Real-Time 3D Shape Reconstruction, Dynamic 3D Mesh Deformation, and High Fidelity Visualization for 3D Video

T. Matsuyama, X. Wu, T. Takai, and S. Nobuhara

Graduate School of Informatics, Kyoto University
Sakyo, Kyoto, 606-8501, Japan

Abstract

3D video[1] is the ultimate image media recording dynamic visual events in the real world as is; it records time varying 3D object shape with high fidelity surface properties (i.e. color and texture). Its applications cover wide varieties of personal and social human activities: entertainment (e.g. 3D game and 3D TV), education (e.g. 3D animal picture books), sports (e.g. sport performance analysis), medicine (e.g. 3D surgery monitoring), culture (e.g. 3D archive of traditional dances) and so on. In this paper, we propose: 1. a PC cluster system for real-time reconstruction of dynamic 3D object action from multi-view video images, 2. a deformable 3D mesh model for reconstructing the accurate dynamic 3D object shape, and 3. an algorithm of rendering natural-looking texture on the 3D object surface from the multi-view video images. Experimental results with quantitative performance evaluations demonstrate the effectiveness of these methods in generating high fidelity 3D video from multi-view video images.

Key words: Dynamic 3D Shape Reconstruction, Real-Time Processing, PC Cluster, Multi-Viewpoint Video, Deformable Mesh Model, Video Texture Mapping

1 Introduction

3D video[1][4] is the ultimate image media recording dynamic visual events in the real world as is; it records time varying 3D object shape with high fidelity

This paper addresses a comprehensive report of our research activities on 3D video. Earlier versions of sections 3.1, 4, and 5.2 were reported in [2] and [3].

surface properties (i.e. color and texture). Its applications cover wide varieties of personal and social human activities: entertainment (e.g. 3D game and 3D TV), education (e.g. 3D animal picture books), sports (e.g. sport performance analysis), medicine (e.g. 3D surgery monitoring), culture (e.g. 3D archive of traditional dances) and so on.

Several research groups developed real-time 3D shape reconstruction systems for 3D video and have opened up the new world of image media [1] [5] [6] [7] [8] [9]. All these systems focus on capturing human body actions and share a group of distributed video cameras for real-time synchronized multi-viewpoint action observation. While the real-timeness of the earlier systems[1] [5] was confined to the synchronized multi-view video observation alone, the parallel volume intersection on a PC cluster has enabled the real-time full 3D shape reconstruction [6] [7] [8] [10].

To cultivate the 3D 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 3D shape reconstruction has been attained only in coarse resolution.
- *High Fidelity:* To obtain high fidelity 3D video in 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:* 3D areas observable by the systems developed so far are confined to small ones (e.g. about $2m \times 2m \times 2m$ in [6]), which should be extended considerably to capture human actions like sports playing.
- *Data Compression:* Since naive data representation of 3D video results in huge data, effective compression methods are required to store and transmit 3D video data[11].
- *Editing and Visualization:* Since editing and visualization of 3D video are conducted in the 4D space (3D geometric + 1D temporal), we have to develop human-friendly 3D video editors and visualizers that help a user to understand dynamic events in the 4D space[12].

This paper first describes a PC cluster system for reconstructing dynamic 3D object action from multi-view video images, by which a temporal series of 3D voxel representations of the object action can be obtained in real-time. Following an overview of our earlier system[6][12][2], we propose a new plane-based volume intersection method for real-time *active* 3D action reconstruction. Then, we present a deformable 3D mesh model for reconstructing the accurate dynamic 3D object shape. With this deformation, we can reconstruct even concave parts of the 3D object, which cannot be managed by the volume intersection. The last part of the paper describes an algorithm of rendering video texture on the reconstructed dynamic 3D object surface and evaluates

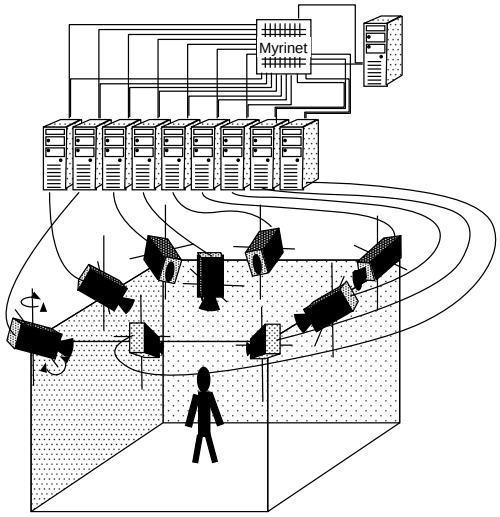


Fig. 1. PC cluster for real-time active 3D object shape reconstruction.

its performance using 3D mesh data with and without the deformation. Experimental results with quantitative performance evaluations demonstrate the effectiveness of these methods in generating high fidelity 3D video from multi-view video images.

2 Basic Scheme of 3D Video Generation

Figure 1 illustrates the architecture of our real-time *active* 3D object shape reconstruction system. Based on the experiences obtained using the first generation system with 16 PCs and 12 off-the-shelf NTSC active cameras [6][12], we developed the second generation system. It consists of

- PC cluster: 30 node PCs (dual Pentium III 1GHz) are connected through Myrinet, an ultra high speed network (full duplex 1.28Gbps). PM library for Myrinet PC clusters[13] 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, 25 PCs have Fixed-Viewpoint Pan-Tilt (FV-PT) cameras[14], respectively, for active object tracking and image capturing. We newly developed a FV-PT camera module, where the projection center stays fixed irrespectively of any camera rotations, which greatly facilitates real-time active object tracking and 3D shape reconstruction. Moreover, digital IEEE 1394 video cameras are employed to enhance video quality (image size: 640×480).
- Camera layout: As will be discussed in Section 3.2.3, the cameras are placed on the floor as well as at the ceiling to capture fully surrounding views of an object, while the first generation system employed ceiling cameras alone.

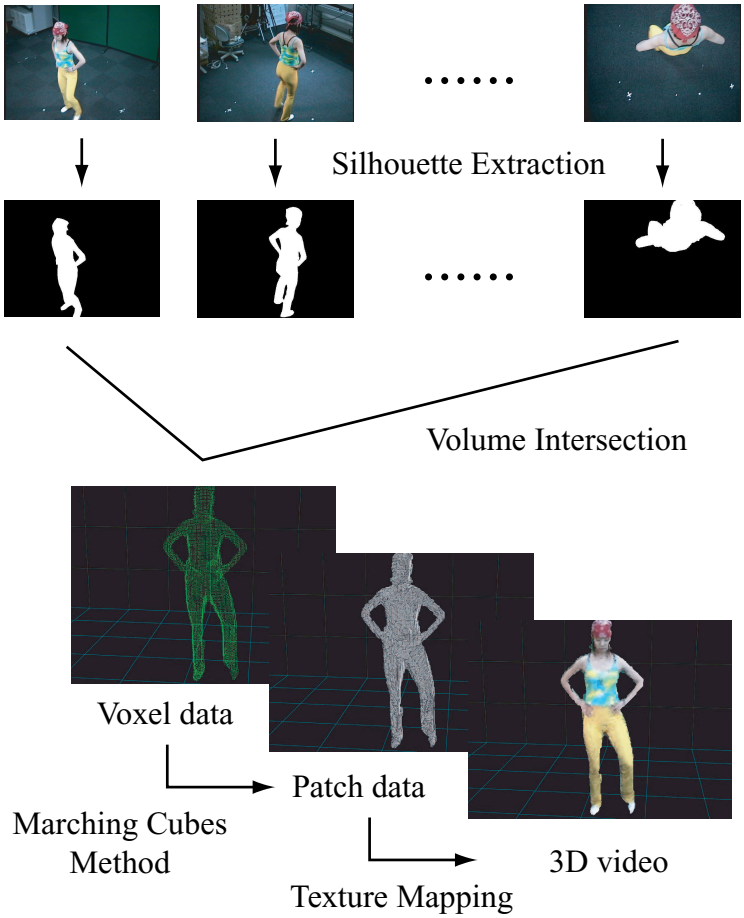


Fig. 2. 3D video generation process

Figure 2 illustrates the basic process of generating a 3D video frame in our system:

(1) **Synchronized Multi-View Image Acquisition**: A set of multi-view object images are taken simultaneously by a group of distributed video cameras (top row in Figure 2).
(2) **Silhouette Extraction**: Background subtraction is applied to each captured image to generate a set of multi-view object silhouettes (second top row in Figure 2).
(3) **Silhouette Volume Intersection**: Each silhouette is back-projected into the common 3D space to generate a visual cone encasing the 3D object. Then, such 3D cones are intersected with each other to generate the voxel representation of the object shape (third bottom in Figure 2).
(4) **Surface Shape Computation**: The discrete marching cubes method[15] is applied to convert the voxel representation to the surface patch representation. Then the generated 3D mesh is deformed to obtain accurate 3D object shape(second bottom in Figure 2).
(5) **Texture Mapping**: Color and texture on each patch are computed from the observed multi-view images (bottom in Figure 2).

By repeating the above process for each video frame, we have a live 3D motion picture.

In the following sections, we describe our real-time 3D shape reconstruction system, deformable mesh model, and high fidelity video texture mapping algorithm.

3 Real-Time Dynamic 3D Object Shape Reconstruction System

This section proposes a real-time dynamic 3D object shape reconstruction system using the PC cluster in Figure 1.

Here we classify the 3D shape reconstruction into the following two levels:

- **Level 1:** Real-time 3D shape reconstruction without camera actions.
- **Level 2:** Real-time 3D shape reconstruction *with* camera actions for tracking a moving object.

For the level one, we give an overview of our earlier system: the plane-based parallel pipeline volume intersection method[6][12][2].

For the level two, i.e. active 3D shape reconstruction of a moving object, we augment the plane-based volume intersection method so that it can cope with dynamic camera actions for tracking a moving object.

3.1 Real-Time 3D Shape Reconstruction without Camera Actions

3.1.1 Parallel Pipeline Plane-Based Volume Intersection Method

Silhouette Volume Intersection [16] [17] [18] [19] [20] [21] [22] [23] is the most popular method for reconstructing 3D object shape from multi-view images (Figure 3). This method is based on the *silhouette constraint* that a 3D object is encased in the 3D frustum produced by back-projecting a 2D object silhouette on an image plane. With multi-view object images, therefore, an approximation of the 3D object shape can be obtained by intersecting such frusta. This approximation is called *visual hull* [24]. Recently, this method was further extended using photometric information to reconstruct more accurate shapes [25].

In the naive volume intersection method, the 3D space is partitioned into small voxels (Figure 4(a)), and each voxel is mapped onto each image plane to examine whether or not it is included in the object silhouette. This voxel-wise

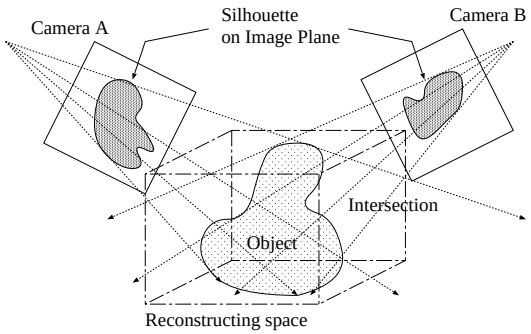


Fig. 3. Silhouette volume intersection.

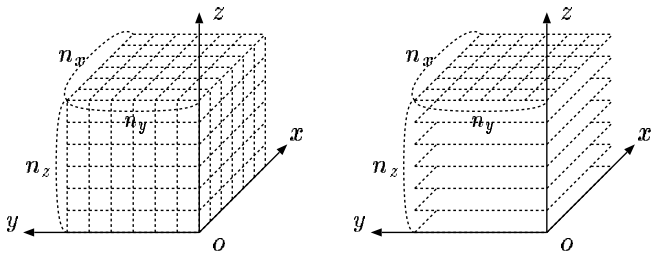


Fig. 4. 3D shape representations: (a) 3D voxel space, (b) parallel plane space.

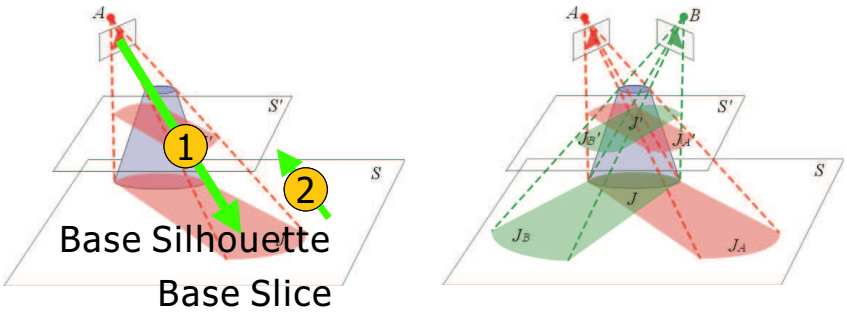


Fig. 5. Plane-based volume intersection method

perspective projection process requires very expensive computation. Borovikov and Davis[7] introduced an octree to represent the 3D voxel space and proposed an efficient projection method.

To accelerate the computation, we proposed the following method in [6]:

1. **Plane-Based Volume Intersection method** : the 3D voxel space is partitioned into a group of parallel planes (Figure 4(b)) and the cross-section of the 3D object volume on each plane is reconstructed (Figure 5):
   (1) Project the object silhouette observed by each camera onto a common base plane (Figure 5 left, ①).
   (2) Project each base plane silhouette onto the other parallel planes (Figure 5 left, ②).

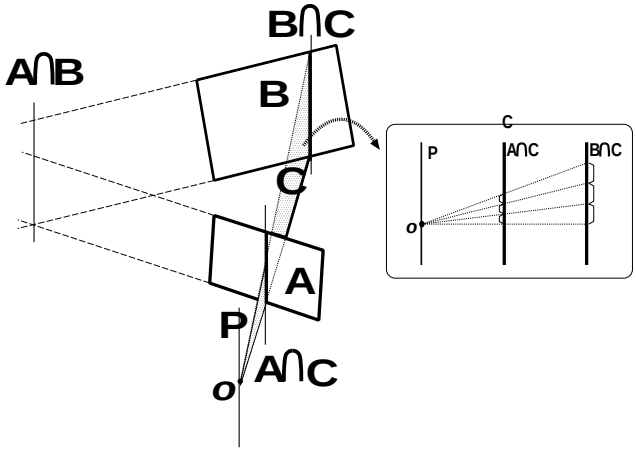


Fig. 6. Linearized PPPP algorithm

(3) Compute 2D intersection of all silhouettes projected on each plane (Figure 5 right).

2. Linearized Plane-to-Plane Perspective Projection(LPPPP) algorithm : Figure 6 illustrates the algorithm. Suppose we want to map a silhouette on plane A onto B. $A \bigcap B$ denotes the intersection line between the planes and O the center of 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 Figure 6), the image data on the intersection line $A \bigcap C$ is projected onto $B \bigcap C$. As shown in the right part of Figure 6, this linear (i.e. line-based) perspective projection can be computed just by scaling operation. By rotation plane C around line P, we can map full 2D image on A onto B.

Note that when A and B are parallel (i.e. in the case (2) above), the projection between them is simplified to 2D isotropic scaling and translation. That is, once we have a base plane silhouette, we can efficiently project it onto the other slice planes just by 2D binary image processing: *Plane-Wise Plane-to-Plane Perspective Projection.*

3. Parallel Pipeline Processing on a PC cluster system : Figure 7 illustrates the processing flow of the parallel pipeline 3D 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 (Figure 7 top row).

(2) *Silhouette Extraction :* Each PC with a camera extracts an object silhouette from the video frame (SIP in Figure 7).

(3) *Projection to the Base-Plane :* Each PC with a camera projects the silhouette onto the common base-plane in the 3D space (BPP in Figure 7).

(4) *Base-Plane Silhouette Duplication :* All base-plane silhouettes are duplicated across all PCs over the network so that each PC has the full set of all base-plane silhouettes (Communication in Figure 7). Note that the data are distributed over all PCs (i.e. with and without cameras) in the system.

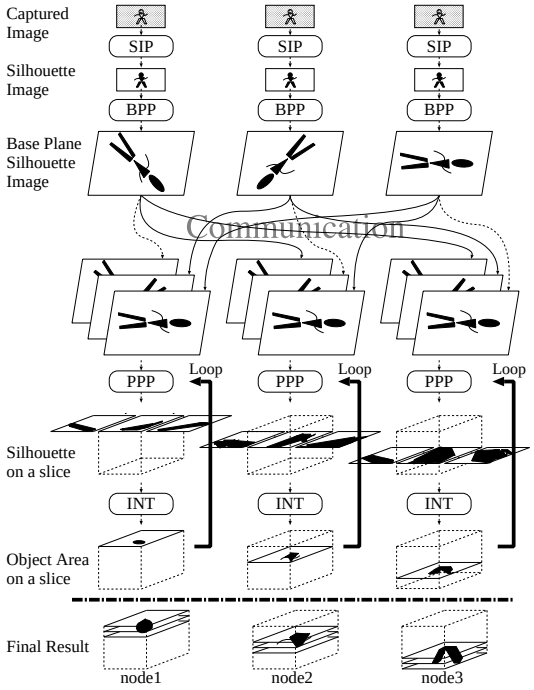


Fig. 7. Processing flow of the parallel pipelined 3D shape reconstruction.

(5) *Object Cross Section Computation :* Each PC computes object cross sections on specified parallel planes in parallel (PPP and INT in Figure 7).

In addition to the above parallel processing, we introduced a pipeline processing on each PC: 5 stages (corresponding to the 5 steps above) for PC with a camera and 2 stages (the steps 4 and 5) for PC without a camera. In this pipeline processing, each stage is implemented as a concurrent process and processes data independently of the other stages.

3.1.2 Performance Evaluation

To analyze in detail the performance of the real-time 3D volume reconstruction system, we used 6 digital IEEE1394 cameras placed at the ceiling for capturing multi-view video data of a dancing human (like in Figure 1). A hardware synchronization module was employed to capture synchronized multi-view object images. The size of input image is 640×480 pixels and we measured the time taken to reconstruct one 3D shape in the voxel size of 2cm$\times$ 2cm$\times$ 2cm contained in a space of 2m $\times$ 2m $\times$ 2m.

In the first experiment, we analyzed processing time spent at each pipeline stage by using 6 - 10 PCs for computation. Figure 8 shows the average computation time [1] spent at each pipeline stage. Note that the image capturing

[1] For each stage, we calculated the average computation time of 100 video frames

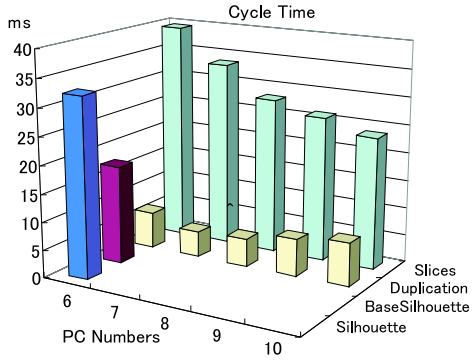


Fig. 8. Average computation time for each pipeline stage.

stage is not taken into account in this experiment.

From this figure, we can observe the followings:

- The computation time for the *Projection to the Base-Plane* stage is about 18ms, which proves the LPPPP algorithm is very efficient.
- With 6 PCs (i.e. with no PCs without cameras), the bottleneck for real-time 3D shape reconstruction rests at the *Object Cross Section Computation* stage, since this stage consumes the longest computation time (i.e. about 40ms).
- By increasing the number of PCs, the time taken at that most expensive stage decreases considerably (i.e. well below 30ms). This proves the proposed parallelization method is effective. Note that while the time spent at the Base-Plane Silhouette Duplication stage increases as the number of PCs is increased, it stays well below 30ms.
- With more than 8 PCs, we can realize real-time (video-rate) 3D 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. Figure 9 shows the throughput [2] to reconstruct one 3D shape. We observe that while the throughput is improved by increasing PCs, it saturates at a constant value in all cases: $80 \sim 90$ms.

Since as was proved in the first experiment, the throughput of the pipelined computation itself is about 30ms, the elongated overall throughput is due to the speed of *Image Capture* stage. That is, although a camera itself can capture images at a rate of 30 fps individually, the synchronization reduces its frame

on each PC. The time shown in the graph is the average time for all PCs.

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

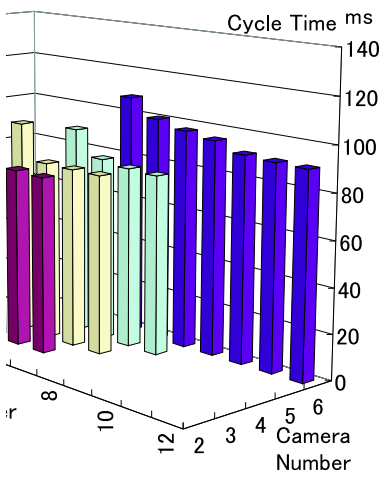


Fig. 9. Computation time for reconstructing one 3D shape.

rate down into half. This is because the external trigger for synchronization is not synchronized with the internal hardware cycle of the camera.

In summary, the experiments proved the effectiveness of the proposed real-time 3D shape reconstruction system: the plane-based volume intersection method, the LPPPP algorithm, and the parallel pipeline implementation. Moreover, the proposed parallel processing method is flexible enough to scale up the system by increasing numbers of cameras and PCs. To realize video-rate 3D shape reconstruction, we have to develop sophisticated video capturing hardwares.

3.2 Real-Time 3D Shape Reconstruction with Camera Actions

While not noted explicitly so far, the plane-based volume intersection method we proposed can exhibit a serious problem: when the optical axis of a camera is nearly parallel to the base plane, the size of the projected silhouette becomes very huge, which damages the computational efficiency. In the worst case, i.e. when the optical axis becomes parallel to the base plane, the projection cannot be computed and the method breaks down.

In the case of static camera arrangements, it is possible to select the base plane so that the worst case can be avoided. But the question of which base plane is optimal for computation remains open. In the case of dynamic camera arrangements, we have to extend the method to keep the computational efficiency as well as avoid the worst case.

Here we first analyze how the computational cost changes depending on the base plane selection and then propose an augmented plane-based volume intersection method for active cameras.

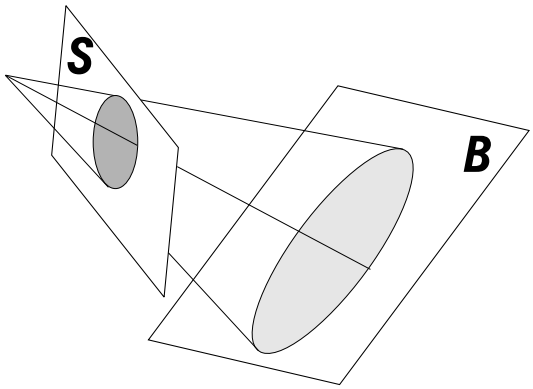


Fig. 10. Projection from an input image screen to the base plane

3.2.1 Optimal Base Plane Selection

Suppose an observed object silhouette on an input image screen S is represented as a circle of radius r, which is projected onto the base plane B as an ellipse (Figure 10). Let θ, f, l denote the dihedral angle between B and S, focal length, and the distance from the projection center to B respectively. The area size s of the projected silhouette becomes:

$$s = \pi r^2 \cdot R(\theta, f, l), \tag{1}$$
$$where \quad R(\theta, f, l) = \frac{l^2}{f^2 \cos\theta}.$$

Thus, the projected silhouette is expanded in the ratio of $R(\theta, f, l)$. Figure 11 shows the graph of R for $\theta \in [0, \pi/2)$. It is clear that R is monotonically increasing with θ and diverges to infinity at $\theta = \pi/2$, which corresponds to the worst case.

In the case of a static arrangement of n cameras, let $\{f_1, f_2, \ldots, f_n\}$, $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ and $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ denote the focal lengths, view directions and positions of the cameras respectively. The base plane can be represented by its normal vector $\mathbf{D} = (D_x, D_y, D_z)^t$ from the origin of the world coordinate system. Given the base plane, θ_i and l_i for each camera can be calculated from $\mathbf{v}_i$, $\mathbf{p}_i$ and $\mathbf{D}$. Let $\{a_1, a_2, \ldots, a_n\}$ denote the area sizes of object silhouettes observed by the cameras. From equation (1), the area size of each projected silhouette becomes:

$$s_i = a_i \cdot R_i = a_i \cdot \frac{l_i^2}{f_i^2 \cos\theta_i} \tag{2}$$

So the optimal selection of the base plane can be achieved by solving

$$\mathbf{D} = \mathrm{argmin} \sum_{i=1}^{n} s_i. \tag{3}$$

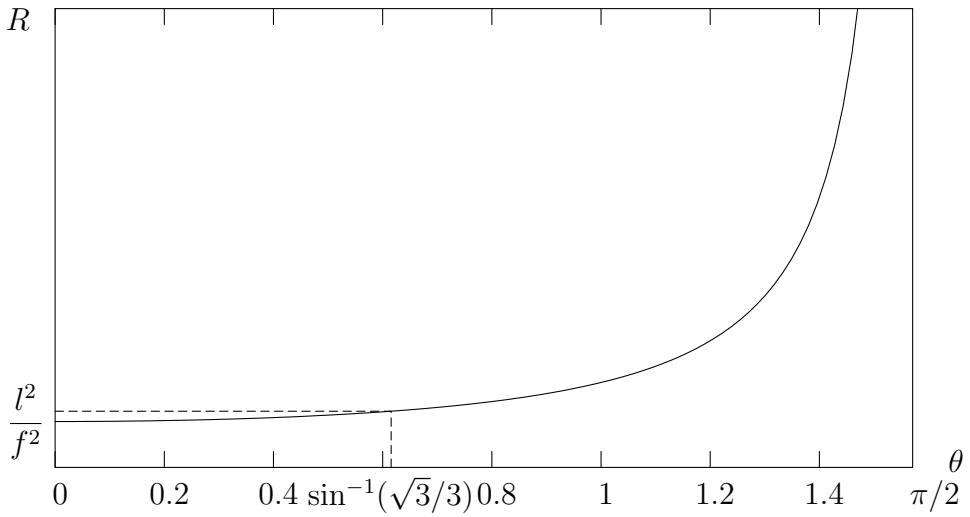


Fig. 11. Size ratio between the projected silhouette and the observed silhouette

Note firstly that it is hard to solve this problem analytically, because we cannot represent $\{a_1, a_2, \ldots, a_n\}$ in analytical forms. Moreover, since $\{a_1, a_2, \ldots, a_n\}$ change dynamically depending on the object action, the optimal base plane selection should be done frame by frame, which introduces a large overhead. Note also that if we changed the base plane frame by frame, we would have to map each reconstructed 3D volume to a common coordinate system since the 3D volume is represented in such coordinate system that is defined by the base plane. This coordinate transformation would also introduce a large overhead.

3.2.2 3-Base-Plane Volume Intersection Method

Considering the above discussions, we augment the plane-based volume intersection as follows:

(1) Let $\mathcal{P}(t)$ denote the rectangular parallelepiped encasing the 3D object at t, whose three faces are parallel to $x-y, y-z,$ and$z-x$ planes of the world coordinate system. Let $\mathbf{P}(t) = (x_{\mathbf{P}}(t), y_{\mathbf{P}}(t), z_{\mathbf{P}}(t))^{\mathrm{t}}$ denote the centroid of $\mathcal{P}(t)$ and $l_x(t), l_y(t), l_z(t)$ the side lengths of $\mathcal{P}(t)$. Note that since we do not know the exact 3D object shape at t, the bounding box $\mathcal{P}(t)$ should be estimated based on the object motion history before t.
(2) Define as the 3 base planes at t those planes whose normal vectors are defined by $\mathbf{D}_x(t) = (x_{\mathbf{P}}(t), 0, 0)^{\mathrm{t}}$, $\mathbf{D}_y(t) = (0, y_{\mathbf{P}}(t), 0)^{\mathrm{t}}$, and $\mathbf{D}_z(t) = (0, 0, z_{\mathbf{P}}(t))^{\mathrm{t}}$. Thus we have 3 mutually orthogonal base planes intersecting at $\mathbf{P}(t) = (x_{\mathbf{P}}(t), y_{\mathbf{P}}(t), z_{\mathbf{P}}(t))^{\mathrm{t}}$.
(3) For each camera, calculate the angles between the camera direction and each of $\{\mathbf{D}_x(t), \mathbf{D}_y(t), \mathbf{D}_z(t)\}$. Let $\{\ \{\theta_{1_{\mathbf{D}_x(t)}}, \theta_{1_{\mathbf{D}_y(t)}}, \theta_{1_{\mathbf{D}_z(t)}}\}\ , \{\theta_{2_{\mathbf{D}_x(t)}}, \theta_{2_{\mathbf{D}_y(t)}}, \theta_{2_{\mathbf{D}_z(t)}}\}$

, ... , $\{\theta_{n_{\mathbf{D}_x(t)}}, \theta_{n_{\mathbf{D}_y(t)}}, \theta_{n_{\mathbf{D}_z(t)}}\}$ } denote such angles.

(4) Select the base plane $\mathbf{B}_i(t)$ for ith camera at t as:

$$\mathbf{B}_i(t) = \begin{cases} \mathbf{D}_x(t), & \text{if } \theta_{i_{\mathbf{D}_x(t)}} = \min(\theta_{i_{\mathbf{D}_x(t)}}, \theta_{i_{\mathbf{D}_y(t)}}, \theta_{i_{\mathbf{D}_z(t)}}) \\ \mathbf{D}_y(t), & \text{if } \theta_{i_{\mathbf{D}_y(t)}} = \min(\theta_{i_{\mathbf{D}_x(t)}}, \theta_{i_{\mathbf{D}_y(t)}}, \theta_{i_{\mathbf{D}_z(t)}}) \\ \mathbf{D}_z(t), & \text{if } \theta_{i_{\mathbf{D}_z(t)}} = \min(\theta_{i_{\mathbf{D}_x(t)}}, \theta_{i_{\mathbf{D}_y(t)}}, \theta_{i_{\mathbf{D}_z(t)}}) \end{cases} \tag{4}$$

(5) By the above base plane selection, a set of cameras are partitioned into 3 groups: cameras in each group share the same base plane.

(6) Suppose $\mathbf{D}_x(t)$ is selected as the base plane for ith camera. Then, compute the silhouette of $\mathcal{P}(t)$ on the base plane by projecting the vertices of $\mathcal{P}(t)$ onto $\mathbf{D}_x(t)$. Allocate a 2D array for storing the base plane object silhouette so that the array encases the projected silhouette of $\mathcal{P}(t)$.

(7) Apply the *Linearized Plane-to-Plane Perspective Projection* to generate the base plane object silhouette for ith camera.

(8) To compute object silhouettes on the other parallel planes, first decompose the volume of $\mathcal{P}(t)$ into the set of slices parallel to the base plane $\mathbf{D}_x(t)$. Let denote such slices as

$$\mathbb{V}_x(t) = \{\mathbf{S}_{\mathbf{D}_{x1}}(t), \mathbf{S}_{\mathbf{D}_{x2}}(t), \ldots, \mathbf{S}_{\mathbf{D}_{xn_x}}(t)\},$$

where n_x stands for the number of the slices, which can be determined by the required spatial resolution. Note that the size of the arrays to store silhouette data on the slices is determined by $\mathcal{P}(t)$: that is, each array stands for a cross section of $\mathcal{P}(t)$.

(9) Project the base object silhouettes on $\mathbf{D}_x(t)$ onto each slice in $\mathbb{V}_x(t)$ by the *Plane-wise Plane-to-Plane Perspective Projection*. Note that since each slice is bounded, it is enough to compute a projected silhouette in such clipped area.

(10) Apply the same processing for those cameras whose base planes coincide with $\mathbf{D}_x(t)$ and intersect projected silhouettes on each slice in $\mathbb{V}_x(t)$. Then, we have a partial 3D object shape computed based on one of the 3 base planes, i.e. $\mathbf{D}_x(t)$.

(11) By conducting the same processing for $\mathbf{D}_y(t)$ and $\mathbf{D}_z(t)$, we have three partial 3D object shapes, which then are intersected to generate the complete 3D object shape at t.

With this 3-base-plane volume intersection method, since $\mathbf{D}_x(t) \perp \mathbf{D}_y(t) \perp \mathbf{D}_z(t)$,

$$\min(\theta_{i_{\mathbf{D}_x(t)}}, \theta_{i_{\mathbf{D}_y(t)}}, \theta_{i_{\mathbf{D}_z(t)}})) \leq \sin^{-1}(\sqrt{3}/3). \tag{5}$$

This means that from equation (1), the area size of a projected object silhouette is bounded by $\frac{\sqrt{6}}{2} \cdot \frac{l_i(t)^2}{f_i^2} a_i(t)$(Figure 11). That is, by this extension,

	YZ	XZ	XY
Camera 1	14.15	51.01	35.43
Camera 2	57.60	24.61	19.70
Camera 3	43.89	39.86	20.78
Camera 4	12.84	54.63	32.31
Camera 5	13.95	53.24	33.20
Camera 6	27.35	62.38	3.51
Camera 7	35.82	54.06	2.51
Camera 8	69.22	20.55	2.91
Camera 9	84.49	5.45	0.76

Table 1
View directions of the cameras: Angles with YZ, XZ, and XY planes[DEG]

not only the worst case can be avoided, but also we can estimate the upper bound of the computational cost, which is very important for the design of the real-time active 3D shape reconstruction.

Note also that

- For each active camera, its corresponding base plane is changed dynamically according to the selection rule in equation (4).
- Such dynamic selection of the base plane does not introduce any significant overhead since the directions of the 3 base planes are fixed.
- By introducing the (estimated) bounding box of the object and making the 3 base planes parallel to the faces of the box, we can confine the sizes of both the base plane silhouette images and the silhouette images on the parallel slices, as well as their locations. This greatly facilitates the computation required in the plane-based volume intersection method.
- While their locations are changed dynamically, the 3 base planes are parallel to the world coordinate system, we can easily trace and compare a temporal sequence of reconstructed 3D volumes.

3.2.3 Performance Evaluation

To evaluate the performance of the 3-base-plane volume intersection method, we compared it with the original plane-based volume intersection method. Figure 12 shows the camera layout and Table 1 the angles between the view directions of the cameras and YZ, XZ, and XY planes. While the cameras in this experiment are fixed, a human changed his poses as illustrated in Figure 13.

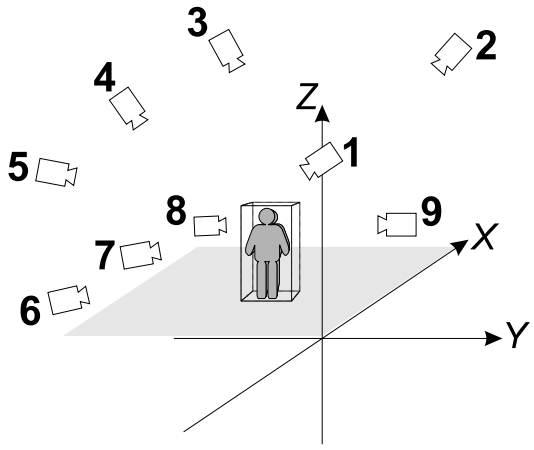
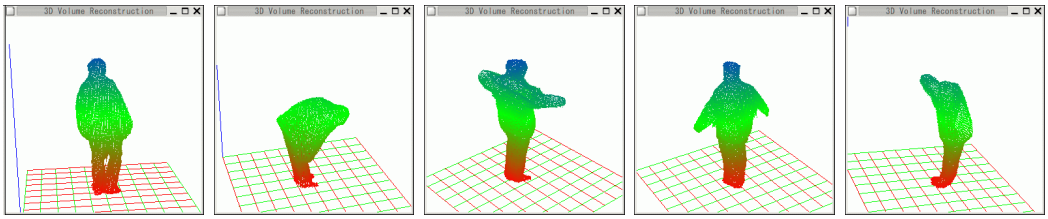


Fig. 12. Camera Layout



pose1 pose2 pose3 pose4 pose5

Fig. 13. 5 different poses

Then, we measured the computation time to project a captured object silhouette onto the base plane by a PC controlling each camera: i.e. the time spent at the *Projection to the Base-Plane* stage. Figure 14 compares the computation times by the original plane-based and 3-base-plane volume intersection methods. From this result, we observe

- In all poses, since the view directions of cameras 6, 7, 8, and 9 are nearly parallel to the XY plane, the original base-plane projection method broke down when their object silhouettes were projected onto the XY plane. Note that in the experiments described in section 3.1.2, all cameras were placed at the ceiling and looking downward as illustrated in Figure 1, by which the break down was avoided.
- For cameras 1, 2, 3, 4, and 5, the computation times by the original base-plane projection method varied depending the poses of a human and the view angles of the cameras. This variation incurs a serious computational problem in realizing real-time processing; the pipeline is clogged depending on human actions.
- The 3-base-plane method never broke down and was very efficient. Moreover, while its computation time varied depending on the human poses and the view angles of the cameras, its variation was small enough. Note that since we used 1cm voxel resolution for the experiments to make clear the computational efficiency, the net computation time exceeded 30msec; we

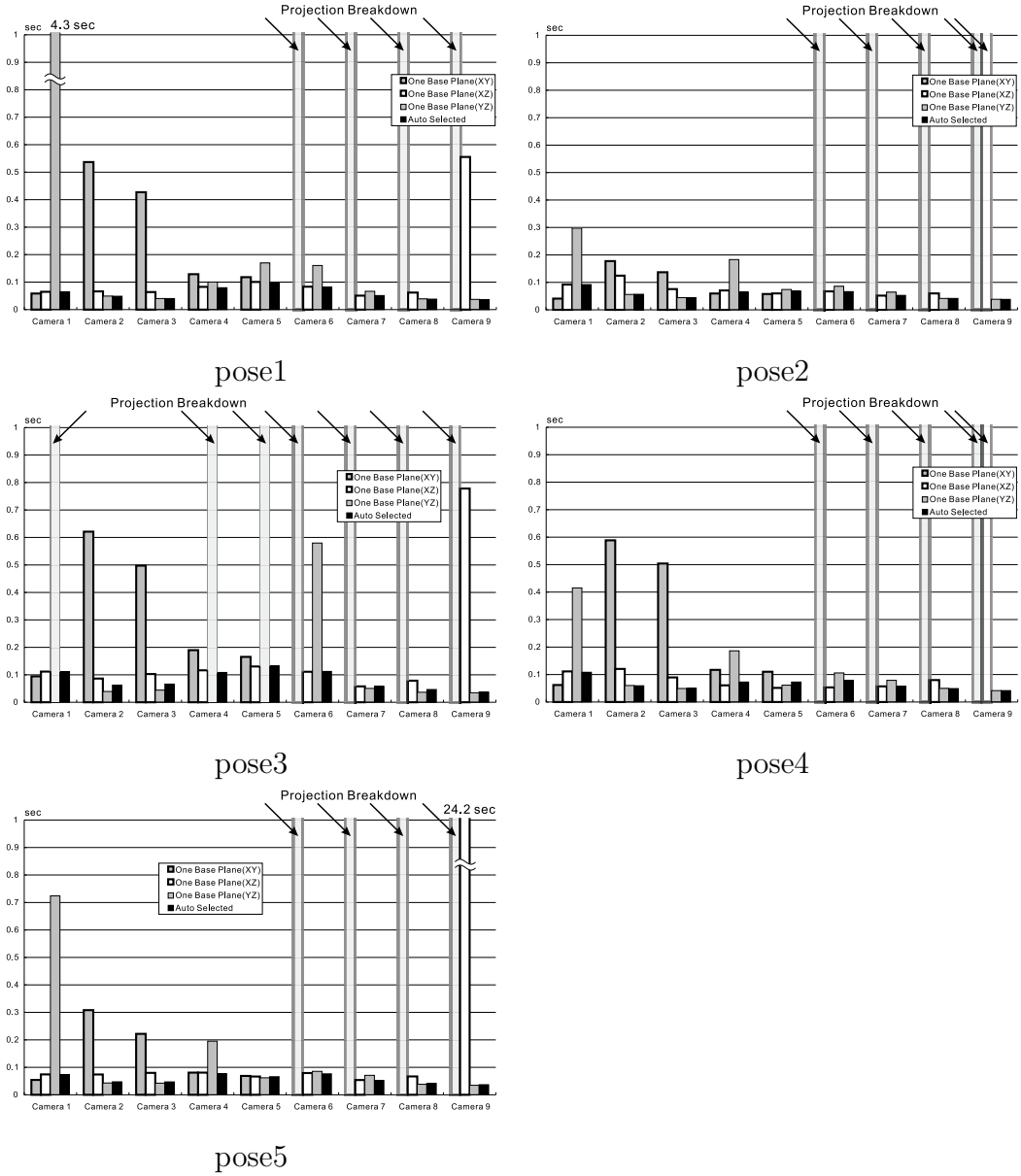


Fig. 14. Computation times of the base-plane projection by the original plane-based and 3-base-plane volume intersection methods.

used 2cm voxel resolution in the experiments in section 3.1.2.

In short, the 3-base-plane volume intersection method is robust and stable against both dynamic object pose changes and camera layout variations as well as efficient enough to realize real-time volume reconstruction.

Currently we are developing a system for real-time active 3D action reconstruction, where the algorithm described in section 3.2.2 is integrated with the real-time active object tracking system described in [26]. The parallel processing method and performance evaluation of the system will be reported in a separate paper.

4 Dynamic 3D Shape from Multi-View Images Using Deformable Mesh Model

As is well known, the volume intersection method is stable, but cannot reconstruct accurate 3D object shape; its output represents just the visual hull of the object and concave portions of the object cannot be reconstructed. In contrast, the stereo method can reconstruct any kind of visible surfaces, but it is difficult to obtain dense and accurate stereo matching. Moreover, since the stereo analysis merely generates a 2.5D shape, multi-view stereos should be integrated to reconstruct the full 3D object shape. Thus, volume intersection and stereo analysis have their own advantages and disadvantages.

To accomplish more stability and accuracy, recent methods propose frameworks to combine multiple reconstruction cues. For example, Fua[27] represented object shape by a 2.5D triangular mesh model and deformed it based on photometric stereo and silhouette constraint. Cross[28] carved a visual hull, a set of voxels given by silhouettes, using photometric properties.

For *dynamic* 3D shape reconstruction, on the other hand, a naive method would be:
Step 1. reconstruct 3D shape for each frame,
Step 2. estimate 3D motion by establishing correspondences between a pair of 3D shapes at frames t and $t+1$.
In such a heterogeneous processing scheme, it is hard to coordinate different computational algorithms conducted in different steps. We believe that a homogeneous processing scheme, i.e., simultaneous recovery of shape and motion is required to attain effective dynamic 3D shape reconstruction. In fact, Vedula[29] showed an algorithm to recover 3D shapes represented by voxels in two consecutive frames as well as the per-voxel-correspondence between them simultaneously.

In this section, we propose a framework for dynamic 3D shape reconstruction from multi-view images using a deformable mesh model[3][30]. With this method, we can obtain 3D shape and 3D motion of the object simultaneously by a single computational scheme.

We represent the shape by a surface mesh model and the motion by translations of its vertices, i.e., deformation. Thus the global and local topological structure of the mesh are preserved from frame to frame. This helps us to analyze the object motion, to compress the 3D data[31], and so on.

Our model deforms its shape so as to satisfy several constraints: 1) "photo-consistency" constraint, 2) silhouette constraint, 3) smoothness constraint, 4) 3D motion flow constraint, and 5) inertia constraint. We show this constraint-based deformation provides a computational framework to integrate several

reconstruction cues such as surface texture, silhouette, and motion flow observed in multi-view images.

In our 3D deformable mesh model, we introduce two types of deformation: intra-frame deformation and inter-frame deformation. In the intra-frame deformation, our model uses the visual hull, a result of the volume intersection, as initial shape and changes its shape so as to satisfy constraints 1), 2) and 3) described above. Volume intersection estimates a rough but stable shape using geometric information, i.e., silhouettes, and the deformable model refines this shape using photometric information. In the inter-frame deformation, our model changes its shape frame by frame, under all constraints 1), ..., 5). This deforming process enables us to obtain the *topologically consistent* shape in the next frame and *per-vertex*-correspondence information, i.e., motion simultaneously.

4.1 Deformable 3D Mesh Model For Intra-frame Deformation

With our deformable mesh model, we can employ both geometric and photometric constraints of the object surface in the reconstruction of its shape; these constraints are not used in volume intersection, stereo, or space carving methods[25].

Our intra-frame deformation algorithm consists of the following steps:

step 1 Convert the visual hull of the object: the voxel representation is transformed into a triangle mesh model by the discrete marching cubes algorithm[15], and this is used as an initial shape.

step 2 Deform the model iteratively:

- **step 2.1** Compute force acting on each vertex.
- **step 2.2** Move each vertex according to the force.
- **step 2.3** Terminate if all vertex motions are small enough. Otherwise go back to 2.1 .

This shape deformation is similar to the 2D approach used in active contour models or "Snakes"[32]; to realize it, we can use either energy function based or force based methods. As described above, we employed a force based method. This is firstly, from a computational point of view, because we have too many vertices to solve energy function (for example, the mesh model shown in Figure 2 has about 12,000 vertices), and secondly, from an analytical point of view, because one of the constraints used to control the deformation cannot be represented as an analytical energy function (see below).

We employed the following three types of constraints to control the intra-frame deformation:

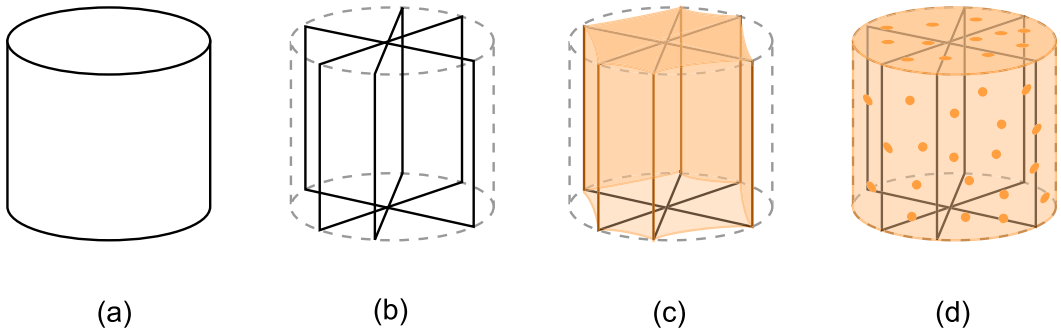


Fig. 15. Frame and skin model

(1) **Photometric constraint**: A patch in the mesh model should be placed so that its texture, which is computed by projecting the patch onto a captured image, is consistent irrespectively of the image onto which it is projected.
(2) **Silhouette constraint**: When the mesh model is projected onto an image plane, its 2D silhouette should coincide with the observed object silhouette on that image plane.
(3) **Smoothness constraint**: The 3D mesh should be locally smooth and should not intersect with itself.

These constraints define a *frame and skin model* to represent 3D object shape:

- Suppose we want to model the object in Figure 15 (a).
- First, the silhouette constraint defines a set of frames for the object (Figure 15 (b)).
- Then the smoothness constraint defines a rubber sheet skin to cover the frames (Figure 15 (c)).
- Finally, the photometric constraint defines supporting points on the skin that have prominent textures (Figure 15 (d)).

In what follows, we describe the forces generated at each vertex to satisfy the constraints.

4.2 Forces at each Vertex

We denote a vertex, its 3D position, and the set of cameras which can observe this vertex by v, $\boldsymbol{q}_v$, and C_v respectively. For example, $C_v = \{\mathsf{CAM}_2, \mathsf{CAM}_3\}$ in Figure 16.

We introduce the following three forces at v to move its position so that the above mentioned three constraints are be satisfied:

External Force: $\boldsymbol{F}_e(v)$

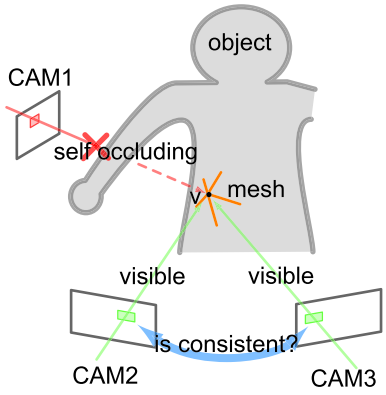


Fig. 16. Photometric consistency and visibility

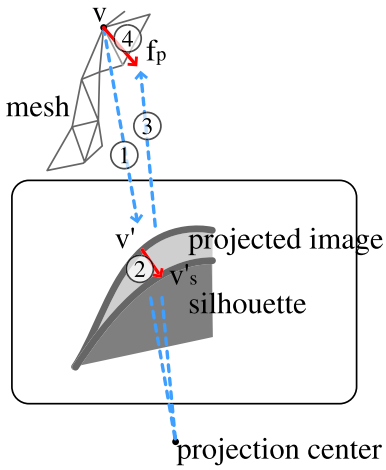


Fig. 17. Silhouette preserving force

First, we define an external force $\boldsymbol{F}_e(v)$ to deform the mesh to satisfy the photometric constraint.

$$\boldsymbol{F}_e(v) \equiv \nabla E_e(\boldsymbol{q}_v), \tag{6}$$

where $E_e(\boldsymbol{q}_v)$ denotes the correlation of textures to be mapped around v (Figure 16) :

$$E_e(\boldsymbol{q}_v) \equiv \frac{1}{N(C_v)} \sum_{c \in C_v} \left\| p_{v,c} - \overline{p_v} \right\|^2, \tag{7}$$

where c denotes a camera in C_v, $N(C_v)$ the number of cameras in C_v, $p_{v,c}$ the texture corresponding to v on the image captured by c, and $\overline{p_v}$ the average of the $p_{v,c}$. $\boldsymbol{F}_e(v)$ moves v so that its corresponding image textures observed by the cameras in C_v become mutually consistent.

Internal Force: $\boldsymbol{F}_i(v)$
Since $\boldsymbol{F}_e(v)$ may destroy the smoothness of the mesh or lead to self-intersection, we introduce an internal force $\boldsymbol{F}_i(v)$ at v:

$$\boldsymbol{F}_i(v) \equiv \frac{\sum_j^n \boldsymbol{q}_{v_j} - \boldsymbol{q}_v}{n}, \tag{8}$$

where $\boldsymbol{q}_{v_j}$ denotes the neighboring vertices of v and n the number of neighbors. $\boldsymbol{F}_i(v)$ act as tension between vertices and keeps them locally smooth.

Note that the utilities of this internal force is twofold: (1) make the mesh shrink and (2) make the mesh smooth. We need (1) in the intra-frame deformation since it starts with the visual hull which encases the real object shape. (2), on the other hand, stands for a popular smoothness heuristic employed in many vision algorithms such as the regularization and active contour models. The smoothing force works to prevent self-intersection since a self-intersecting surface includes protrusions and dents, which will be smoothed out before causing self-intersection.

For the inter-frame deformation, on the other hand, we redefine $\boldsymbol{F}_i(v)$ as a combination of attraction and repulsion between linked vertices, and add a *diffusion* step just after step 2.1 (see below). This is because we do not need (1) described above in the inter-frame deformation process.

Silhouette Preserving Force: $\boldsymbol{F}_s(v)$
To satisfy the silhouette constraint described above, we introduce a silhouette preserving force $\boldsymbol{F}_s(v)$. This is the most distinguishing characteristic of our deformable model and involves a nonlinear selection operation based on the global shape of the mesh, which cannot be analytically represented by an energy function.

Figure 17 explains how this force at v is computed, where $S_{o,c}$ denotes the object silhouette observed by camera c, $S_{m,c}$ the 2D projection of the 3D mesh onto the image plane of camera c, and v' the 2D projection of v onto the image plane of camera c.

(1) For each c in C_v, compute the partial silhouette preserving force $\boldsymbol{f}_s(v,c)$ by the following method.
(2) If
  (a) v' is located outside of $S_{o,c}$ or
  (b) v' is located inside $S_{o,c}$ and on the contour of $S_{m,c}$,
  then compute the shortest 2D vector from v' to $S_{o,c}$ (Figure 17 ②) and assign its corresponding 3D vector to $\boldsymbol{f}_s(v,c)$ (Figure 17 ④).
(3) Otherwise, $\boldsymbol{f}_s(v,c) = 0$.

The overall silhouette preserving force at v is computed by summing up $\boldsymbol{f}_s(v,c)$:

$$\boldsymbol{F}_s(v) \equiv \sum_{c \in C_v} \boldsymbol{f}_s(v,c). \tag{9}$$

Note that $\boldsymbol{F}_s(v)$ acts only on those vertices that are located around the object contour generator[28], which is defined based on the global 3D shape of the object as well as the locations of cameras' image planes.

Overall Vertex Force: $\boldsymbol{F}(v)$
Finally we define a vertex force $\boldsymbol{F}(v)$ with coefficients α, β, γ as follows:

$$\boldsymbol{F}(v) \equiv \alpha \boldsymbol{F}_i(v) + \beta \boldsymbol{F}_e(v) + \gamma \boldsymbol{F}_s(v), \tag{10}$$

where coefficients are constants and examples for typical values will be given in following experiments section. $\boldsymbol{F}_e(v)$ and $\boldsymbol{F}_s(v)$ work to reconstruct the accurate object shape and $\boldsymbol{F}_i(v)$ to smooth and interpolate the shape. Note that there may be some vertices where $C_v = \{\}$ and hence $\boldsymbol{F}_e(v) = \boldsymbol{F}_s(v) = 0$.

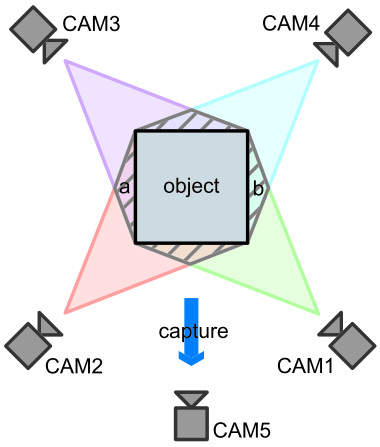
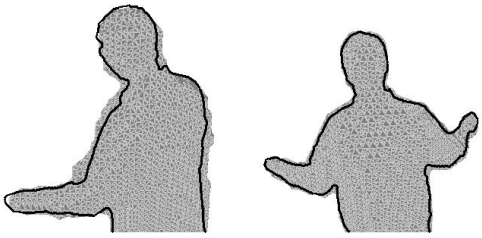


Fig. 18. Camera arrangement



(a) CAM5 (b) CAM1

Fig. 19. Initial shape. (a) is viewed from CAM_5 in Figure 18, (b) from CAM_1

4.3 Performance Evaluation

Real Images Figure 18 illustrates the camera arrangement for the experiments, where we use $\mathsf{CAM}_1, \ldots, \mathsf{CAM}_4$ for shape reconstruction and CAM_5 for performance evaluation. That is, we compare the 2D silhouette of the reconstructed shape viewed from the position of CAM_5 with the actually observed by CAM_5. Note that captured images are synchronized and blur-free.

Figure 19 shows the initial object shape computed by the volume intersection using the images captured by $\mathsf{CAM}_1, \ldots, \mathsf{CAM}_4$, i.e., the visual hull of the object. The shaded regions of (a) and (b) show the projection of the initial shape, that is, $S_{m,5}$ and $S_{m,1}$, respectively. Bold lines in the figures highlight the contours of $S_{o,5}$ and $S_{o,1}$. We can observe some differences between $S_{o,5}$ and $S_{m,5}$, but not between $S_{o,1}$ and $S_{m,1}$. This is because the image captured by CAM_5 is not used for the reconstruction.

In the experiments, we evaluated our algorithm with the following configurations : (a) $\boldsymbol{F}(v) = \boldsymbol{F}_i(v)$, (b) $\boldsymbol{F}(v) = 0.5\boldsymbol{F}_i(v) + 0.5\boldsymbol{F}_s(v)$, (c) $\boldsymbol{F}(v) = 0.3\boldsymbol{F}_i(v) + 0.4\boldsymbol{F}_e(v) + 0.3\boldsymbol{F}_s(v)$. The left and center columns of Figure 20 shows $S_{m,5}$ and $S_{m,1}$ for each configuration together with bold lines denoting the corresponding observed object silhouette contours $S_{o,5}$ and $S_{o,1}$. The graphs in the right column show how the average error between $S_{m,c}$ and $S_{o,c}$ $(c = 1, 5)$ changes during the iterative shape deformation. Note that the processing time of deformation is about 3 minutes for 12000 vertices and 4 cameras.

From these results we can make the following observations:

- With $\boldsymbol{F}_i(v)$ alone (Figure 20(a)), the mesh model shrinks, resulting in a large gap between its 2D silhouette on each image plane and the observed silhouette.

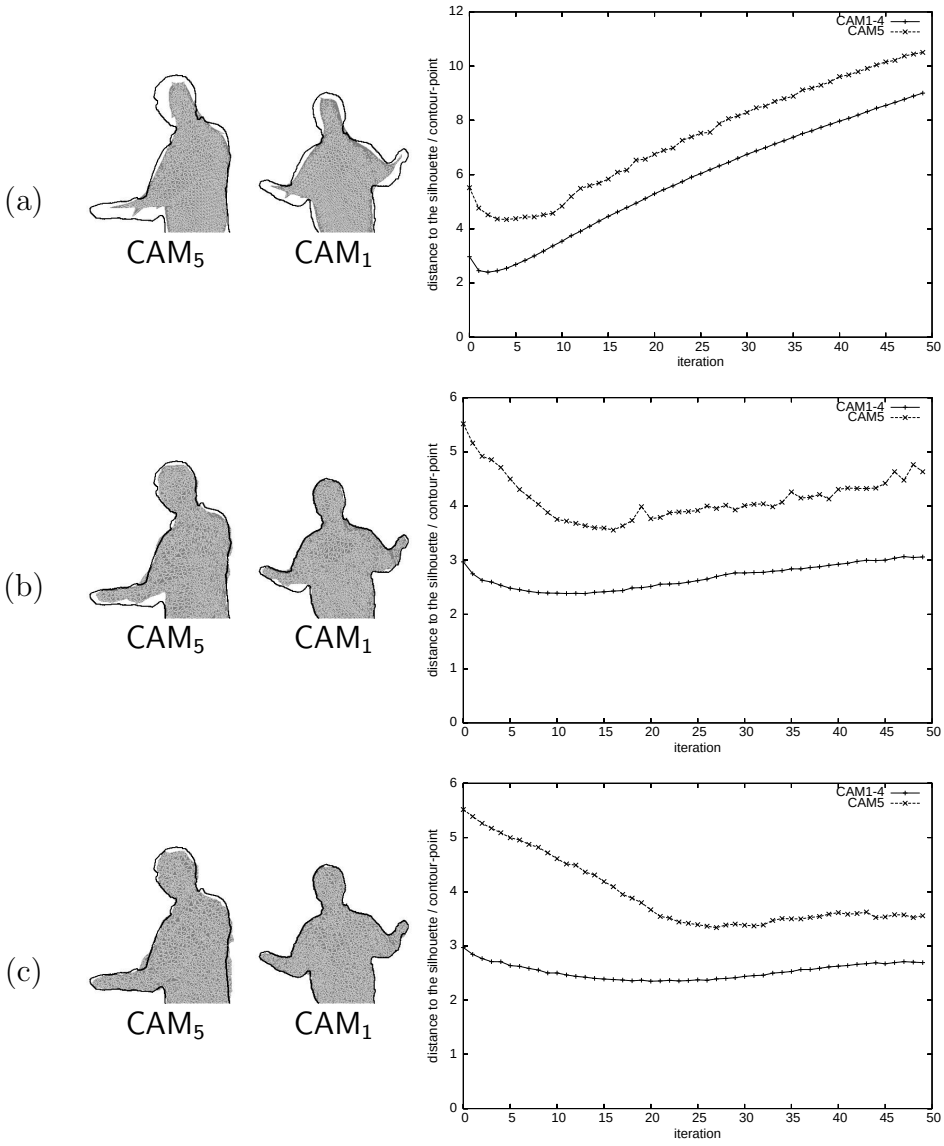
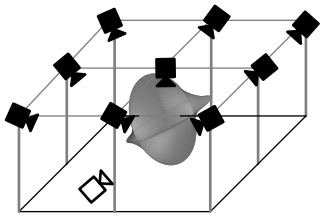


Fig. 20. Experimental results. Top: (a) $\boldsymbol{F}_i(v)$ alone ($\alpha = 1.0, \beta = 0.0, \gamma = 0.0$), Middle: (b) $\boldsymbol{F}_i(v) + \boldsymbol{F}_s(v)$ ($\alpha = 0.5, \beta = 0.0, \gamma = 0.5$) Bottom: (c) $\boldsymbol{F}_i(v) + \boldsymbol{F}_e(v) + \boldsymbol{F}_s(v)$ ($\alpha = 0.3, \beta = 0.4, \gamma = 0.3$)

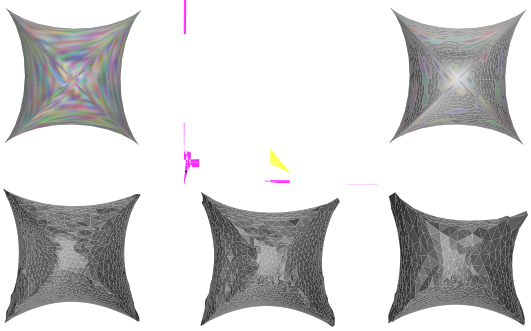
- With $\boldsymbol{F}_i(v)$ and $\boldsymbol{F}_s(v)$, while $S_{m,c}, c = \{1 \dots 4\}$ match well with $S_{o,c}$, $S_{m,5}$, whose corresponding image is not used for the reconstruction, does not deform well (Figure 20(b)).
- With $\boldsymbol{F}_i(v)$, $\boldsymbol{F}_e(v)$, and $\boldsymbol{F}_s(v)$, $S_{m,5}$ matches well with $S_{o,5}$ (Figure 20(c)). This shows the effectiveness of $\boldsymbol{F}_e(v)$.

Note that the values of the coefficients α, β, and γ are given a priori and fixed throughout the iteration.



(a) Camera arrangement

Fig. 21. Camera arra



(a) 100 % (b) 50 % (c) 10 %

Fig. 22. Evaluating the effectiveness of the deformable mesh model. upper: super quadrics ($n = 3.0$, $e = 1.0$) with different degrees of surface textures: k % means k % of the surface area is covered with texture. lower: reconstructed object shapes.

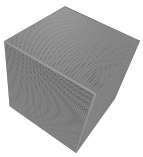
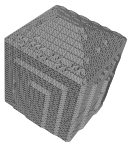
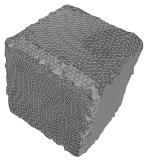
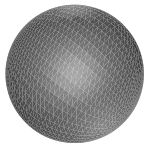
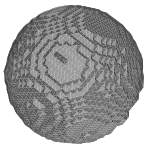
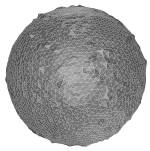
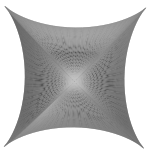
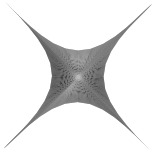
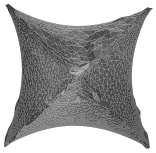
(n, e)	Synthesized object	Visual hull	Deformable mesh model
$(0.0, 0.0)$			
$(1.0, 1.0)$			
$(3.0, 1.0)$			
$(5.0, 1.0)$			

Fig. 23. Reconstruction results for various n and e (with 100% textured surfaces)

Synthesized Images To evaluate the quantitative performance of the mesh model, we conducted experiments with synthetic objects defined by super quadric functions. Super quadric functions are a set of functions defined in terms of spherical coordinates u and v:

$$\begin{aligned} x(u,v) &= a_1 \cos^n u \cos^e v \\ y(u,v) &= a_2 \cos^n u \sin^e v \\ z(u,v) &= a_3 \sin^n u \\ -\frac{\pi}{2} \le u \le \frac{\pi}{2},\ & \pi \le v \le \pi \end{aligned} \tag{11}$$

where n and e denote parameters controlling roundness/squareness, a_1, a_2, a_3 denote scale factors for x, y, z respectively. In this experiment, the values of the coefficients α, β, and γ are the same as used in the experiment with real images, that is, $\boldsymbol{F}(v) = 0.3\boldsymbol{F}_i(v) + 0.4\boldsymbol{F}_e(v) + 0.3\boldsymbol{F}_s(v)$.

Figure 21 illustrates: (a) the camera arrangement for this experiment, (b) the synthesized object, and (c) the visual hull reconstructed by the volume intersection method. We use the 9 black cameras in Figure 21(a) for the reconstruction and the white camera in Figure 21(a) for the evaluation.

Figure 22 shows reconstruction results of objects ($n = 3.0$, $e = 1.0$) having (a) 100%, (b) 50%, and (c) 10% textured surfaces. The percentage denotes the surface area of the object covered with texture. All object images are captured by the white camera in Figure 21(a).

From these results we can observe the following:

- Unlike the visual hull (Figure 21(c)), the mesh model can reconstruct the concave parts of the object (Figure 22(a) and (b)).
- The mesh model does not necessarily require a dense texture (Figure 22(a) and (b)). This is because the *skin* over *frames* (Figure 15) can interpolate the object surface between points with prominent textures.
- The reconstructed shape becomes poor when the object has little texture (Figure 22(c)).

Figure 23 shows reconstruction results of objects defined by various n and e, that is, objects having different concavities. Note that each object has 100% textured surface.

We can observe that the deformable mesh model with fixed coefficients α, β, and γ has limitations in recovering large concavities as well as large protrusions (Figure 23, bottom row). This is because large curvature on a vertex yields a large internal force $\boldsymbol{F}_i(\boldsymbol{v})$, which dominates $\boldsymbol{F}_e(\boldsymbol{v})$ even if the vertex has prominent texture.

Compared with the Space-Carving method[25], which employs photometric consistency as its main reconstruction cue, our approach additionally employs geometric continuity and a silhouette constraint. Such rich constraints make our approach more stable and accurate. Moreover, our deformable mesh model can be extended to dynamic inter-frame deformation, which will enable us to analyze dynamic object motion and realize highly efficient data compression. The next section describes this inter-frame deformation algorithm.

4.4 Dynamic Shape Recovery Using Deformable 3D Mesh Model

If a model at time t deforms its shape to satisfy the constraints at time $t+1$, we can obtain the shape at $t+1$ and the motion from t to $t+1$ simultaneously.

Clearly, the constraints used in the intra-frame deformation should be satisfied and would be sufficient if we had rich texture information all over the object surface. However, to make the dynamic deformation process more stable and reliable, we first modify the constraints used in the intra-frame deformation. Note that

- We have the mesh model at t as well as the captured multi-view images and visual hulls at t and $t+1$. Note that for the first frame, we compute the initial mesh model from the initial visual hull by the intra-frame deformation method described above.
- We redefine $\boldsymbol{F}(v)$ in equation (10) to $\boldsymbol{F}(v_{t+1})$, where v_{t+1} denotes vertex v of the mesh at $t+1$. Note that the position of v_t, $\boldsymbol{q}_{v_t}$, is fixed while that of v_{t+1}, $\boldsymbol{q}_{v_{t+1}}$, changes during the deformation process.
- C_{v_t}, the set of cameras which can observe v_t, does not change throughout the iteration. $C_{v_{t+1}}$, however, may change according to the deformation.

(1) **Photometric constraint**: The textures of each patch in the multi-view images should be consistent in the frames at both t and $t+1$.

According to this redefinition, we modify equation (6):

$$\boldsymbol{F}_e(v_{t+1}) \equiv \nabla E_e(\boldsymbol{q}_{v_{t+1}}), \tag{12}$$

where $\boldsymbol{q}_{v_{t+1}}$ denotes the position of v_{t+1}, and $E_e(\boldsymbol{q}_{v_{t+1}})$ the correlation of textures to be mapped around v (Figure 16) at both t and $t+1$, which is obtained by modifying equation (7) :

$$E_e(\boldsymbol{q}_{v_{t+1}}) \equiv \frac{\sum_{c\in C_{v_t}} \left\| p_{v_t,c} - \overline{p_{v_t,v_{t+1}}} \right\|^2 + \sum_{c\in C_{v_{t+1}}} \left\| p_{v_{t+1},c} - \overline{p_{v_t,v_{t+1}}} \right\|^2}{N(C_{v_t}) + N(C_{v_{t+1}})}, \tag{13}$$

where $\overline{p_{v_t,v_{t+1}}}$ denotes the average of $p_{v_t,c}$ and $p_{v_{t+1},c}$.

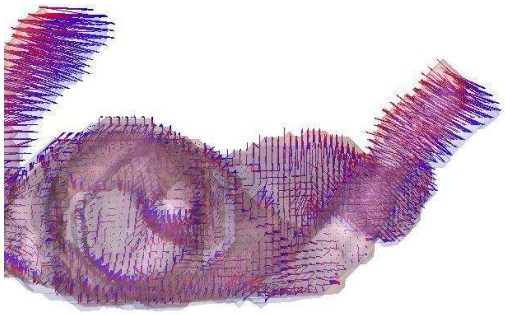


Fig. 24. Roughly estimated 3D motion flow lines from t (in blue) to $t+1$ (in red). The blue and red region denote the visual hull at t and $t+1$ respectively (same as the center column of Figure 25).

(2) **Smoothness constraint**: As described in section 4.2, the internal force defined in equation (8) plays two roles: (1) make the mesh shrink and (2) make the mesh smooth. Role (1) is a reasonable strategy for the intra-frame deformation because its initial shape is the visual hull of the object. In the inter-frame deformation, on the other hand, we should not use (1) since some parts of the mesh should 'shrink' and others 'expand' depending on the object non-rigid motion. So we replace equation (8) with the following one. This new internal force represents the smoothness constraint by connecting vertices with springs and dampers as used in non-rigid skin-modeling[33][34].

$$\boldsymbol{F}_i(v_{t+1}) \equiv \nabla E_i(\boldsymbol{q}_{v_{t+1}}), \tag{14}$$

where $E_i(\boldsymbol{q}_{v_{t+1}})$ denotes the local potential field defined by neighbor vertices of v_{t+1}:

$$E_i(\boldsymbol{q}_{v_{t+1}}) = \sum_{j}^{N} \left(\left\| \boldsymbol{q}_{v_{t+1}} - \boldsymbol{q}_{v_{t+1},j} \right\|^2 + \frac{1}{\left\| \boldsymbol{q}_{v_{t+1}} - \boldsymbol{q}_{v_{t+1},j} \right\|^2} \right) \tag{15}$$

where $\boldsymbol{q}_{v_{t+1,j}}$ denotes the position of the jth neighbor vertex of v_{t+1}, and N the number of neighbors.

(3) **Silhouette constraint**: Equation (9) should be modified to satisfy the silhouette constraint at $t+1$:

$$\boldsymbol{F}_s(v_{t+1}) \equiv \sum_{c \in C_{v_{t+1}}} \boldsymbol{f}_s(v_{t+1}, c). \tag{16}$$

We now introduce additional constraints:

(4) **3D Motion flow constraint**: A mesh vertex should drift in the direction of the motion flow of its vicinity.

(5) **Inertia constraint**: The motion of a vertex should be smooth and continuous.

Drift Force: $\boldsymbol{F}_d(v_{t+1})$
As described in section 4.1, we assume that we have multi-view silhouette images and a visual hull for each frame. With these visual hulls, i.e., sets of voxels, we can compute rough inter-frame voxel correspondences by the point-set-deformation algorithm described in [35] (Figure 24).

This algorithm gives us the voxel-wise correspondence flow from V_t, the voxel set at t, to V_{t+1}, the voxel set at $t+1$. We can represent this flow by a set of *correspondence lines*:

$$L_t = \left\{ l^i | \; i = 1, \ldots, N(V_t) \right\}, \tag{17}$$

where l^i denotes the correspondence line starting from ith voxel in V_t and $N(V_t)$ the number of voxels in V_t. While visual hulls do not represent accurate object shapes, we can use this correspondence as a rough estimate of 3D motion flow.

Once the motion flow is obtained, we define the potential field $E_d(v_{t+1})$ generated by this flow. First, let $l_{v_{t+1}}$ denote the correspondence line in L_t closest to v_{t+1}, $\boldsymbol{p}_{l_{v_{t+1}},v_{t+1}}$ the point on $l_{v_{t+1}}$ closest to v_{t+1}, and $\boldsymbol{s}_{l_{v_{t+1}}}$ the starting point of the correspondence line $l_{v_{t+1}}$. Then, we define the potential field as a function of the distance from v_t to l_{v_t} and the distance from $\boldsymbol{s}_{l_{v_t}}$ to $\boldsymbol{p}_{l_{v_t},v_t}$:

$$E_d(\boldsymbol{q}_{v_{t+1}}) \equiv \|\boldsymbol{s}_{l_{v_{t+1}}} - \boldsymbol{p}_{l_{v_{t+1}},v_{t+1}}\|^2 - \|\boldsymbol{q}_{v_{t+1}} - \boldsymbol{p}_{l_{v_{t+1}},v_{t+1}}\|^2. \tag{18}$$

Finally, we define the drift force $\boldsymbol{F}_d(v_{t+1})$ at vertex v_{t+1} as the gradient vector of $E_d(\boldsymbol{q}_{v_{t+1}})$:

$$\boldsymbol{F}_d(v_{t+1}) \equiv \nabla E_d(\boldsymbol{q}_{v_{t+1}}). \tag{19}$$

Inertia Force: $\boldsymbol{F}_n(v_{t+1})$
If we assume that the interval between successive frames is short enough, we can expect that the motion of the object to be smooth and continuous. This assumption tells us that we can predict the location of a vertex at $t+1$ from its motion history.

We can represent such predictions as a set of prediction lines connecting $\boldsymbol{q}_{v_t}$ and $\hat{\boldsymbol{q}}_{v_{t+1}}$, where $\hat{\boldsymbol{q}}_{v_{t+1}}$ denotes the predicted location of v_{t+1}. Then we can define the inertia force $\boldsymbol{F}_n(v_{t+1})$ in just the same way as the drift force $\boldsymbol{F}_d(v_{t+1})$:

$$\boldsymbol{F}_n(v_{t+1}) \equiv \nabla E_n(\boldsymbol{q}_{v_{t+1}}), \tag{20}$$

where $E_n(\boldsymbol{q}_{v_{t+1}})$ denotes the potential field defined for the set of prediction lines, defined in just the same way as in equation (18).

Overall Vertex Force: $\boldsymbol{F}(v_{t+1})$
Finally we define the vertex force $\boldsymbol{F}(v_{t+1})$ with coefficients $\alpha, \beta, \gamma, \delta, \epsilon$ as fol-

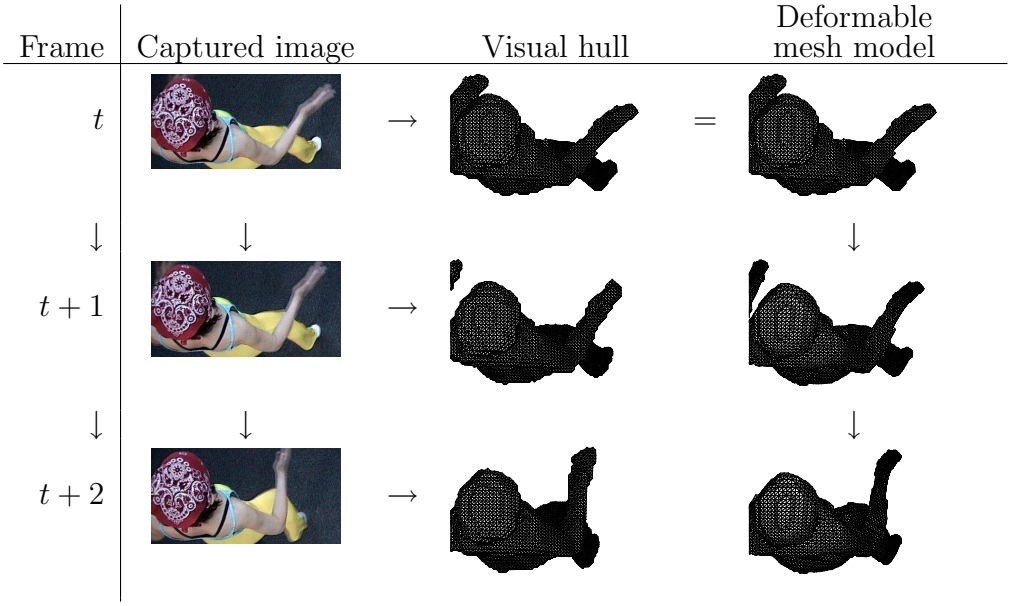


Fig. 25. Successive deformation results (overview)

lows:

$$\boldsymbol{F}(v_{t+1}) \equiv \alpha \boldsymbol{F}_i(v_{t+1}) + \beta \boldsymbol{F}_e(v_{t+1}) + \gamma \boldsymbol{F}_s(v_{t+1}) + \delta \boldsymbol{F}_d(v_{t+1}) + \epsilon \boldsymbol{F}_n(v_{t+1}). \tag{21}$$

Each vertex of the mesh is moved according to $\boldsymbol{F}(v_{t+1})$ at each iteration step until the movement lies under some threshold.

4.5 Experimental Results

Figure 25 and 26 illustrate the inter-frame deformation through 3 successive frames. The columns of Figure 25 show, from left to right, the captured images, the visual hulls generated by the discrete marching cubes method for each frame, and the mesh models deformed frame by frame, respectively. Note that captured multi-view video data are not completely synchronized and include motion blur. In this experiments, we used 9 cameras arranged in the same way as Figure 21 (a) to capture object images. The mesh models consist of about 12,000 vertices and 24,000 triangles, and the processing time per frame is about 10 minutes by PC (Xeon 1.7GHz). Note that the visual hull in frame t was used as the initial shape for the intra-frame deformation and then the resultant mesh for the inter-frame deformation. We used fixed coefficients $\alpha = 0.2, \beta = 0.2, \gamma = 0.2, \delta = 0.3, \epsilon = 0.1$ given a priori.

From these results, we can observe:

- Our dynamic mesh model can follow the non-rigid object motion smoothly.

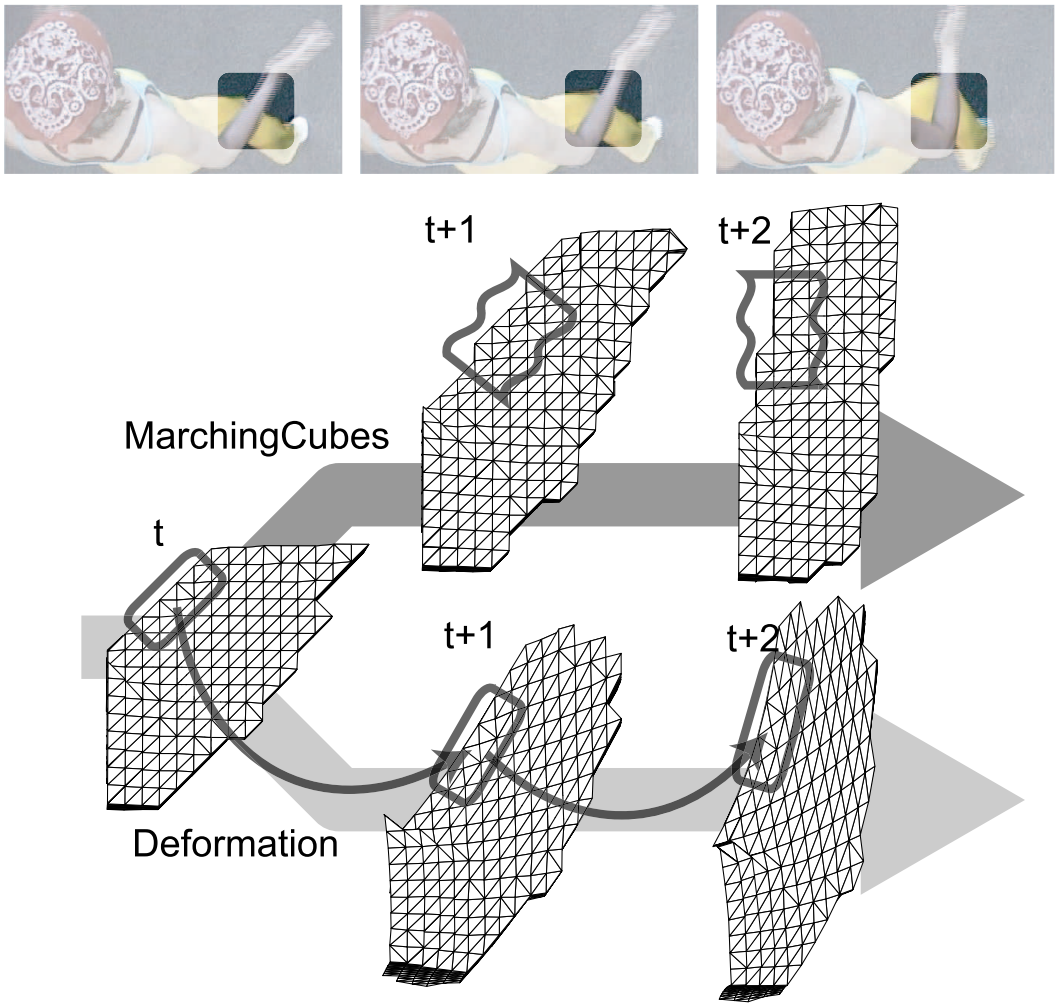


Fig. 26. Successive deformation results (detailed)

- During its dynamic deformation, our mesh model preserves both global and local topological structure and hence we can find corresponding vertices between any pair of frames for all vertices. Figure 26 illustrates this topology preserving characteristic. That is, the left mesh denotes a part of the initial mesh obtained by applying the marching cubes to the visual hull at t. The lower bold arrow stands for the inter-frame deformation process, where any parts of the mesh can be traced over time. Aligned along the upper bold arrow, on the other hand, are parts of the meshes obtained by applying the marching cubes to each visual hull independently, where no vertex correspondence can be established because the topological structures of the meshes are different.

We applied the inter-frame deformation for a long series of frames and observed the following problems:

(1) The mesh cannot well follow motions of non-textured surfaces such as skin areas and sometimes overall mesh shapes deviate from correct ones.

(2) In the dancing lady video sequence, the overall topological structure changes depending on her dancing poses; sometimes hands are attached to her body. The current mesh model cannot cope with such topological changes.

In summary, while we can demonstrate the effectiveness of our dynamic deformable mesh model, it should be improved in the following points:

- To reduce the computation time (i.e. 10 minutes per frame), we have to introduce parallel processing as well as efficient computation schemes such as the coarse-to-fine strategy and/or functional approximations for local surfaces.
- Since the current computation is purely local, bottom-up and data-driven, we should introduce object models to globally control the deformation. Such model information will be useful to obtain physically meaningful deformation. For non-textured surface deformation, firstly, we will achieve more robust motion estimation if we assume articulated motion by increasing the stiffness of springs between vertices[36][37]. Secondly, global object shape models such as articulated shape models will be useful to cope with the topological structure changes described above.

5 High Fidelity Texture Mapping Algorithm

In this section, we propose a novel texture mapping algorithm to generate high fidelity 3D video and then evaluate the effectiveness of the mesh deformation method described in the previous section for generating 3D video . The problem we are going to solve here is how we can generate high fidelity object images from arbitrary viewpoints based on the 3D object shape of limited accuracy. That is, the computed 3D mesh model is just an approximation of the real 3D object shape and include considerable amount of noise.

The input data for the texture mapping algorithm are

- A temporal series of 3D mesh data. We prepare two types of 3D mesh data to evaluate the effectiveness of the mesh deformation for 3D video generation:
  · **Mesh**: 3D mesh data obtained by applying the discrete marching cubes method [15] to the voxel data of each frame.
  · **D–Mesh**: 3D mesh data obtained by applying the intra-frame deformation to **Mesh** of each frame. This is because, as noted at the end of the previous section, the current inter-frame deformation method cannot cope with global topological structure changes. Since the texture mapping algorithm proposed here conducts rendering frame by frame, we have no problem even if the mesh topology changes frame by frame.

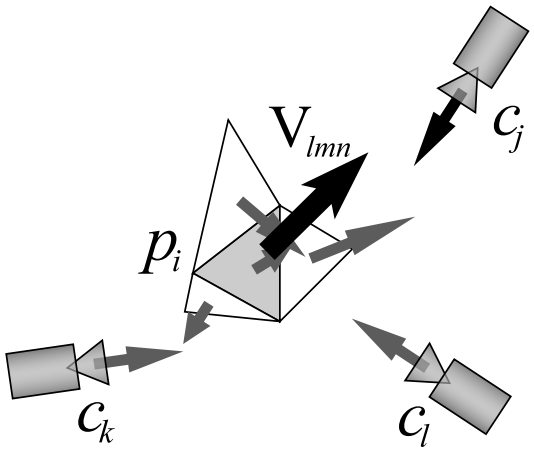


Fig. 27. Viewpoint independent patch-based method

- A temporal series of multi-view video data.
- Camera calibration data for all cameras.

Note that these data were produced by our first generation PC cluster system, since the real-time 3D volume reconstruction by the second generation system described in section 3 is not completed. Note also that since the cameras used in the first generation system do not support a hardware trigger or a shutter speed control, captured multi-view video data are not completely synchronized and include motion blur when a human moves quickly.

5.1 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 3D video, we call it as the Viewpoint Independent Patch-Based Method (VIPBM in short).

Algorithm (Figure 27)

(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 fully textured 3D object shape, which can be viewed from arbitrary viewpoints with ordinary 3D graphic display systems. Moreover, its data size is very compact compared with that of the original multi-view video data.

From the viewpoint of fidelity, however, the displayed image quality is not satisfying;

(1) The best camera c^* for a patch may vary 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 rendering method [38]: a viewpoint dependent vertex-based texture mapping algorithm. In this algorithm, the color (i.e. RGB values) of each vertex of patches is computed taking account of the viewpoint of a viewer and then the texture of each patch is generated by interpolating color values of its three vertices. In what follows, we first define words and symbols to describe the algorithm and then present the computation process, followed by experimental results.

5.2 Viewpoint Dependent Vertex-Based Texture Mapping Algorithm

(1) Definitions
First of all, we define words and symbols as follows (Figure 28), where bold face symbols denote 3D position/directive 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: $\boldsymbol{v}_{eye \to p_i}$
- a viewing direction from c_j toward the centroid of p_i: $\boldsymbol{v}_{c_j \to p_i}$
- vertices of p_i: $\mathbf{v}_{p_i}^k \quad (k = 1, 2, 3)$
- vertex visible from c_j (defined later): $\mathbf{v}_{p_i,c_j}^k$
- RGB values of $\mathbf{v}_{p_i,c_j}^k$ (defined later): $I(\mathbf{v}_{p_i,c_j}^k)$
- 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 such patch ID that is nearest from c_j as well as the

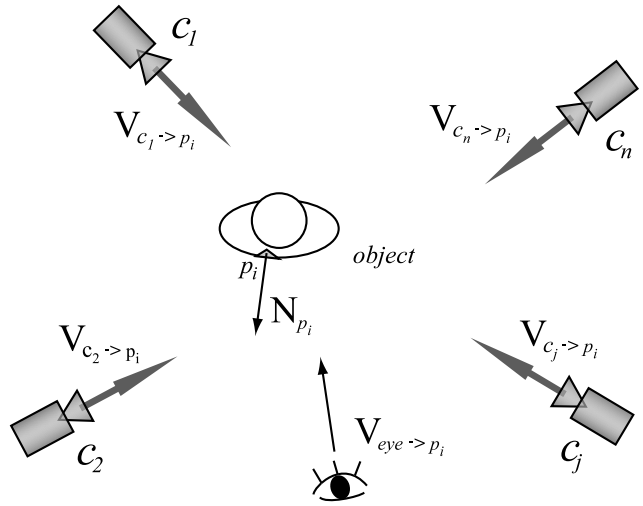


Fig. 28. Viewpoint and camera position

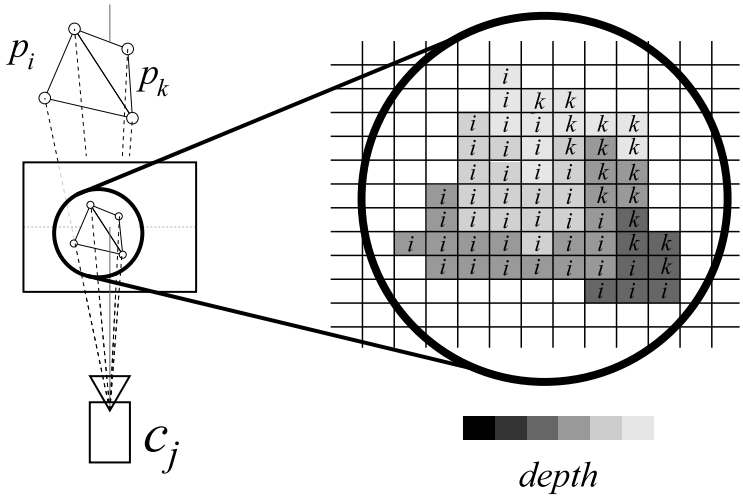


Fig. 29. Depth buffer

distance to that patch from c_j (Figure 29). 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.

$$\boldsymbol{n}_{p_i} \cdot \boldsymbol{v}_{c_j \to p_i} < 0 \tag{22}$$

(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 equation (22) onto the depth buffer B_{c_j}.

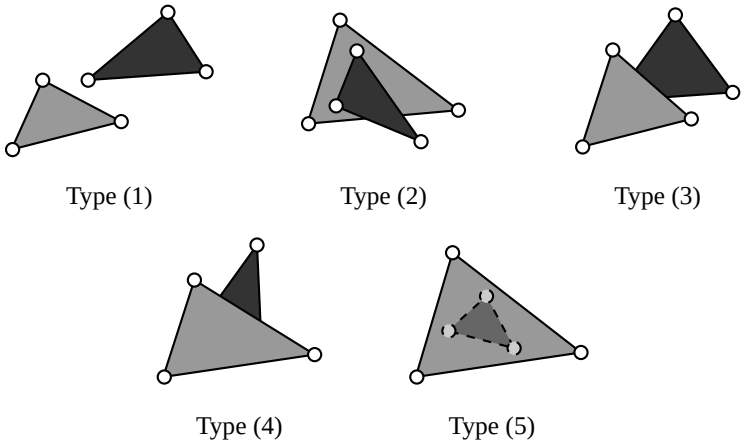


Fig. 30. Relations between patches

(2) Then, check the visibility of each vertex using the buffer. Figure 30 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

$$I(\mathbf{v}^k_{p_i,c_j}) = I_{c_j}(\hat{\mathbf{v}}_{p_i,c_j}), \tag{23}$$

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 $C = \{c_1, c_2, \ldots, c_n\}$. Furthermore, each vertex has information of visibility from each camera.
(2) Specify the viewpoint **eye**.
(3) For each surface patch $p_i \in P$, do 4 to 9.
(4) If $\boldsymbol{v}_{eye\to p_i} \cdot \boldsymbol{n}_{p_i} < 0$, then do 5 to 9.
(5) Compute weight $w_{c_j} = (\boldsymbol{v}_{c_j\to p_i} \cdot \boldsymbol{v}_{eye\to p_i})^m$, 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 to 8.
(7) Compute the normalized weight for $\mathbf{v}^k_{p_i}$ by

$$\bar{w}^k_{c_j} = \frac{w^k_{c_j}}{\sum_l w^k_{c_l}}. \tag{24}$$

Here, if $\mathbf{v}^k_{p_i}$ is visible from camera c_j, then $w^k_{c_j} = w_{c_j}$, else $w^k_{c_j} = 0$.

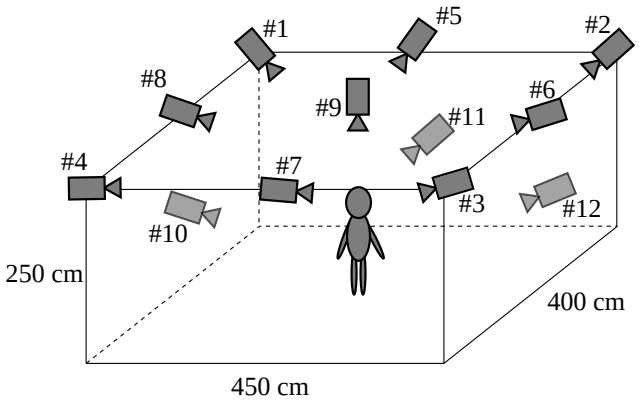


Fig. 31. Camera Setting

(8) Compute the RGB values $I(\mathbf{v}_{p_i}^k)$ of $\mathbf{v}_{p_i}^k$ by

$$I(\mathbf{v}_{p_i}^k) = \sum_{j=1}^{n} \bar{w}_{c_j}^k \; I(\mathbf{v}_{p_i,c_j}^k) \tag{25}$$

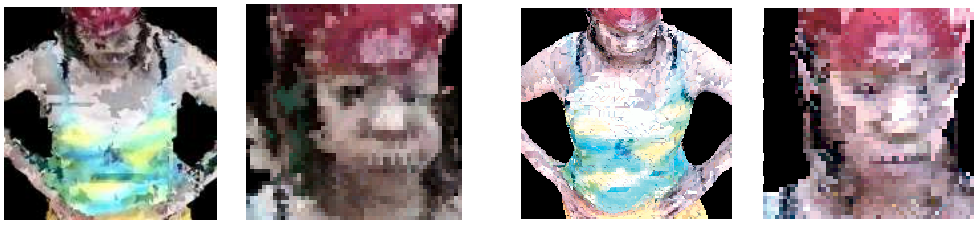
(9) Generate the texture of patch p_i by linearly interpolating RGB values of its vertices. To be more precise, depending on the number of p_i's vertices that are visible from any cameras, the following patch painting processing is conducted:

- **3**.
  Generate RGB values at each point on the patch by linearly interpolating the RGB values of the 3 vertices .
- **2**.
  Compute mean value of the RGB values of the 2 visible vertices, which is regarded as those of the other vertex. Then apply the linear interpolation on the patch.
- **1**.
  Paint the patch by the RGB values of the visible vertex.
- **0**.
  Texture of the patch is not generated: painted by black for example.

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

5.3 Performance Evaluation

To evaluate the performance of the proposed viewpoint dependent vertex-based method (VDVBM in short) and the effectiveness of the mesh deformation, we applied VIPBM and VDVBM to **Mesh** and **D–Mesh** respectively and evaluated the generated images qualitatively.



Mesh **D–Mesh**

Fig. 32. Images generated by the Viewpoint Independent Patch-Based Method

Mesh **D–Mesh**

Fig. 33. Images generated by the Viewpoint Dependent Vertex-Based Method

Figures 32 and 33 show images generated by VIPBM and VDVBM, respectively, using the same frame data of **Mesh** and **D–Mesh** and the same viewpoints. From these images, we can observe

- Comparing those images generated by VIPBM and VDVBM, the former introduces many jitters in images, which are considerably reduced by the latter.
- Comparing those images generated with **Mesh** and **D–Mesh**,
  - VIPBM with **D–Mesh** can generate better images; while many jitters are still included, detailed textures can be observed. This is because the surface normals are smoothed and become more accurate by the mesh deformation.
  - On the other hand, VDVBM with **D–Mesh** does not show any observable improvements and instead seems to introduce more blurring effects. This is because in VDVBM, the viewpoint information to generate an image plays much more important role than the accuracy of the 3D shape. In other words, VDVBM can work well even for 3D object shape data of limited accuracy.

Figure 34 compares images generated by VIPBM and VDVBM with **D–Mesh** to their corresponding original video images. This also verify the effectiveness of VDVBM.

Next, we conducted quantitative performance evaluations of VIPBM and VDVBM with **Mesh** and **D–Mesh**. We calculate RGB root-mean-square (rms) errors between a real image captured by camera c_j and its corresponding images generated by VIPBM and VDVBM, respectively: in generating the

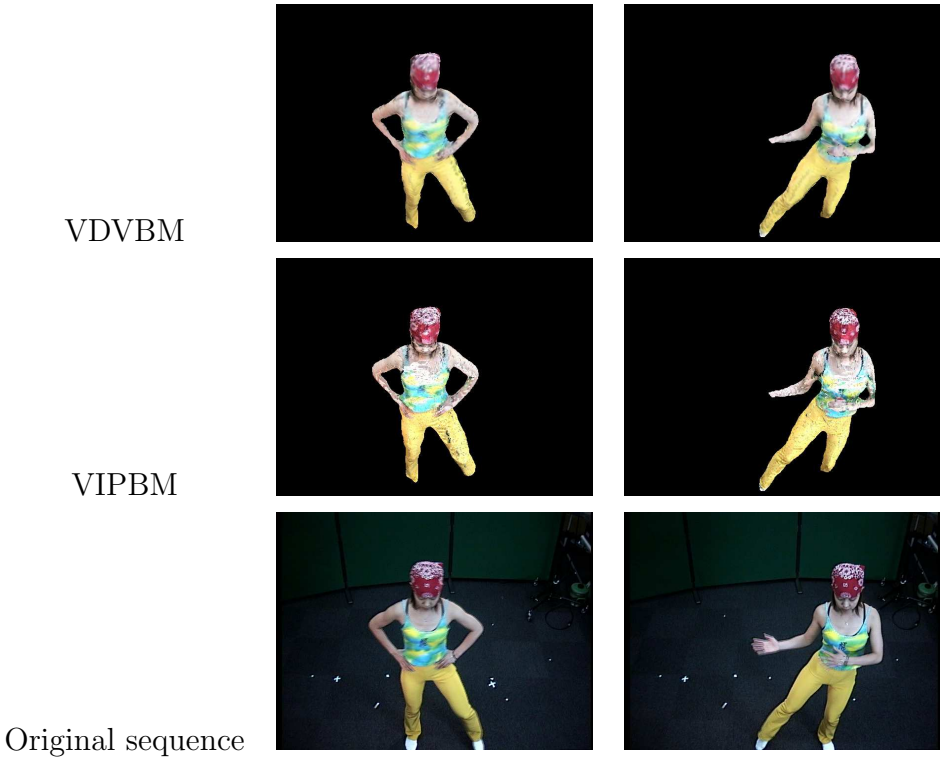


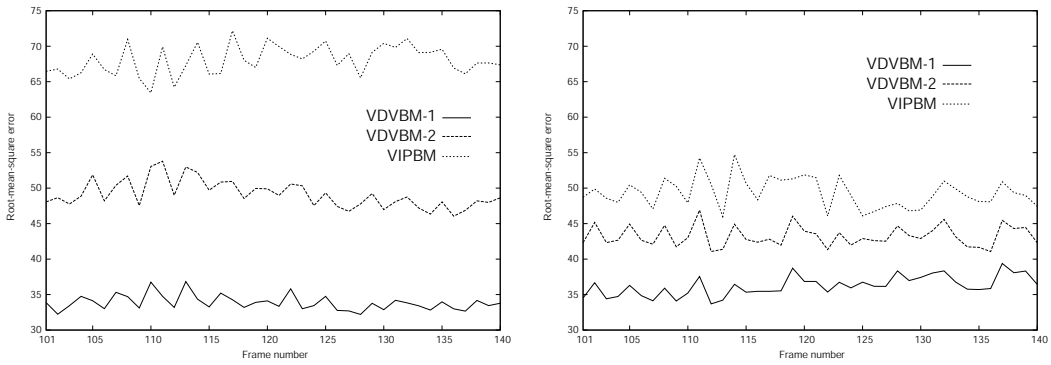
Fig. 34. Sample images of generated 3D video with **D–Mesh**

images, the position and direction of camera c_j are used as those of the viewpoint for the 3D video (i.e. **eye** in VDVBM). To evaluate the performance of VDVBM, we employed two methods: VDVBM–1 generates images by using all multi-view images including real images captured by camera c_j itself, while VDVBM–2 excludes such real images captured by camera c_j. The experiments were conducted under the following settings:

- camera configuration: Figure 31
- image size: 640×480[pixel] 24 bit RGB color
- viewpoint: camera 5
- weighting factor in VDVBM: $m = 5$

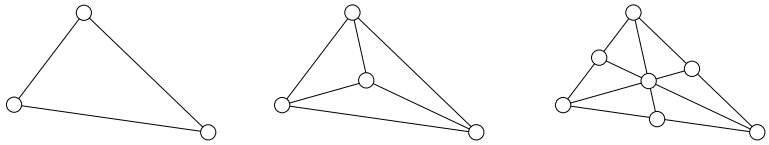
Figure 35 illustrates the experimental results, where rms errors for frame 101 to 140 are computed. This figure proves that

- VDVBM–1 and VDVBM–2 perform better than VIPBM with both **Mesh** and **D–Mesh**.
- Comparing **Mesh** with **D-Mesh**,
  - In VIPBM, **D–Mesh** improves the fidelity of generated images significantly.
  - In VDVBM–2, **D-Mesh** reduces rms errors about 10%.
  - In VDVBM–1, on the other hand, **D-Mesh** increases rms errors slightly.



Mesh D–mesh

Fig. 35. Root-mean-square errors of RGB values (1)



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

Fig. 36. Subdivision of a surface patch

This is because **D–Mesh** introduces blurring effects as discussed before.

Finally, we tested how we can improve the performance of VDVBM by increasing the spatial resolution of patch data. Note that we use **D-Mesh** alone in this experiment. Figure 36 shows the method of subdividing a patch into three (S3) and six (S6) sub-patches to increase the spatial resolution.

To evaluate the physical spatial resolution, we examine the average side length of a patch on the image plane of each camera by projecting original and subdivided patches onto the image plane. Figure 37 shows the mean side length in pixel on the image plane of each camera. Note that since camera 9 is located closer to the 3D object (see Figure 31), 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 Figure 37.

We can observe that the spatial resolution of S6 is approximately the same as that of an observed image (i.e. 1 pixel). That is, S6 attains the finest resolution, which physically represents about 5mm on the object surface. To put this in another way, we can increase the spatial resolution up to the six sub-division, which can improve the quality of images generated by VDVBM.

To quantitatively evaluate the quality achieved by using subdivided patches, we calculated root-mean-square errors between real images and images gener-

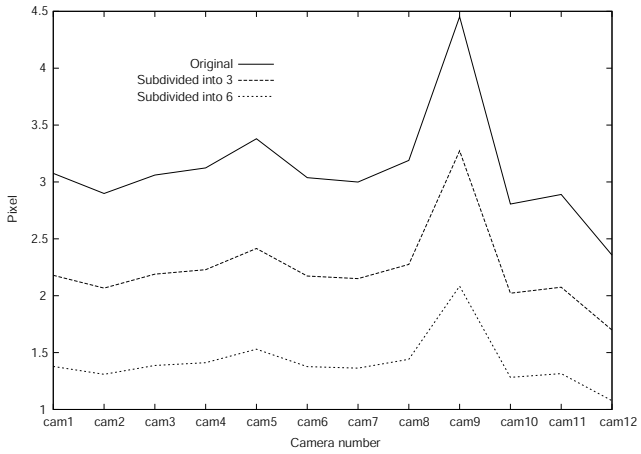


Fig. 37. Mean side length in pixel on image planes of cameras

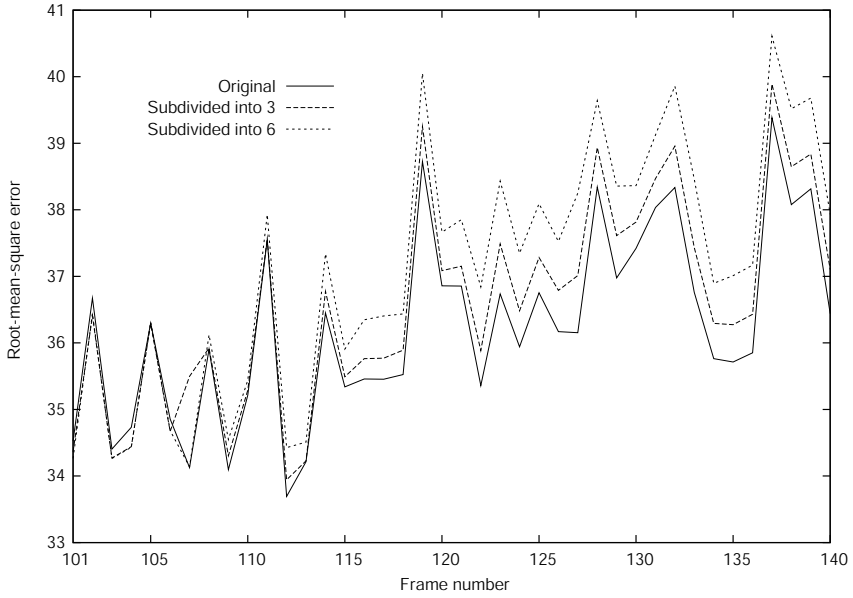


Fig. 38. Root-mean-square errors of RGB values (2)

ated by VDVBM-1 with original, S3, and S6, respectively (Figure 38). From this experiment, we observe that subdividing patches does not significantly reduce the errors. The reasons of this can be considered as follows. As shown in Figure 39, most of the errors arise around the contour of the object and edges of texture (e.g. an edge between skin and clothes, etc.). These errors cannot be reduced by subdividing patches because they come from motion blur or asynchronization, i.e. capturing the images is not perfectly synchronized.

Note that while the total numerical errors are not reduced significantly, the quality of generated images is improved by using subdivided patches. Figure 40 shows that the spatial resolution is increased, i.e. the blurring effect is reduced, by subdividing patches.

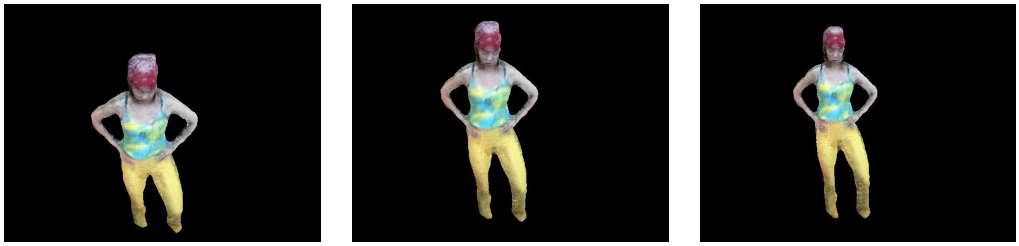
Fig. 39. Difference image between a real image and a generated image (frame #106)

Original Subdivided (S6)

Fig. 40. Example images visualized with original and subdivided patches (frame #103)

Finally, we show examples generated by VDVBM–1 with subdivided patches (S6) viewed from camera 5, 11, and an intermediate point between them (Figure 41). Figure 41 shows that the images generated by VDVBM–1 look almost real even when they are viewed from the intermediate point of the cameras.

By compiling a temporal sequence of reconstructed 3D shape data and multi-view video into a temporal sequence of vertex lists, which takes about 3 seconds per volume, we can render arbitrary VGA views of 3D video sequence at video rate by an ordinary PC.



cam 5 | an intermediate image (cam 5 to cam 11) | cam 11

Fig. 41. Visualized 3D video with subdivided patches (frame#103)

6 Conclusion

We are proposing 3D video as new image media: it records the object's full 3D shape, motion, and surface properties (i.e. color and texture). In this paper, we presented research attainments so far obtained to generate 3D video:

(1) A PC cluster system for real-time reconstruction of dynamic 3D object actions from multi-view video images: we first developed the plane-based volume intersection method, where the 3D voxel space is partitioned into a group of parallel planes and the cross-section of the 3D object volume on each plane is reconstructed. Secondly, we devised the Plane-to-Plane Perspective Projection algorithm to realize efficient plane-to-plane projection computation. And thirdly, to realize real-time processing, we implemented parallel pipeline processing on a PC cluster system. Experimental results showed that the proposed methods works efficiently and the PC cluster system can reconstruct 3D shape of a dancing human at about 12 volume per second in the voxel size of 2cm× 2cm× 2cm contained in a space of 2m × 2m × 2m.

(2) 3-base-plane volume intersection method: we augmented the plane based volume intersection method to cope with dynamic human and camera actions. Experimental results demonstrated its computational efficiency and stability against the dynamic actions.

(3) A deformable 3D mesh model for reconstructing the accurate 3D object shape and motion: we proposed the intra- and inter-frame deformation methods. The former employs the smoothness, silhouette, and photo-consistency constraints to control the deformation, while the latter introduces the 3D motion flow and inertia constraints additionally to cope with non-rigid object motion. Experimental results showed that the mesh deformation methods can significantly improve the accuracy of the reconstructed 3D shape. But their computation speeds are far from real-time: for both the intra- and inter-frame deformations, it took about 5 minutes for 12000 vertices with 4 cameras and 10 minutes for 12000 vertices with 9 cameras by PC (Xeon 1.7GHz).

(4) An algorithm of rendering high fidelity texture on the reconstructed 3D object surface from the multi-view video images: we proposed the viewpoint dependent vertex-based method to avoid jitters in rendered object images which are caused due to the limited accuracy of the reconstructed 3D object shape. Experimental results showed that the proposed method can generate almost natural looking object images from arbitrary viewpoints. By compiling a temporal sequence of reconstructed 3D shape data and multi-view video into a temporal sequence of vertex lists, which takes about 3 seconds per volume, we can render arbitrary VGA views of 3D video sequence at video rate by an ordinary PC.

While we believe that the proposed methods set a milestone to realize 3D video, we still have to develop the followings to make 3D video usable in everyday life.

- higher speed and more accurate 3D action reconstruction methods, especially for the 3D mesh deformation,
- methods of generating more natural images which can work even for 3D shape data of limited accuracy,
- methods of editing 3D video for artistic image content, and
- efficient data compression methods to reduce the huge data size of 3D video.

With these novel technologies, we will be able to open up new image media world and promote personal and social activities in education, culture, entertainment, sport, and so on.

This work was supported by the grant-in-aid for scientific research (A) 13308017 and 13224051. We are grateful to Real World Computing Partnership, Japan for allowing us to use their multi-view video data.

References

[1] S. Moezzi, L. Tai, P. Gerard, Virtual view generation for 3d digital video, IEEE Multimedia (1997) 18–26.

[2] T. Matsuyama, X. Wu, T. Takai, T. Wada, Real-time dynamic 3d object shape reconstruction and high-fidelity texture mapping for 3d video, IEEE Trans. on Circuit and Systems for Video Technology 14 (2004) 357–369.

[3] S. Nobuhara, T. Matsuyama, Dynamic 3d shape from multi-viewpoint images using deformable mesh models, in: Proc. of 3rd International Symposium on Image and Signal Processing and Analysis, Rome, Italy, 2003, pp. 192–197.

[4] T.Matsuyama, X.Wu, T.Takai, S.Nobuhara, Real-time generation and high fidelity visualization of 3d video, in: Proc. of Mirage 2003, 2003, pp. 1–10.

[5] T. Kanade, P. Rander, P. J. Narayanan, Virtualized reality: Constructing virtual worlds from real scenes, IEEE Multimedia (1997) 34–47.

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

[7] E. Borovikov, L. Davis, A distributed system for real-time volume reconstruction, in: Proc. of International Workshop on Computer Architectures for Machine Perception, Padova, Italy, 2000, pp. 183–189.

[8] G. Cheung, T. Kanade, A real time system for robust 3d voxel reconstruction of human motions, in: Proc. of Computer Vision and Pattern Recognition, South Carolina, USA, 2000, pp. 714–720.

[9] J. Carranza, C. Theobalt, M. A. Magnor, H.-P. Seidel, Free-viewpoint video of human actors, ACM Trans. on Computer Graphics 22 (3).

[10] M. Li, M. Magnor, H.-P. Seidel, Hardware-accelerated visual hull reconstruction and rendering, Proc. of Graphics Interface (GI'03) .

[11] T. Matsuyama, R. Yamashita, Requirements for standardization of 3d video, ISO/IEC JTC1/SC29/WG11, MPEG2002/M8107 .

[12] T. Matsuyama, T. Takai, Generation, visualization, and editing of 3d video, in: Proc. of symposium on 3D Data Processing Visualization and Transmission, Padova, Italy, 2002, pp. 234–245.

[13] H. Tezuka, A. Hori, Y. Ishikawa, M. Sato, Pm: An operating system coordinated high performance communication library, *high-performance computing and networking*, lecture notes in computer science, vol. 1225 (1997).

[14] T. Wada, T. Matsuyama, Appearance sphere : Background model for pan-tilt-zoom camera, in: Proc. of 13th International Conference on Pattern Recognition, Vienna, Austria, 1996, pp. A–718–A–722.

[15] Y. Kenmochi, K. Kotani, A. Imiya, Marching cubes method with connectivity, in: Proc. of 1999 International Conference on Image Processing, Kobe, Japan, 1999, pp. 361–365.

[16] H. Baker, Three-dimensional modelling, in: Proc. of Fifth International Joint Conference on Artificial Intelligence, 1977, pp. 649–655.

[17] B. G. Baumgart, Geometric modeling for computer vision, Technical Report AIM-249, Artificial Intelligence Laboratory, Stanford University .

[18] W. N. Martin, J. K. Aggarwal, Volumetric description of objects from multiple views, IEEE Transactions on Pattern Analysis and Machine Intelligence 5(2) (1987) 150–158.

[19] M. Potmesil, Generating octree models of 3d objects from their silhouettes in a sequence of images, Computer Vision,Graphics, and Image Processing 40 (1987) 1–29.

[20] P. Srinivasan, P. Liang, S. Hackwood, Computational geometric methods in volumetric intersections for 3d reconstruction, Pattern Recognition 23(8) (1990) 843–857.

[21] R. Szeliski, Rapid octree construction from image sequences, CVGIP: Image Understanding 58(1) (1993) 23–32.

[22] K. M. Cheung, S. Baker, T. Kanade, Visual hull alignment and refinement across time: A 3d reconstruction algorithm combining shape-from-silhouette with stereo, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2003, pp. 375–382.

[23] W. Matusik, C. Buehler, L. McMillan, Polyhedral visual hulls for real-time rendering, in: Procòf the 12th Eurographics Workshop on Rendering Techniques, 2001, pp. 115–126.

[24] A. Laurentini, How far 3d shapes can be understood from 2d silhouettes, IEEE Transactions on Pattern Analysis and Machine Intelligence 17(2) (1995) 188–195.

[25] K. N. Kutulakos, S. M. Seitz, A theory of shape by space carving, in: Proc. of International Conference on Computer Vision, Kerkyra, Greece, 1999, pp. 307–314.

[26] T. Matsuyama, N. Ukita, Real-time multi-target tracking by a cooperative distributed vision system, Proceedings of IEEE 90 (7) (2002) 1136–1150.

[27] P. Fua, Y. G. Leclerc, Using 3-dimensional meshes to combine image-based and geometry-based constraints, in: ECCV (2), 1994, pp. 281–291.

[28] G. Cross, A. Zisserman, Surface reconstruction from multiple views using apparent contours and surface texture (2000).

[29] S. Vedula, S. Baker, S. Seitz, T. Kanade, Shape and motion carving in 6d, in: Computer Vision and Pattern Recognition (CVPR), Vol. 2, 2000, pp. 592–598.

[30] J. Isidoro, S. Sclaroff, Stochastic mesh-based multiview reconstruction, in: Proceedings of 3D Data Processing, Visualization, and Transmission, Padova, Italy, 2002, pp. 568–577.

[31] H. Briceño, P. Sander, L. McMillan, S. Gortler, H. Hoppe, Geometry videos: A new representation for 3d animations, in: Proc. of ACM Symposium on Computer Animation, 2003, pp. 136–146.

[32] M. Kass, A. Witkin, D. Terzopoulos, Snakes: Active contour models, International Journal of Computer Vision 1 (4) (1988) 321–331.

[33] D. Terzopoulos, J. Platt, A. Barr, K. Fleischer, Elastically deformable models, in: Proc. of SIGGRAPH 87, 1987, pp. 205–214.

[34] D. Baraff, A. Witkin, Large steps in cloth simulation, Proc. of SIGGRAPH 1998 32 (1998) 43–54.

[35] D. Burr, A dynamic model for image registration, Computer Graphics and Image Processing 15 (1981) 102–112.

[36] X. Provot, Deformation constraints in a mass-spring model to describe rigid cloth behavior, in: W. A. Davis, P. Prusinkiewicz (Eds.), Graphics Interface '95, Canadian Human-Computer Communications Society, 1995, pp. 147–154.

[37] J. Christensen, J. Marks, J. T. Ngo, Automatic motion synthesis for 3d mass-spring models, The Visual Computer 13 (1) (1997) 20–28.

[38] P. E. Debevec, Y. Yu, G. Boshokov, Efficient view-dependent image-based rendering with projective texture-mapping, in: Proc. of 9th Eurographics Rendering Workshop, 1998, pp. 105–116.