Cluster Computing 2 (1999) 219–227

Distributed volume morphing

Leewen Lin [a], Chungnan Lee [a] and Tong-Yee Lee [b]

[a] *Institute of Computer and Information Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan*
[b] *Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan*

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 our proposed 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 workload and the workstation cluster for better load balancing and then to improve the performance under highly uneven load situation. The performance evaluation for five load balancing strategies are conducted. 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 [9], 3D polyhedral models [4], 2D rasters [1,10], and 3D rasters [3,6].

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 make interactive feasible. Though volume morphing and rendering can be performed independently among different processors, 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 five strategies to distribute the load 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 the 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 propose a load evaluation function and describe five distributed volume morphing strategies and then evaluate their performance. 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 [6] creates every morphing in two steps, warping and blending as illustrated in figure 1. 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 object partition each processor is assigned a specific subset of the volume data to resample and composite [7,8]. The partial images from each processor must then be composed together to form the image. In contrast, in image partition each processor is only responsible for a portion of the image [8].

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 phases of pipeline to achieve a better speedup. In this study the object partition is used to assign tasks rather than image space, because it needs more communication for the warped volume transmission at the blending phase among slaves than that of parallel volume rendering.

© Baltzer Science Publishers BV

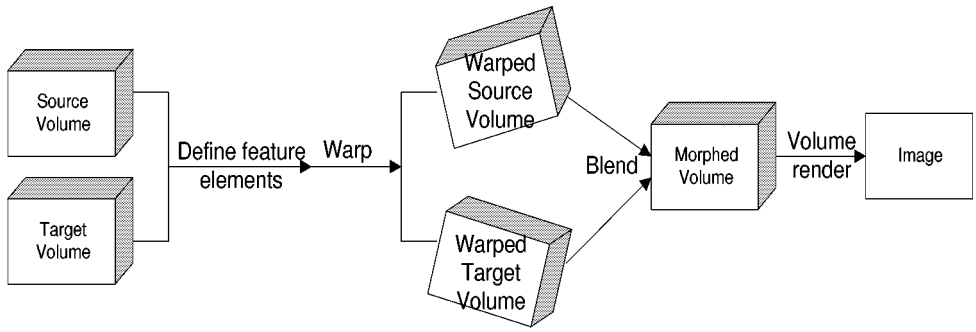

Figure 1. The data flow of a morphing system.

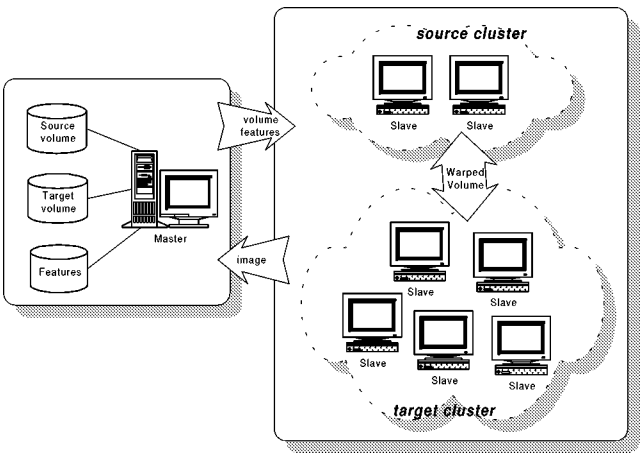

Figure 2. The master-slave model.

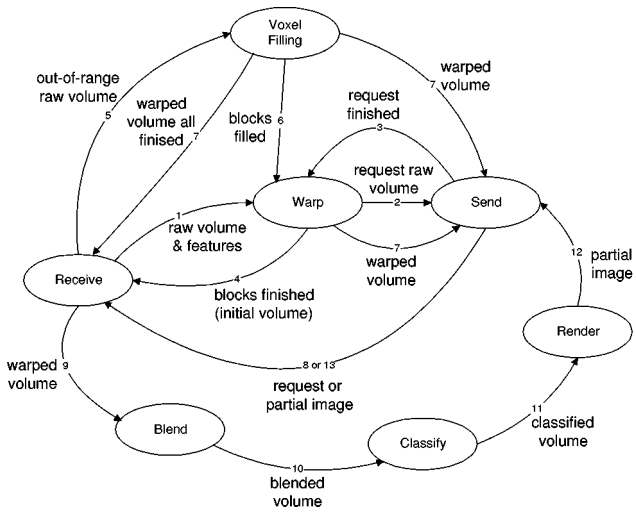

Figure 3. State transition diagram for the slave nodes.

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 the PVM system [2]. 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 do 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 figure 2. There are one *master* node and several *slave* nodes.

The job assignments for 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 volume warping 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 small 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 reduce 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 for the next frame.

The state transition diagram for all stages of volume morphing on the slave nodes is illustrated in figure 3. The items marked with the arrows show the flow of the data needed for the next state. The execution sequence on these arrows is also numbered. There are 7 states for slaves. Not

every slave must walk through all the states for each frame. If the slaves are in the small cluster, they do not have to do blending, classification, and rendering. Those slaves who send the warped volume to the corresponding slaves 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.

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 receives 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 is in the large group, it will receive all the corresponding warped volume from the other group. When it has finished the warping job, it then starts its blending state. Otherwise, it will send its own warped volume to those slaves responsible for blending.
  A slave may send 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 in using its local memory. After it finishes another block, it tries to listen whether the requested data have arrived. If they have arrived, 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 events:
  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 block.

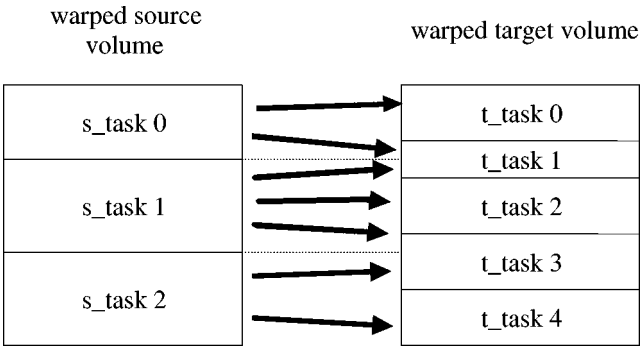

Figure 4. Sending the warped volume from the source to the target groups for blending, or vice versa.

- *Blend*: Only the slaves in the large group have to do the blending job. A slave begins to blend two partial warped volumes, when it receives all the other warped volumes at the same slice position as illustrated in figure 4. In figure 4, there are 3 tasks in the 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 *s_task* 0 in the source, to more than one slave in the target cluster, such as *t_task* 0 and *t_task* 1 in the target. Similarly, a slave in the target, such as *t_task* 3, needs to receive the warped volume from more than one slave such as *s_task* 1 and *s_task* 2 in the source before it can proceed with 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.
- *Classify*: Classifying the blended volume follows the blending state. In this phase, a slave traverses the blended volume in a storage order, computes the opacity of each voxel, and then compares each voxel's opacity to determine if it is transparent or non-transparent. In 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 a front-to-back order. Finally, the slave warps and sends the partial image to the master.

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 is a dominant factor for the computation. So we focus on parallelizing the warping of the two volumes.

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

When a volume is subdivided into sub-volumes, each sub-volume must be overlaid at least 2 slices with its neighboring sub-volumes to avoid ray samples error. The partial images produced by slaves are sent to the master node and

Table 1
The execution time for each phase of sequential morphing algorithm (unit in second) (volume size: $128 \times 128 \times 84$, 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	4.087	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

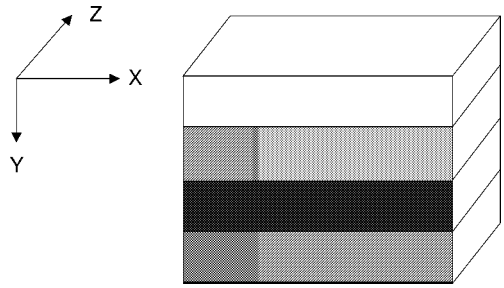

Figure 5. Volume partition in the y-direction.

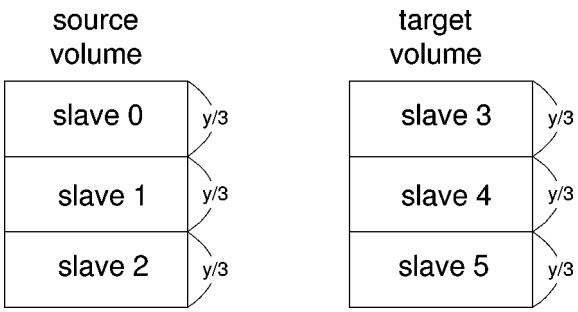

Figure 6. Illustration of strategy 1.

placed in the correct position by the master. Currently, the viewing direction is not considered.

We discuss four strategies: even-group and even-partition, adaptive-group and even-partition, adaptive-group and adaptive-partition, and hybrid, 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 as shown in figure 6.

3.2.2. Strategy 2: Adaptive-group & even-partition

The strategy 1 does not consider the difference of the warping time for different frames. As listed in table 2, it shows the warping time for the source and the target volumes for five consecutive frames. As one can see, 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. So, the strategy 2 is to group the slaves based on the load of the source and the target volumes (see figure 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.

Load evaluation function. At each frame, the time for warping the source and the target volumes may not be

Table 2
The warping time for warping volumes for 5 frames (volume size: $128 \cdot 128 \cdot 84$, 38 sets of features).

Data \ Frame	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

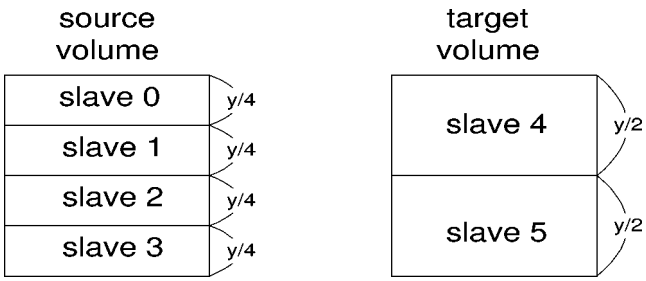

Figure 7. Illustration of strategy 2.

Table 3
The ratio predicted by the load evaluation function.

Data \ Frame	1	2	3	4	5
Brainsmall	70.110	197.374	382.707	625.191	926.192
Sphere	926.844	626.206	384.052	199.245	70.281
Ratio	0.0756	0.3152	0.9965	3.1378	13.1784

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 considerable waiting time for the slowest slave to finish its job.

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:

$$\mathit{load}(e_1, e_2) = \overline{c_1 c_2}|s_1 - s_2|, \tag{1}$$

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. $|s_1 - s_2|$ represents the variance of the feature pair. If a pair of line segments keep consistent, voxels near the segments may stay at their 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 ratio 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.

The master uses the following equation to determine the number of slaves in these two clusters for each frame:

$$\begin{aligned} &\mathit{Source\_slaves\#} \\ &\quad = \mathit{total\_slaves\#} \\ &\qquad \times \big[\mathit{source\_load}/(\mathit{source\_load} + \mathit{total\_load})\big], \end{aligned} \tag{2}$$

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

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

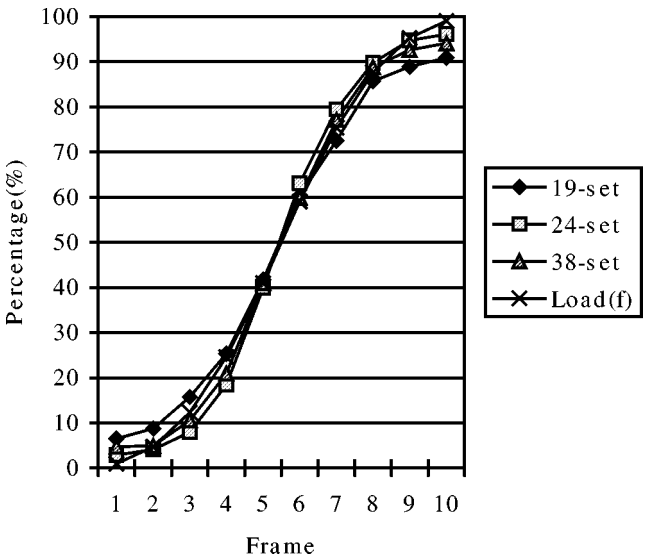

Figure 8. The relationship between the percentage of execution time and the load predicted using equation (4) for three examples with features: 19, 24, and 38 sets of features, respectively.

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 for each frame is illustrated in table 4. Then the master can do dynamic partition of each volume for slaves using the proposed load evaluation function.

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 that all features are line segments; the percentage for the source volume contribution to the total load at frame f can be written as

$$Source\_Load(f) = \frac{f^2}{f^2 + (tot\_frame + 1 - f)^2} \cdot 100\%. \quad (4)$$

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

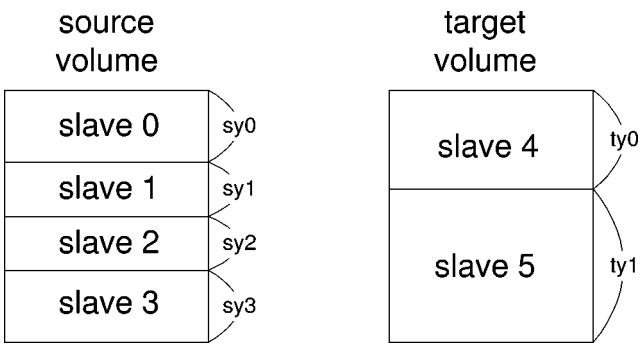

Figure 9. Illustration of strategy 3.

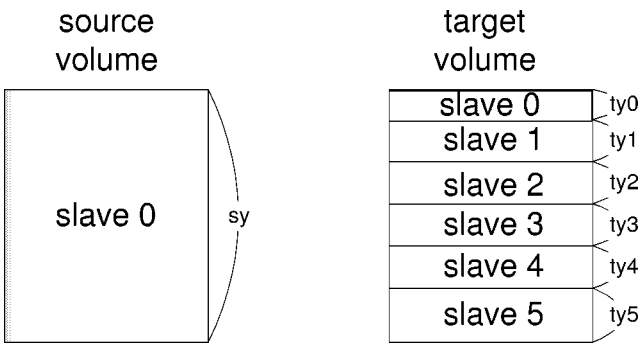

Figure 10. Illustration of strategy 4.

3.2.3. Strategy 3: Adaptive-group and 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 section 3.2.2 to predict the load of each slice in the 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 $Load/(slices \cdot d^2)$, is added to the slices neighboring to those the element crosses. Finally, the master decides the partition based on the average of the load. Figure 9 illustrates this strategy. Each slave is assigned a task of different size.

3.2.4. Strategy 4: Hybrid

In the three strategies mentioned above, each slave warps only data in either 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 light. 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. Figure 10 shows that the *slave* 0 is responsible for the whole source volume and a small portion of the target volume.

4. Implementation

To further decrease the waiting time at the end of each frame, the slave can make request for the 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 finishes sending all 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 the 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 [5]. It combines the advantages of ray casting and splatting algorithms. 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. The re-sampling slices are combined together in the 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.

Figure 11 is a pseudo-code description of the algorithm for the master node. It first sends the raw data of source and target volumes to all slaves and dispatches tasks by the load evaluation function. Then the outer loop iterates over the frames. The master synchronizes slaves after it sends the intermediate features to all slaves. While it waits for messages, it prepares the intermediate features and partitions for the next frame. The inner loop iterates for the number of tasks. Based on the received message, the master determines whether to send the task of the current or the next frame back. After all tasks send messages back, this frame is completed.

The pseudo-code in figure 12 shows the flow control for each slave in the distributed morphing algorithm. First, every slave receives the raw volumes from the master. Every slave loops through the total frames. For each dispatched task, the slave receives the features and partition

```
Send source and target raw volumes to all slaves;
Compute the 1st frame's intermediate feature set;
Evaluate the load of each slice of two volumes;
Do slave & volume partition and dispatch tasks for slaves;
For each frame do the following procedures till all frames are done
{
  Send synchronization signal to slaves;
  Compute the next frame's intermediate feature set;
  Evaluate the load of each slice of two volumes of the next frame;
  Do slave & volume partition for the next frame;
  For ( j=0 ; j<source_task+target_task ; j++ )
  {
    Receive message from slaves;
    Switch ( message )
  Case task_id :
    Dispatch other jobs of this frame to slaves;
  Case image or slave# :
    Dispatch the job of  the next frame to slaves;
  }
}
```

Figure 11. Pseudo-code for the master node.

```
Receive source and target volumes from the master;
For each frame do the following procedures till all frames are done
{
 Receive the number of jobs (warp_job) at this frame;
 For ( j=0; j<warp_job; j++ )
 {
    Receive all the information for warping;
    Warp Volume;
    Receive the synchronization signal from the master;
    If ( the slave belongs to the large group)
    {
      Receive all the needed warped volume from the other slaves
      in the small cluster;
      BlendVolume;
      ClassifyVolume;
      RenderVolume;
      Send the rendered image to the master;
    }
    else
    {
      Send warped volume to the corresponding slaves
      in the other cluster;
      Request the next job from the master;
    }
 }
}
```

Figure 12. Pseudo-code for the slave node.

table of volumes, and warps a portion of the volume. After it receives the synchronization signal from the master, it can transfer or receive the warped volume. If the slave belongs to the large group, it must receive the same size of volume together with its own volume in order to blend, classify, and render the partial volume. Then it sends the image back to the master. Otherwise, the slave sends its own warped volume to the slaves in the other cluster and requests the next job.

5. Performance evaluation

We implement the algorithm of distributed volume morphing using the C language and the PVM (Parallel Vir-

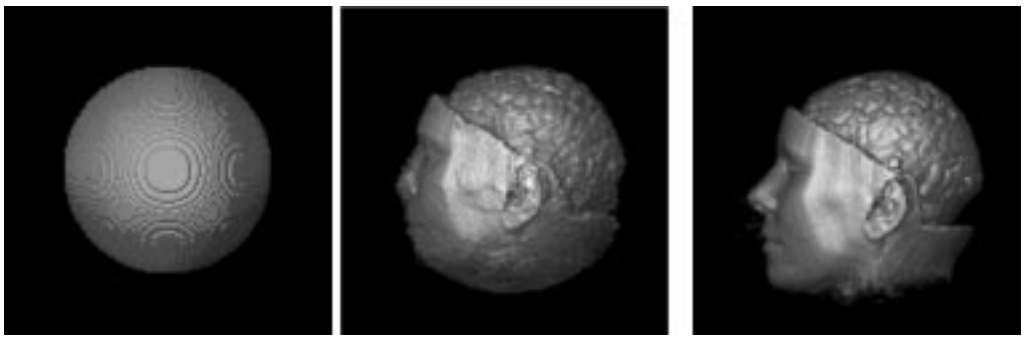

Figure 13. The animation sequence of the sphere to the brainsmall.

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

Strategy \ Slaves	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

tual Machine) platform [2] 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 calls the user's application program that should reside on each machine. Communication and synchronization among user's processes are controlled by the daemon processes. Unlike a shared-memory multiprocessor, the communication overhead of the PVM network platform 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 $128 \times 128 \times 84$ ($x \times y \times z$). 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. Figure 13 shows only the first, the middle and the last animation results of this example.

5.1. Speedup

Figure 14 shows the speedup for four strategies and the strategy '*Request*'. Among the five strategies, the strategy '*Request*' achieves the best speedup of 6.255 when 10 slaves are used.

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

$$parallel\_efficiency = \frac{Speedup}{Speedup_{\text{linear}}}. \tag{5}$$

The parallel efficiency of the strategy '*Request*' compared with that of other strategies using 5 slaves achieves the best result. The parallel efficiency improves from 52.84% using the strategy II to 81.89% using the strategy '*Request*'.

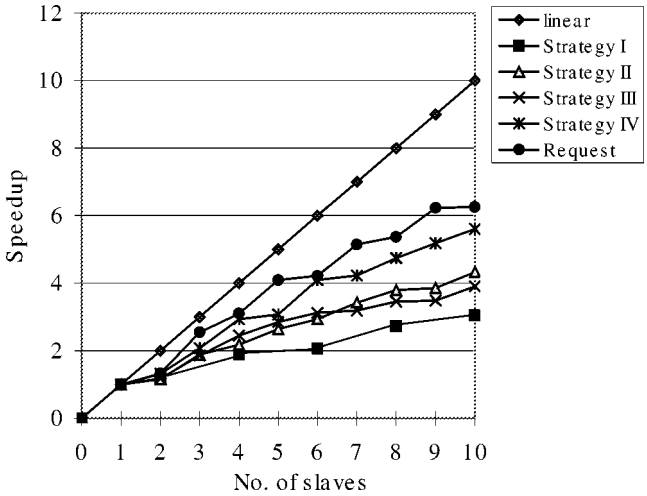

Figure 14. The average speedup of 5 frames.

5.2. Efficiency

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

$$eff = \frac{L_{\text{avg}}}{L_{\text{max}}}, \tag{6}$$

where L_{avg} is the average warping time of all slaves, and L_{max} is the maximum warping time 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, the strategy IV achieves the highest efficiency. It shows our volume partition method is better than even partition. Strategy III has good efficiency at frames 2, 3, and 4, but worse efficiency at frames 1 and 5, due to only one slave responsible

for the source or target volume and very light load at frames 1 and 5.

Furthermore, the warping efficiencies in tables 6 and 7 are better than the speedup drawn in figure 14. It may result from two reasons:

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

2. The slaves must wait for the warped volumes from the other slaves for blending. As a result, it can affect the total time.

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

5.3. Waiting time

We compare the waiting time between the strategy IV and the strategy '*Request*' in table 8. The average waiting 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.

5.4. Time analysis

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

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

where:

$t_{\text{warp}} =$ the execution time used in warping,

$t_{\text{blend}} =$ the execution time used in blending the two warped volume,

$t_{\text{classify}} =$ the execution time used in classifying the blended volume,

$t_{\text{render}} =$ the execution time used in rendering the partial image,

$t_{\text{comm}} =$ the execution time taken for inter-processor communication, including transmission and reception of partial volumes and partition data,

$t_{\text{wait}} =$ the execution time taken for waiting.

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

From table 9, we observe that the communication time contributes only 1.62% of the total parallel morphing operation. Compared with the warping time of the volumes

Table 8
The total waiting time of each slave for 5 frames using the strategy IV and the strategy '*Request*'. The average waiting time of each slave at each frame is shown in the last column.

Strategy \ Slave	1	2	3	4	5	6	7	8	9	10	Average
Request	1.67998	3.37936	11.8729	19.455	11.7725	13.3789	10.0945	15.2861	19.3578	9.87515	2.323044
IV	19.3096	38.8548	46.909	64.4183	55.6067	67.7257	54.6047	62.0297	74.9085	47.9381	10.6461

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)	870.6707	617.8421	2.472054	85.14585	14.12165	14.11944	136.9696
Per (%)	100	70.96163	0.283925	9.779341	1.621928	1.621674	15.7315

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)	733.3984	571.188	2.284154	63.05227	6.847984	6.376199	83.64975
Per (%)	100	77.88237	0.311448	8.597275	0.933733	0.869405	11.40577

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 which amounts to 15.73%. Moreover, the classification time should be also taken into consideration in order to 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 volumes 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 increases. Morphing a large dataset of volume may result in the memory shortage; we can solve this problem by using a cache strategy.

Acknowledgement

This research was supported in part by the National Science Council of Taiwan, under contracts NSC-87-2213-E-110-013 and NSC-88-2213-E-006-037.

References

[1] T. Beier and S. Neely, Feature-based image metamorphosis, in: *Proceedings SIGGRAPH '92*, Vol. 26 (July 1992) pp. 35–42.

[2] 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* (MIT Press, Cambridge, MA, 1994).

[3] T. He, S. Wang and A. Kaufman, Wavelet-based volume morphing, in: *Proceedings of Visualization '94*, eds. D. Bergeron and A. Kaufman (IEEE Computer Society and ACM SIGGRAPH, Los Alamitos, CA, October 1994) pp. 85–91.

[4] J.R. Kent, W.E. Carlson and R. Parent, Shape transformation for polyhedral objects, in: *Proceedings of SIGGRAPH '92, Computer Graphics*, Vol. 26 (July 1992) pp. 47–54.

[5] P. Lacroute, Fast volume rendering using a shear-warp factorization of the viewing transformation, Ph.D. thesis, Stanford University (1995).

[6] A. Lerios, C. Garfinkle and M. Levoy, Feature-based volume metamorphosis, in: *Proceedings SIGGRAPH '95* (1995) pp. 449–456.

[7] 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 (October 1993) pp. 15–22.

[8] J. Nieh and M. Levoy, Volume rendering on scalable shared-memory MIMD architectures, in: *Proceedings of the 1992 Workshop on Volume Visualization*, Boston (1992) pp. 17–24.

[9] T.W. Sederberg and E. Greenwood, A physically based approach to 2-D shape blending, in: *Proceedings of SIGGRAPH '92, Computer Graphics*, Vol. 26 (July 1992) pp. 25–34.

[10] G. Wolberg, *Digital Image Warping* (IEEE Computer Society Press, Los Alamitos, CA, 1990).

Leewen Lin is currently a teacher at Kaohsiung Senior Vocational Commerce School, Kaohsiung, Taiwan. Lin received a B.S. in computer education from National Taiwan Normal University in 1991 and an M.S. in computer science and information engineering from National Sun Yat-Sen University in 1998. Her research interests include computer graphics and parallel visualization.

Chungnan Lee received the B.S. and M.S. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1980 and 1982, respectively, and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, WA, in 1992. He is an Associate Professor in the Institute of Computer and Information Engineering at National Sun Yat-Sen University, Kaohsiung, Taiwan, since 1992. Prior to joining the faculty he was a system manager of Intelligent System Laboratory, a teaching assistant, and a research associate, while persuing his graduate studies at the University of Washington. His current research interests include computer vision, character recognition, computer graphics, Web and Java computing, Web-based knowledge discovery, computer-supported collaborative work, and parallel computing.

Tong-Yee Lee received his B.S. in computer engineering from Tatung Institute of Technology in Taipei, Taiwan, in 1988, his M.S. in computer engineering from National Taiwan University in 1990, and his Ph.D. in computer engineering from Washington State University, Pullman, in May 1995. Now, he is an Assistant Professor in the Department of Computer Science and Information Engineering at National Cheng-Kung University in Tainan, Taiwan, Republic of China. He was with WSU as a Visiting Research Professor at School of EE/CS during 1996 summer. He has been working on parallel rendering and computer graphics since 1992, and has published more than 50 technical papers in refereed journals and conferences. His current research interests include parallel rendering design, computer graphics, visualization, virtual reality, surgical simulation, distributed & collaborative virtual environment, parallel processing and heterogeneous computing.

Reproduced with permission of copyright owner. Further reproduction prohibited without permission.