Parallel Shape Transformation Using Slices

SHAMIMA YASMIN, ABDULLAH ZAWAWI TALIB
School of Computer Sciences
Universiti Sains Malaysia
11800 USM Pulau Pinang, MALAYSIA
shamima@cs.usm.my, azht@cs.usm.my

Abstract: - Parallel computing environment may be useful for faster and real time shape transformation generation. However, it is difficult to parallelised most of the existing shape transformation algorithms. In this paper, we describe parallelisation of a particular shape transformation algorithm based on the basic shape transformation algorithm. ‘Master-Slave’ model is adopted in parallelising the method with one master node and several slave nodes. Initially, one processor each is assigned to source and target data and slicing of both source and target data takes place until the desired number of steps has been reached. Parallelisation is also incorporated in the next step which is the interpolation phase and then also in surface reconstruction phase. We have shown that the proposed parallel version of the shape transformation algorithm is 'embarrassingly parallel'. The ease of implementing this parallel approach has also been compared to similar efforts in other shape transformation algorithms.

Key-Words: - Parallel Morphing, Distributed Morphing, Parallel Shape Transformation

1 Introduction

One important consideration of shape transformation algorithm is whether: the algorithm is capable of real time shape transformation. Parallel computing environment may be used to achieve faster shape transformation generation. Very few works have been reported on parallel/distributed shape transformation [1]. Also it is apparent that while designing shape transformation algorithm, ease of parallelising the algorithm was not in the designer's mind most of the time. Most of the surface-based shape transformation algorithms are discretised into corresponding meshes and involve a lot of interprocess communications [2] [3] [4] [5] [6] [7] [8] [9][10]. These methods are totally inadaptable to parallel computing environment. Point-based shape transformation algorithms have also been found to be quite difficult to parallelise as these algorithms exhibit similar properties like mesh-based shape transformation method and involve a lot of inter process communication [11][12]. Some volume-based shape transformation approaches involving disk field [13] or implicit function [14] seem worthy of parallelising but parallelising of these methods have not yet been reported.

Parallelisation of most of the existing shape transformation algorithms may involve a lot of interprocess communications. On the other hand, our basic algorithm on shape transformation has been designed with the ease of parallelising in mind ([15], [16]). It has been found that our slice-based shape transformation algorithm is more efficient and flexible than other approaches. Slices also scale well depending on the number of objects and their complexity. A brief overview of the basic shape transformation algorithm is described in the next section.

2 Overview of the Basic Shape Transformation Algorithm

The algorithm mainly consists of the following major steps:

- Data Traversal and Slicing of Data;
- Boundary Extraction;
- Boundary Projection;
- Boundary Traversal and Extraction of Point Clouds;
- Interpolation of Point Clouds;
- Orientation and Translation of Point Clouds;
- Surface Reconstruction.

Firstly, all input data are sliced by traversing along their longitudinal direction through continuous binary subdivisions. Slicing of data along the longitudinal direction continues until the number of steps reaches a specified number of steps. After completion of the data traversal, all data will have equal number of slices. Correspondence between each slice of the input data is established. Only boundaries of the slices are extracted. The boundaries are then projected onto the x-z plane. All boundaries from the input data are traversed and are converted to point clouds. Corresponding point clouds are interpolated. The interpolated point clouds are then oriented along the resultant orientation of the corresponding input boundaries and translated accordingly. This process is continued for all the
interpolated point clouds. Finally, transformed surface is constructed from the stack of interpolated point clouds.

3 Parallel Shape Transformation

Basic idea on the parallelisation of the basic morphing method using slices has been described in [15]. This involves two objects i.e. source object and destination object. Shape transformation involving more than two objects can also be parallelised in the same way. It has been shown that binary subdivision and simple data structure are suitable for both data as well as functional partitioning ([15], [16]). Object-based partitioning is used in our algorithm design to assign tasks to the processors. Firstly, all input objects are divided into equal number of slices. Equal number of boundaries is extracted from each input object. Figure 1 shows that equal number of boundaries is extracted from a number of input objects. Each input object can be imagined as an array of boundaries. All input arrays can be subdivided into equal number of portions. Each corresponding portion from each of the input objects forms one group. Each group can be assigned to different processors. Our slice-based methods for shape transformation which involves more than two objects can also be easily adapted to adopt this parallel approach [16].

3.1 Proposed Approach

"Master-Slave" model is adopted in parallelising the method. There can be one or two master nodes and several slave nodes. Figure 2 shows "Master-Slave" model with one master node and several slave nodes. In our parallel approach, the master also plays the role of slave when necessary as shown in the figure.

In the proposed parallel approach, it has been explicitly shown how each and every step of our basic shape transformation algorithm can be parallelised in order to speedup the overall process. Initially, two processors are assigned source and target data each, and both of them play the role of master processors. Figure 3 shows the process of parallelising the first step of the method namely ‘Data Traversal and the Slicing of Data’ when the number of processors is four. Processor ‘0’ and Processor ‘1’ both play the role of master. At Level ‘0’, each master divides the data into two parts. At that time all slaves are in standby ‘Receive’ position. Master keeps one portion of the data to itself and sends the other portion to the slave.

While sending sliced data to the other processors, some information such as the end’s normal of the data, the current number of steps, the remaining number of steps and the serial number of slice are needed to be sent. Once data is sent to a particular processor, it does not need to communicate with any other processor for any further information. Each processor continues to work independently. At ‘Level 1’, data is divided into four parts. Depending on the availability of the processors, sliced data are sent to idle processors. Otherwise active processors execute the job assigned to them. This continues until the desired number of steps has been reached.

Fig. 1. Extraction of Equal Number of Boundaries from All Input Objects.

Fig. 2. Communication between Master and Slave during Parallelisation.
Next comes the interpolation phase. Depending on the availability of the processors, both source and target point clouds are divided into equal number of groups. Corresponding groups of source and target are sent to one particular processor (master) for further processing. In Figure 4, source and target point clouds are divided into four groups and assigned to four different available processors. Groups of source and target point clouds are interpolated in parallel. After interpolation, all interpolated point clouds are assembled by Processor ‘0’ which is the only master processor in this step.

After interpolation, surface reconstruction takes place (Figure 5). Surface reconstruction is performed by considering only two consecutive point clouds. Hence while distributing the interpolated point clouds among the processors, point clouds are interleaved. As shown in Figure 5, Processor ‘0’ is assigned ‘0-8’ point clouds group, Processor ‘1’ is assigned ‘8-16’ point clouds group, ‘16-24’ point clouds group is assigned to Processor ‘2’ and so on. Here, it is found that point cloud ‘8’ is common to both Processor ‘0’ and Processor ‘1’. Again, point cloud ‘16’ is common to both Processor ‘1’ and Processor ‘2’. This interleaving is done in order to minimise interprocess communication as well as for the sake of simplicity. Each processor performs surface reconstruction in parallel. Later each partially reconstructed surface is assembled by the master.
Figure 6 shows the pseudocode for the parallel version of our basic morphing algorithm. In our parallel version of the basic morphing algorithm [2], communication happens when the master processor divides the data among the processors and when the master processor receives data from the processors. For synchronization, the master may also have to wait to receive all data from the rest of the processors. Besides, there is no communication among the processors. Processors do not need to communicate among themselves to retrieve information from each other which can be time consuming. Hence our algorithm can be claimed to be an ‘embarrassingly parallel’ algorithm. Neglecting time requirements for sending data to the slaves as well as for receiving data from them, speed up increases linearly with increase in the number of processors. Hence, in ideal situation speedup may be calculated as follows:

\[
\text{Speedup} = \frac{\text{Runtime}_{\text{sequential}}}{\text{Runtime}_{\text{parallel}}} \approx \text{Number of Processors} \quad (1)
\]

Efficiency also increases with increase in the overall speedup. In most other volume morphing algorithms, interprocess comunication can be a major factor ([11], [17]). If we compare our method with that of Lerois et al. [17], which is tested on distributed computing environment by Lin et al. [1], we find that a lot of interprocess communications is required. In that algorithm, various strategies for load balancing have been used with the best speedup of 6.225 with 11 processors. If interprocess communication is high among the processors, it can drastically slow down the overall speedup. Sometimes this can produce worse result than its sequential counterpart. Although in our parallel approach, we have shown the process of parallelising our basic algorithm on morphing (involving only source and target), parallelising can also be performed when shape transformation involves more than two objects without also any further involvement of communication among the processors. Also surface reconstruction can be carried out by several processors without slightest amount of interprocess communication.

4 Conclusion

This paper presents one further extension of our shape transformation algorithm. In the parallel version, our algorithm is shown to be “Embarrassingly Parallel”. In the version, interprocess communication is almost zero. System loss can only happen when both master and slave wait to receive data from each other. Most of the
other shape transformation algorithms involve a lot of interprocess communications and does not scale well in parallel/distributed computing environment.

References: