Parallel Pipeline Volume Intersection for Real-Time 3D Shape Reconstruction on a PC Cluster

Xiaojun Wu
Department of Intelligence Science and Technology,
Graduate School of Informatics, Kyoto University, Japan.
(Now working at NTT Cyber Space Laboratories, Japan.)

Osamu Takizawa
Fujitsu Nagano Systems Engineering Limited, Japan.

Takashi Matsuyama
Department of Intelligence Science and Technology,
Graduate School of Informatics, Kyoto University, Japan.

Abstract

The human activity monitoring is one of the major tasks in the field of computer vision. Recently, not only the 2D images but also 3D shapes of a moving person are desired in kinds of cases, such as motion analysis, security monitoring, 3D video creation and so on. In this paper, we propose a parallel pipeline system on a PC cluster for reconstructing the 3D shape of a moving person in real-time. For the 3D shape reconstruction, we have extended the volume intersection method to the 3-base-plane volume intersection. By thus extension, the computation is accelerated greatly for arbitrary camera layouts. We also parallelized the 3-base-plane method and implemented it on a PC cluster. On each node, the pipeline processing is adopted to improve the throughput. To decrease the CPU idle time caused by I/O processing, image capturing, communications over nodes and so on, we implement the pipeline using multiple threads. So that, all stages can be executed concurrently. However, there exists resource conflicts between stages in a real system. To avoid the conflicts while keeping high percentage of CPU running time, we propose a tree structured thread control model. As a result, We achieve the performance as obtaining the full 3D volumes of a moving person at about 12 frames per second, where the voxel size is $5 \times 5 \times 5$ *[mm^3]. The effectiveness of the thread tree model in such real-time computation is also proved by the experimental results.*

1. Introduction

The human activity monitoring is a major task in the field of computer vision. And the detailed 3D shape of the human body enables detailed analysis of the human motion and helps greatly to understand the human action. From the viewpoint of the image media technology, the 3D shape model is also desired by the contents creation, such as the free viewpoint video or the 3D video contents[11, 7, 5, 12, 8, 4, 1, 2, 13], and so on. Furthermore, to realize the high reality remote collaboration in the virtual space[9], the 3D shape of the target must be acquired in real time. Therefore, we focus on the real-time 3D shape reconstruction of a moving person.

The volume intersection method[6] is well known to get the full 3D shape of the target. Although only the visual hull which approximately describes the true 3D shape can be obtained by this method, the computation is simple and robust for the real-time system. To accelerate the computation, two kinds of approaches have been proposed: (1) the octree representation based acceleration[10] and (2) the plane-to-plane projection based acceleration[3]. Since the octree representation is optimized for binary silhouettes, there is no room left to extend the binary silhouette images to gray scale or color images. Also, for some sparse objects such as a skeleton, the octree representation may not be compact. Therefore, we employ the plane-to-plane projection based approach as our basic method. The following summarizes the algorithm of the plane-based volume intersection from multi-viewpoint cameras system. Suppose that the 3D space

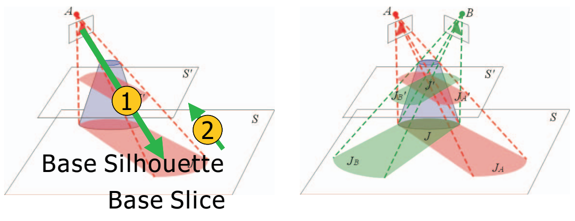

Figure 1. Plane-based volume intersection method

is decomposed into parallel planes(slices) so that the shape are calculated as the cross-sections on each slice. The algorithm is also illustrated on Figure 1.

1. **Base Silhouette Generation(BSG)** : Project the object silhouette observed by each camera onto a common base plane (Figure 1 left, ①).

2. **Visual Cone Generation(VCG)** : Project each base plane silhouette onto the other parallel planes (Figure 1 left, ②).

3. **Visual Cone Intersection(VCI)** : Compute 2D intersection of all silhouettes projected on each plane (Figure 1 right).

While the plane based volume intersection accelerate the computation of the visual hull, it is obvious that the computation of the "BSG" will break down if any input screen of the multi-viewpoint cameras is nearly perpendicular to the base plane. That is, in case of the base plane being statically determined, the computation can not be carried out for arbitrary camera layouts.

In this paper, we propose the 3-base-plane volume intersection, which is an extension of the plane based volume intersection. By this extension, not only the computational breakdown due to the camera layout can be avoided, but also the upper bound of the computational complexity of the BSG can be estimated, which is important for the real-time computation.

We also developed the parallel 3-base-plane volume intersection algorithm and applied it on a PC cluster. In such parallel system, it is important to keep high percentage of the CPU running time on each node for a high performance. On a real system, the CPU idle time is mainly caused by the I/O processing, such as image capturing and communications over network interface. To decrease such idle time, we propose the pipeline processing model on each node. That is, divide the processing on each node into multiple stages keeping I/O processing and computation separately. By implementing the stages as multiple threads, all stages can be executed concurrently. However, there exist two kinds of problems which disturb the concurrency of the stages. The first one is the hardware resource conflict on one PC, and the second one is the extra wait caused by the asynchronous communication over PCs. To get rid of these problems, we propose the tree structured thread control module for the threads scheduling. By this thread tree model, high percentage of CPU running time is achieved while avoiding the resource conflicts and keeping the communication synchronized.

In what follows, we first describe the 3-base-plane volume intersection and its parallelization in details. After that, the thread tree control model is shown. Experiments of tuning the executing order of the threads by changing the structure of the thread tree are also conducted. At last, the performance evaluation experiments are shown to prove the efficiency of the proposed algorithm and the implementation model.

2. The 3-Base-Plane Volume Intersection

2.1. Algorithm of the 3-Base-Plane Volume Intersection

To avoid the fatal case of the base plane selection, we extend the base plane volume intersection to the 3-base-plane method. We determine 3 mutually orthogonal planes as the candidates of the base plane. In fact, the candidates are fixed as the orthogonal planes parallel to the XY, YZ, ZX planes of the world coordinates system respectively. During the "BSG" phase of the plane based volume intersection, for each camera, one of the candidates is selected as its base plane according to the camera viewing direction. During the next "VCG" phase, for each camera, we first decompose the 3D space into isometric parallel slices, which are parallel to its base plane. Also the distance between two neighbor slices equals to the voxel size (= the pixel size of the base silhouette). Hence, for any two cameras, the decomposed slices sets are either identical or perpendicular to each other.

After decomposing the 3D space, we generate the visual cone of each camera by projecting its base silhouette onto the slices. To get the visual hull of all cameras, all of the visual cones, i.e. the silhouettes sets of all cameras need to be intersected. For the cameras whose slices sets are identical, the intersection can be done by intersecting the silhouettes on each slice directly. However, for the cameras whose slices sets are perpendicular, the conversion of the silhouettes set is necessary. Notice that, since the 3 base planes are mutual orthogonal, we can convert one type of the slices set to another type by just switching the coordinates without any heavy computation.

Then, after we have all visual cones calculated as silhouettes on one same slices sets, we can intersects all of the silhouettes on each slice to get the visual hull of all cameras. As a result, while the base plane for each camera is selected as one of the 3 orthogonal planes, all input silhouettes are combined to generate one visual hull. The 3-base-plane algorithm is summarized as follows:

1. **Base Plane Selection & Base Silhouette Generation(BSG)**: By the base plane selection, a set of cameras are partitioned into 3 groups: cameras in each group share the same base plane.
2. **Visual Cone Generation (VCG)**: Project the base object silhouettes onto each slice.
3. **Visual Cone Conversion (VCC)**: Convert the visual cone of each camera into the silhouettes on slices with the same direction.
4. **Visual Cone Intersection (VCI)**: All visual cones are intersected to generate the complete 3D object shape.

It is clear that the fatal problem of the base plane selection can be avoided by selecting the base plane from 3 orthogonal candidates. Comparing with the plane based volume intersection, only the phase "VCC" is added and its computational cost is little.

3. Parallelization of the 3-Base-Plane Volume Intersection

For the parallel processing, we apply the PC cluster architecture for the computation. The base silhouette duplication parallel algorithm is proposed and is shown as follows:

1. **Image Capturing (IC)**
2. **Input Object Silhouette Generation (ISG)**
3. **Base Silhouette Generation (BSG)** On each camera node, generate the base silhouette on the selected base plane.
4. **Base Silhouette Duplication (BSD)** Suppose the cameras are grouped at least 2 groups, the base silhouette duplication is conducted as follows.
   (a) Assign computation nodes to the 3 camera groups to balance the computational cost for the visual hull generation of the 3 camera groups.
   (b) Duplicate the base silhouettes among the nodes in the same group.
5. **Visual Cone Generation and Intersection (VCGI)** Within each group, divide the slices into subsets by the node number of the group. On each node, generate the subsets of the visual cone and compute the intersections of them to get the subset of visual hull on each node.
6. **Visual Hull Conversion (VHC)** On each node, convert the subset of the calculated visual hull to the cross-sections perpendicular to one of the axes, for example, the Z-axis.
7. **Transfer Visual Hull (TVH)** Transfer the converted visual hull to the nodes in the camera group whose base plane is perpendicular to the Z-axis.
8. **Visual Hull Intersection (VHI)** On each node which received the visual hulls, compute the intersection of the subset of the visual hulls to obtain the subset of the 3D shape.

On each node PC, the above parallel algorithm is realized by pipeline processing, i.e. each phase is taken as one stage, and the throughput can be increased greatly. For the pipeline implementation, we proposed the thread three model.

4. The Thread Tree Control Model for the Parallel Pipeline Implementation

4.1. Parallel Pipeline Processing Model

Table 1. Process Type on Camera & Computation Node

Step	Computation	Data I/O	
		Local I/O	Remote I/O
1		IC	
2	ISG		
3	BSG		
4			BSD
5	VCGI		
6	VHC		
7			TVH
8	VHI		

By the parallel algorithm shown above, the computation of the 3-base-plane volume intersection is decomposed onto each PC of the cluster. The processes assigned on each PC can then be categorized as follows, which is also summarized in Table 1.

- **Computation Phase**

  The computation phase is the step where the process time is mainly spent for arithmetic instructions. For the camera node, the steps of ISG, BSG, VCGI, VHC and VHI are categorized as computation phases. For

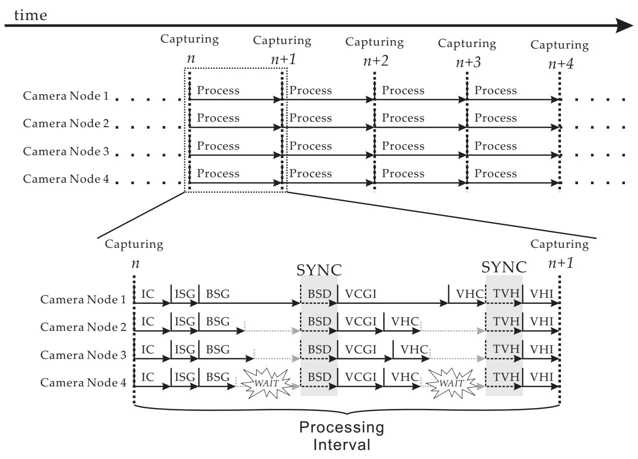

Figure 2. Sequential Processing Model on Node PC

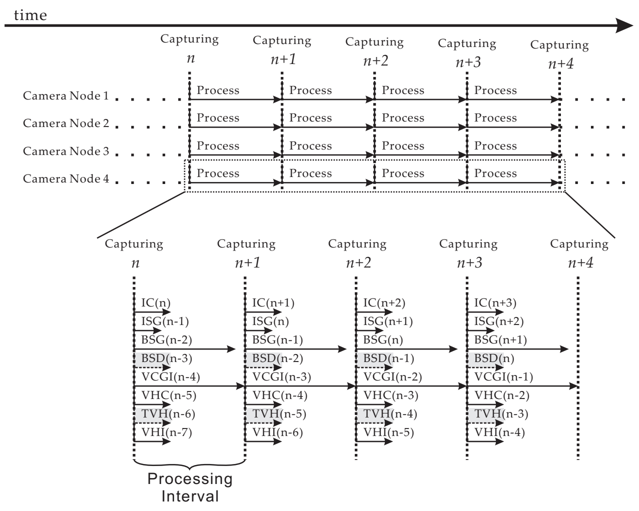

Figure 3. Pipeline Processing Model on Node PC

the computation node, computation phases consists of the steps of VCGI, VHC and VHI.

- **Data I/O**

  The data I/O phase is the step where the process time is mainly spent for reading or writing data between the memory and the I/O devices, for example, storages, the capture board, the network card and so on. While such processes need little CPU power on recent PC architecture, the computation phases is stopped by these processes. On a PC cluster system, the data I/O phases consists of 2 types as local I/O and remote I/O. In the above parallel algorithm, for the camera node, the local I/O phase contains the IC step and the remote I/O phases contain the BSD and TVH steps. For the computation node, all data I/O phases are the remote I/O phases which contain the BSD and TVH steps.

Notice that, while the time spent for the data I/O may be shortened by applying high speed devices, the remote I/O causes synchronization between nodes on a PC cluster, and it may cause extra wait time between the computation phases. Figure 2 shows an example of time flow for 4 camera nodes executing the parallel 3-base-plane volume intersection. On the top of the figure, at each capturing time, the nodes are synchronized. And after the capturing, the computation is distributed on each node. We call the processing between 2 capturing on each node as the processing cycle, and the details of one processing cycle are shown at the bottom of the figure. On each node, the processes are carried out sequentially. At the start, all nodes are synchronized for the image capturing. Since the computation amount of BSG and VCGI depend on the size of the input silhouette and the camera layout, the time for these phases differs on each node. While the end timing of these phases on each node are different, the end timing of the BSD and the TVH are synchronized by the communication between nodes. It is clear that extra wait time is caused by the synchronization, and such wait time will not be shortened even though the time for BSD or TVH are shortened.

To get rid of such extra wait time, we propose the pipeline processing model on each node PC. That is, on the PC cluster, the parallel pipeline processing is carried out like Figure 3 shows.

In the figure, the processing flow of 4 camera nodes is shown on the top, and the details of the processing cycle on one node is shown at the bottom. On each node, the 8 steps of IC, ISG, BSG, BSD, VCGI, VHC, TVH and VHI are executed as 8 stages of the pipeline processing, i.e. each stage is the processing step for each of the continuous 8 frames respectively. In this parallel pipeline processing model, all nodes are only synchronized for the image capturing at the start of each cycle. After that, processes on each node are carried out concurrently. Since both the computation phases and the data I/O phases are executed simultaneously in one cycle, the computation phases will not be stopped by data I/O, including the remote I/O phases. That is, no extra wait time is spent during one cycle in the parallel pipeline processing model. Although the process for one frame is accomplished in multiple cycles, the processing interval is shortened from the summation of all steps + extra wait time to the time of the stage which runs longest.

To realize the pipeline processing model shown in Figure 3 on each node, the concurrent programming platform is required. While most recent operating systems support concurrent programming, there exist the following problems in practice:

- **Hardware Resource Conflict**

  As Figure 3 shows, the BSD and the TVH stages need access to the network devices. If only one network card is available for accessing, the two stages can not be executed at the same time. That is the mechanism for avoiding such resource conflict is required.

- **Limitation of CPU Resource**

  In most recent operating systems, the concurrent programming is realized by the time sharing scheduling (TSS) mechanism. That is, each full program is split into multiple executive units, and by switching the executive unit on CPU in a tiny time interval, multiple programs are executed concurrently in appearance. On such OS, the number of the units concurrently invoked at exactly same time will never be greater than the number of the CPU. Suppose each stage in Figure 3 is the executive unit, in practice, if the CPU number is less than 8, all 8 stages will never be executed at exactly same time, but be executed in some order determined by the OS scheduling mechanism.

- **Stage Order**

  Since not all stages can be invoked at exactly same time in practice, the invoking order of the stages will affect the process time of one cycle. If the time of each stage is static, the optimal order can be determined beforehand. But as mentioned so far, the process time changes dynamically while the shape of the object changes. So the mechanism to change the stage order dynamically is required.

As the summary, both the mechanisms of avoiding resource conflict and managing the stage order are required to realize the parallel pipeline processing for the real-time 3D shape reconstruction.

Furthermore, the parallel pipeline processing model is suitable for a general parallel computation on a PC cluster architecture because the extra wait time caused by the data sharing can be got rid of. Also the problems of the resource conflict and the stage order, i.e. the order of executive unit, are also general problems for the concurrent programming on most recent PCs.

In what follows, the system design to realize the parallel pipeline processing on a PC cluster is shown in details.

4.2. Multi-thread Implementation of Multiple Stages

To realize the parallel pipeline processing on the PC cluster, the following types of modules are designed.

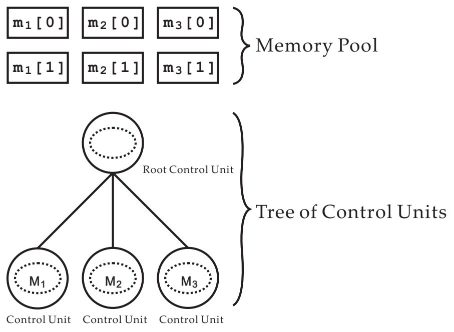

Figure 4. Overview of The Control Module

- **Processing Module** This module consists of the memory addresses for the input and the output, and the program routines of one step in the parallel algorithm. The processing module can be categorized as the following 3 types : Computation Module, Local I/O Module and Communication Module. Each phase of the parallel algorithm is implemented as one processing module and is bound to one thread.

- **Control Module** To realize the parallel pipeline processing, not only the input and the output of each module should be assigned correctly, but also the mechanisms of avoiding resource conflict and managing stage order should be realized. While each processing module is bounded to one thread, the control module is designed to manage the threads and to realize the above tasks. The details of the control module will be described in the next section.

4.3. Thread Tree Control Model for Threads Scheduling

The overview of the control module is shown in Figure 4. The control module consists of a memory pool and multiple control units which are organized in a tree structure. The details of these items are shown below.

By keeping double memory blocks for each phase, each phase can be executed concurrently. By alternating the buffer index between 0 and 1, the last output of the precede phase is given to the subsequent phase as the input, thus just realizes the pipeline processing. By applying this input & output assignment for each cycle, each processing module realizes one stage of the pipeline processing.

The control module consists of multiple control units and each of them is just one thread. All units are organized as a tree structure as shown below. Notice that the structure of the root control unit differs from others and is shown later.

Common Control Unit The following shows the details of the structure of the control unit.

- Parent Handle: The handle of the control unit which is the parent of this control unit.
- Children Handle List: The list contains the handles of the control units which are the children of this control unit.
- Processing Module Handle: By this handle, one processing module is bound with the control unit. The computation or the communication routines of the module can then be called by the control unit through this handle.
- Control Routine: As mentioned above, the control routine is the main routine of the thread. When the thread is created, the following cycle is executed to realize the controlling task.
    1. Sleep until being waken up.
    2. Call the routine of the bound module, using the processing module handle.
    3. When the routine of the bound module ends, wake up all of the child control units.
    4. Sleep until all child control units finish one cycle.
    5. Assign the input and the output for the bound module for the next cycle.
    6. Inform the parent control unit the cycle finishing of this unit.

Root Control Unit The structure of the root control unit is shown below.

- Children Handle List

  This is the same as the common control unit.
- Cluster Synchronization Routine

  These routines are used to synchronize the pipeline processing with other nodes in the PC cluster.
- Control Routine

  When the program starts, the following cycle is executed in the root control unit.
    1. Call the cluster synchronization routine.
    2. Wake up all of the child control units.
    3. Sleep until all child control units finish one cycle.

As the summary, the following mechanisms are introduced in the control module for the pipeline processing.

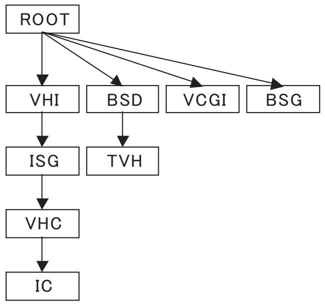

Figure 5. The Control Module for The Parallel Pipeline

- By allocating double memory blocks for each processing module and assigning them as the input and output alternatively, memory access conflicting in the concurrent programming is avoided.
- By organizing the multiple threads, i.e. the control units as a tree structure, the executing order can be easily controlled.

4.4. Control Module for the Parallel 3-Base-Plane Volume Intersection

Figure 5 shows the detailed control module designed for our system. The root control unit has 4 children units[1], which means when the processing cycle start, the 4 stages(“VHI”, “BSD”, “VCGI”, and “BSG”) will run concurrently. Notice that while the “BSD” stage is a communication stage, other 3 stages are computation stages. That is during the wait time caused by the “BSD”, the CPU will not be idle for the computation threads are running. Furthermore, by assigning the other communication stage “TVH” as the child of “BSD”, the network interface conflicts can be avoided. Meanwhile, by starting the communication stage(“BSD”) at the beginning of the processing cycle on each node, the communication stages of all nodes are synchronized and the wait time over nodes can also be shortened.

In what follows, the evaluation experiments are shown to prove the efficiency of the proposed model.

5. Performance Evaluation

In this section, some experiments are conducted to evaluate the performance of the parallel pipeline processing.

[1]The reason for the number of “4” will be described later in the next section.

5.1. Setup of Experiments

9 to 27 nodes PCs are utilized, and 9 of them are assigned as camera nodes, i.e. the nodes with camera connected. The hardware specification is shown in Table 2. The operating system is GNU/Linux of version 2.4.18.

Table 2. System Specification

PC	
CPU	Xeon 3.6Ghz Dual
Memory	2GByte
Camera (Sony DFW-VL500)	
Image size	VGA(640×480)
Frame rate	15 fps (Ext. Trigger)

As described above, by applying thread tree model, multiple stages can be executed concurrently. However, on a real system, only up to the number of CPUs threads can be executed exactly parallel. On our system, by enabling the Hyper-Threading feature, total 4 CPUs are detected by the OS, and total 4 threads can be executed exactly concurrently. So just 4 children units are assigned to the root control unit in Figure 5.

In the following evaluation experiments, we have total 9 camera inputs for acquiring the full 3D shape of a moving person. The number of PC nodes are changed from 9 to 27 sets.

5.2. Total Throughput

At first, the total throughput is measured, by setting the space resolution as 10[mm] and 5[mm][2]. The processing time for one frame is measured and the average of 50 frames is plotted in Figure6. From the graph, In both cases, the processing time is saturated at about 80[msec], i.e. the throughput is about 12 fps, which is limited by the capture rate of the camera. While the camera (SONY DFW-VL500) supports the external trigger mode for the shutter synchronization, the capturing rate is limited at 15 fps when having the external trigger mode on. Furthermore, because the data transfer timing is independent to the shutter timing, to synchronize the capturing process of multiple cameras, the total shutter interval is fixed at about 80[msec], i.e. about 12 fps.

5.3. Processing Time Analysis of the Pipeline Stages

For the detailed processing time inside one processing cycle, the time flow of each stage is measured. Figure 7

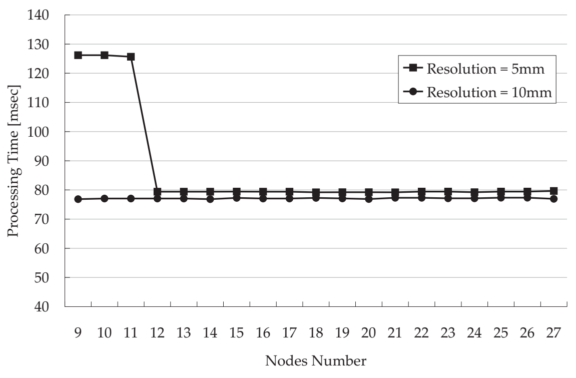

Figure 6. Total Throughput of Parallel Volume Intersection

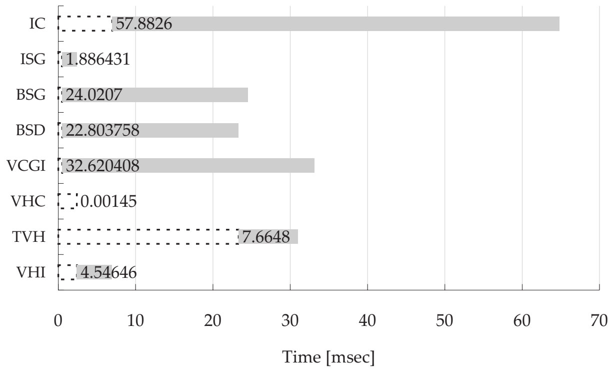

Figure 7. Snap Shot of Processing Time for Each Stage, total nodes number = 27, resolution = 5[mm]

shows one snapshot of the processing time of each stage within one cycle on one of the 27 PC nodes. The resolution is 5[mm]. In the figure, each flowing bar shows the start and the end timing of each stage. The data label on each bar is the interval of the stage in millisecond.

As mentioned before, by enabling the Hyper-Threading feature, up to 4 threads can be executed simultaneously, which is proved by the graph in Figure 7. From the graph, the "IC" time bar is the longest and dominates the total throughput. This proves that the reason of the saturation is the capturing rate of the camera.

Among the computational stages, the "VCGI" time bar is the longest, which is about 30 [msec]. That is, the potential speed at the resolution of 5[mm] is about 30 fps if no limitation of the capturing rate exists.

As a result, the computation rate is over the capturing

[2]That is, the voxel size is $10 \times 10 \times 10$ mm^3 and $5 \times 5 \times 5$ mm^3 respectively.

rate and the real-time computation of the full 3D shape is realized on our system.

6. Conclusion

In this paper, we describe the 3-base-plane volume intersection and its parallelization. By this method, the plane based volume intersection can be conducted with arbitrary camera layouts. This enables to carry out the volume intersection with an active camera task, for example, the 3D shape reconstruction while active tracking. For the parallel pipeline implementation, we proposed the thread tree control model. The high performance of the algorithm and the implementation model is proved by the evaluation experiments.

Furthermore, while one stage is implemented as one thread here, it could improve the throughput much better to split heavy stages into multiple threads. It is obvious that such extension can be achieved easily and dynamically by the thread tree control model.

References

[1] E. Borovikov and L. Davis. A distributed system for real-time volume reconstruction. In *Proc. of International Workshop on Computer Architectures for Machine Perception*, pages 183–189, Padova, Italy, Sept. 2000.

[2] G. Cheung and T. Kanade. A real time system for robust 3d voxel reconstruction of human motions. In *Proc. of Computer Vision and Pattern Recognition*, pages 714–720, South Carolina, USA, June 2000.

[3] R. T. Collins. A space-sweep approach to true multi-image matching. *IEEE Computer Vision and Pattern Recognition*, pages 358–363, 1996.

[4] T. Kanade, P. Rander, and P. J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, pages 34–47, 1997.

[5] I. Kitahara, Y. Ohta, and T. Kanade. 3d video display of sports scene using multiple video cameras. In *Meeting on Image Recognition and Understanding, vol 1*, pages 3–8, 2000.

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

[7] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. In *Proc. of SIGGRAPH 2000*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., July 2000.

[8] S. Moezzi, L. Tai, and P. Gerard. Virtual view generation for 3d digital video. *IEEE Multimedia*, pages 18–26, 1997.

[9] S. N., Y. Y., K. N., E. Y., and K. K. 3d modeling and displaying system for volume communication. In *4th International Symposium on Advanced Fluid Information and Transdisciplinary Fluid Integration*, pages 159–164, 2004.

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

[11] S. M. Seitz and C. R. Dyer. Toward image-based scene representation using view morphing. In *Proc. 13th Int. Conf. on Pattern Recognition, Vol. I*, pages 84–89, 1996.

[12] S. Sugawara, Y. Suzuki, G.and Nagashima, M. Matsuura, H. Tanigawa, and M. Moiriuchi. Interspace: Networked virtual world for visual communication. pages 1344–1349, December 1994.

[13] S. Vedula. *Image Based Spatio-Temporal Modeling and View Interpolation of Dynamic Events*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, September 2001.