Pattern Formation in Lichen

by

Robert Walker Sumner

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2001

Author . . . .
Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by . . . .
Julie Dorsey
Associate Professor of Computer Science and Engineering and Architecture
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Pattern Formation in Lichen
by
Robert Walker Sumner

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Computer generated models are often conspicuously clean and lack the rich detail of the real world. Biological growth on a real object provides a compelling reminder that the object exists in a complex and dynamic environment. Of these biological agents, lichen flora form some of the most beautiful and intricate patterns, often completely covering an exposed rock or tombstone. In this thesis I present a new model of morphogenesis based on scientific findings in the biological literature related to lichen growth. My lichen growth simulation can be used to add growth patterns to synthetic objects. The simulation occurs in the context of a particular object so that the resulting pattern relates to the object's geometric structure.

My mathematical model of lichen growth is based on the Saffman-Taylor instability and the associated equations of Laplacian growth. I solve these equations using a diffusion-limited aggregation simulation, modified to account for the boundary conditions of the problem. I present techniques to visualize the lichen growth by generating images and animations from the simulation output.

Thesis Supervisor: Julie Dorsey
Title: Associate Professor of Computer Science and Engineering and Architecture

Acknowledgments

First and foremost, I would like to thank Professor Julie Dorsey for advising me during this project. Her mentoring, support, and friendship have been important to me during my first three years at MIT. And, I especially appreciate that she dealt with the department for me every time there was a problem with funding.

Professor Przemek Prusinkiewicz was instrumental in the success of this project. His guidance in the early stages of this work allowed me to select a novel area of research from a general idea. His support and encouragement was an inspiration throughout the project.

Thanks also to Allan Edelman for patiently answering my questions about mathematics. And to Professor Jessica Hodgins, my adviser at Georgia Tech. Without her having had faith in me I would not be here today.

I would like to thank Justin Legakis for providing endless coding advice, and for the use of JLLib, his excellent C++ library. And, especially, I'd like to thank him for all the special "features" that he added to JLLib when I needed them.

Thanks to Aaron Isaksen for giving me advice and encouragement about the text, for proofreading, and for making the animation.

The support of my family, during this project and before, has been invaluable. Thanks to my mother Mary, my father Evans, and my older brother Billy.

Finally, I could never have completed this work without the emotional support of my friends. Thank you to Aaron and Annie for always being there for me. To Andy for being so caring and so patient. To Rob for "fun" conversations and fun (no quotes) nights out. And to his mom for the crab apple jelly. To Justin for the inter-office phone technology that he helped me develop, and for all the times he hid peeps in his mouth. Thanks to him and Barb and Chris for providing targets for through-the-ceiling ballistic candy. Thanks to Aseem for starting the 249-fools. To Bryt for solving any problem (and I really mean any) and making life at the lab more fun. To Adel for hanging out with us even though he's from the Lower Graphics Lab. To Max for reminding me that lichen are an important, though overlooked, species. To Mok for calling me Bobo. To Pope for always suggesting that we go out to dinner. To the others in the CGG group, my San Francisco peeps, and everyone else who provided emotional support and friendship while I worked on this project. They made it possible to keep going when things got especially difficult.

CONTENTS

List of Figures

CHAPTER 1

INTRODUCTION

Researchers have long recognized the value of creating realistic images of synthetic scenes. One goal is to generate digital images that are indistinguishable from real photographs, and for some types of scenes, this goal has been achieved. Advanced rendering algorithms simulate sophisticated lighting effects such as color bleeding, caustics, and anisotropic scattering [28, 53]. Despite these techniques, the digital objects themselves often appear too pristine which gives them a distinctly unnatural feel. To address this problem, modelers rely on procedural shaders and texture maps to add the lacking visual detail [22]. However, procedural shaders can represent only certain types of detail, and texture maps often suffer from problems with surface parameterization. Furthermore, generating quality texture maps is a time consuming process that requires a talented artist.

Close examination of almost any object that exists in nature reveals a complex and dynamic microcosm of biological activity. Moss, liverworts, and lichen compete for space on the world's forest floors, tree trunks, rocks, and tombstones. In addition to adding visual beauty, biological growth gives us additional hints about the context in which an object is situated. If we see a rock covered with moss and lichen, we can conclude that it is located in a moist environment. If biological growth is ignored, we can make no such conclusions.

Of the lower plants, lichen make up one of the most diverse set of species, and their rich variety of color and form marvels any other plant group. Lichen survive in almost any habitat, ranging from the desert to frozen Antarctica. They flourish in places that are too harsh or limited for other plants, such as the shear face of a rock, desert sand, animal bones, rusty metal, and tree bark. Their robust biology makes them a ubiquitous part of nature, adding an intricate and subtle beauty to any natural scene.

In this thesis I present a new model of morphogenesis designed to generate the branching patterns of lichen. My lichen simulation can be used to add growth patterns to synthetic objects and is a partial solution to the overall problem that virtual environments are too pristine and lack the rich detail of the real world. The simulated lichen restore some of the missing details and make synthetic objects appear to be integrated parts of a natural environment. The simulation is carried out directly on the surface of a three-dimensional object so that the resulting lichen pattern is generated specifically for a particular object's shape.

My simulation focuses on the two-dimensional branching patterns of lichen. For example, Figure 1-1 shows a simulated lichen growing on a rock and represents a typical growth pattern generated by my method. The heart of the simulation model is based on the Saffman-Taylor instability, originally developed in the context of fluid displacement [46]. The mathematics of this instability lead

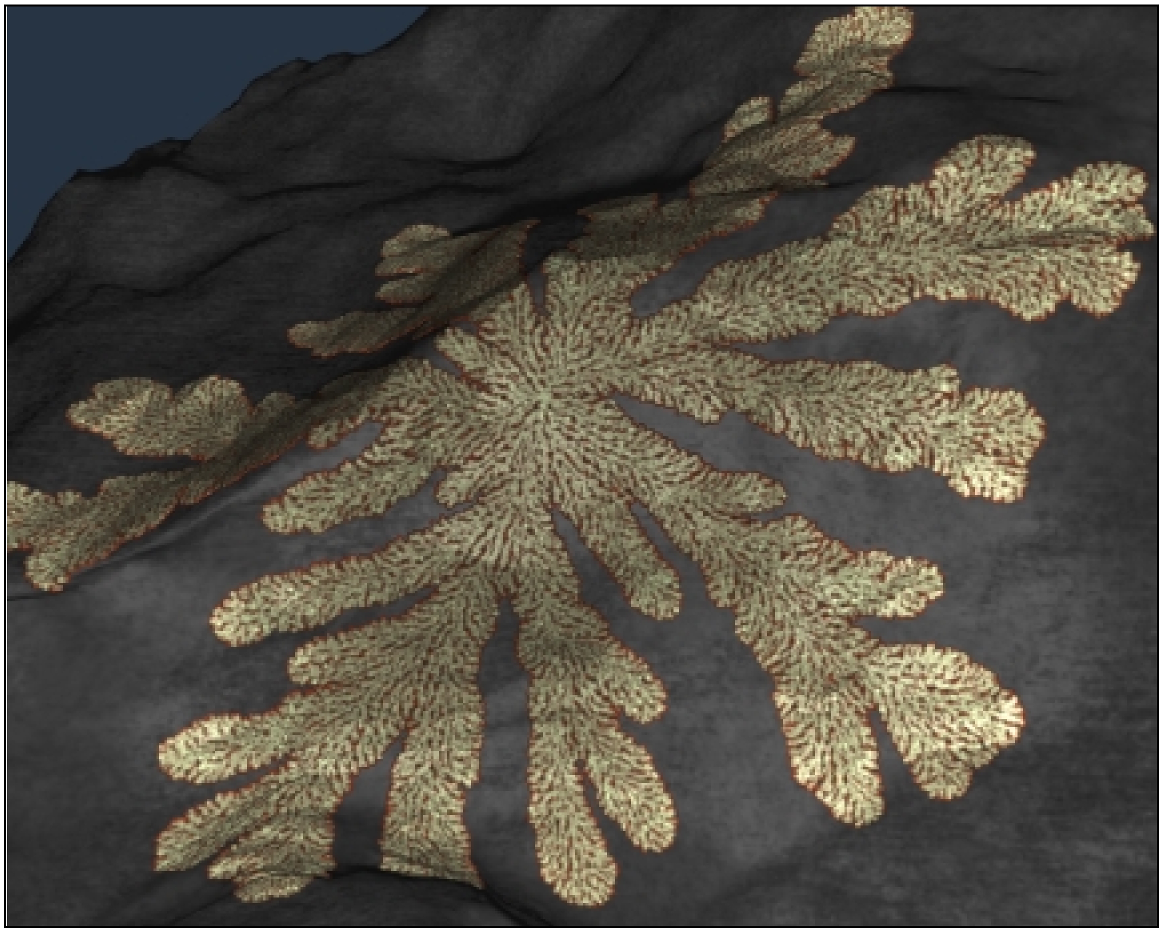


Figure 1-1: This simulated lichen growing on a rock demonstrates the type of patterns that my growth model generates.

to the equations of Laplacian growth. In the context of lichen, the overall idea is that an instability occurs whenever a bulge develops in the lichen's growth front. The instability causes the bulge to grow even more quickly, and it becomes a branch or lobe. Environmental factors compete with the instability to curb the branch's growth. The balance that results from this competition creates such striking patterns.

My research contributes both to biology and to computer graphics. From a scientific standpoint, the main contribution of my work is a mathematical model of lichen growth that incorporates branching. From the point of view of computer graphics, I present a new class of organic patterns and show how to "grow" them on a synthetic object in order to increase its visual detail. These patterns represent a form of texture synthesis in which the texture is generated specifically for the model to which it is applied. I also describe techniques for generating polygonal meshes from the simulation output for rendering purposes as well as animations of the lichen growth.

In the next chapter, I present relevant related work and summarize contributions to the synthesis of patterns and forms for computer imagery. In Chapter 3 I give a brief overview of lichen biology that summarizes their composition, internal structure, and morphology. Then, in Chapter 4, I describe my computational model and how it is related to experimental data on lichen growth. In Chapter 5 I discuss two solution methods that can be used to solve the Laplacian growth equations.

Next, in Chapter 6, I give a detailed description of my simulation, including the major steps and optimizations. In Chapter 7, I present specifics about how I implemented the lichen simulator including the class structure of the base simulation, the run-time graphics layer, and the distributed implementation. In Chapter 8, I present a tool that I developed to extract a polygonal mesh from the simulation output for rendering animation. Finally, in Chapter 9, I show my results and discuss ideas for future research.

CHAPTER 2

Related Work

My research is concerned with morphogenesis, or the development of patterns and forms in the domain of living organisms [42]. Specifically, I have developed a computational model of the morphogenesis of lichen that can be used to visualize their growth patterns. Many researchers have investigated models of morphogenesis that lead to computer imagery. These models can be grouped into two categories: *structure-oriented models* that focus on structural elements that are added to the organism and *space-oriented models* that describe what is located at each point in space. My work deals with the two-dimensional patterns that form as lichen gradually expand across a substrate, which falls into the category of space-oriented models. In the next section, I briefly describe structure-oriented models. Then I concentrate on space-oriented models including lichen growth models, reaction-diffusion, cellular automata, mobile cells, and cluster growth models.

2.1 Structure-oriented models

The string rewriting language of L-systems provides a framework for creating strikingly realistic geometric models of plants and trees [45]. Parametric L-systems incorporate continuous attributes and allow more sophisticated simulations of plant development. Differential L-systems [43] integrate differential equations with the string rewriting language in order to animate the developmental process in continuous time. Finally, [44] adds the environmental influence of pruning to the L-system framework with the goal of generating synthetic topiary, while [35] incorporates a complete bi-directional information exchange with the environment. The latter method can model situations such as trees competing with each other for space or sunlight, as well as other environmental factors. Earlier work on plant development [5, 19] also incorporates interaction with the environment but in a different context than L-systems. The "environment-sensitive automata" of [5] uses ray casting to test for intersections and proximity so that simulated plants avoid obstacles. In [19], plants are generated through a stochastic walk in voxel space that is constrained by geometric-based rules about the environment.

The idea of environmental interaction is important because it can be used to generate models that are closely integrated with their environment. When composing a synthetic scene, one would like the constituent objects to plausibly relate to one another and merge into one cohesive whole. My simulation incorporates geometric information about the environment. The growth process is carried out on the surface of the object so that the resulting lichen pattern is a plausible one for that particular object's shape.

2.2 Space-oriented models

2.2.1 Lichen growth models

Several researchers have proposed mathematical models directly related to lichen growth [41, 50, 1, 12, 24], but these models are quite simplistic from the point of view of morphogenesis. The overall goal of these models is to match measurable growth data as closely as possible. It is difficult to quantify the shape of branched lichen, but some species of lichen form nearly perfect circles. The radii of these circles provide a simple and easily measured quantity to study. In every model of lichen growth proposed so far, the lichen is assumed to form a perfect circle and an equation is developed for the radius of that circle over time. The accuracy of the model is determined by comparing the predicted radius with measured values.

My goals are very different. I am interested in investigating the branching structure of lichen and the class of patterns that they form. In order to capture these patterns I must abandon the simplifying assumption that a lichen is circular and develop a more sophisticated model that incorporates branching. Estimating the validity of my model is more difficult because I have no quantifiable measure of accuracy. Instead, I rely on qualitative means such as comparison with photographs.

2.2.2 Reaction-diffusion

Reaction-diffusion patterns are generated from two or more substances that diffuse through a medium and react with each other. Reaction-diffusion was originally proposed by Turing [51] as a plausible method of morphogenesis. In the field of computer graphics, Turk [52] used reaction-diffusion to generate animal spot and zebra stripe patterns on synthetic objects. His simulation was carried out directly on the surface of an object to avoid problems with texture mapping. Witkin and Kass [54] presented a similar method, but rather than simulating on the surface of an object they address the texture mapping problem by adapting reaction-diffusion to compensate for non-uniform surface parameterization. Meinhardt and Klinger [36] used reaction-diffusion to generate sea shell pigmentation patterns, and Fowler, Meinhardt and Prusinkiewicz [17, 37] used this method to generate realistic images of shells. Like Turk's simulation, theirs was carried out directly on the surface of the object. My work is similar because I also execute my simulation on the object's surface.

2.2.3 Cellular-automata

Cellular-automata can be thought of as a discrete version of reaction-diffusion, where space is represented by a uniform grid. The state of each grid cell is chosen from a finite set and is updated at each time step by a function based on a grid's neighbors. If this function is chosen to be properly based on diffusion, patterns similar to those of reaction-diffusion can be obtained. For example, Young [57] applied cellular-automata to generate animal coat patterns and Camazine [11] used cellular-automata to synthesize the pattern of a rabbit fish. My work is more closely related to reaction-diffusion than cellular-automata because my simulation is continuous and uses no grid.

2.2.4 Mobile cells

Fleischer and Barr [15] developed a simulation system to study patterns generated by discrete cells in a continuous environment. The cells can grow, move, divide, and die in their simulated "petri dish." They applied their model in the field of computer graphics [16] to generate organic patterns such as scales and thorns on synthetic objects. My work follows in the same spirit as that of Fleischer and Barr. I present a new model of morphogenesis and apply this model in the field of computer graphics.

2.2.5 Cluster growth models

In cluster growth models, a cluster gradually expands into its surrounding medium. The cluster is given some initial shape, and expansion occurs based on an aggregation algorithm. Simple algorithms often generate complex structures that resemble certain types of morphologies.

Witten and Sander [56] proposed a cluster growth model called *diffusion-limited aggregation* (DLA) that simulates diffusion using random movements of particles. Particles randomly move through a two-dimensional grid until they collide with and stick to a growing aggregate. Surprisingly, this simple process generates complex branching structures with fractal dimension. Witten and Sander have shown in [55, 56] that the probability distribution of a DLA random walker follows Laplace's equation, which explains the complex patterns that these simulations produce.

Kaandorp [29] used an accretive growth model to simulate three-dimensional formation of corals and sponges. In this iterative model, layers of materials are added to a growing tip. The thickness of the layer can be parameterized such that more growth occurs at the tip than along the sides. If this process is tuned properly, it can result in branching patterns that resemble corals and sponges.

The DLA method has been applied specifically to the Saffman-Taylor instability and the equations of Laplacian growth. Liang [32] solved these equations using two types of random walkers. The first type originates far from the cluster and the second type originates at one of the boundary sites, chosen at random proportional to the local curvature. Meakin, Family, and Vicsek [34] used an off-lattice version of DLA together with a sticking probability based on the local curvature of the cluster to solve the same equations. My work is related to that of Meakin *et al.* because I also use an off-lattice version of DLA with a sticking probability based on the local curvature. However, my random walk simulation occurs on the surface of an arbitrary triangle mesh, rather than in the two-dimensional plane. Furthermore, I present several techniques to optimize the performance of the DLA simulation.

CHAPTER 3

LICHEN BIOLOGY

A lichen consists of a fungus and an alga living together in a symbiotic relationship. The fungus is the visible part of the lichen, while the algae form a thin green layer just under the surface [13]. The fungus provides a physical structure that captures minerals for the algae and protects it from dessication. The algae, in turn, generate food through photosynthesis for the fungus. This relationship allows the lichen to survive and grow in habitats that neither symbiotic partner could exist in alone [33].

3.1 Internal structure

The central vegetive body of a lichen is called the *thallus*, which is surrounded by branches or *lobes* in some lichen [33]. The thallus of most lichen is stratified into layers. The outermost layer, called the *cortex*, is composed of tightly packed fungal filaments that protect the alga cells from intense sunlight and other organisms. Next is the algal cells in the *symbiont layer*. These are followed by a layer of fungal filaments called the *medulla*. Many lichen contain another layer of algal cells, but in some the medulla is attached directly to the underlying substrate [48]. Figure 3-1 shows this internal structure.

3.2 Morphology

Lichen are divided into three morphological groups: *crustose*, *foliose*, and *fruticose*. Figure 3-2 shows examples of these three morphologies. Crustose lichen are attached directly to the underlying substrate in such a way that they are practically inseparable. The margin of crustose lichens is sometimes surrounded by a dark layer of fungal cells, called *hyphae*, which grow faster than the main thallus body. Foliose lichen contain leaf-like branching lobes and are only partially attached to the substrate. The lobes are often arranged in a radial pattern around the thallus, or the lichen's center. Fruticose lichen are shrub-like and stand out from the surface of the substrate. Since fruticose lichen are structurally similar to plants, their form is a good candidate for a structure-oriented model such as L-systems. Our simulation is more appropriate for generating two-dimensional patterns similar to those formed by crustose and foliose lichen.

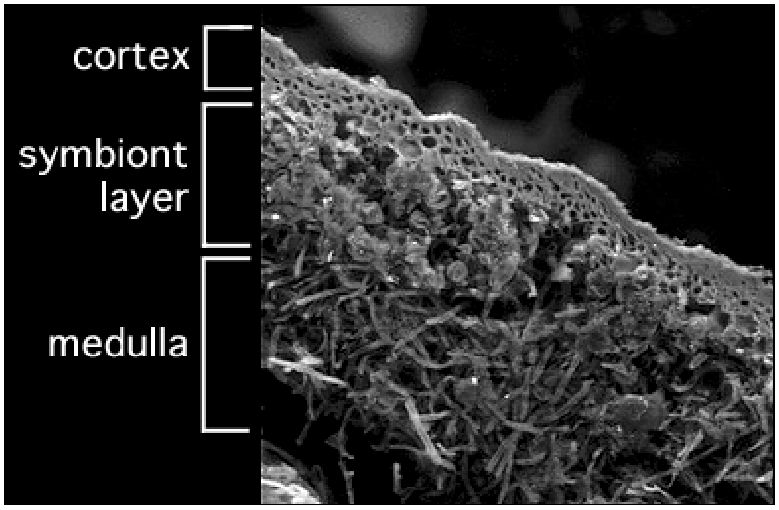


Figure 3-1: Micrograph of the internal structure of the lichen thallus. Photograph courtesy of Ben Waggoner.

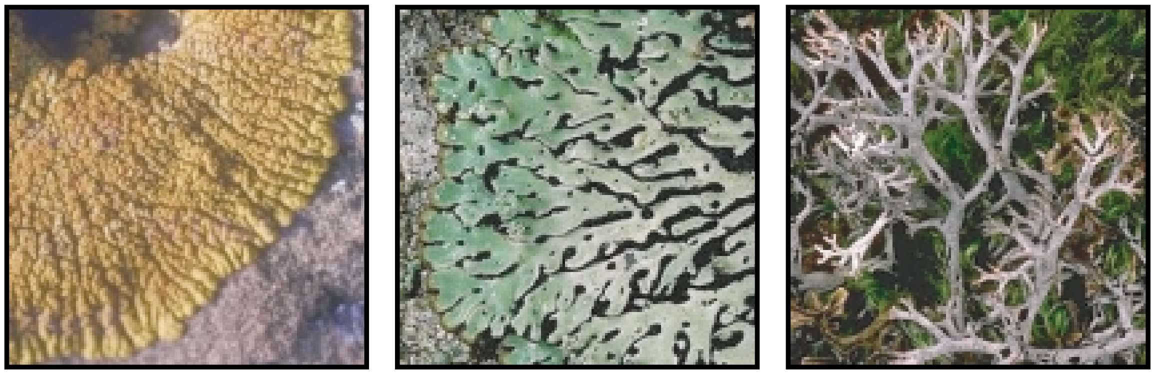


Figure 3-2: The three main morphological categories of lichen, shown above from left to right, are crustose, foliose, and fruticose.

3.3 Lobe growth and division

Numerous species of crustose and foliose lichen have a margin composed of radially oriented lobes. Many researches have studied the properties of these lobes in order to explain how they grow and divide and how the lichen maintains a circular symmetry. However, lichen are notoriously difficult to study because of their extremely slow growth rate. As a result, the level of understanding of lobe growth is rather coarse. No specific biological processes have been identified that concretely explain lobe formation. The current state of the art in understanding consists of qualitative theories based on experimentation. In this section I summarize experiments and theories from the biological literature that are relevant to my growth simulation.

Experiments by Hooker [26] showed that the growth of different lobes within a single lichen varies widely, so that some lobes may grow rapidly while others grow more slowly. However, lichen species exhibiting this differing lobe growth rate nevertheless maintain a roughly circular shape.

This symmetry suggests some sort of communication between different lobes, perhaps through the diffusion of carbohydrates from the thallus center or from one lobe to another. Armstrong has shown, however, that it is unlikely that any such communication occurs [2]. His studies suggest that carbon for radial growth is made within a narrow perimeter around the thallus margin. Later studies [3] support this conclusion, verifying that radial growth depends on carbohydrates produced within the lobe itself. Hooker suggests that circular symmetry is maintained because the radial expansion of lichens occurs through a process of lobe division and engulfment. Slower growing lobes are overtaken and engulfed by faster ones, so that the slow lobes are absorbed into the thallus body and lose their identity.

More recent work by Armstrong [4] supports Hooker's theory of lobe division and engulfment. He studied the lobe growth of *P. conspersa* in cases where lobes either protrude past or lag behind the lichen's margin of growth. In cases where a lobe lags behind the margin, lateral growth of neighboring lobes engulfs and eliminates the retarded lobe. When a lobe protrudes one or two millimeters past the margin, it initially grows very rapidly as a result of reduced competition. Then, the lobe stops growing, possibly because it dries out more rapidly in the unfavorable microclimate beyond the thallus margin. Armstrong concludes that circular symmetry is maintained because lobes that protrude past the margin eventually slow down because of the unfavorable microclimate, and lobes that lag behind the margin are engulfed by adjacent lobes.

Hale studied the growth patterns of individual lobes in the lichen *P. caperata* [20]. He found that the active area of growth is in the lobe tip, with the outermost 1 mm of the lobe contributing 80% to lobe growth. Lobe division begins with the development of several bulges or protrusions at the lobe tip. Lateral bulges die, leaving two or three radial bulges to continue to maturity. These bulges exhibit an initial period of lateral growth where they broaden rapidly, followed by a period of more rapid radial growth. As a lobe develops, its lateral growth roughly equals one-half its radial growth. This relationship holds until the lobe reaches a fixed characteristic width at which time lateral growth ceases.

Armstrong [3] has found that the radial growth of circular thalli increases in smaller thalli, but approaches a constant growth rate in larger thalli. If radial growth is dependent on photosynthesis in a narrow region of the lobe tip, then the change from an increasing to constant growth rate may be related to the formation of marginal lobes. In small thalli, there is little competition at the thallus margin, which allows a great deal of lateral growth. The lateral growth, in turn, results in more radial growth and more lobe divisions. However, the margin quickly becomes dense with lobes as competition increases, and a constant growth rate is reached. Hill [25] has suggested that the ratio of lobe width to length may be a useful taxonomic character. Lobes approach a characteristic length before division that is constant for each species. Thus, ratio of width to length is proportional to the distance between lobe divisions and should be unique for different species of lichen.

CHAPTER 4

COMPUTATIONAL MODEL

When developing any simulation of a real world phenomenon, one major challenge is determining which aspects of the phenomenon to encode in the simulation and which to ignore. In essence, one must distill out the essential characteristics of the phenomenon in question and develop a simulation that addresses these characteristics. It is often not desirable to specifically encode the outward features of the phenomenon. Rather, one strives to understand the internal forces involved and create a computational model based on these forces. Then, the outward features form by a process called *emergence*, because they emerge from the basic rules of the simulation instead of being specifically prescribed.

In this research, my goal was to develop a computer simulation in which the growth patterns of lichen naturally emerge from the rules of the simulation. As a first step toward this goal, I compiled a visual catalog of lichen photographs that captures many salient characteristics of their shape and form (see Appendix A). From these images, I noticed that many lichen exhibit interesting branching patterns that are not found in other flora. Next, I conducted a thorough investigation of the biological literature related to lichen growth and branching, which I summarized in Chapter 3. I selected three main observations based on my investigation. From these observations, I developed a hypothesis to explain the branching patterns of lichen. This hypothesis led me to the science of instability analysis and Laplacian growth, and I developed a computational model based on equations in this domain.

4.1 Key Observations

While investigating lichen biology, I concentrated on research related to growth and branching. I noted three key observations based on lichen experimentation:

1. Radial expansion of lichen occurs through a process of lobe division and engulfment. Lobe division begins with the development of several bulges or protrusions at the lobe tip. Many of these bulges fade away, leaving two or three to mature into new lobes.

2. When a lobe protrudes past the growth margin, it initially grows more rapidly because of reduced competition for space with the other marginal lobes. After this initial period of rapid growth, the lobe's growth is retarded because it becomes desiccated in the unfavorable microclimate beyond the thallus margin.

3. The active area of growth is in a narrow perimeter formed by the outermost one millimeter of the lobe.

4.2 Hypothesis

I used the observations from Section 4.1 to develop a hypothesis of the underlying processes responsible for lichen lobe growth and division. Items 1 and 2 lead to the overall scientific foundation of this hypothesis. As the lobes that make up a lichen's thallus margin grow, they develop many random bulges and perturbations. When a bulge develops that protrudes past the thallus margin, it is suddenly free from competition for space with the other lobes. Because of the reduced competition, the lobe grows more rapidly and protrudes farther outside of the thallus margin. It experiences even less competition which, in turn, causes even more rapid growth. This positive feedback is eventually checked by another process: As the lobe advances past its neighbors, it experiences an increasingly dry microclimate. The farther from the lichen margin, the more harsh the microclimate, and the more retarded lobe growth becomes. The balance between the positive feedback from the reduced lobe competition and the negative feedback from the poor microclimate is such that most small perturbations disappear, while a few continue to grow into mature lobes. I hypothesize that the branching patterns of lichen form as a result of the competition between the instability and a restoring force that exist at the lichen's advancing margin.

4.3 Instabilities

After developing this hypothesis, I searched other scientific literature for information on unstable processes that might result in branching. I found that self-amplifying instabilities similar to the one I described in Section 4.2 occur in other contexts. Two of these instabilities have been widely studied, one in the context of fluid flow and the other in the context of solidification. Both lead to a growth phenomena called *Laplacian growth*.

4.3.1 Saffman-Taylor instability

Saffman and Taylor studied a phenomena called *viscous fingering* in the context of the Hele-Shaw cell [6]. A Hele-Shaw cell consists of two clear rigid plates separated by a small fixed gap. The space between the two plates is filled with one fluid. A different fluid can be injected into the gap through a small hole in the center of the top plate [23].

In their seminal article [46], Saffman and Taylor describe an instability that is generated when one fluid displaces another in a Hele-Shaw cell. This instability occurs whenever a bulge develops in the advancing fluid front. If we assume that the fluid inside the cell is water and the injected fluid is air, then the rate at which the air/water interface advances is proportional to the gradient of the pressure in the water at the interface. Since the value of the pressure in the water is constant at any location far from the interface, the gradient is highest at the tip of a bulge that extends past the bulk of the air bubble. When a bulge develops, it will elongate because of the increased pressure gradient at its tip. The elongated bulge will then have an even steeper gradient, which will cause it to grow even more. This self-amplifying process is referred to as the Saffman-Taylor instability.

In fluid displacement of this type, surface tension plays a crucial role. When a less viscous fluid displaces a more viscous one (as in the example above), the surface tension at the interface between the two fluids acts as a restoring force that smoothes out high-frequency perturbations. In essence, surface tension adds an energy cost to displacements in the interface that prevents unbounded perturbations that would otherwise result if the Saffman-Taylor instability were working alone. As the bubble expands, random fluctuations generate many tiny bulges. The Saffman-Taylor instability causes them to grow exponentially while at the same time the surface tension tends to smooth them out. If the surface tension is an appropriate value, a balance will be reached between these two competing forces, and complex, branching patterns emerge. The formation of these patterns is referred to as viscous fingering.

4.3.2 Mullins-Sekerka instability

Mullins and Sekerka described an analogous instability in the context of solidification [38], such as during ice crystal formation. The rate at which a liquid freezes is proportional to the rate at which heat can be conducted away from the freezing front. This, in turn, depends on the temperature gradient in the surrounding area. Bulges that develop in the solidifying front create a self-amplifying instability called the Mullins-Sekerka instability that is mathematically equivalent to the Saffman-Taylor instability [6].

4.3.3 Laplacian Growth

Pattern formation in viscous fingering is controlled by the pressure field in the fluid medium that surrounds the advancing interface. In solidification, the pattern is controlled by the temperature field around the solidifying front. Both growth processes are referred to as Laplacian growth because the field parameter (pressure or temperature) satisfies Laplace's equation. In this discussion, I describe the equations of Laplacian growth in terms of viscous fingering in a Hele-Shaw cell. An analogous treatment could be made for solidification.

Consider a Hele-Shaw cell in which air is injected into water at a constant rate. Since air is much less viscous than water, the pressure in the air is assumed to be constant. The motion of the two fluids is governed by Darcy's law [46], which asserts that the mean horizontal velocity $\mathbf{u}$ of either fluid averaged across the gap in the cell is

$$\mathbf{u} = -M\nabla p \qquad \text{where} \qquad M = \frac{b^2}{12\mu}. \tag{4.1}$$

Here p is the pressure, b the gap between the plates, and μ the viscosity. Assuming the fluids are incompressible, the equation of continuity gives us Laplace's equation for the pressure field:

$$\nabla \cdot \mathbf{u} = -M\nabla^2 p = 0. \tag{4.2}$$

Since the pressure in the air is constant, the pressure drop across the air/water interface Γ is

$$p_\Gamma = \tau\kappa, \tag{4.3}$$

where τ is the surface tension and κ the local curvature of the interface. Finally, the pressure far from the moving interface is constant:

$$p_\infty = p_0. \tag{4.4}$$

4.4 Lichen Growth Equations

In my hypothesis, pattern formation in lichen is controlled by destabilizing and restoring forces analogous to those in other processes governed by Laplacian growth. Item 3 from Section 4.1 makes this analogy even more plausible. Item 3 states that lobe growth in lichen occurs within a narrow region around the lobe's perimeter. Thus, the portion of the lichen that is actively involved in growth consists of the boundary or interface between the lichen and its external environment. This interface is analogous to the air/water interface of the Saffman-Taylor instability or the solid/liquid interface of the Mullins-Sekerka instability. To complete the analogy, I select the Laplacian growth equations (Equations 4.2, 4.3, and 4.4) as the governing equations of the lichen simulation. My simulation uses these equations to generate the patterns that lichen form as they grow.

It should be noted that the analogy between Laplacian growth and lichen growth is only a hypothesis. As I mentioned in Section 3.3, the scientific understanding of lichen growth is still incomplete. The exact biological processes that govern lobe growth and division are unknown. The connection to Laplacian growth is one that I believe to be plausible and is an advancement over previous mathematical models. As scientists gain a more sophisticated understanding of lichen growth, the analogy may be made more concrete. For example, the Laplacian growth equations from the previous section were developed in the context of fluid flow. In lichen growth, the pressure field must have a different meaning most likely related to the favorableness of growth at a given point. Because the exact growth processes are unknown, I proceed under the assumption that lichen growth is governed by the mathematics of Laplacian growth without specifying the specific biological processes that justify this assumption.

CHAPTER 5

SOLUTION METHODS

Laplace's equation appears in many different contexts throughout science and has received a great deal of attention. One simple solution method is to use standard techniques to solve Equation 4.2 in the entire two-dimensional domain and then compute the gradient with a numerical differencing technique. However, this process is time consuming because we must compute the value of the pressure field everywhere, even though we only care about its gradient at the points that make up the interface.

Two alternate methods that avoid a solution in the full domain are common in the literature: the boundary integral method and diffusion-limited aggregation. While researching this topic, I explored both methods in detail. For my implementation, I chose DLA because it is numerically stable and straightforward to implement. However, both methods are powerful, and I include a discussion of each in this chapter.

5.1 Boundary integral method

The boundary integral method (also called the boundary element method or the panels method) solves Laplace's equation by applying Green's function to replace the required solution in the full two-dimensional domain with an integral equation around a closed boundary. In general, the method can be used to reduce an n-dimensional problem to an $(n{-}1)$-dimensional surface integral equation. Solving the integral equation requires solving a large system of algebraic equations. [49] and [21] both provide excellent treatments of the boundary integral method, though the latter contains several typographic errors in the mathematics.

In application to viscous fingering, Brower, Kessler, Koplik, and Levine [9, 10, 31] and Sander, Ramanlal, and Ben-Jacob [47] present applications of the boundary integral method to solving moving interface problems that incorporate stabilizing and destabilizing forces. They use the standard boundary integral method and then present a technique for advancing the interface forward in time.

To use the boundary integral method, Green's function is applied to reformulate Laplace's equation as an integral equation around the interface:

$$\int_\Gamma p(\mathbf{x})\frac{\partial G(\mathbf{x}',\mathbf{x})}{\partial \hat{n}_\mathbf{x}}d\mathbf{x} = \int_\Gamma G(\mathbf{x}',\mathbf{x})\boxed{\frac{\partial p(\mathbf{x})}{\partial \hat{n}_\mathbf{x}}}d\mathbf{x}. \tag{5.1}$$

Here integration is over the air/water boundary Γ, and the notation $\hat{n}_\mathbf{x}$ refers to the unit normal to the interface at the point $\mathbf{x}$. The boxed formula is the unknown, and

$$\begin{aligned} G(\mathbf{x},\mathbf{y}) &= \ln|\mathbf{x}-\mathbf{y}| && (5.2) \\ \frac{\partial G(\mathbf{x},\mathbf{y})}{\partial \hat{n}_{\mathbf{y}}} &= \frac{(y_1-x_1)\hat{\imath}+(y_2-x_2)\hat{\jmath}}{(y_1-x_1)^2+(y_2-x_2)^2}\cdot\hat{n}_{\mathbf{y}}. && (5.3) \end{aligned}$$

We now discretize the interface into N panels labeled Γ_1 through Γ_N. Equation (5.1) is written as a matrix equation with N equations and N unknowns in the form of $\mathbf{Av}=\mathbf{b}$, where the lhs of (5.1) provides the values for the vector $\mathbf{b}$ and the rhs provides the values for the matrix $\mathbf{A}$. To compute entry $\mathbf{b}_j$ we let the primed point $\mathbf{x}'$ approach panel j and hold it constant as we integrate around the boundary, obtaining the sum:

$$\mathbf{b}_j = \sum_{k\neq j}\left[\int_{\Gamma_k} p(\mathbf{x}_k)\frac{\partial G(\mathbf{x}_j,\mathbf{x}_k)}{\partial \hat{n}_k}\,d\mathbf{x}_k\right], \qquad (5.4)$$

where Γ_k indicates panel k. The integral in each term of the sum is solved numerically using Gaussian quadrature. When evaluating Green's function, there is a singularity in the logarithm when $k=j$. In the limit as $\mathbf{x}$ approaches $\mathbf{x}'$, the vector $\nabla G(\mathbf{x}',\mathbf{x})$ becomes tangent to the boundary and $\nabla G(\mathbf{x}',\mathbf{x})\cdot\hat{n}$ equals zero. So, the case of $k=j$ is left out of the sum in Equation (5.4).

To compute the entries of the matrix $\mathbf{A}$, again we let the primed variable $\mathbf{x}'$ approach a particular panel j. This corresponds to an entire row of the matrix, where entry $\mathbf{A}_{jk}$, $j\neq k$, is given by:

$$\mathbf{A}_{jk} = \int_{\Gamma_k} G(\mathbf{x}_j,\mathbf{x}_k)\,d\mathbf{x}_k. \qquad (5.5)$$

This integral is solved numerically using Gaussian quadrature. The case of $j=k$ is a more complicated situation because of the singularity in the logarithm. Rather than solve this integral numerically, we instead integrate G across the length of the panel explicitly:

$$2\int_0^{l/2}\ln(r)dr = 2\left[\frac{l}{2}\ln\left(\frac{l}{2}\right)-\frac{l}{2}\right] = l\left[\ln\left(\frac{l}{2}\right)-1\right], \qquad (5.6)$$

where r is the distance from the singularity and l is the length of the panel. The result of (5.6) is used to compute the diagonal entries of $\mathbf{A}$:

$$\mathbf{A}_{jj} = l_j\left[\ln\left(\frac{l_j}{2}\right)-1\right]. \qquad (5.7)$$

Once $\mathbf{A}$ and $\mathbf{b}$ have been computed, the solution $\mathbf{v}$ is found with:

$$\mathbf{v} = \mathbf{A}^{-1}\mathbf{b}. \qquad (5.8)$$

We have solved for $\partial p/\partial\hat{n}$ at each panel in the discritized interface and, since the velocity $\mathbf{u}$ of a point on the panel is proportional to ∇p, we know the velocity of each panel in the normal direction. Additional techniques described in [9, 10, 31, 47] can be used to advance the interface forward in time.

5.2 Diffusion-limited aggregation

In order to compute the motion of the interface we need only solve Laplace's equation at points belonging to the interface with the boundary conditions given by Equations 4.3 and 4.4. It is well known [55, 56, 39] that Laplace's equation describes the probability $P(\mathbf{r}, t)$ that a random walker will be at the point $\mathbf{r}$ at the time t. The pressure field in the Hele-Shaw cell is analogous to the probability distribution of the random walker in a DLA simulation. It follows that the probability distribution of a DLA random walker satisfies Equation 4.2 so that the pressure at the interface is equal to the probability of a random walker colliding with the DLA cluster. If the random walkers originate far away from the cluster, the resulting probability distribution satisfies Equation 4.4. However, Equation 4.3 is not satisfied.

To satisfy this boundary condition, note that the pressure at a point $\mathbf{r}$ on the interface Γ can be expressed as an integral over the interface:

$$p(\mathbf{r}) = \int_{\Gamma} G(\mathbf{r}, \mathbf{s}) p_{\Gamma}(\mathbf{s}) d\mathbf{s}, \tag{5.9}$$

where $G(\mathbf{r}, \mathbf{s}) = \ln |\mathbf{r} - \mathbf{s}|$ is the two-dimensional Green's function [40] and can be thought of as the value of an electromagnetic field measured at the point $\mathbf{r}$ and generated by a point source located at $\mathbf{s}$ on a grounded conductor. It is shown in [30] that Green's function is proportional to the number of times the point $\mathbf{r}$ on the interface is visited by random walkers that originate at the point $\mathbf{s}$ on the interface and terminate when the arrive at point $\mathbf{r}$. I define a sticking probability P_s such that a particle that intersects the cluster in our DLA simulation is incorporated into the cluster with probability P_s and continues its walk otherwise. A particle that does not stick can be thought of as starting a new walk originating from the given point on the interface and eventually terminating elsewhere on the interface. Following from the discussion of Green's function, Equation 4.3 is satisfied if P_s is proportional to the local curvature. I compute P_s as

$$P_s(\kappa) = \kappa + B, \tag{5.10}$$

where B is a constant and κ is the local curvature. Section 6.2 describes how I estimate κ. The value of P_s calculated in Equation 5.10 is clamped to 1.0 if it is larger than 1.0, and to some minimum value if it is less than zero. When the DLA simulation is performed in this manner, the boundary of the cluster satisfies Laplacian growth.

Regardless of the formalisms presented in this and the previous chapter, it is important to have an intuitive understanding of the random walk simulation. A DLA simulation has an inherent instability because a portion of the cluster that happens to protrude farther than its neighbors is more likely to be hit by random walkers. Therefore, this portion tends to extend even farther, further increasing its likelihood of getting hit. However, the sticking probability provides a restoring force. When a walker intersects the cluster, it randomly sticks according to a probability based on the local curvature. This acts as a restoring force because protrusions have large, negative curvature and thus a low sticking probability. Concave regions, on the other hand, have large, positive curvature and therefore a high sticking probability. So, even though a random walker is more likely to intersect a protrusion, it is less likely to stick. Furthermore, though the occasions when the walker travels into a concave region of the cluster are less frequent, if the walker does happen to do this, it is almost guaranteed to stick because the sticking probability is high in these areas. In this way, while the random walk itself provides an instability that encourages protrusions to grow, the restoring force of the sticking probability tends to even out the surface. Depending on the relative strengths of the instability and restoring force, the resulting cluster forms a variety of complex and interesting branching patterns.

CHAPTER 6

SIMULATION

6.1 Simulation overview

I solve the equations of Laplacian growth using the diffusion-limited aggregation method described in Section 5.2. I simulate a random walker that moves along the surface of an object until it intersects a cluster of elements. The walker is incorporated into the cluster according to a sticking probability based on the cluster's local curvature. The random walk itself is off-lattice and therefore does not conform to any grid. This prevents bias in the overall shape of the cluster that might otherwise result. The substrate on which the lichen grows is represented as a triangle mesh, and the DLA simulation is carried out directly on this mesh. Each random walker is represented as a sphere with constant radius r. The center of the sphere associated with a walker is constrained to the surface of the triangle mesh so that the intersection of the sphere with the plane of the walker's triangle forms a circle with radius r. As the walkers aggregate, the cluster that they form spreads across the mesh and represents the growing lichen. Figure 6-1 shows a flowchart of the entire simulation. The four major steps are described in the following paragraphs.

Step 1: Initialize simulation To initialize the simulation, create a cluster consisting of a single element somewhere on the mesh. This cluster serves as the seed around which subsequent random walkers aggregate. Figure 6-2 shows a mesh after the initialization step.

Step 2: Create new random walker New walkers should be created at random positions on the mesh R units away from the centroid of the cluster measured along the surface of the mesh. In order for the DLA simulation to approximate the solution to the Laplacian growth equations, this distance must be large enough that the behavior of the walkers is nearly the same as if they originated infinitely far away. With an arbitrary triangle mesh it is difficult to randomly select a point R units away measured along the surface of the mesh. As an approximation, I developed the following algorithm to create a new random walker at an appropriate location.

Consider a sphere with radius R centered at the centroid of the cluster. Then, compute the intersection of the sphere with the edges of the triangles in the mesh. Connect the intersection points to form a series of line segments. Then, randomly select a point on one of the line segments such that there is a uniform distribution across the entire collection of segments considered as a whole. Finally, place the new random walker at this location. Figure 6-3 illustrates this process.

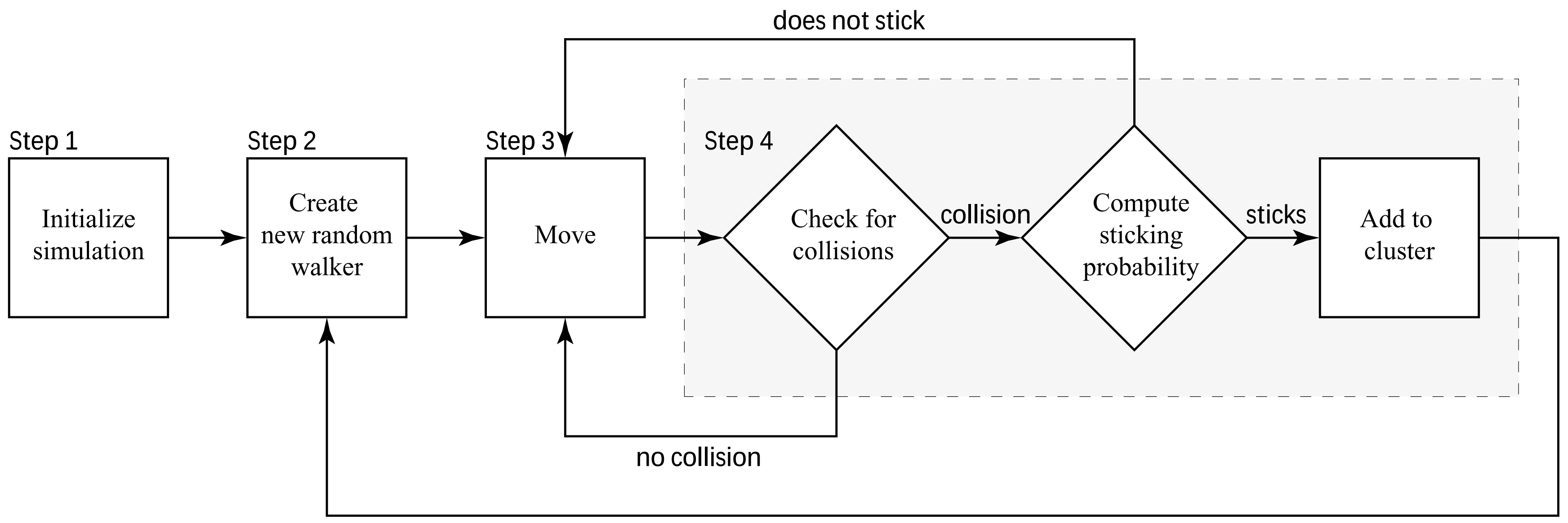


Figure 6-1: This flowchart shows the major steps of the simulation.

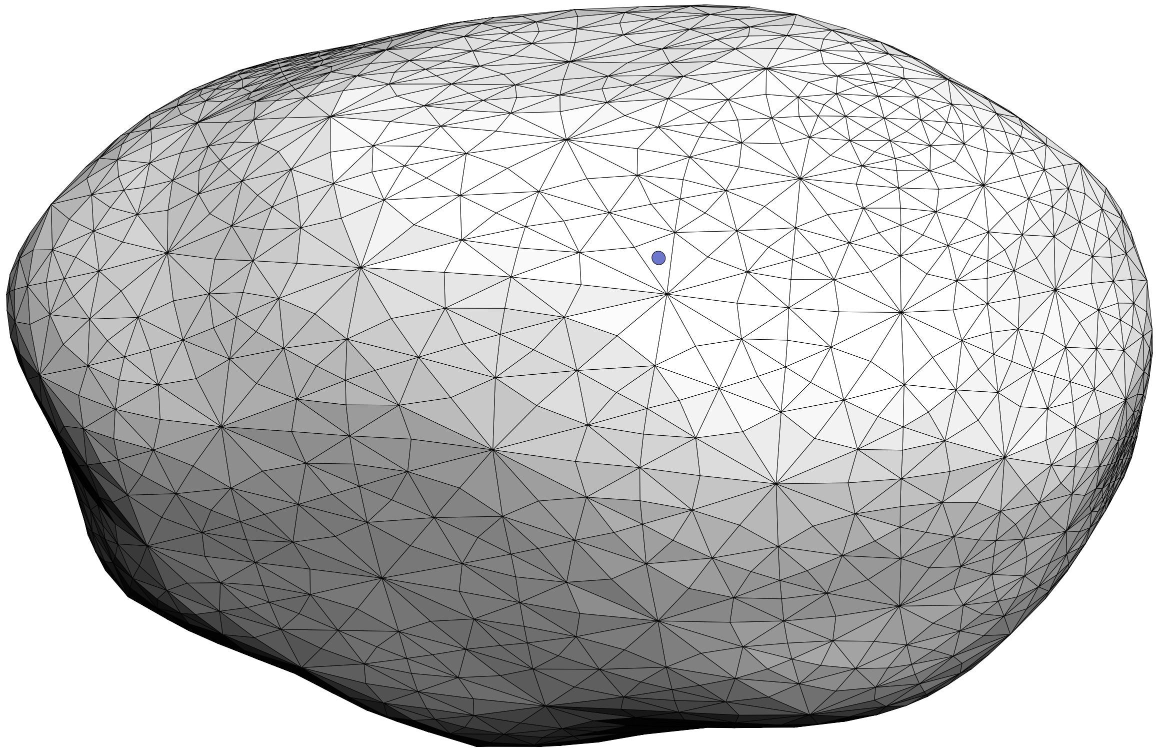


Figure 6-2: In the initialization step, a cluster containing a single element is added to the triangle mesh. The element is shown here as a blue circle drawn on the surface of the mesh. The mesh in this and subsequent examples contains 4,608 triangles with 2,306 vertices and forms the shape of a smooth rock. For illustrative purposes, the element radius used in these examples is about 20 times greater than in a typical simulation.

Using this algorithm, the position of the walker is approximately R units away from the centroid of the cluster measured through three-dimensional space. With certain meshes, the distance measured along the surface of the mesh may be much different. However, I did not notice artifacts from this approximation in any of my simulations. Since the size of the cluster increases as the simulation progresses, the value used for R must also increase. Define the effective radius R_e of the cluster as the distance from the centroid of the cluster to the farthest element in the cluster. For the examples in this thesis, I use R in the range of $1.5R_e$ to $3R_e$.

Step 3: Perform one random step After it has been placed on the mesh, the walker lies within a certain triangle. To perform one step of the random walk, select a random distance d less than or equal to the radius r of the walker, and a random direction θ in the plane of the current triangle. d and θ are chosen such that the resulting target destination $\mathbf{P}$ is equally distributed within the circle of radius r centered at the walker in the plane of the walker's current triangle. If $\mathbf{s}$ and $\mathbf{t}$ are orthogonal unit vectors in the plane of the current triangle and $\mathbf{W}$ is the walker's center point, then d, θ, and $\mathbf{P}$

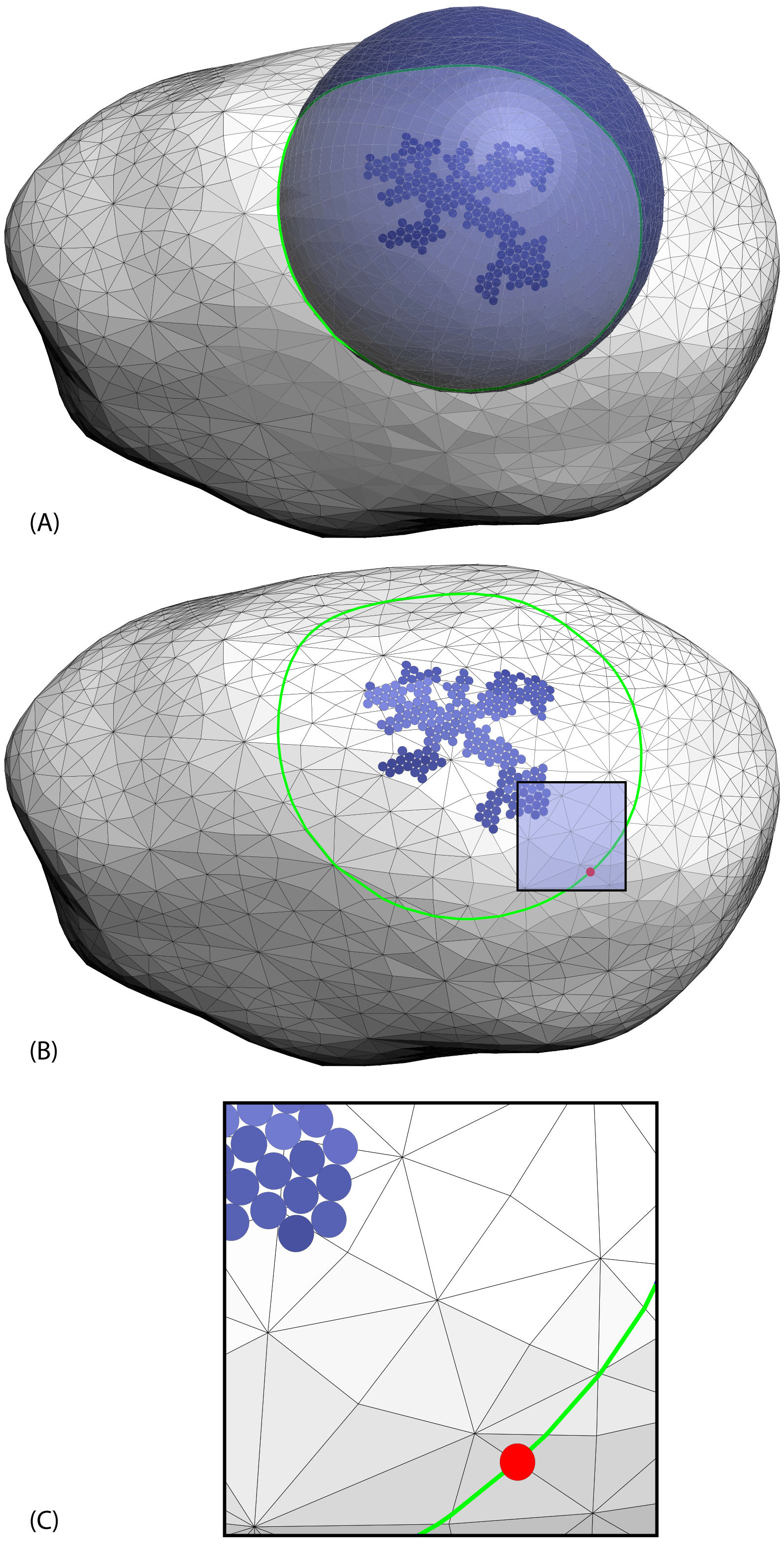


Figure 6-3: (A) To generate a new random walker, a sphere is centered at the cluster with radius R. (B) The intersection of the sphere with the triangles of the mesh generates a collection of line segments, shown in green. (C) Finally, a new walker, displayed in red, is placed at a random position chosen on one of the line segment.

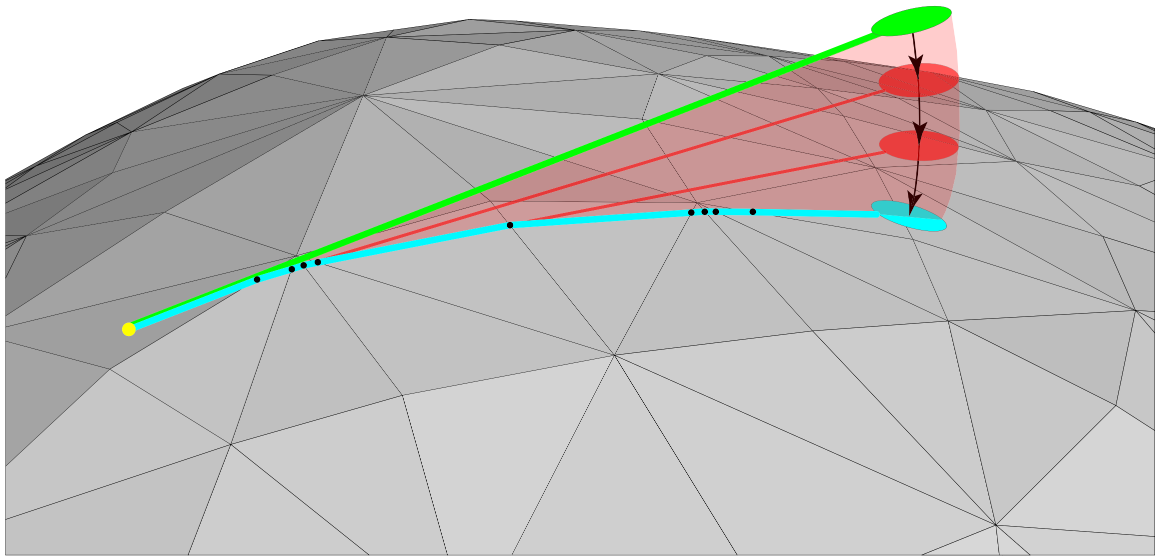


Figure 6-4: A walker, starting from the yellow point, takes a large random step as indicated by the green line. Since this step extends past the edge of the walker's original triangle, the green line is folded down several times, as indicated by the red lines. Its final path after these folds is shown in blue. The small black circles mark the intersection of the path with triangle edges, which is where the folds occur. Random steps of this size occur often because of the optimization described in Section 6.3.3.

are computed as:

$$\begin{aligned} d &= r\sqrt{rand_1} && (6.1) \\ \theta &= 2\pi \cdot rand_2 && (6.2) \\ \mathbf{P} &= (d\cos(\theta)\,\mathbf{s} + d\sin(\theta)\,\mathbf{t}) + \mathbf{W}, && (6.3) \end{aligned}$$

where $rand_1$ and $rand_2$ are random numbers between 0 and 1 chosen with uniform distribution. If the vector from $\mathbf{W}$ to $\mathbf{P}$ lies entirely within the current triangle, simply move the walker to $\mathbf{P}$ and continue on to Step 4. However, if the vector from $\mathbf{W}$ to $\mathbf{P}$ intersects one of the edges of the triangle, let $\mathbf{W}'$ be the intersection point. Rotate $\mathbf{P}$ from the plane of the current triangle into the plane of the adjacent triangle and let $\mathbf{P}'$ be the position after rotation. I use the method from Goldman [18] to compute the necessary rotation matrices. Now, if the vector from $\mathbf{W}'$ to $\mathbf{P}'$ lies entirely within the new triangle, move the walker to $\mathbf{P}'$ and continue on to Step 4. If the vector intersects an edge of the new triangle, then repeat the rotation process. Many rotations may be required before the final destination is reached. Figure 6-4 demonstrates this process.

Step 4: Check for collisions In order to determine whether the walker has intersected any of the elements of the cluster, compute the distance between the walker's position and the position of each element in the cluster. If no intersections are detected, then Step 3 is repeated and another random step is taken. However, if the center of the walker is closer than $2r$ to the center of any element in the cluster, it has penetrated that element. In this case, the walker sticks to the cluster according to a sticking probability P_s based on the local curvature of the cluster at this point of intersection. P_s is defined in Equations 5.10 and 6.4.

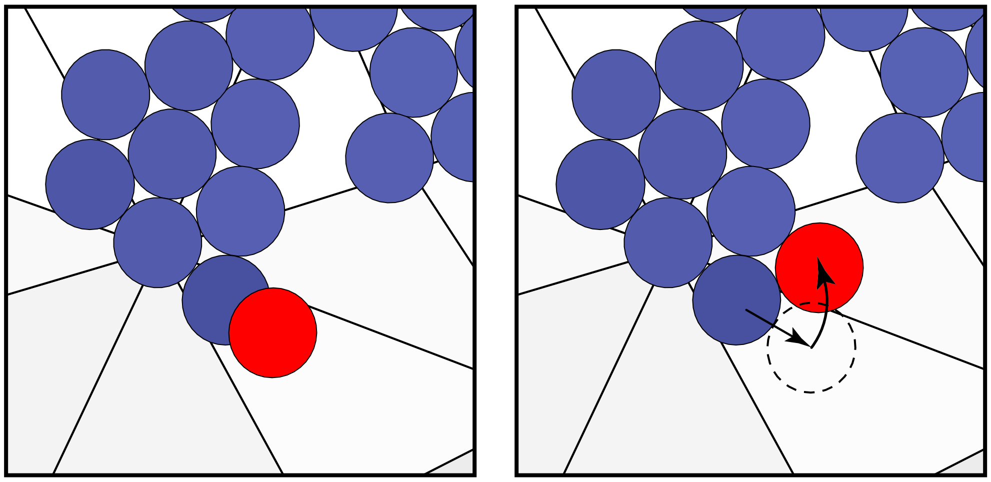


Figure 6-5: Before a walker is added to the cluster, it is moved backward to the point of intersection and then rotated until it touches another element.

If the walker does not stick, return it to its last position before it intersected the cluster and restart its random walk from that point, according to Step 3. If it does stick, first move the walker backward until it just touches the cluster without penetrating. Then, relax its position by rotating the walker around the intersected element until it just touches another element of the cluster (Figure 6-5). This ensures a more dense packing. Finally add the walker to the aggregate and repeat the entire process with Step 2.

6.2 Curvature

The lichen cluster is composed of a collection of elements and has no explicit representation as a curve or surface. When a walker collides with the cluster, some method must be employed to approximate the curvature of the cluster at that point since this value is needed to compute the sticking probability P_s. I use a method that lends itself to the cluster-based structure of the lichen data. The strategy I use to approximate the curvature is to first count the number of elements in the cluster that are within a given radius of the random walker and then take the ratio of this number to the total number of elements that would have been in a densely packed cluster.

Let n be the number of elements within a distance of L from the random walker. Since the surface of the object is approximately flat near the walker, n can be thought of as the number of elements within a circle of radius L. Let N_L be the number of densely packed elements within a half circle of radius L. The curvature, κ, is computed as:

$$\kappa = \frac{N_L - n}{N_L}. \tag{6.4}$$

For a convex region of the cluster, $N_L > n$ and $\kappa > 0$. For a flat region, $N_L = n$ and $\kappa = 0$. For a concave region, $N_L < n$ and $\kappa < 0$. (See Figure 6-6).

I calculate n by iterating over the elements and computing the distance from the center of each element to the center of the walker. I use an optimization similar to that described in Section 6.3.2 to accelerate this process. To compute N_L, note that in a dense packing, the percentage of area taken up by elements can be computed by connecting the center of four nearby elements to form a

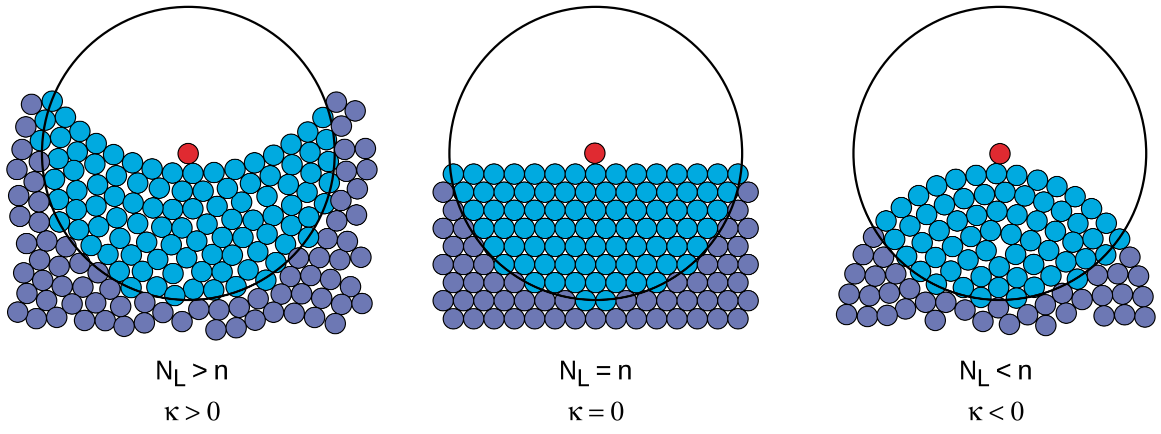


Figure 6-6: When the random walker (shown in red) touches the interface, the curvature is estimated based on n, the number of elements within a circle of radius L, and N_L, the number of densely packed elements within a circle of radius L. (Left) If the interface is concave, $N_L > n$ and the curvature is positive. (Center) If the interface is flat, $N_L = n$ and the curvature equals zero. (Right) If the interface is convex, $N_L < n$ and the curvature is negative.

diamond. This diamond covers one entire element with area πr^2. The area of the diamond itself is $2r^2\sqrt{3}$ (Figure 6-7). Since the diamond shapes can be tiled in the plane, the percentage is found by dividing the circle area by the diamond area to get a value of $\pi/2\sqrt{3}$. Then, the number N_L of elements in a given area A is:

$$N_L = \frac{A\frac{\pi}{2\sqrt{3}}}{\text{area of 1 element}} \tag{6.5}$$

Since A is a half-circle of radius L and the area of one element is πr^2,

$$N_L = \frac{\frac{\pi L^2}{2}\frac{\pi}{2\sqrt{3}}}{\pi r^2} = \frac{\pi L^2}{4r^2\sqrt{3}}. \tag{6.6}$$

6.3 Optimization

Implemented "as is," the algorithms in Section 6.1, though correct, would execute quite slowly. In this section I describe several optimizations that I have developed to accelerate the DLA simulation. With these optimizations, a large lichen cluster containing 100,000 elements can be grown in about 15 minutes using a single 750 MHz PIII processor.

6.3.1 Errant walkers

Nothing in the random walk simulation guarantees that walkers will intersect the cluster or even move toward it. Some walkers may move very far away from the cluster and delay the simulation for a long time before they return to the cluster's vicinity. In order to prevent this delay, compute the distance d_c between the walker and the centroid of the cluster after each step of the random walk (that is, at the end of Step 3). If d_c is greater than a threshold value, simply delete the walker and

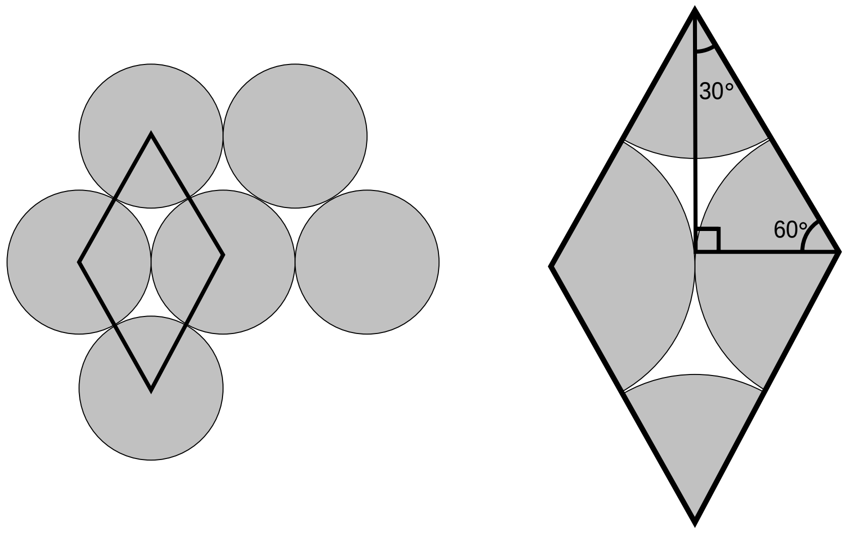


Figure 6-7: To compute the percentage area occupied by elements when the elements are densely packed, connect the centers of four adjacent elements. The gray area inside the diamond is equal to the area of one element, or πr^2. The total area inside the diamond is $2r^2\sqrt{3}$. After dividing, the percentage is $\pi/2\sqrt{3}$.

create a new one as described in Step 2. Since the size of the cluster increases as the simulation progresses, the threshold distance must also increase. In the examples in this thesis, I used threshold values in the range of $1.5R_e$ to $3R_e$, where R_e is the effective radius of the cluster defined in Step 2. Figure 6-8 illustrates this optimization.

6.3.2 Spatial sorting for collision detection

In the collision detection algorithm of Step 4, the distance between the random walker and every element in the cluster must be computed. The cluster size may grow to several hundred thousand elements and the collision test must occur after every random step, so the collision detection algorithm is an important one to optimize. I accelerate this step using a spatial sorting algorithm that allows elements nearby the random walker (if any) to be quickly selected. As the random walk simulation progresses, the triangle of the underlying mesh in which the walker currently resides is always known. Thus, the mesh itself provides a convenient spatial sorting mechanism. When random walkers are incorporated into the cluster, they are sorted by the triangle in which they resided when they intersected the cluster. For a given triangle, the list of elements associated with that triangle can be retrieved in constant time. The collision detection algorithm is modified as follows. Rather than iterating over every element in the cluster, instead consider the set $\mathcal{T}$ consisting of the random walker's current and nearby triangles. For each triangle in $\mathcal{T}$, iterate over all elements associated with that triangle, checking for collisions as defined in Step 4. If the walker must be added to the cluster, add it to the list of elements associated with the triangle in which it resides.

The set $\mathcal{T}$ must contain every triangle that could possibly contain an element with which the random walker has collided. The walker's current triangle is based on the location of the walker's center. Since the walker has a fixed radius r that defines a sphere, it may collide with an element in a nearby triangle. For example, if the walker is close to the edge of its current triangle, it may have collided with an element in the adjacent triangle. Or, if it is close to a vertex, it may have collided

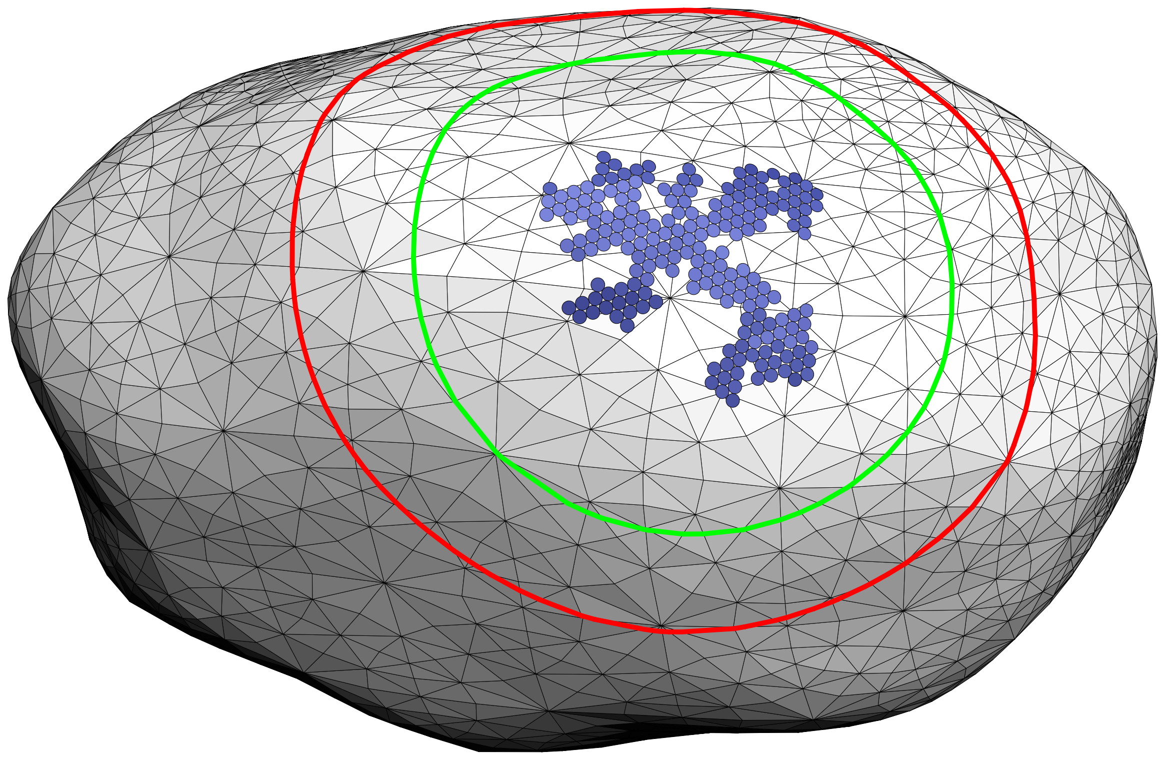


Figure 6-8: New random walkers are created on the inner circle, shown in green. As an optimization, if a walker moves outside of the outer circle, shown in red, it is reset to another position on the inner circle. The sizes of the inner and outer circles are increased as the cluster grows.

with an element in any of the triangles that share the same vertex. Finally, if the triangles in the mesh are small compared to r, the walker may even collide with an element in a triangle that does not share an edge or vertex with the walker's current triangle.

In my implementation, I compute $\mathcal{T}$ as a preprocess using the following $O(n^2)$ algorithm, where n is the number of triangles in the mesh. For each triangle T, compute its center C and consider each of its three vertices. Let q be the distance to the vertex that is farthest from C. Now, create a sphere centered at C with radius $q + r$. Iterate over every triangle in the mesh and collect all triangles that are completely inside of the sphere or that intersect the sphere with an edge. Let this collection be the set $\mathcal{T}$ for triangle T. Figure 6-9 shows the set $\mathcal{T}$ for a given triangle in a mesh.

6.3.3 Large random steps

The distance moved in each random step of Step 3 is small compared to the distance the walker must travel before it collides with the cluster. The walker may perform thousands of random steps before collision. I accelerate the random walk by using two mechanisms that allow the walker to take a random step much larger than r. The length s of this step is chosen to be the greater of the lengths allowed by the two mechanisms.

The first mechanism allows the walker to take a large step when it is outside of the cluster's effective radius. At any given point during the simulation, the cluster's centroid and its effective radius R_e are known. To calculate a large step size, first compute the distance s_c between the walker's current position and the centroid of the cluster. If $s_c > R_e$, the walker is outside of the

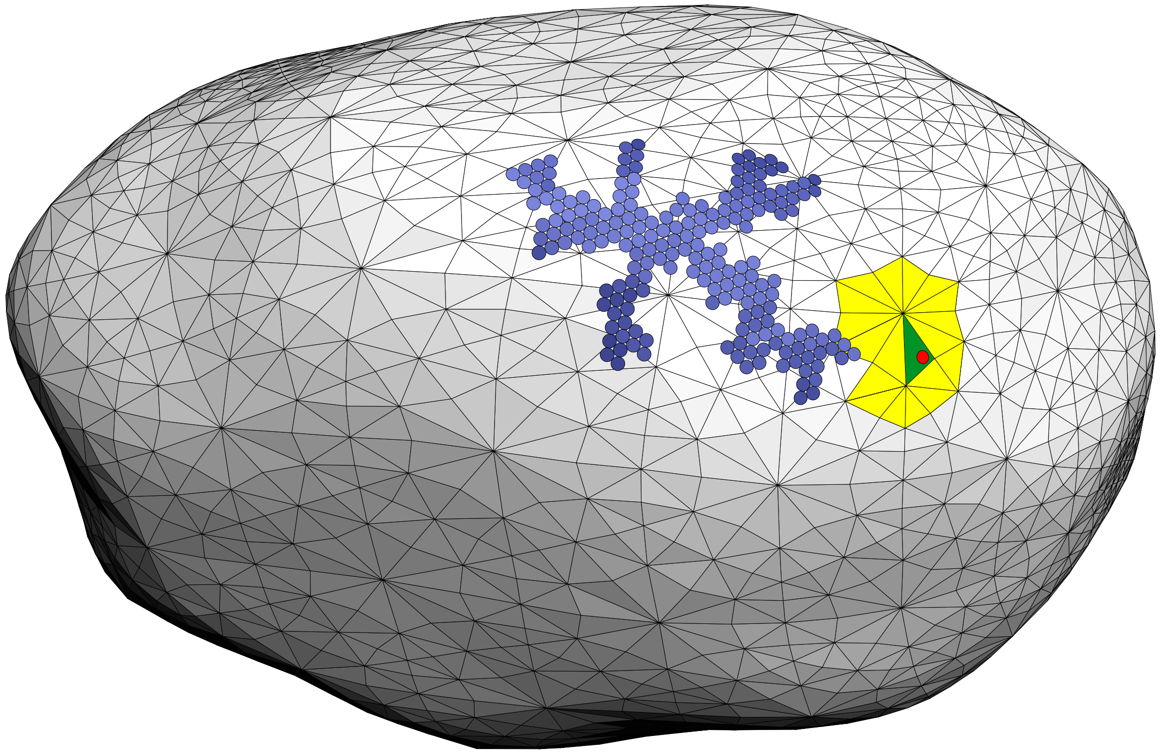


Figure 6-9: The triangle shown in green is the walker's current triangle. For any position in the green triangle, the yellow triangles represent a conservative estimate of all triangles that could contain an element with which the element has collided. The green and yellow triangles together make up the set $\mathcal{T}$.

cluster's effective radius. In this case, allow the walker to take a large step s_1 equal to $s_c - R_e - r$. If the step size is chosen in this way, the walker will not penetrate more than r units into the cluster, which is the same behavior as the unoptimized algorithm.

The second mechanism uses information from the collision detection algorithm to compute a large random step. When the optimized collision detection algorithm is performed, the elements that belong to any triangle within a radius of $q + r$ from the center of the current triangle are tested, as described in Section 6.3.2. A distance s_2 can be computed as follows:

$$s_2 = \min((q + r) - s_t - r, s_e - r), \tag{6.7}$$

where s_t is the distance from the walker's current position to the center of the current triangle, and s_e is the distance from the walker's current position to the closest element in the surrounding region of triangles. If no elements belong to any of these triangles, then $s_e = q + r$. As with s_1, if s_2 is chosen in this way, the random walker may intersect the cluster but will not penetrate it by more than r units.

Since both s_1 and s_2 are valid step sizes, let the actual step size s be the larger of the two. Since the rules used to choose s_1 and s_2 allow them both to be less than r, it is necessary to impose a minimum length on s. If s is less than r, set s equal to r. This prevents the walker from taking exceedingly small steps when it is close to the cluster. Equation 6.1 is modified to use this new step

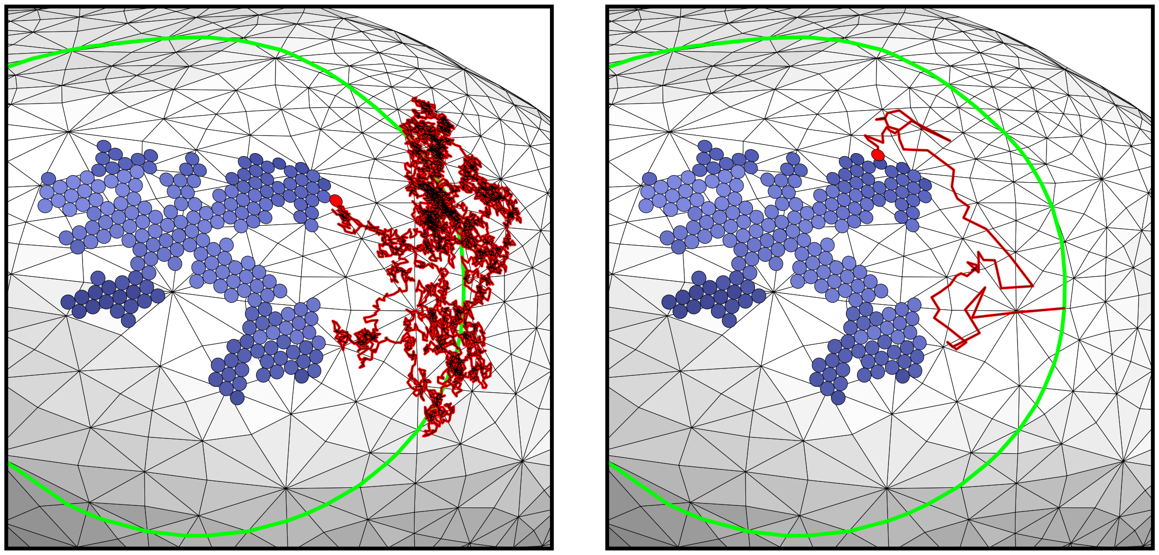


Figure 6-10: The random walk on the left was generated without the large step optimization and contains 4971 segments. The walk on the right used the optimization in Section 6.3.3 and contains only 58 segments.

size:

$$d = s + r\sqrt{rand_1} - r. \tag{6.8}$$

This formula chooses a random distance d between $s - r$ and s. As the walker approaches the cluster, s decreases until it equals r. In this case Equation 6.8 exactly matches the original Equation 6.1. Figure 6-10 shows a random walk with and without the large step optimization.

CHAPTER 7

IMPLEMENTATION

I implemented the lichen simulator in C++ using an object-oriented paradigm. The simulation, including all supporting tools, consists of about 9,200 lines of code and compiles under IRIX 6.5, Linux RH7, and Windows 2000. The entire system can be broken down into four semantic parts: the base simulation, the run-time graphics layer, the distributed layer, and the rendering and animation tools. In this chapter, I present the class structure and functionality of the base simulation. Then I discuss the run-time graphics layer. Finally, I describe the distributed implementation. The rendering and animation tools are covered in the next chapter.

7.1 Base simulation

Six classes comprise the base lichen simulation: `LSimulator`, `LObject`, `LTriangle`, `LAggregate`, `LElement`, and `LCircle`. Figure 7-1 shows a diagrammatic representation of the organization of these classes. The `LSimulator` class is the controlling class of the simulation. It contains the main simulation loop and a `Simulate()` routine that performs a single step of the simulation. The `LSimulator` contains an instance of `LObject`, `LAggregate`, and `LCircle`.

The `LObject` class stores information about the mesh on which the lichen grows. When created, this class loads the mesh from disk as well as an auxiliary file that contains precomputed information about the mesh. The auxiliary file has an entry for each triangle that specifies the three adjacent triangles, a list of the nearby triangles used in collision detection, and a list of the nearby triangles used when computing the sticking probability. After loading this information, the `LObject` class creates a list of `LTriangle` classes that represent the mesh. Each `LTriangle` class contains pointers to its adjacent triangles, as well as to the nearby triangles as specified in the auxiliary file. After the list of `LTriangle`s has been created, information such as the triangle's normal, orthogonal vectors, and the rotation matrices that rotate from the triangle's plane to the plane of the three adjacent triangles is computed and stored in the `LTriangle` class.

The `LAggregate` class is responsible for storing the elements of the lichen cluster. It contains an array of `LElement` lists, such that there is one list for each triangle. These lists are needed for the spatial sorting optimization described in Section 6.3.2. When the `LAggregate` is created, it generates the initial cluster. When an `LElement` is added to the cluster, the `LAggregate` adds the `LElement` to the appropriate list. Since the `LAggregate` has access to all of the cluster information, it contains the routines to perform collision detection and evaluate the sticking probability.

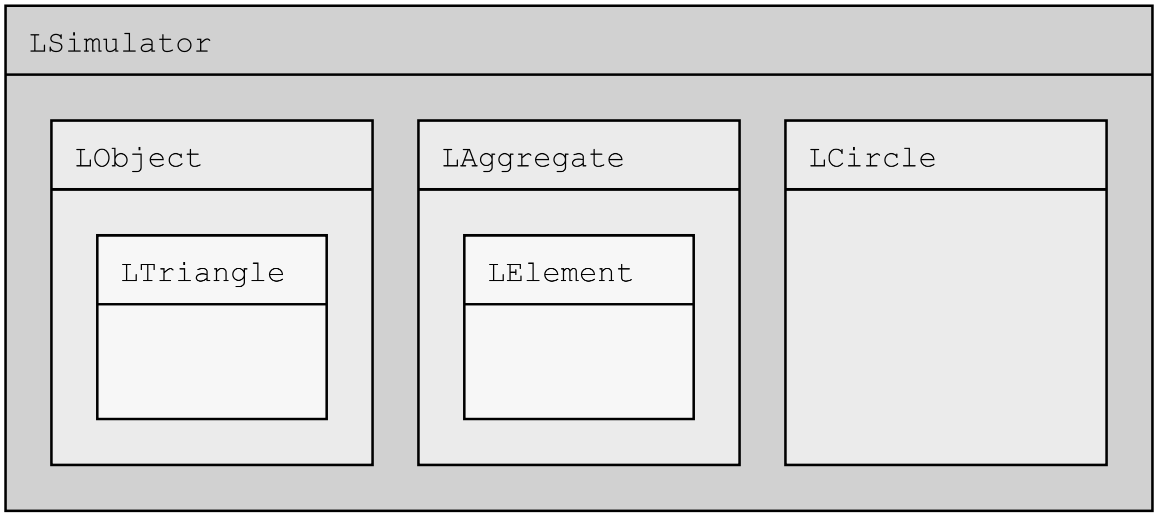


Figure 7-1: The lichen simulation class structure is organized as follows. The `LSimulator` class is responsible for the main simulation loop and contains as data an instance of `LObject`, `LAggregate`, and `LCircle`. The `LObject` class represents the underlying triangle mesh and contains an array of `LTriangle` primitives, each of which stores information about a single triangle. The `LAggregate` class represents the growing cluster and contains lists of `LElement` instances. The `LCircle` class is responsible for maintaining the line segments from which new random walkers start.

The `LAggregate` is also responsible for writing the cluster information to disk as the elements are added.

An instance of the `LElement` class represents a single random walker. When a new walker begins its random walk, the `LSimulator` creates a new `LElement` instance. The `LElement` contains all of the routines needed to perform its random walk and cross from one triangle to another. At each iteration of the main simulation loop, the current `LElement` takes one random step. Eventually, it intersects the cluster and is added to the `LAggregate`.

Finally, the `LCircle` class represents the collection of line segments on which new walkers begin their walk. The `LCircle` class contains a routine to build the line segments by computing the intersection of a sphere with the edges of the triangle mesh. And, it can return a random point along one of these line segments. As the cluster grows, the `LSimulator` instructs the `LCircle` to update its list of line segments.

7.2 Run-time graphics

I perform run-time graphics using the OpenGL graphics library. A subclass of `LSimulator` called `LGLSimulator` contains all of the graphics routines and replaces the main loop of the simulation with one that also contains calls to redraw the graphics and process keyboard and mouse events. The mouse is used to pan and dolly the camera, as well as control a virtual track ball that rotates the object. Pressing keys on the keyboard turn on or off different visualization functionalities. As elements are added to the cluster, the `LGLSimulator` draws the underlying mesh as well as the cluster. The elements of the cluster can be represented either as points or as circles with radius r drawn in the plane of the element's triangle. The color of each element is chosen based on the

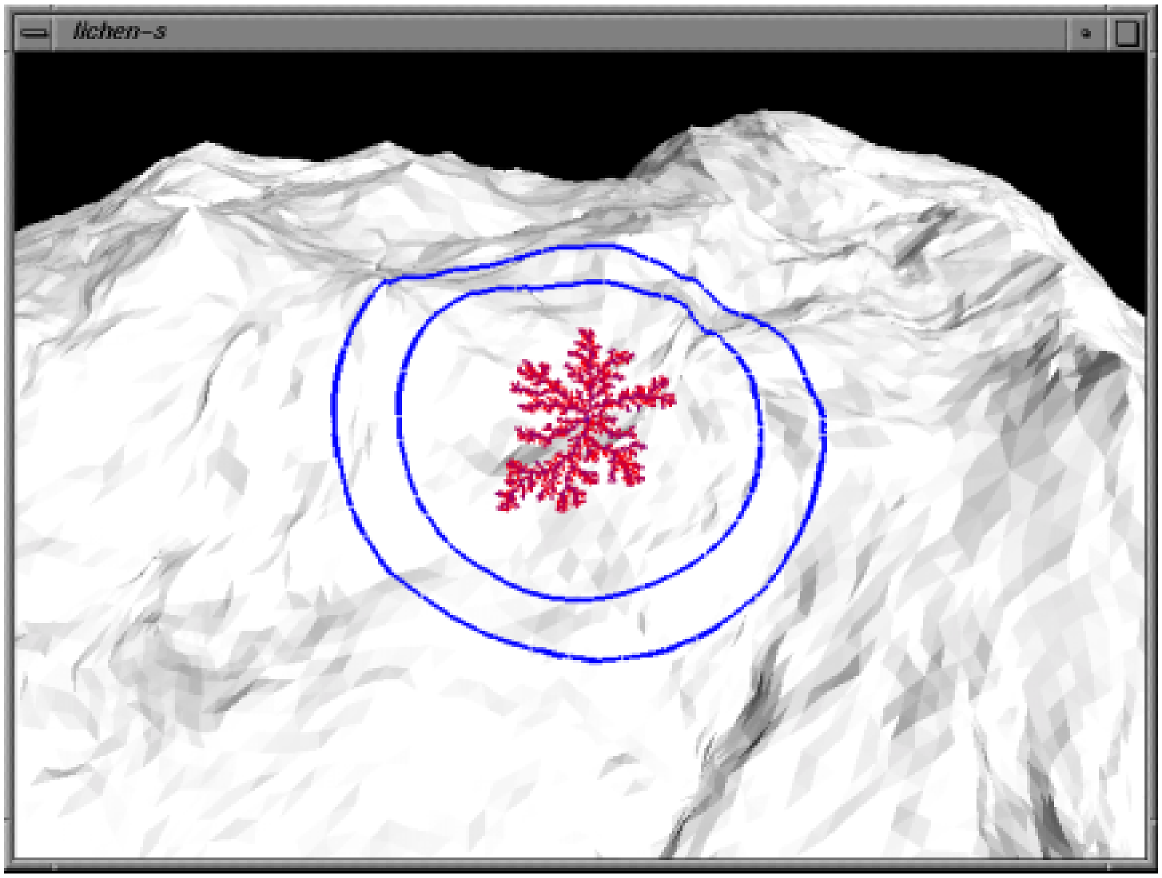


Figure 7-2: As the simulation executes, the run-time graphics layer renders a 3D representation of the scene to a window on the screen, allowing the user to monitor the progress of the cluster growth. The white object is the triangle mesh on which the lichen is growing. New walkers start at random positions on the inner blue circle and are reset to a new position if they stray outside of the outer blue circle. The cluster is drawn by rendering one point for each element, where the color is based on the probability that was computed when the element was added to the cluster. In this example, the cluster is predominantly red because the parameters were set such that random walkers had a high probability of sticking.

sticking probability that was used when the element was added to the cluster. Other visualization options include drawing the walker as it performs its random walk, drawing the line segments contained in the `LCircle` class, highlighting the walker's current triangles and nearby triangles, as well as others. Figure 7-2 shows a screen capture of the lichen simulation window.

In addition to on-screen rendering, I developed a PostScript renderer that uses the OpenGL feedback mechanism to render the scene to an EPS file. I used this functionality to generate many of the figures in this thesis. See Figure 6-3, for example.

7.3 Distributed implementation

I designed and implemented a distributed version of the lichen simulation that uses a client/server architecture to accelerate computation by simulating several random walkers at the same time on

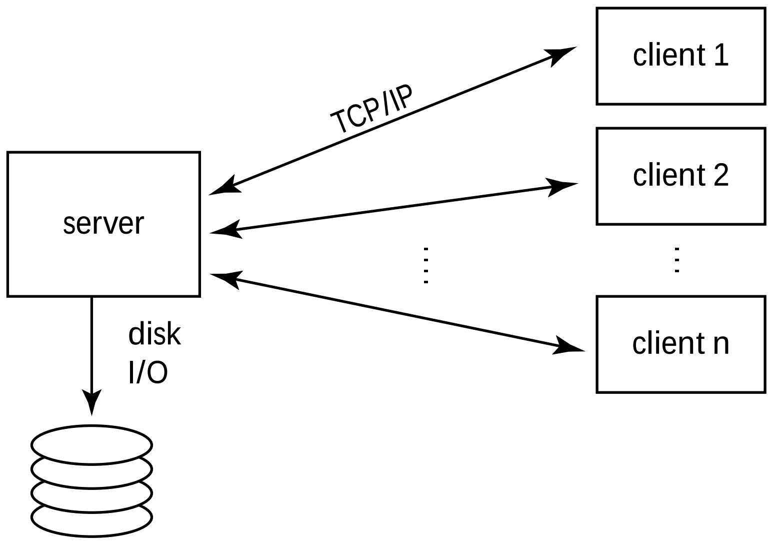


Figure 7-3: In the distributed implementation, many clients connect to a single server. When a client adds an element to it's copy of the cluster, it informs the server and the server then broadcasts a message to all of the other clients instructing them to add the same element to their local copies of the cluster. The server is also responsible for writing the cluster information to disk as the clients inform it of new additions.

different computers. The server stores the cluster information in memory, writes it to disk, and manages all of the clients. The clients execute the actual simulation code. Communication between the clients and the server is accomplished using TCP/IP sockets. Figure 7-3 gives a schematic view of my client/server model.

When the server is started, it loads the files associated with the simulation and initializes the cluster. The server writes the initial cluster information to disk. It then waits for clients to connect. When a client is started, it loads the files associated with the simulation, but does not initialize the cluster. Instead, it establishes a two-way socket communication with the server. After the server accepts the client's connection attempt, the server sends data about all elements in the cluster to the client. The client adds these elements to its copy of the cluster but does not write them to disk. Then, the client begins the simulation. After each iteration in the main simulation loop, the client checks for messages from the server. If a message about a new element is waiting to be read on the socket connection, the client reads the data and adds this new element to the cluster. It continues to read messages until no information is waiting to be read on the socket, at which time it resumes the random walk simulation. When the client's simulated random walker must be added to the aggregate, the client first adds the element to its own copy of the cluster. Then, the client writes the information about the element across the socket to the server. The server reads this information, and then broadcasts it to every other client. After doing so, the server writes the element information to disk. In this way, each time a client adds an element to the cluster, all of the other clients find out about it and add a corresponding element to their local copy of the cluster. To exchange information about an element, only the element's position and triangle index must be transmitted across the socket. New clients can connect to the server at any time, since the server will send a new client information about every element added to the cluster so far.

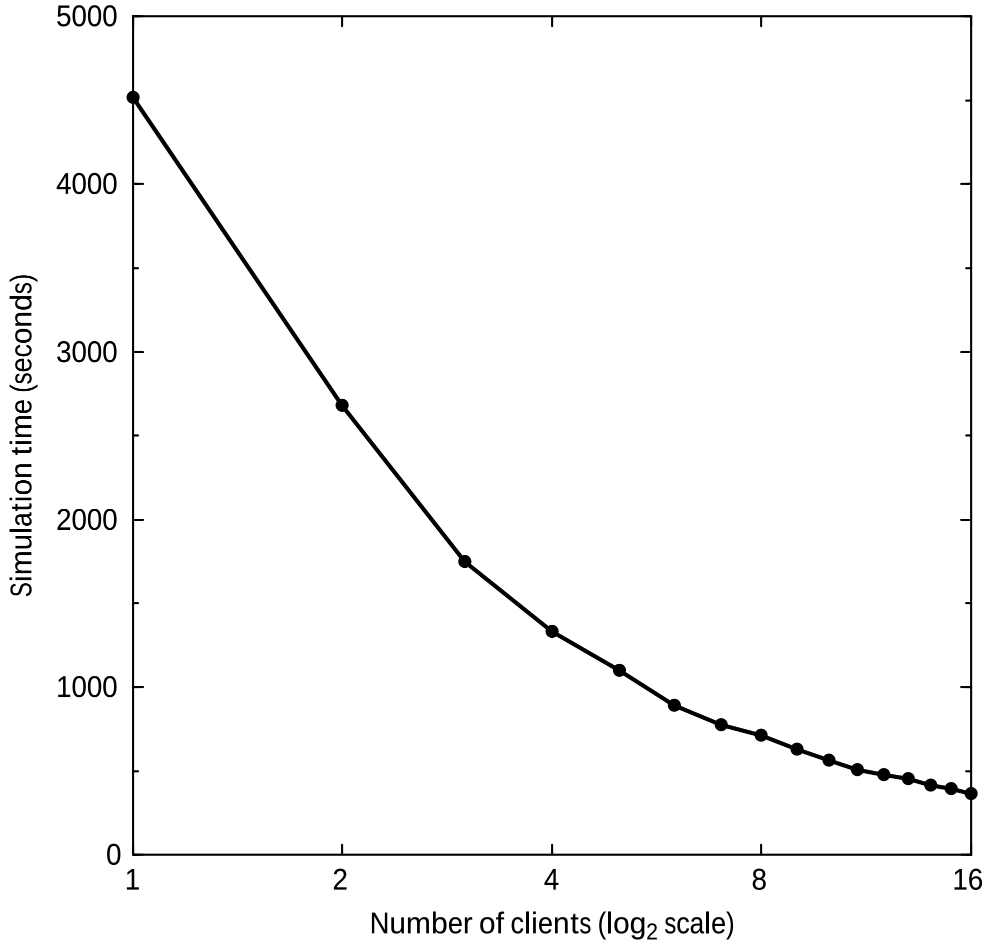


Figure 7-4: The timing results from growing a cluster of exactly 100,000 elements using a varying number of clients are shown above. The horizontal axis is a $\log_2$ scale. Ideally, the curve would be a straight line with slope equal to -1, since this would indicate that doubling the number of clients halved the simulation time. However, in my model, $O(N^2)$ messages are passed where N is the number of clients. As the number of clients increases past 10, the communication overhead becomes significant and prevents a linear speedup. All timing experiments were performed on identical Pentium II 400 MHz machines running Linux.

The client and server are each implemented as a subclass of `LSimulator`, with functionality added to do the network communication. A special GL client is a subclass of `LGLSimulator` and performs the graphic routines but does not execute the simulation code. The GL client allows one to watch the simulation as it progresses.

This design does not guarantee operation equivalent to the serial version of the simulation. In rare cases, two different clients may add a new element to the same location on the cluster at roughly the same time. Since communication with the server occurs only at the end of each simulation step, the second client could add the element to that site before it discovers that the first client has already added an element to the same location. I simply ignore these errors because they are rare and have little effect on the overall shape of the cluster. In practice, I noticed no errors when using between one and ten clients. With more than ten clients, errors became increasingly more common.

I conducted timing experiments to determine the efficiency of my distributed implementation. I ran the client/server simulation many times, growing a cluster of exactly 100,000 elements using from one up to sixteen clients. The results of these experiments are plotted in Figure 7-4.

CHAPTER 8

Rendering and Animation

8.1 Rendering

The primary goal of my research is to assist artists and modelers in creating visually rich digital images by providing a system to add the branching patterns of lichen to synthetic objects. My lichen growth simulation only partially fulfills this goal. It computes the growth patterns but does not provide a way to generate high quality imagery from the raw simulation data. In order to create high quality visualizations of the simulation results, I wrote a tool that transforms the raw output into a triangle mesh that can be merged with a synthetic scene and rendered using standard techniques.

The simulation output consists of an ordered list of information about each element in the cluster such as the element's position. From a biological standpoint, each element in the cluster represents a portion of the lichen. This portion is composed of algal and fungal cells that are stratified into layers, forming a thin volume. My overall strategy to generate a triangle mesh from the element information is to create a 3D function $F(x, y, z)$ that decreases in value with distance from the cluster. Then, I use an isosurface extraction routine to create a triangle mesh that approximates a constant value of this function. For a comparison of the raw simulation data with its extracted triangle mesh, see Figure 8-1.

To compute F, I take the sum of many separate falloff functions, where each function is centered at a different element. If there are E total elements in the cluster, then

$$F(x, y, z) = \sum_{n=1}^{E} f_n(x, y, z). \tag{8.1}$$

Here f_n is the falloff function for element n and is equal to:

$$f_n(x, y, z) = K \exp(d^2 \ln(1/2)/C^2), \tag{8.2}$$

where d is the Euclidean distance from the center of element n to (x, y, z) and K and C are constants chosen by the user. K determines the value of f_n at $d = 0$. C is equal to the value of d for which $f_n = K/2$ and can be thought of as a "half-life" parameter that controls the falloff rate.

After defining F, I use the implicit surface polygonizer developed by Bloomenthal [7] to extract a particular isosurface of the function. In practice, I don't sum over every element each time F is evaluated, since the polygonizer code calls F many times. Instead, I first sort the elements into a

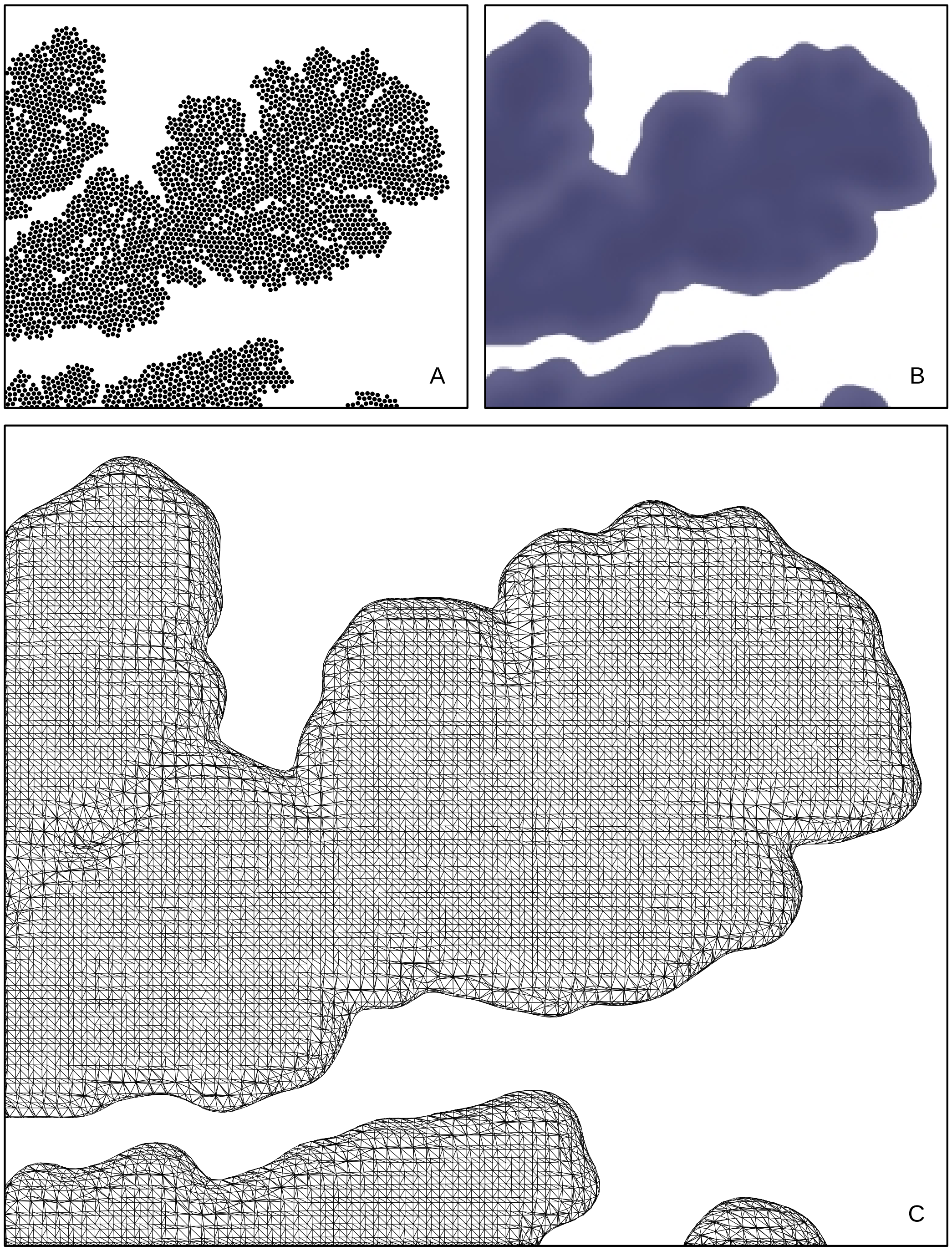


Figure 8-1: A closeup of a lichen cluster, show as (A) raw simulation data; (B) final rendering; (C) wire frame rendering. The slight color variations in the output image occur because the colormap used during extraction assigned less dense regions a lighter shade of blue. The wireframe rendering indicates that the mesh is unnecessarily dense and a great deal of polygon reduction could be performed without loss of detail.

3D grid based on their location in space. Then, when F is evaluated at a certain point, I iterate over the gird cells near the point and sum the functions for the elements that were sorted into those cells. The cutoff distance for this iteration is chosen such that elements that are left out of the sum would contribute a minimal amount to the total.

Two additional parameters control the mesh extraction process. The first parameter makes f_n decrease more steeply in the direction normal to element n. This has the effect of scaling the thickness of the extracted mesh. The second parameter makes f_n decrease more steeply for the outermost ring of elements. This causes the outer edges of the lichen mesh to taper down.

Lichen species exhibit a wide variety of pigmentation, which helps protect them from the sun's radiation. Bright colors such as orange, yellow, red and green as well as duller grays, browns and blacks are all common pigments. Parameters built into my rendering tool allow the user to add color to the resulting mesh. The user supplies a colormap that is parameterized by element density. In this way, dense regions of the cluster, such as the center, can be given a different color than the less dense regions such as the lobe tips. Another parameter allows the user to slightly darken older parts of the cluster. The age of each element is based on the order in which the element was added to the cluster.

My mesh extraction tool generates highly detailed and realistic looking meshes that can be rendered by virtually any rendering package. The pigmentation parameters can be used to mimic the colors of many real lichen. However, one drawback is that the meshes are dense and often contain on the order of millions of triangles. For camera positions where the lichen fills the entire frame, this amount of detail may be desired. However, for a scene with many lichen viewed from a distance, a less detailed mesh would be more appropriate. An adaptive polygonizer might be effective in generating meshes with fewer triangles. Or, a polygon simplifier such as [27] could be used as a post-process to reduce the polygon count.

8.2 Animation

Generating animations of lichen growth is particularly exciting. Many lichen grow only a few millimeters each year, which serves as a major impediment to scientists who wish to perform experiments related to their growth. By extracting animations from my growth simulation data, the scientific community may gain new insights into the lichen growth process.

Generating an animation from the simulation is straightforward. During the simulation, the information about each element is written to the output file immediately after the element is added to the cluster so that the resulting data is sorted by the order in which the elements were added. This ordering can be thought of as a sequence of events evenly distributed in time. That is, if t is a variable representing time, consider the first element to have been added at $t = 1$, the second element at $t = 2$, and so on. Consider this ordering to indicate the element's age so that the first element's age is 1, the second element's age is 2, and the 1000th element's age is 1000. Figure 8-2 shows a lichen cluster that has been colored according to age. The basic strategy for generating an animation of the lichen growth is to take several snapshots of the lichen at different values of t. For each snapshot or frame, the mesh extraction tool described in the previous section is used to extract a mesh that is then rendered using standard rendering techniques. However, for a given frame, only those elements of equal or lesser age are considered in the polygonization algorithm. All elements older than t are ignored.

This process alone yields rather jumpy animations since the elements are added at discrete points in time. I made an additional modification to the mesh extraction tool in order to generate smoother animations. Rather than ignoring all elements older than t, instead I linearly fade in the influence

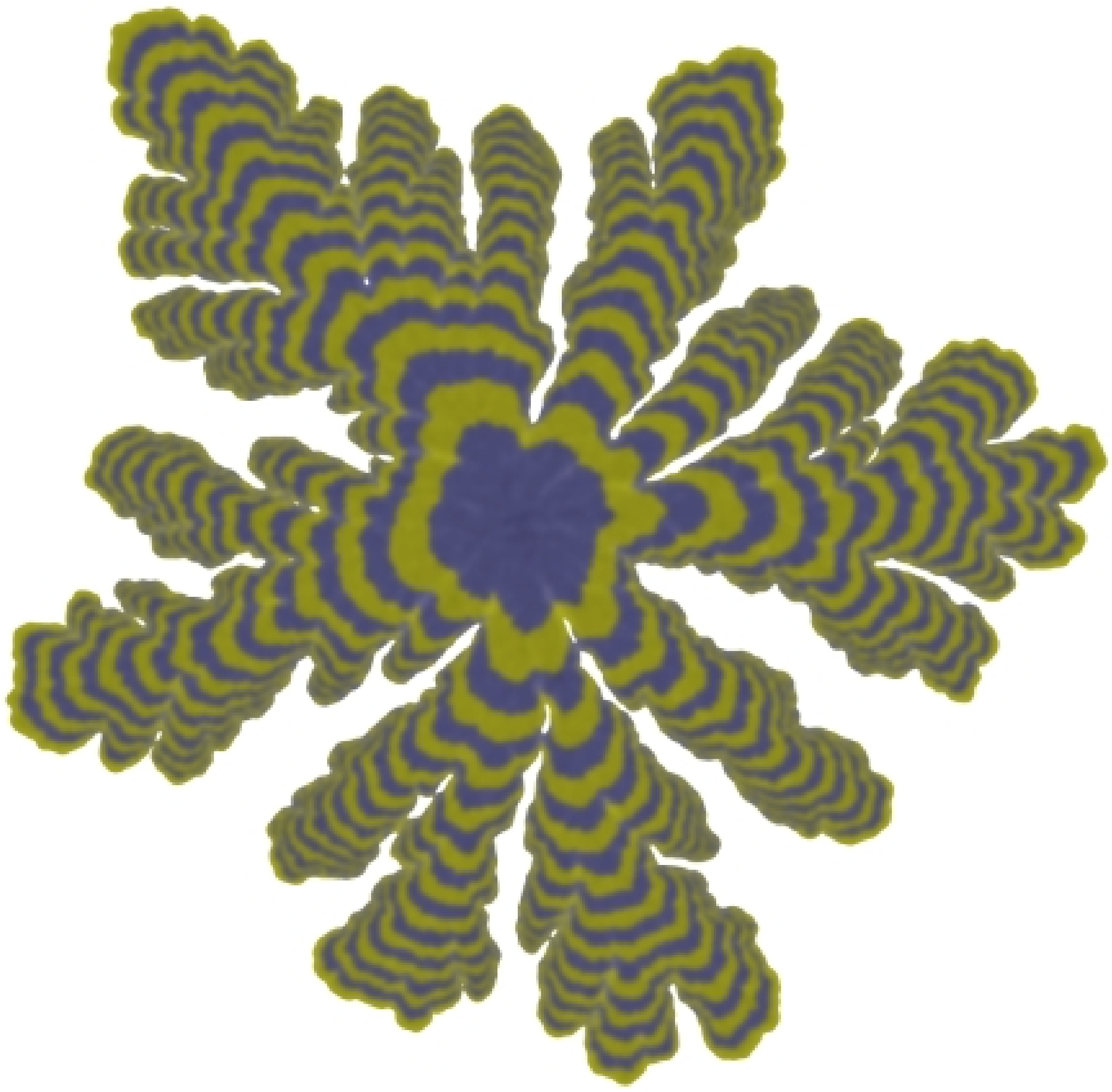


Figure 8-2: The alternating blue and yellow bands in this lichen correspond to the order in which the elements were added to the cluster, and therefore indicate the elements' age. The cluster consists of 219,581 total elements, and the color alternates after every 10,000.

of elements based on their age over a given window of time. If this window is W time units long, for example, element number $t + 1$ will have almost full influence, element $t + (W/2)$ will have half of it's normal influence, and element $t + W$ will have virtually no influence. The influence for a particular element n is modified by scaling the return value of it's falloff function f_n. Since the influence of new elements gradually fades in, the resulting triangle meshes increase in size more smoothly over subsequent frames in the animation. A filmstrip containing five frames from one such animation is shown in Figure 9-2.

CHAPTER 9

DISCUSSION

9.1 Results

In Figure 1-1, I presented one finished rendering created with my simulation. The lichen in this figure extends over an uneven rocky surface and exhibits the characteristic lobe division and engulfment. Figure 9-1 shows a closeup view of this lichen. The three-dimensional structure of the lichen is visible from this vantage point. The lichen's growth clearly corresponds to the object's geometry since it extents over ridges and humps in the rock's surface. Notice how the growth of the small, short lobe in the left side of the image has been inhibited by the its longer neighboring lobes.

Figure 9-2 consists of five equally spaced frames from an animation of the same lichen's growth. Small bulges and perturbations that develop earlier in the lichen's growth later extend into fully grown lobes. The fully grown lichen is about ten centimeters in diameter, indicating that the animation spans thirty years of growth, since a real lichen of that size is likely to be at least that old.

The lichen cluster used in Figures 1-1 and 9-1 and in the animation contains a total of 335,732 elements. The simulation ran for 33 minutes using four Pentium III 700MHz processors.

Figure 9-3 depicts a simulated lichen that has more of a space filling morphology. The thallus margin is uneven, but it hasn't split into independent lobes. This lichen grew near the point where a smaller branch stems from a larger one, and the shape of the lichen pattern reflects this underlying topology. This lichen is similar to *Brodoa oroarctica*, which is shown in the figure as an inset photograph.

Figure 9-4 demonstrates the parameter space of the lichen simulation and shows the range of patterns that it can generate. This figure shows a grid of lichen patterns generated by varying two parameters of the simulation that control the computation the sticking probability. Recall from Equation 5.10 that $P_s = \kappa + B$ where κ is the local curvature and B is a constant. The local curvature is computed as described in Section 6.2 and depends on a constant L, which is the curvature radius. In Figure 9-4, the curvature radius L is varied along the horizontal axis, and the constant B is varied along the vertical axis. All other conditions, including the element radius, were held constant over all simulations. Each lichen cluster was grown until it reached a specific size. Thus, the more dense clusters contain more elements than the less dense ones.

Intuitively, L normalizes the scale of the curvature computation. That is, it determines the size of the concave region that has the greatest positive curvature and the size of the convex protrusion that has the greatest negative curvature. L determines the size a protrusion must be in order to have

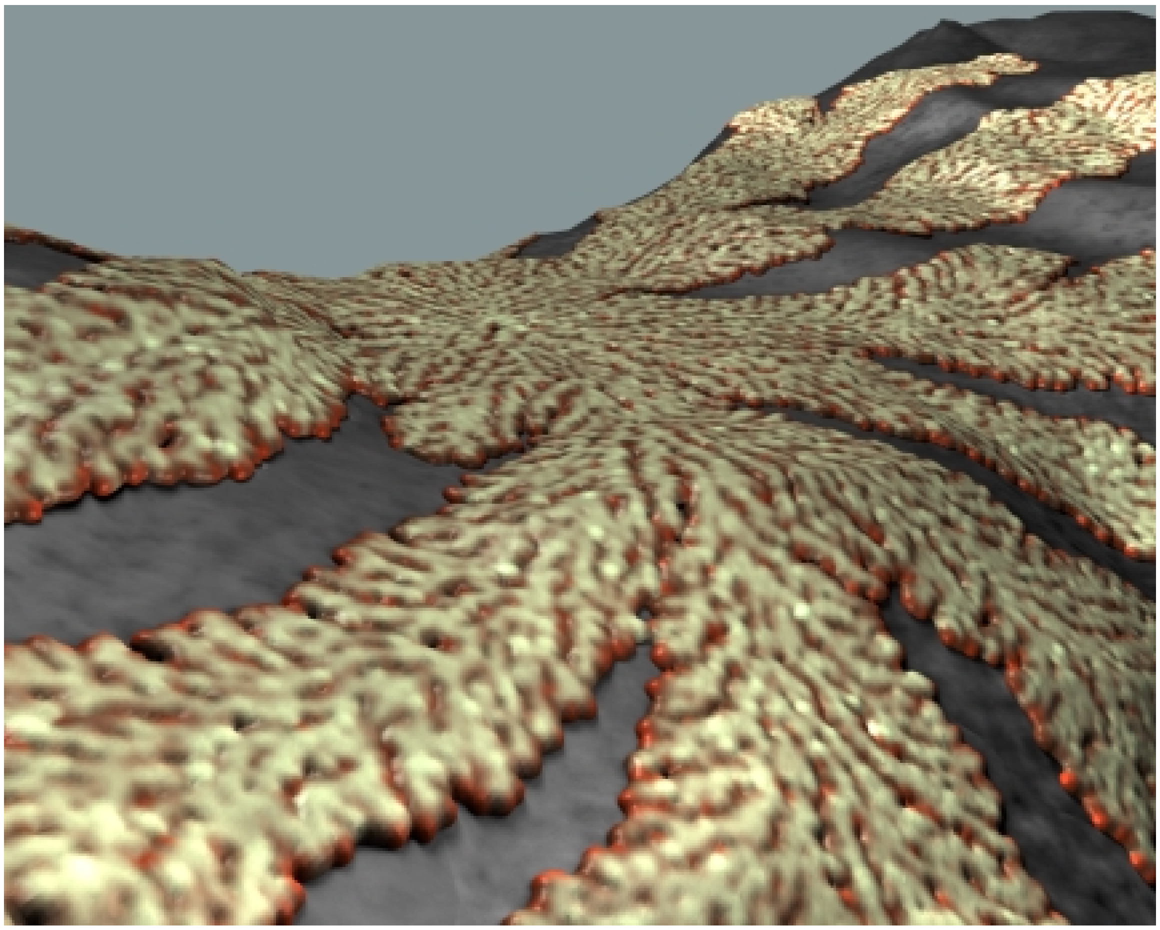


Figure 9-1: A close up view of the lichen shown in Figure 1-1.

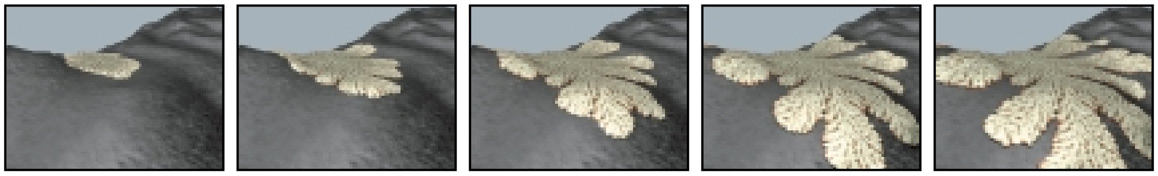


Figure 9-2: Five images from an animation of lichen growth. The images are equally spaced in time.

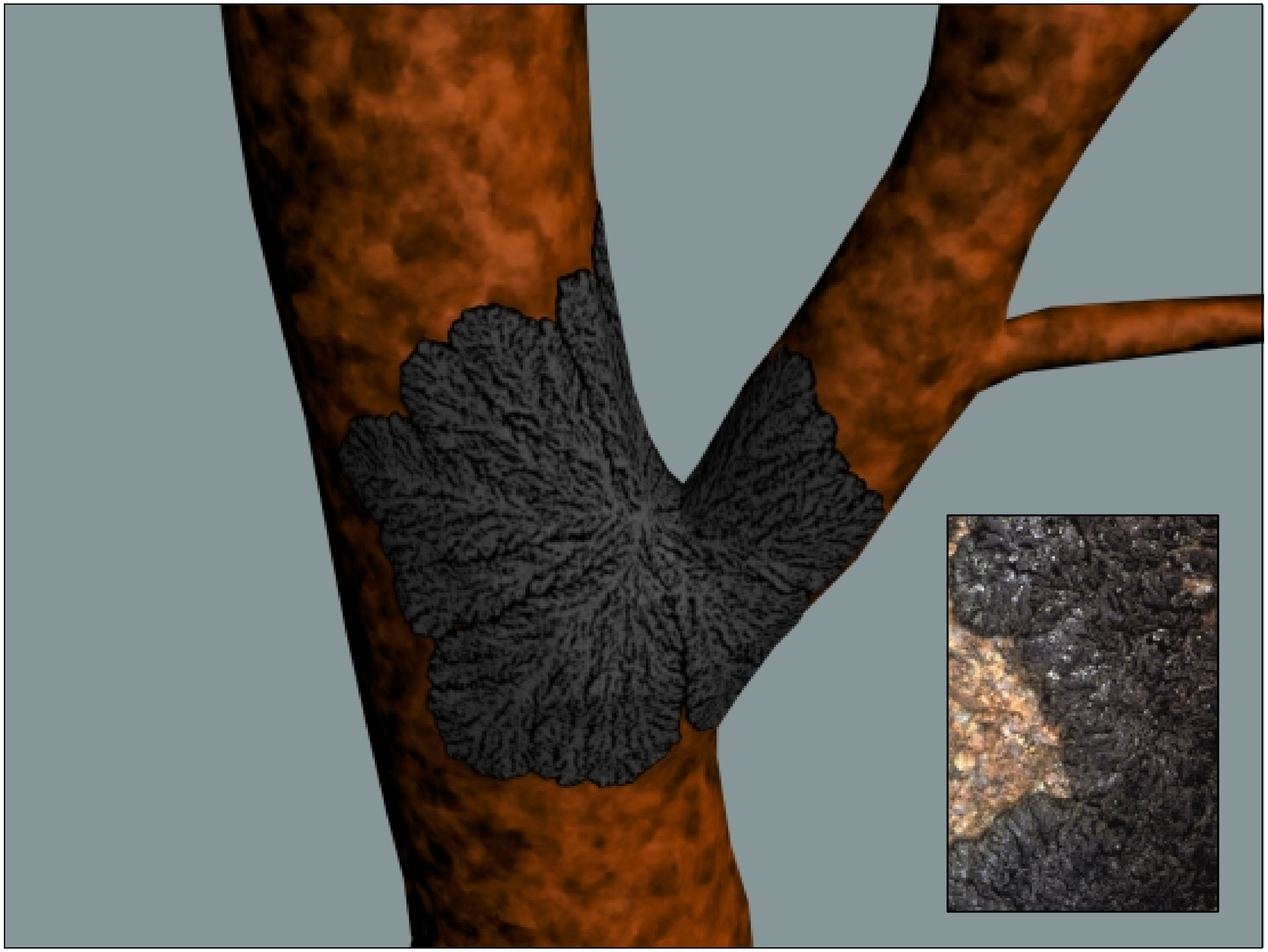


Figure 9-3: This simulated lichen, similar in appearance to the inset photograph of *Brodoa oroarctica* from [33], spans the branch point where the smaller branch stems from the larger bough.

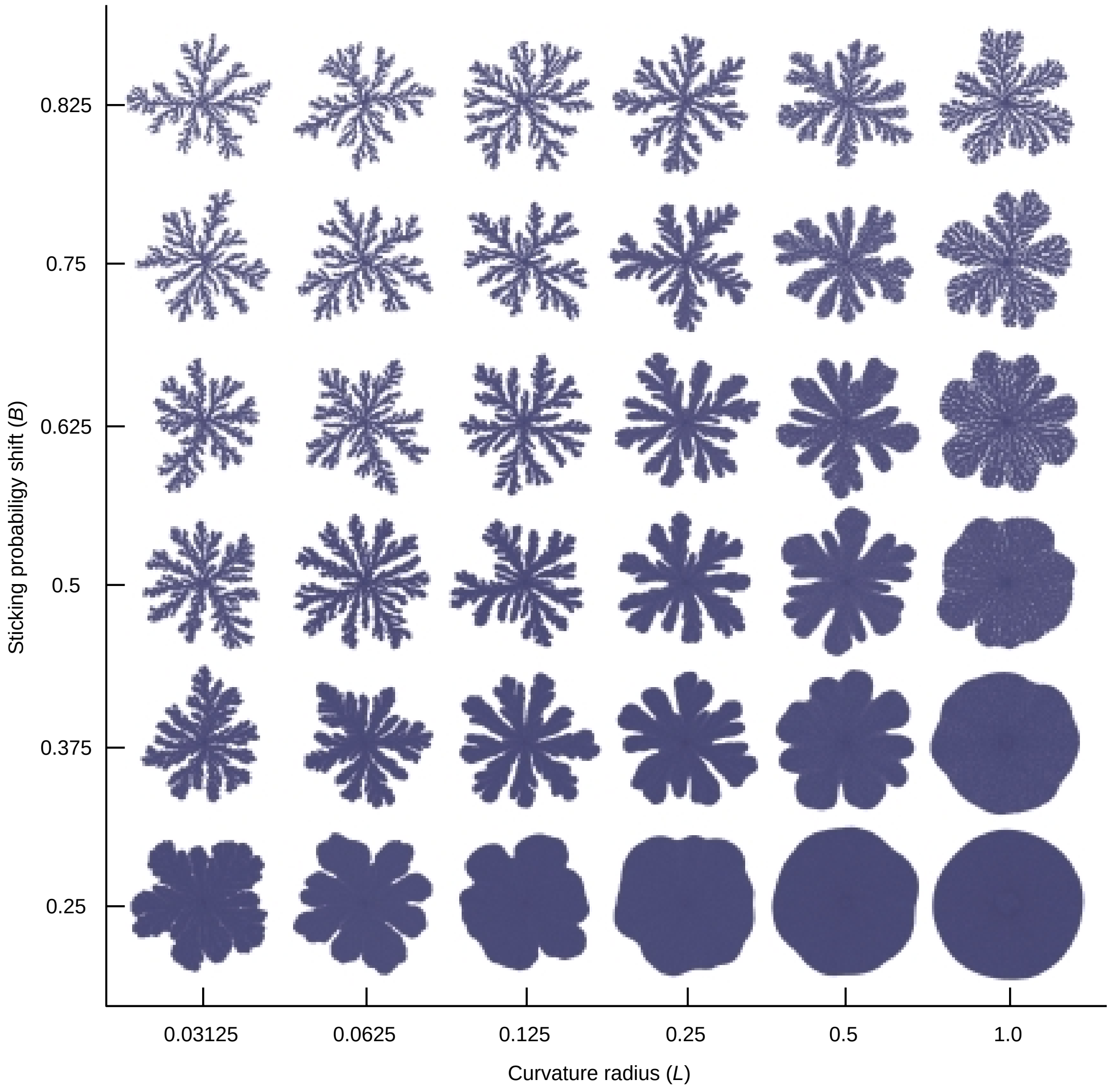


Figure 9-4: The different lichen clusters in this grid demonstrate the range of patterns that the lichen simulation can generate.

an influence on the curvature computation. Protrusions or concave regions much smaller or much larger than L have no influence on the curvature. In this way, L roughly determines the size of the branches of the cluster that develop as the simulation proceeds. This influence can be seen in Figure 9-4 because the lichen clusters on the right side of the grid have larger, fatter branches than those on the left side.

The constant B in Equation 5.10 shifts the sticking probability higher or lower. This shifting has the effect of determining whether the instability inherent in the random walk or the restoring force of the sticking probability is more predominant. If the instability is predominant, random perturbations constantly develop, resulting in a cluster that contains many coarse, uneven branches and has a characteristic fractal appearance. If the restoring force is more predominant, random perturbations are smoothed out and the lichen pattern contains smoother, fatter branches and appears more space filling. This effect is clearly seen in Figure 9-4 because the patterns at the top of the grid have thin, uneven branches while those at the bottom have more gradual, thicker ones.

9.2 Future work

My simulation focuses on the two-dimensional branching morphology of lichen. The lobes of many real lichen sometimes curl up away from the thallus and overlap each other. I would like to explore methods such as those of [14] that could be used to introduce wrinkled, folded, or curled surfaces into my simulation. Furthermore, some lichen thalli exhibit cracking as they age, possibly due to dessication. Changes to the lichen pattern caused by moisture content is another area of future work.

In my system, the location of lichen growth on an object is selected manually by the user. In a real situation, environmental influences such as sun exposure and water flow make it more or less likely for a lichen to grow in a given place. Incorporating environmental influences such as these into my system would make the growth patterns more consistent with the overall scene in which the lichen is situated. Furthermore, lichen rarely grow alone. Instead, entire colonies consisting of several different species often inhabit a substrate. Though scientists have studied this type of interaction, little is know about the interplay between many lichen growing together. Developing a growth simulation on the level of granularity of entire colonies is an exciting area of future research.

As I mentioned in Section 8.1, in my system the lichen pigmentation is controlled by the user as a post process. One of the most stunning aspects of real lichen is their wide variety of color. Thus, it would be interesting to incorporate a pigmentation model into the overall simulation, rather than simply specifying the colors as an afterthought.

Finally, I would like to extend my method to generate the patterns of other phenomenon that exhibit Laplacian growth. For example, the intricate patterns of snowflakes form as a result of the Mullins-Sekerka instability during solidification. As water freezes, the instability causes dendrites to form. The anisotropy of the ice crystals generates preferred orientations. This directional preference, together with surface tension, results in the patterns we associate with ice crystals. The mathematics governing this process is closely related to that of the lichen simulation.

In support of this claim, I have created a snowflake simulation by making only slight modifications to the lichen simulation. Rather than allowing the elements to attach to the cluster at any orientation, I enforced a hexagonal grid on the cluster. This grid mimics the preferred orientations caused by the ice crystal anisotropy. Then, I modified the sticking probability to encourage a new element to orient itself in the same direction as the element of the cluster with which it collided. I made no other modifications to the simulation. In particular, I do nothing to enforce symmetry or to explicitly cause the development of six major branches. Figure 9-5 shows a typical snowflake created with these modifications.

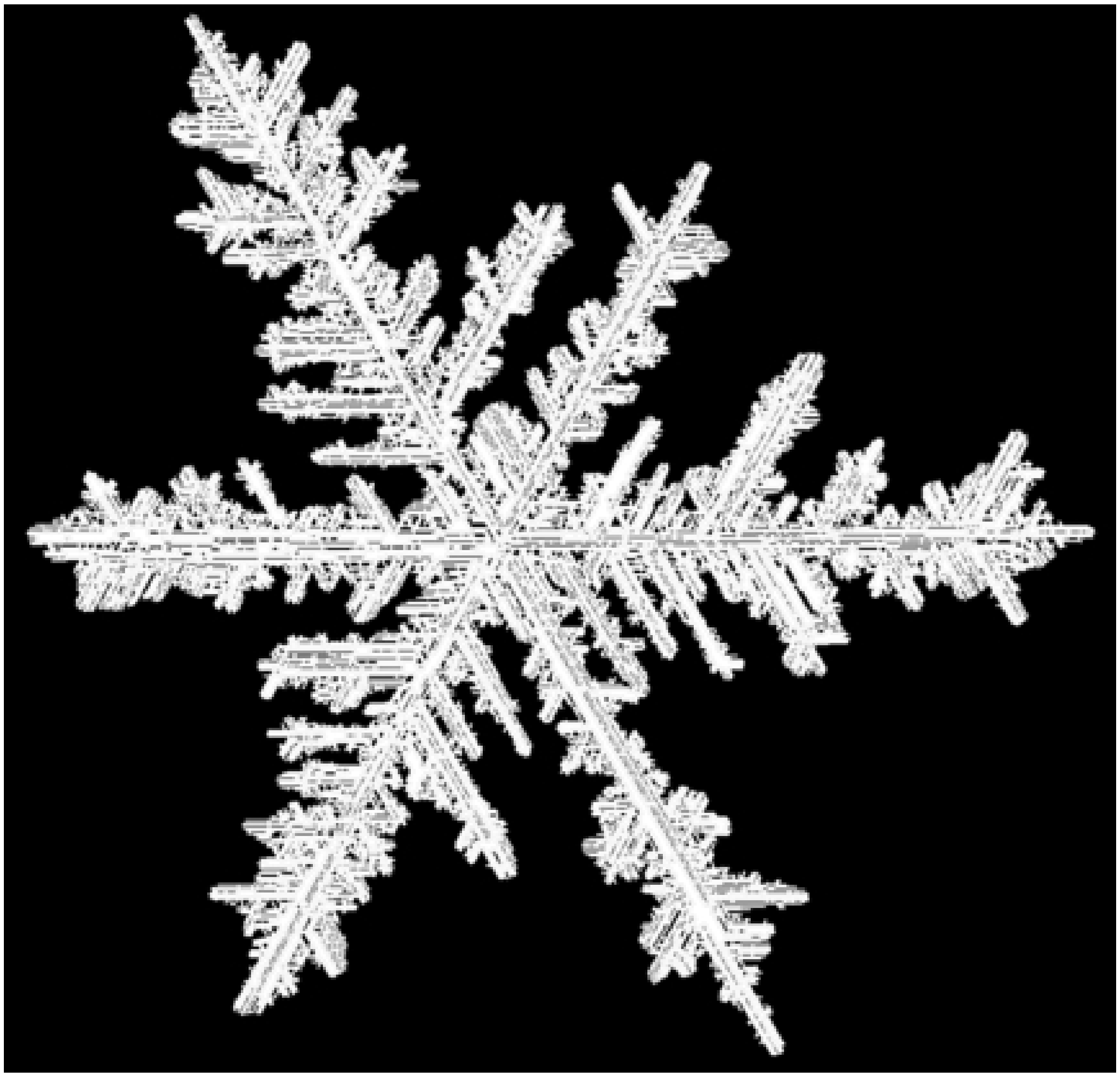


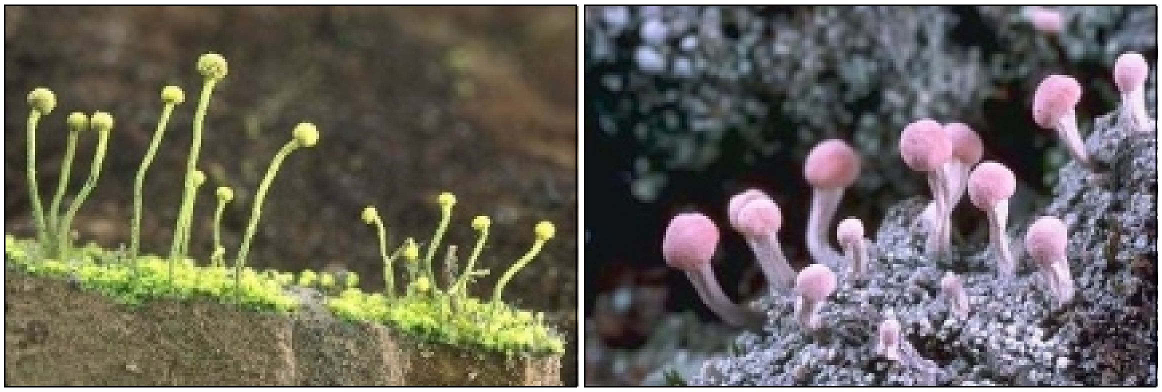
Figure 9-5: This snowflake pattern was generated after making only slight modifications to the lichen simulation. It suggests that my method can be extended to simulate other natural phenomenon that are governed by competition between instabilities and restoring forces.

APPENDIX A

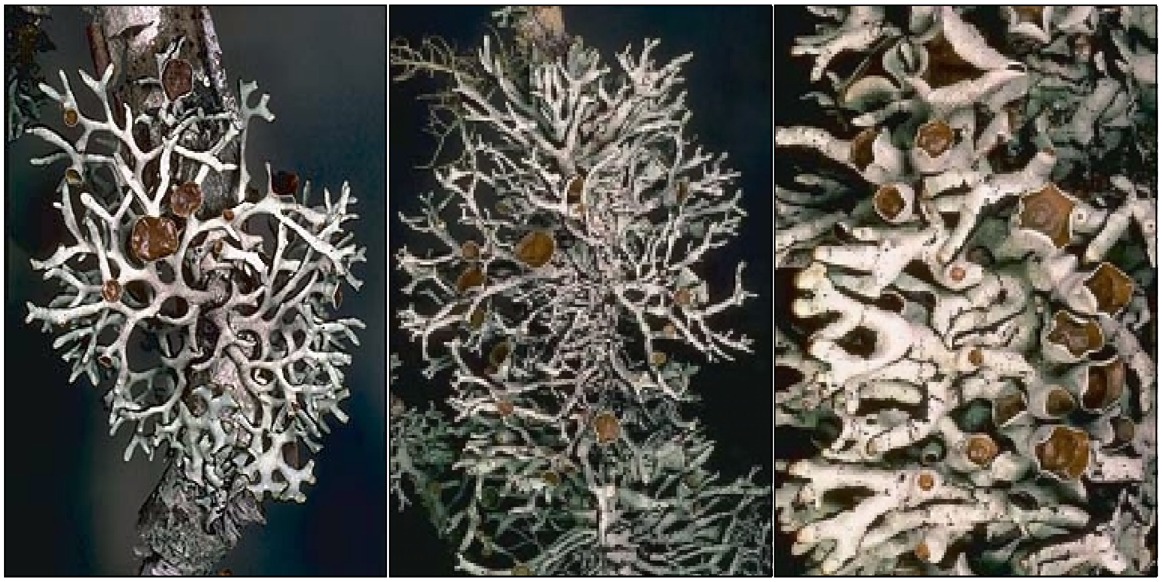
Image Catalog

I compiled this catalog of lichen images as a first step toward understanding their shape and form. Most images are from [8].

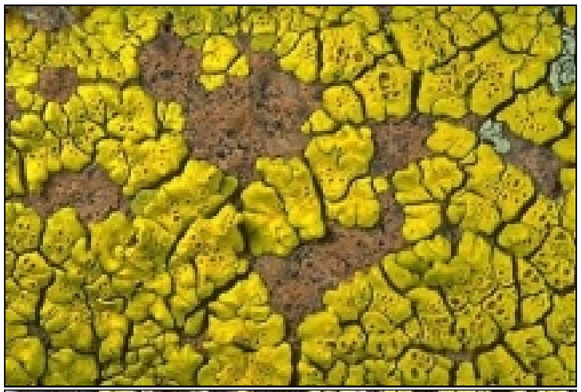
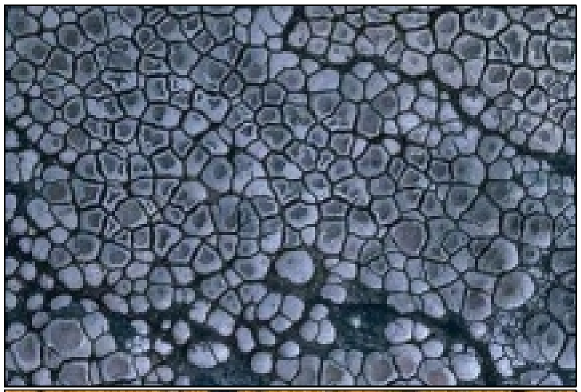
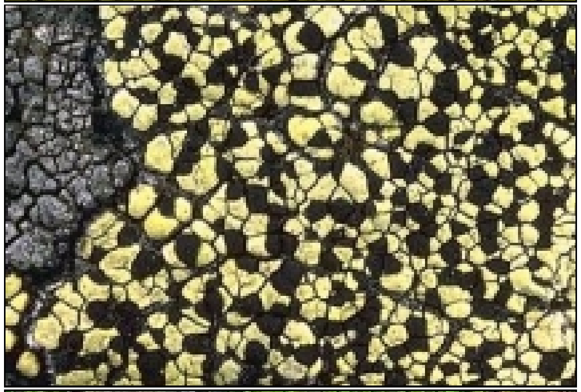
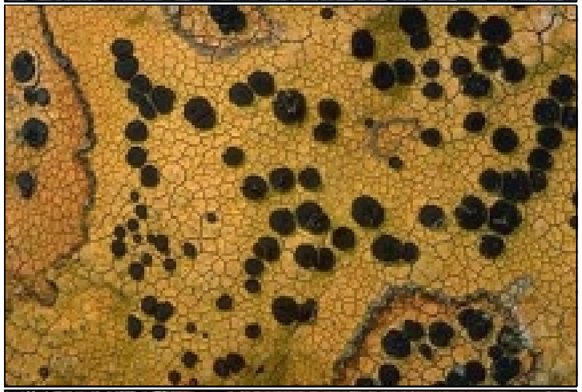
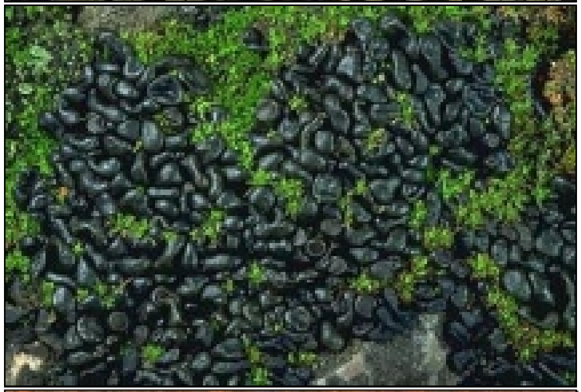
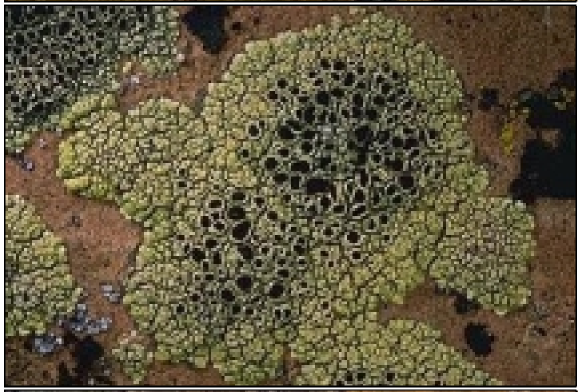
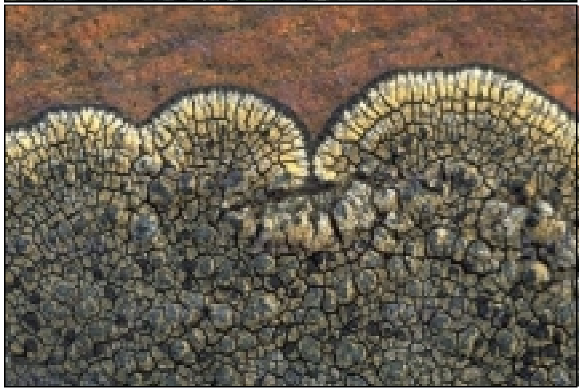
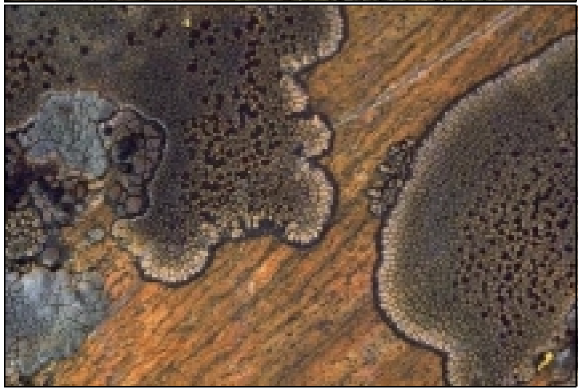
Antenna



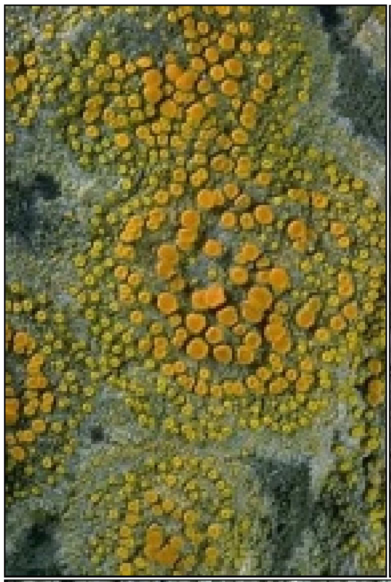
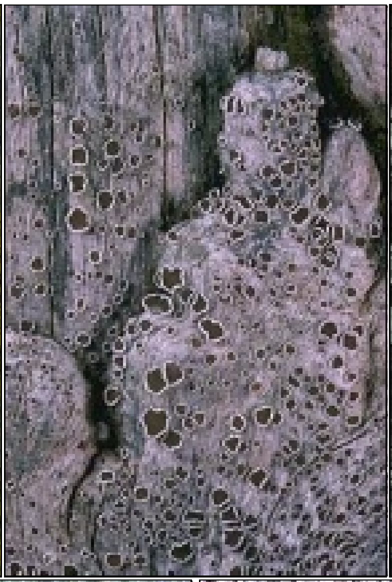
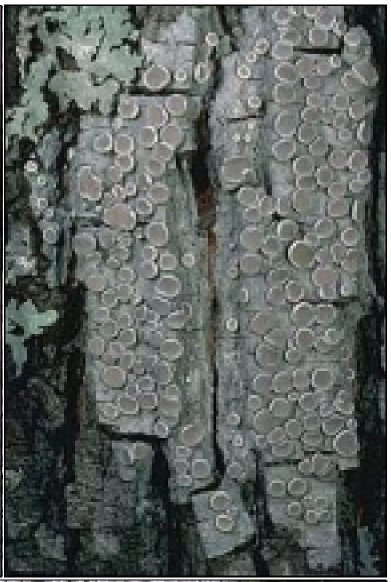
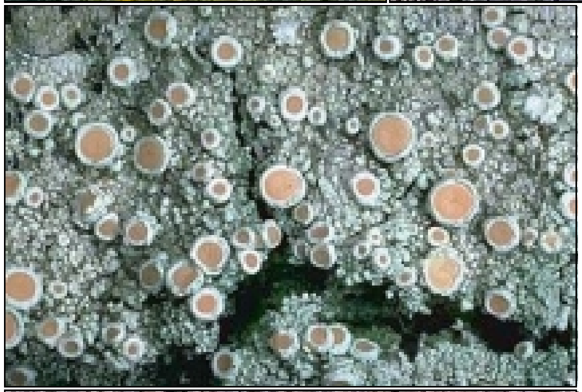
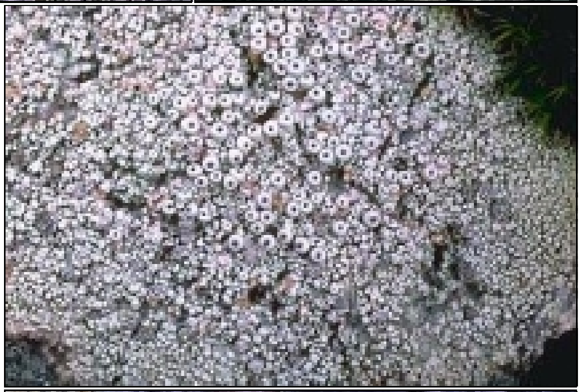
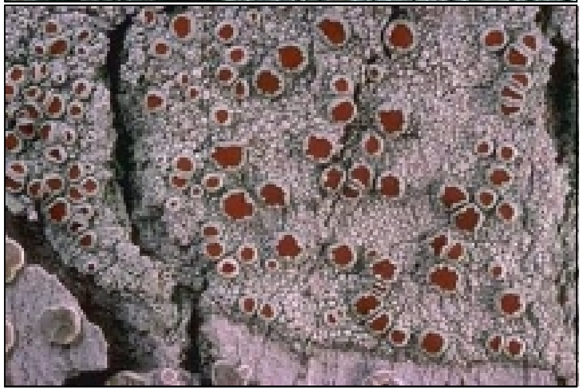
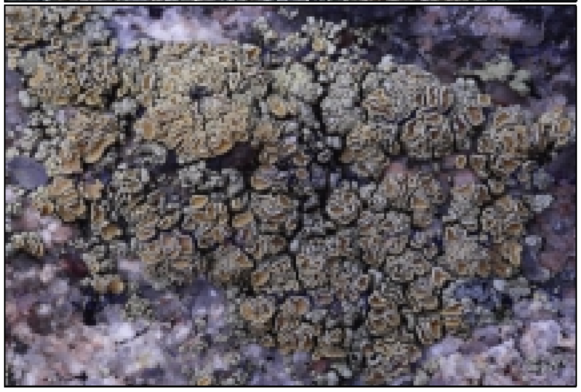
Loose 3D branches



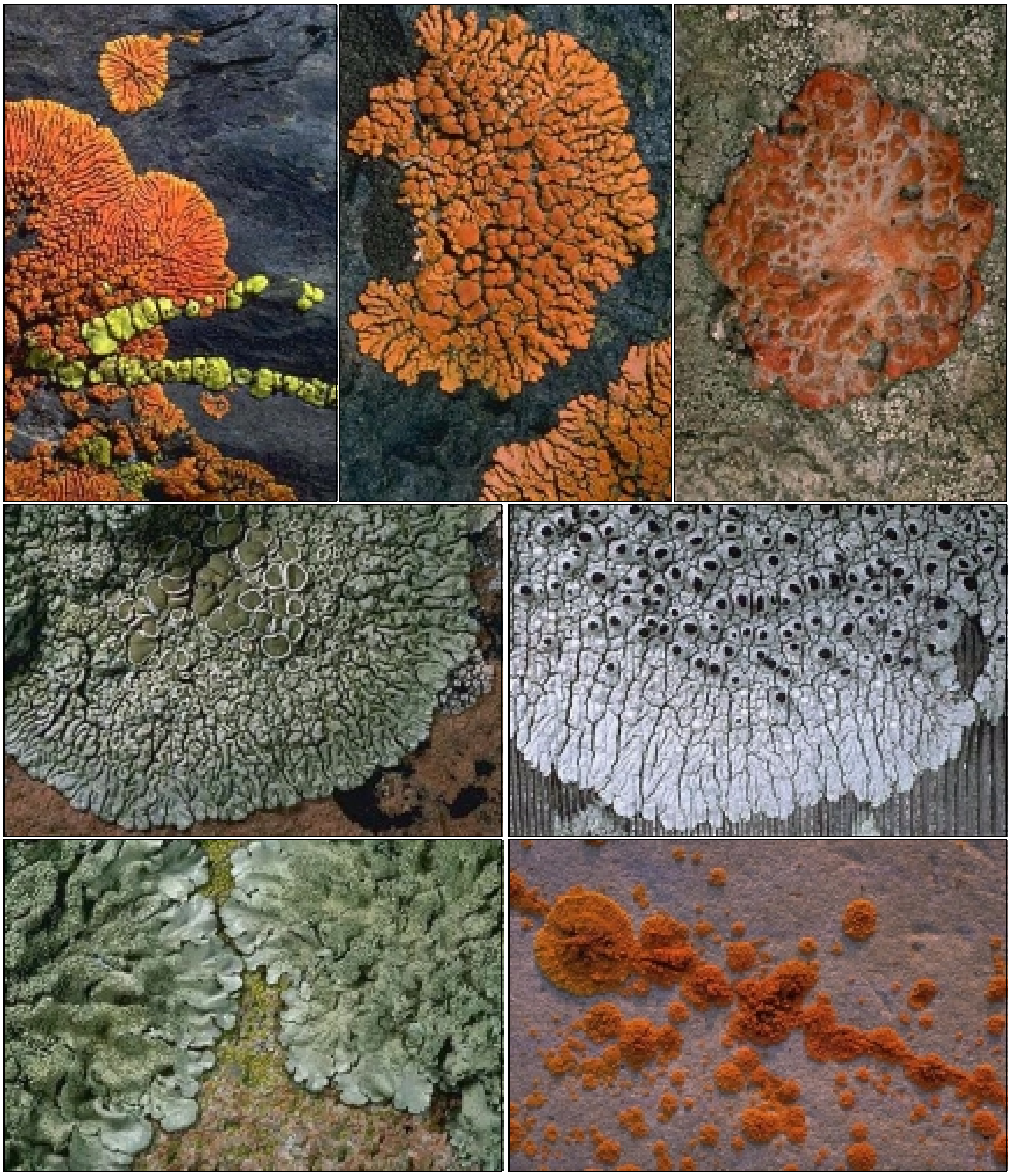
Crust-like

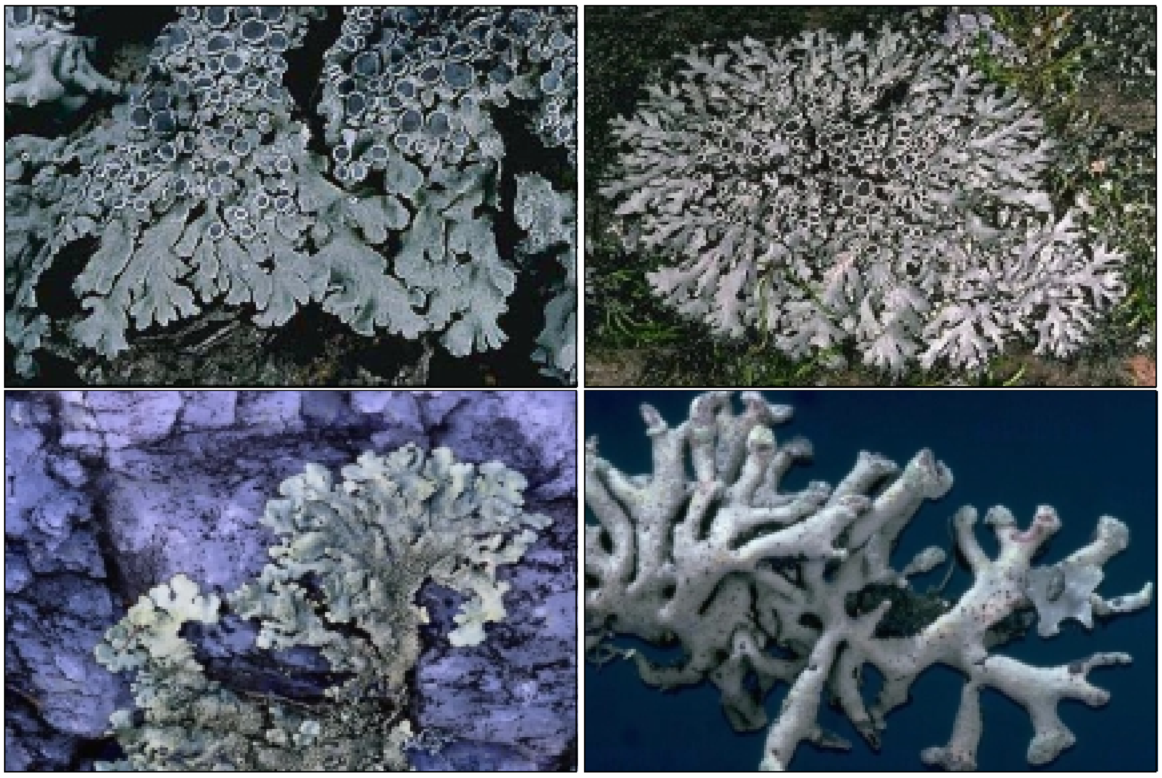
Circular fruits

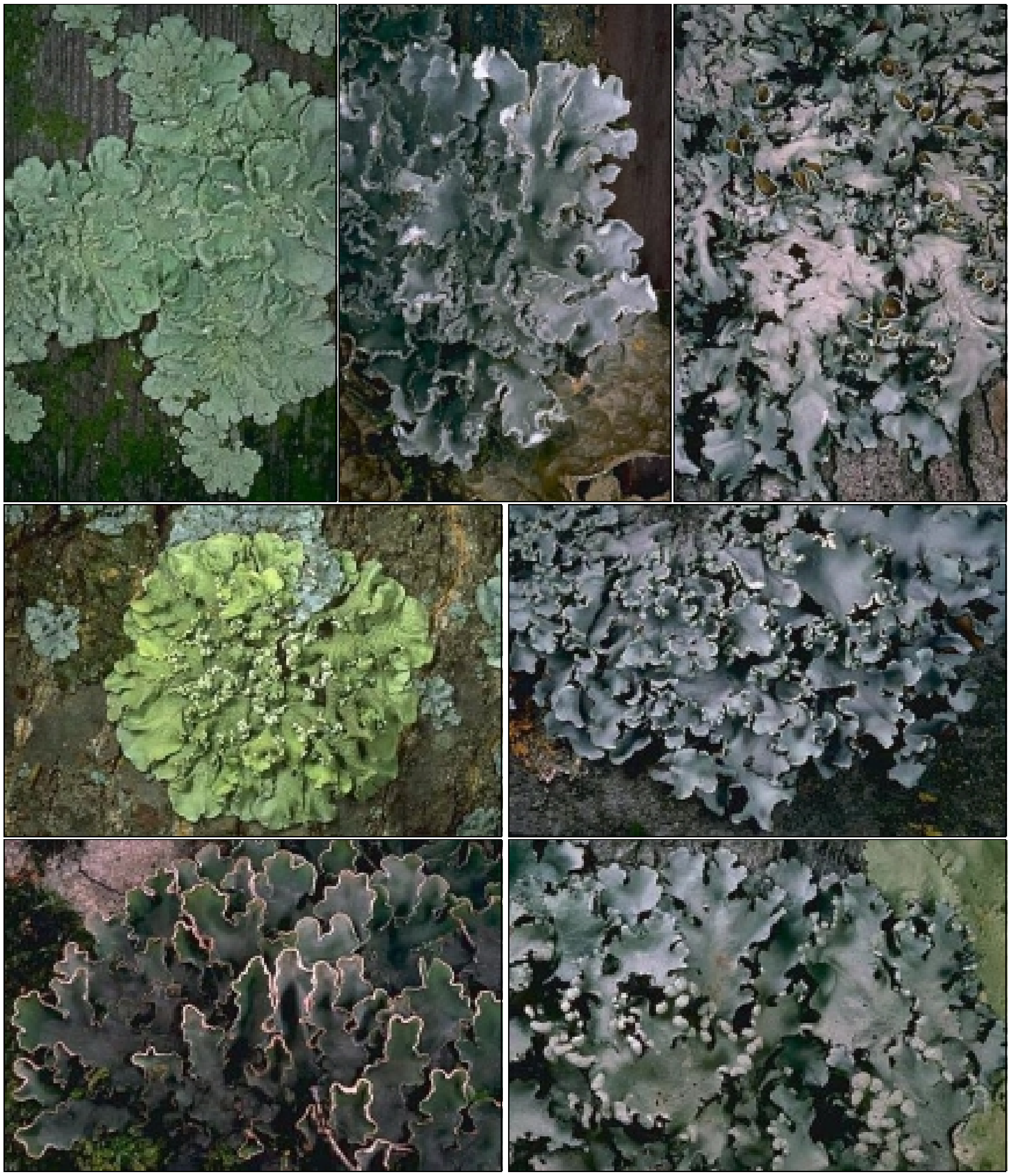
Dense branches close to surface



Lobes



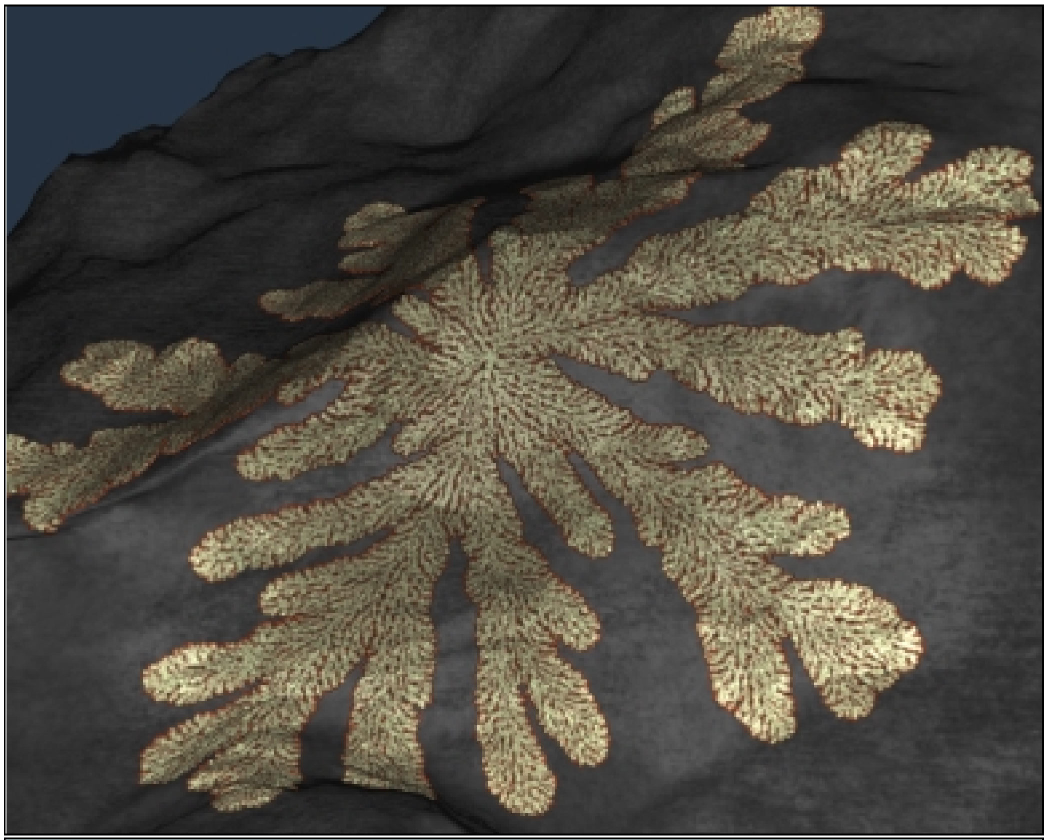
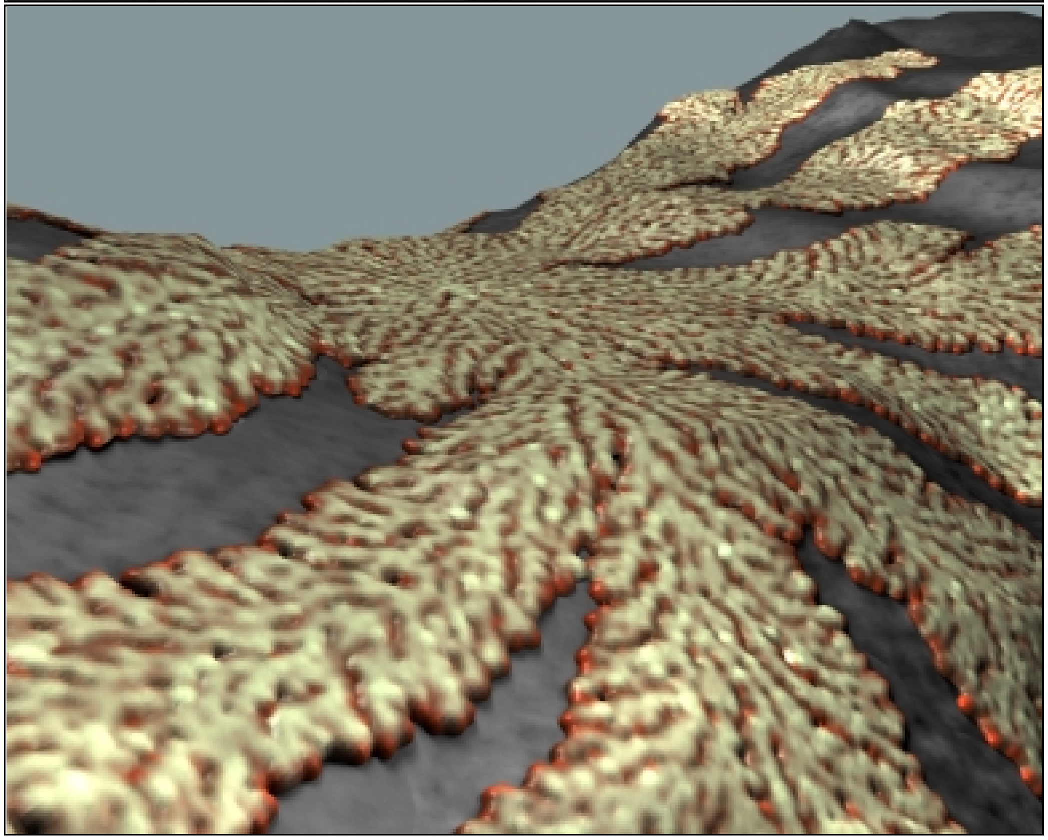
Lettuce

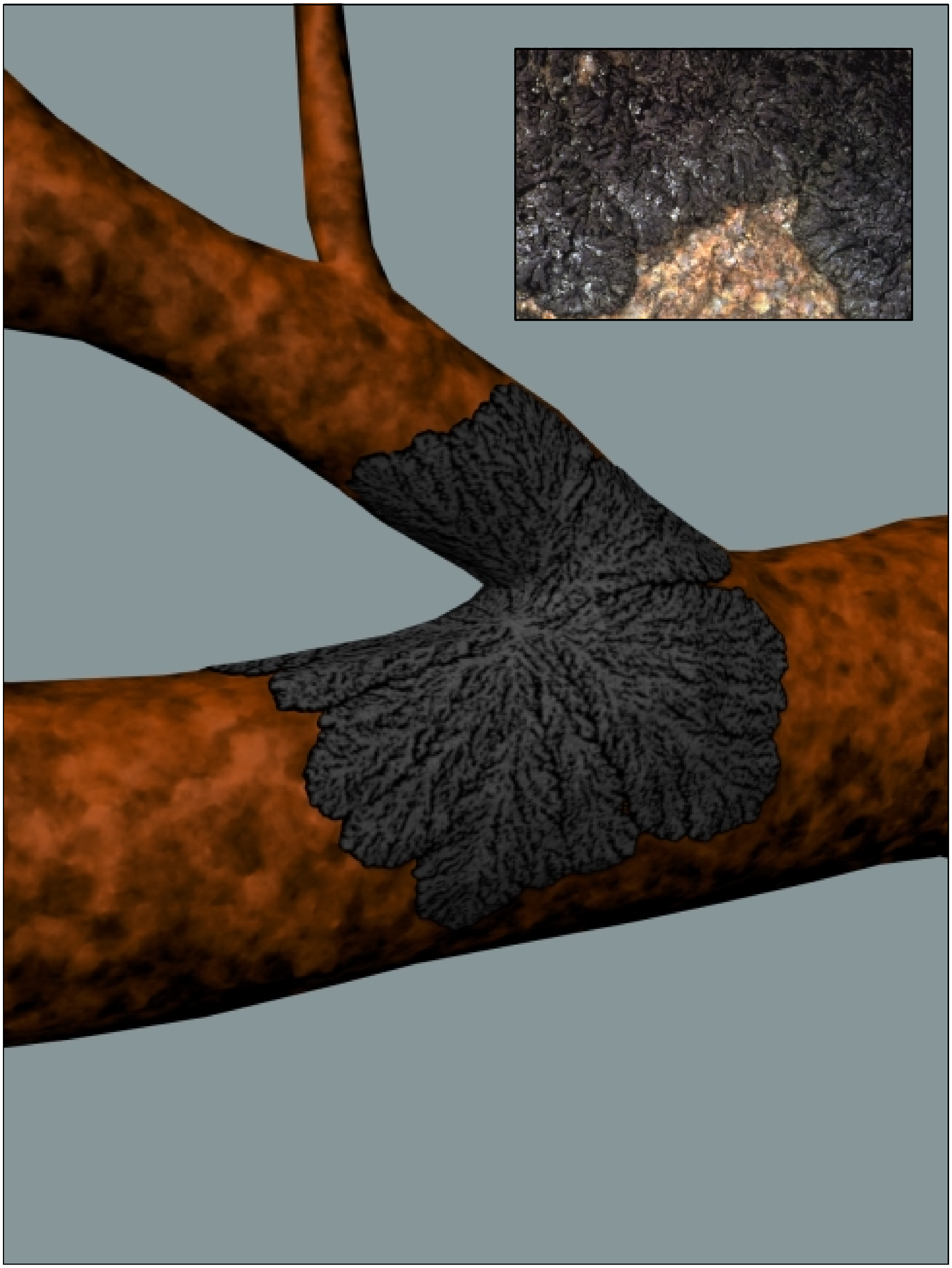


Scenes

APPENDIX B

COLOR PLATES



BIBLIOGRAPHY

[1] P. S. Aplin and D. J. Hill. Growth analysis of circular lichen thalli. *Journal of Theoretical Biology*, 78:347–363, 1979.

[2] R. A. Armstrong. Growth response of lichen thalli with central portions artificially removed. *Environmental and Experimental Botany*, 19:175–178, 1979.

[3] R. A. Armstrong. Experimental studies of lobe growth in the lichen Parmelia conspersa (Ehrh. ex Ach.) Ach. *New Phytologist*, 119(2):315–319, 1991.

[4] R. A. Armstrong. Lobe interactions within the thallus margin and the maintenance of symmetry in the lichen Parmelia conspersa (Ehrh. ex Ach.) Ach. *Symbiosis*, 18(2):129–142, 1995.

[5] James Arvo and David Kirk. Modeling plants with environment-sensitive automata. *Proceedings of Ausgraph 88*, pages 27–33, 1988.

[6] Philip Ball. *The Self-Made Tapestry: Pattern formation in nature*. Oxford University Press, 1999.

[7] Jules Bloomenthal. An implicit surface polygonizer. In Paul S. Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, 1994.

[8] Irwin Brodo, Sylvia Duran Sharnoff, and Stephen Sharnoff. North american lichen project. World Wide Web. `http://www.lichen.com/`.

[9] Richard C. Brower, David A. Kessler, Joel Koplik, and Herbert Levine. Geometrical approach to moving-interface dynamics. *Physical Review Letters*, 51(13):1111–1114, September 1983.

[10] Richard C. Brower, David A. Kessler, Joel Koplik, and Herbert Levine. Geometrical models of interface evolution. *Physical Review A*, 29(3):1335–1342, March 1984.

[11] S. Camazine. Designed by nature. *The world and I*, pages 202–208, March 1993.

[12] S. Childress and J. B. Keller. Lichen growth. *Journal of Theoretical Biology*, 32:157–165, 1980.

[13] J. N. Corbridge and W. A. Weber. *A Rocky Mountain Lichen Primer*. University Press of Colorado, Niwot, 1998.

[14] Daniel M. Dimian. A physically-based model of folded surfaces with an application to plant leaves. Master's thesis, University of Calgary, 1997.

[15] Kurt Fleischer and Alan Barr. A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In C. Langton, editor, *Artificial life III*, pages 389–416. Addison-Wesley, 1993.

[16] Kurt Fleischer, David Laidlaw, Bena Currin, and Alan Barr. Cellular texture generation. *Proceedings of SIGGRAPH 95*, pages 239–248, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.

[17] Deborah R. Fowler, Hans Meinhardt, and Przemyslaw Prusinkiewicz. Modeling seashells. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):379–388, July 1992.

[18] Ronald Goldman. Matrices and transformations. In Andrew S. Glassner, editor, *Graphics Gems*, pages 472–475. Academic Press, 1990.

[19] Ned Greene. Voxel space automata: Modeling with stochastic growth processes in voxel space. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):175–184, July 1989.

[20] M. E. Hale, Jr. Single-lobe growth-rate patterns in the lichen Parmelia caperata. *The Bryologist*, 73:72–81, 1970.

[21] W. S. Hall. *The Boundary Element Method*, chapter 2-3, pages 41–83. Kluwer Academic Publishers, 1993.

[22] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):289–298, August 1990.

[23] J. H. S. Hele-Shaw. The flow of water. *Nature*, 58(1489):34, 1898.

[24] D. J. Hill. The growth of lichens with special references to the modeling of circular thalli. *Lichenologist*, 13:265–287, 1981.

[25] D. J. Hill. Lobe growth in lichen thalli. *Symbiosis*, 12:43–55, 1992.

[26] T. N. Hooker. Lobe growth and marginal zonation in crustose lichens. *Lichenologist*, 12:313–323, 1980.

[27] Hugues Hoppe. Progressive meshes. *Proceedings of SIGGRAPH 96*, pages 99–108, August 1996.

[28] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. *Proceedings of SIGGRAPH 98*, pages 311–320, July 1998.

[29] Jaap A. Kaandorp. *Fractal Modeling: Growth and From in Biology*. Springer-Verlag, 1994.

[30] L. P. Kadanoff. Simulating hydrodynamics: a pedestrian model. *Journal of Statistical Physics*, 39(3–4):267–283, May 1985.

[31] David A. Kessler, Joel Koplik, and Herbert Levine. Geometrical models of interface evolution ii: Numerical simulation. *Physical Review A*, 30(6):3161–3174, December 1984.

[32] Shoudan Liang. Random-walk simulations of flow in Hele-Shaw cells. *Physical Review A*, 33(4), April 1986.

[33] B. McCune and L. Geiser. *Macrolichens of the Pacific Northwest*. Oregon State University Press/U.S.D.A. Forest Service, Corvallis, 1997.

[34] Paul Meakin, Fereydoon Family, and Tamás Vicsek. Viscous fingering simulated by off-lattice aggregation. *Journal of Colloid and Interface Science*, 117(2):394–399, June 1987.

[35] Radomír Mech and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. *Proceedings of SIGGRAPH 96*, pages 397–410, August 1996.

[36] Hans Meinhardt and M. Klinger. A model for pattern formation on the shells of molluscs. *Journal of Theoretical Biology*, 126:63–89, 1987.

[37] Hans Meinhardt, Przemyslaw Prusinkiewicz, and Deborah R. Fowler. *The Algorithmic Beauty of Sea Shells*. Springer, December 1995.

[38] W. W. Mullins and R. F. Sekerka. Morphological stability of a particle growing by diffusion or heat flow. *Journal of Applied Physics*, 34:323–329, 1963.

[39] Johann Nittmann, Gérard Daccord, and H. Eugene Stanley. Fractal growth of viscous fingers: quantitative characterization of a fluid instability phenomenon. *Nature*, 314:141–144, March 1985.

[40] C. Pozrikidis. *Introduction to theoretical and computational fluid dynamics*. Oxford University Press, 1997.

[41] M. C. F. Proctor. The growth curve of the crustose lichen *Buellia canescens* (Dicks.) de Not. *New Phytologist*, 79:659–663, 1977.

[42] Przemyslaw Prusinkiewicz. Visual models of morphogenesis. In Christopher G. Langton, editor, *Artificial Life: An Overview*, pages 61–74. MIT Press, 1995.

[43] Przemyslaw Prusinkiewicz, Mark S. Hammel, and Eric Mjolsness. Animation of plant development. *Proceedings of SIGGRAPH 93*, pages 351–360, August 1993.

[44] Przemyslaw Prusinkiewicz, Mark James, and Radomír Mech. Synthetic topiary. *Proceedings of SIGGRAPH 94*, pages 351–358, July 1994.

[45] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[46] P. G. Saffman and G. Taylor. The penetration of a fluid into a porous medium or Hele-Shaw cell containing a more viscous liquid. *Proceedings of the Royal Society of London, Series A*, 245:312–329, 1958.

[47] L. M. Sander, P. Ramanlal, and E. Ben-Jacob. Diffusion-limited aggregation as a deterministic growth process. *Physical Review A*, 32(5):3160–3163, November 1985.

[48] B. R. Speer and Ben Waggoner. Morphology of lichens. World Wide Web. `http://www.ucmp.berkeley.edu/fungi/lichens/lichenmm.html`.

[49] John C. Tannehill, Dale A. Anderson, and Richard H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*, chapter 6.10, pages 431–439. Taylor & Francis, second edition, 1997.

[50] P. B. Topham. Colonization, growth, succession and competition. In M. R. D. Seaward (ed.), editor, *Lichen Ecology*, pages 31–68. Academic Press, London, 1977.

[51] Alan Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, 237:37–72, August 1952.

[52] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):289–298, July 1991.

[53] Eric Veach and Leonidas J. Guibas. Metropolis light transport. *Proceedings of SIGGRAPH 97*, pages 65–76, August 1997.

[54] Andrew Witkin and Michael Kass. Reaction-diffusion textures. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):299–308, July 1991.

[55] T. Witten and L. Sander. Diffusion-limited aggregation, a kinetic critical phenomenon. *Physical Review Letters*, 47:1400–1403, 1981.

[56] T. Witten and L. Sander. Diffusion-limited aggregation. *Physical Review B*, 27:5686–5697, 1983.

[57] D. A. Young. A local activator-inhibitor model of vertebrate skin patterns. *Mathematical Biosciences*, 72:51–58, 1984.