Simulation of Anisotropic Crystal Etching

Bill Foote
Master's Project Report
Under Direction of
Professor Carlo H. Séquin

Computer Science Division
Department of EECS
University of California
Berkeley, CA 94720
September 12, 1990

ABSTRACT

A series of programs have been developed to model anisotropic etching of crystalline substances. The focus of this work was on the computation of the geometric offset surfaces for a given object, when the etching of different faces progresses at different rates depending of face orientation. Programs have been developed to calculate the emerging shapes for both two- and three-dimensional geometries.

Since complete information about the anisotropic etch rates in all possible directions have been published for only very few combinations of crystals and etch solutions, we also had to write a rudimentary generator that would produce plausible and self-consistent direction-dependent etch rate functions. This modeling proceeds in stages: From the geometry of the crystal lattice, its atom spacings and angles between bonds, some inferences are made about the probability that certain more or less exposed atoms get attacked and removed by the etchant. With this model the etch rates for several key directions, i.e., for the simple crystallographic planes (100), (110), (210), (111), (211), and (221), have been calculated. The etch rates for the direction in between these key orientations are found by interpolation.

The etching simulator then uses such an artificially generated function or any function that may come from experimental observations and applies it to arbitrary polyhedral shapes. For all edges and vertices it first determines what new bevel faces might form because of strong local maxima and minima in the etch rate function. Then all the faces, the original ones as well as the bevel faces, are advanced at their corresponding etch rates and combined into a new consistent surface description of a solid object. The user can specify the total etching time and the number of intermediate states that should be displayed. The shapes obtained from the etching simulator programs are in good qualitative agreement with the kinds of shapes actually observed in the laboratory.

This work was supported in part by the Central Research Labs of Siemens Corp.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In this paper, we describe a system that we have implemented to model the etching of a general shape from a crystalline substance such as silicon. The analysis is divided into three parts: a model of the rate at which faces of various orientations etch, a model of what faces appear at a corner, and a system that applies the ''etching model'' generated by the previous two modules to an arbitrary shape. This analysis is carried out first in two dimensions, then in three.

The etching rate of a substance can be described with a polar diagram. It is difficult, in general, to find complete data on the etching rate of a given substance, such as silicon, being etched with a given etchant. [Seidel, '82] Usually, all that is available are the etching rates for a handful of etching directions. We are interested, however, in studying a variety of situations, and require therefore, an automatic generator of reasonable etch rate polar diagrams.

The other part of this analysis concerns an etching simulator. This is a system that accepts a representation of an etching rate polar diagram and simulates the action of an etchant described by that diagram on an arbitrary shape. The polar diagram may be one created with the model developed in this paper or it may be one that comes from any other source.

This paper is divided into two main sections: a two dimensional analysis, and a three dimensional analysis. The 2D analysis allows many of the issues involved to be explored in a framework that is much easier to understand. The ideas developed in 2D are then applied and extended to the 3D case in the second part of the paper. Each section is divided into four main parts: A derivation of an etching rate model, an analysis of how the geometry in the neighborhood of a corner changes under etching, an etching simulator program, and results obtained with the simulator.

The first section in both 2D and 3D is a derivation of an etching rate model. This analysis allows us to generate plausible polar diagrams of the etching rate, and gives us insight into the etching process.

The next section consists of an analysis of how the local geometry of a shape changes corner under etching. This analysis centers around what happens at a corner of a shape being etched. Experimental data suggests that new faces can develop at the corners of objects being etched. A model of this process is developed in this section. As will be seen, this analysis depends strongly on assumptions made in the preceeding section.

The ideas developed in the preceeding two sections are applied to a working computer simulation of the etching process in the third section. This section explores some of the implementation decisions that were made in the creation of this simulation.

Notes on notation

Throughout this paper, the notation $\vec{v}$ is used to denote a vector. A hat is used to denote a unit vector, thus $\hat{v}$ denotes a unit vector in the direction of $\vec{v}$. Leaving the vector symbol off a vector name denotes the magnitude of a vector, thus $|\vec{v}| = v$.

CHAPTER 2

2D ETCHING RATE MODEL

To know how a shape changes when it is etched, one needs to know how fast material is removed along faces of different orientations. We are interested in creating a system that can handle strongly anisotropic etchings; for example, with Silicon different planes exhibit a difference in etch rate of over 100:1 with certain special anisotropic etching solutions. As mentioned earlier, it is difficult to find complete polar diagrams of the etch rate for any material. We wanted a generator that would produce a wide variety of reasonable and at least consistent diagrams. An ad hoc model of the etching process was therefore assumed, and that model was applied to a very simple model of a crystal lattice. This allowed the derivation of etch rate diagrams that are at least free of internal contradictions. These etch rate functions also lead to the typical shapes found in the reports of experimental work.

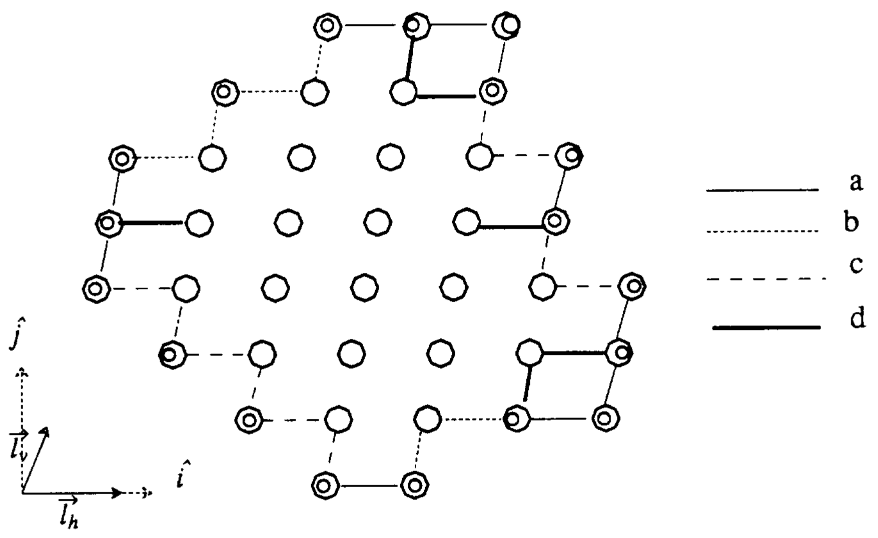


Figure 2.1. This shows the different types of bonds that occur in a crystal. "a" bonds are found along a flat face, and are the easiest to break. "b" bonds are somewhat protected, and "c" bonds are slightly more protected for a biclinic lattice (like the one illustrated). "d" bonds are the most internal and thus protected. The vectors $\vec{l_h}$ and $\vec{l_v}$ specify the distance between atoms horizontally and vertically. Of course, for each bond type there are corresponding vertical and horizontal bonds; there are eight different bond strengths in all (a_v, b_v, c_v, d_v, a_h, b_h, c_h, and d_h) A "bond strength" is represented as the probability of a given bond breaking per unit time.

The derivation of an etch rate diagram is divided into two parts: the determination of the etch rate at several key face orientations, and the determination of the etch rate at an arbitrary orientation by interpolation. Only crystals with one "atom" (or other basic unit of matter) per unit cell were considered. Effects of the etchant working on a substance, such as surface heating, convection, or non-uniform distribution of the etchant were ignored in this analysis. Biclinic lattices (lattices in which the axes are not orthogonal) are allowed. Notationally, the unit cell of the 2D crystal lattice is described with two vectors $\vec{l_h}$ and $\vec{l_v}$ (for horizontal and vertical). To simplify analysis, $\vec{l_h}$ is constrained to lie along the positive x axis, and $\vec{l_v}$ is constrained to either lie along the positive y axis or be within the first quadrant.

The probability of an atom being etched away during one time step is modeled as the product of the probabilities of its bonds breaking. For each atom, only the existing bonds to its four nearest neighbor locations are considered. Bonds at an internal corner or within the crystal are more protected from attacks by the etchant and are thus expected to have a lower probability of breaking. 8 different bond types are distinguished. These bonds are illustrated in figure 2.1.

1. Face Characterization

A face is generally characterized by its Miller indices (i,j). For example, figure 2.2 is an illustration of the (4,1) face. Counting atoms, i is the number of atoms vertical displacement per j atoms horizontally, i.e. in figure 2.2, each step of 1 atom horizontally results in a step of 4 vertically. Another way to view the Miller indices of a face is by considering where a face intersects the unit cell basis vectors $\vec{l_h}$ and $\vec{l_v}$. For any face, the intercepts can be

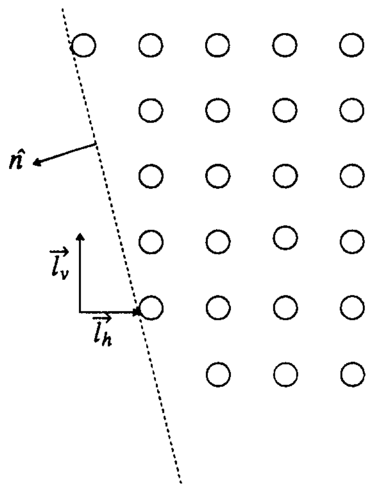


Figure 2.2. This figure shows the (4,1) face, and shows the unit cell basis vectors $\vec{l_h}$ and $\vec{l_v}$. It can be seen that the face intersects the basis axes at $\vec{l_h}$ and at $4\vec{l_v}$. Scaling by $\frac{1}{4}$, the intercepts can be seen to be at $\frac{\vec{l_v}}{4}$ and $\frac{\vec{l_h}}{1}$; the Miller indices are thus (4,1).

scaled such that they occur at $\frac{\vec{l_h}}{i}$ and $\frac{\vec{l_v}}{j}$. It is interesting to note that the Miller indices can be thought of as a scalar multiple of the components of the normal vector to a face in the coordinate system defined by the (possibly non-orthogonal) basis vectors $\vec{l_h}$ and $\vec{l_v}$. In the case where the lattice is square, the Miller indices are thus equal to the components of a vector normal to the face, pointing into the crystal. (In computer graphics, normals are usually thought of as pointing out of an object. In this paper, the computer graphics convention is adopted; the Miller indices are thus the opposite of the components of the vector normal of a face.)

In crystallography, the Miller indices are generally scaled so that they are both integers. For our purposes, it is more convenient to scale them such that one of them is exactly 1 and the other is either 0 or a number with magnitude greater than or equal to 1. Thus, a horizontal face is an instance of the (0,1) face.

Because biclinic crystals are considered, a face with both intercepts having the same sign is different than a face where the intercepts have differing signs. In this paper, such faces are denoted by letting the first Miller index i be negative. For example, figure 2.3 is an illustration of the (–3, 1) face. Using our conventions, all faces can be written as $(1,n)$ or $(n,1)$ (or $(-1,n)$ or $(-n, 1)$) where $n \in [1,\infty)$, except for the horizontal (0,1) face and the vertical (1,0) face.

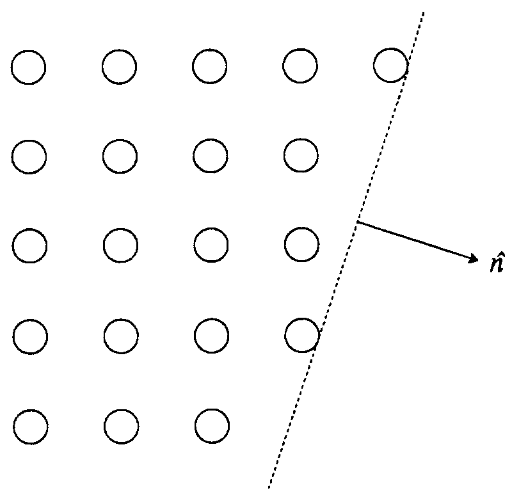


Figure 2.3. This figure shows the (3,-1) face.

2. Etch Rates at Key Face Orientations

As the first step in deriving an etch rate curve from the eight bond strength parameters discussed above, the etching rate for several ''key'' face orientations is first analyzed. Eight such key orientations are analyzed: the orientations that are expected to correspond to local minima on the etch rate curve (which are the (0,1), (1,1), (1,0), and (−1,1) faces), and the orientations that are expected to correspond to local maxima (which are the (1,2), (2,1), (−2,1), and (−1, 2) faces). In this analysis, only exposed surface atoms are considered. The assumption is made that the probability that an atom gets removed is the combination of the probabilities that all the bonds that hold it in place get removed.

2.1. The (0,1) Face

First let us consider the (0,1) face, pictured in figure 2.4. Initially, each atom is held in by three bonds: two a_h bonds and a d_v bond. From the assumptions presented earllier, we conclude that the probability of an atom being etched away from a flawless surface per unit time is $a_h{}^2 d_v$.

Once a surface atom is etched away, its two neighboring atoms are much more exposed; one would therefore expect them to etch away much more quickly. This process can be thought of as a surface atom being initially ''attacked'', and once it is gone, a layer of atoms being stripped off relatively quickly. A snapshot of the process of stripping away a layer is shown in figure 2.5. The probability of the left corner atom being etched away is $a_h b_v$, and the probability of the right corner atom being etched away is $a_h c_v$.

While a layer is being stripped away, other surface atoms are being attacked. To analyze this process, let us denote the probability of a surface atom being attacked as ε, and the two corner atom probabilities as α_1 and α_2, that is, $\varepsilon = a_h{}^2 d_v$, $\alpha_1 = a_h b_v$, and $\alpha_2 = a_h c_v$. Assume that there are simultaneous attacks every D atoms apart. It would take $T_1 = \dfrac{D}{\alpha_1 + \alpha_2}$

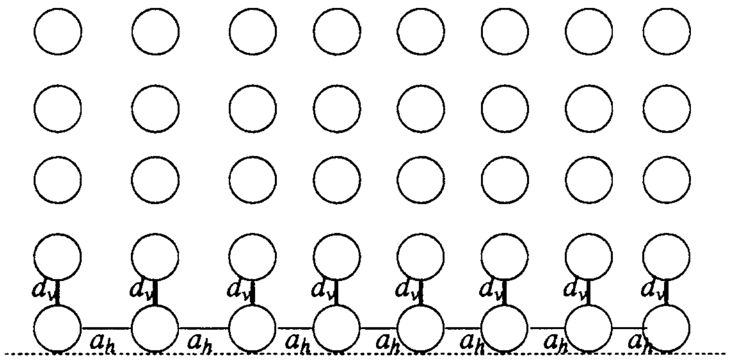


Figure 2.4. This figure shows the (0,1) face, and the bonds relevant to the removal of a surface atom.

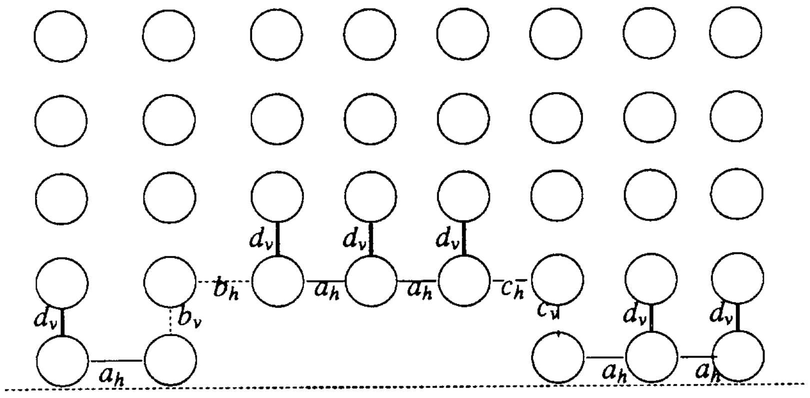


Figure 2.5. This figure shows the (0,1) face shortly after a surface atom has been removed. Note that this has caused two corner atoms to be exposed.

time units to remove one layer. Over D atoms, we expect an exposed atom to be attacked every $T_2 = \frac{1}{D\varepsilon}$ time units. We equate T_1 and T_2 to find D:

$$\frac{1}{D\varepsilon} = \frac{D}{\alpha_1 + \alpha_2}$$

$$D = \sqrt{\frac{\alpha_1 + \alpha_2}{\varepsilon}}$$

The effective rate of material removal r (in atom layers per second) is:

$$r = \frac{1}{T_1} = \frac{1}{T_2} = D\varepsilon = \sqrt{\varepsilon\,(\alpha_1 + \alpha_2)}$$

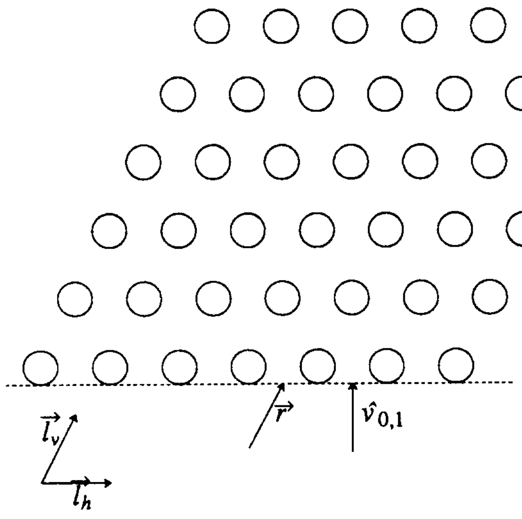


Figure 2.6. This figure shows the (0,1) face of a biclinic crystal. $\hat{v}_{0,1}$ is the unit vector in the direction in which the edge is moving—it is perpendicular to the face. $\vec{r}$ is the product of the atom layer removal rate r with $\vec{l_v}$.

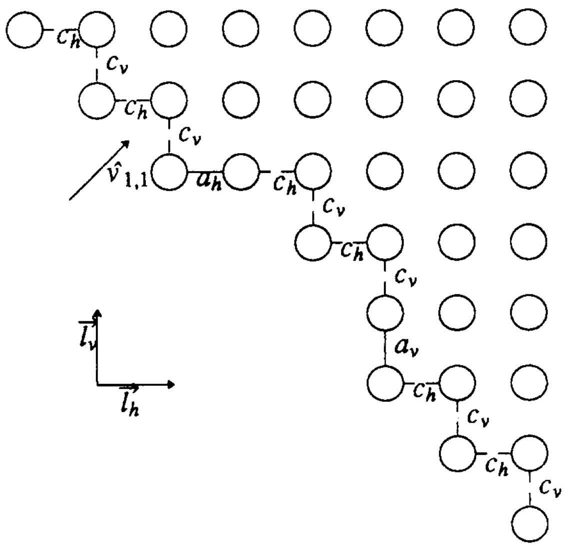


Figure 2.7. This figure shows the (1,1) face of a crystal, shortly after the surface has been attacked.

The time to remove one layer of atoms can conveniently be converted to a velocity vector, denoted by $\vec{v}_{1,0}$. Referring to figure 2.6, the removal of one layer of atoms can be thought of as shifting the face by the vector $\vec{l}_v$. Projecting this vector onto the unit vector in the direction of etching $\hat{v}_{0,1}$ and multiplying by r results in the rate of etching in the (0,1) direction $v_{0,1} = \vec{r} \cdot \hat{v}_{0,1}$. Summarizing and collecting terms, this becomes

$$v_{0,1} = \hat{v}_{0,1} \cdot \vec{l}_v \sqrt{(a_h b_v + a_h c_v)\, a_h^2 d_v}$$

2.2. The (1,0) Face

The (1,0) face follows the same analysis as above. The final expression for the etch rate is:

$$v_{1,0} = \hat{v}_{1,0} \cdot \vec{l}_h \sqrt{(a_v b_h + a_v c_h)\, a_v^2 d_h}$$

2.3. The (1,1) Face

Figure 2.7 shows the (1,1) face. The situation here is analogous to that of the (0,1) face: there are relatively protected "surface" atoms which are attacked, and when one is removed there are two more exposed "corner" atoms. The rate of removal of one layer of atoms r can be obtained from the same formula $r = \sqrt{(\alpha_1 + \alpha_2)\, \varepsilon}$, where in this case the probability of a surface atom attack $\varepsilon = c_v c_h$, and the probabilities of the two corner atoms are given by $\alpha_1 = a_v c_h$ and $\alpha_2 = a_h c_v$. The removal of one layer of atoms may be viewed as shifting the face by $\vec{l}_h$ (or by $\vec{l}_v$—either formulation will derive the same result). Projecting onto the unit vector in the direction of etching $\hat{v}_{1,1}$ and combining terms as before, this becomes

$$v_{1,1} = \hat{v}_{1,1} \cdot \vec{l}_h \sqrt{(a_v c_h + a_h c_v)\, c_v c_h}$$

2.4. The (-1,1) Face

The (-1,1) face has an analysis very similar to that of the (1,1) face, except that the corner bonds are the more exposed "b" bonds instead of being "c" bonds. For the (-1,1) face, $\varepsilon = b_v b_h$, $\alpha_1 = a_v b_h$ and $\alpha_2 = a_h b_v$. This results in a face velocity of

$$v_{-1,1} = \hat{v}_{-1,1} \cdot (\vec{-l}_h) \sqrt{(a_v b_h + a_h b_v)\, b_v b_h}$$

2.5. The (2,1) Face

The (2,1) face, pictured in figure 2.8, is very different from any of the faces analyzed so far. Here, there are a large number of very exposed corner atoms (pictured as double circles). Each such corner atom has a relatively high probability of being etched away of $a_v c_h$. When one of these atoms is etched away, the immediate neighborhood consists of more protected atoms. For this reason, the disappearance of one atom does not cause a layer of atoms to be stripped away. Rather, each corner atom can be considered to be removed independently.

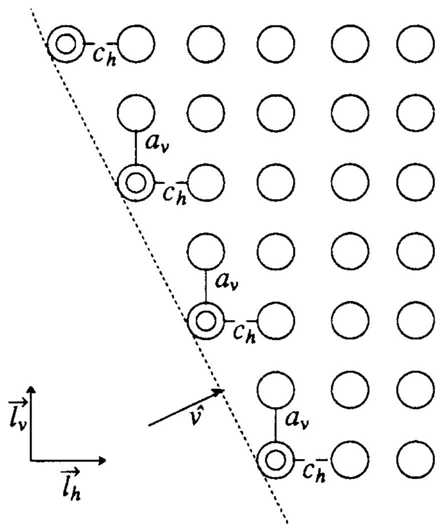


Figure 2.8. This figure shows the (2,1) face of a crystal, as well as the bonds that are considered in the derivation of the etch rate. Only the corner atoms (those represented as double circles) are considered in the analysis. $\hat{v}$ is the unit vector in the direction of etching.

In this model, it is assumed that the corner atoms are etched away at a uniform rate. Each time step T, one "layer" of corner atoms is etched away. The length of a time step is given by $T = \frac{1}{a_v c_h}$. This etching results in the same atomic structure, displaced by $\vec{l}_v$. As before, this displacement can be projected onto the unit normal in the direction of etching $\hat{v}$ and multiplied by the rate of removal of one layer. This done, the etch rate in the (2,1) direction becomes

$$r_{2,1} = a_v c_h \hat{r}_{2,1} \cdot \vec{l}_v$$

2.6. The (1,2), (-1,2) and (-2,1) Faces

The (1,2), (-1,2) and (-2,1) faces have analyses very similar to that of the (2,1) face. The resulting etch rates are listed below:

$$r_{1,2} = a_h c_v \hat{r}_{1,2} \cdot \vec{l}_h$$

$$r_{-1,2} = a_h b_v \hat{r}_{-1,2} \cdot (-\vec{l}_h)$$

$$r_{-2,1} = a_v b_h \hat{r}_{-2,1} \cdot \vec{l}_v$$

2.7. Results/Discussion

With the plausible assumptions that $a_h > b_h \geq c_h > d_h$ and $a_v > b_v \geq c_v > d_v$ and with reasonable choices for the bond breaking probabilities (i.e. $a_h \gg b_h$), the (1,0), (1,1), (0,1) and (−1,1) faces will etch relatively slowly, whereas the (2,1) and (1,2) type faces will etch relatively quickly. This is to be expected, since the etching process is controlled by well-protected atoms in the first case, and by relatively unprotected corner atoms in the second. This is illustrated in Figure 2.9, which shows the etch rates for the 16 key faces.

The 16 points for which the etch rates have been determined correspond to the expected extrema of the etch rate curve according to the model. This model has two kinds of etching processes. At the (2,1), (1,2) and similar faces the etching is controlled by fast corner atoms. These faces correspond to maxima on the etch rate curve, because it is on these faces that there is the highest concentration of corner atoms. At the (1,0), (1,1) and similar faces there is a slower process where a surface atom must first be attacked. After one atom is attacked two corner atoms are exposed that etch at a faster rate, leading to a layer-by-layer stripping of the surface. These faces correspond to the minima of the etch rate curve.

It is interesting to note that perfectly isotropic etcing can be approximated. If a square lattice is assumed (i.e. $\vec{l_h}$ = <1,0> and $\vec{l_v}$ = <0,1>), for symmetry considerations one can set:

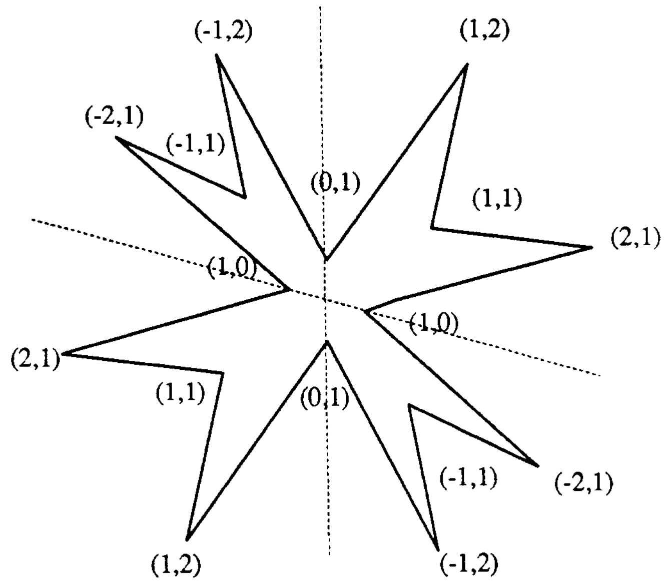


Figure 2.9. This figure shows the etch rates of the key faces for some typical lattice parameters. The etch rates have been connected by straight lines. The actual lattice parameters used were $\vec{l_h}$ = <1,0>, $\vec{l_v}$ = <0.3,1>, $a_h = a_v = 0.2$, $b_h = b_v = 0.01$, $c_h = c_v = 0.008$, and $d_h = d_v = 0.0001$. The resulting rates were $r_{1,0} = 1.15 \cdot 10^{-4}$, $r_{2,1} = 7.84 \cdot 10^{-4}$, $r_{1,1} = 3.71 \cdot 10^{-4}$, $r_{1,2} = 8.11 \cdot 10^{-4}$, $r_{0,1} = 1.20 \cdot 10^{-4}$, $r_{-1,2} = 7.97 \cdot 10^{-4}$, $r_{-1,1} = 3.86 \cdot 10^{-4}$ and $r_{-2,1} = 7.81 \cdot 10^{-4}$.

$a_h = a_v$, $b_h = b_v = c_h = c_v$, and $d_h = d_v$. This leaves three unknowns: a, b and d. Setting the key etch velocities equal, two equations are obtained: $r_{1,0} = r_{2,1}$, and $r_{1,0} = r_{1,1}$ (equations involving the other key rates are equivalent to the two just given). Now one can solve these equations for a and d in terms of b:

$$r_{1,0} = \sqrt{2a^3bd}$$

$$r_{1,1} = \sqrt{ab^3}$$

$$r_{2,1} = \frac{\sqrt{5}\,ab}{5}$$

$$\sqrt{ab^3} = \frac{\sqrt{5}\,ab}{5}$$

$$ab^3 = \frac{a^2b^2}{5}$$

$$a = 5b$$

$$\sqrt{2a^3bd} = \sqrt{ab^3}$$

$$d = \frac{ab^3}{2a^3b} = \frac{b^2}{2a^2} = \frac{1}{50}$$

It is rather surprising that the value of d is independent of the values of a and b. This is due to the fact that the probability of removal of a surface atom on the (1,0) face is modeled as the product of the probabilities of three bonds breaking, whereas the other atoms are modeled as the product of the probabilities of two bonds breaking. If all bond probabilities were scaled up uniformly there would be a relative increase in the etch rate of the (1,0) face. Finally, it must be noted that these equations set the key etch rates equal, but do not necessarily minimize the etch rate deviations for the orientations in between them.

3. Interpolation Between Key Face Orientations

To drive an etching simulation one needs to have the etch rate curve defined at every point, not just at a few key face orientations. Of course, one could simply do linear interpolation (in polar coordinates) between the key points—in effect, connecting the known points on the graph with arcs. For this project, however, a more detailed analysis at what happens at other face orientations was required. In this section the etching process at orientations between key orientations is studied.

3.1. Orientations Between (1,0) and (2,1)

Faces at orientations between (1,0) and (2,1) can be represented by Miller indices $(n,1)$ where $n \in (2,\infty)$. A typical example, pictured in figure 2.10, is the (4,1) face. From the figure, one can see that there are fast corner atoms, with the same bond pattern as was seen on the (2,1) face, as well as atoms with the same bond patterns as was seen on the (1,0) face (these atoms are pictured as triple and double circles, respectively). There is also one other surface atom (with an *acd* bond pattern), but since this atom was ignored on the (2,1) face, it will also be ignored here.

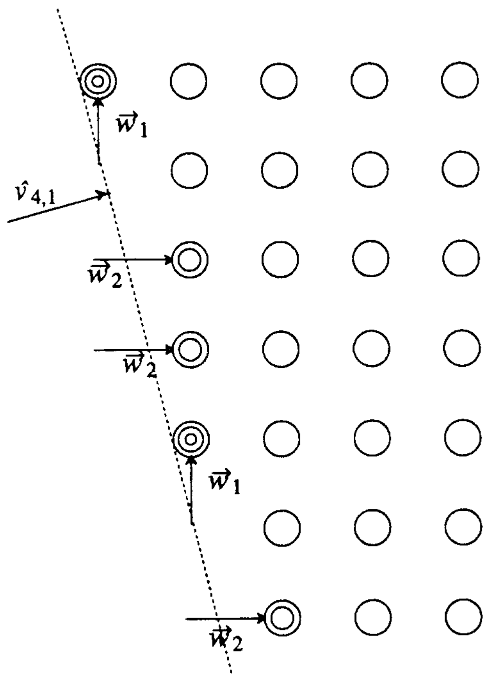


Figure 2.10. This figure shows the (4,1) face, and the vectors $\vec{w}_1$ and $\vec{w}_2$ that represent contributions to the etch rate. $\hat{v}_{4,1}$ is the unit vector in the direction of etching. Triple circles represent quickly etching "corner" atoms, and double circles slow "surface" atoms.

The etching of a surface such as this is inherently a probabilistic process, and is difficult to analyze, especially considering that this would have to be done for a whole family of faces. Given some of the other simplifying assumptions that have already been used in this model, it does not seem that such an analysis is justified. Instead, a further assumption is made: the contribution to the etch rates from the (2,1)-like corner atoms and from the (1,0)-like surface atoms can be computed separately and summed. This neglects the fact that in the analysis of the (1,0) face a relatively long surface over which to operate was assumed. We believe that this effect is negligable, especially when compared to some of the other simplifying assumptions made in this model.

The first contribution analyzed is that of the corner atoms. As on the (2,1) face, the etching away of a corner atom can be thought of as displacing the edge by one atom vertically. The velocity vector $\vec{w}_1$ that represents the speed at which the edge moves in the vertical direction can be calculated. To determine the rate at which the edge actually moves, this vector is projected onto the unit normal in the direction of etching $\hat{v}_{4,1}$. An expression for $\vec{w}_1$ can be obtained by noting that $r_{2,1} = \vec{w}_1 \cdot \hat{v}_{2,1}$. Generalizing for the $(n,1)$ face for any $n \in (2,\infty)$, an expression for the contribution to $r_{n,1}$ from corner atoms is

$$r_{n,1,corner} = \frac{r_{2,1}}{\vec{l}_v \cdot \hat{v}_{2,1}} \vec{l}_v \cdot \hat{v}_{n,1}$$

Note that as n approaches infinity (i.e. as the orientation of the face approaches the (1,0) orientation) $r_{n,1,corner}$ approaches 0, because $\vec{l}_v$ and $\hat{v}_{1,0}$ are orthogonal. As n approaches 2, this expression yields exactly $r_{2,1}$.

For the (1,0)-like surface atoms, more of an approximation must be made. The assumption is made that each surface atoms adds a contribution to the velocity of $\vec{w}_2 = \vec{r}_{1,0}$, which is projected onto $\hat{v}_{4,1}$. To find the net contribution of such atoms over the entire surface, this contribution is scaled by the proportion of atoms that have the (1,0) bond pattern. In Figure 2.10, one can see that half of the atoms are like this. In general, the proportion of (1,0) like atoms will be $\frac{n-2}{n}$. The contribution from (1,0)-like surface atoms is thus

$$r_{n,1,surface} = \frac{n-2}{n} r_{1,0} \hat{v}_{1,0} \cdot \hat{v}_{n,1}$$

Note that as n approaches infinity, this expression approaches $r_{1,0}$ and as n approaches 2, it approaches 0.

Summing the surface and corner contributions, this becomes

$$r_{n,1} = \frac{r_{2,1}}{\vec{l}_v \cdot \hat{v}_{2,1}} \vec{l}_v \cdot \hat{v}_{n,1} + \frac{n-2}{n} r_{1,0} \hat{v}_{1,0} \cdot \hat{v}_{n,1}$$

for any $n \in (2,\infty)$.

3.2. Orientations Between (2,1) and (1,1)

Faces at orientations between (2,1) and (1,1) can be represented by Miller indices (n,1) where $n \in (1,2)$. A typical example, pictured in Figure 2.11, is the (1.5,1) face. As was the case in the previous analysis, there are fast (2,1)-like corner atoms, as well as slow surface atoms, this time with bond patterns like those seen on the (1,1) face.

Again examing the contribution from the corner atoms first, the removal of the corner atoms is viewed as a displacement along the (1,1) direction by one atom. In this case, that's a displacement of $\vec{l}_h - \vec{l}_v$. In figure 2.11, $\vec{w}_1$ represents the velocity in that direction due to the removal of corner atoms. $\vec{w}_1$ can be derived from $r_{2,1}$ with a formula similar to that of the previous section. The resulting contribution to $r_{n,1}$ from the etching of corner atoms is

$$r_{n,1,corner} = \frac{r_{2,1}}{(\vec{l}_h - \vec{l}_v) \cdot \hat{v}_{2,1}} (\vec{l}_h - \vec{l}_v) \cdot \hat{v}_{n,1}$$

where $n \in [1,2]$.

At the slow (1,1)-like surface atoms, the same approximation is made as before. There is one slight complication, though: the Miller index n does not directly tell us the proportion of surface atoms. To calculate this, the "cell size" m is considered. m represents the average height of the cell between two corner atoms. In figure 2.11 the cell size is 3. A general expression for the cell size is $\frac{1}{n-1}+1$. The proportion of surface atoms is $\frac{m-2}{m}$, or

$$\frac{m-2}{m} = \frac{\frac{1}{n-1}-1-2}{\frac{1}{n-1}-1} = \frac{2-n}{n}$$

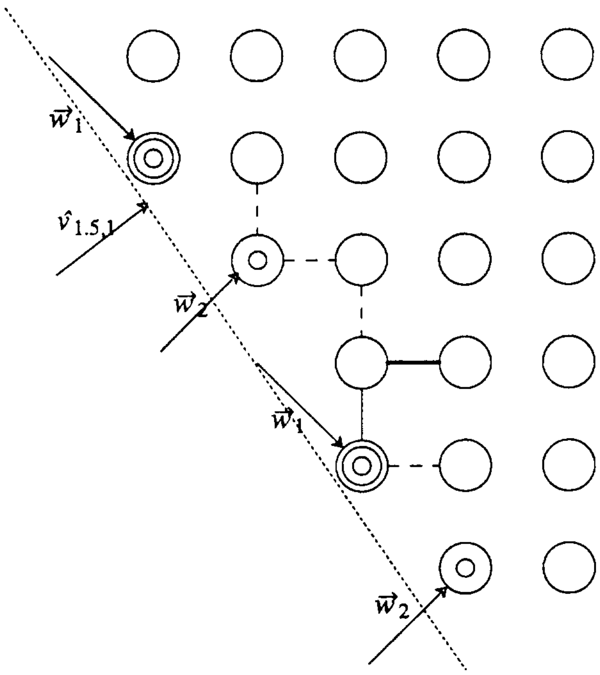


Figure 2.11. This figure shows the (1.5,1) face, and the vectors $\vec{w}_1$ and $\vec{w}_2$ that represent contributions to the etch rate. $\hat{v}_{1.5,1}$ is the unit vector in the direction of etching. Triple circles represent fast "corner" atoms, and double circles slower "surface" atoms.

Using this, the contribution to $r_{n,1}$ from surface (1,1)-like atoms is

$$r_{n,1,surface} = \frac{2-n}{n} r_{1,1}\hat{v}_{1,1}\cdot\hat{v}_{n,1}$$

Summing the surface and corner contributions, this becomes

$$r_{n,1} = \frac{r_{2,1}}{(\vec{l}_h-\vec{l}_v)\cdot\hat{v}_{2,1}}(\vec{l}_h-\vec{l}_v)\cdot\hat{v}_{n,1} + \frac{2-n}{n} r_{1,1}\hat{v}_{1,1}\cdot\hat{v}_{n,1}$$

for any $n \in (1,2)$. Note that at the boundaries of this interval, this equation yields $r_{1,1}$ and $r_{2,1}$ as appropriate.

3.3. Other Orientations

The analysis between other pairs of key face orientations is similar to the two above analyses. The corresponding expressions can be derived by proper substitution of the variables.

4. Results/Discussion

Based on the above analyses, two programs called **etchkey** and **etchinter** have been developed. Together, they calculate a complete etch rate diagram. **etchkey** accepts bond strength parameters, and derives the etch rate for the key face orientations. **etchinter** interpolates between these etch rates to form the complete etch rate diagram. The result of applying these programs to some typical bond strength parameters is shown in figure 2.12.

The interpolation program actually produces a list of points along the etch rate curve. Within the actual etching program (discussed later in this paper) a linear polar-coordinate interpolation is done between adjacent points. To produce a plot, the arcs produced by this linear interpolation are approximated with straight lines. In figure 2.12, there are four interpolated rate points between adjacent key directions (i.e. each pair of key rates is connected by five arcs). Each arc is approximated by eight straight lines.

An interesting etch rate diagram is shown in figure 2.13. This is the diagram derived from the isotropic etching approximation developed earlier. This is not the closest approximation to perfectly isotropic etching that is possible with our model; some kind of least-squares analysis would probably result in a curve that is slightly closer to a perfect circle.

In our implementation the calculation of the key etch rates and the interpolation between these rates has been carefully decoupled. Because of this decoupling, if one has access to etch rates for the key faces that were measured experimentally or derived through a

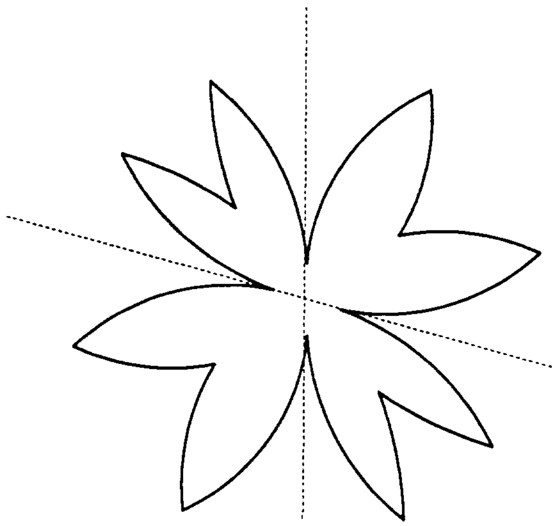


Figure 2.12. This figure shows the complete etch rate diagram for some typical lattice parameters. The actual lattice parameters used were $\vec{l}_h$ = <1,0>, $\vec{l}_v$ = <0.3,1>, $a_h = a_v = 0.2$, $b_h = b_v = 0.01$, $c_h = c_v = 0.008$, and $d_h = d_v = 0.0001$.

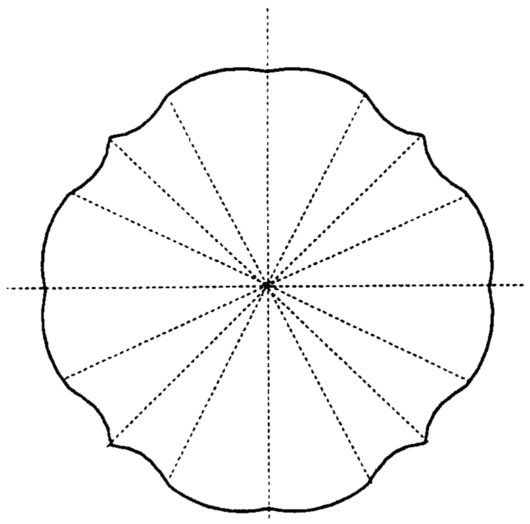


Figure 2.13. This figure shows the etch rate diagram that results when we approximate perfectly isotropic etching by setting the key etch rates equal.

different theory, one can still use the interpolation module developed above.

The model developed above is very simplistic, and contains several approximations. The simple model of bonds that break in a probabilistic manner is probably a poor approximation to the actual mechanisms in any real etching process. We believe, however, that the above analysis is good enough to give reasonable and self-consistent etch rate diagrams for the purpose of simulating the etching process.

CHAPTER 3

2D Local Geometry Analysis

In this chapter it is assumed that the etch rate for any planar crystal face of arbitrary orientation is known, either from expermentally determined data or from the analysis in Chapter 2. Thus, when etching a shape, faces can simply be advanced at the corresponding etch rate. It has been demonstrated experimentally, however, that faces of different orientations that were not present previously can develop at the corners of an object being etched. It is the goal of this chapter to model how that occurs, and develop algorithms that determine exactly what faces have a chance to appear, given a particular starting geometry and etch rate function.

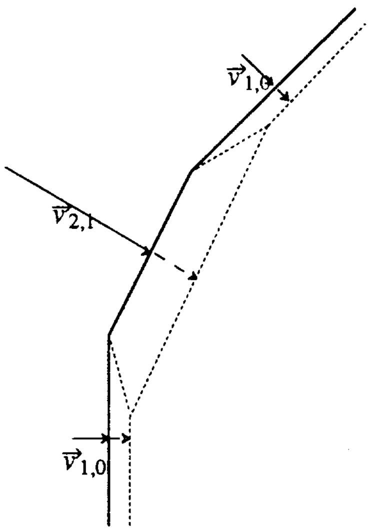


Figure 3.1. This figure shows a face at the quickly etching (2,1) orientation between two more slowly etching faces. The dotted line represents the shape that results after etching for a short period of time. The length of the (2,1) face increases at the expense of the other faces over time.

1. Candidate Faces

At a corner one can imagine that there are an infinite number of microfaces with orientations distributed between the two orientations of the incident faces. Experience, however, shows that at a corner only a few select face orientations will form (in addition to the pre-existing faces). The set of faces that might develop at a corner is called the set of "candidate faces."

At a convex corner, one would expect faces that etch away faster to dominate. Referring to figure 3.1, the (2,1) face etches much more quickly than the surrounding (1,0) and (1,1) faces. This causes its length to increase at the expense of the surrounding faces. If all zero-length microfaces at a corner are considered, one would expect the microfaces with orientations at the maxima of the etch rate curve to increase in length over time. By a similar argument, at a concave corner one would expect the microfaces with orientations at the minima of the etch rate curve to dominate. But is it sufficient to simply consider the extrema? It turns out that in general it is not, although in most cases considering only the extrema is a very close approximation.

To predict at which orientations a microedge will grow to finite size, Frank's two theorems on crystal dissolution are useful. [Frank, '58] The first theorem states that if the rate of etching of a crystal surface at any given time is a function only of orientation, then during etching, the locus of an elemental area of crystal surface with a particular orientation is a straight line. The second theorem states that if the etching velocity is defined as measured normal to the actual crystal surface (as we have done), and the polar diagram of the reciprocal of the etching velocity as a function of orientation is constructed, then the trajectory of an elemental area of crystal surface of given orientation is parallel to the normal to the polar diagram at the corresponding orientation. [Barber, '73] A polar diagram of the reciprocal of the etching velocity is called a "slowness curve".

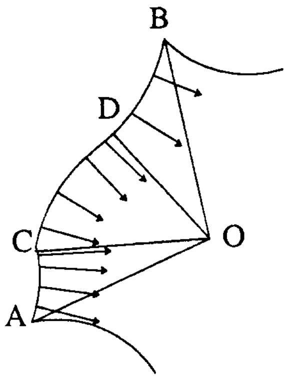


Figure 3.2. This figure shows an example of a portion of a slowness curve, along with normal vectors to the curve.

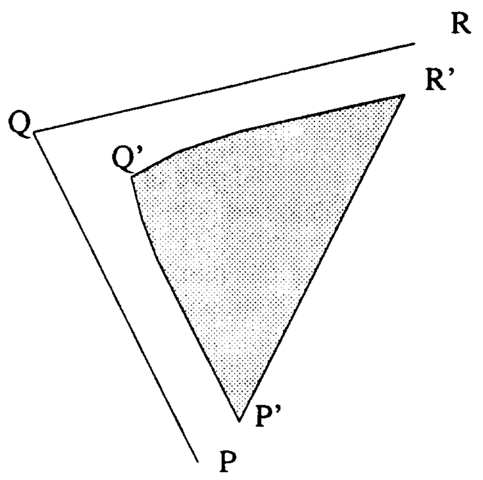


Figure 3.3. This figure shows an example of a crystal, and shows the result of etching that crystal according to the slowness curve pictured in figure 3.2. P'Q'R' is the contour of the crystal after etching for a short time.

This result can tell us directly what orientations of microfaces will survive at a corner. Consider the slowness curve shown in figure 3.2. Etching a crystal with edge PQR (figure 3.3) according to the slowness curve pictured in figure 3.2, the faces initially present are normal to OA and OB. Faces of orientation between these are potentially present at Q. Those having normals between OC and OD have converging trajectories. Being together at the start, they cannot spread apart. Faces normal to OC and OD have parallel trajectories; along this line, therefore, we find both these faces, with a sharp edge between them. Outside OC and OD we have diverging trajectories; on either side of the edge we therefore expect a curved surface.

The above argument applies to a convex corner; at a concave corner the diagram is the same except that the normals to the slowness curve point outwards. The same criterion applies: diverging normal vectors imply that faces of the corresponding orientation will appear.

For our purposes, we are interested in determining which miroface orientations need to be considered for addition at a corner, based on "competition" of the microface with faces of nearby orientation. From Frank's theorems, we conclude that at a convex corner, regions where the slowness curve has concave curvature correspond to orientations of microfaces that must be added; at a concave corner, regions where the slowness curve is convex correspond to the orientations of microfaces that must be added. A region that is neither concave nor convex (i.e. the slowness curve is a straight line) corresponds to orientations of microfaces that don't have to be added at either a concave or a convex corner.

2. Application to our Etching Model

The slowness curve corresponding to the etch rate curve of figure 2.12 is shown if figure 3.4. At the maxima of the slowness curve (i.e. at slowly-etching orientations) outward-pointing normals clearly diverge. This confirms that orientations corresponding to minima on the etch rate curve will develop at concave corners. Similarly, at the minima of the slowness curve (i.e. at quickly-etching orientations) inward-pointing normals diverge, confirming that orientations corresponding to maxima on the etch rate curve will develop at convex corners.

The non-extremal regions of the curve are very nearly straight lines. This means that normals to the etch rate curve are nearly parallel. It turns out that some of these regions are slightly convex, while some are slightly concave. This means that some corners will become slightly rounded during etching. The curvature is so slight, however, that this effect is not noticeable. In other words, given our underlying etch rate model, the set of candidate faces at a corner can be restricted to those orientations at the extrema of the etch rate curve with little loss of accuracy.

An interesting limiting case to discuss is a perfectly isotropic etch rate diagram. In this case, there are no dominant faces at a convex corner. No new faces develop at corners as the

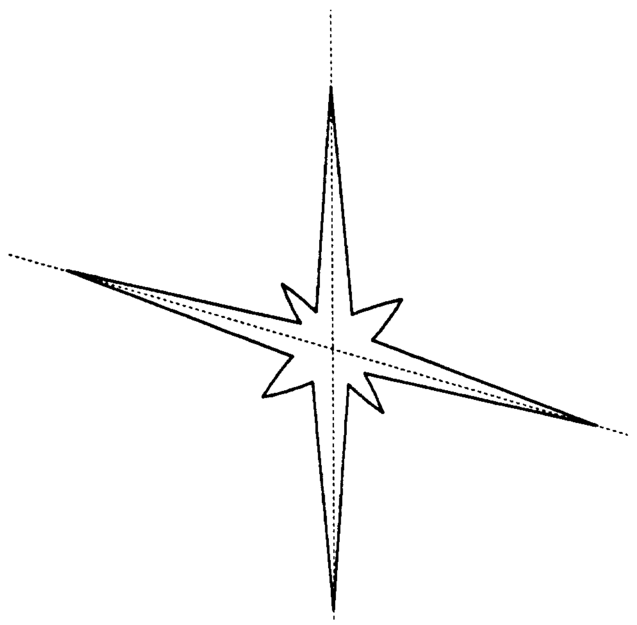


Figure 3.4. This figure shows the slowness curve corresponding to the etch rate curve of figure 2.12. Note that between extrema the curve closely approximates straight line segments.

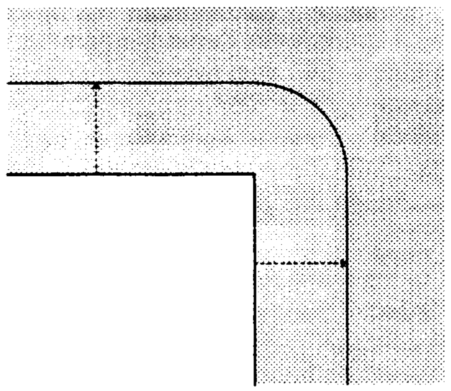


Figure 3.5. This figure shows what would be expected when a concave corner is subjected to perfectly isotropic etching. The thin line represents the object boundary after etching for a short period of time. Notice that the sharp corner becomes a curve.

etching process continues. At a concave corner, however, all orientations between the neighboring faces appear. The result is that a corner develops into a semi-circular arc.

3. Dominant Faces

The above analysis provides the set of candidate faces at a given corner. Not all members of this set, however, will necessarily appear in the object being etched. Whether a particular face develops or not depends on the local geometry.

Consider the concave corner pictured in Figure 3.6. For face c to grow the vertex velocity slowness vectors $\vec{s}_{a,c}$ and $\vec{s}_{c,b}$ must diverge. Recall that the slowness vector of a vertex is the perpendicular to the chord joining the slowness vectors of the two adjacent faces. One can thus see that $\vec{s}_{a,c}$ and $\vec{s}_{c,b}$ can only diverge if $\vec{s}_b$ extends beyond the chord joining $\vec{s}_a$ and $\vec{s}_b$. In general, to determine which virtual faces will grow at a given corner, a polar diagram of the slowness vectors of the two faces and all intermediate orientations can be considered. If the convex hull around all of these points is formed, the vertices of the outside half of the convex hull (i.e. the part of the convex hull that is further from the origin) correspond to orientations of virtual faces that will grow (see Figure 3.7).

At a convex corner, the vertex velocity vectors must again diverge. This situation is pictured in Figure 3.8. In this case, the slowness vector $\vec{s}_c$ must not project beyond the chord joining $\vec{s}_a$ and $\vec{s}_b$. This can be understood intuitively by recognizing that at convex corners, fast etching faces (i.e. short slowness vectors) are favored. To compute which virtual faces will grow at a convex corner, a polar diagram of the slowness vectors of the two faces and all intermediate orientations can again be considered. The vertices of the inside half of a convex hull constructed around these points (Figure 3.9) will correspond to orientations of virtual faces that will grow.

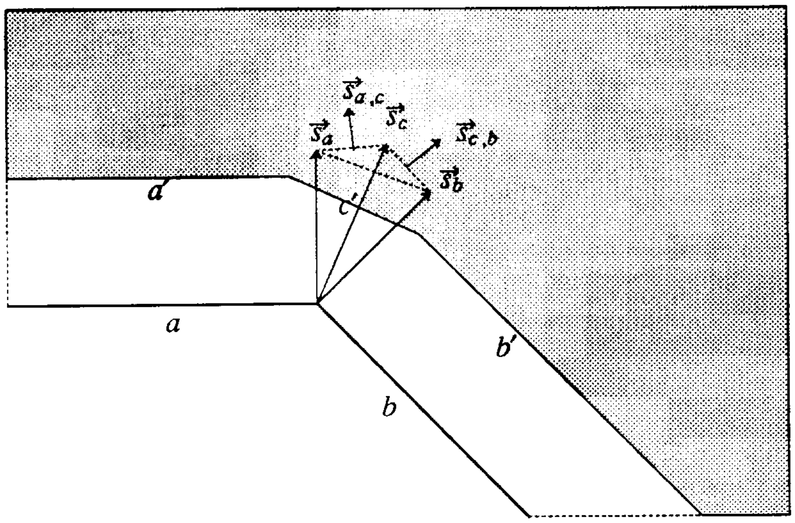


Figure 3.6. This shows a concave corner where the virtual face c grows over time. c will only grow if its slowness vector $\vec{S}_c$ projects beyond the chord joining $\vec{S}_a$ and $\vec{S}_b$.

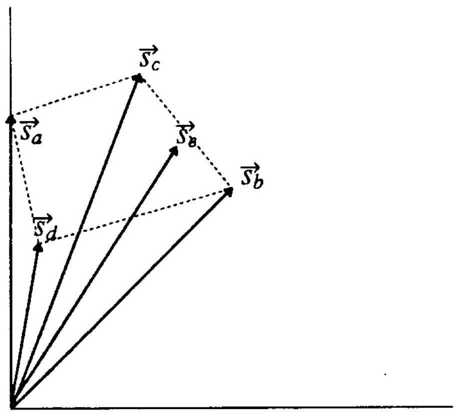


Figure 3.7. This shows the slowness vectors of two faces a and b that form a concave corner, and several intermediate slowness vectors. To determine which of these vectors correspond to virtual faces that will grow over time, the convex hull around the endpoints of all of the slowness vectors is formed. Every vertex on the outside half of the convex hull corresponds to a virtual face that will grow over time.

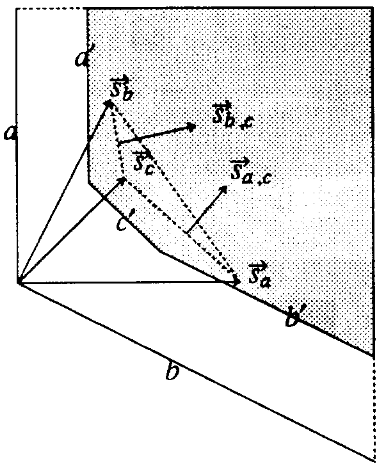


Figure 3.8. This shows a convex corner where the virtual face c grows over time. c will only grow if its slowness vector $\vec{s}_c$ does not project beyond the chord joining $\vec{s}_a$ and $\vec{s}_b$.

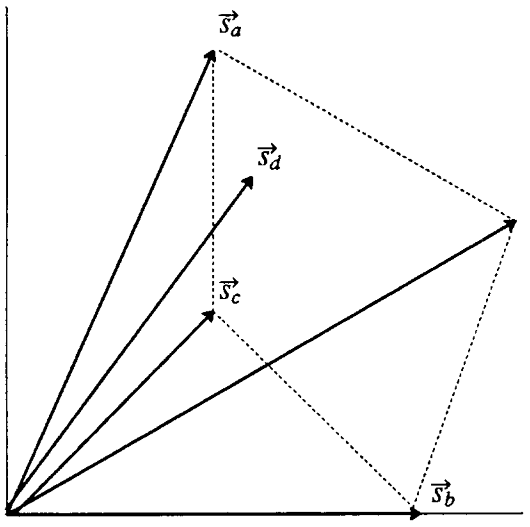


Figure 3.9. This shows the slowness vectors of two faces a and b that form a convex corner, and several intermediate slowness vectors. To determine which faces will grow over time, the convex hull around the endpoints of the slowness vectors is formed. Every vertex on the inside half of the convex hull corresponds to a virtual face that will grow over time.

CHAPTER 4

2D ETCHING SIMULATOR IMPLEMENTATION/RESULTS

The etching simulator was broken into four distinct modules, reflecting the logical divisions in the analysis. This allows the use of only selected parts, if desired. Each functional module shall be discussed in turn.

1. 2detch_rate

The "**2detch_rate**" program implements the model developed in Chapter 2. Supplied with bond strength probabilities, it calculates
the etch rates at all key directions. It then interpolates between the key etch rates. This is not simply a liner interpolation; interpolation is done according to the formulae developed in the latter half of Chapter 2. The number of sample points is a user-specifiable parameter.

2. 2detch_faces

The "**2detch_faces**" program analyzes a curve to determine which regions of the slowness curve have concave curvature, and which regions have convex curvature. This determines what orientations will be included in the set of candidate faces at a corner. As was noted earlier, the slowness diagrams of the etch rate functions produced by **2detch_rate** are very nearly straight lines, so it is sufficient to only include the extrema of the etch rate function. This is not true for all etch rate curves, however. A user is permitted to feed any etch rate curve into **2detch_faces**, not just curves developed with the other two modules. Given any etch rate curve, the **2detch_faces** program outputs a list of candidate face orientations for concave corners, followed by a list of candidate face orientations for convex corners.

The program works by first finding all local extrema of the etch rate curve. Every extremum is tested to see if it is concave or convex. (If it is neither, it is arbitrarily classed as concave—due to floating point inaccuracies, this is an arbitrary choice. It could be classed as convex with little effect on the final result). The region between each pair of successive extrema is then split by evenly distributing a user-specified number of points between them. Each of these points is tested to see if the curve is concave or convex at the given point.

A numerical technique must be used to test for concavity of the slowness curve. To perform this test, the program examines the magnitude of the slowness curve at two angles that lie some small $\Delta\theta$ away from the given orientation. $\Delta\theta$ is chosen to be half the interval between the orientations supplied to **2detch_faces**. A straight line is then constructed between the points on the slowness curve at $\theta = \phi+\Delta\theta$ and $\theta = \phi-\Delta\theta$, where ϕ is the orientation in question. The value of r on this line is computed at $\theta = \phi$, and is compared to the value of the slowness curve at ϕ. If the point on the line has greater r, the curve is concave at that point.

One interesting implementation issue was the calculation of the etch rate at an arbitrary orientation between the points explicitly supplied in the input. The etch rate is input as a list of (rate, orientation) pairs, so **2detch_faces** must interpolate between these points to come up with a continuous function. Linear interpolation in polar coordinates was used for this. For efficiency, the interpolation points are maintained in a sorted array, and a binary search is used to find the bounding interpolation points. This results in an etch rate calculation that is $O(\log n)$. This could be further improved to $O(n)$ with hashing.

One consequence of the linear polar coordinate interpolation is that values of the etch rate curve must be supplied to **2detch_faces** to at least as fine a grade as **2detch_faces** checks for important orientations. If it isn't (i.e. if the etch rate function is undersampled), then **2detch_faces** will be testing the arcs that result from its own linear interpolation for concavity at some points. This arc will always be convex, regardless of the underlying etch rate function.

3. Etching Simulator

The final module is the actual etching simulator. This program (called **2detch**) accepts an initial shape, and a model of the etch rate function such as **2detch_rate** and **2detch_faces** produce. This program then produces a file describing the shape into which the original shape is transformed after etching for a specified period of time. The simulator is also capable of producing a line-drawing animation of the etching process (under X windows). The shape being etched is input and output in the Unigrafix language.

The program begins by reading in the etch rate data and the initial shape. As was noted earlier, new faces may appear at corners. The output of **2detch_faces** contains a list of candidate face orientations. The convex hull algorithm developed in the previous chapter is used to determine which of these candidate faces will actually grow. These faces are added to the corner as infinitesimal "virtual faces".

Once all of the faces are added to the shape, the velocity at which each edge advances is determined from the etch rate curve. Next, the velocity of each vertex is determined, from the following formula:

$$s_v = (\vec{s}_b \cdot \hat{l}_e)$$

where s_v is the slowness value for the vertex, $\vec{s}_e$ is the slowness value of either of the edges, and $\hat{l}_e$ is the unit vector perpendicular to the line that joins the slowness vectors of the two edges.

When all of the vertices have been assigned velocities, each face has its $\frac{dl}{dt}$ calculated, and faces with negative $\frac{dl}{dt}$ are assigned a "time to live", that is, amount of time until they are etched away to zero length.

One complication of the program is that sometimes there are non-local contour changes: a shape is "etched through", that is, one polygon splits and becomes two. To handle this, the program calculates how long it will take each vertex to collide with every face in the object, and assigns to each vertex a "time to collide" which is the minimum such value. One possible speed optimization would be to make this process more efficient, perhaps by maintaing a list of edges with which each vertex could possibly collide, and by partitioning all of the scene space into small local cells.

Once all of the above calculations are complete, the program moves forward by a time step of $t = min(f, v, u)$, where f is the face time to live, v is the vertex time to collide, and u is a user-specified time step. All vertices are moved by $t\vec{v}_v$. The program then makes any contour changes that are necessary (deleting faces that have etched away, and splitting contours if a vertex has collided with a face). It continues to do this until it has reached the specified finish time for etching, or until the entire shape is etched away to nothing.

To view an animation of the etching process, the user may specify a maximum time step per frame. If this is done, a wire-frame drawing of the shape is produced at each time step. There is also an option to pause for a keystroke between frames, and to output intermediate shapes to UniGrafix files.

Figure 4.1 shows an example of the output of this program. The starting shape is a square with vertical and horizontal faces. The etch rate function used was the one developed in Chapter 2 (see Figure 2.12 for the etch rate diagram). From this output one can readily see that the fast (1,2) and (2,1) faces dominate the etching at convex corners. Figure 4.2 shows a shape with a concave corner. At the concave corner, the appearance of the slow faces ((0,1), (1,1) and (-1,1)) is apparent. Figure 4.3 shows a shape that starts with two internal holes. As the etching progresses, these holes merge. Somewhat later, they merge with the outer contour and break the shape open. Toward the end of the etching process, the shape breaks into four separate pieces.

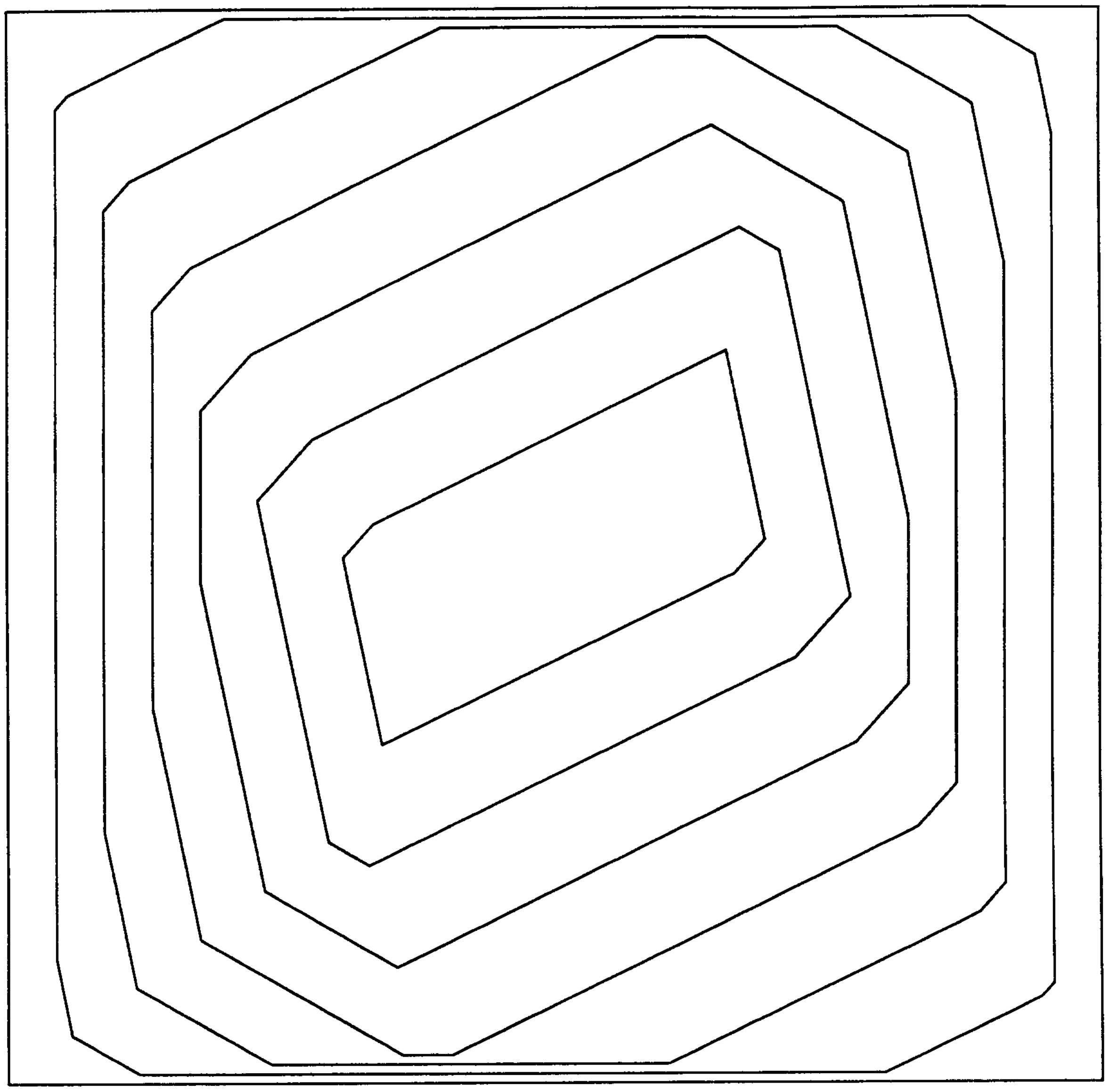


Figure 4.1. This figure shows the etching of an object that starts out as a square. It was etched with the etch rate curve pictured in Figure 2.12. Successive contours of the object at uniform time steps are pictured.

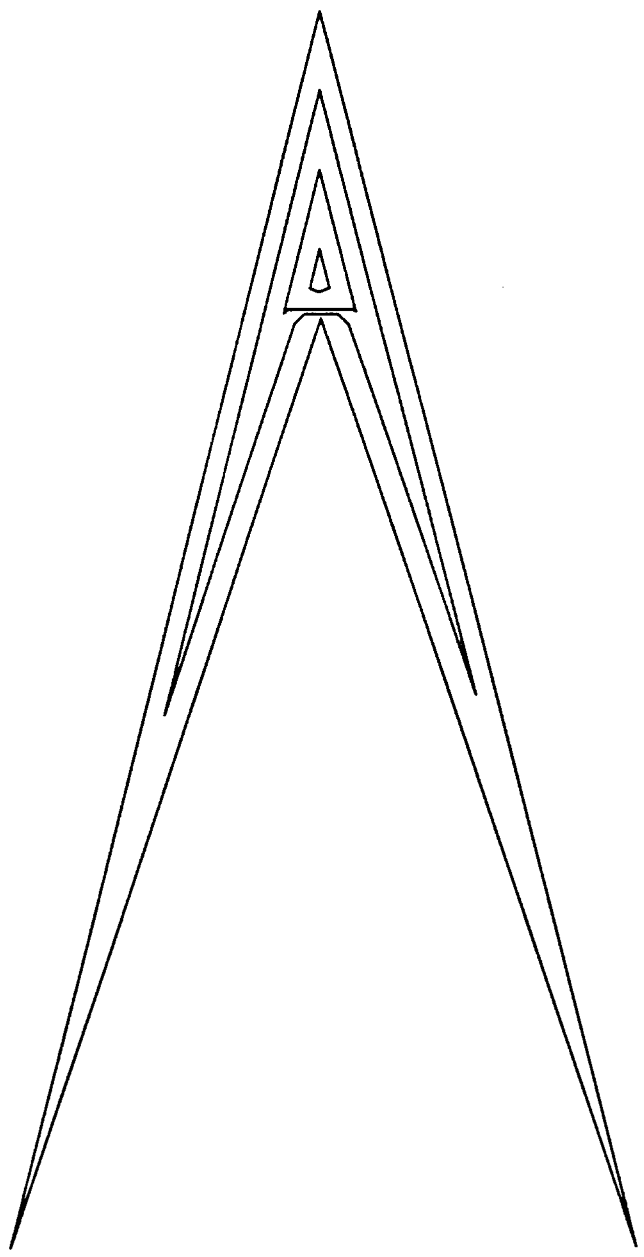


Figure 4.2. This figure shows the etching of an object with a concave corner.

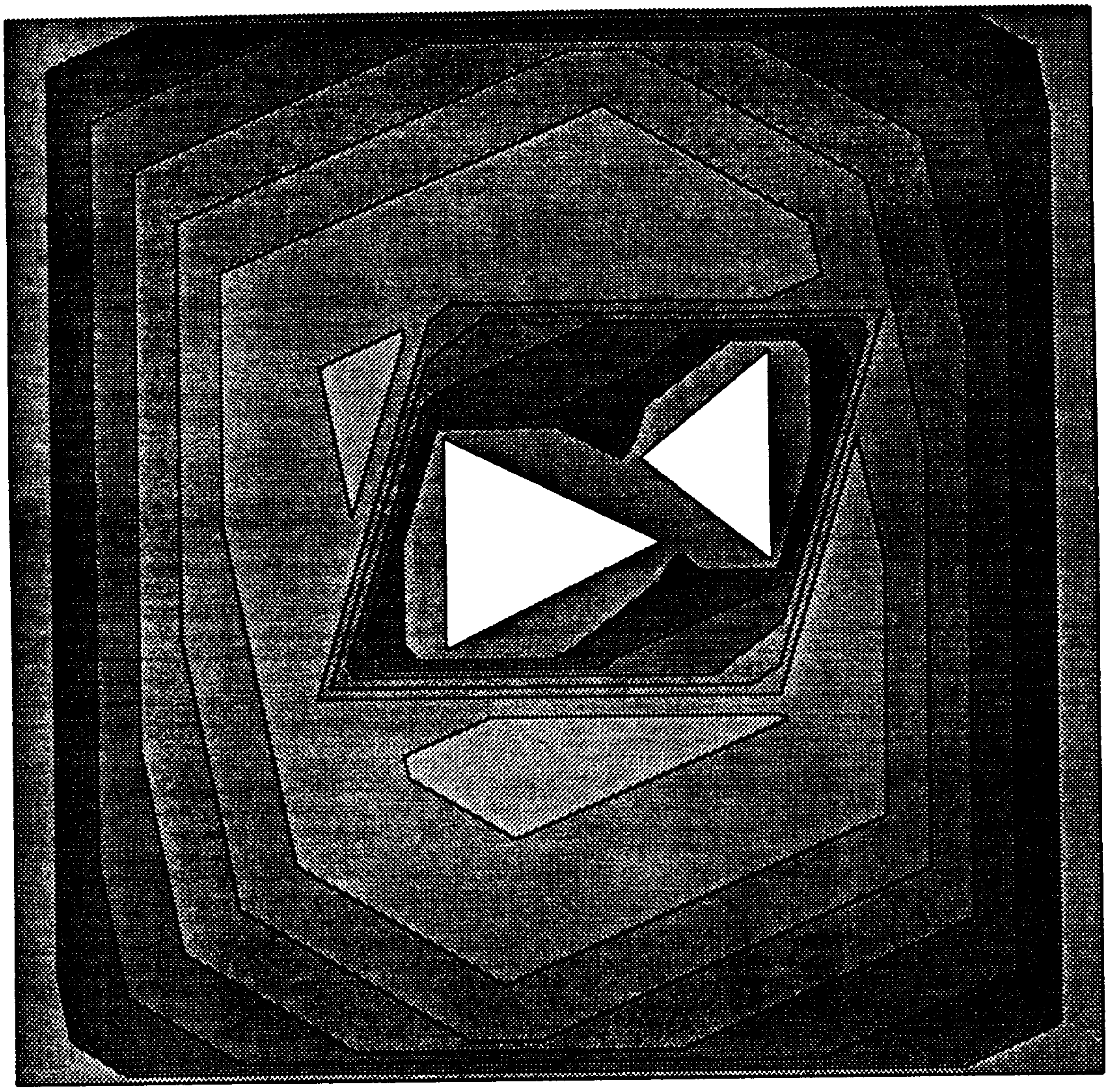


Figure 4.3. This figure shows the etching of an object that starts out having two internal holes.

CHAPTER 5

3D ETCHING RATE MODEL

The development of an etch rate diagram in three dimensions follows a derivation similar to that of the two-dimensional case. Only crystals with one atom per unit cell are considered, and secondary effects of the etchant working on a substance, such as surface heating, convection, and non-uniform distribution of the etchant are ignored. To simplify analysis, in 3D only the case of a cubic crystal lattice is analyzed. Notationally, the unit cell of the 3D crystal lattice is described with the three vectors $\vec{l_x}$, $\vec{l_y}$ and $\vec{l_z}$, in the direction of the cartesian coordinate axes. For a cubic crystal lattice, these will be equal to the unit basis vectors $\hat{i}$, $\hat{j}$ and $\hat{k}$, respectively.

The probability of an atom being etched away during one time step is again modeled as the product of the probabilities of its bonds breaking. For each atom, only the bonds to its six possible nearest neighbors are considered. Bonds in internal corners or within the crystal are expected to have a much lower probability of breaking because the geometrical obstruction renders the attack of the etchant difficult.

1. Face Characterization

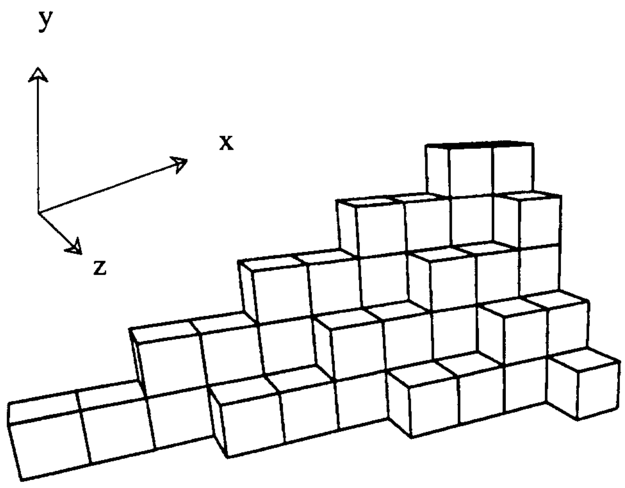


Figure 5.1. This figure illustrates the (1,2,3) face.

As in 2D, in 3D a face is generally characterized by its Miller indices (i,j,k). Figure 5.1 is an illustration of the (1,2,3) face. As in 2D, the Miller indices are determined by considering where a face intersects the axes. For any face, the intercepts can be scaled such that they occur at $\frac{\vec{l_x}}{i}$, $\frac{\vec{l_y}}{j}$, and $\frac{\vec{l_z}}{k}$. One can also think of the Miller indices as a scalar multiple of the components of the normal vector to a face in the coordinate system defined by the basis vectors $\vec{l_x}$, $\vec{l_y}$ and $\vec{l_z}$. In the case where the lattice is cubic, the Miller indices are thus equal to the components of a vector normal to the face.

In crystallography, the Miller indices are generally scaled so that they are all integral. For our purposes, it is more convenient to scale them such that one of them is exactly 1, and the others are either 0 or have magnitude greater than or equal to 1. As an example, the face parallel to the xy plane is the (0,0,1) face.

Because we are only considering crystals with orthogonal basis vectors (i.e. we do not allow for monoclinic or triclinic crystals), the sign of the intercepts of a face with the axes is insignificant. Faces with normal vectors (pointing out of the object) of (1,1,1), (1,-1,1), and (-1,-1,-1) all have the same bond pattern, and therefore the etch rates have the same formulation. In our analysis, we constrain the Miller indices to all be non-negative. In the case of a cubic crystal lattice, the above three faces would all have Miller indices of (1,1,1).

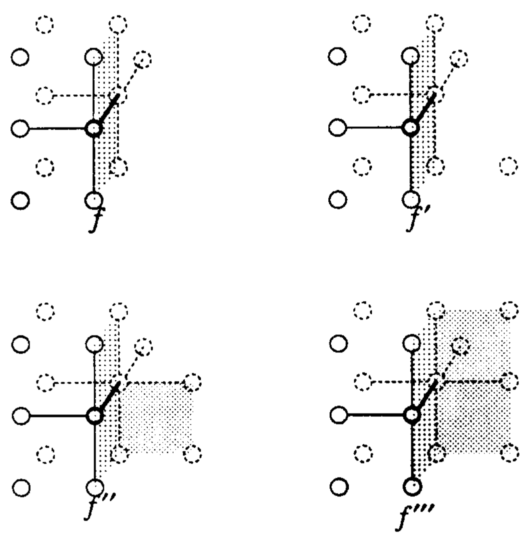


Figure 5.2. This figure illustrates the family of "f" bonds. The bond itself is shown as a heavy line, and the atom being considered for removal is shown as a double circle. The shading indicates the boundary of the object.

2. Bond Classification

We identify three basic types of bonds: "*b*", "*f*", and "*e*". A *b* bond is one that goes into the bulk of the crystal, as in the strongest bond of the (0,0,1) face. An *f* bond (see figure 5.2) is a bond on a face of the surface with two coplanar facets attached. An *e* bond (see figure 5.3) is a bond at a convex edge. The number of primes attached to the latter two indicates how many obstructing atoms forming bonds at concave angles are attached to the other end of the bond at the atom considered for removal. As a further characterization the formulation (*k*—*m.n*) is used. This describes the neighborhood of the bond in question. *k* specifies how many other bonds are attached to the atom that we consider for removal, in a plane normal to the bond. *m* specifies how many other bonds are attached to the bond at its other end in a plane normal to the bond. *n* specifies how many of the eight nearest neighbors in that second normal plane are actually present.

With this, the probabilities for removal of a first individual atom from a particular surface can be readily derived. For this analysis, the following quantitative numbers for bond breaking will be used. The ratios of these rates are based on educated guesses. When used in our simulation program, they seem to give reasonable results that are comparable to some experimentally observed macroscopic etching behavior.

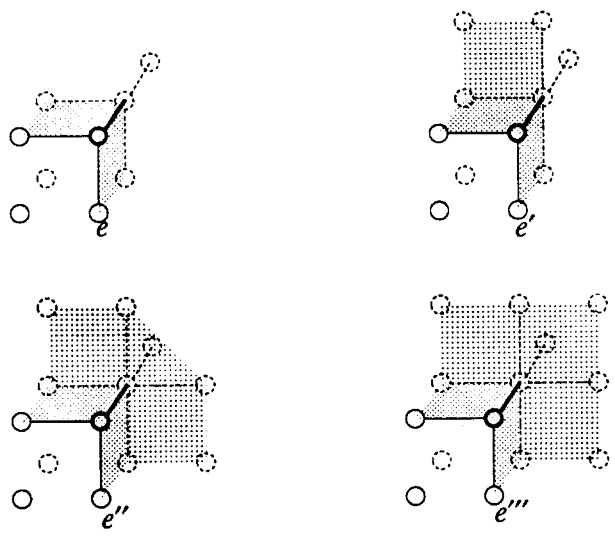


Figure 5.3. This figure illustrates the family of "*e*" bonds. Again, the bond itself is shown as a heavy line, and the atom being considered for removal is shown as a double circle. The shading indicates the boundary of the object.

NAME	CHARACTERISTIC	RATE
b	(4—4.8)	0.003
f'''	(3—4.8)	0.007
f''	(3—4.7)	0.01
f'	(3—3.6)	0.02
f	(3—3.5)	0.05
e'''	(2—4.8)	0.07
e''	(2—4.7)	0.1
e'	(2—3.5)	0.2
e	(2—2.3)	0.5

The further assumption is made that at atoms in concave corners will not be attacked at all, and that atoms loosely attached through 2D "membranes", i.e. (1—x) or (0—x), will be removed immediately.

3. Etch Rates at Key Face Orientations

As a first step in deriving an etch rate surface from bond strength parameters discussed above, the etch rate for several "key" face orientations is analyzed. Six such key orientations are studed: the orientations where the geometry of the face represents some kind of limiting case of the type of bond patterns present on the surface of the crystal. In the 2D case, the key orientations turned out to be the (1,0) face, the (2,1) face and the (1,1) face (the other key faces were geometrically similar to these). In 3D, there are six types of bond arrangements, exemplified by the (1,2,2), (1,1,2), (1,1,1), (0,1,2), (0,1,1) and (0,0,1) faces. These faces are pictured in Figure 5.4. We shall examine each face in turn. As in 2D, we assume that the probability that an atom gets removed is the product of the probabilities that all the bonds that hold it in place get removed.

At several of the face orientations, etching proceeds as an initial "attack" phase, followed by a "stripping" phase, much as was the case for the (1,0) face in 2D. In 3D, of course, the "stripping" phase may proceed along a one-dimensional edge or a two-dimensional surface.

The atom stripping process along a 2D ledge is first re-examined. The analysis in the 2D section is not adequate for use here because it is not very accurate where the corners have almost the same removal probability as the initial attack sites.

a denotes the attack rate on a virginal (1-dimensional) surface for removal of the first atom. c is the removal rate of an atom at the boundary of the stripping area, and l is the removal rate for the last atom between two stripping regions.

For this simple estimate of the compound etchrate produced by the attack-and-strip action on a 1-dimensional surface we make the following assumptions. New holes get started in sequence along the surface, one per clock tick, at regular distances D from one another. After G generations two adjacent holes have merged and a new hole is started in that place in the next generation. The hole starts with size 1 (atom) after the attack, and grows at a rate of $2c$ per generation. The last atom gets removed at rate l, rather than c. Thus it holds:

$$D = 1+l+2cG-c \qquad \text{(equation 1)}$$

The last term appears since the neighbor hole starts one generation later.

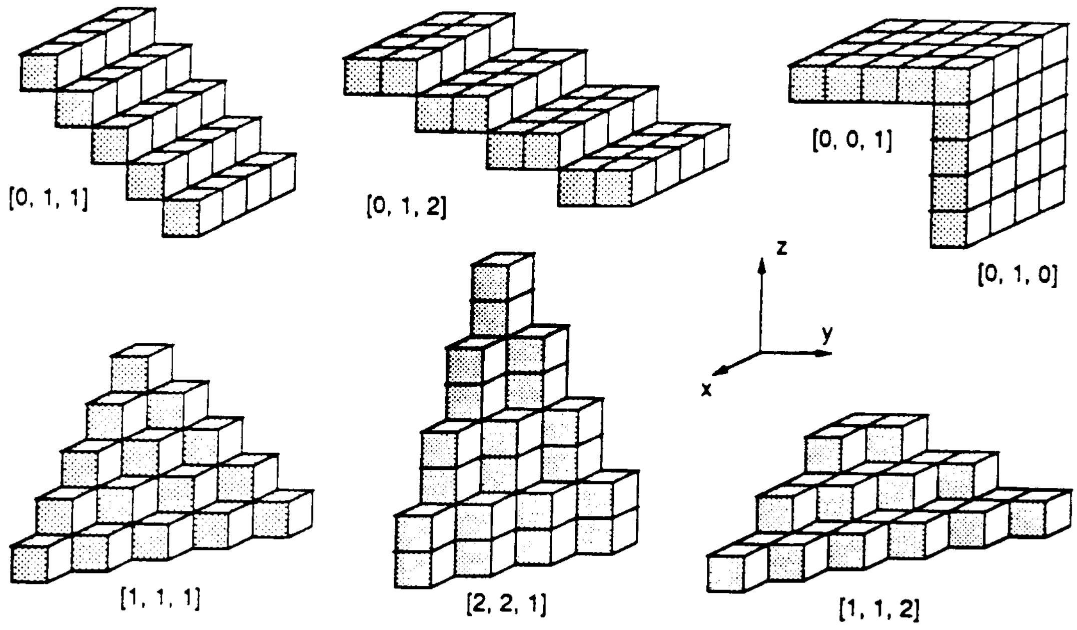


Figure 5.4. This figure illustrates the six key face orientations in 3D.

A simple argument would tell us that we need a surface of size N atoms to get perfectly periodic conditions, where exactly one attack happens in every generation, where $Na = GDa = 1$. However the above situation is the best possible, because every attack at rate 'a' starts successfully a new hole. The calculated compound rate turned out to be too high by at least 50% and sometimes even more than 100% when compared to a statistical simulator that modeled this etching situation. We need to add a correction factor to take into account that sometimes the attack on the surface will not produce a new hole. This is the case when the attack occurs immediately adjacent to an existing ledge (hole boundary). If the attack occurs on the inside, we assume that it has no effect because the atom is protected by the nearby ledge from being removed. If the attack occurs on the corner atom of the ledge, it simply widens the hole, but does not produce a new one. Thus for every existing hole there are 4 atoms where the attack does not start a new hole. Since we assume that there is a hole every D atoms the net rate at wich new holes start is only $a(D-4)/D$. Thus we get the equality:

$$G\ (D-4)\,a = 1 \qquad \text{(equation 2)}$$

Combining equations 1 and 2 we get a quadratic equation for G:

$$1 = Ga(2cG - 3 + l - c)$$

or for the compound rate $R = 1 / G$:

$$R^2 - Ra(l - c - 3) - 2ac = 0$$

The meaningful solution becomes:

$$R = a\left(\sqrt{\frac{2c}{a} + t^2} - t\right) \qquad where\ t = 0.5\,(3 + c - l)$$

We can explore a few special cases:
For l=c this simplifies to:

$$R = \frac{3a}{2}\left(\sqrt{\frac{8c}{9a} + 1} - 1\right)$$

and for $l = c = a$ it becomes:

$$R = \frac{3a}{2}\left(\sqrt{\frac{17}{9}} - 1\right) = 0.5616\, a$$

The last rate is lower than the attack rate a, because we assume that the atoms on the lower side of a ledge do not get attacked. Since the rates c and l are equally low as the attack rate a, there is a high density of small holes and thus nearly half the atoms lie in these protected positions.

We still have a rather idealistic situation as the basis for our analysis. The statistical simulator predicts for this extreme situation a compund rate of $0.45a$. A detailed statistical analysis is beyond the scope of this report. In the situations that we need for the analysis of the key etch rates on 3-dimensional crystals, the above formula agrees with the simulator within 10%. This is good enough given the many rather drastic simplifying assumptions that we had to introduce into the derivation of the key etch rates.

With these formulae, we can proceed to the analyses of the six key faces.

3.1. The (1,2,2) Face

Applying the appropriate bond probabilities into the formulae developed above, one obtains:

Removal of first surface atom: $e''^2 e = 0.0050$
Removal of corner atoms in (1,1) ledge: $e''e'e = 0.0100$
Removal of last atom in ledge: $e'^2 e = 0.0200$
$Frate_{1,2,2} = 0.0017$

Removing one row of atoms like that results in a picture identical to that with which we started; the net face propagation rate, therefore, can be calculated from the cosine of the relevant angles (as in 2D).

3.2. The (1,1,2) Face

Removal of first surface atom: $e'''e'^2 = 0.0028$
Removal of corner atoms in (1,1) ledge: $e'''e'e = 0.0070$
Removal of last atom in ledge: $e'''e^2 = 0.0175$
$Frate_{1,1,2} = 0.0033$

3.3. The (1,1,1) Face

Removal of first surface atom: $e''^3 = 0.0010$
Removal of corner atoms in (1,1) ledge: $e''^2e' = 0.0020$
Removal of last atom in ledge: $e''e'^2 = 0.0040$
$Frate_{1,1,1} = 0.0010$

3.4. The (0,1,2) Face

Removal of first surface atom: $f'''fe^2 = 0.000088$
Removal of corner atoms in (1,1) ledge: $e'''e'e = 0.0070$
Removal of last atom in ledge: 1.00
$Frate_{0,1,2} = 0.0010$

This is the composite recession rate of this ledge. The etch rate perpendicular to the face normal is 0.0008. Because it is needed later we compute the strip rate for an edge between the (0,1,0) and the (0,0,1) faces:

Removal of first surface atom: $f^2e^2 = 0.00063$
Removal of corner atoms in (1,1) ledge: $e'^2e = 0.020$
Removal of last atom in ledge: 1.00
$Erate_{0,1,2} = 0.0044$

3.5. The (0,1,1) Face

Removal of first surface atom: $f'''^2e^2 = 0.000012$
Removal of corner atoms in (1,1) ledge: $e'''^2e = 0.0025$
Removal of last atom in ledge: 1.00
$Lrate_{0,1,1} = 0.00023$

This is the composite removal rate for the first ledge. The adjacent ledges now get removed at the faster rate calculated above for the (0,1,2) face. This constitutes a second level of attack and strip action:

Removal of first row of atoms occurs at rate: $Lrate_{0,1,1} = 0.0002$
Removal of adjacent rows of atoms occur at rate: $Frate_{0,1,2} = 0.0010$
Removal of last row of atoms between strip areas: $Erate_{0,1,2} = 0.0044$
$Frate_{0,1,1} = 0.00042$

3.6. The (0,0,1) Face

Removal of first surface atom: $a = bf^4 = 6.25 \cdot 10^{-9}$

This causes a hole that is bounded by the (1,2) type ledges for which the recession rate has been computed in the (0,1,2) face. With the above rates determined, the compound rate on the (0,0,1) flat can be computed. This now involves a two dimensional stripping action. After the initial attack that removes one atom from the surface it is assumed that there is a spreading square hole that opens up with the rate of the receding staircase steps calculated for the (0,1,2) face ($R = Frate_{0,1,2}$). The area of that hole after G generations is $4R^2G^2$, and on a surface with N atoms there will be NaG holes. In the equilibrium this relation holds:

$$4R^2G^2\,NaG = N$$

thus

$$G = (4R^2a)^{\frac{-1}{3}}$$

G is the number of generations it takes to remove one layer of atoms. The face removal rate is its inverse:

$$Frate_{0,0,1} = (4R^2a)^{\frac{1}{3}} = 2.97 \cdot 10^{-5}$$

3.7. Results

From all of the above-calculated atom-layer removal rates the actual etch rates in lattice constant units are calculated by projecting on the respective face normals. The following rates are obtained:

(1,2,2) Face:	$0.0050 \cdot 0.333 = 0.0017$
(1,1,2) Face:	$0.0033 \cdot 0.408 = 0.0014$
(1,1,1) Face:	$0.0010 \cdot 0.577 = 0.00058$
(0,1,2) Face:	$0.0010 \cdot 0.447 = 0.00046$
(0,1,1) Face:	$0.00042 \cdot 0.707 = 0.00030$
(0,0,1) Face:	$0.000030 \cdot 1.0 = 0.0000305$

Of course, it is permissible to uniformly scale these rates by any real number, since nothing has been said about the units with which the velocity is expressed. Normalized, relative slowness values for the six key directions can therefore be computed:

(1,2,2) Face:	1.0
(1,1,2) Face:	1.2
(1,1,1) Face:	2.9
(0,1,2) Face:	3.7
(0,1,1) Face:	5.6
(0,0,1) Face:	56

4. Etching Rates between Key Orientations

To determine etch rates at face orientations other than the six key rates, we recall that the slowness shape in 2D was very nearly approximated by straight lines connecting the key rates. We assume, by extension, that in 3D the entire etch rate slowness surface can be approximated with planar triangles between the key etch rates. This results in the slowness surface pictured in Figure 5.5.

The difference in etch rates is too extreme for any detail to be visible in Figure 5.5. To make the shape more apparent, in Figure 5.6 every vertex is scaled so that its distance to the origin is the logarithm of the slowness value. This is not the same as a logarithmic plot of the curve: In a true logarithmic plot, the faces would no longer be planar. Finally, Figure 5.7 shows the projection of the key directions onto the unit sphere with the corresponding triangular facets between them.

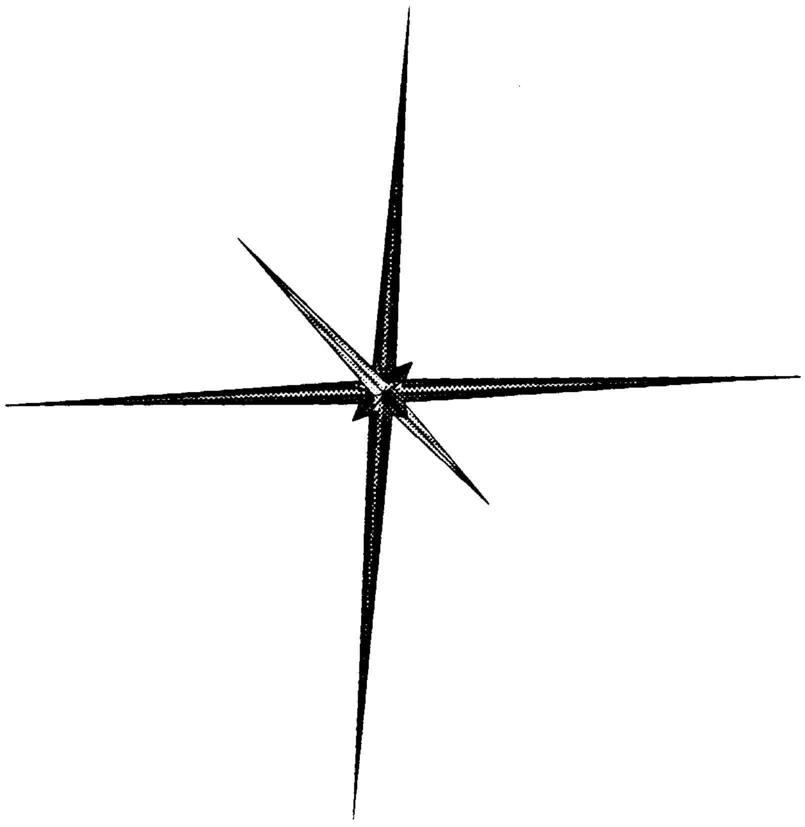


Figure 5.5. This shows the slowness contour of the etch rate function derived in this chapter.

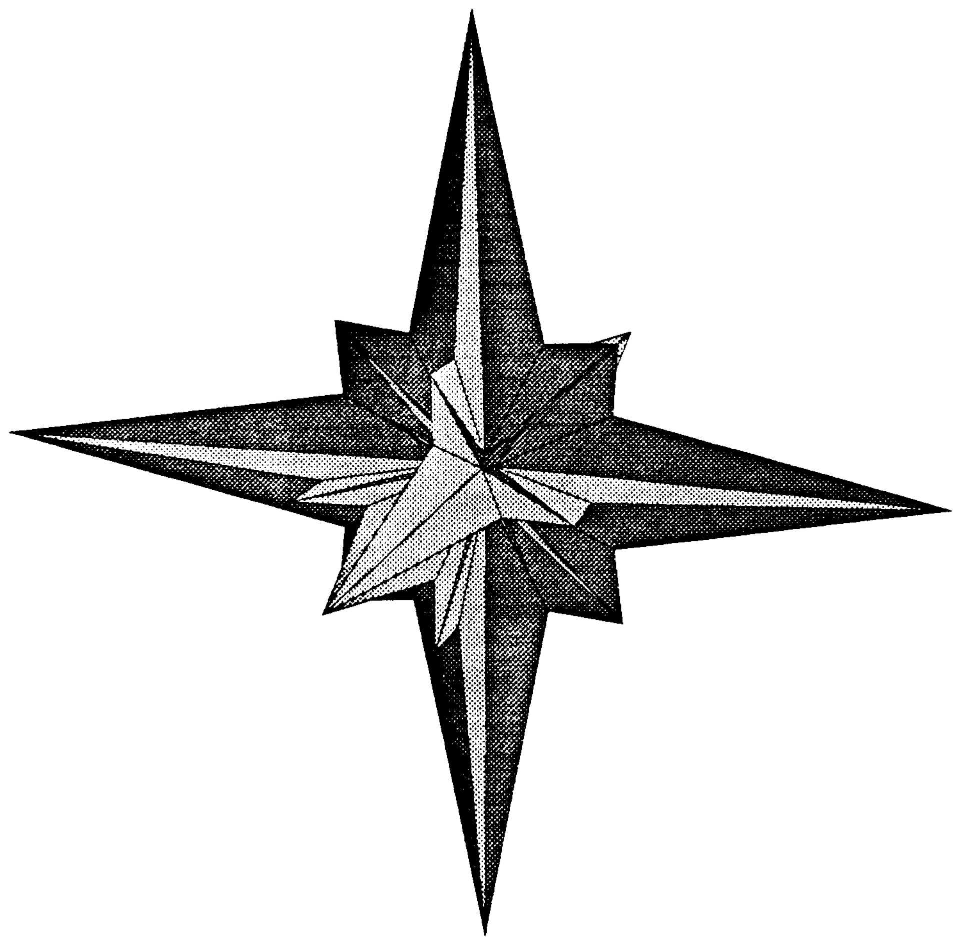


Figure 5.6. This shows the slowness contour of the etch rate function derived in this chapter, where every vertex is scaled such that its distance to the origin is the logarithm of the slowness value.

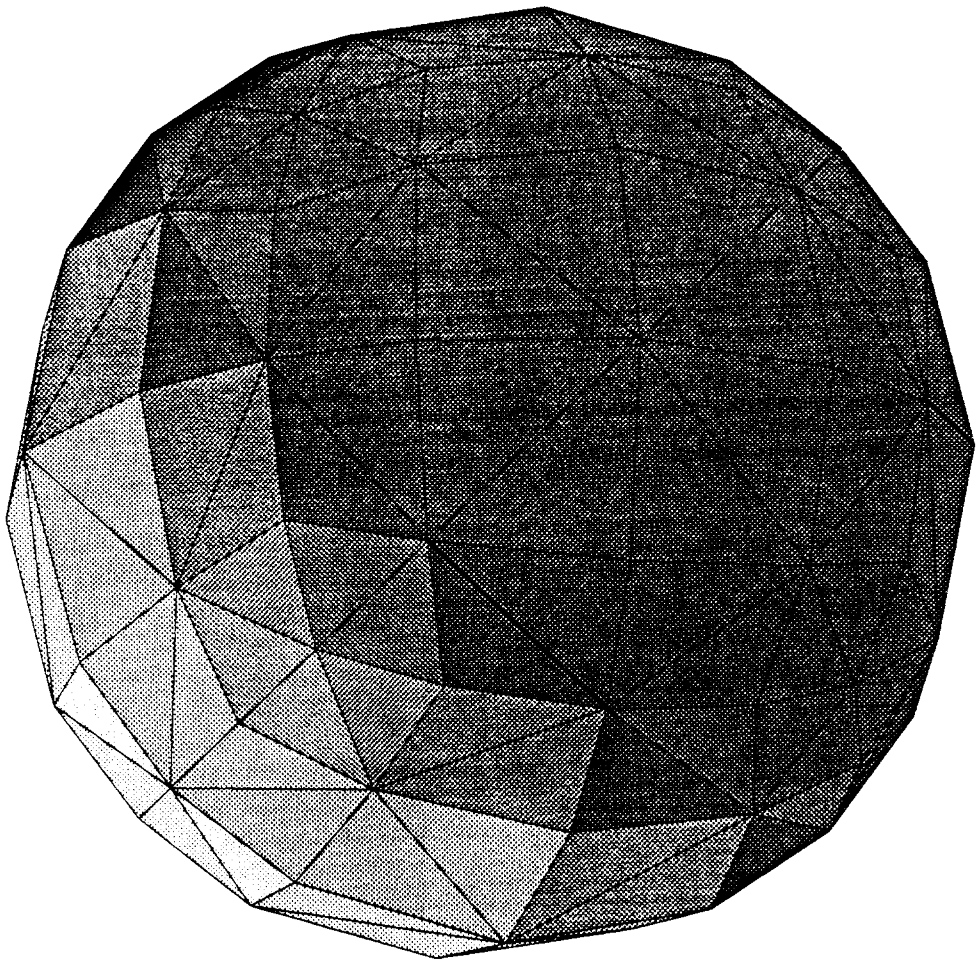


Figure 5.7. This shows the projection onto the unit sphere of the key directions, forming corresponding triangular facets between them.

CHAPTER 6

3D LOCAL GEOMETRY ANALYSIS

In this chapter, the behavior of a three-dimensional shape subjected to etching is analyzed. A model of etching such as that developed in the previous chapter is assumed, specifically, it is assumed that the etch rate can be modeled as a slowness surface composed of planar triangles formed from triplets of slowness vectors at the key etching directions. From this assumption algorithms are developed that allow the simulation of the etching process at the level of faces, edges, and vertices.

This analylsis is not restricted to cubic, or even orthorhombic crystal lattices. The algorithms developed herein can therefore be applied to etch rate functions much more general than the model function developed in the previous chapter.

1. Behavior of Vertices

As in 2D, at a vertex one can imaging a collection of infinitely many infinitesimal faces at all orientations ''between'' the original orientations present. These faces ''compete''; those that grow the fastest will dominate. To determine what will happen to the local geometry around a vertex, one must first determine which faces might potentially form. The infinite set of all possible surface orientations is inspected to decide which of those orientations correspond to faces that could possibly grow to finite size in the shape being etched. It turns out that all but a few surface orientations can be eliminated from consideration with an analysis of their neighborhood in the slowness surface. Aditionally, many faces are precluded from forming by the arrangement of faces at the vertex being considered (because a new face may not cut into the bulk of the shape being etched). We call the set of surface orientations that cannot be eliminated based on the above two constraints the set of ''candidate faces''.

1.1. Candidate Faces

To determine which faces could possibly form at a vertex, an analysis parallel to that of Chapter 2 is followed. This analysis again uses Frank's theorems, which state that new faces will only form at points where the curvature of the slowness curve is either concave or convex. Since our simplified slowness surface is made up of planar faces, the only parts of the slowness surface that correspond to candidate faces are its edges and vertices. Thus, we can restrict the set of candidate faces *a priori* to contain only faces with orientations along edges or on vertices of the slowness surface.

We shall first consider the simplest case: a ''convex vertex'', formed by the meeting of three planar faces. The term ''convex vertex'' is used to mean that that the three edges that meet at the vertex are all convex. As an aid to visualization, the ''vertex figure'' is formed. This is done by projecting onto the unit sphere the normal vectors for all three faces present. The normals of adjacent faces (those that share an edge) are connected with great-circle arcs. This vertex figure can also be projected onto a suitable plane. Figure 6.1 illustrates a vertex

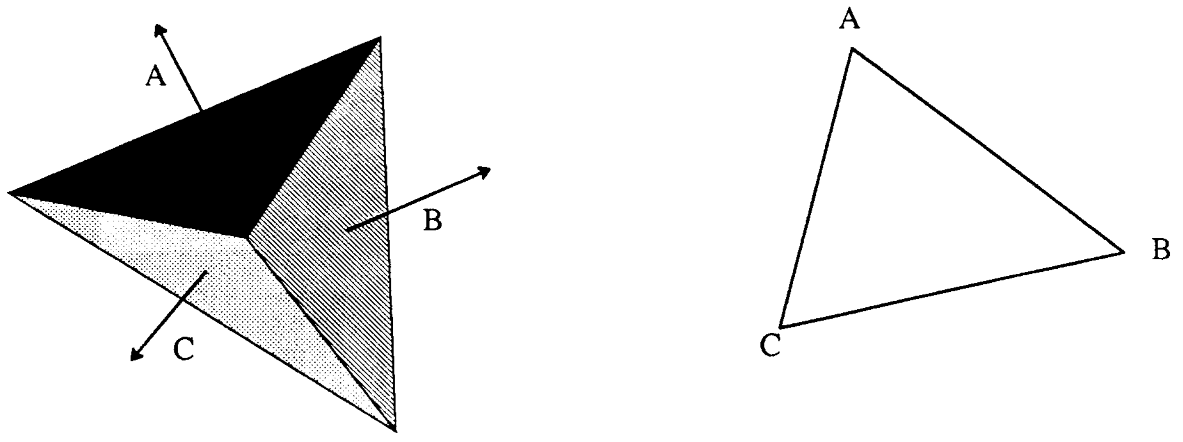


Figure 6.1. This figure illustrates a convex vertex formed by the faces A, B and C, and the corresponding vertex figure.

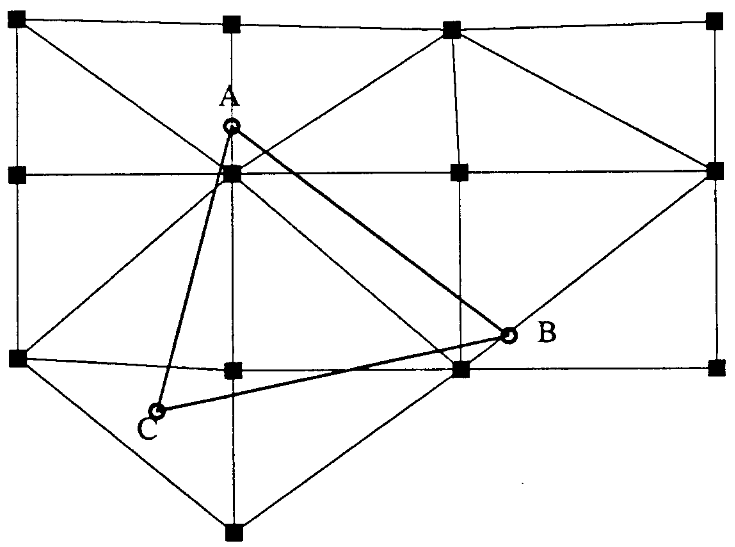


Figure 6.2. This figure shows the vertex figure of a convex vertex (that of Figure 6.1) overlayed with a region of the projection of the slowness shape onto the unit sphere. The given faces are shown as small circles, and the key directions are shown with small squares.

and its corresponding vertex figure. From inspection of the vertex figure, it is clear that the set of candidate faces can be restricted to include only face orientations within the vertex figure. To visualize the two constraints we have placed on the set of candidate faces, it is useful to superimpose the vertex figure of the vertex under consideration onto the triangulated unit sphere of Figure 5.7. This is done in Figure 6.2.

In Figure 6.2 there are two key slowness directions that fall within the vertex figure. This means that the set of potential faces that form at the vertex will contain two members, with normal vectors equal to those two orientations. New bevel faces might also form along an edge of our object at points where the corresponding edges of the vertex figure intersects with an edge of the unit sphere projection of the slowness surface. For example, edge AC has four such crossings; the set of potential faces for edge AC thus contains four members corresponding to those four crossings. Note that one crossing passes directly through the point A on the vertex figure. This implies that a new face might form at this orientation. This point can be eliminated from consideration, because there is already a face of this orientation present in the shape: face A.

The same analysis clearly holds at any vertex that is convex, that is, any vertex that is at the intersection of several edges, all of which are convex. In the case of a vertex formed by more than three faces, the vertex figure would no longer be a triangle, but it would still be a convex polygon. This same analysis also works for a vertex that is (totally) concave, that is, any vertex that is at the intersection of several edges, all of which are concave.

A problem arises at vertices that might loosely be classified as "saddle" vertices, which includes all vertices at the intersection of several edges, some of which are concave and some of which are convex. The simplest such vertex, one at which there are two convex edges and one concave edge, is pictured in Figure 6.3. In the vertex figure, a convex edge is shown as a solid line, and a concave edge as a dashed line.

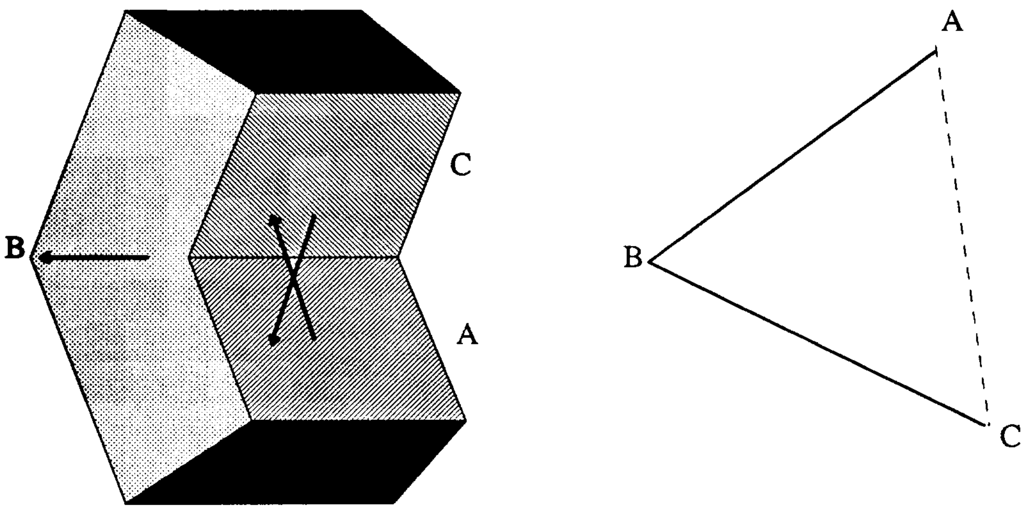


Figure 6.3. This figure illustrates a saddle vertex formed by the faces A, B, and C. Edges AB and BC are convex, whereas edge CA is concave.

One source of new faces at any vertex are the bevel faces that form when an edge splits. The faces will form regardless of the configuration of the vertices at the endpoints of an edge, because their formation does not depend on any process taking place at the endpoints. The formation of bevel faces is discussed in the next section of this chapter.

In the absence of bevel faces, we assert that no new faces can form. Inspection of such vertices reveals that were such a face to be present, it would necessarily cut through the bulk of the object being etched. This would contradict the presence of one or more of the edges that make up the vertex. Whereas the edges of the object being etched may certainly disappear over time, they certainly can not disappear instantaneously (or in infinitesimal time). From these observations, we conclude that an area of a vertex figure bounded by a combination of concave and convex edges represents a range of face normals corresponding to faces that cannot form at the given vertex.

A more complicated case is illustrated in Figure 6.4. This figure shows a saddle vertex formed by four faces. In this case the vertex ''looks'' like a convex vertex: all of the edges fit into one half-space. The point I corresponds to the orientation of the plane that contains edges DA and BC. From inspection, we conclude that the set of candidate faces is the set of vectors that fits within the triangle ABI. The area within triangle CDI is rejected, by extension of the argument for the previous example.

The fourth (and final) case is illustrated in Figure 6.5. This illustrates what might be called a ''true'' saddle vertex: The edges that form the vertex do not all fit in any one half-space. The vertex figure for this vertex is similar to that in Figure 6.3, in that it has no crossings but has a mixture of concave and convex edges. From inspection of the geometrical

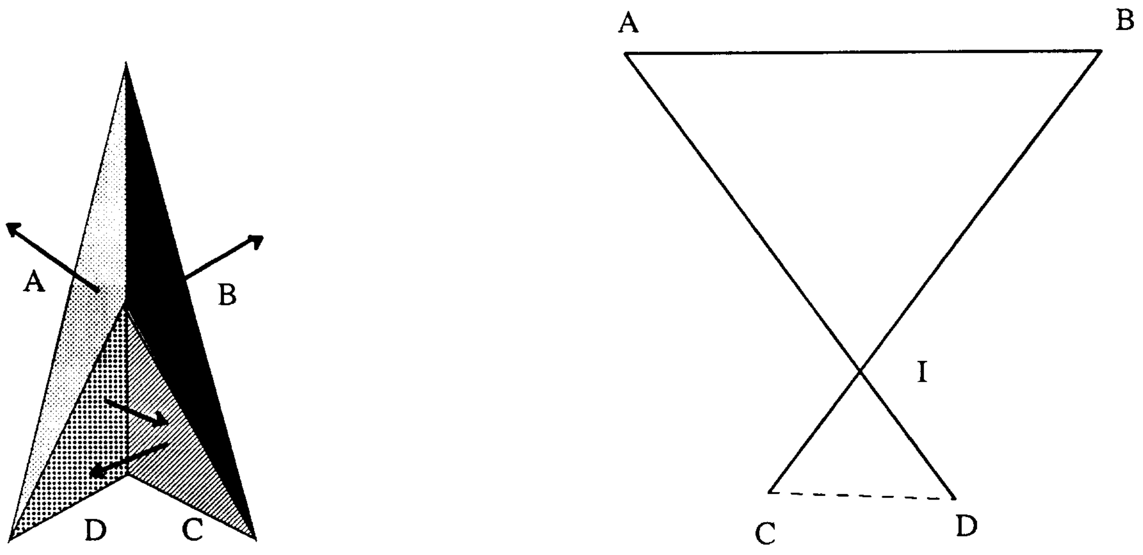


Figure 6.4. This figure illustrates a saddle vertex formed by the faces A, B, C, and D. Edges AB, BC and DA are convex, whereas edge CD is concave. I is simply the point at which the vertex figure crosses itself.

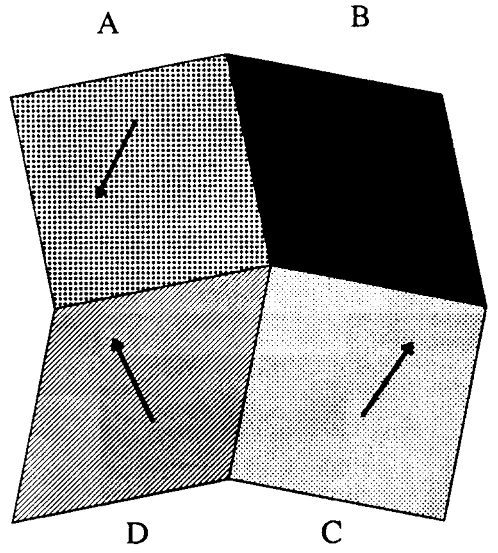
shape and analogy with other vertex figures, we conclude that no new faces form at such vertices.

We assert that all possible vertex configurations can be classed as one of the above four types. The above analysis leads directly to a relatively simple algorithm to determine the set of candidate face orientation. This algorithm accepts a vertex figure (in which all edges are colored either "concave" or "convex"), and returns a set of candidate face orientations (for the formation of new faces at the vertex). It is presented in C-like pseudocode:

```
candidate_faces(vertex_figure)

    -- Returns a set of candidate face orientations

    If vertex_figure contains any crossings,
        split vertex_figure into figures f 1 and f 2 on either side of a crossing,
        return union(candidate_faces(f 1), candidate_faces(f 2)),
    else if the edges of vertex_figure aren't all the same color,
        return the null set,
    else
        return the candidate face orientations within vertex_figure
            (since we know that vertex_figure is a convex polygon).
```



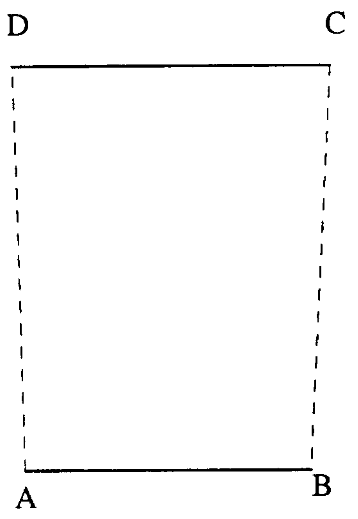


Figure 6.5. This figure illustrates a saddle vertex formed by the faces A, B, C, and D. Edges AB and CD are convex, whereas edges BC and DA are concave.

1.2. Dominant Faces

In this section, an algorithm is developed that will tell us which of the candidate faces will "dominate" and eventually grow to finite size. Such faces (those that actually end up growing to finite size) are called "virtual faces". The following analysis, therefore, determines which of the candidate faces dominate, and form virtual faces. These faces are called "vertex virtual faces", because they form at a vertex. This is as opposed to "edge virtual faces" (also called bevel faces) that form along an edge.

At a convex vertex, the analysis from Chapter 2 can be naturally extended to three dimensions. The convex hull around the slowness vectors of the candidate face orientations is again constructed. Those slowness vectors that are on the inner part of the convex hull that is visable from the origin correspond to virtual faces. Similarly, at a concave vertex, the slowness vectors on the outer part of the convex hull (invisable from the origin) correspond to virtual faces.

Saddle vertices require some more analysis. If a saddle vertex has any candidate faces, they must have been from a part of the vertex figure that had edges that were all colored "concave" or "convex". If we remember from what kind of region each candidate face came, this can be used as a criterion for whether the corresponding slowness vector must be on the inner or the outer part of the convex hull in order to be considered a virtual face.

2. Behavior of Edges

As noted before, edge virtual faces can form at face orientations where the corresponding edge of the vertex figure crosses a triangle edge of the tessellated unit sphere. For any edge, therefore, the set of these crossings constitutes the set of candidate faces. By direct extention from 2D, the (one dimensional) convex hull around the slowness vectors of the candidate faces can be formed. For a convex edge, those slowness vectors on the inner part of the convex hull will be those corresponding to virtual faces. For a concave edge, the slowness vectors on the outer part of the convex hull will be those corresponding to virtual faces.

3. Splitting a Vertex

Once the set of virtual faces present at a vertex is known, the next task is to split that vertex appropriately. For example, a convex vertex at the intersection of three faces that has one vertex virtual face present and no edge virtual faces will split into three vertices. If we call the three original faces A, B and C, and we call the vertex virtual face D, then the three resulting vertices will be ABD, BCD and CAD.

It is useful to consider at this point what determines a vertex. Essentially, a vertex is unambiguously determined by the intersection of 3 linearly independent planes. If there are four planes present at a vertex, that vertex might potentially split into two vertices, although these vertices might be coincident. This consideration in mind, it is useful to define an "s-vertex" (for "simple vertex") as the point at the intersection of three planes. When a vertex is analyzed, it is sufficient to produce a list of s-vertices. Each of these s-vertices will have the same initial position in space, but they will have potentially different velocity vectors.

3.1. Vertex Splitting without Vertex Virtual Faces

Initially, we analyze how a vertex splits in the absence of vertex virtual faces. At this point an example would be helpful. Consider the shape shown in Figure 6.6, where faces A, B, C and D meet at a convex vertex.

The shape in Figure 6.6 can be represented schematically with the intersection tree of Figure 6.7. Each node of the tree corresponds to an s-vertex, and each line corresponds to an f-edge. The heavy lines that are unterminated at one end correspond to the edges originally present in the shape.

A more complex case is pictured in Figure 6.8. This shows a more complicated shape that contains edge virtual faces. A complete intersection tree can be constructed by operating on a set of the f-edges originally present at the vertex.

We define a set α that is derived from the original edges present at the vertex. We call the new edges in α "f-edges" (for "future edge"). They are located at the position in space that is the sum of the edge's velocity and its position, that is, they lie where the original edge would have moved in one unit time. Each f-edge is also characterized with the two faces that share the edge, a point on the edge, and a vector pointing along the edge. This vector must point toward the next intersection point of that edge (i.e. away from the remote vertex on this edge). For this reason this vector is called the "toward vector", and is denoted with, for example, "toward(AB)". Each f-edge is also marked as convex or concave. The original set α for the intersection tree shown in Figure 6.7 would contain AB, BC, CD, DE, EF, FG and GA.

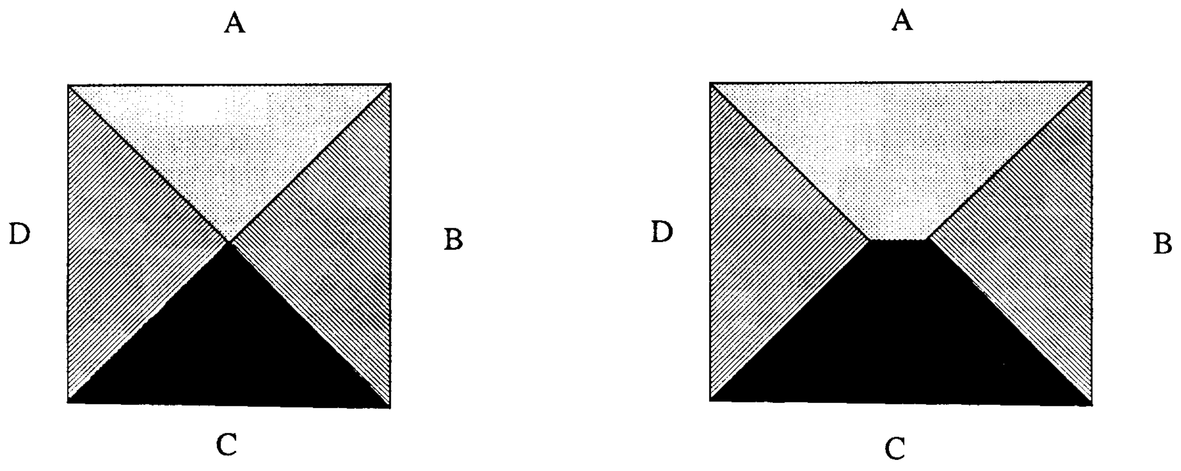


Figure 6.6. The figure shows a view from above a convex vertex formed by four faces A, B, C and D. The figure on the right shows how this vertex might develop (assuming no virtual faces) if it splits into vertices ABC and CDA.

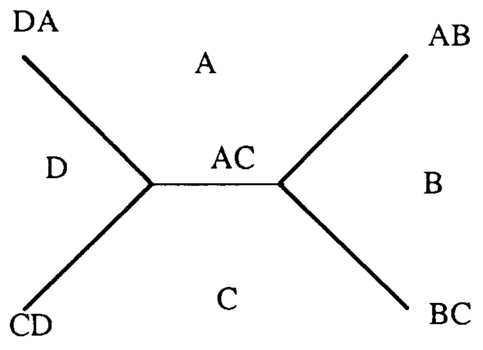


Figure 6.7. This figure shows the intersection tree corresponding to the shape pictured in Figure 6.6. The heavy lines are the edges present in the original shape. In this figure the edges have been individually labeled for greater clarity; this will not be done in subsequent intersection trees.

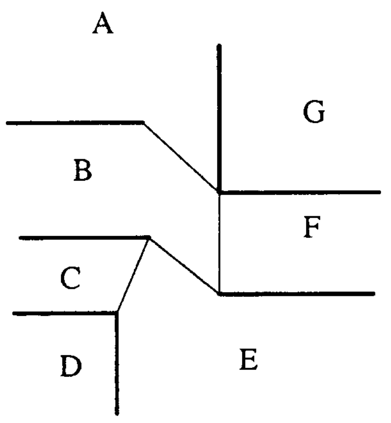


Figure 6.8. This figure shows the intersection tree corresponding to a complex shape containing several edge virtual faces. Faces A, D, E and G were originally present at the vertex, and faces B, C and F are edge virtual faces. Again, the heavy lines are the edges present in the original shape.

The set α may be simplified, under certain conditions, by intersecting two adjacent f-edges. For example, the above α can be simplified by intersecting the f-edges CD and DE, producing a single f-edge CE. For each original f-edge, the toward vector and convex flag was already determined. When a new f-edge is created, however, these values must be calculated.

To analyze this, consider the intersection of three general planes, A, B, and C (in counterclockwise order viewed from outside of the shape being etched). The edges between these faces are called AB, BC, and CA. Each edge XY has a vector toward(XY) associated with it that points from the remote vertex on that edge toward the intersection point currently being considered. Each edge also has a flag convex(XY) that denotes if the edge is convex or concave. For the purposes of simplifying the set α, the f-edge pair CA and AB are intersected to produce the edge BC.

In the following, saying that one vector is "equal" to another means that it points in the same direction—the magnitude is not important. Two vectors being "not equal" (i.e. "!=") means that they point in directions opposite to each other. The notation "v1 d= v2" is a weaker statement: it means that the vectors have a non-negative dot product (intuitively, it means that they point in a similar direction).

This analysis uses the following theorems:

```
Theorem 1:
  normal(C) cross normal(A) = toward(CA) iff convex(CA)

Theorem 2:
  toward(CA) cross toward(AB) != normal(A)
    iff convex(CA) = convex(AB) && convex(CA) != convex(BC)

Corollary 2a:
  toward(CA) cross toward(AB) = normal(A)
    --> convex(CA) != convex(AB) || convex(CA) = convex(BC)

Corollary 2b:
  toward(CA) cross toward(AB) != normal(A)
    --> convex(CA) = convex(AB) && convex(CA) != convex(BC)

Theorem 3:
  convex(CA) = convex(AB) -->
    convex(CA) iff normal(A) d= toward(BC)

Corollary 3b:
  convex(CA) && convex(AB)
    --> normal(A) d= toward(BC)

Corollary 3c:
  !convex(CA) && !convex(AB)
    --> normal(A) !d= toward(BC)
```

Given complete information on two edges that are meeting at an intersection point we must determine the convex() and toward() values of the third vector at this point. Put more formally:

Given:	toward(CA)	convex(CA)	normal(A)
	toward(AB)	convex(AB)	normal(B)
	normal(C)		

We wish to find:

toward(BC) convex(BC)

Additionally, it is not know known *a priori* whether A, B, and C are in clockwise or counterclockwise order.

First, Theorem 1 is used to check that A, B, and C are in counterclockwise order, namely:

normal(C) cross normal(A) = toward(CA) iff is_convex(CA)

If this generates a contradiction, B and C are swapped.

Following this, some detailed analysis is required to deduce convex(BC). First, using Theorem 2 to break the analysis into two cases:

Case 1: toward(CA) cross toward(AB) != normal(A)

convex(BC) = !convex(CA) Directly from Theorem 2.

Case 2: toward(CA) cross toward(AB) = normal(A)

Case 2a: convex(CA) = convex(AB)
--> convex(BC) = convex(CA) Directly from Theorem 2.

Case 2b: convex(CA) != convex(AB)

convex(CA) can't be determined. We mark convex(CA) as UNDETERMINED.

Now it is possible to determine a value for toward(BC) using Theorem 1:

normal(B) cross normal(C) = toward(BC) iff convex(BC)

If convex(BC) is UNDETERMINED, toward(BC) is computed as though BC were convex. When the other intersection point of that edge is found, if it turns out that this guess was incorrect, toward(BC) is reversed and convex(BC) is set FALSE.

As was noted above, adjacent f-edges may intersect each other under certain conditions. Obviously parallel f-edges may not intersect. Another case where f-edges may not intersect is pictured in Figure 6.9. in this case, edge AD would be intersected by edge AC at a point x before its origin. This would imply that toward(AC) were incorrect, which contradicts the analysis that produced toward(AC). This means that edges AC and AD cannot intersect each other, and another pair of f-edges must be tried.

To find the complete intersection tree at a vertex, we use a depth first recursive search over the set α, applying the f-edge intersection operation defined above. This algorithm is called **split_vertex()**.

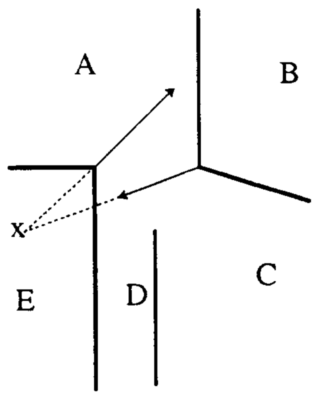


Figure 6.9. This figure shows a pair of f-edges AD and AC that can not intersect. Remember that each f-edge has a direction vector, and cannot intersect at a point opposite it.

```
split_vertex(α)

  -- Returns an intersection tree corresponding to what the vertex v will split
     into under etching. Returns a null tree if α violates a constraint.

  If α has exactly 3 members,
     return the s-vertex formed by the three faces present in α.
  fi
  For each pair of f-edges a ,b on α do:
     Let c be the f-edge obtained by intersecting a and b .
     If no constraints were violated,
        replace the pair a ,b with c in α,
        let T be split_vertex(α),
        if T is not the null tree,
           add the branch corresponding to c to S ,
           return S ,
        fi,
        restore α to its original value,
     fi.
  rof
  Return the null tree.
```

3.2. Vertex Splitting with Virtual Faces

We now consider forming the intersection tree in the presence of vertex virtual faces (vv-faces). Such an intersection tree is shown in Figure 6.10. We start with the intersection tree T formed by the **split_vertex()** algorithm, to which we add branches for the edges originally present at the vertex (i.e. the original members of α). These additional branches are called "leaves", because they are unterminated at one end.

Note that branches of the intersection tree are very much like f-edges. They are formed by the intersection of two faces, and they can be marked as either convex or concave.

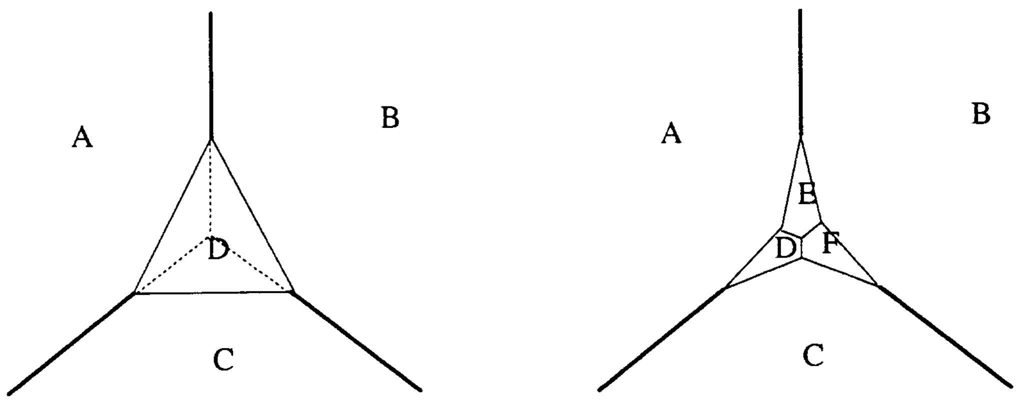


Figure 6.10. This figure shows the intersection tree of two possible evolutions of a vertex that originally has faces A, B, and C, and at which one or more vertex virtual faces form. The dashed lines on the left represent what the intersection tree would look like in the absence of vertex virtual faces.

Branches are also bounded by faces at either end (in figure 6.7, branch AC is bounded by faces D and B). Branches also have a "toward" vector that points along the edge from one bounding face to the other.

Vertex virtual faces are introduced into *T* by intersecting a branch *b* of *T* with a vv-face. For example, a branch AB intersecting with a vv-face C produces two new branches, AC and BC. When this is done, the convex flag and toward vector of each of the two new branches must be calculated. This turns out to be much simpler than when two f-edges intersect, because either both new edges are convex, or both are concave. Because of this, we can apply Theorem 3 directly:

convex(BC) iff normal(C) = toward(AB)
and
convex(CA) iff normal(C) = toward(AB)

Similarly, toward(BC) and toward(CA) come directly from Theorem 1:
normal(B) cross normal(C) = toward(BC) <--> convex(BC)
and
normal(C) cross normal(A) = toward(CA) <--> convex(CA)

The intersection point of a vv-face with the two planes that make up a branch might lie outside the boundaries of that branch. If this is the case, that vv-face does not intersect the given branch.

We introduce a vv-face into an intersection tree by finding the appropriate intersections of that vv-face with branches in the intersection tree, and making the appropriate changes to the intersection tree. This is done for each vv-face in turn. The algorithm that does this is called **push_vvface()**. It makes use of the **find_intersections()** algorithm that proceeds down an intersection tree until it finds all of the intersections of a vv-face with a given subtree.

```
push_vvface($T$, $vv-face$)

  -- Introduces the vertex virtual face $vv-face$ into intersection tree T

  For each branch $b$ of $T$,
    mark $b$ as not yet visited,
  rof.
  For each leaf branch $b$ of $T$,
    find_intersections($b$, $vv-face$),
  rof.
  For each unvisited branch $b$ of $T$,
    remove $b$ from T,
  rof.


find_intersections($b$, $vv-face$)

  -- Finds all intersections of $vv-face$ with the branch $b$ or its subtree, and adds
     sub-branches, as appropriate

  If $b$ is marked as visited,
    return,
  fi.
  Mark $b$ as visited.
  If $vv-face$ intersects $b$,
    break $b$ 's links with its subtree,
    find or create two branches $c$ and $d$ from the intersection of $b$ with $vv-face$,
    mark $c$ and $d$ as visited,
    link $c$ and $d$ to $b$,
  else
    for every sub branch $s$ of $b$
      find_intersections($s$, $vv-face$),
    rof,
  fi.
```

3.3. Linking up Contours

Once all of the s-vertices for the entire shape have been determined, all of the contours on the shape must be re-linked. This is a relatively simple procedure, accomplished by the routine **fix_contours()**. In any implementation of this algorithm, some care must be taken to ensure that the new contours run clockwise or counter-clockwise, as appropriate.

3.4. Re-analysis of Features

Once a vertex splits, it is possible that features will develop that make it possible for virtual faces to form that were not possible with the initial shape. This can happen, for example, at a saddle vertex formed by four faces. Should such a vertex split into two vertices, one of them may have a larger set of candidate virtual faces than was originally present. Any implementation of the above algorithms must re-analyze new vertices and new edges to see if additional virtual faces form.

fix_contours()

```
-- Introduces new s-vertices into a shape

For each face F do:
  If F has any vertices that split,
    -- Replace F with a new list of edges:
    let FP be an empty list of edges,
    pick any edge F,A,
      -- F and A are the faces that determine the edge
    let v1 be the vertex F,A,X,
      -- X is some other face
    let v2 be the vertex F,A,Y where Y != X,
    repeat
      add the edge v1, v2 to FP,
      let v1 be v2,
      let X be A,
      let A be Y,
      let v2 be the vertex of the form F,A,Y where Y != X,
    until we get back to the original v1, v2,
    replace F with FP,
  fi.
rof.
```

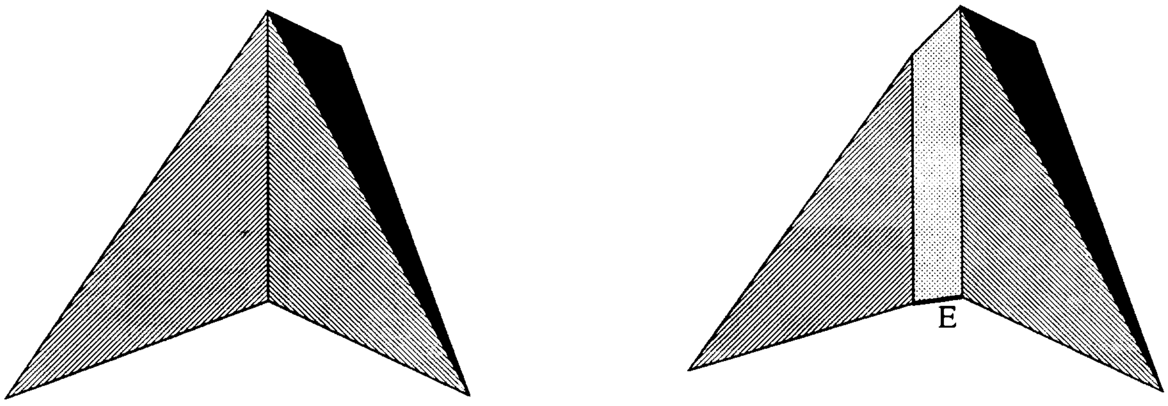


Figure 6.11. This figure shows a shape with a saddle vertex, and the result of etching that shape in the presence of one bevel face. As soon as the edge E grows to finite size, bevel faces might form along it.

A particularly interesting situation can occur when a shape such as that pictured in Figure 6.11 etches. As the initial shape etches, a bevel face developes. The bottom edge bounding this face is called E. Once this edge grows to finite size it is possible for bevel faces to form along it. Assume that such a face does indeed form, and further assume that that face has an etch rate significantly faster than any of the surrounding faces. The result is shown in

Figure 6.12.

As illustrated in Figure 6.12, the new bevel face etches ''up'' the shape. Note that once it becomes triangular, it leaves behind a vertex that has exactly the same configuration as the original vertex. A new bevel face will therefore be formed along the edge that gets ''left behind''. This will cause a new edge like the edge ''E'' to form, thus forming another face

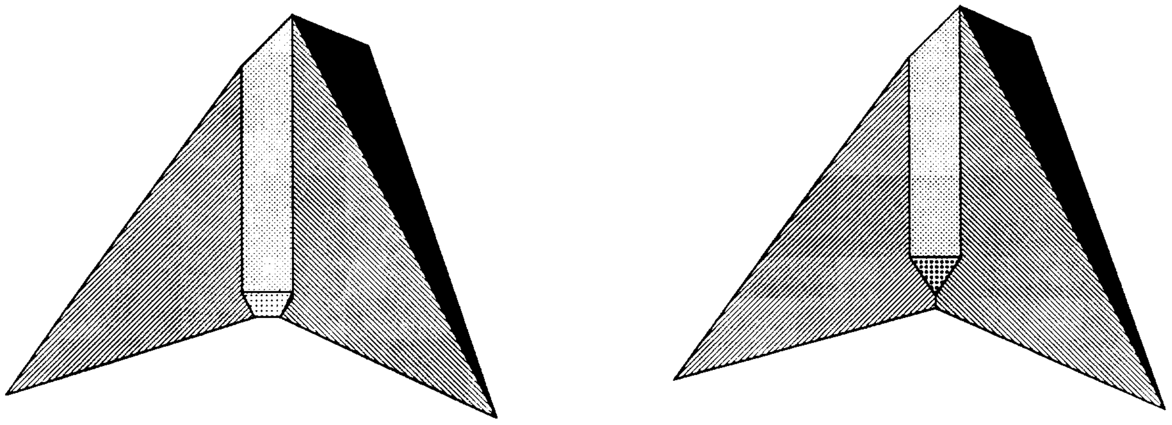


Figure 6.12. This figure shows the result should a bevel face with a relatively fast etch rate actually form along the edge E.

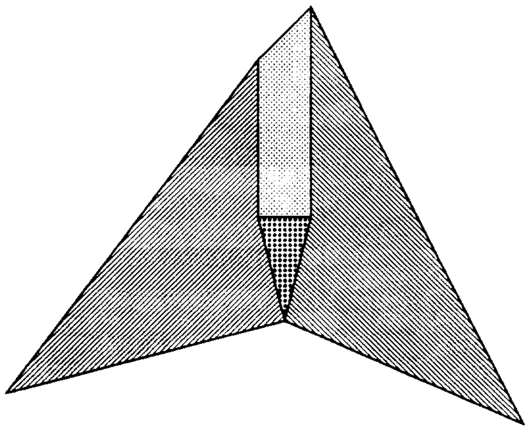


Figure 6.13. This figure shows the macroscopic result of the process pictured in the previous two figures. The triangular face shown is composed of several microscopic facets that progress up the shape.

etching up the shape. This unstable cycle will repeat very quickly on the microscopic level. The result macroscopically will be a triangular face that is "anchored" at one end to the original vertex, with a growth rate that is constrained by the growth rate of the bevel face above it. This is pictured in Figure 6.13. This result is interesting, because this triangular face will not necessarily be etching at the "natural" etch rate of a face with its orientation. This does not cause a contradiction with the etching rate model, because a different etching mechanism is at work.

Any etching simulator implementation must account for faces such as these. The solution we have adopted is to only re-examine features added to a shape when their size exceeds a user-specified tolerance. The result will be a sequence of ledges of virtual faces with standard orientations that together approximate the triangular face with an unusual orientation.

4. Removing Features

After etching for some time, features on a shape will disappear. Since this project only concerns itself with local contour changes, it is sufficient to detect when the length of an edge decreases to zero. Of course, many of the edges that are added start out with a length of zero, so it is important to be careful to only detect when an edge with a negative growth rate assumes a length of zero.

Removing features is not as simple as removing edges that shrink to nothing. First of all, the resulting vertex must be re-analyzed to find any new virtual faces, and to determine how the vertex splits. Beyond this, there are two somewhat tricky contour changes that must be accounted for: "handles" and "slits". These are pictured in Figures 6.14 and 6.15. While it is easy to see visually what must be done in these cases, the actual implementation turned out to be fairly complicated.

5. Masks

It is important that an etching simulator be able to model masks. This doesn't present a fundamental algorithmical challenge, but some comments are in order.

Masks can be supported by introducing the appropriate special cases into the relevant algorithms. First of all, there must be a way for the user to specify that a given face is masked. A masked face will, of course, have an etch rate of zero. Given that masked faces may be present, some statements can be made about how they behave. To analyze this, first consider the behavior of edges.

Obviously, at an edge between two masked edges no virtual faces will form. At an edge between a masked face an an unmasked one, there are three possible cases: The edge can be concave, convex, or neither (an edge is neither concave nor convex if it is between two faces that have equivalent normal vectors).

If the edge is convex, the edge develops as if there were an overhang (Figure 6.16). If the edge is concave, all proceeds normally at the beginning, that is, the slower candidate edge virtual faces tend to dominate. After a short time, an unmasked area directly adjacent to the masked area will be exposed. This will eventually cause behavior such as that seen at a convex edge. An edge that is neither concave nor convex will act like a concave edge. In all cases an overhang will develop.

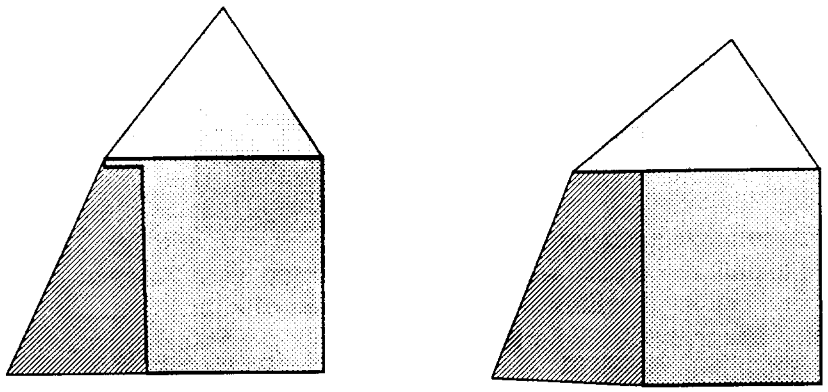


Figure 6.14. This figure shows a "handle" that results when an edge that separated two parallel edges disappears. The image on the left represents an object just before the "handle" appears, and the image on the right shows the same object just as the critical edge disappears.

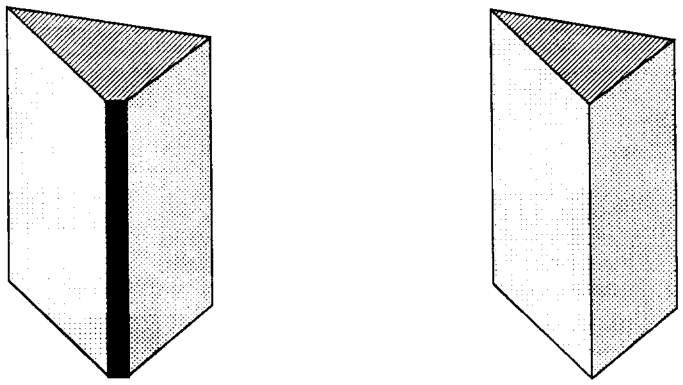


Figure 6.15. This figure shows a "slit" that results when a face disappears, but it still has two finite-length edges. Again, the left image shows an object just before the "slit" appears, and the right image shows the object just as the critical edges disappear.

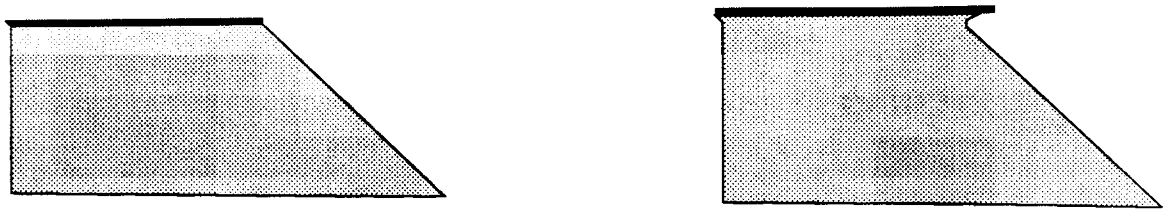


Figure 6.16. This figure illustrates the behavior of a convex edge between masked and unmasked faces.

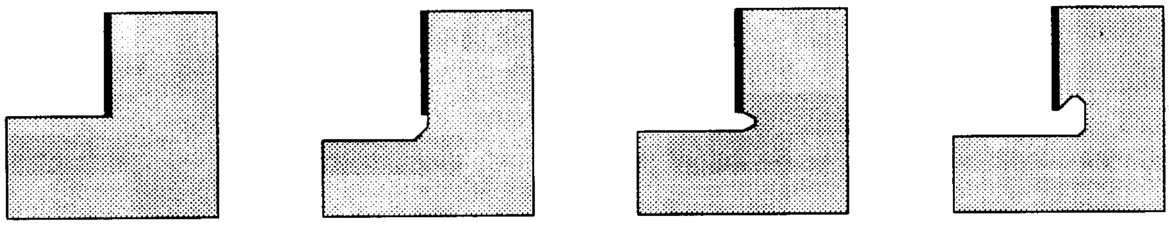


Figure 6.17. This figure illustrates the behavior of a concave edge between masked and unmasked faces. Only etching at the vertex in question and the faces adjacent to it are shown.

The vertex virtual faces that form in the presence of masks will, of course, be different than what happens in their absence. At any such vertex, two of the edges present will necessarily be edges along the masked face. These edges will both be either concave, convex, or neither. The correct behavior of the vertex with respect to vertex virtual faces can be deduced in a straightforward way by assuming that an "overhang" such as that pictured in Figure 6.17 is present.

CHAPTER 7

3D ETCHING SIMULATOR IMPLEMENTATION

The algorithms presented in the previous chapter lead in a straightforward way to a procedural implementation of a program that simulates the etching process. Such a program has been implemented in C using the UniGrafix graphics description language and the ug3 data structures. [Séquin, '85] For the most part, the simulator consists of a simple, brute-force implementation of the algorithms outlined in the previous chapter. In this chapter, we describe in a general way the software interface and pick out some highlights of the implementation. Some general observations about the ug3 data structures are also presented, as well as some observations about the difficulties inherant in the writing of applications such as this.

1. Software Interfaces

The first software interface is that which is used to read shapes into and out of the etching program. These three-dimensional shapes are represented with ascii text files in the UniGrafix language.

The etching simulator also lets one specify different etch rate functions. A description of an etch rate function may be stored in an ascii text file. This file contains a representation of the three basis vectors of the slowness shape. There is no requirement that these vectors be orthogonal. Following this, slowness values at the 98 key directions should be supplied. As a shortcut, the program allows one to specify only the first six slowness values. In this case, the program applies symmetry transformations to obtain the other key directions (i.e. it takes the (112) rate for the (11-2) direction). For details of the representation of an etch rate curve, refer to Appendix 1.

The third software interface allows the user to set parameters such as how long the given shape should be etched. These are controlled with command line arguments. Again, for complete details refer to the man pages (see appendix 2).

The program can only produce ''snapshots'' of the etching process in UniGrafix files. Using the ''Animator'' package being developed at U.C. Berkeley on a Silicon Graphics Iris workstation one may also view the etching process in real time.

2. Implementation Highlights

2.1. Etch Rate Model

From the etch rate function that the user specifies, the program must produce an etch rate for any arbitrary face orientation. The code that handles this is contained in the file slowness.c. There is another basic operation that is supported by this code: the selection of a set of candidate virtual faces, both at a vertex or along an edge of the object being etched.

A central problem in the supported operations is determining if a direction vector penetrates a certain triangular facet of the tessellated unit sphere. To do so, it is sufficient to solve a simple linear system of equations. Call the positions of the vertices of the facet $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$. Call the vector that we are testing $\vec{dir}$. The following system of equations must be solved:

$$a\hat{v}_1 + b\hat{v}_2 + c\hat{v}_3 = \hat{dir}$$

This system can be solved with a single 3x3 matrix inversion. If the coefficients a, b and c are all positive, the vector $\vec{dir}$ penetrates the given facet.

To calculate the slowness value at an arbitrary face orientation, the first step is to determine the slowness facet within which the normal vector of the face is contained. This is accomplished by checking all slowness facets with the above relation, until the correct one is found. As an optomization, the slowness facets are kept on an octree, and an inexpensive extent calculation is used to reject facets. With the correct facet found, the following formula yields the slowness value $\vec{s}$:

$$\vec{s} = \vec{d} \cdot abs\left(\frac{\vec{p} \cdot \vec{n}}{\vec{d} \cdot \vec{n}}\right)$$

where $\vec{d}$ is a unit vector in the direction of the slowness value, $\vec{p}$ is a vector from the origin to any point on the slowness facet, and $\vec{n}$ is the normal vector of the slowness facet.

To compute the set of candidate faces at a vertex, the convex subareas of the vertex figure of interest are first determined as was described in Chapter 5 with a straightforward algorithm. That analysis yields convex polygonal shapes. These shapes are triangulated to reduce the problem to finding the key etching directions that lie within such triangular slowness figures.

The final operation that needs to be supported is the finding of the set of candidate edge virtual faces. The normal vectors of the two faces in the original shape that make up the given edge are denoted by $\vec{\sigma}_1$ and $\vec{\sigma}_2$. The goal is to find all of the intersections of the great circle segment between these two vectors with edges between slowness facets. Every such edge is considered in turn. The vertices of these edges projected onto the unit sphere are denoted by $\vec{v}_1$ and $\vec{v}_2$.

Each vector pair determines a great circle that lies in the plane of the two vectors. A vector $\vec{l}$ along the intersection of two such planes is computed. This vector is normalized to produce $\hat{l}$. The two great circles intersect on the unit sphere at the points $\hat{l}$ and $-\hat{l}$. The vector $\vec{l}$ can be found from:

$$\vec{l} = (\vec{\sigma}_1 \, X \, \vec{\sigma}_2) \, X \, (\vec{v}_1 \, X \, \vec{v}_2)$$

If $\hat{l}$ is between $\vec{\sigma}_1$ and $\vec{\sigma}_2$, and it is also between $\vec{v}_1$ and $\vec{v}_2$, then the two great circle segments intersect at the point $\hat{l}$. The same test can be applied to the vector $-\hat{l}$. If neither test succeeds, the two great circle segments do not intersect.

2.2. Finidng Vertex/Edge Virtual Faces

The code to find vertex and edge virtual faces is located in the files vvface.c and evface.c, respectively. These modules were developed from the above algorithms in the obvious way, with sets implemented as linked lists. Particular attention was paid to making the program robust in the face of floating point inaccuracies. For example, the algorithm calls for the "first" intersection along an f-edge. In the actual implementation, the "first" intersection is actually a set of intersections, all within some value ε of the intersection that has the actual least value numerically.

2.3. Vertex Splitting

The code to split vertices is located in the file vsplit.c. This code is, also, largely a straightforward implementation of the algorithms presented in the previous chapter.

One significant optimization was made in this module: The algorithm calls for substantial re-calculation of intersection points. This is essentially the calculation of the intersection of three planes. This is a somewhat expensive floating point operation (it involves a matrix inversion, which is fairly expensive in the ug3 implementation). To avoid re-calculation, previously calculated values are stored on a hash table. To form a unique key, the addresses of the three faces that are being intersected are used. These addresses are first sorted to put them in a canonical order, and then their hex representation is concatenated to form a unique string key. This simple optimization yields a significant performance improvement.

2.4. Contour Re-linking

The code that re-links contours once the vertices are split is located in the file contours.c. One subtlety of this code is that contours must run either clockwise or counter-clockwise, as appropriate. The etching simulator program does not care which way they run, because it maintains its own normal vectors, but most programs that use the resulting UniGrafix file(s) usually will insist on this being correct. The approach taken is to ensure that one edge on every contour is found whose direction is known. Once this is found, the algorithm automatically makes all of the other edges point the proper way.

Finding one edge with known direction is easy with a contour that is being re-linked (i.e. for a face that was already present in the shape). It is somewhat more complicated for virtual faces. In this case, a search is made for any known contour that shares an edge with the virtual face. That edge must run in the opposite direction on the virtual face's contour.

2.5. Feature Removal

The event that triggers the removal of a feature from a shape is that of an edge's length decreasing to zero. When this happens, the edge is removed from both faces in which it occurs. When an edge is removed like this, further action might have to be taken: a face might disappear, or a "slit" or a "handle" (described previously) might appear. Each of these three possibilities must be explicitly checked for in any affected face.

A garbage-collection strategy was adopted to implement this. After any edge is removed, all faces in the object are checked for inconsistencies. First any face with two edges or fewer is removed. The next step is to check for slits and handles, and make the appropriate contour changes. Following this, any edge statements that are not referenced anywhere are removed (this is not automatically handled by the UniGrafix library). Next, all unreferenced vertices are removed. Finally, any vertices at which the local geometry has changed are re-examined for virtual faces.

3. The ug3 Data Structures

The ug3 data structures provided the framework around which the program was written. Unfortunately, the ug3 data structures are just that: data structures. They are not complete enough to be considered an abstract data type, because there are not enough high-level primitives supported. In addition, maintaining consistency of the data structures was a significant problem. Several of the specific limitations of ug3 are discussed in some detail in comment statements in the code, particularly at the beginning of the file main.c.

The only real general conclusion that I can draw from my experience with the ug3 data structures is that it is, in general, difficult to represent three-dimensional objects in a computer. Any attempt to create an abstract data type to represent such objects requires careful consideration and design, and involves a number of engineering trade-offs. In my opinion, important judgement criteria are speed, memory efficiency, avoidance of redundant information, and degree to which the implementation enforces type consistency.

I'd like to say a few words about the last point, because I think that this criterion is too often neglected in the design of data structures. ug3 was build around variant structures in standard (non-ansi) C. As supplied, ug3 enforces almost no type checking whatsoever. For example, every contour has a list of edges associated with it. This edge list is actually a linked list of pointers to ''UniGrafix statements'', which may be contours, edges, vertices, colors, etc. There is nothing that requires that the members of an edge list actually be edges! This is obviously undesirable. I took some steps to compensate for this lack in the ug3 implementation: For one thing, I wrote my program in ansi C, and I wrote routine prototypes for all of the ug3 library routines I called. This saved a tremendous amount of debugging time. Additionally, I wrote my own access routines for the ug3 data structures, and I added dynamic type checking wherever possible. This caught several errors for me, but this approach is not good software design (it attacks the problem at the wrong level of abstraction, and it's slow, too!). In my opinion, it would have been eminently preferable had the ug3 data structures and the ug3 library been designed with the idea of enforcing type consistency from the beginning.

CHAPTER 8

3D ETCHING SIMULATOR RESULTS

The results of etching various shapes is presented in the following pages. These shapes were etched with the etching simulator, and rendered with either the ''ugplot'' or ''ugdisp'' renderer.

The first series of images (Figure 8.1) shows the results of etching a cube shape that has faces perpendicular to the coordinate axes. Initially the cube obtains 48 virtual faces. Each edge has two virtual faces from the fast-etching (2,1,0) family. Each vertex has three virtual faces, all from the (2,2,1) family (the fastest orientation in the etch rate function).

The following example (Figure 8.2) shows a saddle vertex of the type studied in detail in Chapter 6. The top vertex is a ''mostly convex'' saddle vertex, that is, one in which all of the edges fit into one half-space, a half-space that also contains the bulk of the object. The bottom saddle vertex is neither ''mostly convex'' nor ''mostly concave''.

Next are several more complicated shapes. (Figures 8.3 through 8.6). These examples show many interesting and complicated structures. The next two examples (Figures 8.7 and 8.8) demonstrate the effects of etching a shape in which some faces are masked.

Figure 8.9 shows a shape that etches according to the process discussed in Chapter 6 in the ''Re-analysis of Features'' section. To make the process more visable, a contrived slowness diagram that is not similar to the model developed in Chapter 5 was used. This slowness shape only has vertices at the (1,0,0) orientations (where the slowness is 4), and at the (1,1,0) orientations (where the slowness is 1). The small facets etching up the shape are at the (0,1,–1) orientation. To make the facets fast enough to produce the ''ripple'' effect, the slowness value at this orientation was set to 0.1. This slowness function is not meant to be realistic; it was produced specifically to make the ''ripple'' effect as visable as possible.

Finally, there is an example of an object with a large number of concave vertices. An easy way to generate such an object is to turn a convex polygon ''inside-out'' by making the face normals point into the object. This has been done in Figure 8.10. The starting shape was a regular octahedron. For this figure, a slightly different etch rate function was used. The same key face slowness values were used, but the following basis vectors were used:

$\vec{l}_x = <1.0, 0.0, 0.0>$
$\vec{l}_y = <0.2, 1.0, 0.0>$
$\vec{l}_z = <0.0, 0.1, 1.0>$

After etching for sufficient time, this shape becomes a skewed cube with only (1,0,0)-type faces.

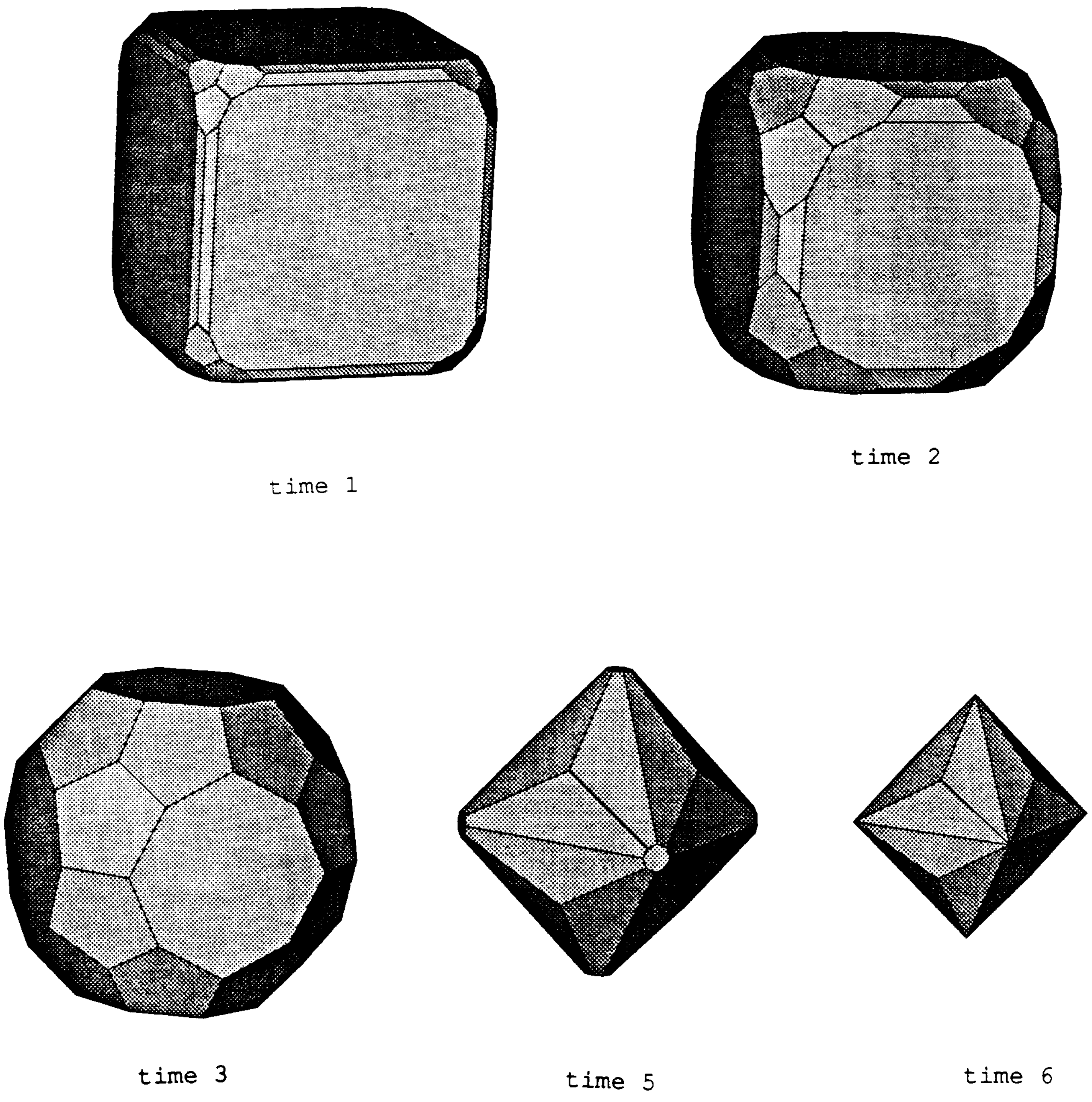


Figure 8.1. These pictures show the results of etching a cube with faces normal to the coordinate axes.

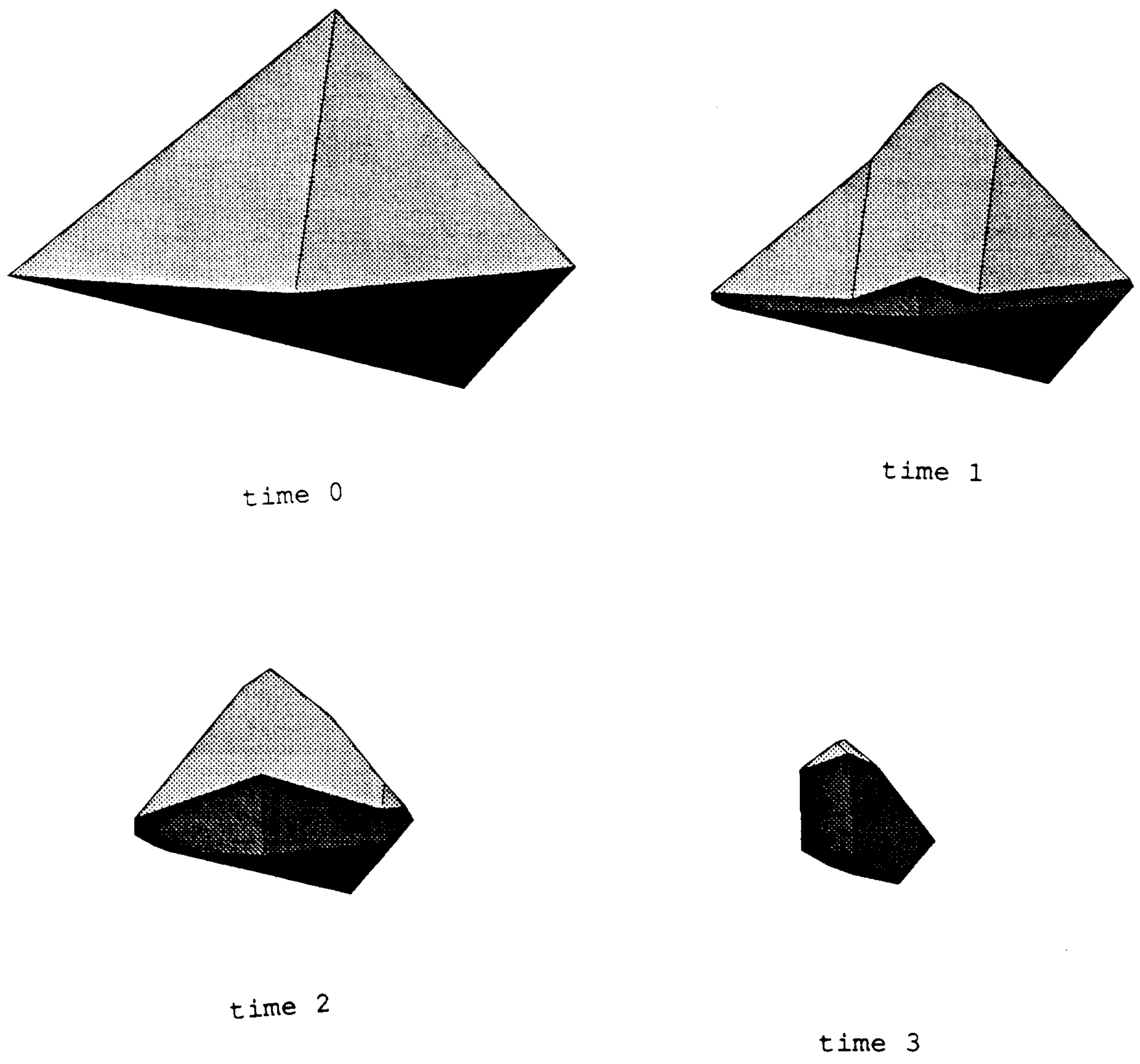


Figure 8.2. These pictures show the results of etching a shape with two saddle vertices.

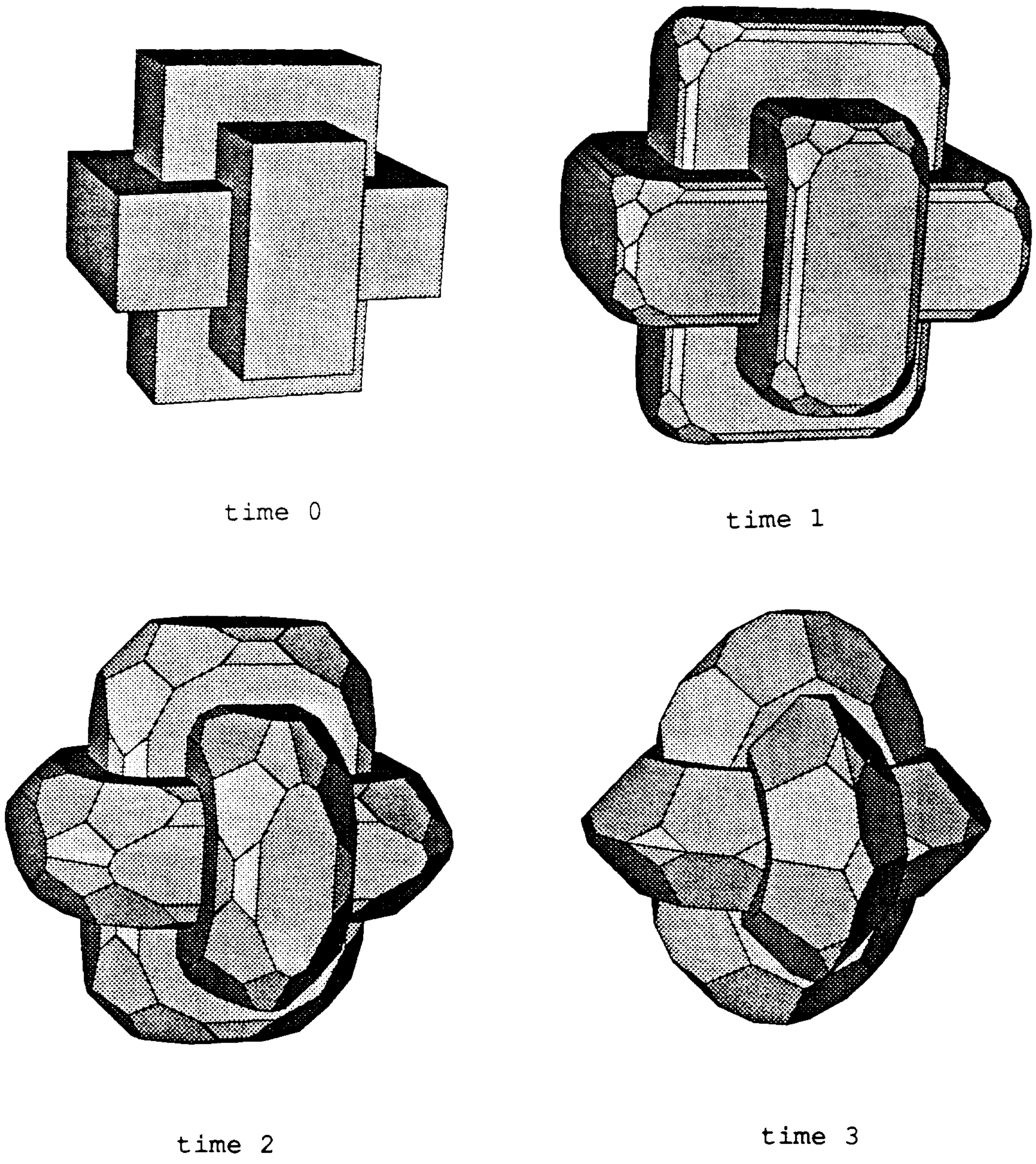


Figure 8.3. These pictures show the results of etching a somewhat complicated shape. The original object contains only (1,0,0) type faces.

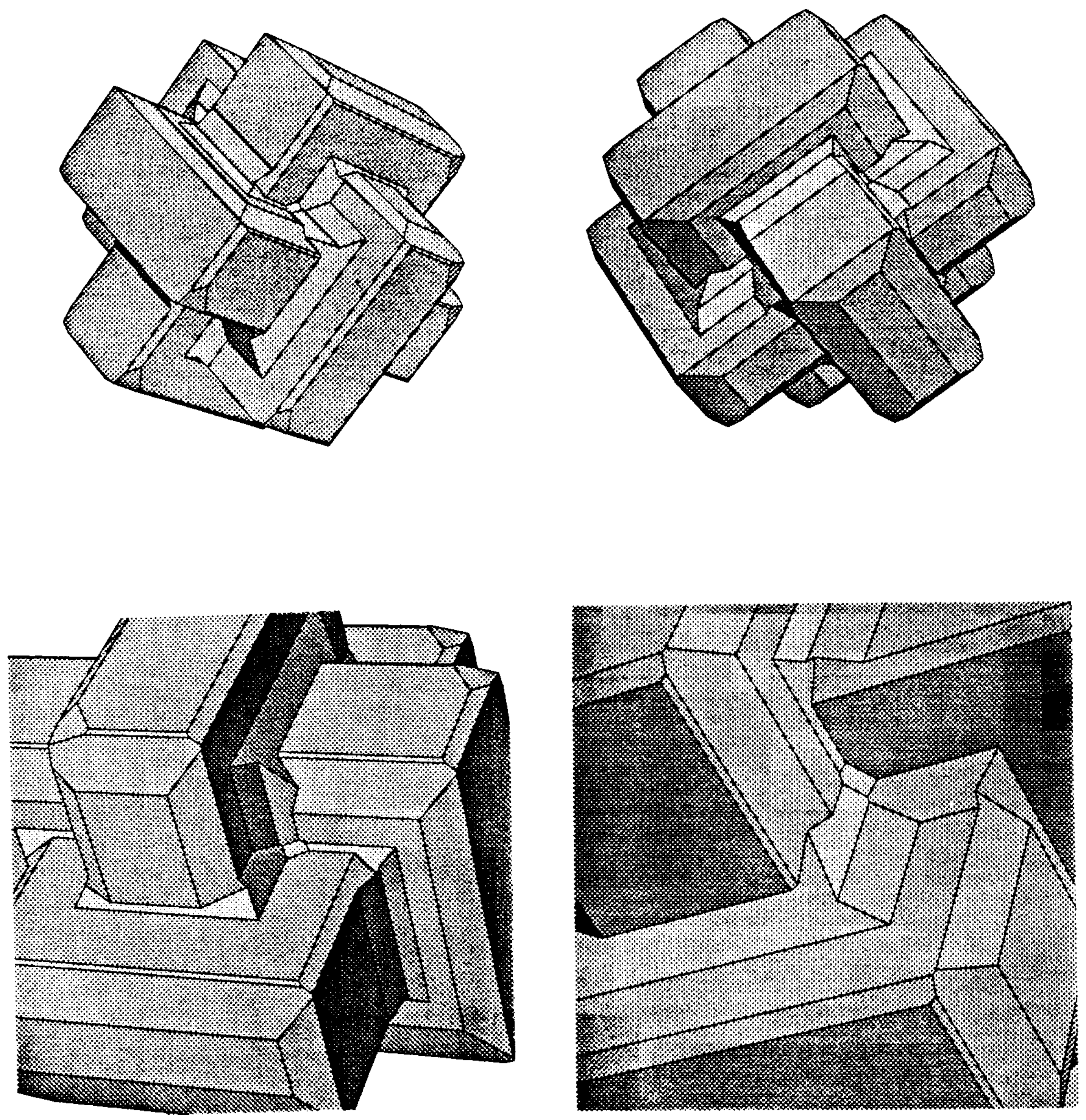


Figure 8.4. These pictures show four views of the result of etching the same shape as in Figure 8.3 rotated with respect to the etch rate function so that the original faces in the object etch relatively quickly. This causes interesting formations at the concave vertices. These pictures were not drawn to scale.

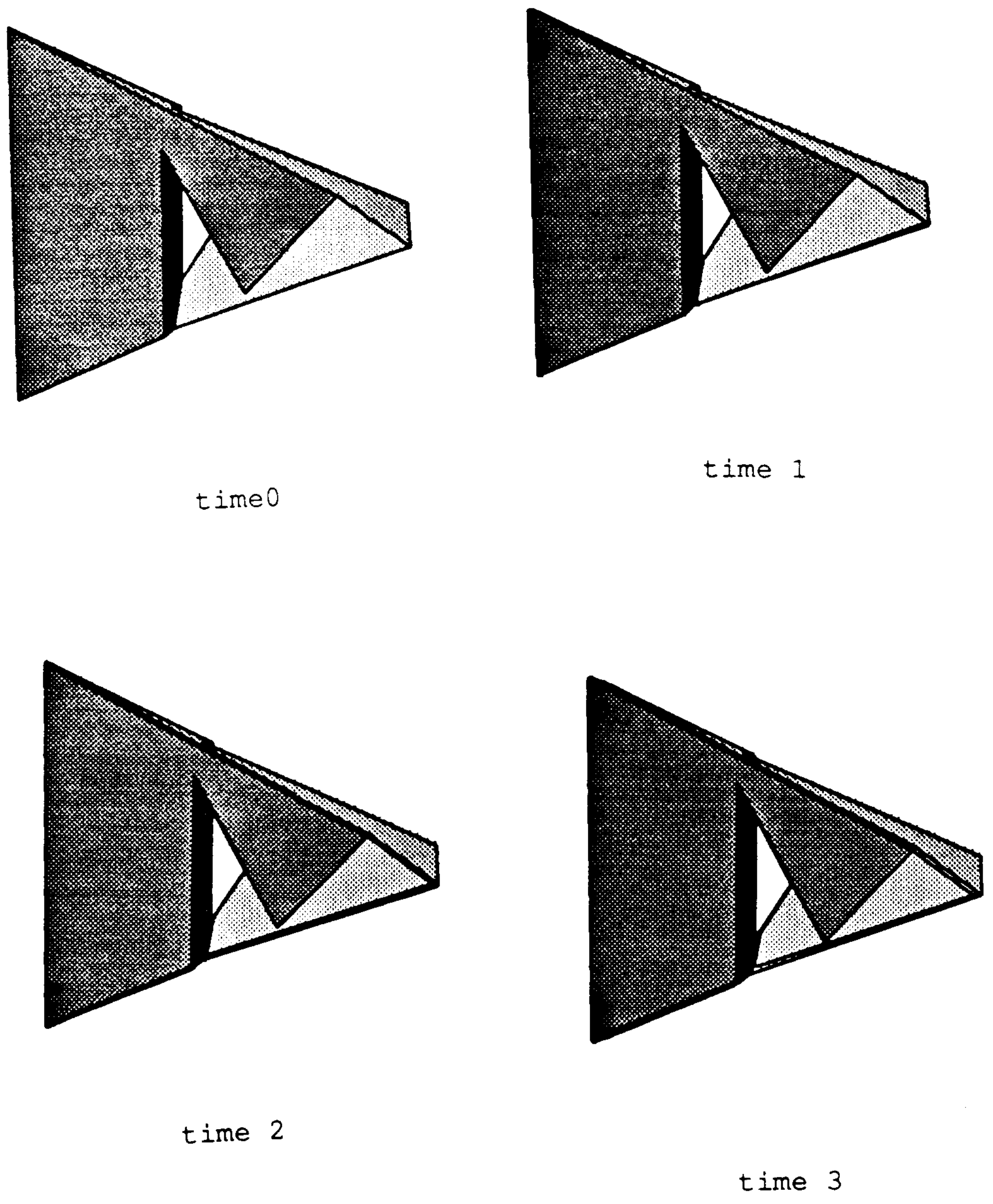


Figure 8.5. These pictures show the results of etching a Szilassi minimal torus.

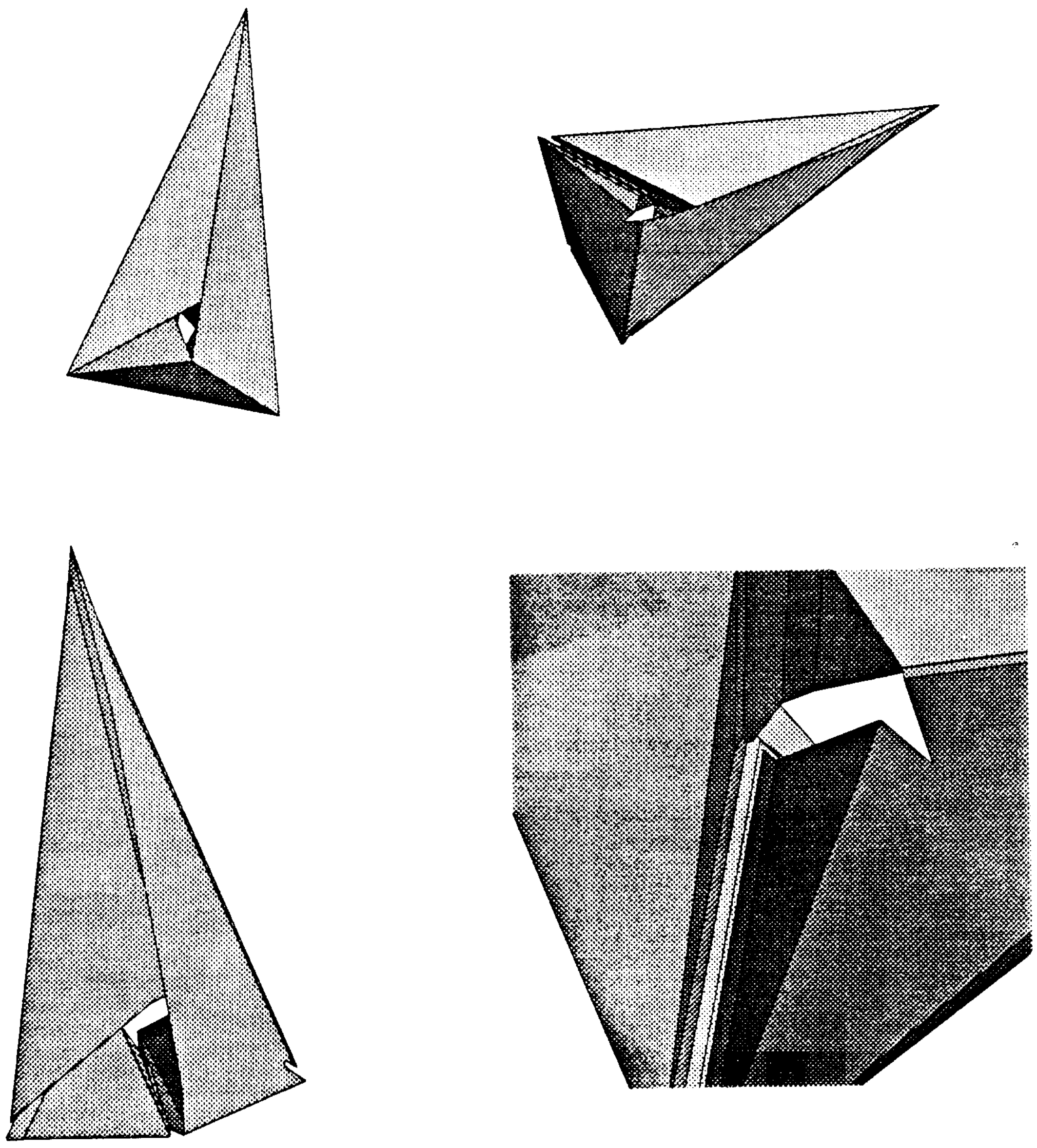


Figure 8.6. These pictures show the results of etching a Csaszar minimal torus. The picture in the upper left hand corner is the original torus, and the other pictures show three different views of the results of etching this shape for a short time. These pictures were not drawn to scale.

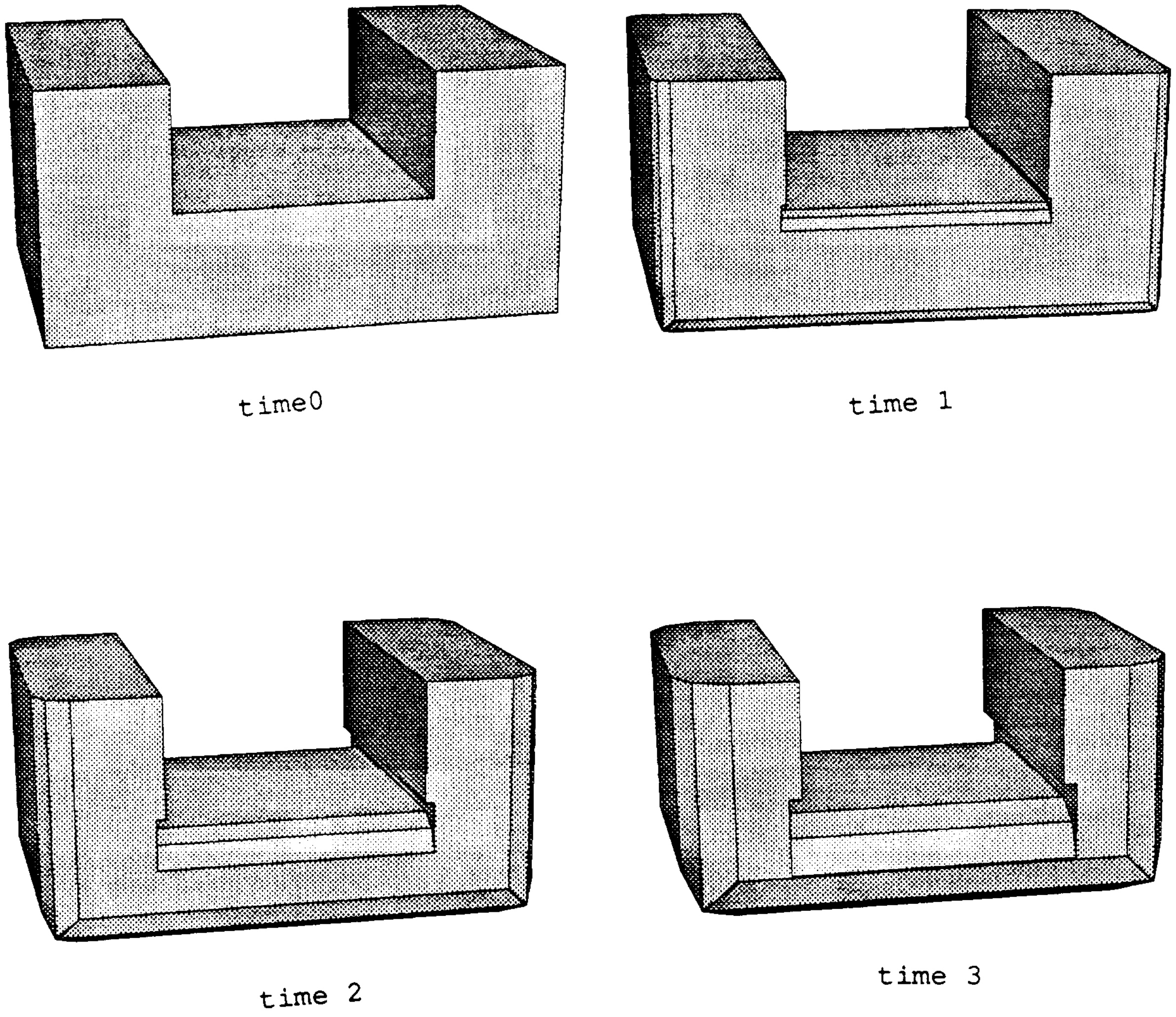


Figure 8.7. These pictures show the results of etching a shape that has some faces masked. The masked faces are the two top faces, and the two vertical faces on the inside of the "U". To make the ledge more visible, this shape was etched with an etching function that had a smaller differential between the slowest and fastest rates. The etching rates used were $s_{0,0,1} = 4.0$, $s_{0,1,1} = 2.0$, $s_{1,1,1} = 1.5$, $s_{0,1,2} = 1.3$, $s_{1,1,1} = 1.2$ and $s_{1,2,2} = 1.0$.

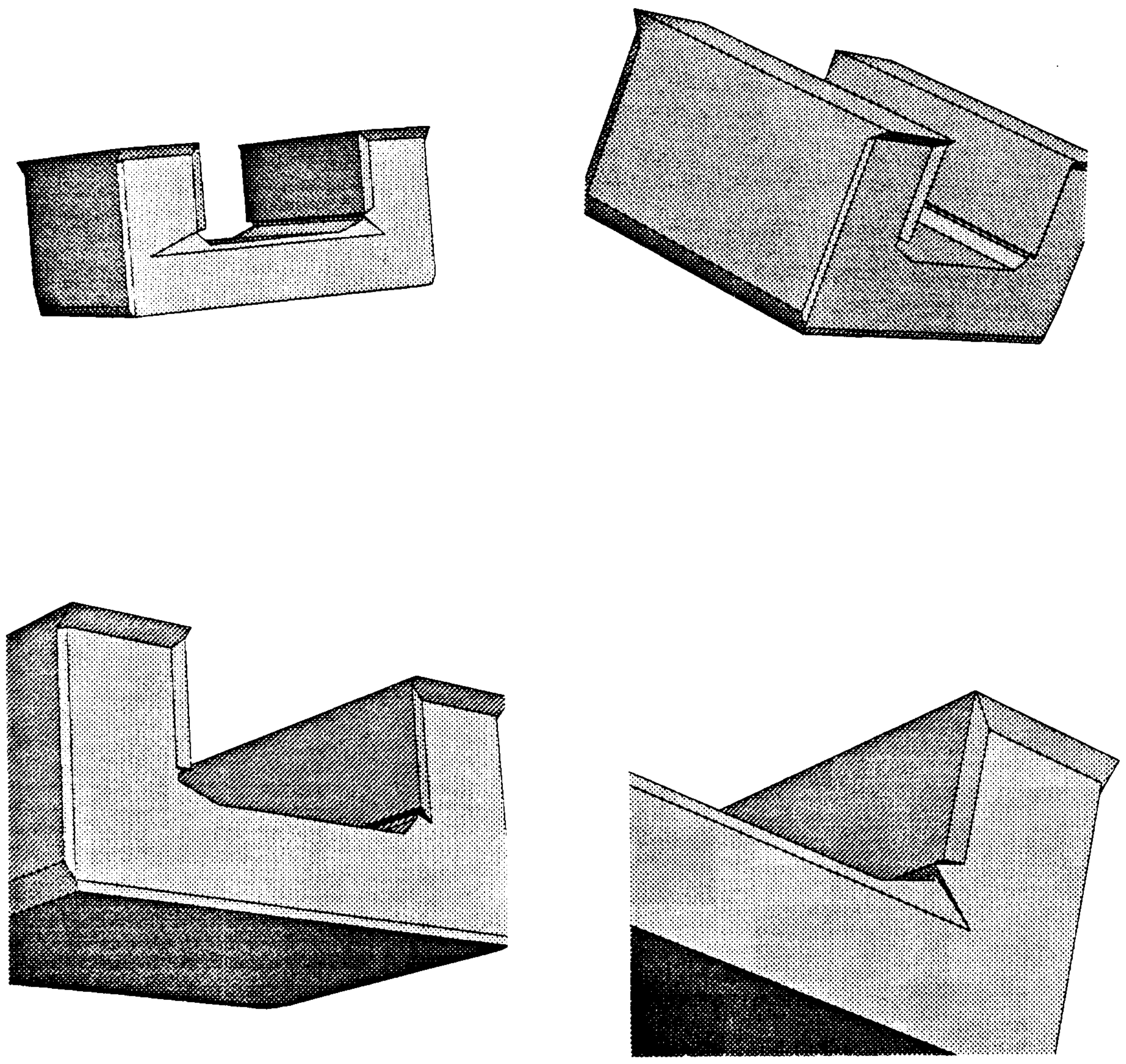


Figure 8.8. These pictures show four views of the results of etching the "U" shape of Figure 8.9 rotated so that the original faces are in directions that etch more quickly. The normal etch rate function was used for these pictures. These pictures were not drawn to scale.

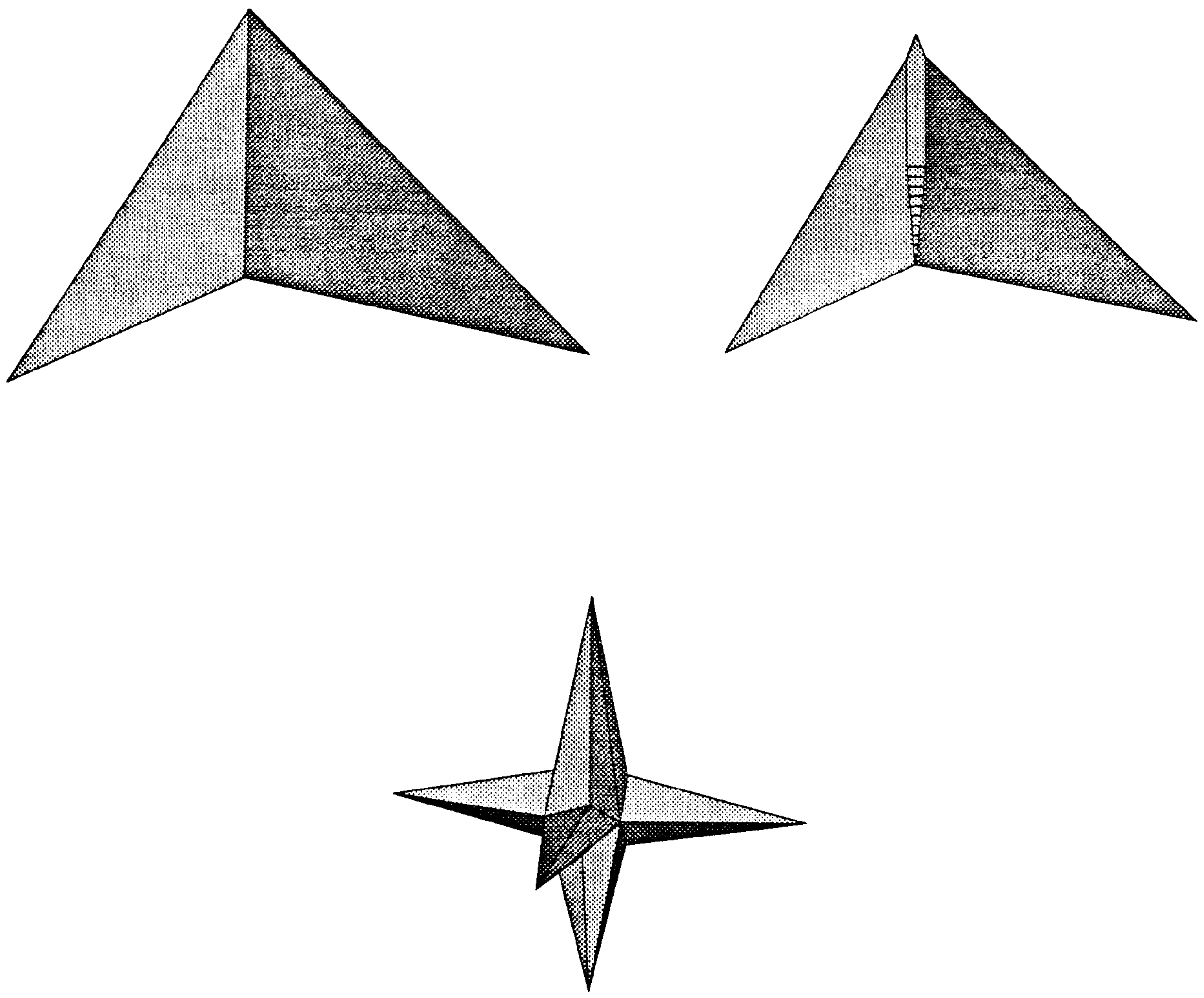


Figure 8.9. These pictures show the result of etching a shape such as that discussed in the ''Re-analysis of Features'' section of Chapter 6 (see Figure 6.13). The top left picture is the original shape, and the top right picture is the result after etching it for a short time. The picture on the bottom is the slowness shape that was used.

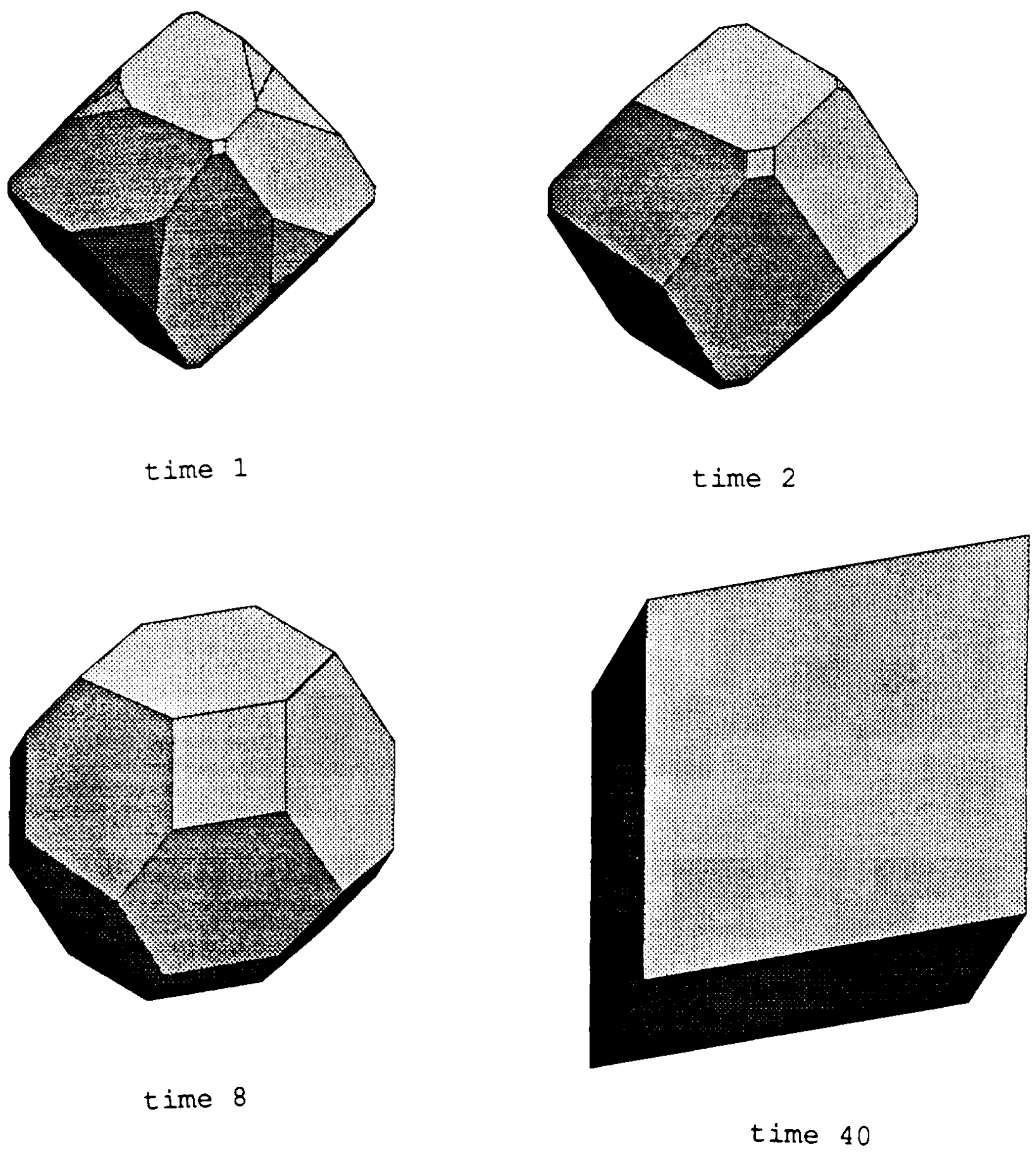


Figure 8.10. These pictures show the results of etching a regular octahedron that has been turned "inside-out" so that the normal vectors point into the object. The etching simulator treats this like a figure with all concave edges that slowly grows as it is "etched". The etching function used for these pictures was non-orthogonal, which causes the extra triangular faces visable at times.

CHAPTER 9

SUMMARY

In this project, we began by making a model of the etching behavior of a simple 2D solid. From this model, we were able to obtain some plausible values for etch rates, but more importantly, we gained an intuitive insight into the shapes that plausible and self-consistent etching functions might assume. We concluded that such functions can be approximated by a set of slowness values at key etching directions (the (1,0), (1,1), and (2,1) families) which are connected by straight lines. Using our etching model, we also built a 2D simulator that produced shapes that agree closely with the expected results. [Séquin, '89]

Building on our experiences in 2D, we did a less detailed analysis in 3D with the sole aim of producing plausible and self-consistent etch rates at the key 3D etching directions (the (1,0,0), (1,1,0), (2,1,0), (1,1,1), (2,1,1) and (2,2,1) families). Based on the intuition gained in 2D, we assumed that an etch rate function could be closely approximated by a slowness shape that consists of triangular facets between the key etching directions.

Using this model of etching, we built a first prototype of a 3D etching simulator. Due to time constraints, this simulator does not handle non-local contour changes (as when a hole gets etched through a solid). Still, this simulator provided us with a testbed for our etching model. The results were positive: We saw the kinds of results that we expected to see based on the shapes that appear in the literature describing actual etching experiments on silicon.

To our etching simulator, we added the ability to mask off certain portions of a solid from the ''etchant''. This was done in the hope that our etching simulator could be used as a simulation tool by those who are exploring micro-fabrication.

Acknowledgements

Many thanks to Carlo Séquin, my advisor, for his ideas and encouragement, and for putting up with the many obstacles we faced. I also thank Kevin Smith, for his help in unraveling the ug3 data structures, and Tien Truong for his convex hull module. I'd also like to thank the Free Software Foundation, for making available their excellent ''gcc'' compiler and ''gdb'' debugger, without which the debugging process would have been much more gruelling.

BIBLIOGRAPHY

H. Seidel and L. Csepregi, ''Studies on the Anisotropy and Selectivity of Etchants Used for the Fabrication of Stress-free Structures,'' 1982 Spring Meeting, Electrochemical Society, Montreal, Canada, Abstract No. 123, (1982).

F. C. Frank, ''Growth and Perfection of Crystals'' (John Wiley and Sons, New York, 1958) p. 411.

D. J. Barber, F. C. Frank et al, ''Prediction of ion-bombarded surface topographies using Frank's kinematic theory of crystal dissolution'', J. Mater. Sci. 8 (1973) 1030.

C. H. Séquin, ''Microfabrication on the Macintosh,'' Proc. of the 5th USENIX Workshop on Computer Graphics, Monterey. CA, Nov. 1989, p. 1-15.

C. H. Séquin, '' The Berkeley UNIGRAFIX Tools, Version 2.5'', CS Division Report No. UCB/CSD 86/281, University of California, Berkeley, CA, 1985

Appendix 1

Manual Pages

NAME

etch – simulates anisotropic etching

SYNOPSIS

etch [*options*]

DESCRIPTION

etch is a program that reads a shape described by a UniGrafix file, and subjects that shape to anisotropic etching. One may use the default etch rate function (see MS Report by Bill Foote), or specify a different one. The etch rate function may be specified by a series of slowness values at ''key'' faces, or it may be specfied by a UniGrafix file that describes a slowness surface. The program can output a series of UniGrafix files at different times, or it can be used on an Iris workstation to produce an animation of the etching process.

A mask over any face may be specified by giving that face a color called ''masked''. If the shape's UniGrafix file has colors called ''color_001'', ''color_011'', ''color_012'', ''color_111'', ''color_112'', ''color_122'' and ''color_evface'' then virtual faces will be colored apropriately (edge virtual faces that don't happen to be at key etch rates will be colored ''color_evface'').

The available *options* are:

–**slowness** <*file*>
: Tells the program to read a series of slowness values at ''key'' faces from *file*. Call the program with ''–slowness_help'' for a description of the format of this file.

–**slowness_help**
: Tells the program to output a help message that describes the format of an etch rate function for the ''–slowness'' option.

–**slowness_print**
: Tells the program to produce a UniGrafix description of the slowness shape on **stdout**.

–**slowness_log**
: The same as ''–slowness_print'', except every vertex is scaled such that its length is the logarithm of its slowness value.

–**slowness_unit**
: The same as ''–slowness_print'', except that every vertex is normalized. This produces the projection of the key etch rates onto the unit sphere with the appropriate triangular facets.

–**slowness_uni** <*file*>
: Tells the program to read the slowness shape from the UniGrafix file *file*. It is assumed that this shape is composed entirely of triangular facets, and that it describes a valid function in 3D. Such a surface must be a single-valued shell (in polar coordinates) surrounding the origin.

–**i** <*file*> Default: **stdin**
: Specifies the initial shape to be etched.

–**m** <*number*>
: Including the ''–m'' option will cause ''snapshots'' of the etching process to be output. If the ''–p'' option is not specified, output will go to the files ''out.000'', ''out.001'', etc. One file will be created every *number* units of simulated time.

–**p**
: Used in conjunction with the ''–m'' option, tells the program to output each frame to the file ''out'', and wait for a keystroke after each file is output.

–**limit** <*number*>
: Tells the program to terminate once it has produced *limit* snapshots (see the ''–m'' option).

–**f** <*number*>
: Tells the program to terminate when the simulated etching time reaches *number*. If the ''–m''

option was not specified the shape (after etching) will be output to **stdout**.

–reexamine_len <*number*> Default: infinity (i.e. no edge re-examination)
Tells the program to examine an edge that is added at a virtual face to see if it splits once its length becomes *number*.

–vface_tolerence <*number*> Default: 1 degree
Tells the program to not add a virtual face if its normal is within *number* degrees of any face in its neighborhood.

–no_virtual
Tells the program to not add any virtual faces.

–d Produces debugging information on **stdout**.

–file <*file*>
Causes options to be read from *file* instead of from the command line.

FILES

~ug/bin/etch

SEE ALSO

2detch(UG)

BUGS

Non-local contour changes are not detected.

AUTHOR

Bill Foote

NAME

2detch – simulate the etching of a 2 dimensional polygonal figure.

SYNOPSIS

2detch etchmodel [*options*] < startshape > finalshape

DESCRIPTION

2detch is a program which reads an etching model (such as is produced by *2detch_faces*) and an intial shape, and calculates what that shape will look like after etching for the specified time. The starting and final shapes are in UniGrafix format. The format of an etch model is described in 2detch_faces(UG). In addition to generating the final shape in UniGrafix format, *2detch* will display intermediate shapes on a graphics screen.

The available *options* are:

–f *<number>* Default: f=1
: This sets the finish time for etching. The final UniGrafix file will be the result of etching the initial shape until the finish time.

–m *<number>* Default: m=0
: This sets the time step between subsequent frames for a ''movie''. For each ''frame'' a UniGrafix file is generated, with file names in the sequence ''etch.000'', ''etch.001'', etc.

–s *<number>* Default: s=0
: This sets the time step between subsequent frames for the graphic display. If this value is 0 (the default), no graphics display will be generated.

–u *<file_name>* Default: stdin
: Specifies the UniGrafix file of the original shape.

–p If -p is present, etch will pause between screens (until return is pressed). Entering "?" will give a list of options that can be entered at this point. One of the options allows you to generate a UniGrafix file of an intermediate shape.

–file *<file_name>* Default: none
: Directs the program to take all command line arguments from the file file_name rather than from the command line. If this argument is present it must be the only argument on the actual command line.

–d Turns on debug mode and prints information about program execution to **stderr**. In addition, some extra information about the final shape is included as a comment in the output file.

IMPLEMENTATION

After reading in the starting shape, **2detch** adds all ''important'' edges at each corner, first checking that the resulting edge will indeed grow. Following this, it calculates an edge velocity for each edge, then a vertex velocity for each vertex. Each edge is assigned a ''time to live'', that is, the amount of time until that edge is etched away to nothing. Similarly, vertices are assigned a ''time to collide'', that is, the amount of time until they collide with an edge. At each timestep, vertices are moved the appropriate amount, and contour changes are checked for. If a contour change happens (i.e. an edge disappears) some of the surrounding velocities and time factors are re-calculated.

EXAMPLE

```
cat rate_parameters | 2detch_rate | 2detch_faces > etchmodel
2detch etchmodel -f 10 -s .1 < startshape > finalshape
```

SEE ALSO

2detch_rate(UG), 2detch_faces(UG)

AUTHOR

Bill Foote

NAME

2detch_rate – generate etch rate curve

SYNOPSIS

2detch_rate [*options*] < parameters > ratecurve

DESCRIPTION

2detch_rate is a program which reads the characteristic etching parameters of a 2D crystal, and generates the etch rate curve. It is expected that the eleven parameters (lh, lvx, lvy, ah, bh, ch, dh, av, bv, cv and dv) will be present in order in the file ''parameters''. The etch rate curve is output in polar coordinates as a sequence of (r,theta) points, one pair per line, followed by a short comment (surrounded by {}) indicating the original parameters. It is expected that this output will be used by a program that performs linear interpolation in polar coordinates between two adjacent points. All angles are measured in radians.

The available *options* are:

–n <*number*> Default: n=4
This sets the number of points on the etch rate curve to specify between minima and maxima. This influences how accurately the etch rate curve is approximated. Setting this value too high could result in slower execution of programs that use this curve.

–r <*number*> Default: r=0.0
This is an angle in radians that specifies the ''curve rotation factor'', i.e. how much the shape is rotated around the origin. The lattice size vectors lh and lv are constrained to have positive components, so this parameter allows one to represent a crystal with an arbitrary orientation.

–d Turns on debug mode and prints information about program execution to **stderr**.

IMPLEMENTATION

This program works by first determining the orientations of the sixteen ''key'' rates ((1,0), (2,1), (1,1), (1,2), (0,1), (-1, 2), etc.). It then evenly divides the regions between successive pairs of ''key'' rates, and adds new orientatons to the list. Following this, it calculates the etch rate at each orientation from the lattice parameters. Finally, it sorts the list of orientations, and outputs it in polar form.

EXAMPLE

```
cat rate_parameters | 2detch_rate | 2detch_faces > etchmodel
2detch_etch etchmodel -f 10 -s .1 < startshape > finalshape
```

SEE ALSO

2detch_faces(UG), 2detch_etch(UG), 2detch_rate_to_ug(UG)

AUTHOR

Bill Foote

NAME

2detch_faces – determine ''important'' faces for an etch rate curve

SYNOPSIS

2detch_faces [*options*] < ratecurve > etchmodel

DESCRIPTION

2detch_faces is a program which reads an etch rate curve (such as is produced by *2detch_rate*) and determines the ''important'' faces implied by that curve. The important faces are those that will become visible after etching at a corner for sufficient time. An example of an important face is the <2,1> edge.

2detch_faces expects the input to be a series of (r,theta) points, one pair per line, optionally followed by a comment. The only requirement is that the comment commence with a non-numeric character. Output consists of these elements: the original etch rate curve, the word ''concave'', numbers indicating the orientations of important edges at a concave corner, the word ''convex'', numbers indicating the orientations of important edges at a convex corner, and the comment taken from the input. All angles are measured in radians. An etch rate curve is defined to be the linear interpolation (in polar coordinates) between adjacent points specified in the input.

The available *options* are:

–i <*number*> Default: i=.01
: This sets the minimum time rate of change of an edge for it to be considered important. This applies to concave-corner edges as well as to convex-corner edges.

–a <*number*> Default: a=0.0
: This sets the minimum angle between two non-minimal important edges at a concave corner. Setting this parameter imposes another constraint on the number of important edges at a concave corner.

–d Turns on debug mode and prints information about program execution to **stderr**.

IMPLEMENTATION

For any etch rate curve, the edges corresponding to local maxima are always important for convex corners. Some other edges may be important – this is a function of the second derivative (with respect to theta) of the etch rate function. For concave corners, all edges corresponding to local minima are important, but some other faces may be important, too.

2detch_faces first determines the local minima and local maxima of the etch rate curve. It then subdivides the region between a local maximum and the nearest local minimum until the time rate of change of the length of an edge is less than the minimum. This works for both concave and convex corners.

NOTE: For most etch rate curves, the only important faces are those that correspond to the local minima and maxima of the etch rate curve.

EXAMPLE

```
cat rate_parameters | 2detch_rate | 2detch_faces > etchmodel
2detch_etch etchmodel -f 10 -s .1 < startshape > finalshape
```

SEE ALSO

2detch_rate(UG), 2detch_(UG)

AUTHOR

Bill Foote

NAME

2detch_rate_to_ug – transform polar rate curve into unigrafix

SYNOPSIS

2detch_rate_to_ug [*options*] < etchmodel

DESCRIPTION

2detch_rate_to_ug is a program which reads an etching model (such as is produced by *2detch_faces*) and produces a straight-line approximation of the curve in unigrafix format. This allows one to plot the rate curve using ugplot.

The available *options* are:

–n <*number*> Default: n=8
: This sets the number of lines with which to approximate each polar segment. Since the rate curve is interpolated in polar coordinates, it is really a curved shape between two points on the curve. This parameter sets how many straight lines are used to approximate this curve.

–d Turns on debug mode and prints information about program execution to **stderr**.

EXAMPLE

```
cat rate_parameters | 2detch_rate | 2detch_rate_to_ug | ugplot -dp
```

SEE ALSO

2detch_rate(UG)

AUTHOR

Bill Foote

Appendix 2

Description of 3D etch rate file format

RATE FILE HELP

The etch slowness curve is specified by three basis vectors, followed by a number of slowness values at key etching directions. The basis vectors determine the crystal axes. They are called l(x), l(y) and l(z), and are input in cartesian coordinates (x, y, then z). The file should begin with 9 numbers, for l(x), l(y) and l(z) (in that order). The program normalizes these vectors to unit length.

Following this, there should be a number of slowness values at key etching directions. A direction is specified by a vector in the direction of the face normal (pointing outwards). In the notation below, (012) is a face who's normal is in the direction of 0*l(x) + 1*l(y) + 2*l(z).

There are a total of 98 key etch rate directions. An etch slowness file can have slowness values for all 98 key etch rate directions. If desired, the file can contain slowness values for only the first six key etch rate directions (i.e. the first line below). If this is done, the program will assume that the etch slowness curve is completely symmetrical (i.e. the (012) etching slowness is the same as (021), (102), (201), (210), and (120).).

The program expects the slowness values in the following order:

```
( 0 0 1)  ( 0 1 1)  ( 1 1 1)  ( 0 1 2)  ( 1 1 2)  ( 1 2 2)
( 0 1 0)                      ( 0 2 1)  ( 1 2 1)
          ( 1 0 1)            ( 1 0 2)            ( 2 1 2)
          ( 1 1 0)            ( 1 2 0)            ( 2 2 1)
( 1 0 0)                      ( 2 1 0)  ( 2 1 1)
                              ( 2 0 1)

( 0 0-1)  ( 0 1-1)  ( 1 1-1)  ( 0 1-2)  ( 1 1-2)  ( 1 2-2)
( 0-1 0)  ( 0-1 1)  ( 1-1 1)  ( 0-2 1)  ( 1-2 1)  ( 1-2 2)
          ( 1 0-1)            ( 1 0-2)            ( 2 1-2)
          ( 1-1 0)            ( 1-2 0)            ( 2-2 1)
(-1 0 0)  (-1 1 0)  (-1 1 1)  (-2 1 0)  (-2 1 1)  (-2 2 1)
          (-1 0 1)            (-2 0 1)            (-2 1 2)

                              ( 0-1 2)  ( 1-1 2)
                              ( 0 2-1)  ( 1 2-1)
                              (-1 0 2)  (-1 1 2)
                              (-1 2 0)  (-1 2 1)
                              ( 2-1 0)  ( 2-1 1)
                              ( 2 0-1)  ( 2 1-1)

          ( 0-1-1)  ( 1-1-1)  ( 0-1-2)  ( 1-1-2)  ( 1-2-2)
                              ( 0-2-1)  ( 1-2-1)
          (-1 0-1)  (-1 1-1)  (-1 0-2)  (-1 1-2)  (-2 1-2)
          (-1-1 0)  (-1-1 1)  (-1-2 0)  (-1-2 1)  (-2-2 1)
                              (-2-1 0)  (-2-1 1)
                              (-2 0-1)  (-2 1-1)

                                                  (-1 2 2)

                                                  ( 2-1 2)
                                                  ( 2 2-1)


                                                  (-1 2-2)
                                                  (-1-2 2)
                                                  ( 2-1-2)
                                                  ( 2-2-1)
                                                  (-2 2-1)
                              (-2-1 2)

                    (-1-1 2)
                    (-1 2-1)


                    ( 2-1-1)


(-1-1-1)            (-1-1-2)  (-1-2-2)
                    (-1-2-1)
                              (-2-1-2)
                              (-2-2-1)
                    (-2-1-1)
```

The program reads each octant of the etch rate curve. Each octant is divided up into 6 sections, each of which contain slowness points from the key etching directions. Along section boundaries adjacent sections share key etching directions -- if the program has already read the slowness for a given key etching direction, it uses the value it has already read. This is why there are gaps in the above list of key etch directions.