The Sixteenth International Symposium on Artificial Life and Robotics 2011 (AROB 16th '11),
B-Con Plaza, Beppu,Oita, Japan, January 27-29, 2011

Three-dimensional Morphogenesis by Cell Division and Death in Viscoelastic Amorphous Computing

Eisuke Arai, Fumiaki Tanaka and Masami Hagiya

Department of Computer Science, Graduate School of Information Science and Technology, University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, JAPAN.
(Tel : +81-3-5841-4093)
(arai917@is.s.u-tokyo.ac.jp)

Abstract: Amorphous computing is a computational paradigm for realizing global computation as a result of local communication among identical computational particles which are irregularly distributed over space. Each computational particle has a small computing power and a small amount of memory, and executes an identical program with no synchronization. In the original model of amorphous computing, particles did not have motility resulting from physical interactions and could not form any significant structures by their self-organization in three-dimensional space. In this paper, we propose a new approach to construct desired structures of programmable particles in three-dimension by extending the original model of amorphous computing. In addition to general operations in the original model, we introduce operations for generation and destruction of particles. Furthermore, we allow particles to have slight motility by viscoelasticity, because we assume physical interactions among particles. So far, we have built a simulator to observe behaviors of moving particles in three-dimensional space and control their morphogenetic process. In this simulator, the particles execute an identical program which includes the operations for generation and destruction. We have investigated and optimized parameters of the simulator so that the particles moderately cluster and form some fixed structure.

Keywords: Amorphous computing, Morphogenesis, Viscoelasticity, Three-dimension**.**

I. INTRODUCTION

Models of amorphous computing consist of a collection of small computational particles distributed irregularly over space where each particle has a small computing power and a small amount of memory. These individual particles are entirely identical and not synchronized. Each particle can communicate with a few nearby neighbor particles. Although all particles are programmed identically, each particle is distinguished from other particles by its local state which includes the list of neighbor particles, distance from each neighbor, and local variables. The study of amorphous computing was introduced by Abelson et al. in [1] and the computational models based on the idea of amorphous computing were summarized in [2]. Among the models, that of programming amorphous medium was established by Beal et al. [3]. Amorphous medium is the continuous limit of models of amorphous computing in which particles are infinitesimal and distributed densely over space. Proto [4] is a Lisp-like programming language which targets such amorphous medium. As an example of programming amorphous medium, a sensor network was implemented in Proto [5]. The advantage of amorphous computing is its robustness compared with other general computational paradigms because the original model of amorphous computing was inspired by biological systems [6]. Thus there has been developed an application of amorphous computing in which cells such as E coli are targeted [7]. Needless to say, one of the goals of such applications is to construct shapes and structures composed of particles, just as biological systems realize morphogenesis.

In this paper, we present a new approach to three-dimensional morphogenesis in a model of amorphous computing. We give particles slight motility that depends on interactions among them, and implement the generation and destruction operations in addition to the operations in the original model of amorphous computing. Two-dimensional morphogenesis in a model of amorphous computing was investigated in a preliminary attempt [8]. Although a three-dimensional approach to morphogenesis in amorphous computing was also proposed [9], motility of particles was not based on their physical interactions but due to predefined background signals, and particles did not execute an identical program. In our approach, we assume that each particle executes an identical program and moves automatically by viscoelastic force. The viscoelasticity of particles is generated according to distance from the neighbors and density of the neighbors. We refer to smooth particle hydrodynamics (SPH [10]) in order to simulate the viscoelastic behaviors of particles and implement calculation of

viscoelastic force using a method established by Clavet et al. [11]. We need to update neighbors of each particle at every simulation step, because particles have motility and change their positions. Thus we have implemented efficient neighbor search strategy of particle-based fluids [12].

II. THE MODEL

Our simulation model is based on the original amorphous computing model. The originality of our work is roughly divided into four important points. In this section, we explain these points individually.

1. Motility of computational particles

We introduce motility of particles, because we assume that particles are like biological organisms or biomolecules. Thus it is natural that there are physical interactions among particles such as viscosity and elasticity. We use the well known method for particle-based viscoelastic fluid simulation in order to add viscoelastic force to particles. This force is assumed to be produced by interactions among nearby particles and affects them. For each particle, we need to calculate force by all interactions with its neighbors at every computation step. Although we introduce motility, we do not add to particles the ability to move freely because we only allow particle to be affected by external force.

2. Implementation of particle operations

To execute programs written in the style of amorphous computing in our model, it is necessary to implement several basic operations and build them into particles. We can execute complex programs by systematically combining these operations. Concretely, we implement the following operations.

A. Minimum and maximum operations

In the minimum operation, each particle obtains the minimum of the specified values owned by all neighbor particles by repetition of local communication. A particle refers to the specified values of neighbors, finds the minimal value of them and then stores this value into itself in a single communication step. Similarly, the maximum operation allows each particle to find the maximal value of those owned by all neighbors.

B. Gradient operation

The gradient operation is a way for each particle to know distance from a specified particle as the gradient value. Each particle finds the local minimal value of neighbor gradient values and the local maximal value of ranges to its neighbors which ranges are known as a priori information. Sum of these local values is stored as new gradient value in single communication step of the operation. By repetition of the local communication, all particles know their distance values from the specified particle.

C. Generation and destruction operations

Inspired by cell division and death, we implement the generation and destruction operations which increase and decrease particles, respectively. In the generation operation, a particle reproduces itself with its own state at an adjacent point. On the other hand, in the destruction operation, a particle kills itself. In our model, we do not consider the energy consumption with reproduction of particles and the remains of particles in the destruction operation.

3. Mechanism of our simulator

We built a simulator which executes the particle operations just as we introduced above. After initialization, the simulator repeats updating states of particles and visualizing them. The single update process executes the necessary tasks as follows. At first, the simulator executes the neighbor search for all particles, calculates viscoelastic force between all particle pairs, and then moves all particles by the calculated force. Next, the simulator executes the neighbor search again because neighbor particles are changed by their movement. Finally, each particle communicates with its neighbors and updates its state which includes some values related to the operations. Only one communication step for one particle is allowed in a single update. Thus, by repetition of the update process, the value for each operation converges. In our simulation, we use modestly converged values for a cascade of operations.

4. Dynamic neighbor search in 3D space

In our system, it is necessary for each particle to always know its neighbor particles so that it can contact and interact with each other. Since we assume that particles move by some operations, we have to dynamically update neighbors at every computation step of each operation. To search for neighbors, we need not check all particles, but need to check only some particles which are included in particular divisions of space called cells. Each cell is a 3D grid with size of the neighborhood radius of a particle. We divide space into such cells and index them depending on coordinates of

their minimal corner. Furthermore we associate each particle with its corresponding cell index. For searching for neighbors of a particle, it is only necessary to check particles inside 26 cells around the cell containing the particle. In addition to this cell indexing approach, we use also subdivision of the cell in order to further reduce search space.

III. EXAMPLE

In this section, we demonstrate the process to form a tubular structure as an example of our approach. We explain an algorithm for modeling tubular structure and then show simulation results of this example using our simulator.

1. Algorithm for modeling tubular structure

We assume that there is one computational particle at the initial condition. This particle begins the generation (divide) operation and keeps on dividing until the number of particles is sufficient to form any structure. After the particles are clustered, one particle on the surface of the cluster is selected as a source. By searching for the particle which has the minimal number of neighbors in all particles, we can find the source on the surface. We then execute the gradient operation from the source and choose the destination particle which has the maximum gradient value, that is to say, the farthest particle from the source. The destination is also assumed to be on the surface of the cluster. Figure.1 shows the state of choosing the source and destination particles in the cluster.

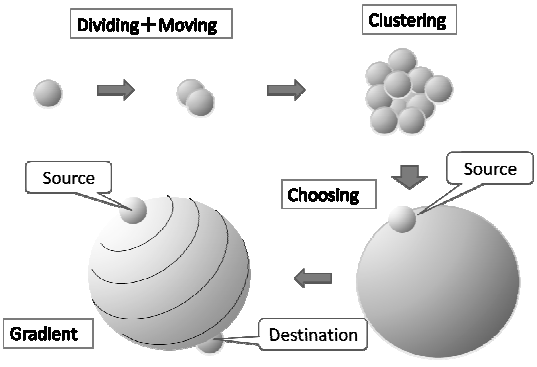

Fig.1. Choosing specific particles

After both source and destination particles are found, each particle starts the gradient operation towards the destination to determine the specific particles lying between the source and the destination. We call the line of these particles "path". For a particle on path, the sum of its gradient values from the source and from the destination is almost equal to distance between the source and the destination. Neighbors of particles on the path comprise a channel which connects the source and the destination with slight width. In Figure.2, it is shown how to make the path and channel in the cluster. After determination of the particles in channel, each particle in the channel executes the destruction operation and disappears from the space.

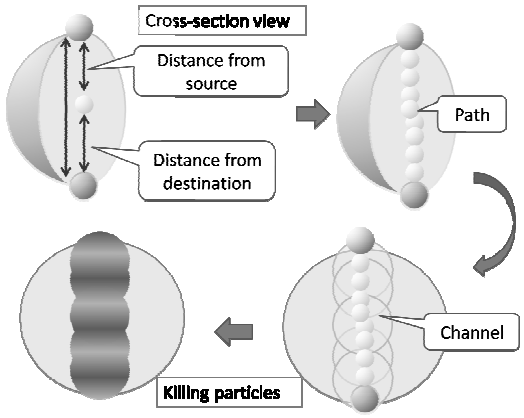

Fig.2. Creating tubular structure

An important issue in this algorithm is that each particle executes an identical program and communicates with only its neighbors. Although each particle has no a priori knowledge about other particles except for its neighbors, it is distinguished from other particles by its internal state such as the gradient value. Therefore, we can control identically programmed particles in order to model desired structures like this example.

2. Simulation results

Using our simulator we could construct a cluster of 500 particles which is shown in Figure.3. There was only one particle at the initial condition and then the the number of particles started increasing by the generation operation. To cluster particles as shown in Figure.3, we investigated and optimized some parameters of the simulator by trial and error. These parameters include neighborhood radius, spring constant, density constant and other values related to calculate viscoelastic force.

Fig.3. A cluster of 500 particles

After clustering particles and finding the source, each particle executed the gradient operation from the source and then could determine the destination particle which had the maximum gradient value. Each particle also could judge whether it was on the path and furthermore in the channel. The particles in the channel executed the destruction operation and disappeared. As a result, we obtained a tubular structure like a bead (see Figure.4).

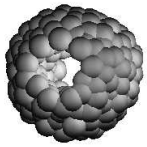

Fig.4. A tubular structure

IV. CONCLUSION

In this paper we have presented a new approach for constructing a specified structure from a cluster of computational particles which are programmed to execute an identical operations. To cluster the particles we implemented the generation operation and gave them slight motility based on viscoelasticity among particles. As an example of our approach, we demonstrated construction a tubular structure by each particle's executing an identical program. Although we do not have actual implementation of such programmable particles at present, we expect that we can possibly use living cells for our research in future, because a cell behaves according to its program, that is DNA.

V. FUTURE WORKS

Our future work can be considered in several ways as follows.

・We need to increase efficiency of each simulation step because we want to enlarge the size parameters of our model such as the simulation space and the number of particles. To this end, we will attempt to optimize the dynamic neighbor search and the calculation process of local interactions.

・Although we currently assume a spherical cluster of particles in our model, we need to consider a cluster of any shape like a distorted one.

・In this paper, we demonstrated construction of a tubular structure. We want to extend our study to establish a new method for constructing arbitrarily complex structures.

REFERENCES

[1] H. Abelson, D. Allen and D. Coore et al. (2000), Amorphous computing. Communications of the ACM, 43:74-82

[2] H. Abelson, J. Beal and G.J. Sussman (2007), Amorphous Computing. MIT Technical Report 2007-030.

[3] J. Beal (2006), Amorphous Medium Language. MIT Technical Report 2006-040.

[4] J. Beal and J. Bachrach (2006), Infrastructure for engineered emergence in sensor/actuator networks. IEEE Intelligent Systems:10-19

[5] J. Bachrach and J. Beal (2006), Programming a Sensor Network as an Amorphous Medium. MIT Technical Report 2006-069.

[6] J. Beal and G. Sussman (2005), Biologically-Inspired Robust Spatial Programming. MIT Technical Report AI Memo 2005-001.

[7] J. Beal and J. Bachrach (2008), Cells Are Plausible Targets for High-Level Spatial Languages. Self-Adaptive and Self-Organizing Systems Workshops, 2008: 284-291

[8] A.Kondacs (2003), Biologically-Inspired Self-Assembly of Two-Dimensional Shapes Using Global-to-Local Compilation. IJCAI'03 Proceedings of the 18th international joint conference on Artificial intelligence.

[9] A. Bhattacharyya (2006), Morphogenesis as an Amorphous Computation. Proceedings of the 3rd conference on Computing frontiers:53-63

[10] J.J. Monaghan (1988), An introduction to SPH. Computer Physics Communications 48(1):89-96

[11] S. Clavet, P. Beaudoin, and P, Poulin (2005), Particle-based Viscoelastic Fluid Simulation. Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation:219-228

[12] J. Onderik and R. Durikovic (2008), Efficient Neighbor Search for Particle-based Fluids. Journal of Applied Mathematics, Statistics and Informatics (4):29-43