wise PPPP: As shown in Fig. 5, the projection between two parallel planes is simplified to 2-D isotropic scaling and translation as follows:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = s \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{3}$$

where s represents the scaling and (t_x, t_y) the translation vector. Equation (3) shows that this transformation requires two additions and two multiplications per point. Since this transformation is a pure 2-D geometric transformation, 2-D image processing hardware can be employed to accelerate the computation.

To realize real-time 3-D shape reconstruction, we next propose a parallel pipeline processing method for the above-mentioned plane-based volume intersection.

D. Parallelized Volume Intersection Method

The process of the plane-based volume intersection method can be divided into the following stages:

1) Back-projection:
   a) projection from the image plane of each camera onto the common base plane (linear PPPP);
   b) projection from the base plane to the other parallel planes (plane-wise PPPP);
2) Silhouette intersection on each plane.

To make this processing parallel on our PC cluster, we observe the following.

- Since the process a) of stage 1 is closely connected with image capturing and silhouette extraction processes, it should be executed on the same PC that captures an image.
- Since the silhouette intersection on each plane can be done independently of the silhouette intersection on the other planes, we partition a set of parallel planes into a group of subsets and assign a subset to each PC, which computes the silhouette intersection on each plane included in its assigned subset.
- To realize the above parallel silhouette intersection, we have to make each PC have a full set of multiview silhouettes. That is, after computing its own base plane silhouette, each PC broadcasts that data to all of the other PCs. As will be proved later, this broadcasting does not introduce large overhead, because the data size transmitted is small (i.e., 2-D bit image data representing the base plane silhouette) and the network speed is very high. Note that this silhouette duplication enables completely parallel silhouette intersection on the planes without any overhead.

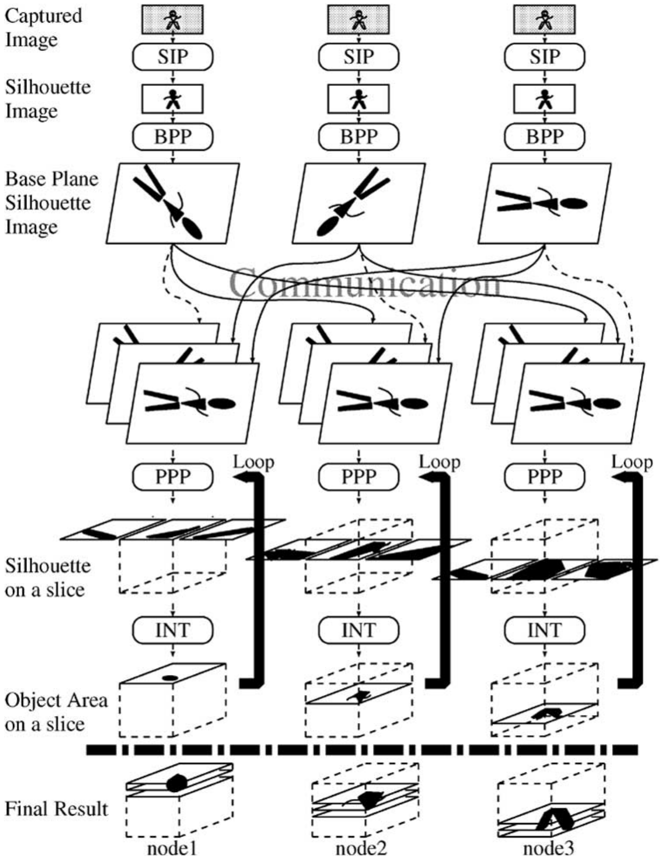


Fig. 6. Processing flow of the parallel pipelined 3-D shape reconstruction.

Fig. 6 illustrates the processing flow of the parallel pipelined 3-D shape reconstruction. It consists of the following five stages:

1) *Image Capture:* triggered by a capturing command, each PC with a camera captures a video frame (see the top row of Fig. 6).
2) *Silhouette Extraction:* each PC with a camera extracts an object silhouette from the video frame (see the second row from the top of Fig. 6).
3) *Projection to the Base Plan :* each PC with a camera projects the silhouette onto the common base plane in the 3-D space (see the third row from the top of Fig. 6).
4) *Base-Plane Silhouette Duplication:* all base-plane silhouettes are duplicated across all PCs over the network so that

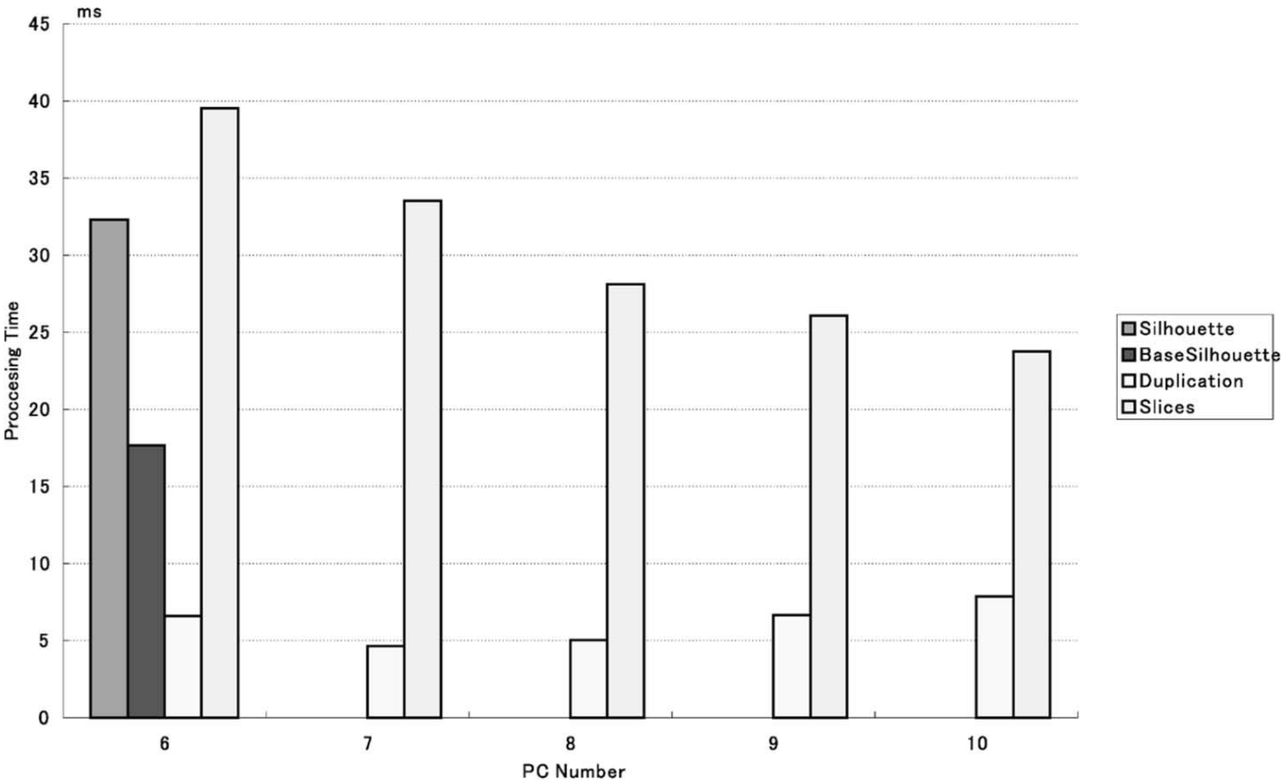


Fig. 7. Average computation time for each pipeline stage.

each PC has the full set of all base-plane silhouettes (see the fourth row of Fig. 6). Note that the data are distributed over all PCs in the system (i.e., PCs with and without cameras).

5) *Object Cross-Section Computation:* each PC computes object cross sections on specified parallel planes in parallel (see the three bottom rows of Fig. 6).

In addition to the above parallel processing, we introduced pipeline processing on each PC: five stages (corresponding to the five steps above) for a PC with a camera and two stages [steps 4) and 5)] for a PC without a camera. In this pipeline processing, each stage is implemented as a concurrent process and processes data independently of the other stages. Note that, since a process on the pipeline should be synchronized with its preceding and succeeding processes and, moreover, the stage-5 silhouette intersection cannot be executed until all silhouette data are prepared, the output rate (i.e., the rate of the 3-D shape reconstruction) is limited to the rate of the slowest stage.

E. Performance Evaluation

In the experiments of the real-time 3-D volume reconstruction, we used six digital IEEE1394 cameras (Sony DFW-VL500) placed at the ceiling (as in Fig. 2) for capturing multiview video data of a dancing human. We will discuss their synchronization method later. The size of the input image is 640 × 480 pixels and we measured the time taken to reconstruct one 3-D shape in the voxel size of 2 cm × 2 cm × 2 cm contained in a space of 2 m × 2 m × 2 m.

In the first experiment, we analyzed the processing time spent at each pipeline stage by using 6–10 PCs for computation. Fig. 7 shows the average computation time[4] spent at stages "Silhouette Extraction," "Projection to the Base-Plane," "Base-Plane Silhouette Duplication," and "Object Cross-Section Computation." Note that the image capturing stage is not taken into account in this experiment and will be discussed later.

From this figure, we can observe the following.

- The computation time for the *Projection to the Base Plane* stage is about 18 ms, which proves that the accelerated PPPP algorithm is very efficient. To verify the computational efficiency of the accelerated PPPP algorithm, we replaced it with a naive silhouette projection method, where an object silhouette in an image plane is projected onto the base plane pixel by pixel. Fig. 8 compares the average computation time spent at each pipeline stage between 6 PCs with the accelerated PPPP (left) and 6–12 PCs with the naive projection (right). In all of the cases, we used six cameras. This figure shows that the accelerated PPPP algorithm plays a crucial role in realizing real-time processing. Note that, in the latter cases, the silhouette duplication stage was also elongated considerably. The reason for this may be that the thread scheduling for the pipeline processing introduced additional overheads since the base silhouette projection stage took a very long time compared with the other stages.
- As is shown in the leftmost plots in Fig. 7, with six PCs (i.e., with no PCs without cameras), the bottleneck for real-time 3-D shape reconstruction rests at the *Object*

[4] For each stage, we calculated the average computation time of 100 video frames on each PC. The time shown in the graph is the average time for all PCs.

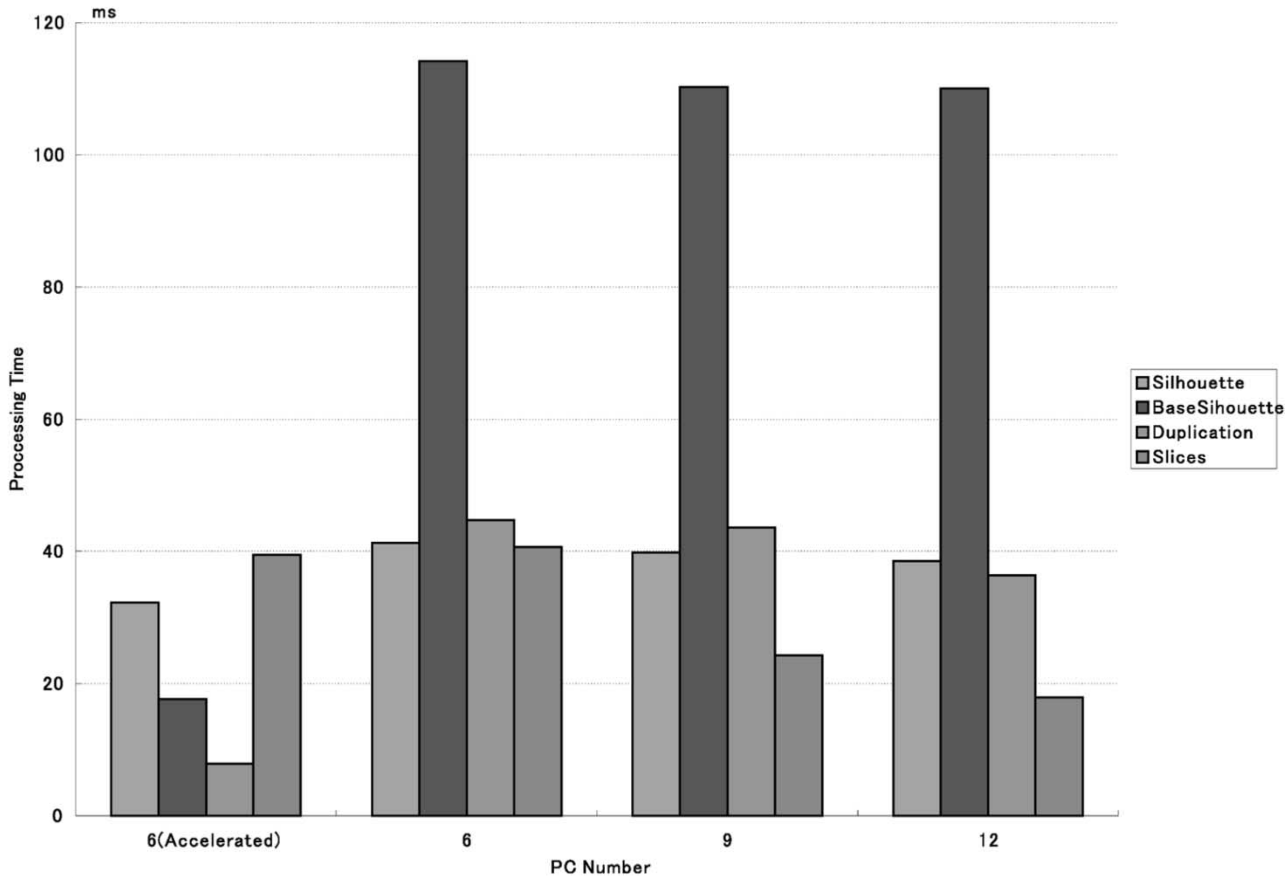
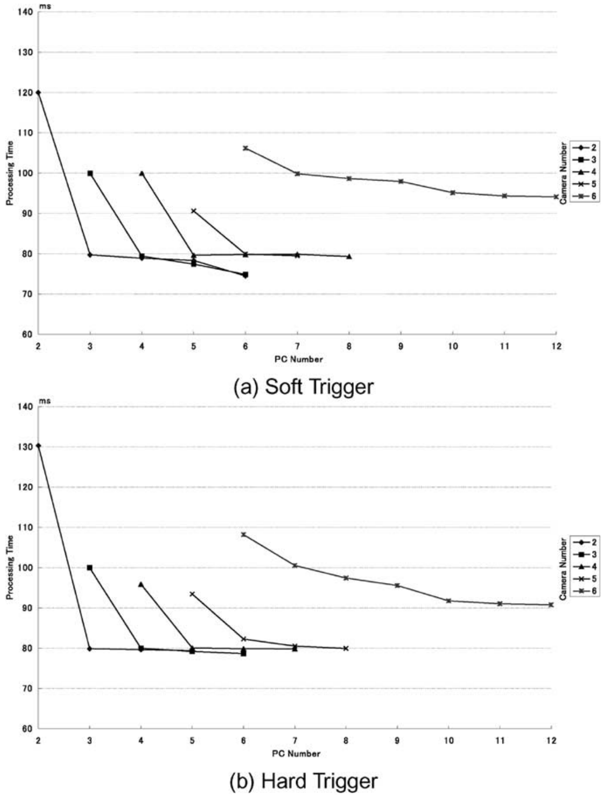


Fig. 8. Performance of the accelerated PPPP algorithm.



(a) Soft Trigger

(b) Hard Trigger

Fig. 9. Computation time for reconstructing one 3-D shape.

Cross-Section Computation stage, since this stage consumes the most computation time (i.e., about 40 ms).

- By increasing the number of PCs, the time taken for that most expensive stage decreases considerably while slightly increasing the data duplication overhead (the right part of Fig. 7). This proves that the proposed parallelization method is effective.
- With more than eight PCs, we can realize video-rate 3-D shape reconstruction.

In the second experiment, we measured the total throughput of the system including the image capturing process by changing the numbers of cameras and PCs. Fig. 9 shows the throughput[5] to reconstruct one 3-D shape.

In our PC cluster system, we developed two methods for synchronizing multiview video capturing: use an external trigger generator (hard trigger) and control the cameras through network communication (soft trigger).

From Fig. 9, we make the following observations.

- In both synchronization methods, while the throughput is improved by increasing PCs, it saturates at a constant value in all cases: 80∼90 ms in both methods.
- Comparing the hard and soft triggers, they show almost similar performance.

Since, as was proved in the first experiment, the throughput of the computation itself is about 30 ms, the elongated overall throughput is due to the speed of the image capture stage as well as the overhead involved in the process synchronization during the pipeline processing. That is, although a camera itself can capture images at a rate of 30 fps individually, the image capture

[5] The time shown in the graph is the average throughput for 100 frames.

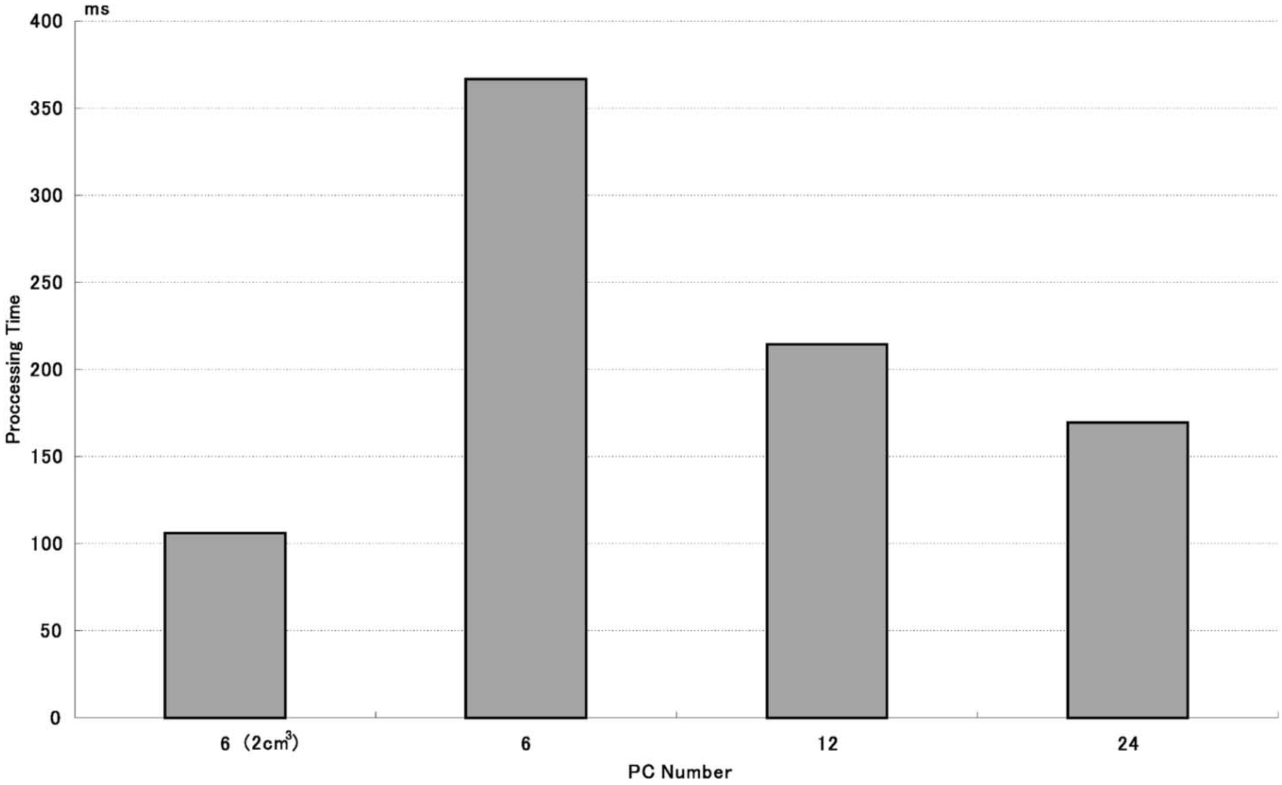


Fig. 10. Performance evaluation in the case of 1 cm × 1 cm × 1 cm voxels.

synchronization reduces its frame rate by half. This is partly because the external hard and soft triggers for synchronization are not synchronized with the internal hardware cycle of the camera and partly because it takes some time to synchronize PCs and transfer image data to PC memory:

- First of all, the camera we used (Sony DFW-VL500) reduces its frame rate down to 15 fps in the external hardware trigger mode. This is a major reason why the overall throughput is reduced in the hard trigger method.
- In the soft trigger method, since the actual image capturing is done based on the internal clock of each camera, the capturing timing varies from PC to PC slightly (i.e., at most 33 ms). This internal clock-driven image capturing also introduces a delay between the capturing command issued by a PC and the actual image capturing. Note that the hard trigger guarantees exactly synchronized image capturing.
- We use the isochronous data transfer mode of IEEE 1394 and a Linux device driver, which introduce some overheads for synchronized image data transfer via an IEEE 1394 line and buffering in the driver software.

In the third experiment, we increased the resolution: 1 cm × 1 cm ×1 cm voxels in a space of 2 m × 2 m × 2 m. Fig. 10 illustrates the overall throughputs for 6 PCs at the 2-cm voxel resolution (left) and 8–24 PCs at the 1-cm voxel resolution (right). In all cases, we used six cameras. While we can speed up the computation by increasing PCs, it will not possible to realize over 10 volume per second with the current system at a resolution of 1-cm voxels.

In summary, the experiments proved the effectiveness of the proposed real-time 3-D shape reconstruction system: the plane-based volume intersection method, its acceleration algorithm, and the parallel pipeline implementation. Moreover, the proposed parallel processing method is flexible enough to scale up the system by increasing the numbers of cameras and PCs. While we used off-the-shelf devices, we have to develop sophisticated video capturing hardware including cameras to realize video-rate 3-D shape reconstruction. To increase the voxel resolution, we have to employ faster PCs and/or make full use of graphics hardware.

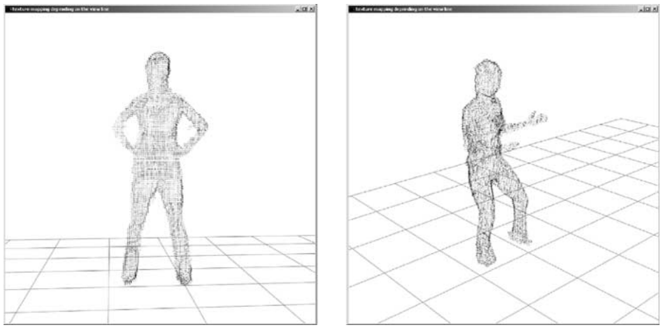

Texture mapping depending on the view line
Texture mapping depending on the view line

Fig. 11. Voxel representations of a 3-D object behavior.

IV. High-Fidelity Texture Mapping Algorithm

Fig. 11 illustrates snapshots of 3-D voxel data of a dancing person at a resolution of 1 cm × 1 cm × 1 cm reconstructed by the system described above. Then we apply to each voxel data the discrete marching cubes method [18] to convert the 3-D object shape into the triangular patch representation. Fig. 12(a) illustrates a close-up of the generated triangular patch data. As

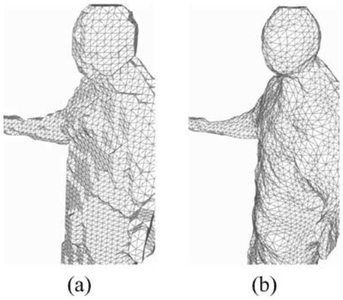


Fig. 12. (a) Surface patch model generated by the discrete marching cube method. (b) Surface patch model after deformation.

is well known and obvious from this figure, the 3-D shape generated is not smooth and its concave parts (e.g., neck) are not well reconstructed. To solve these problems, we developed a deformable 3-D mesh model, which uses photometric and motion information as well as multiview silhouettes [19]. Fig. 12(b) illustrates the result of the deformation, where a more accurate and smooth 3-D object shape is obtained.

Since the currently implemented deformation program requires large computation time (about a few minutes per frame) due to its naive iterative optimization process and a large number of vertices (i.e., 12 000–15 000 vertices per volume at 1-cm voxel resolution), the real-time processing is broken down at this stage and the subsequent texture mapping process is done as postprocessing, even if the mapping itself runs almost in real time by a PC with a modern graphics engine.

In this section, we propose a novel texture mapping algorithm to generate high-fidelity 3-D video. The problem we are going to solve here is how we can generate high-fidelity object images from arbitrary viewpoints based on the 3-D object shape with limited accuracy.

[3] first mapped each surface patch back to multiview images to obtain multiview textures for each patch and then took a weighted average of those textures. As will be described later, such view-independent patch-based texture mapping introduces jitters due to inaccurate 3-D object shape and/or misalignments involved in the camera calibration.

On the other hand, [11] and [13] employed image-based rendering methods to generate arbitrary view images based on 3-D shape data reconstructed from multiview images, where a virtual view direction is specified to control the blending process of multiview images. While such image-based rendering methods can avoid jitters in generated images, image sharpness is degraded due to the blending operation.

In what follows, we first describe a naive rendering method which is similar to [3] and show problems in a view-independent patch-based rendering method. Then, we improve the method by introducing image-based rendering techniques.

A. Naive Algorithm: Viewpoint-Independent Patch-Based Method

We first implemented a naive texture mapping algorithm, which selects the most ”appropriate” camera for each patch and then maps onto the patch the texture extracted from the image observed by the selected camera. Since this texture mapping is conducted independently of the viewer’s viewpoint of 3-D video, we call it the viewpoint-independent patch-based method (VIPBM).

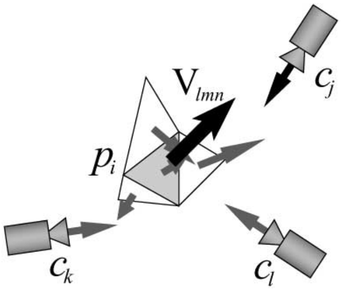


Fig. 13. VIPBM.

Algorithm (Fig. 13)

```
1) For each patch $p_i$, do the following processing.
2) Compute the locally averaged normal vector $V_{lmn}$ using normals of $p_i$ and its neighboring patches.
3) For each camera $c_j$, compute viewline vector $V_{c_j}$ directing toward the centroid of $p_i$.
4) Select such camera $c^*$ that the angle between $V_{lmn}$ and $V_{c_j}$ becomes maximum.
5) Extract the texture of $p_i$ from the image captured by camera $c^*$.
```

This method generates a fully textured 3-D object shape, which can be viewed from arbitrary viewpoints with ordinary 3-D graphics display systems. Moreover, its data size is very compact compared with that of the original multiviewpoint video data.

From the perspective of fidelity, however, the displayed image quality is not satisfying for the following reasons.

1) Due to the rough quantization of patch normals, the best camera c^* for a patch varies from patch to patch even if they are neighboring. Thus, textures on neighboring patches are often extracted from those images captured by different cameras (i.e., viewpoints), which introduces jitters in displayed images.
2) Since the texture mapping is conducted patch by patch and their normals are not accurate, textures of neighboring patches may not be smoothly connected. This introduces jitters at patch boundaries in displayed images.

To overcome these quality problems, we developed a viewpoint dependent vertex-based texture mapping algorithm. In this algorithm, the color (i.e., RGB value) of each patch vertex is computed taking into account the viewpoint of a viewer, and then the texture of each patch is generated by interpolating the color values of its three vertices.

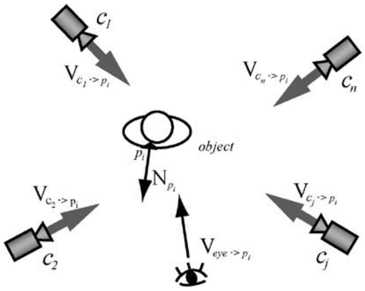


Fig. 14. Viewpoint and camera position.

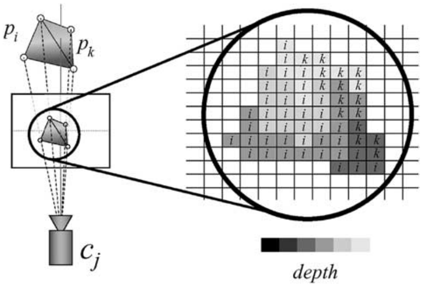


Fig. 15. Depth buffer.

B. Viewpoint-Dependent Vertex-Based Texture Mapping Algorithm

1) Definitions: First of all, we define words and symbols as follows (Fig. 14), where bold face symbols denote 3-D position/direction vectors:

- a group of cameras: $C = \{c_1, c_2, \ldots, c_n\}$;
- a viewpoint for visualization: **eye**;
- a set of surface patches: $P = \{p_1, p_2, \ldots, p_m\}$;
- outward normal vector of patch p_i: $\boldsymbol{n}_{p_i}$;
- a viewing direction from **eye** toward the centroid of p_i: $\boldsymbol{v}_{\text{eye}\to p_i}$;
- a viewing direction from c_j toward the centroid of p_i: $\boldsymbol{v}_{c_j\to p_i}$;
- vertices of p_i: $\mathbf{v}^k_{p_i}$ $(k = 1, 2, 3)$;
- vertex visible from c_j (defined later): $\mathbf{v}^k_{p_i,c_j}$;
- RGB values of $\mathbf{v}^k_{p_i,c_j}$ (defined later): $I(\mathbf{v}^k_{p_i,c_j})$;
- a depth buffer of c_j: $\mathbf{B}_{c_j}$. Geometrically, this buffer is the same as the image plane of camera c_j. Each pixel of $\mathbf{B}_{c_j}$ records the patch ID that is nearest from c_j as well as the distance to that patch from c_j (Fig. 15). When a vertex of a patch is mapped onto a pixel, its vertex ID is also recorded in that pixel.

2) Visible Vertex From Camera c_j: The vertex visible from c_j, $\mathbf{v}^k_{p_i,c_j}$, is defined as follows.

1) The face of patch p_i can be observed from camera c_j, if the following condition is satisfied:

$$\boldsymbol{n}_{p_i} \cdot \boldsymbol{v}_{c_j\to p_i} < 0. \tag{4}$$

2) $\mathbf{v}^k_{p_i}$ is not occluded by any other patches.

Then, we can determine $\mathbf{v}^k_{p_i,c_j}$ by the following process.

1) First, project all the patches that satisfy (4) onto the depth buffer B_{c_j}.

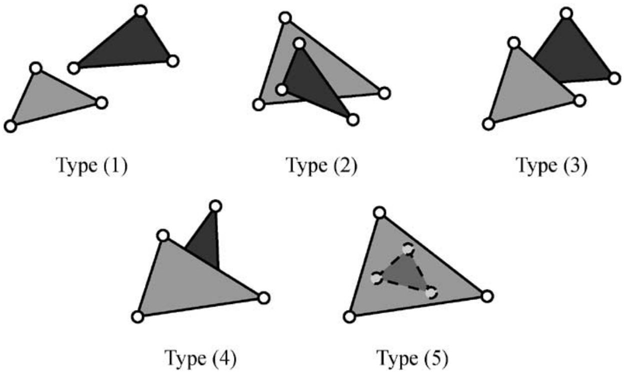


Fig. 16. Relations between patches.

2) Then, check the visibility of each vertex using the buffer. Fig. 16 illustrates possible spatial configurations between a pair of patches: all the vertices in type (1) and (2) are visible, while in type (5) three vertices of the occluded patch are not visible. In type (3) and (4), only some vertices are visible.

RGB values $I(\mathbf{v}^k_{p_i,c_j})$ of the visible vertex $\boldsymbol{v}^k_{p_i,c_j}$ are computed by

$$I(\mathbf{v}^k_{p_i,c_j}) = I_{c_j}(\hat{\mathbf{v}}_{p_i,c_j}) \tag{5}$$

where $I_{c_j}(\mathbf{v})$ shows RGB values of pixel $\mathbf{v}$ on the image captured by camera c_j and $\hat{\mathbf{v}}^k_{p_i,c_j}$ denotes the pixel position onto which the vertex $\mathbf{v}^k_{p_i,c_j}$ is mapped by the imaging process of camera c_j.

3) Algorithm:

```
1) Compute RGB values of all vertices visible from each camera in $C = \{c_1, c_2, \ldots, c_n\}$.
2) Specify the viewpoint **eye**.
3) For each surface patch $p_i \in P$, do steps 4)-9).
4) If $\boldsymbol{v}_{eye\to p_i} \cdot \boldsymbol{n}_{p_i} < 0$, then do 5)-9).
5) Compute weight $w_{c_j} = (\boldsymbol{v}_{c_j\to p_i} \cdot \boldsymbol{v}_{\text{eye}\to p_i})^m$, where $m$ is a weighting factor to be specified *a priori*.
6) For each vertex $\mathbf{v}^k_{p_i}$ $(k = 1, 2, 3)$ of patch $p_i$, do 7) and 8).
7) Compute the normalized weight for $\mathbf{v}^k_{p_i}$ by
```

$$\bar{w}^k_{c_j} = \frac{w^k_{c_j}}{\sum_l w^k_{c_l}}. \tag{6}$$

```
Here, if $\mathbf{v}^k_{p_i}$ is visible from camera $c_j$, then $w^k_{c_j} = w_{c_j}$, else $w^k_{c_j} = 0$.
8) Compute the RGB values $I(\mathbf{v}^k_{p_i})$ of $\mathbf{v}^k_{p_i}$ by
```

$$I(\mathbf{v}^k_{p_i}) = \sum_{j=1}^{n} \bar{w}^k_{c_j}\ I(\mathbf{v}^k_{p_i,c_j}). \tag{7}$$

```
9) Generate the texture of patch $p_i$ by linearly interpolating RGB values of its vertices. To be more precise, depending
```

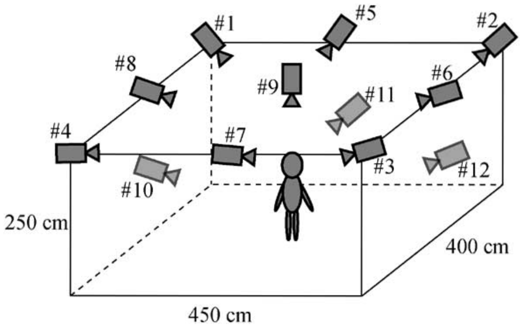


Fig. 17. Camera setting.

on the number of vertices with nonzero RGB values, the following processing is conducted:

- **Three vertices:** generate RGB values at each point on the patch by linearly interpolating the RGB values of three vertices.
- **Two vertices:** compute mean values of the RGB values of the two vertices, which is regarded as those of the other vertex. Then apply the linear interpolation on the patch.
- **One vertex:** paint the patch by the RGB values of the vertex.
- **No vertex:** texture of the patch is not generated: painted black, for example.

By the above process, an image representing an arbitrary view (i.e., from **eye**) of the 3-D object is generated.

C. Performance Evaluation

We evaluate the the performance of the proposed viewpoint-dependent vertex-based method (VDVBM) under the camera configuration shown in Fig. 17 with the VIPBM qualitatively. Fig. 18 compares those images generated by VDVBM-1, VDVBM-2, and VIPBM with an original video image. Note that, to evaluate the performance of VDVBM, we employed two methods: VDVBM–1 generates images including real images captured by camera $c_j =$ **eye** itself (i.e., cameras 5 and 11 in Fig. 18, respectively), while VDVBM–2 excludes such real images captured by camera c_j. We can observe that VIPBM introduces many jitters in images, which are considerably reduced by VDVBM.

Then, we conducted quantitative performance evaluations. That is, we calculate RGB rms errors between a real image captured by camera $c_j =$ **eye** and its corresponding images generated by VIPBM, VDVBM–1, and VDVBM–2, respectively.

The experiments were conducted under the following settings:

- image size: 640 × 480[pixel] 24-b RGB color;
- viewpoint (**eye**): camera 5;
- weighting factor in VDVBM: $m = 5$.

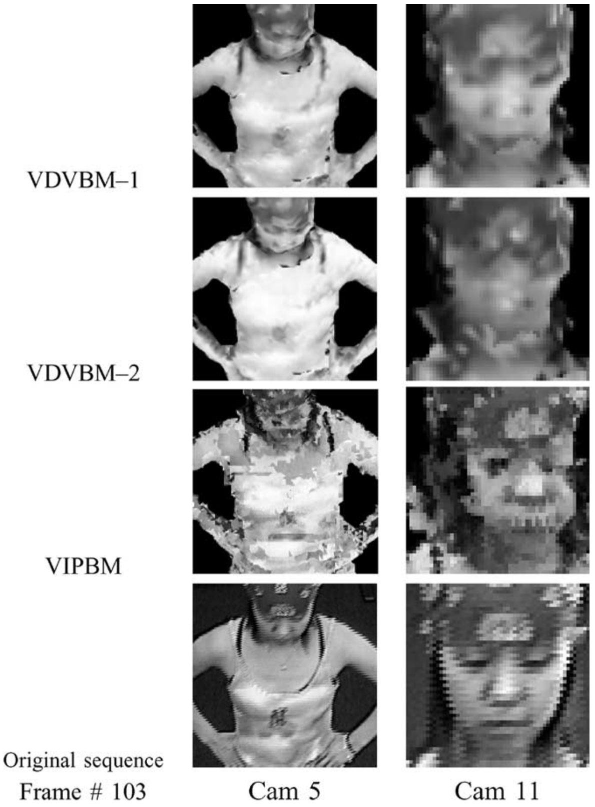


Fig. 18. Cropped images generated by VDVBM-1, VDVBM-2, VIPBM, and the original sequence.

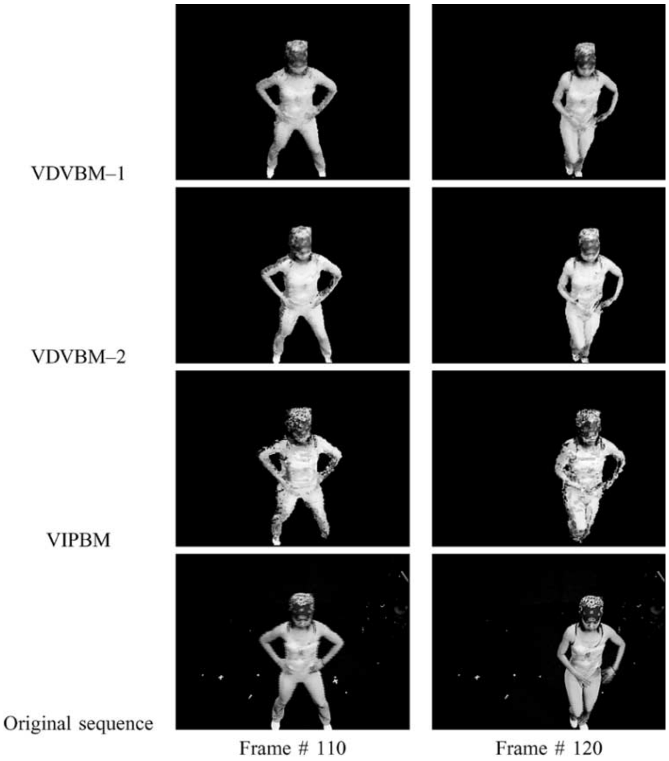


Fig. 19. Sample images of 3-D video generated by VDVBM–1, VDVBM–2, and VIPBM from $\mathbf{eye} = \mathrm{cam}\ 5$ in Fig. 17.

The superiority of VDVBM and its high-fidelity image generation capability can be easily observed in Fig. 19, where real and generated images for frames 110 and 120 are illustrated. Fig. 20 illustrates the experimental results, where rms errors for frames 95–145 are computed. This figure shows that VDVBM performs better than VIPBM.

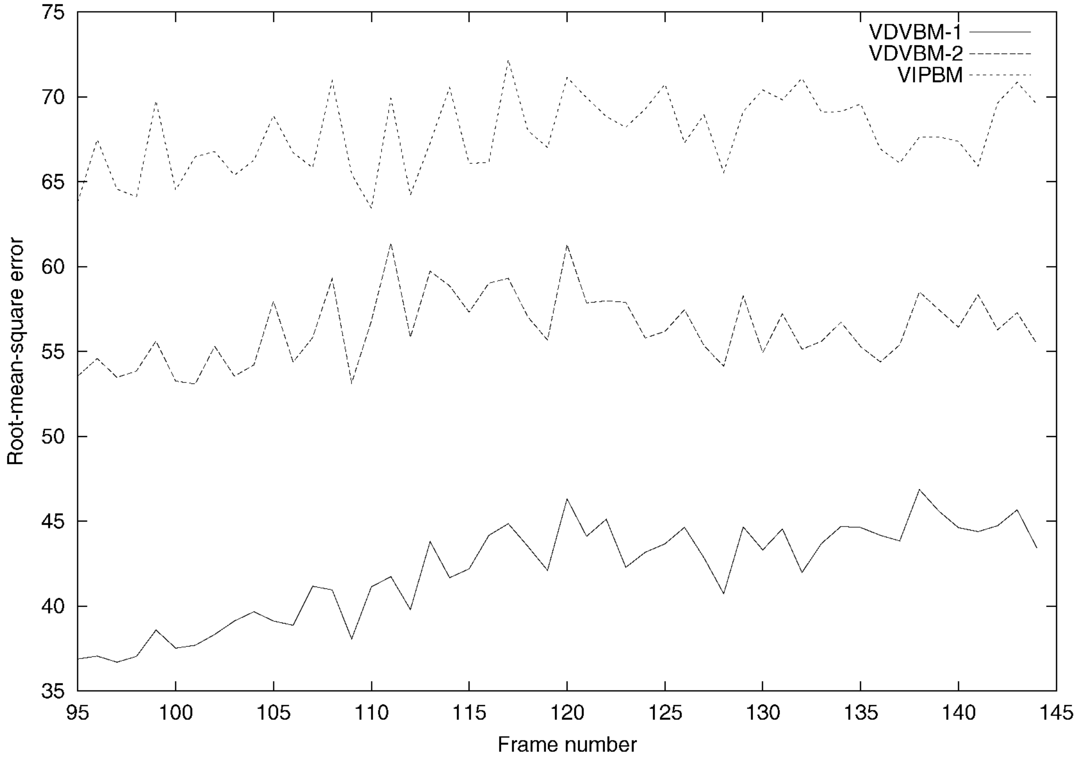


Fig. 20. RMS error of RGB value (1).

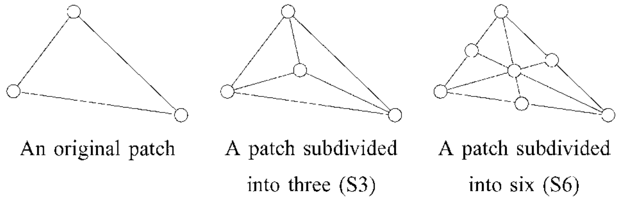

An original patch
A patch subdivided into three (S3)
A patch subdivided into six (S6)

Fig. 21. Subdivision of a 3-D surface patch.

Next, we tested how we can improve the performance of VDVBM by increasing the spatial resolution of the 3-D object surface patch data. Fig. 21 shows the method of subdividing a patch into three (S3) and six (S6) subpatches to increase the spatial resolution.

We examine the average side length of a projected patch on the image plane of each camera by projecting original and subdivided 3-D surface patches onto the image plane. Fig. 22 shows the mean side length of the patches projected on the image plane of each camera. Note that, since camera 9 is located closer to the 3-D object (see Fig. 17), object images captured by it become larger than those by the other cameras, which caused bumps (i.e., larger side length in pixel) in the graphs in Fig. 22.

We can observe that the spatial resolution of S6 is approximately the same as that of an observed image (i.e., one pixel). That is, S6 attains the finest resolution, which physically represents about 5 mm on the object surface. To put this another way, we can increase the spatial resolution up to six subdivisions, which improves the quality of images generated by VDVBM.

To quantitatively evaluate the quality achieved by using subdivided patches, we calculated rms errors between real images and images generated by VDVBM-1 with the original patches, S3, and S6, respectively (Fig. 23).

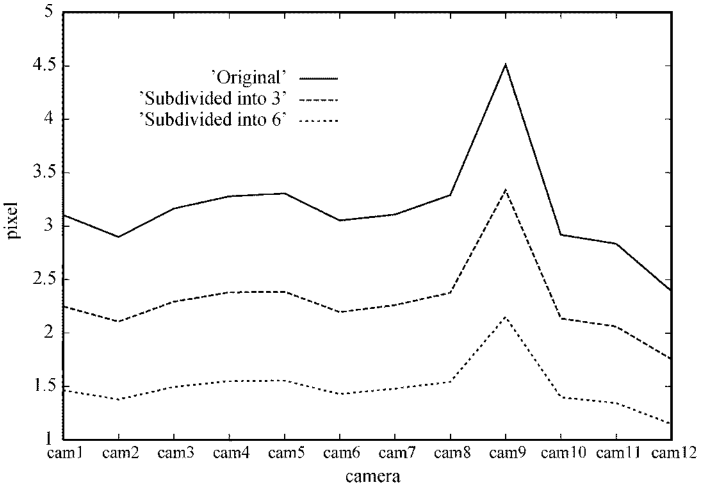


Fig. 22. Mean side length (in pixels) of patches projected on the image plane of each camera.

Fig. 23 shows that subdividing patches does not numerically reduce the errors. The reason for this observation is as follows. Fig. 24 shows the spatial distribution of the color difference between a real image and a generated image, from which we can see that large errors arise around the object contour and texture edges. These errors are difficult to reduce by subdividing patches because they come from motion blur, the misalignment of the camera calibration, or the asynchronization by the soft trigger image capturing. Fidelity of images generated with subdivided patches, however, is improved in smooth object surface areas (Fig. 25). Thus, subdividing patches is effective from a fidelity point of view.

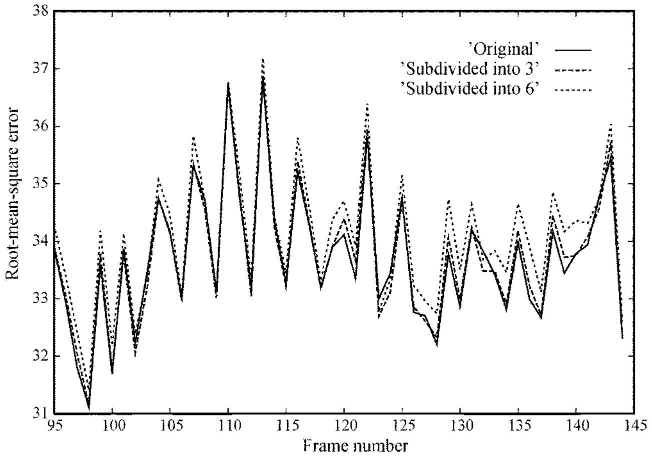


Fig. 23. RMS errors of RGB value (2).

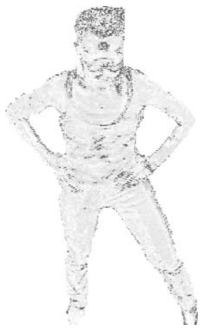


Fig. 24. Color difference between a real image and a generated image (frame 106).

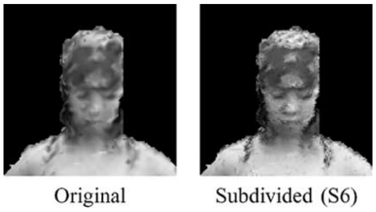


Fig. 25. Example images rendered with original and subdivided patches (frame 106).

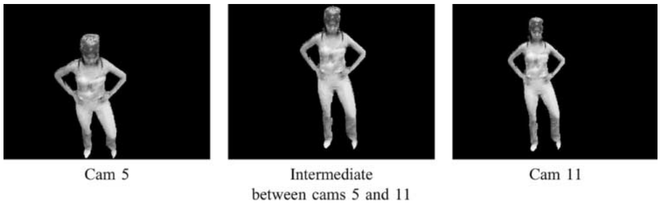


Fig. 26. Visualized 3-D video with subdivided patches (frame 103).

Note that the zigzag patterns in Figs. 20 and 23 were caused due to the imperfectness of the synchronization of the image capturing process. Since we used the soft trigger mode to capture multiview video data in this experiment, the image capture timing varied slightly PC by PC. This timing deviation sometimes increased the inaccuracy of the reconstructed 3-D shape, because the dancer moved her hands rather quickly.

Finally, we show examples generated by VDVBM–1 with subdivided patches (S6) viewed from cameras 5 and 11, and an intermediate point between them (Fig. 26). Fig. 26 shows that the images generated by VDVBM look almost real even when they are viewed from the intermediate point of the cameras.

For rendering 3-D video data as in Fig. 26, we used a popular PC: (CPU: Xeon 2.2 GHz, Memory: 1 GB, Graphics Processor: GeForce 4 Ti 4600, Graphics library: DirectX 9.0 b) and used the following two stage process.

1) First, compile a temporal sequence of reconstructed 3-D shape data and multiview video into a temporal sequence of vertex lists, where multiview RGB values are associated with each vertex. It took about 2.8 s to generate a vertex list for a frame of 3-D video.
2) Then, with the vertex list sequence, arbitrary VGA views of the 3-D video sequence can be rendered at 6.7 msec/frame. Thus, we can realize real-time interactive browsing of 3-D video with a PC. Note also that, since we can render a pair of stereo images in real time (i.e., 14 ms/stereo-pair), we can enjoy pop-up 3-D image interactively with a 3-D display monitor.

V. Conclusion

Three-dimensional video records an object's full 3-D shape, motion, and surface texture. In this paper, we first proposed a real-time parallel pipeline volume intersection method on a PC cluster: the plane-based volume intersection method, its acceleration algorithm, and the parallel pipeline implementation. The quantitative performance evaluations demonstrated that the acceleration and parallelizing algorithms we proposed are very efficient and enabled us to reconstruct a dynamic full 3-D shape over 10 volume per second at a 2 cm × 2 cm × 2 cm voxel resolution.

In the latter half of the paper, we proposed a high-fidelity texture mapping method. The qualitative and quantitative performance evaluations demonstrated that the proposed texture mapping method can produce object images from arbitrary viewpoints in almost the same quality as real video data.

As listed in the introduction, to make 3-D video usable in everyday life, we still have to develop methods of:

- higher speed and more accurate 3-D behavior reconstruction;
- 3-D shape acquisition in a widespread area and for multiple people;
- more natural image generation;
- effective data compression;
- editing 3-D video for artistic image contents.

Acknowledgment

The authors are grateful to Real World Computing Partnership, Japan, for allowing us to use their multiviewpoint video data.

References

[1] X. Wu and T. Matsuyama, "Real-time active 3-D shape reconstruction for 3-D video," in *Proc. 3rd Int. Symp. Image and Signal Processing and Analysis*, 2003, pp. 186–191.
[2] T. Matsuyama and T. Takai, "Generation, visualization, and editing of 3-D video," in *Proc. Symp. 3-D Data Processing Visualization and Transmission*, 2002, pp. 234–245.

[3] S. Moezzi, L. Tai, and P. Gerard, "Virtual view generation for 3-D digital video," *IEEE Multimedia*, pp. 18–26, 1997.

[4] G. Cheung and T. Kanade, "A real time system for robust 3-D voxel reconstruction of human motions," in *Proc. Computer Vision and Pattern Recognition*, 2000, pp. 714–720.

[5] T. Kanade, P. Rander, S. Vedula, and H. Saito, "Virtualized reality: Digitizing a 3-D time-varying event as is and in real time," in *Mixed Reality*, Y. Ohta and H. Tamura, Eds. Tokyo, Japan: Ohmsha, 1999, pp. 41–57.

[6] T. Wada, X. Wu, S. Tokai, and T. Matsuyama, "Homography based parallel volume intersection: Toward real-time reconstruction using active camera," in *Proc. Computer Architectures for Machine Perception*, 2000, pp. 331–339.

[7] E. Borovikov and L. Davis, "A distributed system for real-time volume reconstruction," in *Proc. Computer Architectures for Machine Perception*, 2000, pp. 183–189.

[8] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka, "A stereo machine for video-rate dense depth MApping and its new applications," in *Proc. Computer Vision and Pattern Recognition*, 1996, pp. 196–202.

[9] J. Mulligan, V. Isler, and K. Daniilidis, "Trinocular stereo: A real-time algorithm and its evaluation," *Int. J. Comput. Vis.*, vol. 47, no. 1/2/3, pp. 51–61, 2002.

[10] H. Hirschmuller, P. R. Innocent, and J. Garibaldi, "Real-time correlation-based stereo vision with reduced border errors," *Int. J. Comput. Vis.*, vol. 47, no. 1/2/3, pp. 229–246, 2002.

[11] T. Naemura, J. Tago, and H. Harashima, "Real-time video-based modeling and rendering of 3-D scenes," *IEEE Comput. Graph. Applicat.*, vol. 22, pp. 66–73, Mar./Apr. 2002.

[12] P. Kauff and O. Schreer, "An immersive 3-D video-conferencing system using shared virtual team user environments," in *Proc. ACM Conf. Collaborative Virtual Environments*, 2002, pp. 105–112.

[13] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. MacMillan, "Image-based visual hulls," in *Proc. SIGGRAPH*, 2000, pp. 369–374.

[14] H. Baker, D. Tanguay, I. Sobel, D. Gelb, M. Gross, W. Culbertson, and T. Malzbender, "The coliseum immersive teleconferencing system," in *Proc. Int. Workshop Immersive Telepresence*, 2002.

[15] M. Turk and G. Robertson, "Perceptual user interfaces," *Commun. ACM*, vol. 43, no. 3, pp. 33–34, 2000.

[16] T. Matsuyama and N. Ukita, "Real-time multi-target tracking by a cooperative distributed vision system," *Proc. IEEE*, vol. 90, pp. 1136–1150, July 2002.

[17] T. Matsuyama and R. Yamashita, "Requirements for Standardization of 3-D Video," ISO/IEC JTC1/SC29/WG11, MPEG2002/M8107, 2002.

[18] Y. Kenmochi, K. Kotani, and A. Imiya, "Marching cubes method with connectivity consideration" (in Japanese), *PRMU–98–218*, pp. 197–204, 1999.

[19] S. Nobuhara and T. Matsuyama, "Dynamic 3-D shape from multi-viewpoint images using deformable mesh model," in *Proc. 3rd Int. Symp. Image and Signal Processing and Analysis*, 2003, pp. 192–197.

[20] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato, "PM: An operating system coordinated high performance communication library," in *High-Performance Computing and Networking*, P. Sloot and B. Hertzberger, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1225, Lecture Notes in Computer Science, pp. 708–717.

[21] T. Wada and T. Matsuyama, "Appearance sphere:Background model for pan-tilt- zoom camera," in *Proc. 13th ICPR*, 1996, pp. A-718–A-722.

[22] T. Matsuyama, "Cooperative distributed vision – Dynamic integration of visual perception, action, and communication," in *Proc. Image Understanding Workshop*, 1998, pp. 365–384.

[23] M. Potmesil, "Generating octree models of 3-D objects from their silhouettes in a sequence of images," *Comput. Vis., Graph, Image Processing*, vol. 40, pp. 1–29, 1987.

[24] W. N. Martin and J. K. Aggarwal, "Volumetric description of objects from multiple views," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 150–158, Feb. 1987.

[25] A. Laurentini, "The visual hull concept for silhouette based image understanding," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, pp. 150–162, Feb. 1994.

[26] ——, "How far 3D shapes can be understood from 2D silhouettes," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 188–195, Feb. 1995.

[27] P. Srivasan, P. Liang, and S. Hackwood, "Computational geometric methods in volumetric intersections for 3-D reconstruction," *Pattern Recognit.*, vol. 23, no. 8, pp. 843–857, 1990.

[28] R. Szeliski, "Rapid octree construction from image sequences," *Proc. CVGIP: Image Understanding*, vol. 58, no. 1, pp. 23–32, 1993.

[29] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," in *Proc. IEEE Int. Conf.Computer Vision*, 1999, pp. 307–314.

[30] J. Semple and G. Kneebone, *Algebraic Projective Geometry*. Oxford, U.K.: Oxford Sci., 1952.

Takashi Matsuyama received the B.Eng., M.Eng., and D.Eng. degrees in electrical engineering from Kyoto University, Kyoto, Japan, in 1974, 1976, and 1980, respectively.

He is currently a Professor in the Department of Intelligence Science and Technology, Graduate School of Informatics, Kyoto University. His research interests include knowledge-based image understanding, computer vision, image media processing, and artificial intelligence. He has authored or coauthored approximately 100 papers and books including two research monographs, *A Structural Analysis of Complex Aerial Photographs* (New York: Plenum, 1980) and *SIGMA: A Knowledge-Based Aerial Image Understanding System* (New York: Plenum, 1990). He is on the editorial boards of *Computer Vision and Image Understanding, Pattern Recognition,* and the *International Journal on Computer Vision.*

Prof. Matsuyama is a Fellow of the International Association for Pattern Recognition and a member of the Institute of Electronics, Information and Communication Engineers of Japan, the Information Processing Society of Japan, the Japanese Society for Artificial Intelligence, and the IEEE Computer Society. He was the recipient of six best paper awards from Japanese and international academic societies including the Marr Prize at ICCV'95.

Xiaojun Wu received the B.Eng. degree in electrical engineering and the M.S. degree in informatics, intelligence science, and techology from Kyoto University, Kyoto, Japan, in 1998 and 2000, respectively.

He is currently a Research Assistant with the Department of Intelligence Science and Technology, Kyoto University. He is currently involved with the field of computer vision, focusing on the 3-D shape reconstruction, and real-time parallel proccessing technology.

Takeshi Takai received the B.Eng. degree in electrical engineering from Doshisha University, Kyoto, Japan, in 1998 and the M.Eng. degree in information processing from Nara Institute of Science and Technology, Ikoma, Japan, in 2000.

He is currently a Research Assistant with the Department of Intelligence Science and Technology, Kyoto University, Kyoto, Japan. His research interests include computer vision, computer graphics, and virtual reality.

Toshikazu Wada (M'03) received the B.Eng. degree in electrical engineering from Okayama University, Okayama, Japan, in 1984, the M.Eng. degree in computer science from Tokyo Institute of Technology, Tokyo, Japan, in 1987, and the D.Eng. degree in applied electronics from Tokyo Institute of Technology, Tokyo, Japan, in 1990.

He is currently a Professor with the Department of Computer and Communication Sciences, Faculty of Systems Engineering, Wakayama University, Wakayam, Japan. His research interests include pattern recognition, computer vision, image understanding, and artificial intelligence.

Prof. Wada is a member of the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan, the Information Processing Society of Japan, and the Japanese Society for Artificial Intelligence. He was the recipient of the Marr Prize at the International Conference on Computer Vision in 1995, the Yamashita Memorial Research Award from the Information Processing Society Japan (IPSJ), and the Excellent Paper Award from the IEICE of Japan.