

CAD/CAM for Nanoscale Self-Assembly

Ari Requicha
and Daniel
Arbuckle

University of
Southern
California

IEEE *Computer Graphics and Applications* has a long tradition of addressing CAD/CAM problems, a tradition that extends even beyond the well-known special issue on solid modeling edited by Herb Voelcker and Ari Requicha in March 1982. Here, we discuss the conversion of a standard representation for a solid object (for example, a boundary representation) into a set of complex message-passing instructions for manufacturing the object by a distributed system—a swarm of robots.

Our work's motivation comes from the need to develop methods for building devices and systems at the nanoscale. Nanotechnology is currently hampered by a lack of manufacturing processes capable of both high throughput and high resolution. Direct-writing systems such as electron beam or scanning-probe microscopy are serial and slow but achieve high resolution, whereas various lithographic techniques are parallel and fast but typically have resolutions on the order of the tens of nanometers. Self-assembly, that is, the spontaneous assembly of components into the desired structures, is a promising concept that might lead to a revolution in nanomanufacturing. Self-assembly processes are all around (and inside of) us. For example, atoms arrange themselves into crystals, and biomolecules assemble themselves into organelles, cells, and tissues. Many of these processes are driven by thermal agitation, which brings components to positions and orientations where they contact others and are recognized by them. Molecular recognition often plays an important role in these processes.

Our approach

Several groups are studying artificial, synthetic self-assembly systems—for example, Len Adleman's at the University of Southern California and Erik Winfree's at the California Institute of Technology—but the structures built thus far tend to be symmetric and not suitable for many of the envisaged applications in nanoelectronics, nanoelectromechanical systems, or nanobiotechnology. The components used in this work are passive, that is, their motion is random and caused by the environment, and their actions are limited to attaching themselves to other components when certain interface conditions are satisfied. At USC's Laboratory for Molecular Robotics, we have been studying an alternative paradigm, called *active*

self-assembly, in which the components are simple robotic agents. Our agents also execute environment-driven random walks, but can sense other agents, move in simple ways, attach and detach themselves to and from other agents, execute simple computational rules, and—importantly—send messages to other agents with which they are in contact. These are characteristics we can reasonably expect future nanorobots to have.

We have shown that active self-assembly can build arbitrary polygons in the plane, and we believe that we can extend this approach to three dimensions. The resulting polygons are approximated by spatial enumerations, that is, by sets of pixels on an orthogonal grid.¹ We assume that the agents are square and that each occupies exactly one pixel. Each agent initially executes a random walk, and when it encounters another agent, it might attach itself to it and exchange messages. All of the agents are identical and have the same program, which consists of a set of rules. (This program can itself be considered a process-oriented or procedural representation of the desired shape.)

We have investigated two approaches to active self-assembly. The first approach programs the agents as finite-state machines (FSMs), and the second one uses purely reactive agents without memories. The two approaches can both build the same structures, but the second has the interesting properties of self-repair and (to a limited extent) self-reproduction. We also have built two compilers, one for each type of agent, which convert a CAD representation of the desired polygon into a program for the self-assembling agents. We outline the two approaches later, and details appear elsewhere, as well as in Arbuckle's PhD dissertation, which is in preparation.²⁻⁴

Finite-state machine agents

Figure 1 shows several stages in the construction of a polygon's boundary by an FSM-controlled agent swarm. Different colors denote the states: gray is a random walking or wandering state, red is a seed state, and blue is a boundary state. The figure pictorially shows the applicable FSM rules on the right. Build edge rules are indicated by B1, B2, B3, and so on—one for each edge. Build rules have one integer parameter, which is a hop count that determines an edge's length. As an edge progresses, the hop count is decremented.

Let's follow the construction in Figure 1 by applying the rules shown. We outline the process briefly as follows: The structure begins with a seed (see Figure 1a). The seed continuously sends a build message, per rule 1. The parameter 2 in the build message is the length of the edge to be built. When a wandering agent contacts the seed (see Figure 1b), it receives the build message, attaches itself to the seed and forwards another build message with a lower hop count (see Figure 1c), according to rule 2. Figure 1d is an intermediate stage in the construction. A vertex is reached when the hop count is zero (see Figure 1e). Then, a new edge is started with a length specified in the rules, for example, a (B1,0) message will trigger a (B2,2) message by rule 3, where the parameter 2 is the length of the second edge. The process continues (see Figure 1f) until we build the square shown in Figure 1g. (Rules similar to 1 through 4 are used to complete the square's boundary.) The compiler automatically builds these rules. If an agent contacts another agent that is not attempting to communicate outward from the contact side, there will be no permanent attachment.

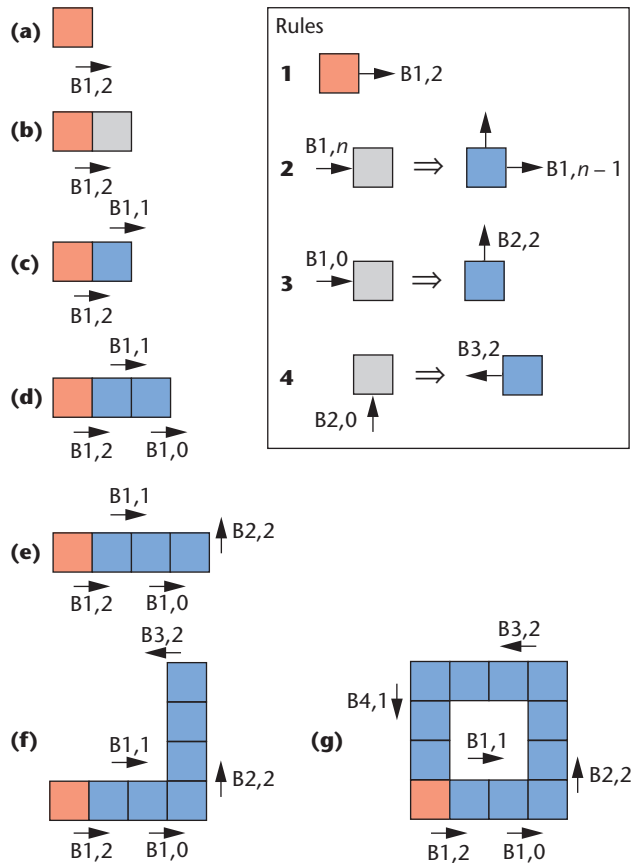
Once a polygon's boundary is complete, its interior is filled in by using a simple rule: when a wandering agent contacts a boundary agent, both move in by one unit, and the inside agent is released and continues to wander, until the whole interior is full. We use this same approach to fill polygons built by the reactive scheme introduced later.

Figure 2 shows a disconnected polygon built by the FSM technique described here. Perhaps the major disadvantage of this approach is its dependence on a special agent that serves as the seed. As a result, if the seed is lost due to a malfunction or because the structure is broken by some external disturbance, the process fails. The reactive approach discussed next can regenerate a structure from any of its parts and therefore is self-repairing.

Reactive agents

Here the agents have no state. They are programmed by a set of rules that tell them what to do when they receive specific messages. The typical reaction is to send one or more messages in the directions specified by the rules. There are grow edge (GE), grow vertex (GV), acknowledge edge (AE), and acknowledge vertex (AV) messages. The messages in the simplified example of Figure 3 (next page) have up to three parameters. Thus, for example, in GE(X, 1, 2), X is the identity of the edge, 1 denotes the position of the agent within the edge, and 2 is the edge length. An edge of length 2 is composed of three agents, with positions 0, 1, and 2; the vertices are not considered part of the edge. As in the previous approach, hop counters keep track of edge length. Figure 4 shows pictorially the rules relevant to this example.

Figure 3 reveals two sets of messages, growth and acknowledgment messages, propagating along the structure, in opposite directions, one set with increasing hop counts and the other with decreasing hop counts. This ensures that any connected part of the structure is stable over time and will lead to the regeneration of the entire structure. To see why this is true consider just the two first agents, A and B. Note that the

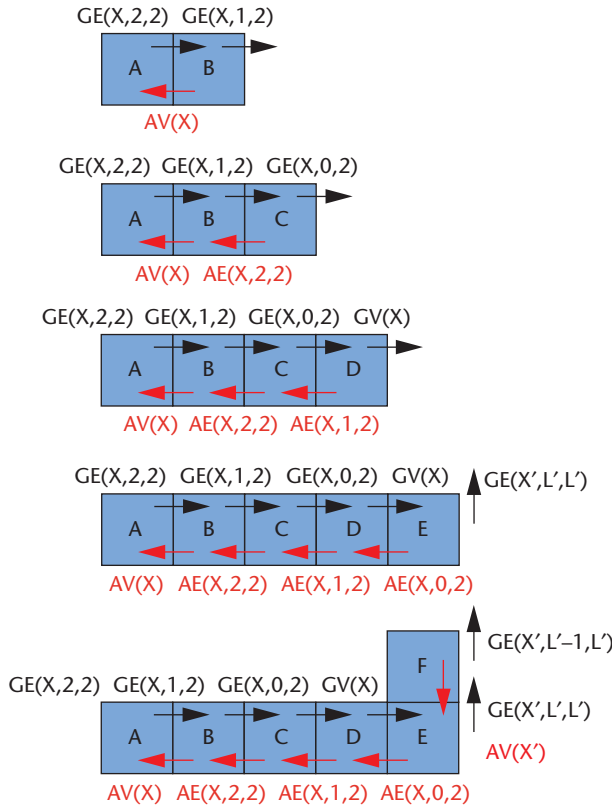


1 Building a square's boundary by using agents programmed as finite-state machines. The states are shown by the agents' colors.

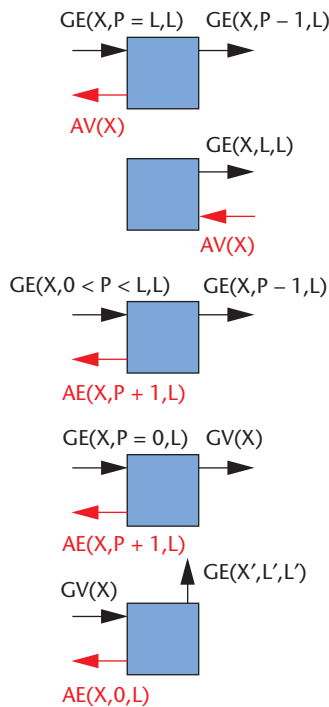


2 A complex, disconnected polygon built by finite-state agents.

GE message sent from A to B triggers an AV message sent back to A. This in turn causes the same GE message to be sent to B again, and so on. The two agents form a stable loop that will remain attached, exchange messages indefinitely, and serve as a starting point for building the whole structure. The same reasoning holds for any other connected portion of the structure, and therefore the structure will repair itself as long as at least a pair of connected agents is available. In addition, if the structure is broken and the pieces moved away, each stable pair will build an identical copy of the original structure, in a process reminiscent of cell mitosis. Figure 5 illustrates this self-reproduction process. The initial key-shaped structure was broken during con-



3 Building an L-shape with reactive agents.



4 Reactive rules used for the example in Figure 3.

struction and two pieces moved away, resulting in self-repair of the original structure and building of two additional copies.

The CAD/CAM link

A swarm of agents is a distributed manufacturing system, albeit an unusual one. The agents might become the desired structure once it's finished, or serve as a scaffold onto which the desired components can attach—perhaps by passive self-assembly. The agent programs are manufacturing instructions. Deriving such manufacturing instructions from a CAD model is the essence of the CAD/CAM link, which is as important today as it has ever been.

Since any common solid representation can be converted into a boundary representation, we can assume that the desired objects are defined by their boundaries. Thus, in a 2D polygonal world, we assume that our objects are represented by their edges. In the schemes presented here, we first build an object's boundary and then fill its interior—using the filling process described in the section on FSMs. The filling scheme is simple and is equivalent to moving agents across the boundary. (More efficient filling schemes exist, but they are more complicated and will be ignored here.) Message passing with hop counters controls edge building, as implied in the examples presented previously.

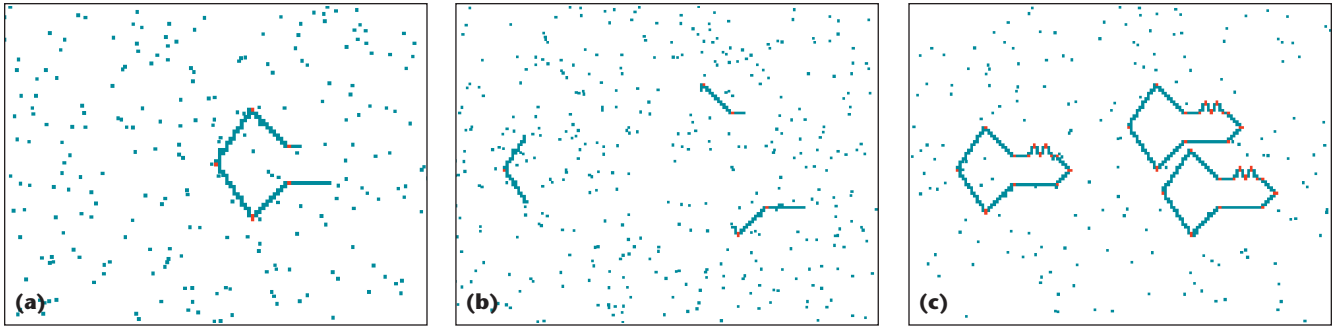
For concreteness, consider the reactive-agent scheme. The following is a simplified description of the compiling process that generates reactive rules from the CAD data. Each edge in the polygon representation has a label or identifier. For each edge we generate a build rule with a hop counter that determines the edge's length. An agent that attaches itself to an edge's end knows it's a vertex because it receives a message with a zero hop count. Therefore, in addition to edge rules, we need to generate vertex rules, so that a vertex at the end of edge labeled X knows that it must start issuing build messages for the edge Y that follows X in the boundary. Of course, there are many complications, but the essence of the process is simple.

Figures 1 through 4 only show examples of orthogonal edges, but this scheme can build polygons with edges at nonorthogonal angles, as Figure 5 shows. Building a nonorthogonal edge is similar to scan converting a line segment, since the edge must be approximated by pixels (agents). Each edge is processed by the compiler using a modified Bresenham algorithm. Because our agents cannot be connected diagonally, vertex to vertex, the Bresenham algorithm must be modified to ensure that connections are established only between the sides of the square agents.

The compiler completely and automatically generates the rules for driving the self-assembly process. In other words, no human intervention is needed to convert the CAD data into the manufacturing instructions.

Conclusion

It has been tacitly assumed until now that manufacturing at the nanoscale will be an extension of its macro- or microcounterparts. But this assumption might be partially or totally wrong. Instead, distributed systems such as those described here might have an important role to play. These systems exploit randomness rather than striving for deterministic, precise positioning of components, and have some of the desirable characteristics of bio-



5 (a) An intermediate stage of the construction of a key-like object. (b) Breaking the object into three pieces. (c) The three fully built resulting objects.

logical systems, for example, robustness and self-repair. Traditional CAD models still seem appropriate for many applications at the nanoscale, but the CAD/CAM link must be rethought and adapted to the new and emerging manufacturing processes and constraints.

Our nontraditional approach to building objects is independent of spatial scale but is most attractive at the nanoscale. The major challenge we face toward implementing active self-assembly today comes from the hardware side: we need to build large numbers of small robots. But micro and nanorobotics are progressing rapidly and should come to the rescue in the not-too-distant future. ■

Acknowledgments

This article is based in part on a keynote address titled “And Now for Something Completely Different: Building Shapes by Self-Assembly,” presented by Ari Requicha at a joint session of the ACM Symposium on Solid Modeling and Applications, and of the Shape Modeling International Conference, Genova, Italy, 7–11 June 2004. This work was supported in part by the National Science Foundation under grants IIS-99-87977, EIA-01-

21141, and DMI-02-09678, and Cooperative Agreement CCR-01-20778; and by the Okawa Foundation.

References

1. A.A.G. Requicha, “Representations for Rigid Solids: Theory, Methods, and Systems,” *ACM Computing Surveys*, vol. 12, no. 4, 1980, pp. 437-464.
2. D.J. Arbuckle and A.A.G. Requicha, “Active Self-Assembly,” *Proc. IEEE Int’l Conf. Robotics and Automation (ICRA)*, IEEE Press, 2004, pp. 896-901.
3. D.J. Arbuckle and A.A.G. Requicha, “Shape Restoration by Active Self-Assembly,” *Proc. Int’l Symp. Robotics and Automation (ISRA) CD-ROM*, IEEE Press, 2004, pp. 173-177.
4. D.J. Arbuckle and A.A.G. Requicha, “Self-Repairing Self-Assembled Structures,” to be published in *Proc. IEEE Int’l Conf. Robotics and Automation (ICRA)*, 2006.

Readers may contact Ari Requicha at requicha@usc.edu and Daniel Arbuckle at daniel.arbuckle@usc.edu.

Readers may contact Miguel Encarnação at me@imedia-labs.com.

Evaluating the Performance of Software Engineering Professionals

By Lawrence Peters
Software Consultants Int.

Surprisingly, the most common means of reviewing software engineering professionals actually have the effect of demotivating them and reducing their performance level. This ReadyNote advocates an alternative method for evaluating personnel based on the Balanced Scorecard. \$19
www.computer.org/ReadyNotes

IEEE ReadyNotes

