1998 International Computer Symposium
Workshop on Computer Graphics and Virtual Reality
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

DISTRIBUTED VOLUME MORPHING

Leewen Lin[1], Chungnan Lee[1*] and Tongyee Lee[2]

[1]Institute of Computer and Information Engineering

National Sun Yat-Sen University

Kaohsiung, Taiwan, R.O.C.

[2]Department of Computer Science and Information Engineering

National Cheng-Kung University

Tainan, Taiwan, R.O.C.

Abstract

3D morphing is a popular technique for creating a smooth transition between two objects. In this paper we integrate volume morphing and rendering in a distributed network environment to speed up the computation efficiency. We describe the system architecture of distributed volume morphing and the proposed algorithms, along with their implementation and performance on the networked workstations. A load evaluation function is proposed to partition the work-load and the workstation cluster for load balancing and then to improve the performance under highly uneven load situation. The analysis of performance for five load balancing strategies are performed. Among them, the strategy 'Request' performs the best in terms of speedup.

1. Introduction

Volume morphing is a technique for generating smooth 3D image transformation between two objects. A source model is mapped to a target model by incrementally computing a function that converges the shape (and color) of the source to the target. It has been used in entertainment industry for a long time and can also be used as a tool for illustration and teaching purposes. Methods have been developed to deform various types of objects such as 2D polygons [1], 3D polyhedral models [2], 2D rasters [3][4], and 3D rasters [5][6].

Volume rendering, which often follows the morphed volume construction, is a method for producing an image from a 3D array of sampled scalar data. However, it is a computationally intensive application. Hence, many researchers use parallel computers to speed up the computation and attempt to make it more interactive. Volume morphing and rendering can be performed independently among different processors. Unfortunately, their computation time and inter-communication are irregular and unpredictable. Hence, it is necessary to come up with some strategies to achieve fast volume morphing and rendering in the distributed environment.

In this paper, we focus on distributed volume morphing with rendering process. Because of the large amount of volume data and the high computational cost, we introduce a master-slave structure to parallelize volume morphing. We propose and evaluate five strategies to achieve fast computation. A load evaluation function is used to predict the execution time of warping a volume for each frame. Based on prediction, the master can dynamically divide slaves into two groups for each frame in advance. Also, we use the prediction function to make volume partition for adaptive load balancing.

The remainder of the paper is organized as follows. Section 2 begins with a brief survey of previous volume morphing algorithms, volume rendering and parallel techniques. In Section 3 we evaluate five distributed volume morphing strategies and propose a load evaluation function. Section 4 describes the details and pseudo code for implementation. Section 5 gives the results of performance evaluation of five strategies. Finally, we conclude the paper with suggestions for future work in Section 6.

2. Prior Work

Feature-based volume morphing [5] creates every morphing in two steps, warping and blending as illustrated in Fig. 1.

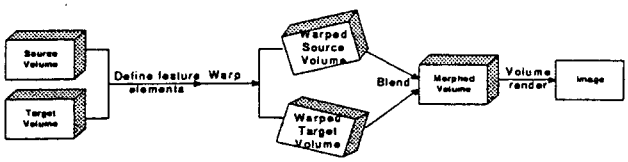

Fig. 1 The data flow of a morphing system.

The first step in the volume morphing pipeline is to warp the source and target volumes S and T into volumes S and T. The animator identifies two corresponding features in S and T, by defining a pair

of elements (e_s, e_t). These features should be transformed from one to the other during the morph. In feature-based morphing, elements come in pairs, one element in the source volume S, and its counterpart in the target volume T.

Two general types of task partitions for parallel volume rendering algorithms are object partitions and image partitions. In an object partition each processor is assigned a specific subset of the volume data to resample and composite [8][9]. The partial images from each processor must then be composited together to form the image. In contrast, in an image partition each processor's task is to compute a portion of the image [9]. Each image pixel is computed by only one processor.

Different from previous work in parallel rendering, there are two novelties in our research. First, we parallelize volume morphing that is not reported in the literature so far. Furthermore, volume morphing, which is a pipeline work of warping, blending, and rendering, is much more complicated than volume rendering. We must synchronize all these phases of pipeline to achieve a better speedup. In this case the object partition is used to assign tasks rather than image space. Because of the warped volume transmission at the blending phase, it needs more communication among slaves than that of parallel volume rendering.

3. Parallel Algorithms for Volume Morphing

In this section, we describe the parallel computation of feature-based volume morphing. The computation is performed on a clustered-network environment by running PVM as parallel computing platform [10]. To achieve a better load balancing, we propose a load evaluation function to evaluate workload before assigning slaves into two groups: source and target groups, dynamically. This function is also used to make volume partition for load balancing of tasks.

3.1 The Master-Slave Model

The overall structure of the distributed volume morphing system is a master-slave architecture as illustrated in Fig. 2. There are one *master* node and several *slave* nodes.

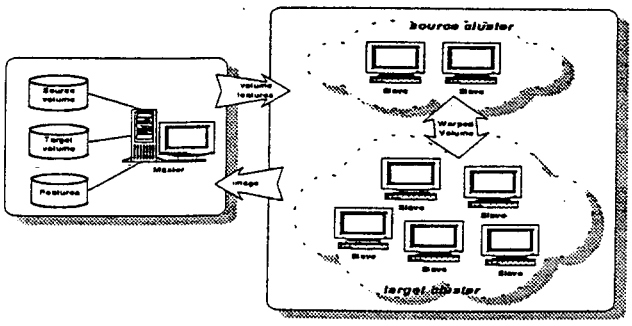

Fig. 2 The Master-Slave Model.

The jobs of the slaves are completely dependent on the master. The master divides all the slave nodes into two *clusters* for each frame. One cluster is responsible for the source and the other is responsible for the target volume warping. For example, the source cluster warps the source volume into intermediate volume and the target cluster does the target one for the first frame. For the next frame the role for the slaves in two clusters may be swapped to achieve a better load balancing. At the beginning of each frame, the master must pass the features and the volume data to every slave, and send the information of one cluster, like task id and partition table, to the other cluster. When the slave nodes receive these messages, they can begin to warp, blend, classify, and render. Meanwhile, the master waits for the rendered images, prepares the next frame information, and sends more data requested by the slaves. The final images are saved to the file system by the master node.

To reduce the amount of work in the next stage, the slaves in a smaller cluster will send its warped volume to the slaves in the other cluster for blending when they finished their warping job. For example, suppose that the source cluster consists of 3 slaves and the target cluster consists of 4 slaves, then the source cluster must send the warped volume to the target cluster. In this way, we can make the volume size for classification as small as possible. When a target slave receives the warped source volume needed for blending from the source slaves, it continues to do blending, classification, and rendering. After it finishes the rendering work, it sends the result of partial rendering image back to the master node and requests more information about the next frame.

The state transition diagram for the processing of all stages of slaves is illustrated in Fig. 3. The items marked on the arrows show the flow of the data needed for the next state. The sequence of execution on these arrows is also marked. There are 7 states for slaves. Not every slave must walk through all the states for each frame. If the slaves are in the smaller cluster, they do not have to do blending, classification, and rendering. Those slaves who send the warped volume to the correspondent slaves, they can request the master node to send a new job. So they can continuously work on the new frame without waiting for the other cluster.

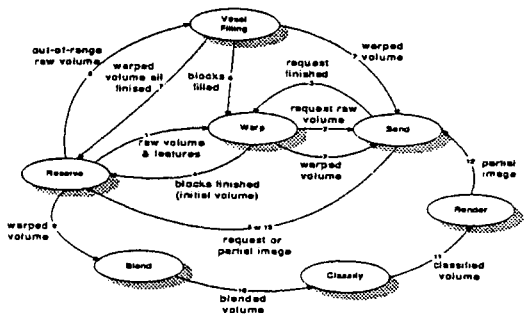

Fig. 3 State transition diagram for the slave nodes.

The states for the slave node are described as follows:

- *Receive*: At the beginning of each frame each slave must receive the morphing information, such as feature sets, task size, volume-partition table, etc., from the master. When the slave receive those data, it switches to the "warp" state.
- *Warp*: A slave warps the block volume dispatched by the master in this phase. If the slave in the larger group, it will receive all the correspondent warped volume from the other group when it finishes the warping job and then starts its blending state. Otherwise, it will send its own warped volume to those slaves responsible for the same position.

A slave may send several requests to the master to ask for more raw data other than it owns in local memory, then it goes back to warp the unfinished data. Hence, the local machine has more flexibility on using the local memory. After it finishes another block, it tries to listen whether the requested data are arrived. If they are, it comes back to the "receive" state to receive them and transits to fill the empty voxels.
Send: A slave will change to the "send" state when it needs to send messages or data to the master or the slaves under the following conditions:

1. To request more raw volume from the master.
2. To send warped volume to other slaves.
3. To send rendered partial image to the master or to request a new task.

- *Voxel Filling*: When the slave receives the raw volume sent by the master, it begins to compensate the empty warped voxels. As soon as it finishes, it goes back to warp next blocks.
- *Blend*: Only the slaves in the larger group have to do the blending job. A slave begins to blend two partial warped volumes, when it receives all the other warped volume at the same slice position as illustrated in Fig. 4. In Figure 4, there are 3 tasks in source group and 5 tasks in the target group. A slave in the source cluster may need to send its own warped volume, such as task 0 in the source, to more than one slave in the target cluster, such as task 0 and task 1 in the target. Similarly, a slave in the target, such as task 3 needs to receive the warped volume from more than one slave such as task 1 and task 2 in the source before it can proceed blending. We use a linear weighted function $w(t)$ to interpolate their voxel values. Again, those volumes may come from different slaves. At the next step, it renders the partial image.

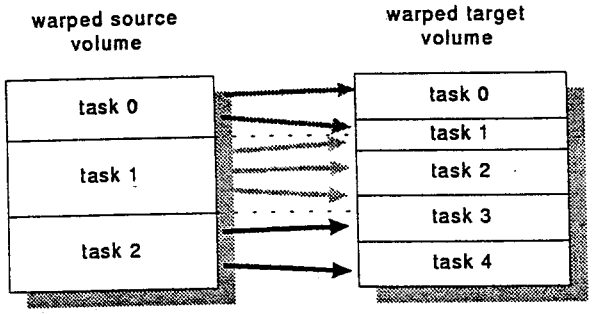

Fig. 4 Sending the warped volume from and to the source and the target groups for blending

- *Classify*: Classifying the blended volume follows the blending state. In this phase, a slave traverses the blended volume in storage order, computes the opacity of each voxel, and then compares each voxel's opacity to determine if it is transparent or non-transparent. By this way, the slave constructs the run-length encoded volume for rendering.
- *Render*: At the last stage, the slave renders the partial image depending on the classified volume. It computes the shear and warp factors of the viewing transformation matrix, and composites each slice of the volume into the intermediate image in front-to-back order. Finally, the slave warps and sends the partial image to the master and the next frame begins.

3.2 Load Balancing Schemes

The sequential morphing algorithm contains three phases: warping and blending the source and target volumes, classifying the blended volume, and rendering the image, each of which can be parallelized. However, from Table. 1, we can find that the warping phase dominates the execution time of the sequential algorithm. So we focus on parallelizing the warping of the two volumes.

Table. 1 The execution time for each phase of sequential morphing algorithm (unit in second) (volume size: 128*128*84, image size: 256*256)

Task \ Frame	1	2	3	4	5
Warp Brain	5.131	16.670	41.150	73.692	102.048
Warp Sphere	136.044	81.578	39.526	15.194	6.238
Blend	0.785	0.697	0.689	0.790	0.696
Classify	15.248	14.087	13.689	12.247	8.845
Render	1.348	1.037	1.141	1.035	0.563
Total	158.555	114.067	96.194	102.957	118.389

We attempt to parallelize the morphing algorithm using an object partition in which each slave is assigned a portion of voxels partitioned in y-direction as shown in Fig. 5.

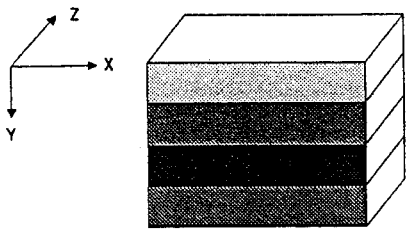

Fig. 5 Volume partition in y direction

When a volume is subdivided into subvolumes, each subvolume must overlay at least 2 slices with its neighboring subvolumes to avoid ray samples error. The partial images produced by slaves are sent to the master node and placed in the correct position by the master. However, the viewing direction is not considered yet.

We discuss the four strategies in the next four Subsections and the strategy "Request" in Section 4.

3.2.1 Strategy 1: Even-Group & Even-Partition

This is the simplest method among the four strategies. In this strategy, the master divides the slaves into two groups with the same number of slaves. Then it dispatches the same size of volume for each slave to warp (Fig. 6).

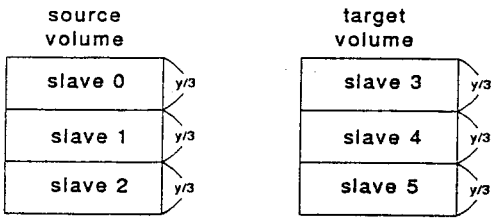

Fig. 6 Illustration of Strategy 1

3.2.2 Strategy 2: Adaptive-Group & Even-Partition

The strategy 1 does not consider the difference of the warping time between frames as listed in Table 2. So, the strategy 2 is to group the slaves based on the load of the source and target volume (see Fig. 7). Then, these slaves in the same group will obtain the task with the same volume size for warping. We use the variation sum of each feature pair to predict the loads of the two volumes. This function is described as follows.

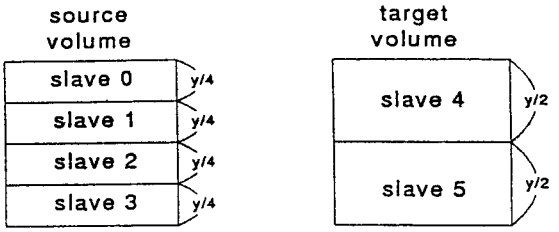

Fig. 7 Illustration of Strategy 2

- **Load Evaluation Function**

At each frame, the time for warping the source and the target volumes may not be the same due to different computational complexity for the source and the target volumes. If we evenly divide slaves into two groups, there must be a considerably waiting time for the slowest slave to finish its job. Table. 2 shows the warping time for source and target volumes for five consecutive frames. As one can see that the warping time is different from frame to frame. At the first frame, the ratio of the warping time between the brainsmall dataset and the sphere dataset can be as small as 0.0378, but at the fifth frame the ratio of warping time between two datasets is as large as 16.3465.

Table. 2 The warping time for warping volumes for 5 frames

(volume size: 128*128*84, 38 sets of features)

Frame / Data	1	2	3	4	5
Brainsmall	5.145	17.532	41.181	74.136	102.051
Sphere	136.001	82.855	39.530	15.678	6.243
Ratio	0.0378	0.2116	1.0418	4.7287	16.3465

Under the circumstances, the load imbalance will degrade the performance of the algorithm. Hence we propose a load evaluation function for each line feature pair to predict the load of warping a volume as follows:

$$load(e_1, e_2) = \overline{c_1 c_2}^* \mid s_1 - s_2 \mid \qquad (1),$$

where $\overline{c_1 c_2}$ is the translating distance of a set of features, s_1 and s_2 are the lengths of the source and target features, respectively. They represent for the variance of the feature pair. If a pair of line segments keep consistent, voxels near the segments may stay at its original position so that they need more interpolation instead of inverse mapping by all features. Based on the load evaluation function, we add the loads of all pairs for both the brainsmall and the sphere datasets at each frame and use the proportion of two datasets to divide slaves into two groups. The calculated data are listed in Table. 3. Now as one can see the ratio predicted by the load function is similar to that of Table. 2.

Table. 3 The ratio determined by the load evaluation function

Frame / Data	1	2	3	4	5
Brainsmall	70.110	197.374	382.707	625.19	926.192
Sphere	926.844	626.206	384.052	199.245	70.281
Ratio	0.0756	0.3152	0.9965	3.1378	13.1784

In order to achieve a better load balancing, the master uses the following equation to determine the number of slaves in these two clusters for each frame.

$$Source\_slaves\#=total\_slaves\#*[source\_load/(source\_load+total\_load)] \quad (2),$$

where the *source_load* is the load of source calculated by equation (1) and the *total_load* is the sum of the source load and the target load. Suppose there are 10 slaves, the number of slaves for each cluster assigned by the master is illustrated in Table. 4. Then the master can make dynamic partition of each volume for slaves using the same proposed load evaluation function.

Table. 4 The number of slaves of two clusters for each frame. Ten slaves are divided into two groups using the load evaluation function.

Frame / Data	1	2	3	4	5
Brainsmall	1	2	5	7	9
Sphere	9	7	4	2	1

Because we interpolate the source and target features at each frame, so the intermediate feature is given by

$$e = \frac{f}{tot\_frame+1}e_s + \frac{tot\_frame+1-f}{tot\_frame+1}e_t \quad (3),$$

where f is the index number of the current frame, and tot_frame is the number of total frames that the user wants to produce.

Assume all features are line segments, the percentage that the source volume consititutes at frame f can be written as

$$Source\_Load(f) = \frac{f^2}{f^2+(tot\_frame+1-f)^2} * 1 \quad (4)$$

Fig. 8 shows the relation between the percentage of execution time and the load predicted using equation (4) of three examples which use 19, 24, and 38 sets of features over 10 frames. The trends of the four curves are almost the same. We can see that in the middle frame the percentages of four curses are almost the same. And at the beginning and ending frame, the percentages for these three examples are just a little more or less than that of the predicted one.

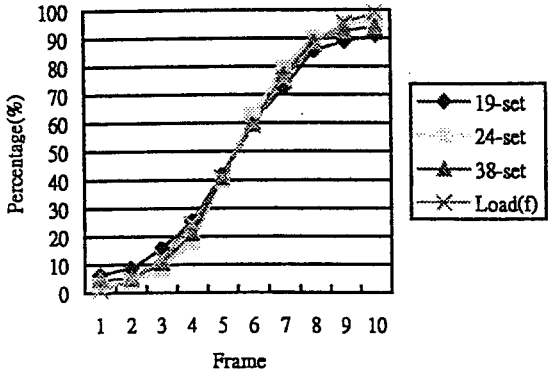

Fig. 8 The relationship between the percentage of execution time and the load predicted using equation (4) for three examples (features: 19, 24, and 38 sets).

3.2.3 Strategy 3: Adaptive-Group & Adaptive-Partition

Different from Strategy 2, this strategy adopts an adaptive volume partition because the distribution of features is not even, and the warp-load of each sub-volume may not be equal. We use the load evaluation function described in Subsection 3.2.2 to predict the load of each slice in y-direction. First, we compute the load of each feature element and find which slices the element crosses. Next, the average load of each element is added to the correspondent slices' loads and a decreasing load, which is defined as $\frac{Load}{slices * d^2}$, is added to the slices neighboring to those the element crosses. Finally, the master decides the partition by the average of the load. Fig. 9 illustrates this strategy. Each slave is assigned a task of different size.

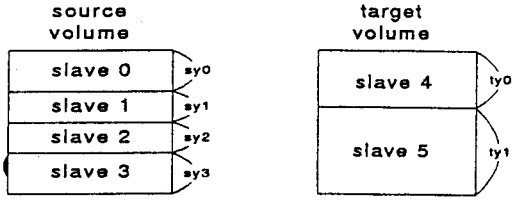

Fig. 9 Illustration of Strategy 3

3.2.4 Strategy 4: Hybrid

In the three strategies mentioned above, each slave warps only data in the source volume or the target volume. But at some frames, the variation in one volume is so small such that the slave's load is still too small. Under the circumstances, the master will assign the slave more task from the other volume. The loads of two volumes are added to be a total load. The adaptive partition is still used and volumes are equally partitioned by the average of the total load. If the slave has two tasks at one frame, it requests the other when it finishes the first one. Fig. 10 shows that the *slave 0* is responsible for the whole source volume and a small portion of the target volume.

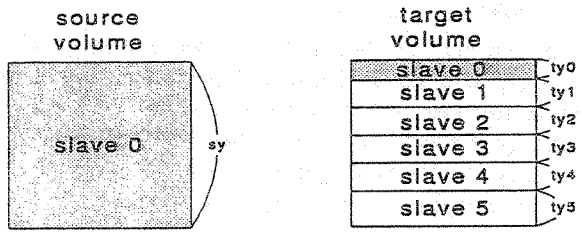

Fig. 10 Illustration of Strategy 4

4. Implementation

To further decrease the waiting time at the end of each frame, the slave can make request for next task as soon as it finishes the current one. The master will compute the intermediate features, group slaves, and make volume partition after it has sent all of the information about the current frame. Then it waits for the slaves' responses and sends the information of the next frame to the slaves. The slaves can warp each frame at different time to save unnecessary waiting time. However, for the synchronization reason, the master sends the synchronization signal to inform slaves to make sure that all slaves work at the same frame. Thus, the slave who warps first will not send the warped volume to the wrong frame. Incorporating the strategy 4 with the mechanism described above it becomes the strategy "Request".

In the distributed computing environment, the communication cost can be a dominant factor in the morphing process. So we send the source and target raw volumes to all the slaves at the beginning, each slave holds these data till all tasks are done.

In volume rendering, we adopt Locroute's Volpack library of fast volume rendering that uses a shear-warp factorization of the viewing transformation [7]. It combines the advantages of ray casting and splatting algorithms and is said to be the fastest volume rendering method in the literatures so far. Locroute chooses the shear transformation such that the viewing rays become perpendicular to the slices of the volume. The shear is implemented by translating and resampling each slice of volume data. Projection is then trivial: the resampling slices are combined together in front-to-back order using the "over" operator to form an intermediate image. Finally the intermediate image must be transformed into the correct final image by applying an affine 2D warp. The warp is relatively inexpensive, because it operates on 2D images rather than the 3D volume data.

5. Performance Evaluation

We implement the algorithm of distributed volume morphing using the C language and the PVM (Parallel Virtual Machine) platform [10] on a network computing environment with SUN SPARC5 workstations. To run a program under PVM, the user first executes a daemon process on the local host machine, which in turn initiates daemon processes on all other remote machines. Then the user's application program, which should reside on each machine. Communication and synchronization among these user processes are controlled by the daemon processes. Unlikely a shared-memory multiprocessor, the communication overhead of the network environment must be handled carefully.

The two datasets we used in these experiments are a brainsmall and a sphere volume data. The sizes of them are 128*128*84 (x*y*z). We define 38 sets of features for each volume to produce the morphing sequences. Among 38 sets of features, there are 14 points and 24 segments. Fig. 11 shows only the first, the middle and the last animation results of this example.

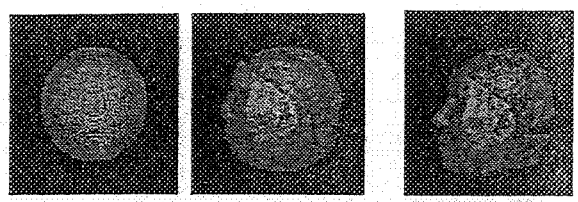

Fig. 11 The animation sequence of the sphere to the brainsmall

- Speedup:

Fig. 12 shows the speedup for four strategies and strategy *'Request'* described in Section 4. Among five strategies, the strategy *'Request'* achieves the best speedup of 6.255 when 10 slaves are used.

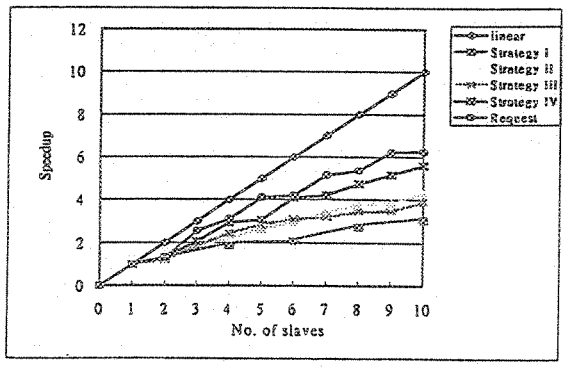

Fig. 12 Speedup for the average time of 5 frames.

Table 5 shows the parallel efficiency for each strategy using different slaves. The parallel efficiency is defined as follows:

$$parallel\_efficiency = \frac{Speedup}{Speedup_{linear}} \quad (5)$$

- **Efficiency:**

We use a load balance equation to evaluate the warping time. The load balance (or efficiency) of a computation is the ratio of the average computer load to the maximum computer load,

$$eff = \frac{L_{avg}}{L_{\max}} \quad (6)$$

where L_{avg} is the average warping time of all slaves, and $L_{\max}$ is the maximum one of all slaves at each frame. The efficiency results using 5 and 10 slaves are listed in Tables 6 and 7. Of all four strategies, strategy IV achieves the highest efficiency in the two tables. It shows our volume partition method is better than even partition. Strategy III has good efficiency at frame 2, 3, and 4, but worse efficiency at frame 1 and 5. It is the reason that only one slave responsible for the source or target volume and their loads are very light at these 2 frames.

Table 5. The parallel efficiency for all strategies (%).

Slaves / Strategy	2	3	4	5	6	7	8	9	10
I	57.80	-	47.30	-	34.40	-	34.11	-	30.42
II	57.80	62.80	54.43	52.84	48.93	48.91	47.39	42.89	43.28
III	58.60	61.97	61.20	57.04	52.03	45.54	43.14	38.68	39.07
IV	65.80	69.20	73.35	61.52	68.22	60.36	59.29	57.56	56.02
Request	66.37	84.82	77.58	81.89	70.26	73.49	67.07	69.23	62.56

Table. 6 The efficiency of four strategies using 5 slaves.

Frame / Strategy	I	II	III	IV
1	-	0.6353	0.5389	0.8938
2	-	0.6193	0.9116	0.8644
3	-	0.6303	0.7388	0.7114
4	-	0.6103	0.7591	0.7655
5	-	0.6007	0.6096	0.8136

Table. 7 The efficiency of four strategies using 10 slaves.

Frame / Strategy	I	II	III	IV
1	0.4045	0.6853	0.4086	0.6903
2	0.3372	0.5634	0.7657	0.7950
3	0.4780	0.4813	0.7368	0.7394
4	0.3577	0.5585	0.7591	0.7732
5	0.3422	0.6148	0.5468	0.8290

Furthermore, the warping efficiencies in Tables 6 and 7 are better than the speedup drawn in Fig. 12. It may result from two reasons:

We only take consideration about the load of warping time. However, the partition size will influence the classification time too. A larger partition will take longer classification time. When the number of slaves increases, the classification time becomes more dominant.

The slaves must wait for the warped volumes awhile from other slaves for blending. It also affects the total time.

- **Waiting time:**

We compare the waiting time between strategy IV and strategy *'Request'* in Table 8. The average time of each slave for each frame is about 2.32 seconds using the strategy *'Request'*. It is less than 10.65 seconds of the strategy IV. The waiting time (2.32 sec) constitutes 12.3% of the average time (18.87 sec) of one frame.

Table. 8 The total waiting time of each slave for 5 frames using strategy IV and strategy *'Request'*. And the average time of each slave at each frame is also computed.

Slave / Strategy	1	2	3	4	5
Request	1.6799	3.3793	11.872	19.455	11.772
IV	19.309	38.854	46.909	64.418	55.606

Slave / Strategy	6	7	8	9	10	Average
Request	13.378	10.094	15.286	19.357	9.8751	2.323044
IV	67.725	54.604	62.029	74.908	47.938	10.6461

- **Time Analysis:**

Combining the contributions of all phases, we find the total execution time for the algorithm:

$$t = t_{warp} + t_{blend} + t_{classify} + t_{render} + t_{comm} + t_{wait} \quad (7)$$

where

t_{warp} = total time used in warping

t_{blend} = total time used in blending the two warped volume

$t_{classify}$ = total time used in classifying the blended volume

t_{render} = total time used in rendering the partial image

t_{comm} = total time taken for inter-processor

communication, including transmission and reception of partial volumes and partition data

t_{wait} = total time taken for waiting

Tables 9 and 10 show the analysis of the total execution time of 10 and 5 slaves into its components.

Table. 9 The analysis of time taken for various tasks using 10 slaves.

	t	t_{warp}	t_{blend}	$t_{classify}$	t_{render}	t_{comm}	t_{wait}
Time(sec)	870.67	617.81	2.472	85.145	14.125	14.114	136.96
Per(%)	100	70.963	0.283	9.7793	1.6218	1.6216	15.75

Table. 10 The analysis of time taken for various tasks using 5 slaves.

	t	t_{warp}	t_{blend}	$t_{classify}$	t_{render}	t_{comm}	t_{wait}
Time(sec)	733.39	571.188	2.2841	63.052	6.8479	6.3761	83.649
Per(%)	100	77.882	0.3114	8.5972	0.9337	0.8694	11.405

From Table 9, we observed that the communication time contributes only 1.62% of the total parallel morphing operation. Compared with the warping time of the volumes by all slaves which takes up 70.96% of the total time, the communication is not the main factor to improve time efficiency in our distributed morphing algorithm. However, a better partition scheme may help to reduce the total waiting time that amounts to 15.73%. Moreover, the classification time should be also taken into consideration in order to further reduce the waiting time.

6. Conclusions

In this paper, we have presented a parallel volume morphing algorithm for a networked cluster of workstations. The algorithm divides the computation load of warping across all processors by the load evaluation function. Based on the proposed function we could predict the ratio of the warping times for the source and the target volume to improve the efficiency. This function was used to divide slaves into two clusters and make partition of the volumes. The slaves can work on the next task without waiting for the other when warping the volume of new frame. We have evaluated the performance of five strategies. The results show that the strategy "Request" performs the best.

Performance could be further improved by considering the interpolating time for the blocks without features. Alternatively, the classification time becomes more dominant when the number of slaves increased. To morph a large dataset of volume may result in the memory shortage, we can solve the problem by using a cache strategy.

References

[1] T. W. Sederberg and E. Greenwood, "A Physically Based Approach to 2-D Shape Blending," In *Proceedings of SIGGRAPH '92, Computer Graphics*, vol. 26, pp. 25-34, July 1992.

[2] J. R. Kent, W. E. Carlson, and R. Parent, "Shape Transformation for Polyhedral Objects, " In *Proceedings of SIGGRAPH '92, Computer Graphics*, vol. 26, pp. 47-54, July 1992.

[3] T. Beier and S. Neely, "Feature-Based Image Metamorphosis," In *Proceedings SIGGRAPH '92*, volume 26, pp. 35-42, July 1992.

[4] G. Wolberg, Digital Image Warping, IEEE Computer Society Press, Los Alamitos, Calif., 1990.

[5] A. Lerios, C. Garfinkle, and M. Levoy, "Feature-Based Volume Metamorphosis," In Proceedings SIGGRAPH '95, pp. 449-456, 1995.

[6] T. He, S. Wang, and A. Kaufman, "Wavelet-Based Volume Morphing, " In D. Bergeron and A. Kaufman, editors, Proceedings of Visualization '94, pp. 85-91, Los Alamitos, CA, Oct. 1994. IEEE Computer Society and ACM SIGGRAPH.

[7] P. Lacroute, Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. PhD thesis, Stanford University, 1995.

[8] K-L Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh, "A Data Distributed, Parallel Algorithm for Ray-Traced Volume Rendering," In Proceedings of the 1993 Parallel Rendering Symposium, San Jose, Oct. 1993, pp. 15-22.

[9] J. Nieh and M. Levoy, "Volume Rendering on Scalable Shared-Memory MIMD Architectures," In Proceedings of the 1992 Workshop on Volume Visualization, Boston, pp. 17-24.

[10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, PVM: Parallel Virtual Machine- A Users' Guide and Tutorial for Networked Parallel Computing, The MIT Press, 1994.