Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 1494 – 1502

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Agent-based algorithm for spatial distribution of objects

Nathan Collier[a,*], Marcin Sieniek[b]

[a]*Applied Mathematics and Computational Science and Earth and Environmental Sciences and Engineering*
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
[b]*AGH University of Science and Technology*
Kraków, Poland

Abstract

In this paper we present an agent-based algorithm for the spatial distribution of objects. The algorithm is a generalization of the bubble mesh algorithm, initially created for the point insertion stage of the meshing process of the finite element method. The bubble mesh algorithm treats objects in space as bubbles, which repel and attract each other. The dynamics of each bubble are approximated by solving a series of ordinary differential equations. We present numerical results for a meshing application as well as a graph visualization application.

Keywords: bubble mesh, meshing, graph visualization, agent-based

1. Introduction

This paper describes an agent-based algorithm for the even distribution of objects in a region. The technique is a generalization of the ideas originally presented in the work of Shimada [1–4] for the meshing of domains for use in the finite element method (FEM). The finite element method for solving partial differential equations [5–7] depends on a partitioning of the domain into elements. The elements are usually polyhedra, commonly triangles, quadrilaterals, tetrahedra, and hexahedra. This partitioning process is referred to as meshing in the finite element community.

In typical FEM simulations, the boundary of the domain of interest is originally specified. The task of meshing is two-fold: (1) generate a set of vertices in the interior of the domain of interest and (2) determine the connectivity of these vertices which will define the elements. The mesh generated has direct consequences on the quality of approximation of the FEM. Elements which have high aspect ratios (so called sliver elements) are undesirable, leading to a decrease in approximability of gradients of a finite element space. An evenly distributed set of vertices in the domain interior is critical to generating a mesh suitable for the FEM. Shimada's work, called bubble mesh, treats the vertices of the discretization as bubbles, which repulse each other when too close. He then proposes the solution of the bubble dynamics as a series of ordinary differential equations in time.

In this work we will generalize this idea as a method for even distribution of objects in space. Then we apply it to the visualization of graphs using a force-directed approach [8, 9]. A drawback of force-directed approaches to graph visualization is long run times. We will demonstrate that our approach is amenable to an agent-based implementation and therefore is trivially parallelizable on shared memory architectures.

*Corresponding author
Email address: nathaniel.collier@gmail.com (Nathan Collier)

doi:10.1016/j.procs.2012.04.164

2. Generalized Bubble Mesh Algorithm

We will represent N bubbles as a set of tuples containing the bubble center location and radius, $\{(\mathbf{x}_i, r_i)\}_{i=1,N}$. The bubble dynamics are governed by a set of ordinary differential equations,

$$\{M\ddot{\mathbf{x}}_i + C\dot{\mathbf{x}}_i + K\mathbf{x}_i = \mathbf{f}_i\}_{i=1,N} \tag{1}$$

subject to initial conditions and where $\dot{\mathbf{x}}_i$ and $\ddot{\mathbf{x}}_i$ are the first and second time derivative of the bubble centers, $\mathbf{f}_i$ is a force composed of inter-bubble interactions, and M, C, and K are constants. The system corresponds to a mass-spring-damper, well known from the field of Newtonian physics. Although the systems are linked by the forcing, we will solve them independently. The exact dynamics are not of interest in these applications, only that an even distribution of objects in space is achieved.

We will solve these equations using standard ordinary differential equation techniques. We do not care about accuracy in our solver, only for stability. To simplify the notation, we will consider the solution of one ODE system and do the same for each system. We rewrite the second order ODE as a system of first order ODEs by introducing a dummy variable $\mathbf{z} = \dot{\mathbf{x}}$.

$$\begin{cases} \dot{\mathbf{x}}_i = \mathbf{z}_i \\ \dot{\mathbf{z}}_i = \frac{1}{M}\left(\mathbf{f}_i - C\mathbf{z}_i - K\mathbf{x}_i\right) \end{cases}$$

These expressions for the first time derivatives are used to evolve the system discretely in time using a forward Euler approach, detailed in algorithm 1. We will indicate the time step with the variable t in superscript. The algorithm continues until the largest bubble velocity falls below a tolerance, V_{rest}.

Algorithm 1 Second order ODE solve algorithm for the i^{th} system

1: Set initial conditions $\mathbf{x}_i^0 = \mathbf{x}_i(0)$, $\mathbf{z}_i^0 = \mathbf{0}$
2: $t = 1$
3: **while** $\max\left(\{\mathbf{z}_i^{t-1}\}_{i=1,N}\right) > V_{rest}$ **do**
4: Compute inter-bubble force $\mathbf{f}_i$
5: $\dot{\mathbf{z}}_i^t = \frac{1}{M}\left(\mathbf{f}_i - C\mathbf{z}_i^{t-1} - K\mathbf{x}_i^{t-1}\right)$
6: $\mathbf{z}_i^t = \mathbf{z}_i^{t-1} + \Delta t\, \dot{\mathbf{z}}_i^{t-1}$
7: $\mathbf{x}_i^t = \mathbf{x}_i^{t-1} + \Delta t\, \mathbf{z}_i^{t-1}$
8: $t = t + 1$
9: **end while**

2.1. Designing inter-bubble forces

When bubbles come close together, they exert a force on each other. It is useful to define $\mathbf{n}_{ij}$ as the unit normal vector pointing from the center of the i^{th} bubble toward the center of the j^{th} bubble. Similarly we define the distance between bubble centers as d_{ij}.

When using the method for meshing, the radii of each bubble are used to define a region of influence around a bubble center. If bubbles overlap, then they repel each other, if two bubbles come close to each other, they attract each other, and if they are just touching, the bubbles are in perfect balance. We can define the forcing of the i^{th} bubble with respect to the j^{th} bubble as

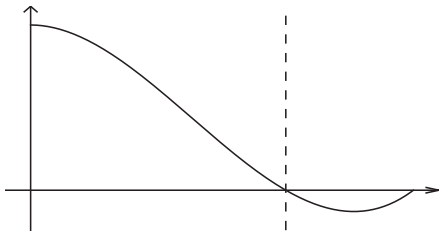
$$\mathbf{f}_b(r) = \begin{cases} \left(\frac{5}{14r_0^2}r^3 - \frac{19}{28r_0}r^2 + \frac{9r_0}{28}\right)\cdot \mathbf{n}_{ij} & \text{if } r < \frac{3}{2}r_0 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

where $r = d_{ij}$ and $r_0 = r_i + r_j$. This function is plotted in figure 1a.

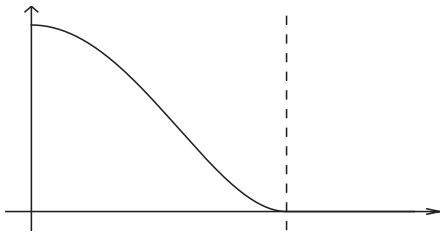
In other situations, it may be desirable to simply repel bubbles. In this case, we can define

$$\mathbf{f}_r(r) = \begin{cases} \left(1 - \left(\frac{r}{r_0}\right)^2\right)^2 \cdot \mathbf{n}_{ij} & \text{if } r < r_0 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

where $r = d_{ij}$ and $r_0 = Ar_i$. The constant A can be used to increase the area of influence around the bubble. This function is plotted in figure 1b.



(a) $\mathbf{f}_b(r)$, force function which repels overlapping bubbles and attracts those that are close.



(b) $\mathbf{f}_r(r)$, force function which only repels.

Figure 1: Sample inter-bubble force functions

Finally, it may be desirable to attract certain bubbles, for example, in graph visualization we want related items to be attracted to each other. In this case we add a force analogous to a spring,

$$\mathbf{f}_s(r) = k\,(r - r_0) \cdot \mathbf{n}_{ij}$$

where r_0 is the undeflected length of the spring, in this case $r_0 = Bd_{ij}$ and k is the spring constant and B is a constant used to increase the undeflected length.

3. Agent-based Approach

The method presented in section 2 is particularly well-suited for distributed execution, due to some of its features:

- the task can be trivially decomposed to a set of subtasks,
- solving an equation based on inaccurate or incomplete knowledge about the rest of the system still leads to improvement of the global state thanks to its stability,
- only limited communication is required to propagate changes in the system,
- the tasks are not tied to any specific resources, so they can be migrated between computational nodes as such need arises.

In the following subsections we develop a formal description of the concurrency model using the agent-based methodology.

Agent architecture

The Multi-Agent System (MAS) is composed of a few elements:

- an agent platform, which abstracts the computation from the physical resources,
- a set of agents - intelligent, active objects,
- an environment, agents can interact with.

We associate an agent A_i with each bubble $(\mathbf{x}_i, r_i)$. Since all bubbles are similar, the agents are homogeneous, have the same set of goals and display the same capabilities. For a given agent, anything else it can observe (including other agents) constitutes the environment.

Let us specify the MAS corresponding to the aforementioned problem in terms of Belief-Desire-Intent (BDI) methodology. BDI is an established agent architecture, which formalizes agent's knowledge, goals and strategies ([10], [11]). In this approach, *beliefs* donate an agent's picture of the environment, *desires* stand for agent's goals and *intents* represent a set of possible actions an agent can take.

Agent's beliefs

The exact solution of problem (1) would require each equation in the system to be solved based on the up-to-date solutions of all other equations. In the language of agents, it would mean the continuous need to acquire a global view of the whole system. Not only would this break the assumption of agent's incomplete knowledge, imposed by the majority of the agent research community ([12], [13]), but it would also result in an inefficient program, due to repetitive global barriers.

Instead, based on the observation that the actual bubble dynamics is irrelevant in this application, we equip the agent solely with the partial, presumably out-dated knowledge of its surroundings i.e. its *belief* on what the environment looks like.

Agent's desires

Agent's goals are three-fold. Primarily, it desires to determine its equilibrium position in space, optimizing the distances to neighboring bubbles. As a secondary goal, an agent prefers that its bubble stay close to the bubbles it is incident with in the corresponding graph (i.e. there exists an edge between them). Finally, when not necessary, an agent would better not move its bubble.

Agent's intents

This part of an agent's definition reduces to only two possibilities. To achieve its objectives, at each stage an agent can make its bubble attract or repel the neighboring bubbles.

It is important to note that we do not impose the above bahavior as a chain of conditional expressions. Rather than that, the above reasoning is realized with a set of self-adjusting formulae, which originate from the word of physics. This way, agent's 'intelligence' is donated in an elegant, 'fuzzy' way, making the agent's software code clean and concise.

Communication

An agent A_i needs to communicate with the environment in order to:

- notify other agents about the changes to $\mathbf{x}_i$,
- collect from other agents the changes to $\{(\mathbf{x}_j, r_j)\}_{j\neq i}$,
- determine whether the stop condition is met.

This can be realized in several ways. For example agents can pass the updates according to a predefined tree-shape scheme or update A_j with the position $\mathbf{x}_i$ whenever they need $\mathbf{x}_j$ for further computing.

In case of an underlying shared-memory architecture, a more straightforward approach can be taken: Bubbles' positions can be stored in the shared memory and updated disregarding the potential race conditions, when bubble's position is determined based on yet-to-be-updated information about the position of its neighbors. As we show in section 4 such a 'negligent' behavior is not only well justified by the computational cost, but it also degrades the results in an acceptable manner.

4. Numerical Results

In this section we present some results which demonstrate the efficacy of the algorithm proposed. Furthermore we will demonstrate that the results are qualitatively insensitive to parallelization.

Meshing

The bubble mesh algorithm was originally intended to be used in the point insertion process for meshing. Given the boundary data for the country of France, we use the bubble mesh algorithm to smoothly insert points into the domain. The boundary nodes of the representation were obtained from the `CountryData` function in `Mathematica` [14]. We used `Triangle` [15] to triangulate this boundary data set to obtain a triangulation of the interior. We then create bubbles for each node in the mesh, where the radius of each bubble is chosen to be the length of the longest edge connected to the node. This initial configuration is shown in figure 2a, where each bubble had its radius drawn to scale and colored based on its size. The algorithm is then run only using the inter-bubble force labeled, $f_b(r)$. The result after smoothing can be seen in figure 2b.

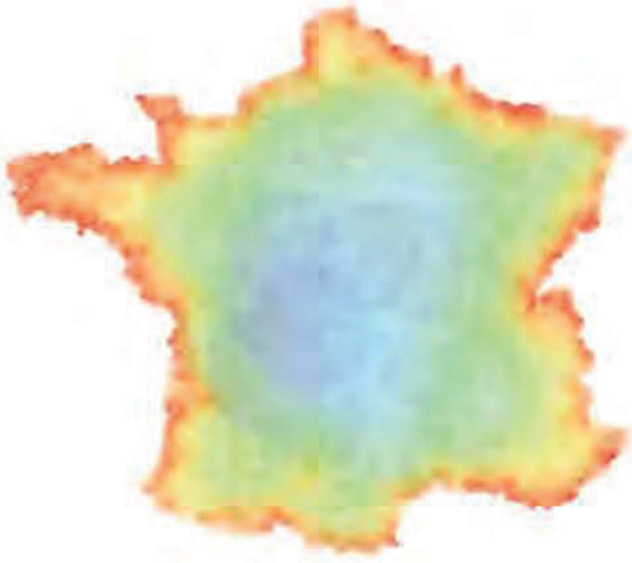
Graph Visualization

The first author of this paper contributes to an area of computational mechanics known as isogeometric analysis. To create a realistic test case, we took a community bibliographic file for publications in isogeometric analysis and cross-referenced coauthors of papers. An author is represented by a bubble whose radius corresponds to the number of publications the author has in the field. Two authors are linked if they publish a paper together.

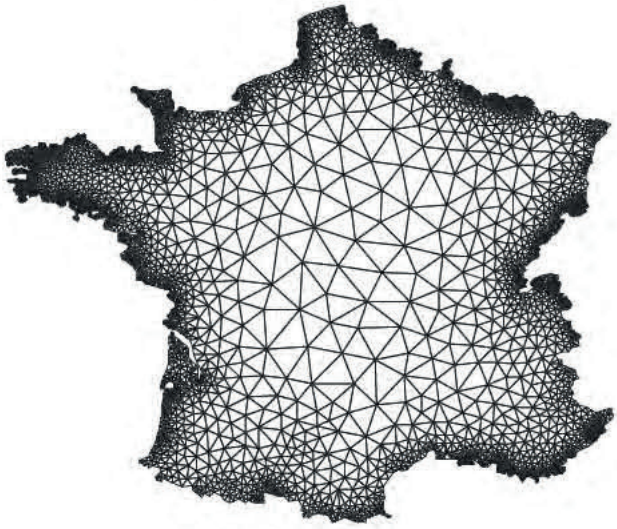
All bubbles repel each other, using the force function $\mathbf{f}_r(r)$ as defined previously in this paper. The area of influence has been increased substantially to keep the bubbles apart such that the text may be legible. For bubbles whose corresponding authors have published a paper together, we add a spring force, $\mathbf{f}_s(r)$ which attract the two bubble centers. The global mass constant M is chosen uniformly for each bubble. The dampening coefficient C is chosen such that the system is over-damped. A global stiffness coefficient K is added such that all bubbles tend to remain anchored to the center of the image.

We initialize the graph by placing all bubble centers in a circle as shown in figure 3. We then solve for the bubble centers using the system of ODEs defined previously. We continue solving until the velocity of the bubble centers slows to a specified resting velocity, V_{rest}. A sample final configuration is shown in figure 4a.

An implementation of the algorithm was evaluated on the above problem. A sample configuration is shown for the parallel version using eight agents in figure 4b, where each agent here is in charge of a collection of bubbles. Qualitatively, the solution is unaffected by the parallelization even though the total simulation time is different for each thread.



(a) Initial bubble configuration



(b) Smoothed mesh

Figure 2: Example mesh of France, smoothed by the bubble mesh algorithm.

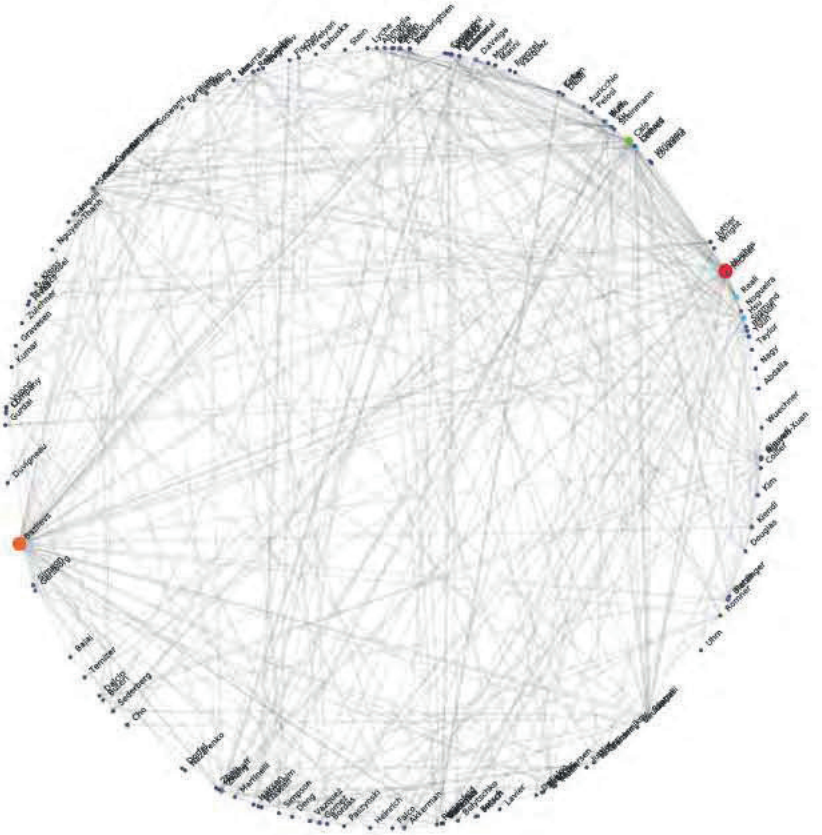
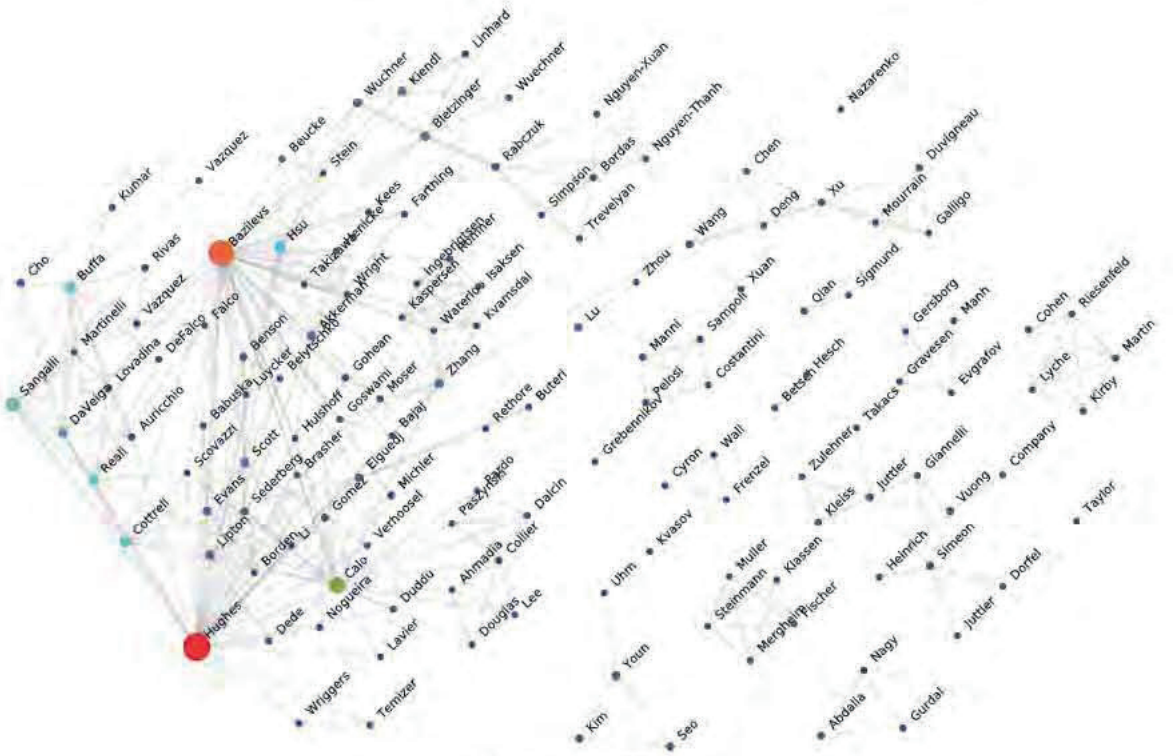


Figure 3: Initial configuration for visualization



(a) Computed sequentially

(b) Computed using 8 agents

Figure 4: Visualization of graph of coauthors who publish in the area of isogeometric analysis

5. Conclusions

We have presented an extension of the bubble mesh algorithm as a generic algorithm for the arrangement of objects in space. We further demonstrate that the algorithm is well-suited to an agent based approach. We present numerical results which demonstrate the algorithm's usefulness. Furthermore these numerical tests show that while specific dynamics may vary when using the agent-based approach, the outcome is qualitatively the same.

Acknowledgements

The work reported in this paper was funded by Polish National Science Center grant no. NN 519447739.

References

[1] K. Shimada, D. Gossard, Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing, in: ACM Third Symposium on Solid Modeling and Applications, 1995, pp. 409–419.
[2] A. Y. Shimada, K., T. Itoh, Anisotropic triangular meshing of parametric surfaces via close packing of ellipsoidal bubbles, in: The 6th International Meshing Roundtable, 1997, pp. 375–390.
[3] S. Yamakawa, K. Shimada, High quality anisotropic tetrahedral mesh generation via packing ellipsoidal bubbles, in: The 9th International Meshing Roundtable, 2000, pp. 263–273.
[4] S. Yamakawa, K. Shimada, Anisotropic tetrahedral meshing via bubble packing and advancing front, in: First M.I.T. Conference on Computational Fluid and Solid Mechanics, 2001.
[5] T. Hughes, The finite element method: linear static and dynamic finite element analysis, Dover, 1987.
[6] L. F. Demkowicz, Computing with *hp*-ADAPTIVE FINITE ELEMENTS: Volume 1 One and Two Dimensional Elliptic and Maxwell problems, CRC Press, United States of America, 2007.
[7] G. Strang, G. Fix, An analysis of the finite element method, Wellesley Cambridge Press, 2008.
[8] T. M. J. Fruchterman, Edward, E. M. Reingold, Graph drawing by force-directed placement (1991).
[9] P. Eades, Graph drawing by force-directed placement, Congressus Numerantium (1984) 149–160.
[10] M. Rao, A.S. Georgeeff, Modelling rational agents within a BDI architecture, in: Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning, KR & R, Cambridge, USA, 1991, pp. 473–484.
[11] M. Rao, A.S. Georgeeff, BDI agents: From theory to practice, in: L. V. (Ed.), Proceedings of the First Internation Conference on Multiagent Szstems, AAAI Press Menlo Park, Cambridge, USA, 1995, pp. 312–319.
[12] G. Dobrowolski, Technologie agentowe w zdecentralizowanych systemach informacyjno-decyzyjnych, Uczelniane wydawnictwa Naukowo-Dydaktyczne, 2002.
[13] K. Cetnarowicz, Problemy projektowania i realizacji systemów wieloagentowych, Uczelniane wydawnictwa Naukowo-Dydaktyczne, 1999.
[14] I. Wolfram Research, Mathematica 8.0, Wolfram Research, Inc., 2010.
[15] J. R. Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, in: M. C. Lin, D. Manocha (Eds.), Applied Computational Geometry: Towards Geometric Engineering, Vol. 1148 of Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 203–222, from the First ACM Workshop on Applied Computational Geometry.