International Journal of Software Engineering and Its Applications
Vol.8, No.6 (2014), pp.183-194
http://dx.doi.org/10.14257/ijseia.2014.8.6.15

GPU-driven Parallel Processing for Realtime Creation of Tree Animation

Sang-Min Song, Young-Min Kang, Kang-Hyuk Lee and Soo-Yol Ok

Department of Game Engineering, Tongmyong University
{songsmir, ymkang, klee, sooyol}@tu.ac.kr

Abstract

The technological demand for graphically generating natural plants in real time recently have been more and more increasing in a variety of interactive content-creating areas such as computer games. In this paper, we propose a GPU-driven high-speed parallel processing algorithm for generating trees and their branches and leaves in real time. The method that we propose ensures the realistic generation of a multitude of trees and motions of their branches and leaves triggered by external force while it maintains the stability of the system.

Keywords: *CUDA, Parallel Computing, Hierarchical Shape Matching, L-System*

1. Introduction

Although the technological demand for graphically generating natural plants in real time recently has been more and more increasing in a variety of interactive content-creating areas such as computer games, the existing techniques turned out not to be viable for realtime tree animation [4, 1]. There are two common ways of generating motions of trees in real time: One is to simulate motions of trees by calculating their physical properties, and the other is to animate motions of trees by approximating them. The physics-based approach uses equations of motion, which is computationally expensive. Also, it does not ensure the numerical stability of the system, making it difficult to generate motions of trees in real time and thus difficult to apply to interactive contents like computer games. The skeleton-based approach is not able to yield realistic tree animation, thus resulting in poor immersive quality, since it does not depend on the calculations reflecting physical properties.

To improve these problems, some of the authors proposed elsewhere a method of generating realistic tree animation by applying the HSM (Hierarchical Shape Matching) technique that obviates the need for inducing complex equations of motion [2]. In this approach, the system simulates the local motions of a tree and then simulates again the global motions of the tree by utilizing the HSM technique, which is one of the ways to deform objects. The HSM technique ensures faster processing of motion generation than the physics-based approach, provides the robust numerical stability, and allows for more realistic rendering of motions. However, it also suffers from heavy computational costs and difficulties with realtime rendering.

To remedy the problems with our previous approach, we propose an improved algorithm to locally simulate the motions of a tree and those of its branches and leaves by way of CUDA-based realtime parallel processing.

ISSN:1738-9984 IJSEIA

2. Related Work

The expression of motions caused by external forces (*e.g.*, wind) is a very important element in enhancing the realistic quality of digital contents such as computer games. Various attempts have been made to simulate of motions of trees triggered by wind [1, 3]. In this line of research, wind-triggered motions of trees and their branches were reproduced by performing the physical simulation in which the force exerted on each branch of a tree is taken into consideration. While this physics-based approach has the merit of faithfully reproducing the effects of the field of wind and interaction forces between branches, it suffers from the problem of incurring heavy computational costs. Moreover, it does not secure the graphic fidelity in such a way that the motion of each and every leaf attached to a branch is expressed.

The method proposed in [4] provides a way of expressing effects of wind in real time for a large-scale forest scene by breaking down a tree into a limited number of control points, modeling the motion of the tree based on the oscillation function of the control points, and performing the GPU-driven calculation of the motion simulation. However, this method does not allow the user to control the wind in an interactive way. It also has a limit to expressing the naturalistic motion of the leaves, since their motion is constrained by the motion of the branches. Deformation-based animation is proposed in [5]. In this method which improves the shape matching technique proposed in [6] and implements the deformation simulation on the voxel grid, the vertices in the tree geometry are translated and rotated in accordance with the external force of the wind. It has the merit of easily making out clusters when simulating the motion of tree on the voxel grid by using the deformation technique and separately applying rendering and simulation. But in order to get a complicated deformation, the number of clusters naturally increases, leading to more computational costs. [7] proposed the method to calculate elasticity by using mesh vertices only, define the fast summation in terms of generating the path on a contour line and apply the techniques of volume preservation and vibration suppression. Nevertheless, this method is not able to get an optimal solution since the path-generating algorithm is based on empirical values.

In this paper, in an attempt to improve the shape matching method proposed in [6], we propose a GPU-based parallel processing method for creating realistic tree animation in real time by modeling numerous parts in a tree based on hierarchical shape matching without recourse to complicated motion equations such as equations of fluid dynamics.

3. Parametric L-System-based Generation of Tree Appearance and Shape Matching Technique

In this paper, we employ the parametric L-system for generating the shape of a virtual tree. L-system is a technique to represent the growth of a tree by extending a set of functional strings. Figure 1 shows the system flow according to which the motion of a tree is generated.

From the careful observations of the motion of a tree in the real world, we know that the tree is moving agreeably to the external force of wind and returns to the original state in due course of time. This means that motions of a tree can be described as deformations of an object. In order to create realistic tree animation, we adopt a shape matching technique [6], one of the techniques of the soft body dynamics to deal with deformations of soft objects such as cloth. When an object was deformed by an external force, this technique expresses the deformation of an object as finding the goal position which points to the initial appearance of the tree and moving the object to that position. Let us consider a deformable object. The set of the vertices of the object is denoted by X. The original center of mass can be denoted by p_0. If

the deformation can be described with a linear transformation A, then new X' can be expressed as follows:

$$X' = AX + p_0 \tag{1}$$

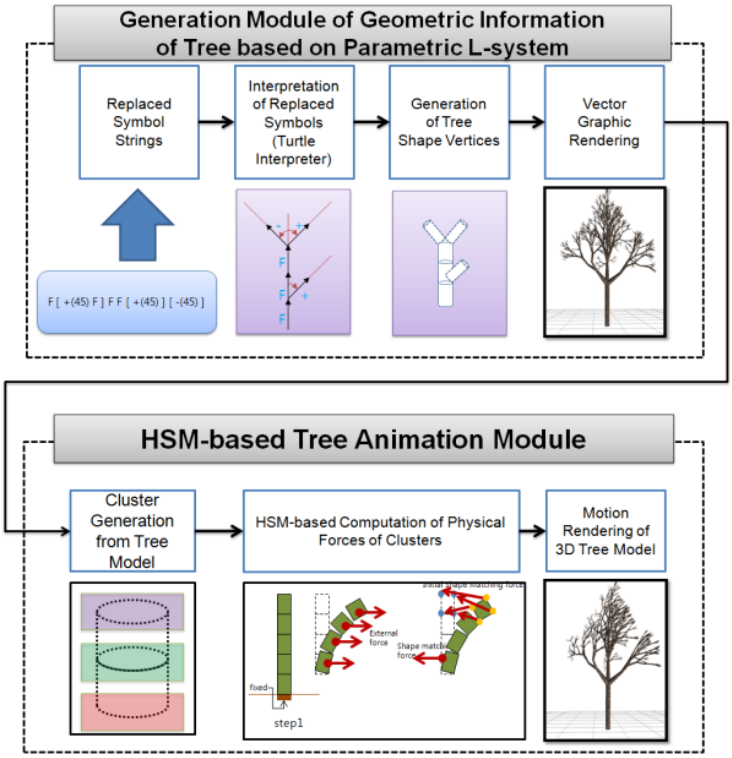

Figure 1. System Flow for Generating the Motion of a Tree

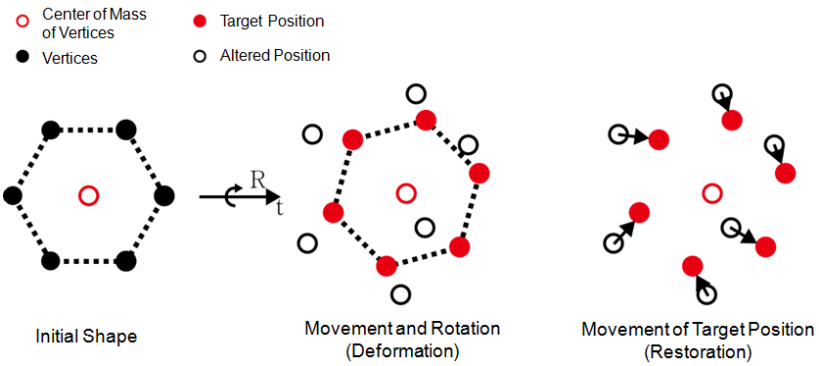

Figure 2. How the Shape Matching Technique Works

If the object has exactly the same shape as the initial state, the linear transformation should involve only rotation R and translation T. In this case, the new state can be expressed as follows:

$$X' = RX + (T + p_0) \tag{2}$$

where $TX + p_0$ is the new center of mass p_1.

Eq. 2 can be considered to be the target shape that the deformable object should be restored to. In fact, the deformable object undergoes the deformation triggered by the external force. Therefore, the current state X' cannot be accurately obtained with R and T. If we find the R and T that optimally transform the original shape close to the current shape, we can obtain the target shape G. Each vertex x_i should be animated to move back to the goal position g_i in the set G,

$$w_i R\left(x_i^0 - x_{cm}^0\right) + x_{cm} - x_i \tag{3}$$

where x_i is the location of the vertex i, and w_i, the weight of the vertex. x_{cm}^0, denotes the center of mass of the object at the initial state while x_{cm} is the current state of mass. Whereas the translation transformation T can be easily computed with Eq. 3, R cannot be easily obtained. Therefore, we first obtain a linear transformation A, and approximate the R by removing the scaling factors in the transformation, as formulated in Eq. 4.

$$m_i A\left(x_i^0 - t_i\right) + t - x_i \tag{4}$$

$$\sum_i m_i (A\left(x_i^0 - t_i\right) + t - x_i)^2 = 0 \tag{5}$$

The optimal linear transformation matrix can be obtained by the least square method as formulated in A of Eq. 4. Eq. 5 can be solved on A as follows:

$$\begin{aligned} A &= (\sum_i m_i p_i q_i^T)(\sum_i m_i q_i q_i^T)^{-1} \\ &= A_{pq} A_{qq} \end{aligned} \tag{6}$$

By using the optimal linear transformation matrix A and approximating the rotation matrix R, we can obtain g_i, the goal position of particle x_i as follows:

$$g_i = R\left(x_i^0 - t_0\right) + t \tag{7}$$

The force of shape matching can be obtained by applying Eulerian integration to the goal position g_i.

$$v_i = (t + h) = v_i(t) + \alpha \frac{g_i(t) - x_i(t)}{h} + h \frac{F^{ext}(t)}{m_i} \tag{8}$$

Figure 8 formulates the required velocity to easily compute the goal location with the rigidity control parameter α ranging from 0 to 1.

4. Shape Matching for Realistic Tree Animation

The hierarchical shape matching (HSM, hereafter) technique that we propose is for creating realistic tree animation. As shown in Figure 3, the branches and leaves of a tree are made up of the independent clusters each of which maintains its appearance by virtue of shape matching.

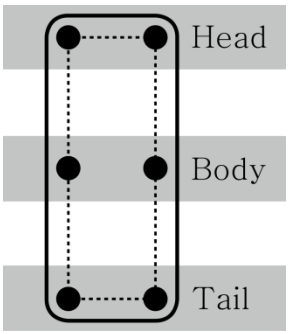

Figure 3. The Structure of a Cluster

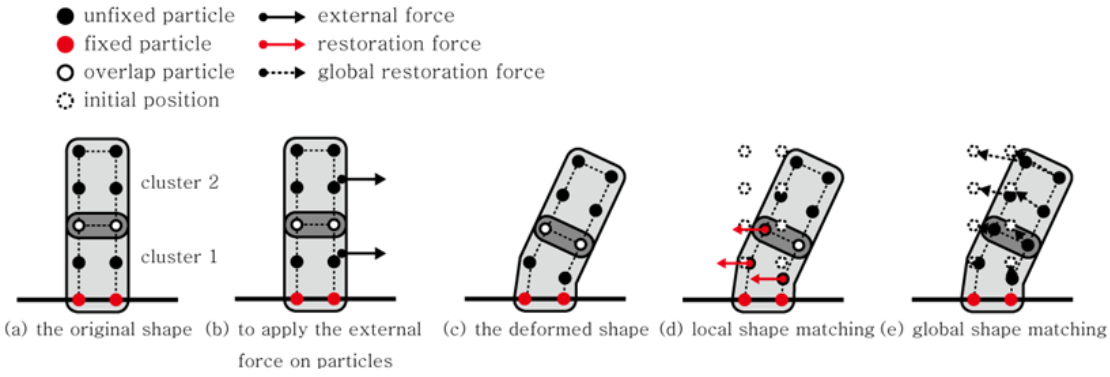

Figure 4. Conceptual Principle of Tree Animation based on HSM

The HSM technique performs shape matching for each and every independent branch and leaf of a tree, and then translocates the shape of the whole tree to the initial position. Figure 4 shows the concept of HSM with the local and global shape matching, in which the branches and leaves structured into the corresponding clusters form the initial state by connecting themselves to each other. Although HSM serves to create realistic motions of a tree whose appearance is generated by L-system, it suffers from the burden of heavy computation when the number of branches and leaves is massive, which is not rare in reality.

5. Realtime Tree Animation Using CUDA Parallel Computing

HSM enables each cluster to maintain its shape, and the Head and Tail of one cluster are linked to the Tail and Head of the other cluster, as shown in Figure 5. These linked structures between clusters ingenerate the transmission of force in terms of which the motions of a tree are created. In order to perform the parallel computing of HSM-based data structure of a tree, the most efficient way to do it is to compute cluster by cluster. Each cluster basically undergoes shape matching, and HSM performs the same number of computations as the number of vertices in a cluster. CUDA ensures the very fast processing for these repeating computations. Since parallel computing can be most effective when the data do not interfere with each other, however, the HSM-based linked structures between clusters inherently have a fundamental problem of being unable to achieve the maximum efficiency of parallel computing.

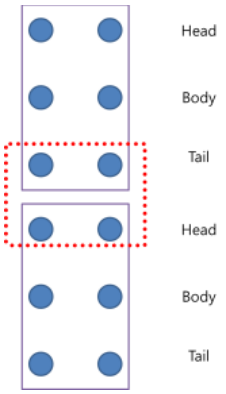

Figure 5. Linked Structure of a Cluster

5.1. Data Structure for CUDA Parallelization

HSM uses the linked structures of clusters to generate the motions of a tree. To easily compute a number of linked structures between clusters, HSM employs the tree data structure well-suited to linked structures. Figure 6 shows that HSM-based tree data structure is altered to the linear data structure for CUDA parallelization of HSM.

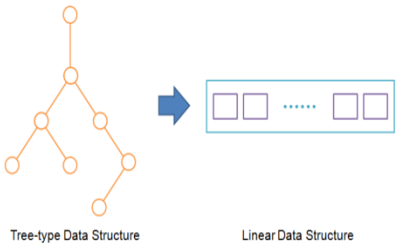

Figure 6. Alteration of Data Structure

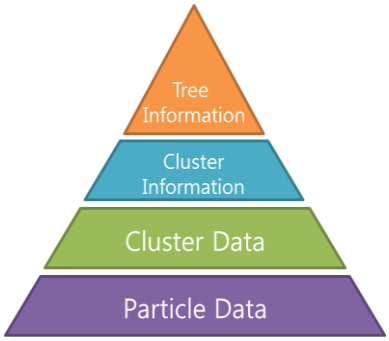

Figure 7. Hierarchical Information of a Tree for Parallel Processing

Much information is needed for copying the data of HSM to CUDA memories. For CUDA-based parallel processing, all the information on CPU for generating the motions of a tree must be transferred to GPU. We classified the information for parallel processing into Tree Information, Cluster Information, Cluster Data and Particle Data, as in Figure 7.

First of all, Tree Information contains the information of the total number of clusters forming a tree and the maximum number of child clusters. Cluster Information has the information of kinds of clusters, data locations of clusters, and particle locations of clusters.

In CUDA-based parallel processing, the kind, data and particle locations of a cluster can be easily obtained from the ID of the cluster.

Cluster Data consists of the link information between clusters, the state information of clusters, and data for computing shape matching. Cluster linking on CPU can be easily implemented in terms of tree structure using pointers. But in CUDA-based parallel processing, the data should be organized as a float-type linear array, so the linked structures between clusters must be separately stored. Particle Data carries the information of particles in a cluster. Particles on CPU basically need to have the information on acceleration, velocity, location, and mass. For CUDA processing, however, the data should be shared so that the current position, original position, acceleration, velocity and mass need to be put into the graphic buffer. Also, the additional data for rendering—texture coordinates, normal vectors and fixed (Boolean) data for identifying whether or not a vertex is fixed—are required.

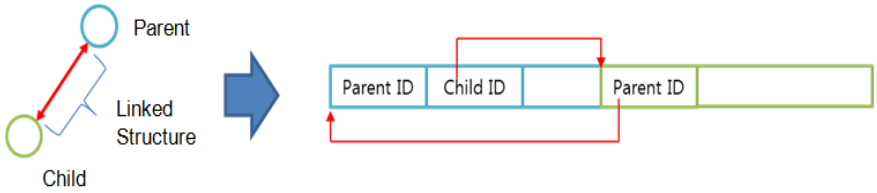

Figure 8. Linearization of Linked Structure

As shown in Figure 8, the HSM linked structures between clusters must be altered to the corresponding linear structures. To linearize the linked structures, the additional information of Parent ID, Children ID and Children Count is needed. Parent ID and Children ID serve as the linked pointers to parents and children in the tree structure, and have the values of the parent position and the children position in the array.

In addition to the linked information of clusters, there exist the data to check the use of global shape matching, and Original CM, Current CM, A3pq and A that store the results of HSM computation. Original CM and Current CM indicate the initial shape position x_i^0 and the current shape position $x_i$in Formula. 6. A3pq is A_{pq} in Formula. 6, and A is identical with the linear transformation matrix A.

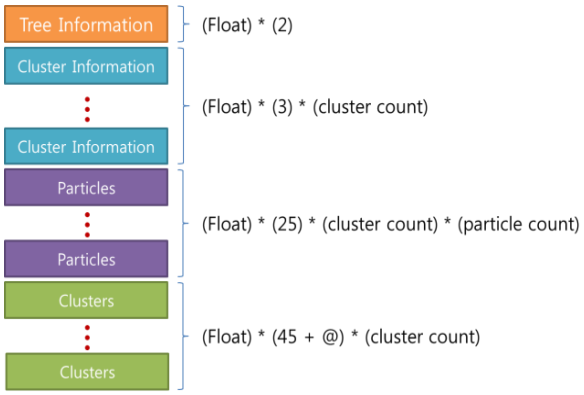

Figure 9. Linearized Data Structure and Data Size

Figure 9 shows the way of putting data into the actual CUDA memory. The total size of memory depends on the number of clusters and particles. @ of cluster data refers to the maximum number of children.

5.2. Computing the Motion Generation of a Tree

CUDA parallelization of HSM can be divided into preprocessing and processing by frame. Preprocessing computes only once the value of the center of mass in the initial position. This value is computed only once on CPU and stored in original CM of CUDA data. Included as the values computed by frame are the center of mass of the current particle CM, the linear transformation matrix A, and A3pq necessary for finding A.

The order of HSM-based motion generation is shown in Table 1.

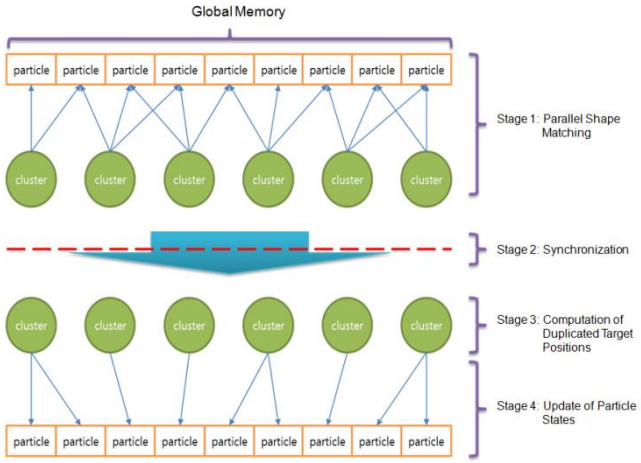

Figure 10. Computing Method using Parallel Processing Technique

Table 1. Motion Simulation Procedure for a Tree Model using the Multiple Cluster Method

1. Find the rotation matrix of the cluster R^c.
2. Find the goal position on the vertex of the cluster g_i^c.
   $g_i^c = R^c((x_i^0)^c - t_0^c) + t$
3. Find the goal position of the overlapped vertex G_i.
   $G_i = \sum_{c=0}^{n} g_i^c$
4. Find the goal position of the overlapped vertex g_i.
   $g_i = \frac{G_i}{n}$
5. Find the force of shape matching F^{sm}.
   $F^{sm} = \alpha(g_i - x_i(t))$
6. Approximate the velocity by using the external force F^{ext} and the force of shape matching F^{sm}.
   $v_i(t+h) = v_i + \frac{F^{sm}}{h} + h\frac{F^{ext}}{m_i}$
7. Approximate the position by using the approximated velocity.
   $x_i(t+h) = x_i(t) + hv_i(t+h)$

To compute the motion of a tree using CUDA, as shown in Table 1, the system goes through the four stages of processing in Figure 10. The stage 1 performs parallel of processing of HSM for all clusters. After synchronizing the clusters in stage 2, the goal positions of duplicated particles are computed in stage 3, and the states of the particles are renewed in stage 4.

- Stage 0: Advance Preparation
  Prior to Stage 1, the CUDA blocks and threads should be divided. CUDA processes the data divided into Grid, Block and Thread. The clusters are set up as Block, and the particles, as Thread. The reason for setting up the clusters as Block is to use the shared memory when computing the particles. If the shared memory is initialized block by block and the particles exist in other blocks, it cannot be used. Also, each particle is given a thread so that parallel processing can be extended from the unit of clusters to the unit of particles.
- Stage 1: Finding Linear Transformation Matrix A by clusters
  For parallel processing of HSM, the optimal linear transformation matrix A should be found by clusters. Prior to finding A, the changes of particle positions caused by external and gravitational force are computed by using Eulerian integrals, after which the goal position and the force are initialized. Without the initialization of the goal position, errors occur when computing the duplicated target positions in Stage 3. After computing the changes of particle positions, the optimal linear transformation A is to be found using the altered postions of particles.
- Stage 2: Synchronization by Clusters
  After finding the optimal transformation A of clusters, all the clusters should be synchronized. Without this synchronizing process, the optimal linear transformation may refer to the values of clusters not found by it, leading to computational errors.
- Stage 3: Finding the Duplicated goal positions
  After going through Stage 2, all clusters have their optimal linear transformation A. By using A, the goal positions are found. When finding the goal positions, the optimal linear transformation A of the linked clusters of parents and children is needed. In so doing, the clusters using the particles must set up all the goal positions, and thus they use the method of accumulating the data. In case the goal positions are not initialized, the error values can be assigned as the goal positions, so the process of initialization is done in Stage 1. If Stage 1-3 are performed at one go, as are in the ordinary HSM method, they cannot be processed in a parallel way. In other words, computing HSM in the four stages is solely for parallel processing.

6. Experimental Results

The method proposed in this paper was implemented and experimented on the system with Intel 3.30GHz i7 CPU, 24G DDR3 RAM and Nvidia GTX 580 GPUs running on Windows 7 OS.

Table 2. Speed of Parallel Processing (Frames Per Second) Applied for a Single Tree

cluster	particles	FPS
5	300	2,200
15	900	1,800
67	3,444	1,400
293	13,548	1,000
867	38,772	670
2,230	99,816	350
4,975	211,812	175

Table 2 shows the frames per second when HSM-based tree animation was done using GPU-driven parallel processing.

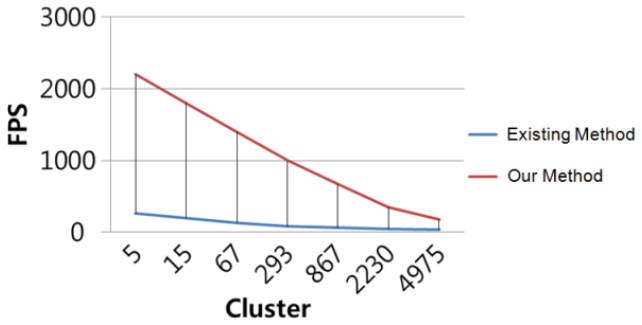

Figure 11. Comparison of Efficiency of CPU-based Processing with CUDA Parallel Processing

Figure 11 is the graph in which the existing HSM method is compared with ours. The existing method shows the performance of 32 FPS when a tree grows to its last stage, our method maintains the performance of 175 FPS, which obviously is well-suited to real-time processing.

Table 3. FPS (Frames Per Second) of Multiple Tree Simulation

cluster	particles	FPS
50	3,000	260
150	9,000	210
670	34,440	160
2,930	135,480	110
8,670	387,720	70
22,300	998,160	36
49,750	2,118,120	17

Figure 12 shows the experimental results of generating ten trees and animating their motions. Although the number of clusters and particles exponentially increases with the progress of each stage, our method not only ensures the numerical stability but also shows the ability to express the realistic tree animation.

Table 3 shows the result of simulating ten trees, using the method that we proposed. In the last stage of the tree-growing process, 17 FPS is achieved.

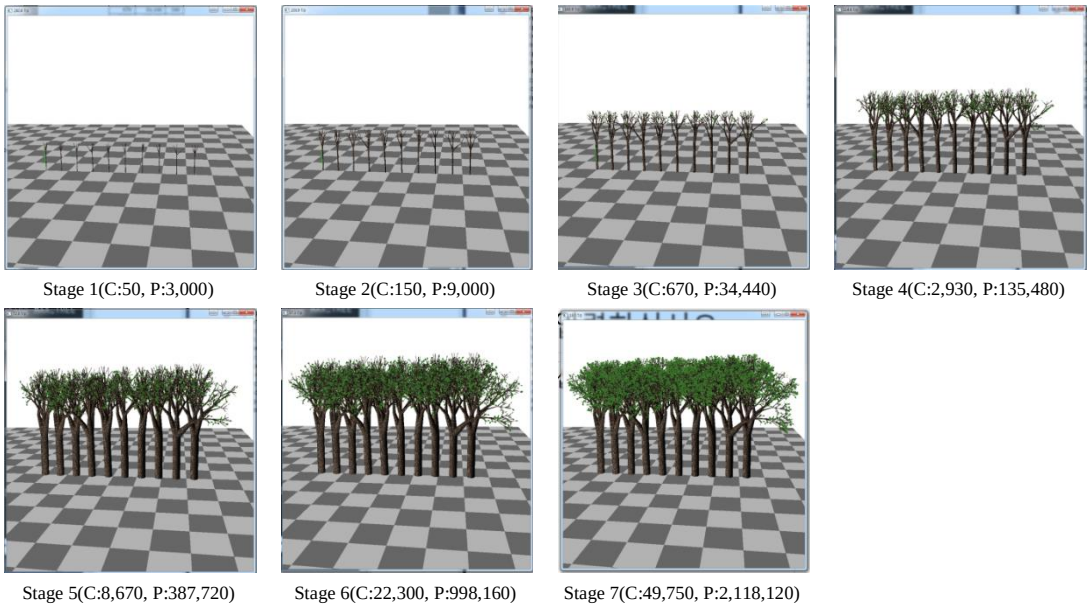

Figure 12. Clusters and Particles in Conformity with the Growth of a Tree

7. Conclusion

In this paper, we have proposed a method of parallel processing in terms of which the speed down problem of the HSM method is substantially improved that occurs as the number of clusters increases. With its improvement, it still runs short of dealing with things like a forest with a large number of trees. Nevertheless, our method is viable for realistically simulating motions of a modest number of trees interactively with the user, though it is not able to simulate a huge number of trees, which may be needed in heavy real-time programs such as computer games.

Acknowledgements

This work was supported in part by Busan Brain 21 (BB21) support program supervised by Busan Human Resources Development Institute, Korea.

References

[1] N. J. Oliapuram and S. Kumar, “Realtime forest animation in wind”, Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing, ICVGIP ’10, New York, NY, USA, **(2010)**, pp. 197–204.
[2] S.-M. Song, Y.-M. Kang, E.-J. Lee and S.-Y. Ok, “Realtime forest animation in wind”, Multimedia and Signal Processing: Second International Conference, CMSP, CMSP ’12, Springer, Berlin/Heidelberg, Germany, vol. 346, **(2012)**, pp. 475–482.
[3] Y. Akagi and K. Kitajima, “Computer animation of swaying trees based on physical simulation”, Computers and Graphics, **(2006)**, pp. 529–539.
[4] J. Diener, M. Rodriguez, L. Baboud and L. Reveret, “Wind projection basis for real-time animation of trees”, Computer Graphics Forum, **(2009)**.
[5] D. Steinemann, M. A. Otaduy and M. Gross, “Fast adaptive shape matching deformations”, Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA ’08, Aire-la-Ville, Switzerland, Switzerland, pp. 87–94, **(2008)**.

[6] M. Müller, B. Heidelberger, M. Teschner and M. Gross, "Meshless deformations based on shape matching", ACM SIGGRAPH 2005 Papers, SIGGRAPH '05, New York, NY, USA, **(2005)**, pp. 471-478.
[7] R. Diziol, J. Bender and D. Bayer, "Robust real-time deformation of incompressible surface meshes", Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '11, New York, NY, USA, **(2011)**, pp. 237-246.

Authors

Sang-Min Song, received his B.Eng. (2011) and M.Eng. (2013) degrees in game engineering from Tongmyong University. He is currently a Ph.D. student of Dept. of Computer Media Engineering at Tongmyong University. He is interested in GPU-based parallel algorithms for realtime animation and rendering. His research interests also include realtime animation of massive crowd and various game technologies.

Young-Min Kang, received his B.Sc., M.Sc. (1999), and Ph.D. (2003) degrees from Pusan National University. He was a temporary researcher at MIRALab in University of Geneva, Switzerland. He also worked as a researcher at Electronics and Telecommunication Research Institute (ETRI), Korea. His research interests include physically based animation, realtime rendering, general purpose GPU programming and game programming. He is currently an associate professor in Dept. of Game Engineering at Tongmyong University, Korea.

Kang-Hyuk Lee, received his Ph.D. (1993) from University of Illinois at Urbana-Champaign, USA. During 1993-1996, he was with KAIST (Korea Advanced Institute of Science and Technology) and KISTI (Korea Institute of Science and Technology Information) as a postdoctoral research fellow and a senior research scientist, respectively. Since 1997, he has been teaching in Dept. of Multimedia and Game Engineering at Tongmyong University, Korea. His research interests include VR-based gaming and socio-cultural implications of gaming.

Soo-Yol Ok, received his Ph.D. degree (2001) from Tsukuba University, Japan. He was a special researcher at Japan Communication-Broadcasting Organization from 2001 to 2002. He worked as a researcher at Communication Research Laboratory, Japan from 2002 to 2004. He is currently an associate professor and the head of Super Computing Center for Application at Tongmyong University. His research interests include GPU-based parallel algorithms for 3-D graphics and image.

