Growth Forms

George W. Hart
Computer Science Dept.
Stony Brook University
Stony Brook, NY 11790 USA
george@georgehart.com

Abstract

This paper presents work-in-progress on techniques for procedurally "growing" organic-looking forms. The goal is to design and implement techniques for expressing simple algorithmic operations that can be combined in various ways to produce forms that develop from an amorphous blob to a structured cellular assemblage. Growth "buds" in a cellular assemblage each execute short programs that determine local parameters of growth, e.g., directionality, branching, cell size, cell shape, and color. These are executed in a software simulator package that generates both graphical (2D) and geometric (3D) output. Every step along the way determines a triangulated manifold boundary that can be fabricated by solid freeform fabrication equipment. I plan to incorporate these techniques into future sculpture that combines mathematical ideas with organic sensibilities.

Introduction. Nature provides an enormous range of attractive organic forms which have inspired artists through the ages [2]. In previous work, I have created simulated organic forms reminiscent of underwater sea life via mathematical models of an entire organism's static structure [3]. This paper explores simple techniques for mimicking various organic forms via a simple dynamic growth process, e.g., Figure 1. The goal is to create organic-looking geometric sculpture which can be built on 3D printing machines. It relates to well-known techniques of L-Systems [6] but is designed to create smooth surfaces with a triangulated manifold boundary that can be fabricated by solid freeform fabrication equipment.

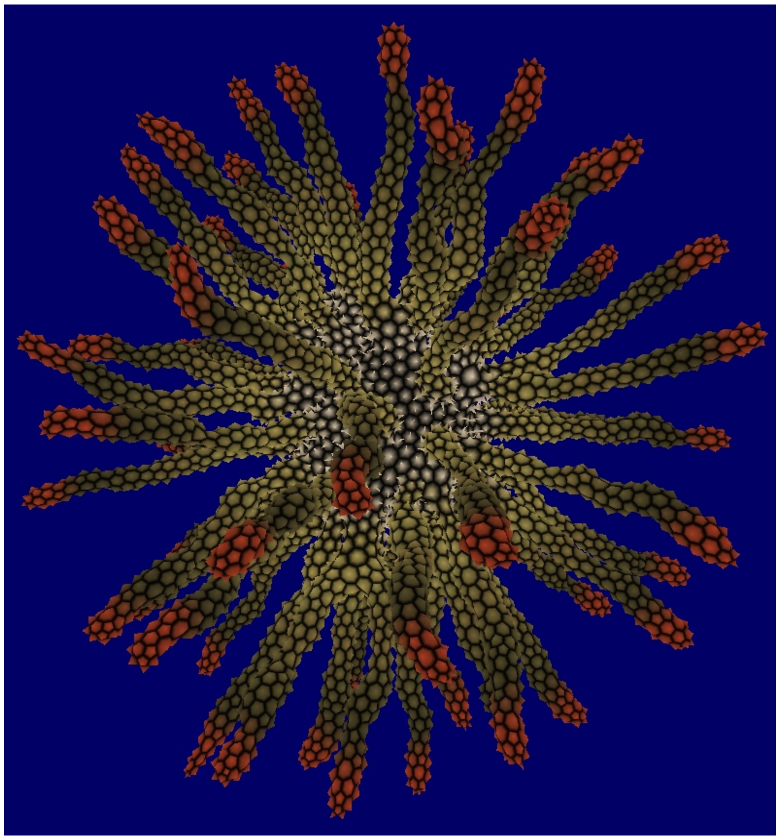

Figure 1: *A cellular form that develops by outward growth at the tip of each arm.*

A form here is represented as a network of “cells” on its surface, analogous in some ways to skin cells on an organism. Each cell has properties assigned to it, including size, color, and texture (e.g., bumpy or smooth). A cell may divide into two cells, which are topologically connected into the space that the single cell formerly occupied. Their geometry is then determined by a spring energy relaxation model that calculates new positions for the cells so all are the appropriate distance from their neighbors. As a consequence, after cell division the surface bumps out slightly to accommodate the additional cell. Some cells contain a special entity, called a “bud” that controls the growth and cell properties in their local area. Each bud follows an algorithmic program, allowing it to control how nearby cells divide, to change its growth properties over time, to split into two or more new buds, to check for collisions, and to do other simple operations. There may be several different bud types and any number of buds, each independently following the program for its bud type. The behavior for each bud type is specified in advance using a simple programming language. The growing areas around the various buds develop and interact to result in complex, visually attractive overall structures. Due to many random choices of exactly which cell divides when, the same program when executed repeatedly gives output that appears like different individuals of a single species. The project is not intended as a simulation of particular organisms or natural mechanisms. The goal is simply to produce sculptural forms that mimic the forms of various natural biological processes. The best means to understand the growth dynamics is to view the screen-capture videos at [4].

Origins: Symmetry via Growth. I began this project when thinking about how icosahedral symmetry rarely appears in nature. Certain microscopic viruses and radiolarians have icosahedral forms but, to my knowledge, no larger multicellular organisms do. As a fan of icosahedral symmetry, I could envision a simple growth process that in principle could lead to larger scale icosahedral organisms. The mechanism appears similar to natural biological growth processes, and nature has an enormous range of techniques, so why does this not appear in nature?

The proposed growth mechanism involves buds that promote growth in their local area. It is apparent from observing the type of apical growth one observes, e.g., in trees, that there is something special at the tip of each branch causing it to elongate. Suppose twelve such buds were randomly positioned in a small spherical blob of cells, each causing growth in their local area, perhaps by producing some sort of “growth hormone” that diffuses away from them and causes nearby cells to divide. If two buds happened to be very close to each other, the area between them should receive twice the dose of hormone and so grow faster than average, pushing the two buds away from each other. Conversely, if buds are very far from each other, most of the area between them has little hormone so does not grow. This appears to provide a stabilizing feedback mechanism that would, after some time, cause buds to be roughly equally spaced from their neighbors. So if twelve buds started out randomly distributed in a spherical ball of cells, I envision this mechanism producing an even distribution of buds, i.e., positioned as the vertices of an icosahedron. If the bud regions continued to grow, I would expect a kind of 12-armed almost-regular icosahedral “starfish” to result. But I know of no such biological organism. So I decided to “play god” and see if I could create one. And while I was at it, I made an environment for creating a range of other organic-looking forms that develop by bud-based growth.

Local cell division. In this model, cells are like small balls that stick to their neighbors in a monolayer surface or skin. Each is surrounded by a small number of adjacent cells, most typically five, six, or seven. This adjacency relation determines a graph I call the “cell graph.” Each cell is represented by a vertex. When cells are adjacent, there is an edge connecting the corresponding vertices. In most examples below, the skin has no “donut holes,” i.e., it is of genus zero, so the cell graph is planar, but that is not essential to the method. One example is shown of a process in which holes are grown, making high-genus forms.

The growth algorithms below ensure that the cell graph always corresponds to a triangulated surface. This simplifies some of the algorithms and makes it easy to convert to the STL file format for 3D printing machines. Furthermore, it corresponds visually to what one finds in nature, where cells in surfaces usually meet in groups of three [7]. However, unlike most real organisms, these forms are empty of all structure.

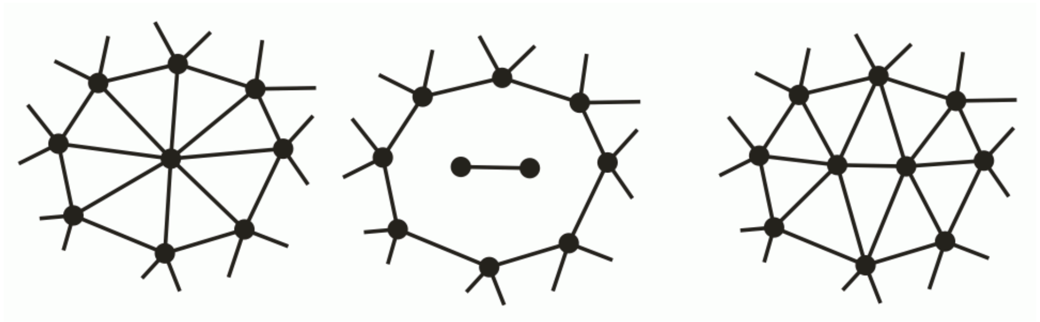

Figure 2: *Cell division process. If the center cell at left is chosen for division, it is temporarily removed and replaced with two cells that join each other. The surrounding n cells (here, n=8) connect via n+2 edges to the two new cells, to make a triangulated graph. Here, the topmost and bottommost of the eight surrounding cells each end up with one more edge than they began with.*

Given a cell graph and a cell to divide, we apply the procedure illustrated in Figure 2. It is a special type of "vertex splitting" designed to maintain a triangulated graph of approximately balanced degree. The vertex corresponding to the cell that divides is removed, temporarily creating an *n*-gon face if the cell had *n* neighbors. Two new vertices are placed in this face and connected to each other. These two cells are then connected to the *n* surrounding vertices, using exactly *n*+2 edges so that the result is triangulated. This means that most of the surrounding vertices connect to just one of the two new cells, but two of the surrounding vertices connect to both of the new cells. For those two surrounding vertices, their neighbor count will increase by one. As we do not want vertices of very high degree, we choose the surrounding vertex of lowest degree for one that will increment. The other is chosen to be opposite the first in the *n*-gon, as otherwise the two new cells would have unbalanced degrees. (If *n* is even, this completely determines the opposite vertex, but if *n* is odd we can round up or down, and do so in a way that increments the degree of the one with lower degree if they are not equal, otherwise randomly.)

Position determination. Cell division, as described above, is a topological operation on the cell graph. To form a geometric surface, we need to embed this graph in 3D space by assigning a position to each vertex. This is accomplished here by a "spring embedder" algorithm which treats each vertex as a massless node and each edge as a spring with a preferred length. We define an energy function associated with vertex pairs that is quadratic in the difference between the actual length and the preferred length. We choose coordinates for each node to minimize the sum of these spring energies. The actual global minimum is difficult to compute, but a simple iterative algorithm can repeatedly adjust one node at a time to minimize its contribution to the energy. This can be accomplished with a straightforward Newton-Raphson numerical method which converges very quickly. The actual method applied is similar to the method of [5] but adapted for 3D. (Another difference from [5] is that we use a spring for each pair within a distance of at most four hops, taking the shortest distance in the graph as the target distance. This is fast and seems to work because of the triangulated nature of these graphs.)

After a cell divides and the cell graph is updated, the spring embedder algorithm is used to locate the positions in space of the two new cells and relocate their surrounding *n*-gon. Assuming all edges are specified to be of unit length, this requires that the surface bump out to accommodate two cells where there was formerly one. (The preferred edge lengths can be specified to be any positive length, not just unity, as discussed below. Whatever they are, the algorithm seeks an embedding with actual lengths that closely match them.)

Bud behavior. Our forms are represented by a cell graph (its topology) and a position for each cell (its geometry). Given a form, the cell division and spring embedder mechanisms above are applied to produce a slightly modified form whenever a cell divides. The overall process begins with a small blob of cells, e.g., a tetrahedron of four cells, as that is the simplest triangulated polyhedral manifold. One possible mode of behavior is to allow any cell to divide, perhaps choosing at random repeatedly. We call this method "general growth" and starting from a tetrahedron, it produces "blobs" of arbitrary size.

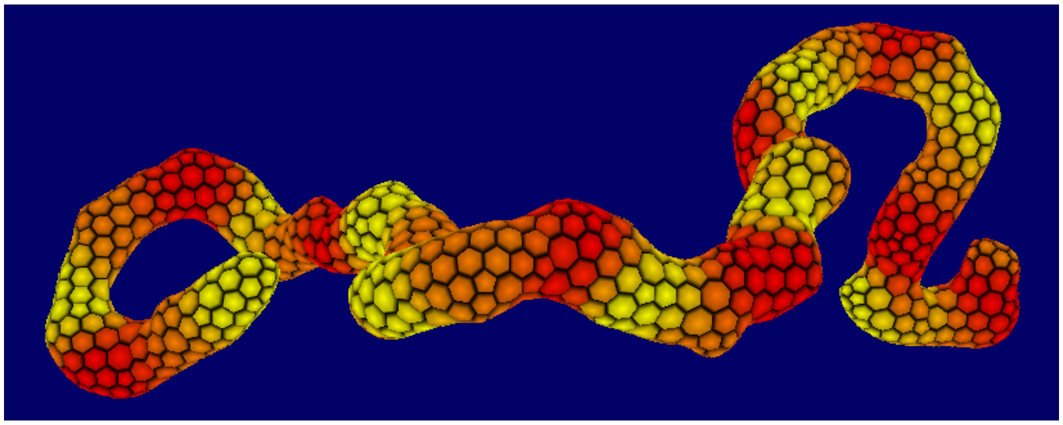

Figure 3: *Having just a single bud results in a snake-like form. If no specific tropism is specified, it wanders in a type of self-avoiding random walk in 3D.*

To control the form, we introduce *buds*, which may be thought of as small microcontrollers that sit inside some cells. In the examples below, there may be ten thousand or more cells, but at most a few dozen of them contain a bud. Each bud has a set of parameters that control which nearby cells should divide and how frequently. Currently this is specified in terms of the number of "hops" in the cell graph, i.e., a bud may specify that a randomly chosen cell within distance three of it is to divide. As nearby cells divide, the bud-containing cell is pushed along, like the tip of a growing plant. By restricting the hop distance to be small, a thin tube is formed. By allowing it to be larger, a fatter branch is generated. Conceptually, this corresponds to the bud producing "growth hormone" that diffuses either to only the adjacent cells or more widely to a larger neighborhood of cells.

If a bud-containing cell divides but the bud does not divide, the bud may end up in either of the two children cells. Generally, this is done at random, but it is natural to allow for a number of variations on this idea. For example, the bud may also split, so it becomes two independent buds, one in each of the two cells. In that case the two buds each continue independently to make nearby cells divide. They soon are pushed away from each other and become two independent branches. Continuing in this way, with repeated bud division, results in fractal branching structures.

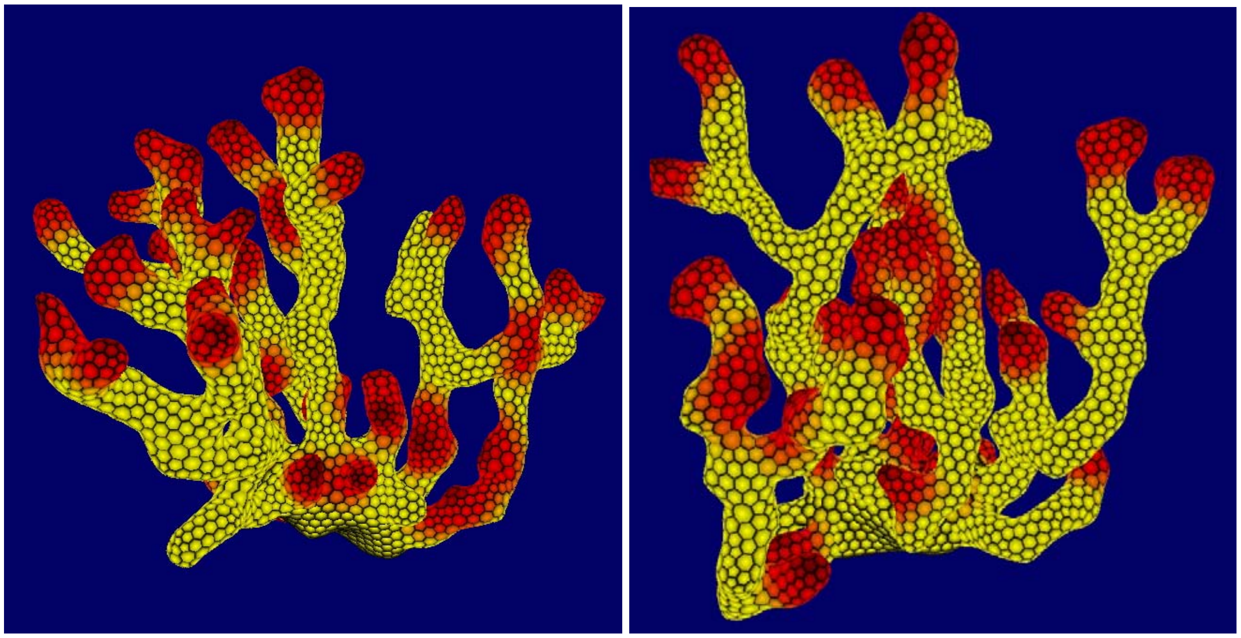

Figure 4: *Allowing buds to occasionally split into two buds gives a branching fractal form. Due to randomness of which cell is chosen to divide, one script run twice produces different "individuals" within a "species".*

When a bud-containing cell divides, we can take that opportunity to introduce a gentle steering mechanism. If we place the bud in the higher of the two cells, as measured by Z coordinate, the net effect is to introduce a vertical tropism. Without such a tropism, the tube takes a kind of a random walk through space, as in Figure 3. This is because the cell with the bud is “pushed forward” by nearby cell divisions, which depend on a pseudorandom number generator, sometimes favoring left or right, etc. With the vertical tropism, as in Figure 4, there is a feedback mechanism that encourages the bud to be at the top of the growing region. (One may think of this physically as floating buoyantly within the cell before division, so it ends up in the upper of the two children cells, or perhaps as attracted to light trickling down to the seafloor from the sea surface, but I can avoid biology and think of it merely in terms of which cell’s position maximizes a dot product with a target heading.)

Other bud properties control cell color and cell texture. When a bud causes a cell to divide, the two new cells are assigned the color and texture specified in the bud. So for example, if this bud color is gradually changing over time, the branch will have a gradient of colors along its length. The colors can be used artistically in computer graphics output. But for physical fabrication, I do not have access to a color 3D printer, so I introduced some parameters of cell texture to appear in uncolored fabricated output. For example, each cell can be flat, a bump, indented like a dimple, spiked with a hair, or any shape in between. As the position data for each cell is converted to an STL file for 3D printing, these parameters are used to determine the exact cell geometry. Details are omitted here, as techniques for geometrically texturing small regions are well-known [1].

Another variation on bud division is that buds contain a “line flag” which if set to *true* specifies they should maintain a connected line with neighboring buds. If before cell division a bud is in a cell adjacent to another cell containing the same kind of bud, but after cell division it is no longer adjacent, insert a new bud in the cell that separates the previous neighbor. The effect of this is that there is always an unbroken line of these buds. This can be used to create growth along an edge of a leaf-like form. Such a line can be seen in Figure 5, producing a negatively curved surface.

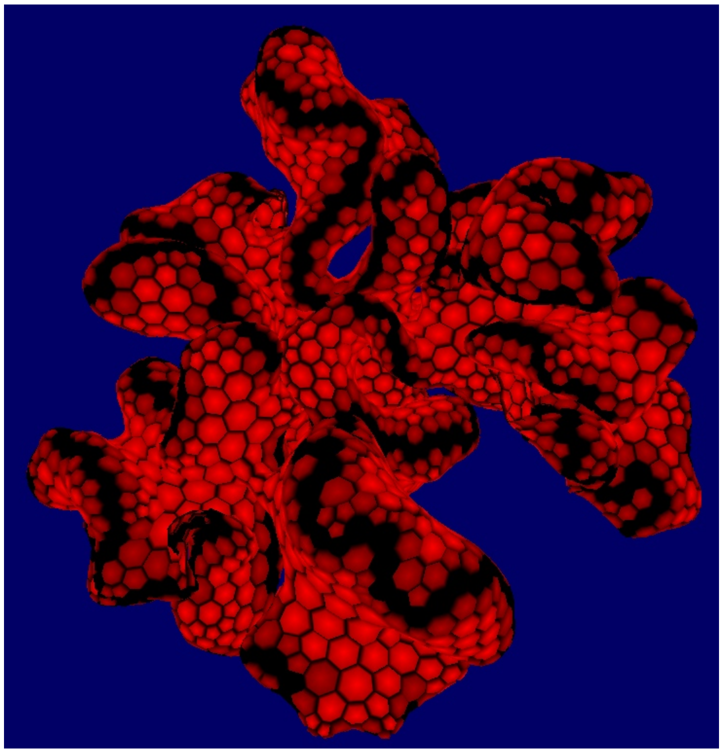
Figure 5: *The result of keeping a single line of buds (dark) is a negatively curved form.*

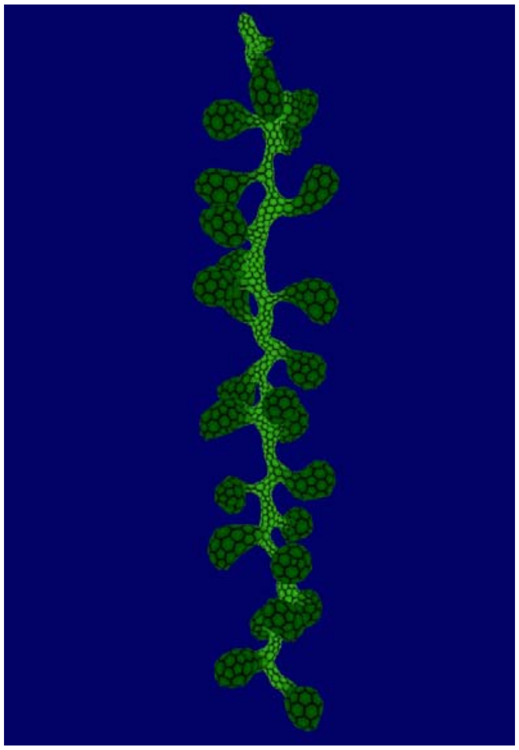
Figure 6: *Result of a program with two bud types, one for stem and one for pods.*

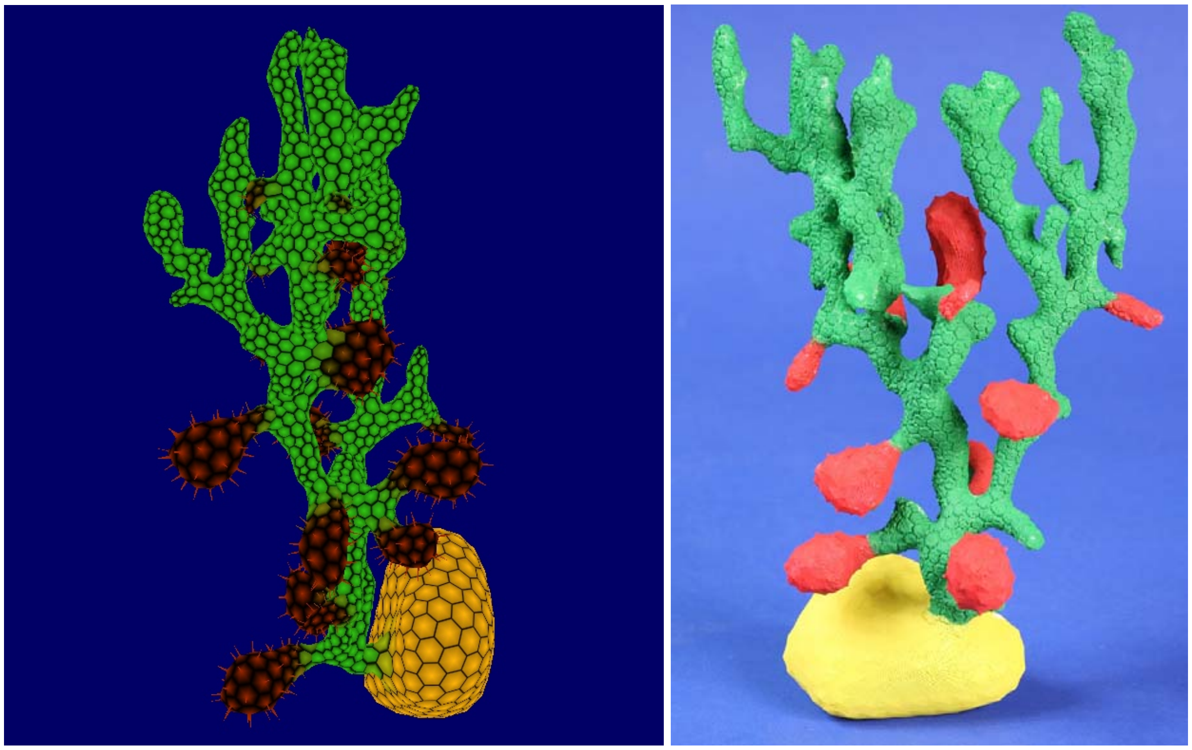
Figure 7: *Three types of buds allow a form with a blobular root, a branching stem, and spiky fruits, which are older and riper at the bottom. At right is a 3D printed model (hand painted) based on a similar run of the same program.*

Buds can also specifically create new buds of other bud types in nearby cells. The new bud then starts following its own program to control the region around it. For example, to produce Figure 6 a single "stem" bud grows upwards forever to make the main branch. At intervals, it produces a "pod" bud that grows out from the stem into a pod. The pod bud is programmed to make larger cells of a different color, and after a finite number of cell divisions it dies. In Figure 7, the program is very similar, except the program for the stem bud splits into two after a certain period of growth and the pod bud specifies a spiky texture.

In physical organisms, a growing region may collide with another so each is constrained in shape by the other. Here, a bud will "freeze," i.e., stop executing its program, if the region it controls is about to collide into other parts of the form. This is checked after each cell division by standard software techniques, using a specifiable tolerance of closeness. As a result, a branch will not grow into or through others, and the surface remains a triangulated manifold.

Syntactic representation. Bud behavior has a temporal component. At any moment, there can be different buds following the same program, but at different steps in the process. For example, a pod bud which was created early on a stem may have completed the pod program, while another pod bud higher up the stem is half complete, and another near the newest part of the stem is just beginning. This sort of behavior is naturally accommodated by having a syntactic representation for the bud behavior and what is effectively a time sharing operating system which keeps track of the "program counter" for each bud.

A typical program looks like Figure 8, which is read and interpreted by the software (described below) to produce the form in Figure 7. This program defines three bud types, called `Base`, `Stem`, and `Pod`. Each is processed in a procedural manner; the lines are executed in order. By convention, the form always starts as a tetrahedron of four cells with one bud of the first type given in the program. In this case, there will be a bud of type `Base` which creates the blob at the bottom of the form. The `yellow` and `flat` commands set parameters for color and texture, then the `blob 300` command results in general growth for 300 steps. In the third line, a bud of type `Stem` is created and inserted into an adjacent cell. As that is the end of the

Base program, the Base bud then dies. Meanwhile, the Stem bud sets its parameters to head upwards and make green cells with a bumpy texture. The fat 0 command sets its "growth hormone diffusion radius" to be small. It grows a thin tube for 40 cells, leaving a pod bud in one of them three quarters of the way along. It reaches the last line of its program and dies just after it creates two new Stem buds, which will follow the same Stem program from the beginning. Finally, the Pod buds at first grow like a stem for ten cells, and then become red with a spiky texture. The size 1.3 command overrides the default cell size of 1, making longer edges in the spring embedding algorithm, and the fat 4 command leads to a blobbier shape. Figure 7 shows a seven-inch plastic model of the result, which was produced by fused deposition modeling, then hand painted.

```
Base:                 Stem:               Pod:
  yellow, flat          head up             grow 10
  blob 300              green, bumpy        red, spiky
  create Stem           fat 0               size 1.3
                        grow 30             grow 20
                        create Pod          size 2.0
                        grow 10             fat 4
                        create Stem 2       grow 20
```

Figure 8: *Program to create branching stem with spiky red pods seen in Figure 7. These programs are usually written linearly, but this is broken into three columns for exposition.*

When writing a time-sharing computer operating system, there are many details of synchronization and fairness to be worked out, e.g., what portion of the machine's CPU cycles should be assigned to each user, job, and thread. There are analogous issues here, to specify the relative growth rates of each bud's area. I have temporarily addressed this by giving each bud an equal share of the growth but allowing a sleep command (not shown) which I can use to slow down a bud's program in places. Future work should address this more systematically, as the balance of growth rates strongly affect the form's appearance.

Software. The images here and the videos on the web page [4] show the output of software I have written which implements all the above ideas. It is currently about 2000 lines of Java code and generates any of these forms in real time, in under a minute. A parser interprets the bud code and a time sharing scheduler puts each bud in control briefly in an interleaved manner. Meanwhile a separate graphics thread continually displays the current state of the form and allows it to be rotated on the screen under mouse control. It is very much experimental software, with various additional algorithmic ideas implemented or partially implemented and still being explored, which may or may not survive my testing. In addition, there are user interface functions for reading and saving the bud programs, executing programs one step at a time, outputting STL files for 3D printing, and various options for coloring, smoothing, and filtering the final output. I am exploring many experimental features, so the program is not in a stable enough state to make it publicly available any time soon. For example, Figure 9 shows the result of an experimental process that grows holes. Two cells near a bud are removed and the resulting boundaries are stitched together with a tunnel through the interior, which takes the form of an antiprism joining the temporary boundaries. This operation produces holes in many random directions and increases the genus by one each time it occurs. This does not correspond to any biological process I know of, but makes for interesting sculptural surfaces.

Conclusions and Future work. The forms presented here are inspired by nature, but the mechanisms do not necessarily simulate nature. I'm not a biologist, just an algorithm designer attempting to find simple ways to derive a wide range of interesting 3D organic forms. These forms may appear very different from the geometric style of my main body of sculptural work, but my intention is to use them as components and/or ornaments in larger structures that express an overall geometric aesthetic. That larger plan will hopefully be presented in future work, but must remain vague for the moment as I focus on this subproblem.

Rapid prototyping models of forms shown here are easily made. I am working on additional types of simple bud behavior which I expect will result in other visually interesting patterns of growth and form. These will be coded and tested with the intention to produce a large variety of forms from a small inventory of mechanisms. In addition, because the bud programs are syntactic, one might use genetic or evolutionary algorithm techniques to evolve them according to some fitness criterion, e.g., to maximize subjective beauty.

Finally, I can sometimes grow an icosahedral form, but it happens rarely. My 12-armed blobs usually have a less regular set of directions, so the icosahedral ones appear to be mostly due to luck. In the context of these growth mechanisms, it seems there is nothing particularly stable about the icosahedral form, as opposed to other 12-vertex forms such as the cuboctahedron.

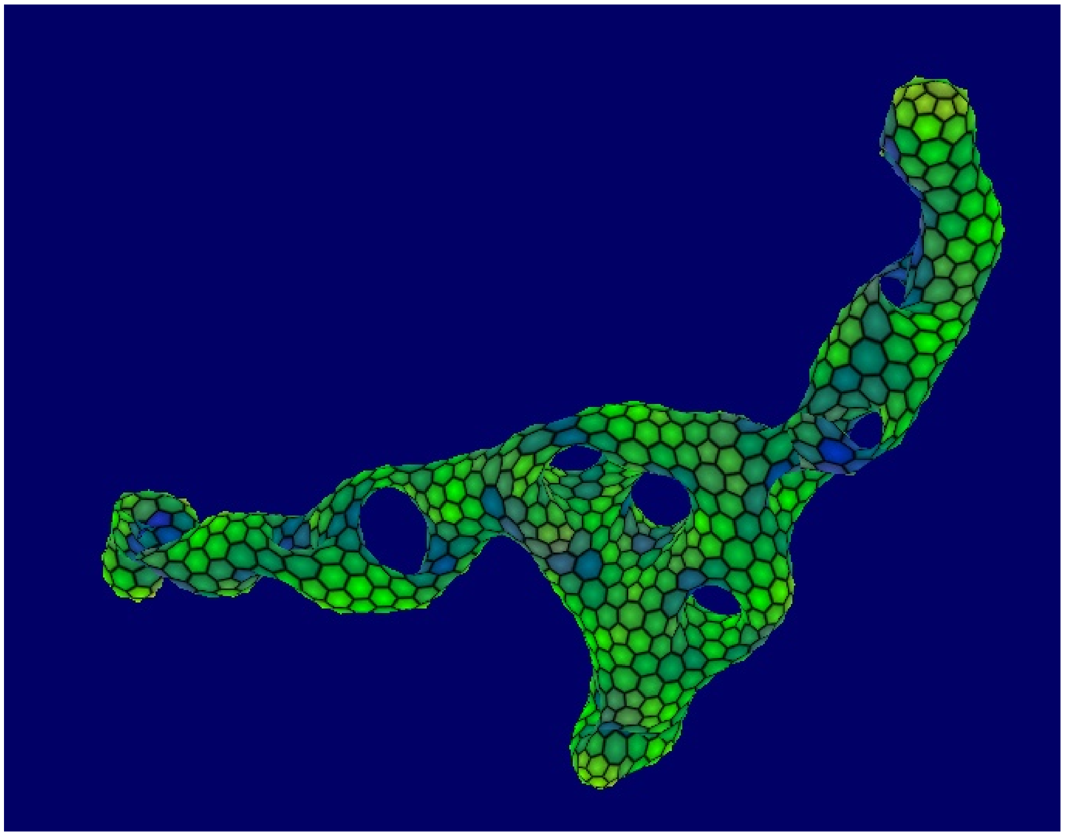

Figure 9: *High-genus form results from growing holes.*

References

[1] Kurt Fleischer et al., “Cellular Texture Generation,” SIGGRAPH 1995.

[2] Ernst Haeckel, *Artforms in Nature,* Verlag des Bibliographischen Instituts, 1904.

[3] G. Hart, "An Algorithm for Constructing 3D Struts," *Journal of Computer Science and Technology,* 24:1, 2009, pp. 56-64.

[4] G. Hart, videos at **`http://www.georgehart.com/Growth/growth.html`**

[5] Tomihisa Kamada and Satoru Kawai, “An Algorithm for Drawing General Undirected Graphs,” *Information Processing Letters*, 31:7-15, 1989.

[6] Przemyslaw Prusinkiewicz and Aristid Lindenmayer, *The Algorithmic Beauty of Plants,* Springer, 1991.

[7] D’Arcy Wentworth Thompson, *On Growth and Form,* Cambridge, 1952.