A Simulator for the Topological Modeling of Ceramic Structures Using Local Rules

by

Vinay Pulim

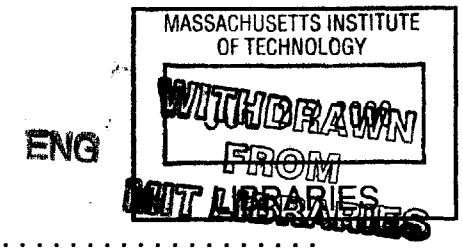
Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1999
[June 1999]



ENG

Author . . . . . .
Department of Electrical Engineering and Computer Science
January 14, 1999

Certified by. . . . . .
Bonnie Berger
Samuel A. Goldblith Associate Professor of Applied Mathematics
Thesis Supervisor

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

A Simulator for the Topological Modeling of Ceramic Structures Using Local Rules

by

Vinay Pulim

Submitted to the Department of Electrical Engineering and Computer Science
on January 14, 1999, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis paper details the development of a simulator for modeling the assembly of ceramic structures on a computer of workstation size. Determination of amorphizability of non-periodic structures is a difficult problem in materials science. In order to address this, a simulator was developed that utilizes a simple mass-spring model to represent structure, and from this derives topological properties useful in determining stability. Application of the local rules theory of Berger *et al.* allows for a significant increase in performance over slower molecular dynamics based methods.

The paper covers various aspects of the simulator's design and use. First, it gives background information about ceramic compounds and their topological representation, describes the theory of local rules, and explains how the theory motivates the development of the simulator. Then, it describes the constraints used in designing the simulator. Next, it details the implementation of the simulator itself including the user interface, its functionality, as well as the underlying algorithms. Following an evaluation of the simulator, the paper then provides several sample applications. Finally, it draws some conclusions about the simulator and discusses possible future improvements.

Thesis Supervisor: Bonnie Berger
Title: Samuel A. Goldblith Associate Professor of Applied Mathematics

Acknowledgments

Thanks to Bonnie Berger for taking me on as a UROP my sophomore year and getting me started on research at such an early age. Her willingness to provide both advice and financial support has made the last few years so painless.

Thanks to Esther Jesurum for being with me since the beginning of the glass project and being the best partner I could ever hope for. Without her intuitive grasp of both mathematics and materials science, the problems we faced could never have been solved.

Thanks to Linn Hobbs for all of his guidance and support during the development of the simulator and for constantly providing me with new programming challenges.

Thanks to Russell Schwartz for answering all of my questions about the virus shell simulator, and helping me to overcome so many obstacles during the design of the glass program.

Finally, I would also like to acknowledge all of the virus shell assembly code from [2, 21, 22] on which the glass simulator is based.

This thesis work contributed to several proceedings and articles: [17, 8, 14, 9, 15, 13, 16].

Contents

Chapter 1

Introduction

In recent years, as nuclear power grows in popularity, there has been an increased effort to find adequate materials for the long-term storage of nuclear waste. These materials have structures that allow them to remain stable even when exposed to disordering radiation. Ceramics such as periodic crystals and aperiodic glasses are excellent candidates and are being studied extensively. In the case of crystalline materials, stability can be measured by amorphizability — how easily a crystal is made aperiodic after exposure to radiation. Glasses, however, are by nature aperiodic, so their stability is more difficult to determine. Methods such as X-ray crystallography are not so informative for aperiodic materials. Instead, we take a more topological approach and examine the network of connections between the atoms comprising a glass. We will show how certain characteristics of these networks can provide much insight into physical properties of the compound.

Previous work involving hand modeling has provided valuable insights into aperiodic networks [20, 7] and proved the usefulness of a topological model. Topological analysis can predict amorphizability in crystalline networks as well. Dove *et al.* [4] studied what they refer to as "rigid unit modes" to differentiate between flexible and non-flexible topological networks, the latter being less stable. Jacobs and Thorpe [10] developed an algorithm to determine "floppy modes" in structures, areas that are easier to amorphize. Finally, Gupta and Cooper [6] developed a method known as elementary constraint counting that predicted relative amorphizabilities of crystalline

structures.

Recently, a theory of local rules was developed by Berger *et al.* [2] to help describe the assembly of icosahedral virus shells. It states that both periodic and aperiodic structures can be assembled using a set of rules that only describe how components interact with their local neighbors. It has already been used to model viral shell malformations and other aperiodic structures. Although it has been applied to the topologically "two-dimensional" shell surface, local rules theory can also be applied to the three-dimensional volume in which glass amorphization takes place.

This thesis paper describes the development and evaluation of a new local rules based simulator for the study of silica compounds. Chapter 2 provides background information necessary for understanding the motivation and theory behind the development of the simulator. Chapter 3 discusses design specifications for the functionality, performance, and portability of the simulator. Chapter 4 describes, in detail, how the simulator was implemented and discusses some of the algorithms involved. Chapter 5 evaluates the simulator and determines how well it has met the design specifications. Chapter 6 gives examples of some of the applications of the simulator. Finally, Chapter 7 draws several conclusions about the thesis project and suggests potential future improvements.

Chapter 2

Background and Definitions

This chapter describes the problem of determining the stability of ceramic compounds and how it motivates the development of a "local rules" based simulator. Section 2.1 describes the tetrahedral network representation of ceramic structures and how it leads to a topological model. Section 2.2 describes the local rules method of growing ceramic compounds. Section 2.3 provides the motivation for this thesis project.

2.1 Tetrahedral Networks

In order to develop a topological model of ceramic networks, we must examine their physical structure. Since much of this thesis deals with silica glasses we will focus our attention on silicon based compounds. In the case of SiO_2, Si_3N_4, and SiC crystals the X-Si-X bond angles (X=O, N, or C) are perfectly tetrahedral. Hence, we can describe our model as a network of vertex sharing tetrahedra. In SiO_2, an oxygen is shared between two tetrahedra at each vertex. Similarly, in Si_3N_4, a nitrogen is shared between three tetrahedra at each vertex, and in SiC, a carbon is shared between four tetrahedra. The connectivity of these networks is illustrated in Figure 2-1. This model is simpler than an atom based network and allows for faster growth simulation and topological analysis.

Several formal definitions are required in order to understand the topological properties of a network; we use those of Marians and Hobbs [18]. A *ring* is a connected

sequence of either edges or tetrahedra (nodes); formally, a ring in a network is a closed path where each node on the path appears exactly once. For vertex-sharing tetrahedra, rings through the edges of the tetrahedra and rings through the center of the tetrahedra (nodes) are topologically equivalent; the former must be defined to exclude the tetrahedra themselves (*i.e.*, no more than one edge of a tetrahedron can be part of a ring). A *local cluster* of a node n is the union of all nodes that are in some primitive ring (defined below) of node n. A *collision cascade* is the sequence of events including breaking bonds and rotating and translating nodes following a disordering event. A collision cascade computation terminates when the structure has once again reached its lowest possible energy state.

We use Marians' [20] definition of a *primitive ring* which states that a ring in a network is primitive if at least one of the two paths between any pair of nodes of the ring is a minimal path. This is effectively the same definition adopted by Goetzke and Klein [5], although their terminology differs from ours (they define as a "ring" what we call a primitive ring, and a "cycle" to be what we call a ring.)

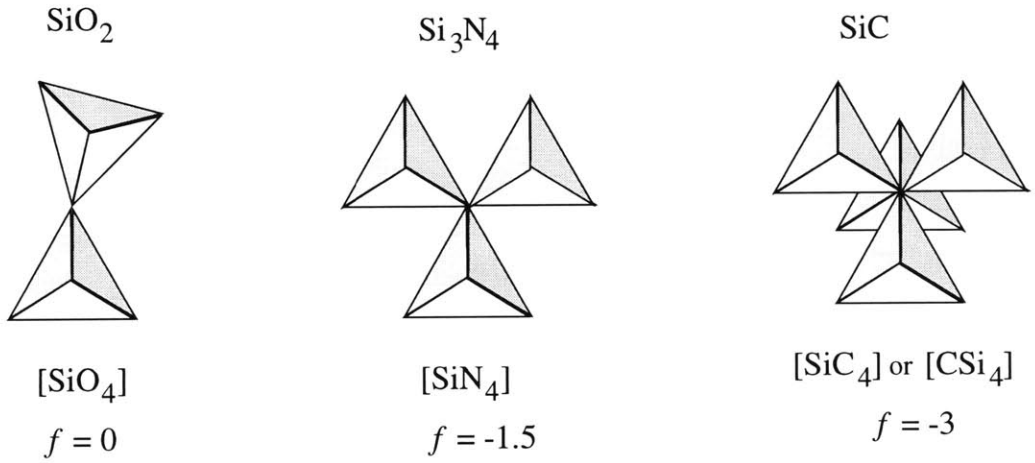


Figure 2-1: Vertex-sharing schemes for SiO_2, Si_3N_4 and SiC.

Once a network is generated one can easily derive its topological properties from the connectivities of the tetrahedra. The network may be described mathematically as a regular graph with tetrahedra represented by nodes and connections to other tetrahedra represented by edges. Periodic networks have a regularly repeating arrangement of nodes, whereas aperiodic networks do not have this property.

2.2 Local Rules

Before one can create a topological model for a compound, it is necessary to first simulate the growth of the physical model it derives from. Previously, researchers used molecular-dynamics simulations to create accurate models of silica compounds [24]. Although this approach yields good results, finding accurate model parameters is difficult, and the simulations are computationally expensive and time-consuming. Alternatively, we can apply local rules theory to this problem.

Local rules were first used to develop a language to help describe and simulate the assembly of icosahedral virus shells [2]. The study of virus shell structure and assembly is crucial for understanding how viruses reproduce and how anti-viral drugs might interfere with assembly of virus shells. One of the most notable aspects of virus shells is their highly regular structure: they are generally spherical and possess strong symmetry properties without being periodic; almost all human viruses (*e.g.*, rhinovirus, poliovirus, herpesvirus) and many plant and animal viruses are icosahedral. These icosahedral shells are constructed of repeated protein subunits—or coat proteins—which surround their condensed DNA or RNA genomes. A given shell usually consists of hundreds of copies of one protein, but sometimes copies of two or three different proteins.

At first glance, the assembly of the viral shells seems easy to understand because the structure is so regular. In fact, it has been difficult to determine the actual pathway through which the subunits interact to form a closed shell composed of hundreds of subunits; in icosahedral viruses this has been particularly difficult to explain because very often the same protein occurs in non-symmetric positions.

The theory of local rules can also explain several anomalies, such as the polyomavirus shell structure and spiraling malformations [1, 2]. It can also be used to model other malformations and possible aperiodic structures as a result of changing various model parameters [23].

Berger and Muir [21, 2] developed a "toolkit" to model virus shell assembly on a Silicon Graphics Indigo 2 computer. The toolkit has been used to explore the

tolerance margins of these shells and possible deformities that may result. The binding interactions and angles of the shell proteins can be varied up to 8% randomly in every direction and roughly the same shell is produced; however, if a hexagon occurs in place of a pentagon, then spiraling occurs. Schwartz and Berger [22] have also developed a parallel implementation on the Thinking Machines CM-5 computer that substantially speeds up the algorithm. The eventual goal of this work is to develop a computer simulation that allows biologists to view and interfere with the virus assembly process on screen, a process which, like restructuring following radiation-induced disorder, cannot be observed directly in nature.

Recently, Schwartz, *et al.* have built a kinetics based model of virus shell assembly [23]. The result was the development of a new simulator that allowed for more realistic modeling of viral shell assembly, providing a tractable middle ground between abstract local rules modelling and more low-level molecular dynamics modeling. It provided the user with more versatility in modeling the different variations on local rules and the way in which they were implemented. It allowed the user to specify many physical properties of the model such as the bond strengths and energies, as well as the size, shape, mass, and bond configuration of individual particles. In addition, the new simulator provided a model for conformational switching, allowing a protein to shift its configuration probabilistically.

The parallels to the problem of understanding the assembly of ceramic networks are strong, and the methodologies are already well explored. Several differences in the analogy, however, present concerns which must be addressed. First, viral shell assembly takes place on a topologically "two-dimensional" surface, whereas amorphization takes place within a three-dimensional volume. On the other hand, the surface is closed in three-dimensions and not initially constrained to be, and Marians [19] has shown the importance of two-dimensional subnetworks in the three-dimensional structure of network solids. Second, in the more abstract local rules simulation, energy minimization is accomplished with simple spring constant response. This may seem overly simplistic compared to the empirical potentials and three-body interactions employed in molecular-dynamics simulations (*e.g.* [24]), but it must be remembered

that such codes do not assume a priori information about coordination units which we know to exist in such assemblies. In fact, minimization of overall bond-length deviation proved a useful approach for the study of the results of bond-switching as an initial approach to structural collapse during amorphization of quartz [11].

2.3 Motivation

Earlier hand-assembly exercises involved constructing physical models [19, 18]. Much of this kind of modeling can now be done directly on a computer of workstation size, owing to advances in both computer hardware and fast computational algorithms. Because performing topological analysis by hand on the more complex structural modifications of silica is extremely tedious and difficult to verify, we needed to establish computer-based algorithms to erect models of each of the crystalline polymorphs so that analysis could be performed without error and with a significant increase in speed. The need for these algorithms motivated us to build a computer simulator that would complete results from earlier hand-counting of crystalline polymorphs [19] and allow us to explore the structural limits of these polymorphs.

Chapter 3

Design Specifications

This chapter describes the requirements the simulator must meet in order to be a useful tool for ceramic network assembly and stability analysis. Section 3.1 describes the functions the simulator must be able to execute. Section 3.2 describes performance requirements of the program. Finally, Section 3.3 describes portability requirements of the simulator.

3.1 Functionality

Given a set of local rules, the simulator must be able to assemble the ceramic network that derives from these rules. The simulator represents the network as a simple mass-spring model with polyhedral nodes as masses and internodal connections as springs. Starting from either an initial user-defined node or from a previously assembled structure, the simulator proceeds by adding one node at a time until a user-specified maximum number of nodes is reached. Ideally, nodes are added to the structure in an order that ensures symmetry to prevent the formation of unbalanced high-energy structures.

In addition, the simulator should be able to perform various statistical computations on an assembled structure. First, it must be able to compute primitive rings through a given node and from these compute a local cluster as defined in Section 4.5.2. Second, it must be able to calculate the density of the completed structure.

Third, it must be able to calculate the distribution of bond angles of both Si-X-Si and X-Si-X angles (X = O, C, or N). Finally, it must be able to calculate the distribution of distances between all pairs of atoms. These statistics are necessary to accurately determine the amorphizability of a compound.

Another requirement of the simulator is that it must have a user interface that is easy to use yet versatile enough to allow for a wide range of simulation parameters. This can be accomplished by ensuring that more commonly used functions are easily accessible, in the main menu for instance, leaving more esoteric functions in submenus. For instance, functions to load rules files and assemble structures should be easier to access than functions to adjust the maximum recursion depth for the ring finding algorithm. In addition, the simulator must allow the user to visualize assembled structures in a clear and natural three-dimensional graphics format.

3.2 Performance

Performance requirements must also be considered when designing the simulator. There is no set limit to how long the program can take to finish a simulation. However, the purpose of using local rules was to provide a much faster alternative to molecular dynamics based simulations. In order to be useful, our simulator must be significantly faster. More specifically, we should be able to assemble structures on the order of a thousand nodes per hour on a single workstation-sized machine.

In addition, we must ensure that an increase in speed is not offset by a significant decrease in accuracy. Local rules allow us to bypass many of the complex molecular dynamics computations in which error accumulation is an issue. Our simulator need only focus on correct application of local rules when deriving new nodes and accurate minimization of energy as the structure grows. The complexity of the numerical methods used will be determined by the speed requirements. Slower, simpler algorithms may suffice as long as accuracy is not compromised. Simpler time-stepping algorithms, for instance, may be used over more advanced forward and backward Euler methods if error does not increase between steps and if simulations still finish

within the required time. In addition, the simulator must ensure that basic physical laws are not violated. For instance, springs must not intersect and steric constraints must be enforced at tetrahedral corners.

Finally, the user interface and the graphics display should come second to the computation of a structure. The simulator should not check for user input or update the interface while a simulation is running so that every processor cycle can be dedicated to performing assembly calculations. For the same reason, the user should have the option of not updating graphics during assembly.

3.3 Portability

Another goal of the project is to make the simulator available to the widest audience possible. First, it should be platform independent. The code should compile without problems on any UNIX-based machine. Second, the program should be able to run simulations on machines without a three-dimensional graphics output device, i.e. on any machine besides the SGI. This can be accomplished by simply not displaying graphics on such machines and ensuring other aspects of the program still function correctly. Finally, in order to incorporate the program into a web server, it should support a command line interface and should produce output in the three-dimensional VRML format. Web browsers with VRML support can then view the output directly within the application.

Chapter 4

Implementation

This chapter describes the basic components of the simulator in detail and provides an overview of the major algorithms used.

4.1 Polyhedron Abstraction

The basic unit in every simulator model is the polyhedral node (also simply referred to as a node, and not to be confused with a vertex) which consists of an atom located at the center of a regular polyhedron and four atoms located at each of its vertices. See Figure 4-1. The parameters that can vary between models include the center atom type, the vertex atom type, the vertex degree (the number of other nodes that can connect at a vertex), and the polyhedron degree (the number of vertices in the polyhedron). In addition, for a given model the user can define multiple node types and how they attach to each other.

Each polyhedral node has an associated data structure containing information needed during a simulation. This includes the coordinates of the center of the node, vectors to each of its corners from the center, rotation vectors, its polyhedral degree and maximum number of neighbors, optimal bond angles with each of its neighbors, and pointers to each of its neighbors. Much of this information is derived from the local rules detailed in the next section, and it is used for breadth first growth as discussed in Section 4.5.1.

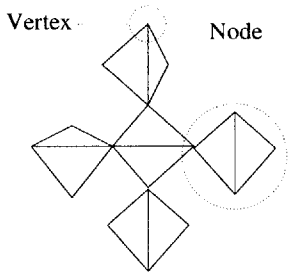


Figure 4-1: Definition of vertex and node.

4.2 Local Rules

In this section we show how we have adopted the local rules-based approach discussed in Section 2.2 for rapid self-assembly of crystal structures and adapted many of the codes developed for self-assembly of virus shells, using tetrahedra as the building blocks instead of protein units, with several modifications. Whereas virus shell formation requires that proteins form a basically spherical shell (with topologically two-dimensional rules governing assembly), we have to provide rules governing connections in three dimensions. On the other hand, the surface is closed in three dimensions and does not have to be initially constrained to be so. Unlike the bounded virus shell which is all surface, topologically three-dimensional structure models need to be large enough that surface properties do not dominate.

We first explain how to generate a set of rules governing the self-assembly of a tetrahedral crystalline network and how simulated growth of the crystal is effected using these rules. The approach is applicable to other network structures, with coordination polyhedra other than tetrahedra, but we will limit ourselves to tetrahedral networks with no more than three neighbors at each corner. The rules must provide instructions of the following sort:

- **Number of types.** This instruction informs the simulator how many types of inequivalent tetrahedra exist; each type, while still geometrically identical to all other tetrahedra, follows its own set of rules. It is sometimes convenient to identify more than one of a type when, in fact, all tetrahedra are topologically identical; for example when chirality is present or because it is easier to

conceptualize the self-assembly with more types.

- **Connectivity.** This instruction defines the type of polyhedron in terms of the number of vertices (in this case "4" for a tetrahedron) and the number of additional polyhedra connected per vertex (in this case "1", "2", or "3" for O, N, and C respectively).

- **Orientation of the initial node.** We define arbitrarily a canonical orientation of the regular tetrahedron, inscribed in a unit cube centered at the origin, with vertices designated 0, 1, 2, 3 at the four cube corners defined by Cartesian coordinate vectors [-0.5, -0.5, 0.5], [-0.5, 0.5, -0.5], [0.5, 0.5, 0.5] and [0.5, -0.5, -0.5]. A rotation is applied to this tetrahedron to define the orientation of the initial node, defined by rotations about the x, y, z axes of the unit cube.

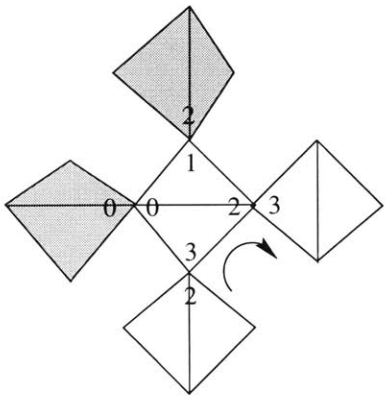
- **Rules for each vertex.** For a given vertex, this instruction informs the simulator about the relative orientation and type of the neighboring tetrahedra to which this vertex connects, as well as the vertex designation of those neighbors whose vertices it then shares.

In summary, given an initial tetrahedron, we need to know the rules governing the interactions at each vertex. With vertex-sharing silicas, for instance, the rules for each vertex will dictate the position of one neighboring tetrahedron.

When adding new nodes, the transformation matrix associated with a given node's neighbor is applied to a copy of that node to rotate and translate it into the correct position. In a sense, a clone is made of the parent node and then transformed relative to the parent. In addition, three dimensional arrays also store information about a neighbors' node types and corner numbers. The local rules also indirectly specify the optimal bond angle and torque angle between a given node and each of its neighbors. The simulator calculates these angles by actually growing a temporary structure with a root node of the current node type and then adding each of the neighbors while measuring the resulting bond angles.

Local rules are also used during growth to attach pre-existing nodes together. When two nodes are within a sufficient distance from each other, the simulator checks

their types and determines whether or not they can attach to each other under their local rules. In addition, during optimization steps, nodes are transformed relative to each other to insure the bond angles between them are as close to the optimal bond angles as possible.




```
Keatite
3
4 1                    0
REGULAR
z 45 y -24 ;
1 0  z 180 ;
1 2  z 90 ;
0 3  z 90 ;
0 2  z -90 ;
```

```
4 1                    1
REGULAR
x -20 z 45 ;
0 0  z 180 ;
2 0  y 180 z 90 ;
0 1  z -90 ;
2 1  y 180 ;
```

```
4 1                    2
REGULAR
z 45 y -24 ;
1 1  z -90 y 180 ;
1 3  y 180 ;
2 3  z 90 ;
2 2  z -90 ;
```

Figure 4-2: Local rules for assembly of keatite, using three node types corresponding to two inequivalent tetrahedron environments (shaded, unshaded). The arrow represents a four-fold spiral out of the plane of the illustration.

Figure 4-2 demonstrates how rules are derived for keatite and shows excerpts from its corresponding rules file. A more detailed description of the format of rules files is given in Section 4.4.1.

4.3 The Graphical User Interface

4.3.1 Buttons

The user interacts with the simulator through a graphical user interface. The main window is shown in Figure 4-3. It consists of a rules box and twenty buttons which

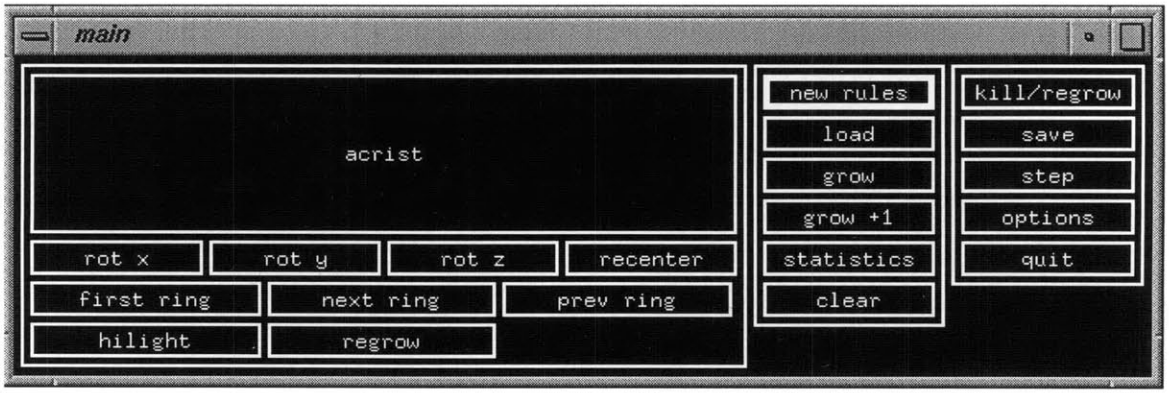


Figure 4-3: Main window of the graphical user interface.

control the simulation. The function of each of the components is explained in this section.

The rules box simply displays the name of the primary rules set being used in the current simulation. Any operations performed that do not ask the user to specify a rules set will use the ones displayed in the rules box.

The first row of buttons beneath the rules box are used to control the user's viewpoint within the graphics display. The "rot x", "rot y", and "rot z" buttons rotate the viewpoint around each of the three coordinate axes. In addition, the "recenter" button is used to return the viewpoint to the default position along the positive y-axis with the positive z-axis pointing upwards and the positive x-axis pointing to the left.

The next row contains the ring viewing buttons which are only useful after a local cluster has been computed. The "first ring" button displays the first ring in the local cluster, and the "next ring" and "prev ring" buttons cycle through the rest of the rings.

The bottom row contains the "highlight" button, which displays the current structure with underbound nodes highlighted in green, as well as the "regrow" button which draws only the regrow region on the graphics displays after a collision cascade has been computed.

The "new rules" button allows the user to read in a new set of rules. These rules will affect only new nodes added to the existing structure and will not change the rules associated with nodes that are already present. This is useful for the simulation

of collision cascades which require two sets of rules, one for the original structure and another for the regrow region. The simulator searches for rules files in the current directory as well as the "rules" subdirectory.

The "save" button writes the contents of the current simulation to a data file, including the simulations parameters, the location and states of all of the nodes, their corresponding adjacency matrix, and the rules associated with each node. The button first prompts the user for a file name before continuing. The "load" button reads in a data file and updates the graphics display to show the newly loaded simulation. By default, the simulator saves and loads files from a subdirectory with the same name as the currently loaded rules set.

The "grow" button adds nodes to the simulation using the currently loaded rules set. It first prompts the user for the number of nodes to add before continuing. The "grow +1" button is very similar to the "grow" button except that it does not prompt the user and it adds only one node to the simulation. This is useful for a step-by-step visualization of a simulation.

The "kill/regrow" button begins a collision cascade simulation. First, the user is prompted for the size of the regrow region. Then, the nodes in the regrow region are disturbed randomly before being allowed to reattach. These nodes are subjected to the last rules set loaded before the "kill/regrow" button was pressed.

Figure 4-4 shows the sub-window of the "options" button. This sub-window contains various simulation parameters that the user can change. These include the turning on or off of graphics, optimization, spinodal decomposition during regrowth, and VRML output. In addition, the user can set various constants such as the spring constant, depth of search for rings, optimization threshold, radial distribution function (RDF) radius, bond lengths, and molar masses.

The "statistics" button brings up a sub-window listing various operations that can be performed on the simulation. The "node number" button will create a local cluster for the node entered into the text box. The "sample size" button will create local clusters for a random sampling of nodes with the sample size specified in the text box. Furthermore, the "distances" button will calculate RDF distances for a random

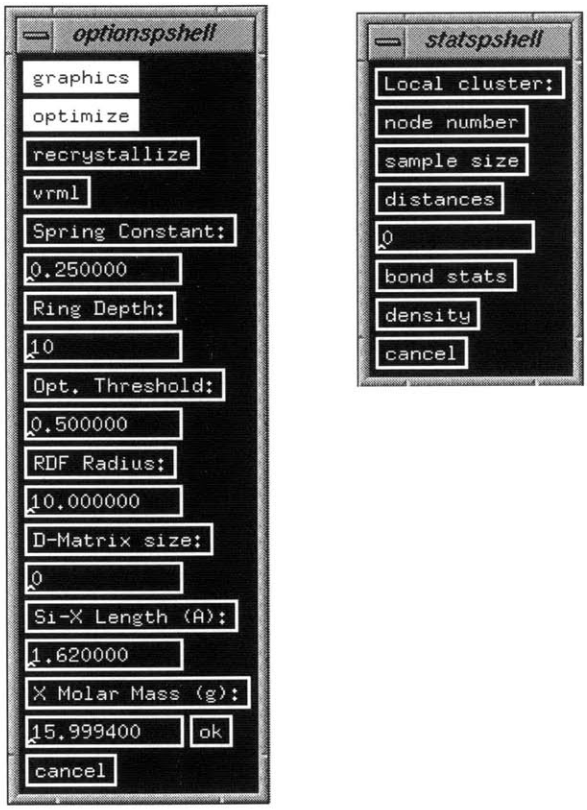


Figure 4-4: The options (left) and stats (right) sub-windows.

sampling of nodes. The "bond stats" button will calculate a variety of statistics such as the bond angle distribution, bond lengths, and the number of underbound nodes. Finally, the "density" button will calculate the approximate density of the current structure.

The last two buttons, "clear" and "quit" perform the obvious operations of destroying the current structure and exiting the program.

4.3.2 Graphics Display

The graphics display provides the user with a three-dimensional visual representation of the simulation. Each node is represented by a shaded yellow polyhedron, except for the root node which is shaded red and underbound nodes which are shaded green. However, during local cluster ring displays, the color scheme changes slightly. Nodes in the local cluster will be shaded light blue and the currently highlighted ring will

be dark blue. In every display, springs which connect polyhedra at their corners will always be represented by red lines.

The graphics in the simulator are produced using the OpenGL graphics library. OpenGL is available on a variety of platforms and allows for very portable code without a loss in graphics quality. In addition, the simulator allows the user to disable graphics so that it is still possible to run simulations on platforms that lack the proper support.

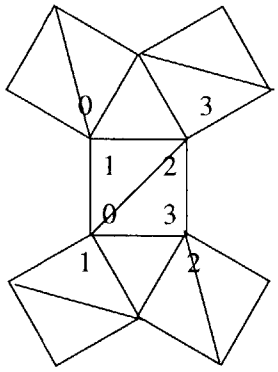
Although the graphics display is very useful for visualizing simulations, it does act as a bottleneck during the simulation, especially when one is dealing with large structures. The simulator may have to wait for the graphics to be displayed before continuing calculations. To solve this problem, the user can disable the graphics display before a simulation is run and then re-enable the display once it is complete.

4.4 Data Files

The simulator uses three types of data files: rules files, grow files, and VRML files. Rules files define each node type, describe how they interact, and set up an initial node. Grow files contain the entire state of a simulation and can be used to interrupt a simulation and resume it at a later time. VRML files are similar to grow files except they only contain enough data to reconstruct a graphical representation of the simulation, and they are only readable by VRML browsers and OpenInventor compatible viewers. Data such as simulation parameters, adjacency information, and rules are not saved with VRML files. This section describes each file type and provides an overview of their implementation.

4.4.1 Rules Files

Figure 4-5 illustrates a rules file for growing idealized quartz. The first line tells the simulator how many node types there are, where each node-type represents locally-identical tetrahedra (in this case, there is only one type, type "0", because every tetrahedron in quartz follows identical rules). The second line gives the basic network




```
Quartz
1
4 1
REGULAR
;
0 1   z 60 ;
0 0   z -60 ;
0 3   z 60 ;
0 2   z -60 ;
```

θ = 155.6°

Figure 4-5: Local rules for assembly of idealized quartz, depicting the canonical orientation of the initial tetrahedron and the assembly scheme for first five tetrahedra,with x-axis horizontal, y-axis vertical, z-axis orthogonal to the plane of the illustration.

connectivity; "4 1" signifies a structure in which each subunit has four vertices and each vertex seeks one neighbor (equivalently, two tetrahedra meet at each vertex). The third line indicates whether the polyhedron is regular (if it is not, we must additionally specify the initial coordinates of its vertices). The fourth line specifies any rotation applied to the initial canonical node resolved into rotations about x, y and z (in this case, there are none). The remaining lines specify rules for each vertex. The first of these provides rules for vertex 0 (in this case, vertex 0 is connected to a tetrahedron of type 0 at its vertex 1). Then the rotation applied to that neighboring tetrahedron is explicitly given (in the example, vertex 0 is connected to vertex 1 rotated 60° around z).

4.4.2 Grow Files

Grow files are binary files and are, therefore, not directly readable by a text viewer. This is done in order to conserve disk space and to allow for faster saving and loading of files. The first element of a grow file is the version number of the simulator used to create it. This information is used by the simulator to insure that the grow file is compatible with the current version. The version number is followed by information about the nodes, such as the number of node types, the polyhedron degree of each type of node, the number of neighbors at each corner, and the number of nodes. Next

is a listing of every node and all of the information associated with each one. This includes such information as the node's location and the rules associated with the node. After this, the grow file contains a listing of the adjacency information which specifies how each node connects to its neighbors. This is followed by the listing of the kill region nodes, if any, and finally the pre-computed distance matrix, if it exists.

4.4.3 VRML Files

VRML files provide an easy way to export the graphics displayed by the simulator to other viewers. They contain VRML code that describes the location and orientation of each node in the simulation and no other information. The VRML files can be read by any VRML 1.0 compatible viewer, which includes all VRML browsers as well as OpenInventor compatible viewers. To create VRML output files, the VRML option must be checked in the options menu.

4.5 Major Algorithms

4.5.1 Breadth First Growth

New nodes are added to the structure in breadth first order. That is, the corners of older nodes are filled before corners of the newer nodes, similar to the way a breadth first search of a tree checks all of the nodes on one level before moving down to the next level. For instance, assume the simulator is currently working on the parent node A, which has four unfilled corners. It would first try to fill all of node A's corners before attempting to fill the corners of any of its children. This breadth first order insures that the growing structure is well balanced and symmetric, and it models how glass is believed to grow in nature.

The breadth first growth is actually implemented by a queue which initially contains just the root node. This node is then removed and each of its corners are checked to see if they can attach to nearby nodes. Since this is the first node, it will not have any neighbors, thus new nodes are added to each of its corners. These new

nodes are then added to the end of the queue. The first node in the queue is then removed and each of its corners are checked for neighbors. If none exist, then nodes are added to each of its corners. These new nodes are then added to the end of the queue, and the next node is removed form the head of the queue. This process is continued until either the queue is empty or the maximum number of nodes has been reached. During the processing of nodes, some problems may arise. For instance, no neighboring nodes may exist near an unfilled corner of a node, and there may not enough room to add a new node. In this case, nearby bonds may have to be broken and reattached to accommodate the new node. A detailed explanation of the breadth first growth algorithm is given here in pseudocode:

```
queue is initialized to empty
if no nodes exist in the structure
  add new node to queue
else
  for each node i in the structure
    if node i has an unfilled slot
      add node i to queue
    end
  end
end
while queue is not empty
  remove node i from the head of the queue
  for each corner j of node i
    for each slot k of corner j
      if k is empty
        try to attach a nearby node
        if no nearby node could be attached
          add a new node to node i at slot k
          if successfull
            add the new node to tail of queue
```

```
            else
              try to break and reattach to a nearby node
            end
          end
          optimize the structure
        end
      end
    end
  end
```

4.5.2 Ring Finding and Local Clusters

As stated earlier, we use Marians' [20] definition of a primitive ring which states that a ring in a network is primitive if at least one of the two paths between any pair of nodes of the ring is a minimal path. This definition yields directly the following simple approach to finding the local cluster. First, we enumerate all rings which include n. This enumeration can be achieved by conducting a breadth-first search (enumeration of closest nodes first) from node n and finding all simple paths terminating at node n, where a simple path has no node appearing more than once. We confirm that a ring is primitive as follows: For each pair of nodes on the ring, we ascertain that the graph distance between them is equal to the minimum ring distance between the two nodes (there are two paths between each pair of nodes on a ring). By eliminating all non-primitive rings, what remains is the local cluster for node n.

Goetzke & Klein [5] suggest an efficient algorithm for determining all the (primitive) rings in a network, which proceeds as follows:

1. Compute a distance matrix such that the graph distance between nodes i and j appears in row i, column j. They suggest computing these distances by computing all pairs of distance 1 (first neighbors), then all pairs of distance 2, then all pairs with distance 3, and so on until all possible pairs are analyzed.

2. Fix an edge e and perform step 3 (below). Remove e from the graph. If fewer than three edges remain, finish. Otherwise, examine the distances in the new graph; if a distance between the two nodes is greater than their distance in the previous graph, change the respective value in the distance matrix to the value "infinite." Repeat step 2.

3. Determine all rings containing e. Start with rings of length 3 and stop if length $(2d + 2)$ has been reached (where d is the maximum distance in the distance matrix, also called the diameter of the graph). For determination of rings with length λ, the following procedure is used: Mark one of the two nodes of the fixed edge e as a starting point; choose an arbitrary node not yet considered with distance $\lambda/2$ to the starting point. Try to find two distinct paths between the chosen node and the starting-point, alternating step-by-step from one path to the other.

Since we would rather concentrate on what Goetzke & Klein refer to as "rings of interest," we let the user define the maximum length of a ring to search for (where maximum length < 20 appears reasonable). Also, instead of computing all the rings in the entire structure, we simply compute the local clusters for individual nodes (in other words, we ignore step 2 and do only step 3). Our algorithm then proceeds as follows:

1. Define $N_i(n)$ to be the number of nodes of graph distance i to node n. For a chosen n, compute an all-pairs distance matrix of size $D \times D$, where $D = \sum_{i=1...d} N_i(n)$ and d is half the user-specified maximum ring length. The $D \times D$ matrix includes all-pair distances for nodes within distance d of node n. We may use an all-pairs shortest path algorithm of our choice; we chose a variation of Dijkstra's algorithm for simplicity [3].

2. Find a ring containing n. This is done by performing a breadth-first search from n back to n; if the search is longer than the longest ring of interest, start a new search. Ascertain that this ring is primitive by comparing the shortest

ring distance between each pair of nodes to their actual shortest distance as computed in Step 1. If it is primitive, enqueue the ring.

In Step 1, we have generally restricted our "rings of interest" to length 15, which means $d = 8$, so this step is rather fast. In our Step 2, enumerating all rings is related to the square of the size of the local neighborhood D (sum of all the nearest neighbors up to N_d). Eliminating rings that are not primitive takes time related to the square of the length of the ring (to look up the distances of all pairs on the rings).

In Step 3 of Goetzke & Klein's algorithm, they use a time-saving technique. They offer that enqueuing all the rings and then dequeuing all that are not primitive is too expensive, so they look for primitive rings as they go. Whereas, the technique they use certainly finds only primitive rings, it may fail to find all primitive rings. The discrepancy arises because, in considering a ring of length λ, they find the "chosen point" at distance $\lambda/2$ from the starting point. They then find two non-intersecting paths of length $\lambda/2$. There is, however, allowance in the definition of a primitive ring for one path on the ring to be larger than the shortest path distance. It is not clear how their procedure will find these rings as well (for example, for another chosen edge e in a future iteration of their Step 3), whereas ours does because we enumerate all rings for each node in turn.

4.5.3 Local Distance Matrices

We refer to the variable sized all-pairs distance matrix described in Step 1 above as a local distance matrix (LDM). It is very similar to a normal distance matrix except in the way that it is calculated and in the amount of information that it contains. An LDM only contains enough information to calculate a local cluster for a single test node. More specifically, it contains the distances between the test node and all other nodes that are reachable through a breadth first search to a user specified depth. Since every primitive ring through the test node is a subset of this set, no more information is needed than is contained within an LDM. The decrease in execution time when using LDMs in place of a global distance matrix is significant and allows

for a constant order of growth in time with respect to the size of the structure.

4.5.4 Density Calculations

The simulator uses two methods for computing density. The first method simply samples several spherical regions within a structure and averages the densities computed within each one. This method is very fast without much loss in accuracy. The second method computes a convex hull around the entire structure and then computes the enclosed volume and mass, and from this the density. This method, although slower and less accurate, is useful for computing the density of an entire structure (not just regions within it). The method is less accurate because it is extremely difficult to compute the convex hull of extremely porous structures.

4.5.5 Optimization

The simulator uses a simple spring model to represent the forces between nodes. Each node is connected to each of its neighbors with a single spring. During each step of optimization the springs are relaxed and allowed to exert various forces on the nodes. The basic forces are translational forces, rotational forces, bending forces, and twisting forces. The virus shell assembly simulator, which essentially modeled networks in two dimensions, required fewer springs to model the various interactions between nodes. For instance, nodes in the virus shell simulator were modeled as spheres without any associated orientation information; whereas, in the glass simulator, nodes are represented as three-dimensional tetrahedra that are rotated around the x-, y-, and z-axis by a fixed amount. Additional rotational forces are required to ensure that nodes are properly oriented. The translational force exerted by a spring pulls the center of mass of a polyhedral node along the spring's axis. The translational forces applied to each of the node's corners are averaged and applied to the center of the node, changing only its position but not affecting its orientation. The rotational force exerted by a spring rotates the node around its center allowing its corner to get closer to its neighbor. These forces are also averaged and rotate the node accordingly.

The bending forces do not result directly from the springs. Rather, they are forces created to push bond angles between adjacent nodes closer to the optimal bond angle determined by the rules. The bending force can be thought of as a force exerted by a bent spring that is trying to return to its original low energy state. Finally, the twisting forces are forces that attempt to push the torque between two nodes connected by a spring to an optimal torque angle that has already been computed from the rules.

Since the polyhedral nodes are abstractions of physical structures they are subject to several constraints. Most importantly, the polyhedron represented by an individual node must remain rigid during the course of a simulation. In addition, the angle formed by two neighboring polyhedra is limited by both a minimum and a maximum value, since bonds cannot bend past certain thresholds. Finally, due to steric constraints polyhedra cannot overlap or intersect each other. The simulator ensures that none of these constraints are violated during the course of an experiment.

As mentioned earlier in Section 4.5.1, during growth there is sometimes not enough room for a new node to be inserted. In this case, nodes near the current parent node must be broken apart and reattached to either the parent node or other nearby nodes. This procedure involves searching for a pair of nodes whose point of attachment is closest to the parent node's current corner. These nodes are then broken apart and one of them is reattached to this corner. The other node can either attach to another corner of the parent node or it can rejoin the breadth first growth queue and later attach to another free node. This breaking and reattaching greatly reduces the number of underbound nodes when growing an irregular glass structure and during collision cascades. The optimization algorithm is summarized here in pseudocode:

```
while total spring energy > threshold
  for each node i in structure
    for each corner j of node i
      for each slot k of corner j
        calculate forces on slot k of corner j of node i
        apply forces to node i and move it accordingly
```

```
        place node i in the correct bucket
      end
    end
  end
  calculate total spring energy of structure
end
```

The forces acting on a given slot are calculated as follows:

```
dir is the translational force vector
torqs is the array of rotational torque vectors
initialize dir to the zero vector
initialize torqs to empty
add a radial expansion force to dir if performing a collision cascade
if slot is not empty
  get node i connected to slot
  if spring to node i is too long then break it and return
  calculate offset to node i from slot
  add a force proportional to offset to dir
  calculate torque vector resulting from offset and add to torqs
  determine optimal bond angle by averaging angles from both nodes
  calculate torque vector needed to reach optimal bond angle
  add it to torqs
  calculate torque vector needed to reach optimal twist angle
  add it to torqs
  perform a nearest neighbor search from current slot
  for each node j in nearest neighbor list
    for each corner k of node j
      if k is within repulsion threshold and all of its slots are full
        calculate torque needed to repulse slot from corner k
```

```
        add it to torqs
      else if k has an empty slot
        calcualte torque needed to attract slot to corner k
        add it to torqs
      end
    end
  end
end
```

4.5.6 Buckets and Nearest Neighbor Search

A time consuming step during growth and optimization is searching for nodes that are within a certain distance of a given node. Initially, this was accomplished by calculating the distance to every node in the structure and checking whether or not it was within the required threshold. However, as structures get very large, the time required to perform this calculation grows very quickly. To solve this problem, the simulator implements buckets, sub-divisions of the simulation workspace which keep track of every node within its region. Buckets were also used in virus shell assembly for performance increase [2]. Using buckets, to find neighboring nodes, the simulator only needs to search neighboring buckets instead of searching the entire structure. This allows for a nearest neighbor search time that grows linearly with the size of the structure. A single bucket is implemented by the following structure:

```
structure {
  size:  integer
  list:  array of nodes
} Bucket
```

The global variables `b_width`, `b_height`, and `b_depth` determine the dimensions of the three-dimensional array containing all of the buckets in the simulation. In

addition, the global variable `b_size` determines the dimensions of the cubical volume of each bucket. When a node is added to the structure it is assigned to the bucket containing the space which the node is in. The actual insertion of the node into the bucket is accomplished by adding the node to the bucket structure's `list` element and incrementing its `size` element.

Buckets allow for extremely fast nearest neighbor searches, on the order of O(1). The actual algorithm used to find the nodes near a node n is given here:

```
let r be the search radius from node n
let buckets be the array containing all buckets intersected by
  a sphere s of radius r centered at node n
for each bucket b in buckets
  for each node i in b
    if the coordinates of node i are within sphere s
      add i to nearest neighbor list
    end
  end
end
return nearest neighbor list
```

4.5.7 Collision Cascades

Collision cascades simulate the effects of irradiation on a structure. The simulator performs collision cascades using two different methods. The first method, recrystallization, removes all the nodes within the affected region and then adds new nodes to the boundary until the vacuum is filled. New nodes are added according to a different set of rules from those used to grow the original structure. A common action sequence that a user might follow is to first grow a large structure using rules set A, remove a fifth of the nodes located in the center of structure, load a new rules set B, and then regrow the deleted nodes. A decription of the recrystallization algorithm follows:

```
queue is initialized to empty
for each node i in the irradiated region
  for each corner j of node i
    for each slot k of corner j
      if the node attached at slot k is outside the irradiated
        region and it is not already in queue then add it to
        queue
      end
    end
  end
end
for each node i in the irradiated region
  remove node i from structure
end
while queue is not empty
  remove node i from the head of the queue
  for each corner j of node i
    for each slot k of corner j
      if k is empty
        try to attach a nearby node
        if no nearby node could be attached
          add a new node to node i at slot k
          if successfull
            add the new node to tail of queue
          else
            try to break and reattach to a nearby node
          end
        end
        optimize the structure
      end
```

```
    end
  end
end
for each node i in the structure
  if node i has an unfilled slot then add it to the queue
end
while queue is not empty
  remove node i from the head of the queue
  for each corner j of node i
    for each slot k of corner j
      if k is empty
        try to attach a nearby node
        if no nearby node could be attached
          try to break and reattach to a nearby node
        end
        optimize the structure
      end
    end
  end
end
```

The second method of performing collision cascades is spinodal decomposition. This method involves disturbing the nodes in the irradiated region by breaking all bonds and randomly rotating and offsetting nodes. Then, the nodes are allowed to move and reattach themselves using a different rules set until all bonds are reformed. The algorithm used for spinodal decomposition is as follows:

```
for each node i in the irradiated region
  disturb node i by translating and rotating it slightly
end
```

```
queue is initialized to empty
for each node i in the structure
  if node i has an unfilled slot then add it to the queue
end
while queue is not empty
  remove node i from the head of the queue
  for each corner j of node i
    for each slot k of corner j
      if k is empty
        try to attach a nearby node
        if no nearby node could be attached
          try to break and reattach to a nearby node
        end
        optimize the structure
      end
    end
  end
end
for each node i in the structure
  if node i has an unfilled slot then add it to the queue
end
while queue is not empty
  remove node i from the head of the queue
  for each corner j of node i
    for each slot k of corner j
      if k is empty
        try to attach a nearby node
        if no nearby node could be attached
          try to break and reattach to a nearby node
        end
```

```
          optimize the structure
        end
      end
    end
  end
```

The best method to use for collision cascades varies between structures and it is up to the user to decide which is better.

Chapter 5

Evaluation

This chapter assesses how well the implementation meets the requirements specified in Chapter 3. Section 5.1 evaluates how well the simulator meets the functionality requirements. Section 5.2 evaluates how well the simulator meets the performance requirements. Section 5.3 evaluates how well the simulator meets the portability requirements.

5.1 Functionality

The simulator meets all of the functional requirements specified in Chapter 3. It models a ceramic network as a mass-spring system, adds nodes according to a given set of local rules, and constantly minimizes the energy of the structure as it grows. It successfully grows all compounds for which rules were defined, including crystal and glass forms of quartz, cristobalite, moganite, keatite, coesite, tridymite, silicon carbide, and silicon nitride. Figure 5-1 shows the results from running two simulations of β-cristobalite growth. The first structure is a crystal of β-cristobalite, and the second is an amorphous version grown using deviant rules. The simulation ensures balanced structures by adding nodes in a breadth first manner, *i.e.*, all of the vertices of an existing node are filled before those of a new node.

In addition, the simulator performs all required statistical computations. These include the calculation of primitive rings and local clusters, densities, and bond angle

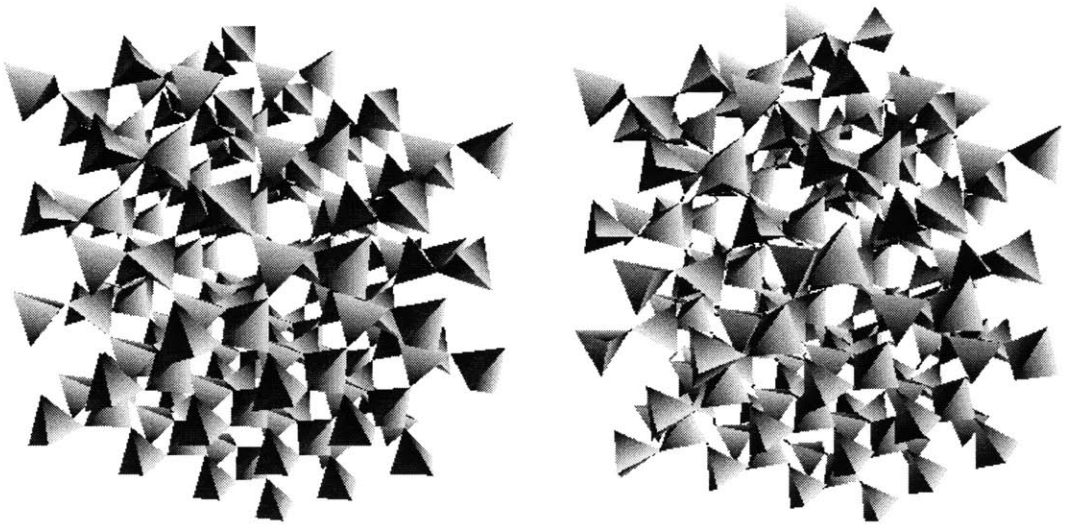


Figure 5-1: Comparison of 200-node models of (a) β-cristobalite and (b) cristobalite-like glass. The darker tetrahedron represents an underbound node.

and length distributions. The simulator calculated primitive rings using standard algorithms except for the use of several local distance matrices in place of one global distance matrix for efficiency. Densities were computed using two different methods. The first method involving the calculation of a convex hull was not as accurate as averaging the densities of randomly distributed spheres within the structure, so the later was used in the final version of the simulator. Table 5.1 and Figures 5-2 and 5-3 show the results of various statistical calculations performed on silica compounds.

The user interface consists of two parts: a control window with buttons for performing functions and setting options and a graphical output window for displaying structures as they are assembled. The interface adequately satisfies the requirements of ease of use and versatility by placing buttons for the most commonly used functions on the top-level of the control window and placing buttons for less commonly used functions in lower sub-menus. For instance, to load a new rules file, the user need only click on the "new rules" located on the main menu, but to change the spring constant the user must first select the options button and then alter the spring constant field.

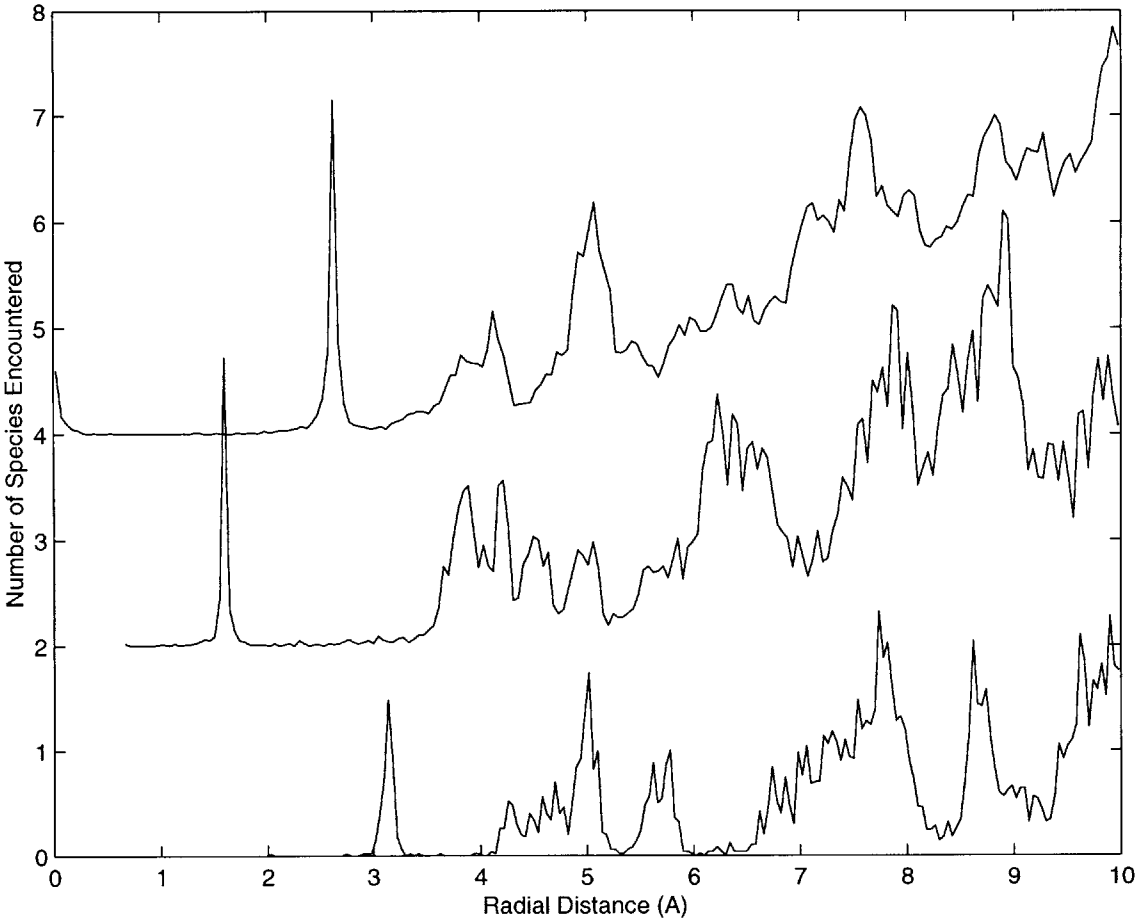


Figure 5-2: Partial radial density functions (RDFs) for O-O (top), Si-O (middle) and Si-Si (bottom) correlations in the quartz-like a-SiO_2 model with 5° initial node offset from idealized quartz. The Si-O and O-O plots have been displaced vertically by 2 and 4 units respectively. The correlation variations closely follow correlation histograms for idealized crystalline quartz.

5.2 Performance

Figure 5-4 summarizes performance results from several speed tests. The graph shows that growth time is linear with size when optimization does not take place. Linearity is achieved through the use of buckets which allow for constant-time nearest neighbor searches that are needed for assembly. Optimized growth time, however, is exponential with size. This can be explained by the additional spring force calculations needed at each time step. As the structure grows linearly in size, the total number of spring force calculations grows exponentially. When the number of nodes in a

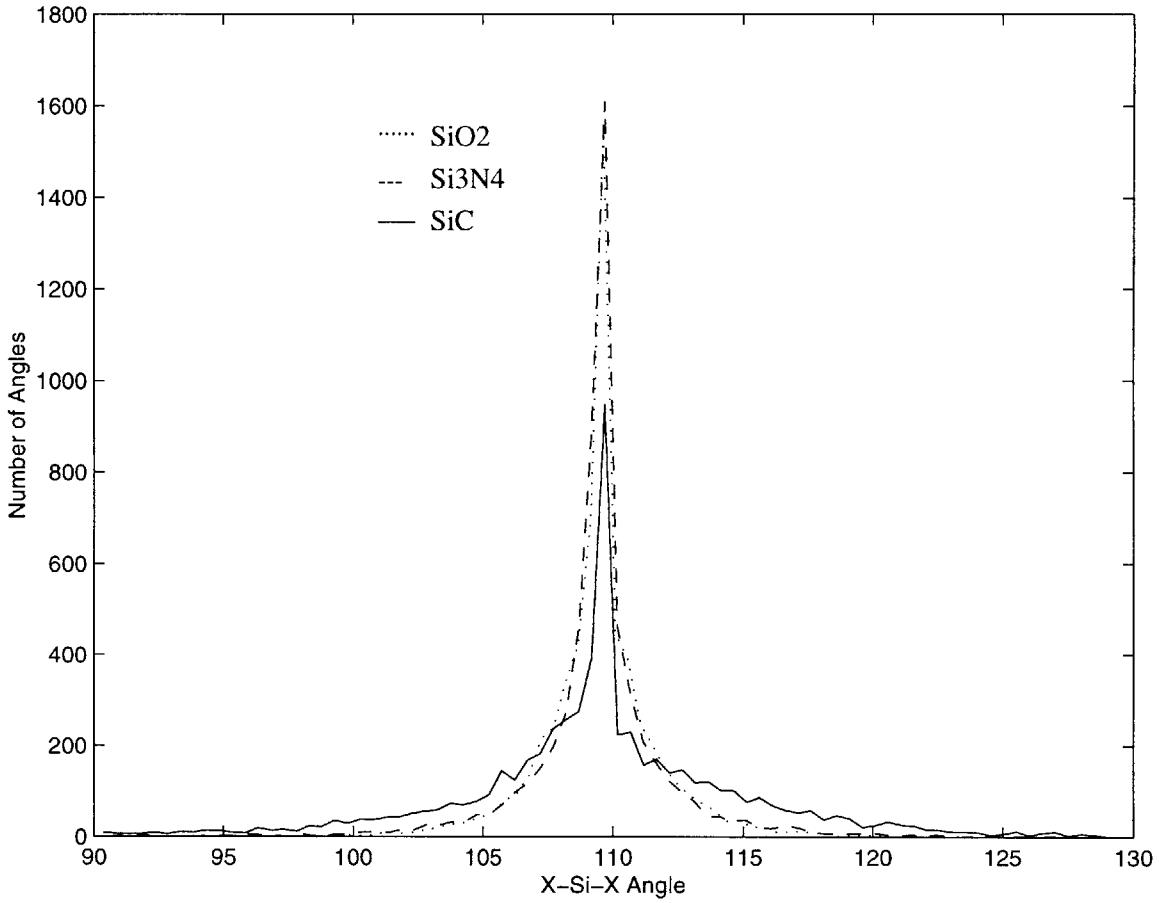


Figure 5-3: X-Si-X bond angle distributions for amorphous structures.

structure exceeds approximately 4000 nodes, the simulator will fail to meet the speed requirement of 1000 nodes per hour. However, with the use of parallel algorithms, faster processors, and more efficient algorithms this limitation can be removed as described in Section 7.2.

To test accuracy, results were compared to actual values determined from laboratory experiments. Table 5.2 shows a comparison of densities derived from the simulator with actual densities observed in nature. Model densities very closely match actual densities, satisfying the accuracy requirements of the simulator.

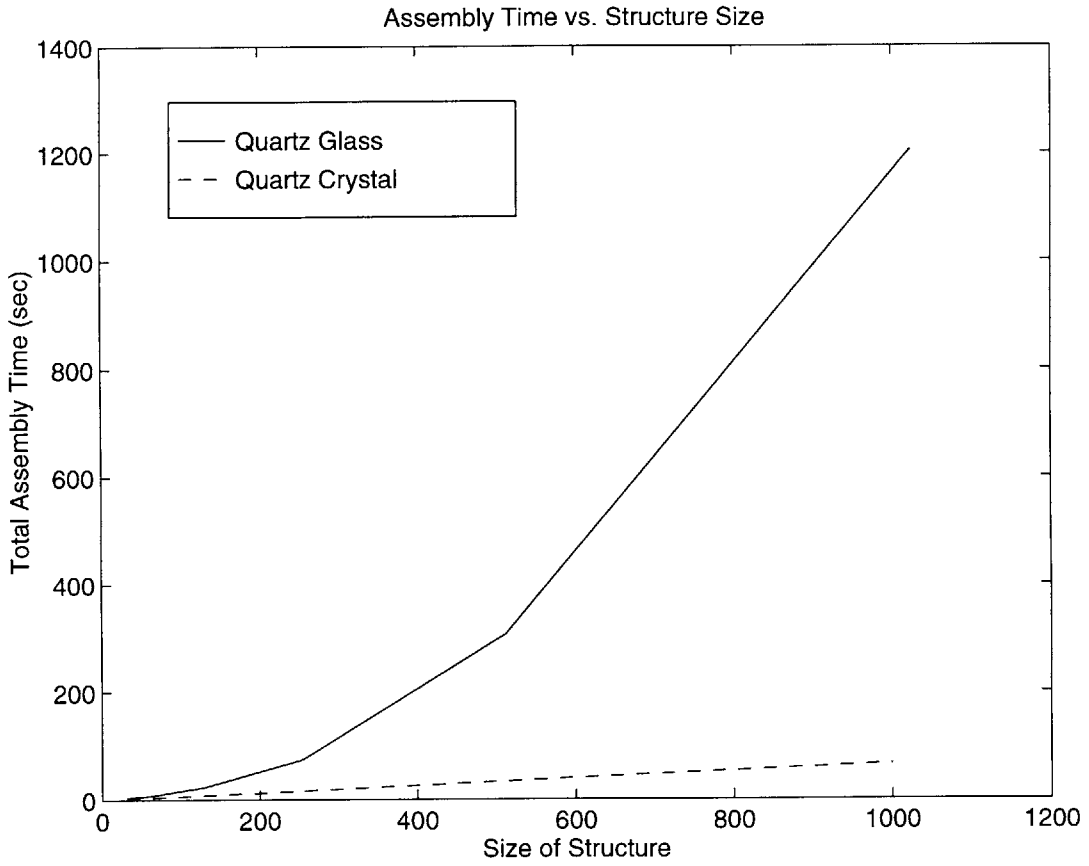


Figure 5-4: Comparison of calculation speed between amorphous (optimized) and crystalline (unoptimized) structures of varying size. The rate of growth of calculation time is linear when optimization is off, but it is exponential when optimization is on.

5.3 Portability

The simulator meets all specified portability requirements. It compiles on a variety of platforms including Silicon Graphics, Sun, DEC, and Intel based machines. On machines without 3D graphical devices (*i.e.*, all machines except for Silicon Graphics), the graphics are not displayed but all computations can still be performed. Aside from graphics, there are no differences between platforms.

In addition, the simulator can produce output in the form of VRML files which can later be incorporated into a web interface.

Polymorph	Node	Si-O-Si θ°	Ave Si-O-Si θ°	Model Density $\times 10^3 kg/m^3$
ideal-Cristobalite	0	180	180	1.92
β-Cristobalite	0	148	148	2.17
α-Cristobalite	0	145.3	145.3	2.33
ideal-Tridymite	0	180	180	1.92
HP-Tridymite	0	138.8 (3), 180	149.5	2.22
x-Tridymite	0	123.7 (3), 180	137.8	2.54
ideal-Quartz	0	155.6	155.6	2.37
β-Quartz	0	150.9	150.9	2.51
α-Quartz	0	143.6	143.6	2.77
Keatite	0	147.8, 150.5 (2), 164.0	154.6	2.61
	1	147.8 (2), 164.0 (2)		
ideal-Moganite	0	143.6	152.7	2.24
	1	180 (2), 143.6 (2)		
β-Moganite	0	139.8 (2), 146.5 (2)	149.5	2.39
	1	139.8, 148.5, 165.9 (2)		
α-Moganite	0	125.2 (2), 143.9 (2)	137.5	2.72
	1	125.2, 143.9, 146.3 (2)		
Coesite	0	140.9, 144.7, 148.9 (2)	150.8	2.82
	1	144.7, 148.9 (2), 180		
β-Si_3N_4	0	109.5, 120, 125.1×2	119.8	3.29
α-Si_3N_4	0	110.6, 114.2, 117.2, 118.1		
		118.2, 118.4, 119.6, 124.9		
	1	111.7, 113.1, 116.8, 120.3		
		120.6, 122.2, 123.0, 124.9	118.3	3.22
	2	110.5, 115.6, 116.9, 117.3		
		117.8, 117.8, 117.8, 124.8		
	3	111.7, 113.1, 117.1, 117.7		
		117.9, 118.4, 119.6, 124.5		
β-SiC	0	109.5	109.5	3.21
α-SiC	0	109.5	109.5	3.24

Table 5.1: Si-X-Si Angles and Densities for Rules-Generated Crystal Polymorphs

Polymorph	Model Density $\times 10^3$ kg/m^3	Actual Density $\times 10^3$ kg/m^3
α-Cristobalite	2.33	2.33
β-Cristobalite	2.17	2.21
α-Quartz	2.77	2.65
β-Quartz	2.51	2.53
α-Si_3N_4	3.22	3.20
β-Si_3N_4	3.20	3.20
α-SiC	3.24	3.22
β- SiC	3.21	3.22

Table 5.2: Densities for Silicas Modeled

Chapter 6

Applications

This chapter demonstrates the usefulness of the simulator and provides two example applications. Section 6.1 demonstrates how the simulator can be used to perform local cluster analysis of compounds. Section 6.2 shows how it can also be applied to experiments in amorphization.

6.1 Local Cluster Analysis

One advantage of using a computer over hand modeling is that a computer can locate primitive rings much more quickly and with more accuracy. With hand modeling, rings can only be identified by tedious inspection and even when a ring is found, it is difficult to determine if it is primitive. Using the algorithms described in Chapter 4 a computer can compute rings almost instantaneously.

Global ring distributions are not enough to describe the full topological properties of structures and cannot differentiate between them. A more useful description of the topology is the local cluster of a node within the structure (see Section 4.5.2). Just as the unit cell can differentiate local symmetries between crystals, the local cluster can differentiate local topologies in both crystals and non-crystalline structures. In this application, we will examine the local clusters of various crystals and glasses and use them to differentiate between compounds and gain insight into their physical properties.

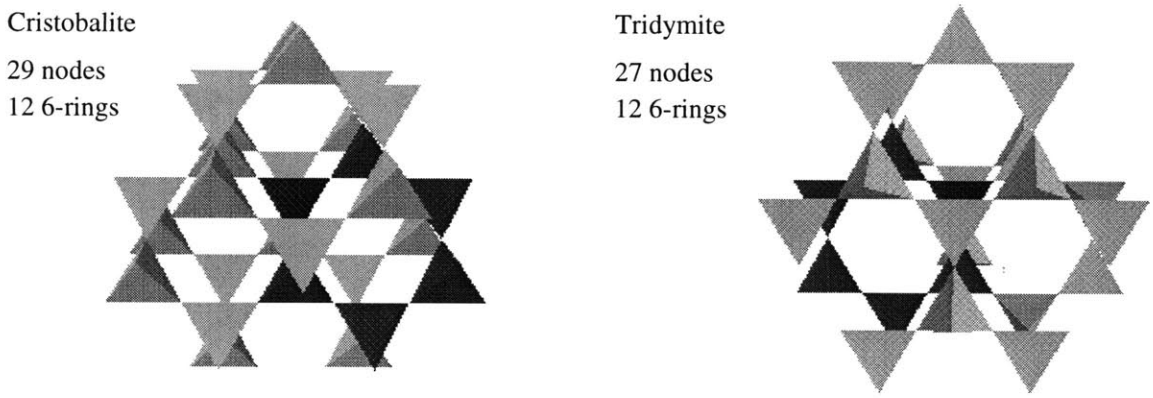


Figure 6-1: Local clusters for cristobalite and tridymite. Both of the clusters contain 12 6-rings through the center node, but differ in number of tetrahedra. One 6-ring is highlighted in each cluster.

Figure 6-1 shows local clusters for tridymite and cristobalite. These two polymorphs of silica are very similar in structure. In each structure, the tetrahedra are structurally equivalent, *i.e.*, the local cluster is the same regardless of which tetrahedron is chosen. Each of their local clusters contain 12 6-rings, two through each edge or 12 through each node. Only the size of the cluster distinguishes the two compounds: tridymite contains local clusters with 27 nodes, while cristobalite contains local clusters with 29 nodes.

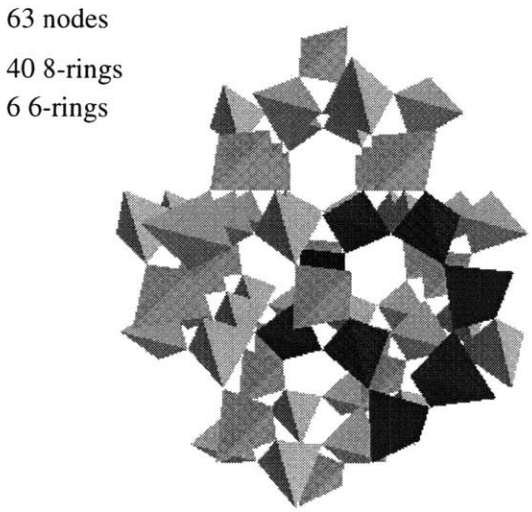


Figure 6-2: The local cluster of quartz, comprising 63 nodes and containing 40 8-rings and 6-rings. A convoluted 8-ring is highlighted.

Quartz and keatite are also interesting to examine. The local cluster for quartz is

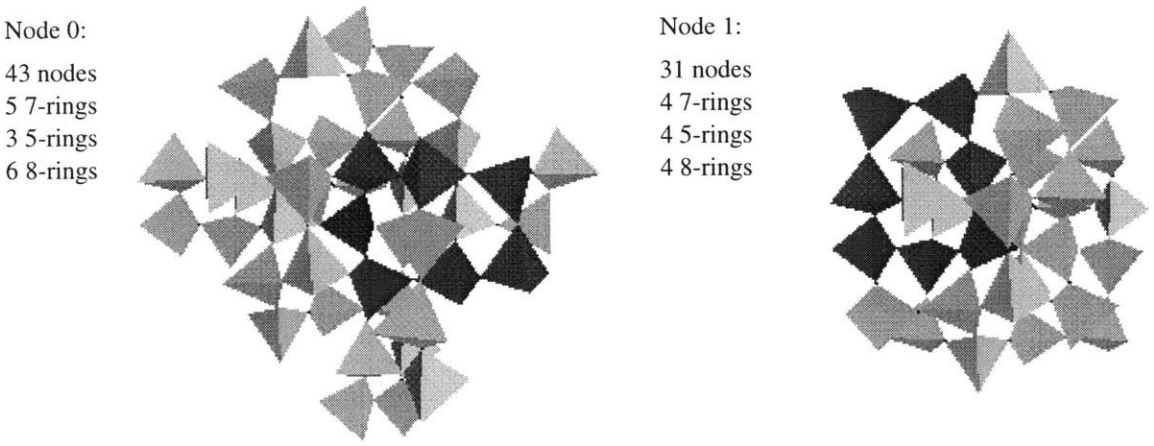


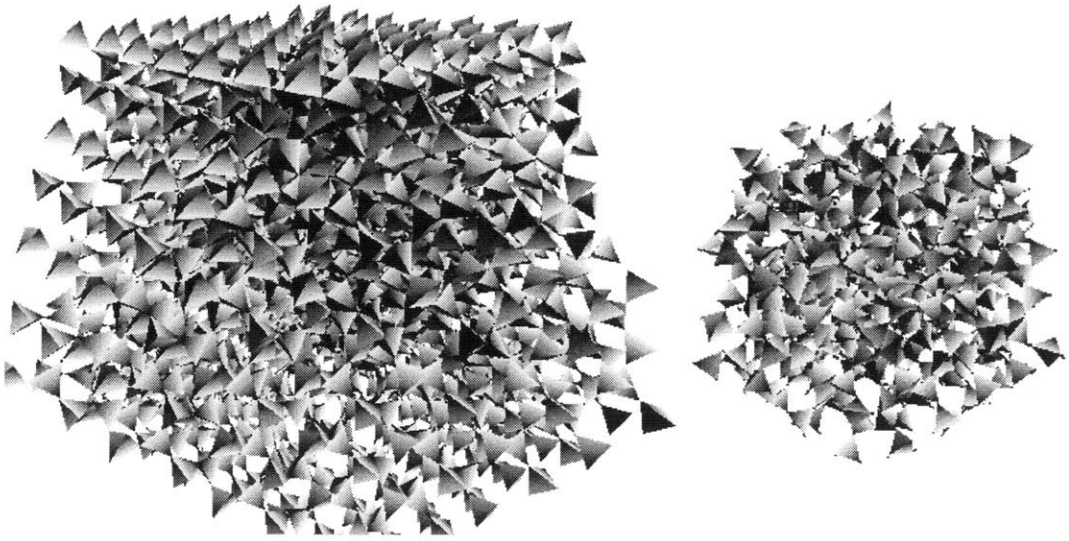
Figure 6-3: The two local clusters of keatite, corresponding to the two inequivalent tetrahedron environments. Both contain odd 5- and 7-rings, in addition to 8-rings. Two 7-rings are highlighted.

shown in Figure 6-2. Quartz contains local clusters of size 63 composed of 6 6-rings and 40 8-rings. It is interesting to note the correlation between the higher density of quartz with the presense of more 8-rings. The longer 8-rings allow for more folding and leads to denser packing of tetrahedra. Keatite is structurally much different from quartz. It has two types of local clusters corresponding to different tetrahedral types. They are shown in Figure 6-3. The first cluster contains 3 5-rings, 5 7-rings and 6 8-rings, and the second contains 4 5-rings, 4 7-rings and 4 8-rings. The presence of 5- and 7-rings in keatite, a crystal, is surprising because odd-membered rings are commonly associated with aperiodic compounds.

This application demonstrates the usefulness of a computer simulation over hand modeling. The local clusters derived above were computed nearly instantaneously using the simulator. Hand modeling would have required hours of tedious visual examination to compute just one local cluster, with no guarantee of accuracy. The simulator allows researchers to focus more on the theoretical implications of local clusters rather than on their actual computation.

6.2 Amorphization Experiments

Another application of the simulator is to model a disordering event such as radiation which can transform a periodic crystal into a structure with amorphous regions. How well a material is able to propogate this disorder and keep tetrahedral distortion at a minimum is an indication of its stability. Although experiments were run on a variety of substances, the results from only two of the simulations are presented here.



(a) (b)

Figure 6-4: The results from α-cristobalite regrown using β-cristobalite rules. (a) is the entire resulting structure and (b) is only the regrown region.

In each experiment, a model comprising 2000 tetrahedra was assembled using a set of local rules. Then a region of 400 nodes was disrupted via spinodal decomposition (see Section 4.5.7) and allowed to reform according to another set of local rules. In the first experiment, the disordered region of α-cristobalite was regrown using β-cristobalite rules, and in the second experiment, the disordered region of α-SiC was regrown using the same set of α-SiC rules. The results of these experiments are shown in Figures 6-4 and 6-5. Stretched springs (represented by lines) between tetrahedra represent distortions resulting from regrowth.

Disorder was propagated easily in the case of α-cristobalite. The resulting structure had very few underbound nodes and showed very little tetrahedral distortion.

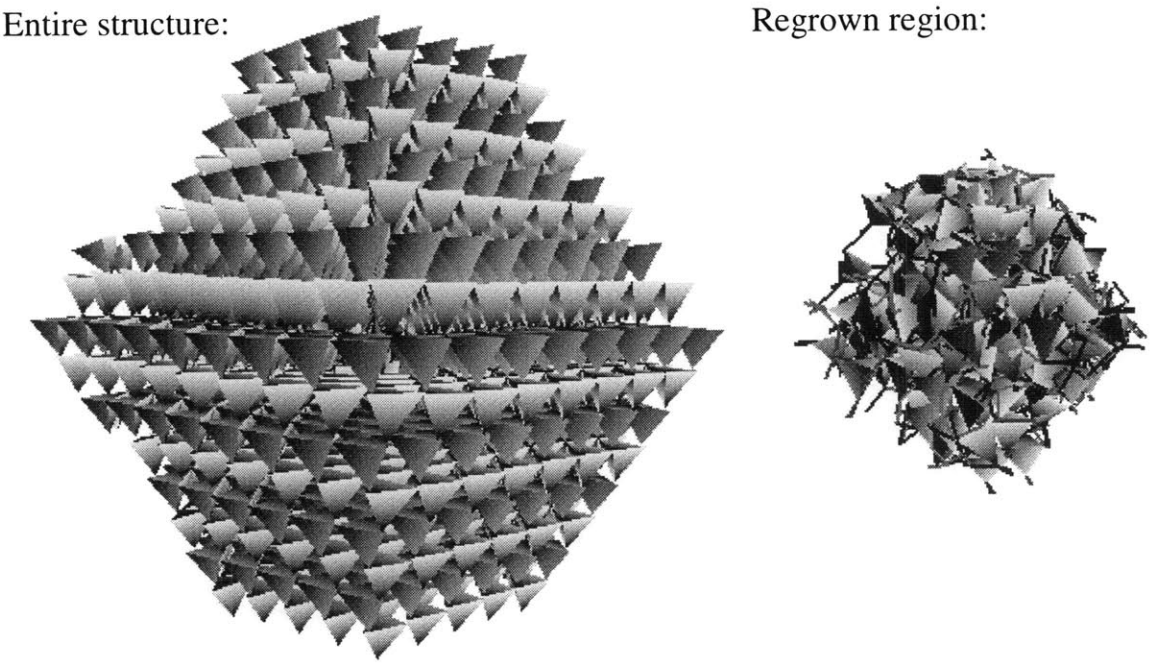


Figure 6-5: Cascade simulation for a a) 2000 tetrahedra of α-SiC in which b) the embedded central 400 tetrahedra have been disordered and rebonded using α-SiC rules. Many underconnected tetrahedra (darker) and large remanent optimization spring segments remain.

This can be explained by the independence of inter-tetrahedral Si-O-Si angles from each other and from the O-Si-O intra-tetrahedral angles [12]. By contrast, SiC could not acceptably propagate disorder. The resulting structure was substantially underconnected with a large amount of tetrahedral distortion. This is most likely the result of a strong interdependence of the four Si-C-Si angles at each vertex and an overall lack of topological freedom in the structure.

Chapter 7

Discussion

7.1 Conclusions

In this thesis paper we have described the development of a simulator for the modeling of ceramic structures. The simulator meets all of its design specifications and provides useful information about the structure and stability of various silicon-based compounds (glass and glass-like). It assembles compounds at a much faster rate than similar molecular dynamics based simulators, and confirms the topological properties of structures that were previously analyzed through slower hand modeling experiments. In addition, it provides a simple user interface and displays its results on an interactive three-dimensional display. All of these features of the simulator together provide a very useful tool for materials scientists who wish to explore various structural properties of ceramic compounds. The simulator provides an alternative to more expensive and time-consuming laboratory experiments, and it provides information that may be otherwise impossible to gather in a physical setting.

This project raises an important point: brute force methods such as molecular dynamics simulations may not always be the best way to approach a complicated problem like glass assembly. Instead, it may prove more useful to make approximations in areas of the model that do not require much accuracy. For instance, the simulator obtained correct results even though it modeled complicated binding interactions between atoms as simple springs. In addition, instead of relying on com-

plicated molecular interactions to model the addition of new atoms, the simulator relied on a set of local rules to determine how new atoms would be oriented. Although these rules were derived from pre-existing empirical data, possibly even from molecular dynamics experiments, they only had to be determined once. As a result, the same work is not repeated between experiments.

Although the simulator has many advantages over actual laboratory experiments, we must keep in mind that it is in the end performing a simulation. All computer simulators are based on theory, and as a result they make many approximations to the real world and rely on various simplifying assumptions. Since in most cases, theory differs from practice one must be extremely cautious when using results from the simulator. They should be confirmed in the laboratory before being put to practical use. The simulator should be used as a tool to aid in laboratory work rather than as a replacement for it.

7.2 Potential Improvements

One potential improvement to the simulator is the implemention of parallel algorithms. Optimization of an assembled structure could be accomplished faster by splitting the work among several processors. This is feasible since there is little interdependence between non-adjacent regions. Large network optimization problems such as this are very conducive to parallelization and would see a vast improvement in speed. A parallel implementation of virus shell assembly on a CM-5 machine has already demonstrated substantial speed increases [22].

Adding kinetics to the existing model is another potential improvement. The success of kinetics has already been demonstrated by the virus shell assembly simulator of Schwartz, *et al.* [23]. Kinetics represent a tractable middle-ground between abstract local rules and more physical molecular dynamics. It provides some of the computational advantages of local rules, as well as the physical accuracy of molecular dynamics.

Another area which could be improved is network optimization. Currently, simple

time-stepping algorithms are used to converge on a minimum energy solution. More advanced Newton and Euler numerical methods could be used which would require much less work to reach the same solution. There is a tradeoff, however, between speed and complexity of the algorithms.

Furthermore, the user interface has several limitations that need to be addressed. First, the current simulator does not allow direct manipulation of the 3D model in the display window. Instead, it relies on buttons to indirectly rotate the structure. In addition, the user interface does not provide a utility to assist the user in constructing rules files. Determining the initial offsets and rotations is a very difficult task and, therefore, needs to be simplified. Finally, there is currently no world wide web interface to the simulator. The construction of a web page would make the simulator available to a much greater audience and would also allow the design of the HTML based user interface to be independent from the core of the simulator.

The simulator has already demonstrated potential as a useful tool for material scientists interested in analyzing the amorphization properties of periodic and aperiodic glasses. With these and other improvements, it may one day be used to uncover new facts about glasses that could never have been learned through regular laboratory experiments or molecular dynamics simulations.

Bibliography

[1] Bonnie Berger and P. W. Shor. Local rule switching mechanism for viral shell geometry. MIT Laboratory for Computer Science Technical Memo 527, 1995.

[2] Bonnie Berger, P. W. Shor, L. Tucker-Kellogg, and J. King. A local rules based theory of virus shell assembly. *Proceedings of the National Academy of Science, USA*, 91:7732–7736, 1994.

[3] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*, chapter 26, page 550. The MIT Electrical and Computer Science Series. Massachusetts Institute of Technology, Cambridge, Massachusetts, third edition, 1990.

[4] M.T. Dove, A.P. Giddy, and V. Heine. Rigid unit mode model of displacive phase transitions in framework silicates. *Transactions of the American Crystallographic Association.*, 27:65–75, 1991.

[5] K. Goetzke and H.J. Klein. Properties and efficient algorithmic determination of different classes of rings in finite and infinite polyhedral networks. *Journal of Non-Crystalline Solids*, 127:215–220, 1991.

[6] P. K. Gupta and A. R. Cooper. Topologically disordered networks of rigid polytopes. *Journal of Non-Crystalline Solids*, 123:14, 1990.

[7] Linn Hobbs. Network topology in aperiodic networks. *Journal of Non-Crystalline Solids*, 192:79–91, 1995.

[8] L.W. Hobbs, C.E. Jesurum, V. Pulim, and B. Berger. Local topology of silica networks. *Philosophical Magazine*, 78:679–711, 1998.

[9] L.W. Hobbs, C.E. Jesurum, V. Pulim, and B. Berger. Topological modeling of cascade amorphization in network structures using local rules. *Materials Science and Engineering A*, 253:16, 1998.

[10] D.J. Jacobs and M.F. Thorpe. Generic rigidity percolation: the pebble game. *Physical Review Letters*, 75:4051–4054, 1995.

[11] U. Jain, A.C. Powell, and L.W. Hobbs. Simulation of the crystal-to-amorphous transformation in irradiated quartz. In *Materials Research Society Symposium Proceedings*, volume 209, pages 201–206, 1991.

[12] C.E. Jesurum. *Local-rules based topological modeling of ceramic network structures.* PhD dissertation, Massachusetts Institute of Technology, Department of Mathematics, 1998.

[13] C.E. Jesurum, V. Pulim, B. Berger, and L.W. Hobbs. Topological modeling of amorphized tetrahedral ceramic network structures. *Proceedings of the Fall Meeting of the Materials Research Society*, 1996.

[14] C.E. Jesurum, V. Pulim, and L.W. Hobbs. Modeling the cascade amorphization of silicas with local-rules reconstruction. *Nuclear Instruments and Methods B*, 141:25–34, 1998.

[15] C.E. Jesurum, V. Pulim, and L.W. Hobbs. Topological modeling of amorphized tetrahedral ceramic network structures. *Journal of Nuclear Materials*, 253:87–103, 1998.

[16] C.E. Jesurum, V. Pulim, L.W. Hobbs, and B. Berger. Modeling of topologically disordered tetrahedral network structures using local rules. *ICDIM*, 1996.

[17] C.E. Jesurum, V. Pulim, L.W. Hobbs, and B. Berger. Modeling collision cascade structure of SiO_2, Si_3N_4 and SiC using local topological approaches. *Materials Research Society*, Fall Meeting 1997.

[18] Carol Marians and Linn Hobbs. Local structure of silica glasses. *Journal of Non-Crystalline Solids*, 119:269–282, 1990.

[19] Carol Marians and Linn Hobbs. Network properties of crystalline polymorphs of silica. *Journal of Non-Crystalline Solids*, 124:242, 1990.

[20] C.S. Marians. *A language for the study of silica networks.* PhD dissertation, Massachusetts Institute of Technology, Department of Materials Science and Engineering, 1988.

[21] Doug Muir. Simulating the formation of viral protein shells. SM thesis, M.I.T., Cambridge, MA, 1994.

[22] R.S. Schwartz. A multi-threaded simulator for the kinetics of virus shell assembly. SM thesis, M.I.T., Cambridge, MA, 1996.

[23] Russell Schwartz, Peter W. Shor, Peter E. Prevelige, Jr., and Bonnie Berger. Local rules simulation of the kinetics of virus capsid self-assembly. Biophysical Journal, 1998.

[24] P. Vashishta, A. Nakano, R.K. Kalia, and I. Ebbsjö. Molecular dynamics simulations of covalent amorphous insulators on parallel computers. *Journal of Non-Crystalline Solids*, 182:59–71, 1995.