MICROSTRUCTURE EVOLUTION AND INTERCONNECT RELIABILITY

by

WALID R. FAYAD

Ingenieur de l'Ecole Polytechnique, France, 1995
Ingenieur de l'Ecole Centrale de Paris, France, 1995
S.M. Civil and Environmental Engineering, MIT, 1997

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY IN ELECTRONIC MATERIALS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000

Signature of Author _____
Department of Civil and Environmental Engineering
March 17, 2000

Certified by _____
Carl V. Thompson
Stavros Salapatas Professor of Materials Science and Engineering
Thesis Supervisor

Certified by _____
Jerome J. Connor
Professor of Civil and Environmental Engineering
Thesis Committee Chairman

Accepted by _____
Daniele Veneziano
Chairman, Departmental Committee on Graduate Studies

MICROSTRUCTURE EVOLUTION AND INTERCONNECT RELIABILITY

by

WALID R. FAYAD

Submitted to the Department of Civil and Environmental Engineering on March 17, 2000 in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electronic Materials

ABSTRACT

In the context of predicting the effects of geometry, microstructure, and processing conditions on electromigration (EM) induced interconnect failure, normal grain growth in thin films was studied, analytic models were built for the grain structure statistics in 2D and 3D interconnects, and simulation programs were developed for generation of process and complex-geometry-sensitive interconnect structures. The models were validated through simulations and experiments and were integrated into tools for circuit-design-level interconnect reliability predictions.

The universal scaling behavior of 2D normal grain growth was demonstrated and characterized using a simulation of 2D grain growth (GGSim). We showed that the constant rate of change of the average grain area is equal to the grain boundary mobility constant μ. We also found that the steady state grain size distribution obtained using our simulation technique, as well as those reported in experiments on simple model systems and those reported for very different simulation techniques, are all very well fit by a Weibull distribution function with the dimensionless parameter $\beta = 5/2$, and are better fit by this function than the lognormal, Gamma or Rayleigh functions.

The 2D simulation was used to simulate the development of film structures with drag-induced lognormal grain size distributions from which interconnect strips were etched and then annealed, in order to analyze the statistics of as-patterned, as well as post-pattern annealed, interconnect grain structures. These statistics were characterized as a function of the ratios of the line-widths to the initial-grain-sizes. Among the important findings is that polygranular cluster and bamboo segment length distributions for as-patterned lines are best fit by Weibull distribution functions. Analytic formulae describing grain structure statistics were reported, for usage in EM simulations and reliability predictions. A differential model predicting the evolution of the polygranular cluster length distribution during post-patterning annealing was developed. It was shown that the rate of bamboo-segment nucleation per unit time and unit of untransformed length is proportional to μ/w^3, and is negligible in the growth-dominated steady-state. The cluster shrinkage velocity was demonstrated to reach a constant steady-state value proportional to μ/w (assuming constant and uniform μ). This was shown to lead to a time-invariant,

steady-state exponential cluster length distribution with an average cluster length proportional to the strip width, and a cluster length fraction decaying exponentially with $\tau=\mu t/w^2$. The distribution of grain lengths in the resulting final bamboo grain structure is well fit by a lognormal distribution, with a median grain length scaling with the line width, and a line-width-independent normalized deviation in the grain length. This result was used to show, using an EM simulation, that grain-orientation-dependent variations in surface diffusivities constitute a likely cause for the variabilities in lifetimes observed experimentally.

The 2D simulation GGSim was also substantially modified to simulate the patterning of interconnect features of general shapes from polygranular thin film structures, as well as to simulate further grain structure evolution due to post-patterning annealing in these complex shapes. A grain structure extraction tool, PolySeg, was developed to allow extraction of the atomic transport details in the case of complex interconnect trees for EM reliability predictions using EM simulations.

To assess the 3D effects on grain structure evolution, and therefore on interconnect reliability, a soap froth experiment was used to study 3D normal grain growth in long rectangular prisms. The kinetics were found to scale with the normalized time $\mu t/w^2$ (with w being the largest of the two prism cross-sectional dimensions). It was found that the normalized duration of the conversion from 3D (non-columnar) to 2D (columnar) structures and the normalized duration of the initial phase during which the structure was polygranular became longer as w/h approached 1. The same results obtained in the 2D case for the scaling behaviors of the bamboo nucleation rate and the polygranular cluster shrinkage rate were demonstrated. Based on a 2D approach, a prism-geometry-sensitive analytic model was developed for the transformation to fully-bamboo structures. These results were compared with preliminary results obtained using a 3D grain growth simulation and qualitative agreement was demonstrated.

We have successfully captured with simple analytic models as well as elaborate simulations the physics of microstructure evolution in complex patterned thin-film structures. In particular, we have developed an array of models and simulations that can be used to investigate the impact of geometry and process history on microstructure evolution, and ultimately on EM-induced failure statistics.

Thesis Supervisor: Carl V. Thompson
Title: Stavros Salapatas Professor of Materials Science and Engineering

To My Parents, Yolla and Rahif, with Love.

Acknowledgments

More than four years pondering this research have come to an end. I have enjoyed every day of them. I feel a sense of achievement that I owe to a number of people.

My thesis supervisor, Professor Carl Thompson, offered me, along with the uninterrupted guidance throughout the years, the freedom that made this period of my life a very happy and fulfilling one.

I am grateful to Professor Jerome Connor, my thesis committee chairman, for his enriching advice, and to Professor Donald Troxel who served on my thesis committee.

I wish to thank Professor Harold Frost for his insights in the research and the open and friendly conversations we had.

My research group was tolerant and helpful. I am grateful to Vab, Brett, Srikar, Yonald, and Robert for their assistance. I would like to mention Steve especially for helping me solve all my computer problems.

My friends supported me constantly, but I owe them more for being my friends. As the proverb says, once you get them, you're stuck with them. The friendships of Jean-Claude Saghbini, Joseph Saleh (the power to listen), Saadeddine Mneimneh, Mauro Kobrinsky, Richard Rabbat, Kamal Hayek, Issam Bazzi, Karim Karam, and Ziad Younes are among the achievements I appreciate most.

I will not list my few old friends, although I still love them as much (once you're in, you're in forever). But I do owe them a lot, especially Bassam, the best friend, best brother, and best man.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Interconnects are the thin metal lines connecting the millions of electronic devices composing a single Integrated Circuit. These lines have various geometries and are obtained by depositing a thin metal film over a dielectric layer and then patterning it. Aluminum, Copper and their alloys are the most widely used metallization materials. Al alloys are easy to deposit and pattern, and they have good adherence to the silicon substrate [Pra 83]. Cu alloys on the other hand have a lower resistivity than Al but are more difficult to etch, susceptible to corrosion and require a diffusion barrier layer to prevent diffusion of Cu through the dielectric and into the silicon devices [Li 93]. Although with different characteristics, when these metal interconnects are subjected to high current densities, they all may fail due to a current-induced diffusive phenomenon called electromigration, first identified by Blech and Sello in 1966 [Ble 66].

Electromigration-induced failure is the primary failure mechanism of interconnects in modern integrated circuits. Design limits imposed due to metal reliability concerns are often very conservative. Worst case assumptions are often made because of the current absence of a sound basis for more accurate assessments. For example, a common assumption is to consider all lines in an IC to be at the maximum allowed current density, even though most are not. In addition, dc current assumptions are made although most lines are subjected to less damaging pulsed dc or bi-directional currents. Most importantly, design practice is still generally based on worst-case assumptions about

feature sizes. Cumulating these worst case assumptions leads to designs with excessive reliability, and, as a cost, reduced performance.

It is now possible to use the newly available feature size extraction tools along with interconnect-geometry dependent design rules to predict the reliability of individual circuit elements, leading to a significantly less conservative estimation of the overall reliability. However, these feature-size dependent design rules are expected to be a strong function of the internal grain structure of the metal, which in turn depends on the deposition process as well as the post-patterning thermal history. Building upon past research, this thesis contributes to the prediction of the microstructure evolution and its impact on electromigration in metal interconnects.

1.1 Electromigration-Induced Failure of Interconnects

High current densities in interconnects cause them to fail due to electromigration. The continuous miniaturization in the IC designs leads to interconnects with smaller cross-sections and therefore subjected to higher current densities, causing electromigration to occur at an accelerated rate and increasing the risk of electromigration-induced failure.

Electromigration refers to current-induced atomic diffusion due to a momentum transfer from the electrons to the atoms in the presence of an electric field. Momentum transfer causes the atoms to move in the direction of the electron flow. The flux of atoms due to the electron wind force can be expressed as the product of the mobility and the net driving force [Hun 61] by

$$J_e = \frac{DC}{kT} Z^* eE = \frac{DC}{kT} Z^* e\rho j , \qquad (1.1)$$

where D is the effective diffusivity, C is the concentration of the migrating atoms, k is Boltzmann's constant, Z^* is the effective charge of the atoms, and e is the fundamental charge. The electric field, E, is the product of the resistivity ρ by the current density j.

Electromigration-induced failure of interconnects occurs at sites of atomic flux divergence. If the incoming flux of atoms is less than the outgoing flux, atoms are depleted from that site, leading to tension which can cause void formation. If the void spans the width of the line, open circuit failure usually results. On the other hand, if the incoming flux of atoms is greater than the outgoing one, the atoms accumulate, leading to compressive stresses, and generating hillocks. Hillocks can cause a fracture in the surrounding passivation layer, leading to metal extrusion and an electrical short circuit to a neighboring conductor. Microstructural features such as grain boundary triple junctions [Ber 69] and local variation in grain sizes [Kin 80], local temperature changes [Heu 78], or the presence of vias connecting different layers are the main causes of flux divergences in Al polycrystalline lines. Electromigration, and the damage it can cause in interconnects, have been the subject of intense experimental and modeling work over the past two decades. Several volumes of the Materials Research Society's Symposium Proceedings are dedicated to this field (Materials Reliability in Microelectronics, Vol. I to VIII). Recent reviews include [Tho 93, Hu 95].

1.2 Electromigration Model

The most widely accepted description of electromigration in encapsulated metal lines is a one-dimensional model, originally formulated by Blech and Herring to describe the steady-state [Ble 76], and extended by Korhonen et al. to include transients [Kor 93], and in alloyed systems behavior [Kor 95]. This analysis was generalized and implemented in an electromigration simulation tool MIT/EmSim by Park et al. [Par 99] as summarized below.

The model assumes the interconnect is a one-dimensional line, embedded in a rigid matrix, and hydrostatically stressed. The gradients in chemical, electrical, and mechanical potentials give rise to forces on the atoms, and in turn to a flux of matter. Equation (1.1) expresses the flux due to the electric field. Following Herring [Her 50], the flux due to gradients in the hydrostatic stress (J_σ) can be expressed as

$$J_\sigma = -\frac{DC}{kT}\Omega\nabla\sigma, \tag{1.2}$$

where Ω is the atomic volume of the diffusive species. If μ is the chemical potential, the flux due to chemical interaction (J_μ) is in general

$$J_\mu = -\frac{DC}{kT}\nabla\mu. \tag{1.3}$$

The total atomic flux (J) is therefore

$$J = \frac{DC}{kT}(Ze\rho j - \Omega\nabla\sigma - \nabla\mu). \tag{1.4}$$

Assuming the number of atoms in the interconnect is conserved at all times, conservation of mass requires the continuity equation

$$\frac{\partial C}{\partial t} = \gamma - \nabla J, \tag{1.5}$$

where γ is the rate of change of the vacancy concentration. Assuming, finally, a vacancy concentration in equilibrium with the stress, the stress increment corresponding to a concentration change can be expressed as

$$d\sigma = -B\frac{dC}{C}, \tag{1.6}$$

where B is an appropriately defined elastic modulus.

Equations (1.4), (1.5), and (1.6) form a complete set for numerical integration to track electromigration-induced stress evolution. Based on different failure criteria definition [Par 99], times to failure information can then be predicted. The kinetics of stress evolution is directly affected by the details of the transport mechanisms, which in turn depend strongly on the microstructure.

1.3 Effects of Microstructure

Electromigration-induced stress evolution is a function of the spatial dependence of the atomic diffusivity resulting from the interconnect grain structure. In general, diffusion can occur through the metal lattice, down dislocation cores, along grain boundaries, and along the interfaces separating the interconnect from the matrix. The effective, one-dimensional, diffusivity scalar (D) can, in general, be expressed as

$$D = D_L + A_d \rho_d D_d + \frac{D_{gb} \delta \alpha_{gb}}{d_{50}} + \sum_i \frac{D_i \delta \alpha_i}{w_i}, \tag{1.7}$$

where D_L, D_d, D_i, and D_{gb} are the diffusivities through the lattice, dislocation cores, any possible interface, and grain boundaries, respectively. A_d is the cross-sectional area of the dislocation core and ρ_d is the dislocation density oriented along the line length. The w_i's refer to the width, or thickness (depending on the interface considered), assuming a rectangular interconnect cross-section. δ is the width of the diffusive path along the interfaces and grain boundaries, and α is the segregation coefficient. d_{50} is the median grain size of the thin film the interconnect was etched from. Table 1.1 summarizes the numeric values characterizing the different diffusion mechanisms in the case of Al and Cu [Fro 82]. The table supports that diffusion in Cu is slower than in Al which in turn explains why Cu-based interconnects are more resistant to electromigration-induced failure than Al-based interconnects. The table also indicates, in particular, that the

difference in effective diffusivity between two different mechanisms such as grain boundary diffusion and surface diffusion, for example, can be of several orders of magnitude. This, as we will show, means that microstructure-induced variations in the possible diffusion paths and the magnitudes of diffusivities will have a great impact on the electromigration-induced stress evolution and therefore on interconnect reliability.

Table 1.1: Diffusion characteristics in Cu and Al. The values reported have been collected by Frost et al. in [Fro 82] and by Neumann et al. in [Neu 72].

Material	*Copper*	*Aluminum*
Lattice Diffusion:		
$D_L = D_{L0} \exp(-Q_L/kT)$ with		
Pre-exponential, D_{L0} (m^2/s)	2.0×10^{-5}	1.7×10^{-4}
Activation Energy, Q_L (eV)	2.04	1.47
Grain Boundary Diffusion:		
$\delta D_{gb} = \delta D_{gb0} \exp(-Q_{gb}/kT)$ with		
Pre-exponential, δD_{gb0} (m^3/s)	5.0×10^{-15}	5.0×10^{-14}
Activation Energy, Q_{gb} (eV)	1.08	0.87
Core Diffusion:		
$A_d D_d = A_d D_{d0} \exp(-Q_d/kT)$ with		
Pre-exponential, $A_d D_{d0}$ (m^4/s)	1.0×10^{-24}	7.0×10^{-25}
Activation Energy, Q_d (eV)	1.21	0.85
Surface Diffusion:		
$D_S = D_S \exp(-Q_S/kT)$ with		
Pre-exponential, D_S (m^2/s)	2.0	-
Activation Energy, Q_S (eV)	2.12	-

Figure 1.1 shows a planar view of a 1 μm thick Al-alloy polycrystalline thin film deposited on a silicon substrate. In such a structure, individual crystals or grains meet at geometrically well defined grain boundaries. Usually, the metal films have a columnar grain structure, so that all grain boundary planes are perpendicular to the plane of the substrate, as depicted in Fig. 1.2. The simplest interconnect features generated by etching such films are lines with different widths. The grain structure of such lines is very sensitive to the value of the aspect ratio w/d_{50} of the line width w by the median grain size d_{50} of the film from which line is etched, as shown in Fig. 1.3.

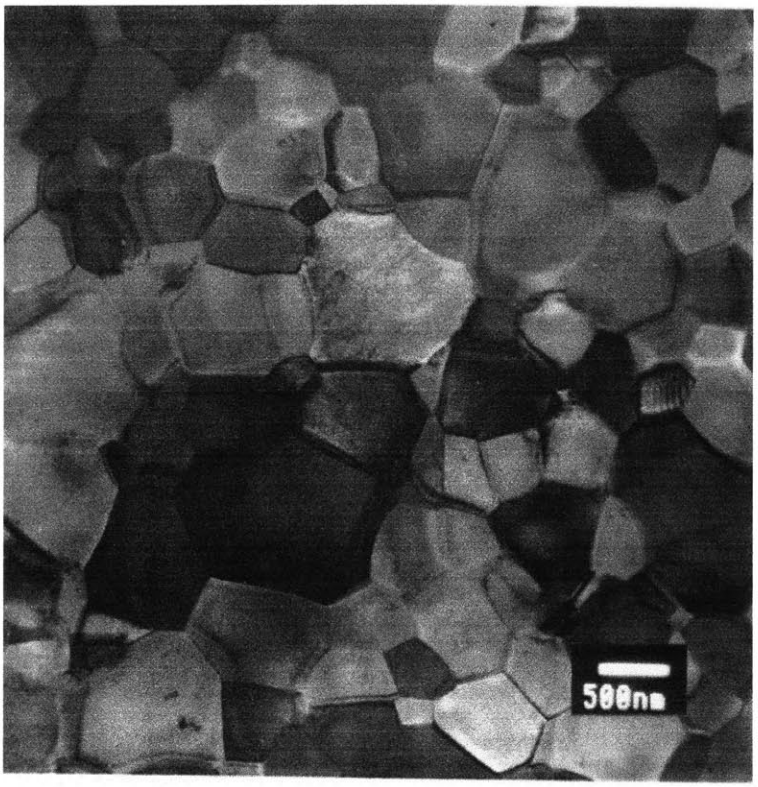


FIG. 1.1: Planar view of a 1 μm thick Al-alloy polycrystalline thin film.

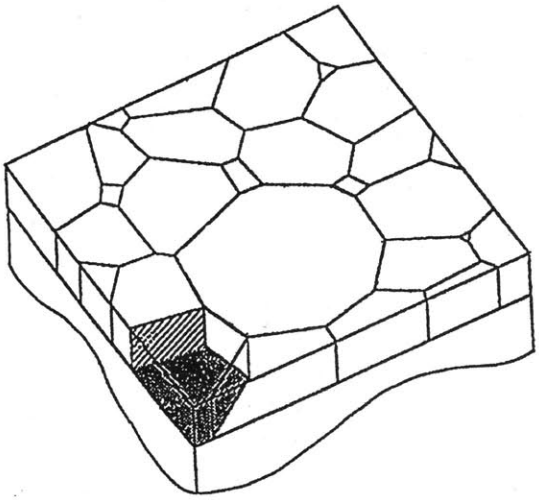


FIG. 1.2: 3D picture of a thin film showing the columnar grain structure of metallization.

When the width of the line is significantly greater than the median grain size, $w/d_{50} >> 1$, the line displays a continuous network of grain boundaries. As the width of the line decreases and approaches the median grain size, more grains span the width of the line, leading to a near-bamboo structure, usually observed when $w/d_{50} \leq 1$. In such a structure, we distinguish the *bamboo segments*, formed by one or more neighboring grains that span the width of the line, and the *polygranular clusters* which are segments with a continuous grain boundary path along the length of the line. As w/d_{50} decreases, the average polygranular cluster length decreases, approaching a fully bamboo structure when $w/d_{50}<<1$, with few, generally short, polygranular clusters in the line.

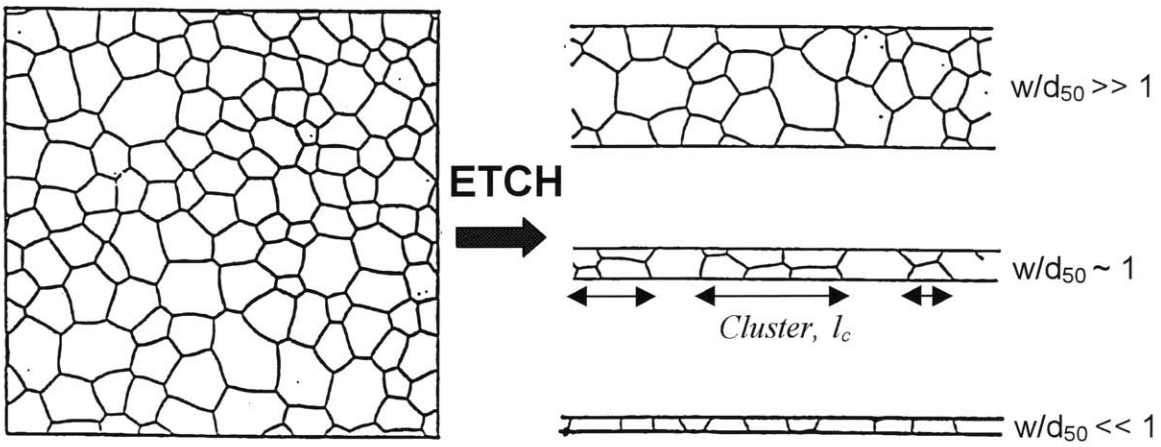


FIG. 1.3: Grain structure of interconnects of different width (w) etched from a continuous film with a median grain size d_{50}.

Electromigration-induced failure of interconnect lines, which usually have a near-bamboo grain structure, is coupled to the line's microstructure through the difference in diffusivity in polygranular clusters and in bamboo segments. Diffusion alone does not cause failure. It is the atomic flux divergence at certain sites that causes the lines to fail. Because the diffusion along the Aluminum-oxide interface is slower than along the grain boundaries, atomic fluxes inside polygranular clusters are initially larger than those in bamboo segments. Therefore, locations where polygranular clusters and bamboo segments meet are sites of a large flux divergence [Kor_2 93, Kin 80]. Atoms deplete at

the upwind end of a polygranular clusters and accumulate at the downwind ends giving rise to tensile and compressive stresses, respectively. The resulting stress gradients, in turn, generate back-stress forces which oppose the electron-wind force. The stress in a polygranular cluster reaches a steady state stress when the two opposing forces are balanced. This is depicted in Fig. 1.4 showing the stress evolution due to electromigration in a portion of a near-bamboo interconnect with two polygranular clusters [Kno 95, Kno 97]. With time, the stresses continue to rise within the cluster until a quasi-steady state condition is attained where the cluster stresses do not change appreciably, and the maximal stress within each cluster is proportional to the cluster length. At very long times, stress variations occur throughout the line, and stress coupling between clusters causes the maximum line stress to increase until a final steady state is reached. If failure is reached whenever the maximum stress in the line reaches a critical, or failure, stress, it becomes clear from the preceding developments that the failure times, and the failure times statistics, in interconnects depend crucially not only on the polygranular cluster length distribution in the lines, but also on the length distribution of the spacings between clusters, or equivalently, the bamboo segments. In turn, these distributions depend strongly on the grain structure and the geometry through the value of the aspect ratio w/d_{50}.

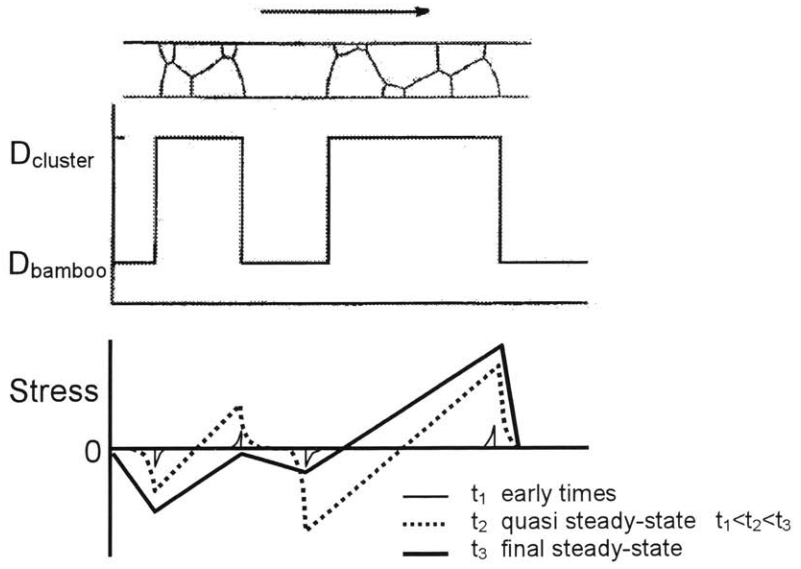


FIG. 1.4: Electromigration-induced stress evolution in a portion of a near-bamboo interconnect with two polygranular clusters [Kno 95, Kno 97]. The electron current direction, or equivalently the direction of flow of matter is indicated by an arrow.

Microstructure therefore affects interconnect reliability through the interplay between grain size (d_{50}) and line geometry (w). The interconnect microstructure, and hence interconnect reliability, depends critically on the interconnect thermal *processing* history [Cho 89]. When annealed, the interconnect with an initially polygranular structure may evolve towards a near-bamboo structure and eventually a fully bamboo structure, through the process of grain growth driven by the reduction of grain boundary energy [Wal 91, Wal_2 91]. Because interconnect reliability is a strong function of the nature of the microstructure, quantitative prediction of the effect of thermal processing on microstructure, and specifically, on polygranular cluster and bamboo segment length distributions, is key to an accurate prediction of the effects of processing on interconnect reliability.

1.4 Design Rules and Reliability Prediction

Failure of interconnects strongly depends, as we have seen, on the microstructural characteristics of the lines. Therefore, interconnect lifetime prediction, or interconnect reliability, constitutes a statistical problem. Solving this problem is required for the design of integrated circuits. It can be done experimentally, or based on the development of analytic models, or based on simulations. The most common experimental procedure in evaluating electromigration-limited reliability is the making of lifetime measurements obtained from tests carried out under accelerated conditions (high temperature, e.g. 200-250° C, and high current density, e.g. 10^6 A/cm^2) [Bla 67]. A population of identical lines is tested under the same conditions in order to capture the lifetime statistics. The commonly agreed on results are that failure time statistics of interconnects obey a lognormal distribution characterized by a median time to failure (MTTF) t_{50} and a deviation in time to failure (DTTF) σ_f. The major problem in interconnect reliability prediction is knowing how to correctly scale the experimentally observed values of t_{50} and σ_f from test conditions to service conditions.

The conventional scaling methodology is based on the following empirical relationship proposed by Black [Bla 67], and relating t_{50} to the temperature and current density:

$$t_{50} = Aj^{-n} \exp(\frac{E_a}{kT}), \tag{1.8}$$

where A is a constant related to the geometry and microstructure of the lines, j is the current density, n is the current density exponent generally accepted to be equal to 2, E_a is the apparent activation energy of failure, the value of which is determined experimentally and depends on the dominant diffusion mechanism (0.4-0.8 eV for polygranular Al-alloy interconnects), and T is the temperature.

This conventional methodology for electromigration-induced failure characterization poses many problems. Although Black's equation predicts a current density dependence for the median time to failure, the current density dependence of the deviation in the failure times is not predicted, and more importantly the relationship between the lifetime components, t_{50} and σ_f, and the lines geometrical characteristics (length and width for flat straight lines) is not determined. In addition, the effects of microstructure variations resulting from processing variations are also not accounted for.

We propose to develop experiments and experimentally verified simulations of grain structure evolution, in order to build analytic models which can be used in electromigration simulations to predict the dramatic effects of microstructure characteristics and feature sizes, as well as the effect of processing, on interconnect reliability. We have developed a powerful design tool which can be used with improved IC feature extraction tools to calculate and apply process-sensitive reliability rules to reliability assessments of full or partial IC designs. Figure 1.5 is a chart of the different components of the interconnect reliability prediction tool. The reliability estimation is based on three computational tools: ERNI, GGSim, and MIT/EmSim [Demo 98]. ERNI, which is being developed by Chery et al., extracts the interconnects from the circuit

layout, and bins them according to geometry, boundary conditions, and current information. The interconnects are present not only as straight lines but also in more complex shapes, such as L shapes, T shapes, or more generally, complex tree geometries. GGSim simulates grain structure evolution in such patterns as a function of materials, processing conditions, and geometry. GGSim is a grain growth simulation tool [Fro 88], which we have extended to simulate the patterning of general-shaped interconnects as well as further post-patterning annealing-induced evolution. PolySeg is the program, written in the context of this thesis, that maps the grain structure generated in GGSim into a diffusivity profile suitable for input to electromigration simulations. Alternatively, in the case of interconnect straight lines, the task of generating process and geometry-sensitive realistic microstructural input to EM simulations performed by GGSim and PolySeg can be by-passed, using EmSimGen, an analytic model-based interconnect generation tool developed also in the context of this thesis. Finally, the stress evolution associated with electromigration, and hence the failure times, can be predicted using MIT/EmSim [Par 99]. This approach leads to the fabrication of integrated circuits with accurately predicted reliability and better performance.

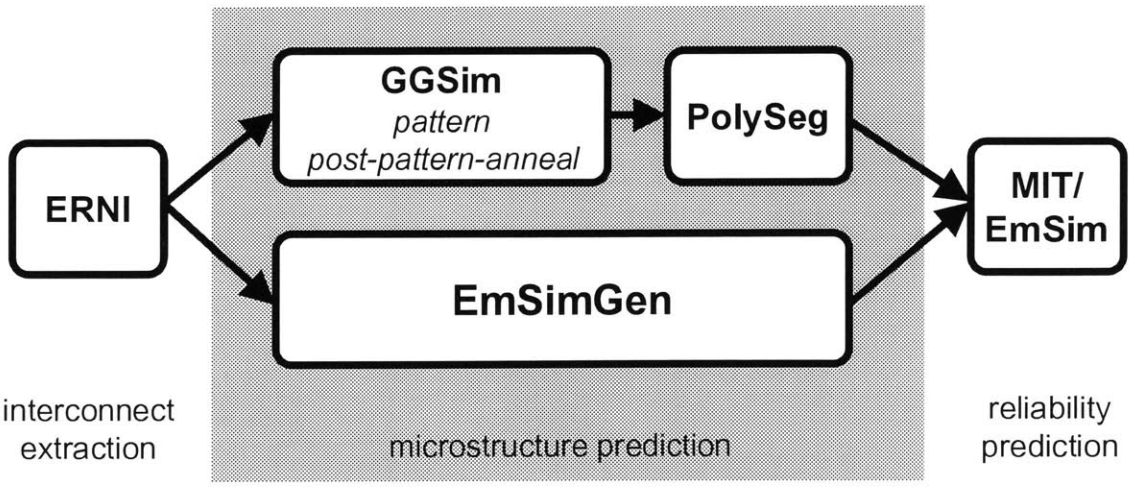


FIG. 1.5: Electromigration-limited reliability prediction tool. Pattern, Post-Pattern Anneal, and PolySeg are additional programs brought into the grain growth simulation tool GGSim in this thesis. EmSimGen is the interconnect generation tool based on analytic models developed in this thesis as well.

1.5 Thesis Organization

A detailed understanding of the microstructures of interconnects and their process-sensitive evolution is essential to accurately assess electromigration-induced stress evolution, and hence the reliability, of interconnects. It is the goal of this thesis to quantitatively predict microstructure evolution and its effects on electromigration through the development of analytic models as well as new simulation and experimental techniques for grain structure and grain growth effects in both thin films and interconnect structures etched from such films.

In Chapter 2, we present the grain structure characteristics of a thin film undergoing normal, idealized, two dimensional grain growth. Specifically, we show the type of grain size distributions resulting from purely capillarity-driven growth.

When other driving forces are incorporated, such as three dimensional grooving effects [Mul 58, Fro 90] or solute drag effects [Fro 94], grain growth yields stagnant structures with lognormal grain size distributions, as observed experimentally [Pal 87, Tra 88]. Such films are then etched to make interconnect structures. In Chapter 3, we present an analytic model describing the grain structure in interconnect strips generated in this manner using an existent grain growth simulation tool GGSim [Wal 91]. We also show how this model is used with an electromigration simulation tool MIT/EmSim [Par 99] to predict lifetime information.

Post-patterning annealing of interconnect strips leads to further growth and grain structure evolution, thereby affecting electromigration-limited reliability. Building upon past results [Wal 92], we outline in Chapter 4 of this document, a geometry-dependent analytic model for the post-patterning annealing-induced grain structure evolution in two dimensional thin film strips.

The short Chapter 5 is dedicated to the reliability of bamboo interconnects. It draws on Chapter 4's results concerning bamboo grain structures to present a model for the

texture effects on electromigration reliability, and shows that a variable grain orientation accounts accurately for the variabilities in lifetimes observed experimentally.

Chapter 6 addresses an issue of practical interest. Driven by the fact that real interconnects can have various geometries and are not always linear, we present in the first part of Chapter 6 PATTERN, a simulation tool designed as an extension to GGSim, to handle the etching of interconnects with general shapes [Fay 97], as well as simulations of further evolution induced by a post-patterning annealing. We use this tool to explore end effects in straight line interconnects, as well as grain structure statistics in special geometries. We then develop PolySeg, a GGSim-based interconnect grain structure extraction tool and demonstrate how it can be used, in conjunction with ERNI and MIT/EmSim in lifetime predictions of interconnects with general features. In particular, we simulate the reliability of interconnect trees, and that of wide polygranular interconnect lines. Finally, we present EmSimGen, a process and geometry sensitive interconnect generation tool based on the analytic models of grain structure evolution outlined in chapters 3, 4, and 5.

Chapter 7 presents experimental results obtained for the evolution to bamboo of soap froth structures in rectangular prisms. It addresses the three-dimensional effects to be accounted for in the grain structure models when the aspect ratio (ratio of the line height to the line width) approaches 1. A 3D-geometry-sensitive analytic model is derived for the microstructure evolution and validated with the froth experiments. It is then implemented in EmSimGen for generation of geometry and process sensitive interconnect structures.

Chapter 2

Grain Structure Resulting from Normal 2D Grain Growth

2.1 Introduction

In order to develop a more detailed understanding of grain growth in bulk systems, there has been extensive modeling and simulation work on the simpler problem of grain growth in two dimensions [Abb 92,Yos 95, Sta 93]. The results of this work are often applied to, and found consistent with, experiments on soap froths evolving between parallel glass sheets and with other simple experimental, quasi-2 dimensional systems [Gla 92, Sta 93]. In both experiments [Sta 93, Sti 90, Ber 90] and computer simulations [Sro 84, Fro 88, Nag 90, Gil 96, Mar 96, Fan 97], the initially disordered grain structures evolve, after transients, into a scaling steady state in which the average grain area increases linearly with time, and the distributions of the grain or cell areas (normalized by the average grain area) and the number of sides per grain become time-invariant. In this chapter, we describe this steady state in detail, as characterized using a previously described technique for simulation of grain growth [Fro 88]. We show that the grain size distribution in the scaling state is best fit by a Weibull distribution function instead of the commonly-cited lognormal or Rayleigh distribution functions. We also apply a recent approach [Mul 98] along with our results to model the steady-state average number of sides per grain as a function of the reduced grain size.

2.2 Overview of the Simulation

Our grain growth simulation is based on a front tracking, curvature driven, 2-D model [Fro 88] in which the locations of boundary segments are specified by arrays of points. In our simplest model, the local boundary migration velocity, v, is determined by local curvature, κ, as:

$$v = \mu \kappa, \tag{2.1}$$

where μ is the product of the grain boundary mobility and the grain boundary energy. We here assume that boundary mobility and energy are identical for all boundaries and for all boundary orientations. As can be seen in Fig. 2.1, the boundaries meet at triple junctions that are repositioned every time step in order to ensure 120° contact angles. The simulation handles topological changes such as the switching of grain neighbors (T1 events [Wea 84]) and the annihilation of small grains (T2 events [Wea 84]).

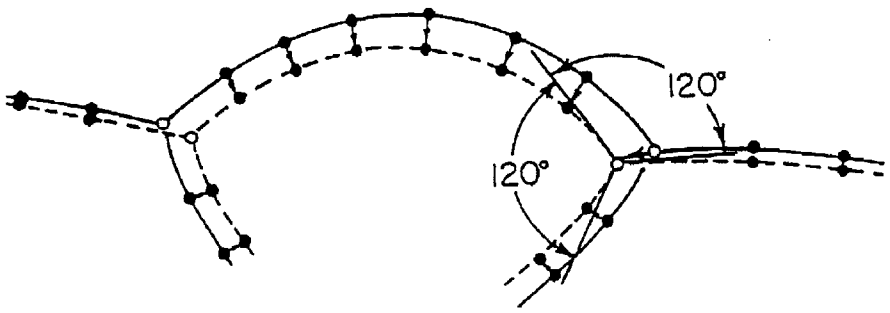


FIG. 2.1: Simulation of 2D normal grain growth. In each time step, points are moved according to $v = \mu\kappa$ and a force balance is imposed at triple junctions. During normal grain growth, in which boundaries have uniform energy and mobility, the angle between intersecting boundaries is 120°.

Figure 2.2 shows part of an initial structure obtained after nucleation and growth to impingement of about 40,000 grains. For this initial structure, the nucleation rate and growth rate were set so as to produce an average grain area of unity for the ideal case of

continuous nucleation at random locations followed by growth towards impingement at a constant, uniform rate (known as the Johnson-Mehl structure [Fro 85]). For numerical convenience, we rejected a small portion of the random nucleation events which were very close to pre-existing grains, so as to avoid very short boundary segments. The initial structures therefore had about 6% fewer grains than expected from the idealized conditions. Figure 2.2 also shows part of the steady state structure obtained at two subsequent times. We are able to simulate large enough initial structures to retain a large number of grains upon reaching the steady state regime, thus allowing the treatment of statistical characteristics with greater accuracy than in previous publications. The 200 x 200 array used in Fig. 2.2 reached the scaling state with about 20,000 grains remaining. The scaling state is characterized by a constant value of 1.52±0.02 for the ratio of the average of the squared grain area to the squared average of the grain area. This is depicted in Fig. 2.3 where we have also plotted the evolution with normalized time ($\tau=\mu t/\langle A\rangle_{t=0}$) of the ratio of the average of the squared grain diameter (area equivalent) to the squared average of the grain diameter. The steady-state value found for the latter ratio is 1.17±0.02.

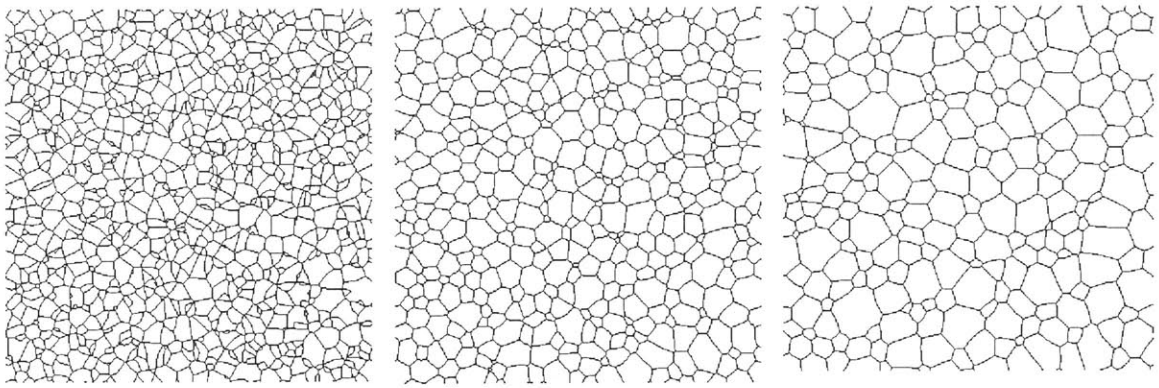


FIG. 2.2: Grain structure at three normalized times, τ, with average grain areas <A>, in units of the average area of the structure resulting from the Johnson-Mehl nucleation and growth to impingement model. The structure at $\tau = 3.0$ is statistically similar to the structure at $\tau = 1.0$. The normalized time τ is related to the real time t by $\tau=\mu t/A_0$ where A_0 is the initial average grain area after growth to impingement.

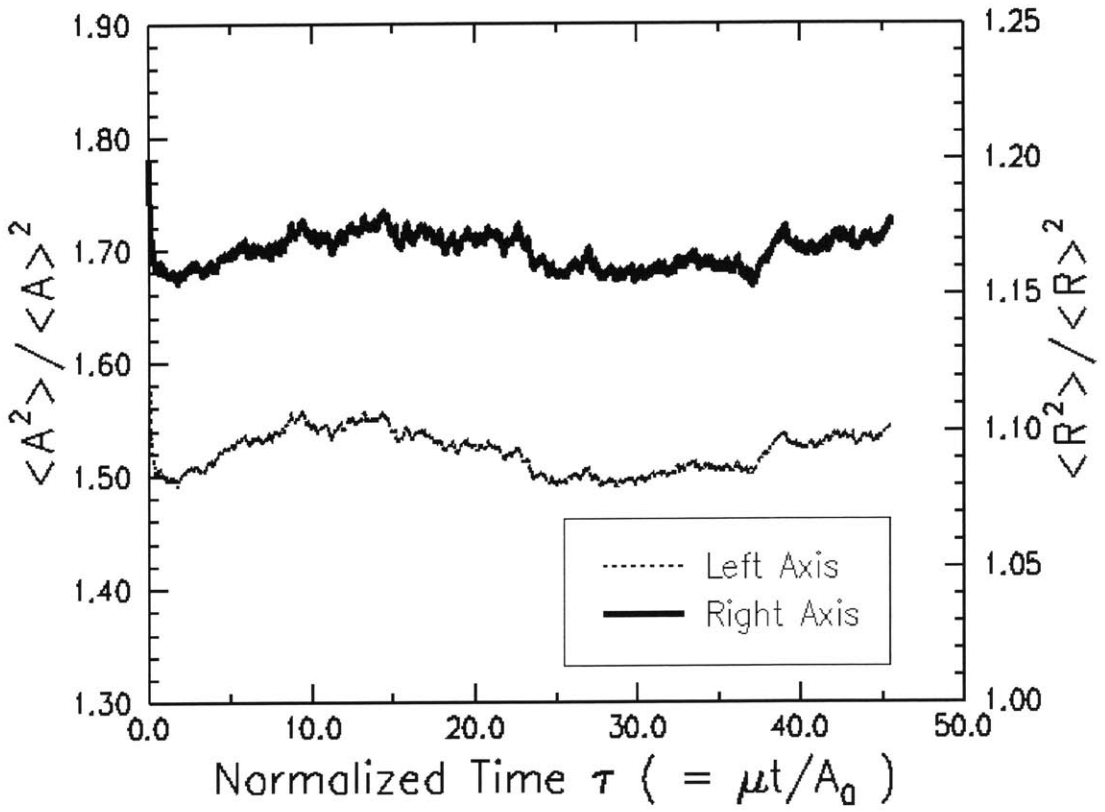


FIG. 2.3: Self-similarity. Plot of the evolution of the ratio of the average of the squared grain area to the squared average of the grain area, $\langle A^2\rangle/\langle A\rangle^2=1.52\pm0.02$ (left axis), and the ratio of the average of the squared grain radius to the squared average of the grain radius, $\langle R^2\rangle/\langle R\rangle^2=1.17\pm0.02$ (right axis).

It has been previously shown [Fro 88] that in the steady state the rate of increase of the average grain area $\langle A\rangle$ is constant. Figure 2.4 depicts the evolution with normalized time of both the average grain area and the square of the average grain radius. It shows that the rate of increase of the average area is the mobility constant:

$$\frac{d\langle A\rangle}{dt} = (1.02 \pm 0.02)\mu \,. \qquad (2.2)$$

We find that the relative error in the evaluation of this kinetic constant is smaller than 2%. It has also been previously shown [Wal 91] that this simulation obeys a relationship similar to the Mullins-von Neumann [Mul 56] growth law but applied to the average behavior of grains within a topological class:

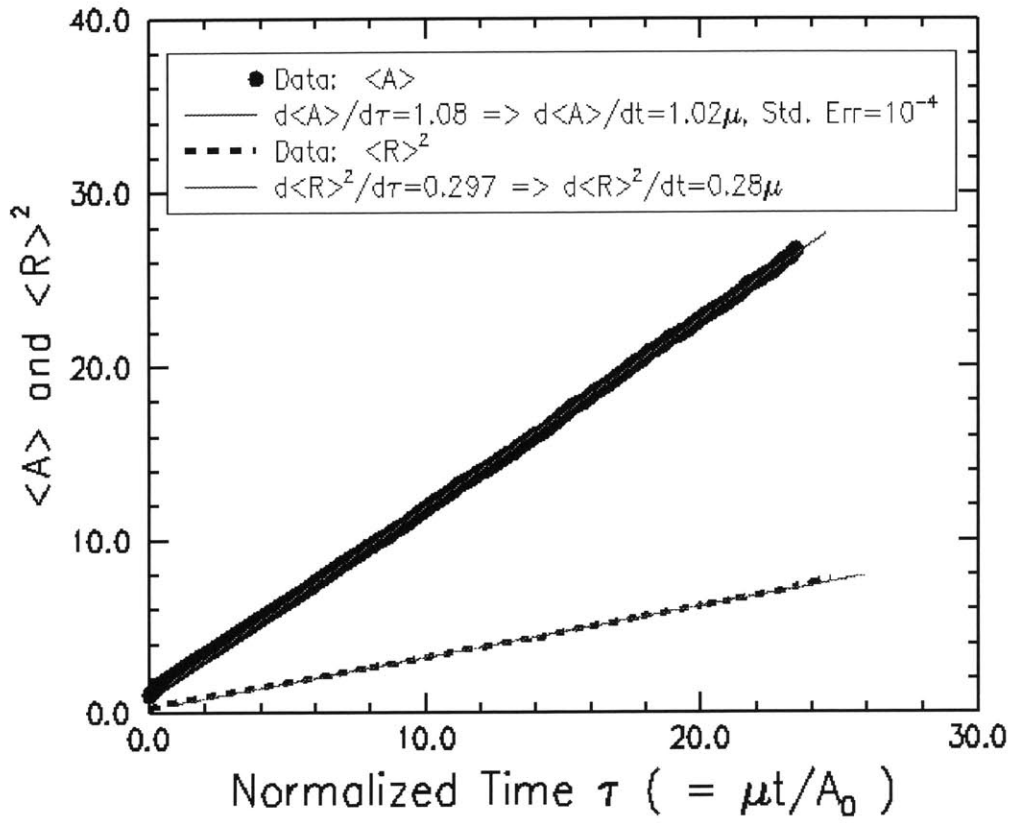


FIG. 2.4: Evolution with normalized time of both the average grain area and the square of the average grain radius.

$$\frac{d < A_n >}{dt} = \mu \frac{\pi}{3}(n-6), \tag{2.3}$$

where $<A_n>$ is the average area of grains with n sides, as has been demonstrated for another simulation [Fan$_2$ 97]. Figure 2.5 further indicates that more than 95% of the n-sided grains with n<11 accurately obey the Mullins-von Neumann law. Under the statistical self similarity hypothesis, Mullins [Mul 86] showed that

$$\frac{1}{\mu}\frac{d < A >}{dt} = \frac{2\pi}{3}\frac{< A >^2}{< A^2 >}\sum_{n=0}^{\infty}\alpha_n(n-6), \tag{2.4}$$

where α_n is the total area fraction of n-sided grains. We find that our simulation conforms to this relationship between the growth kinetics and the structure topology, and

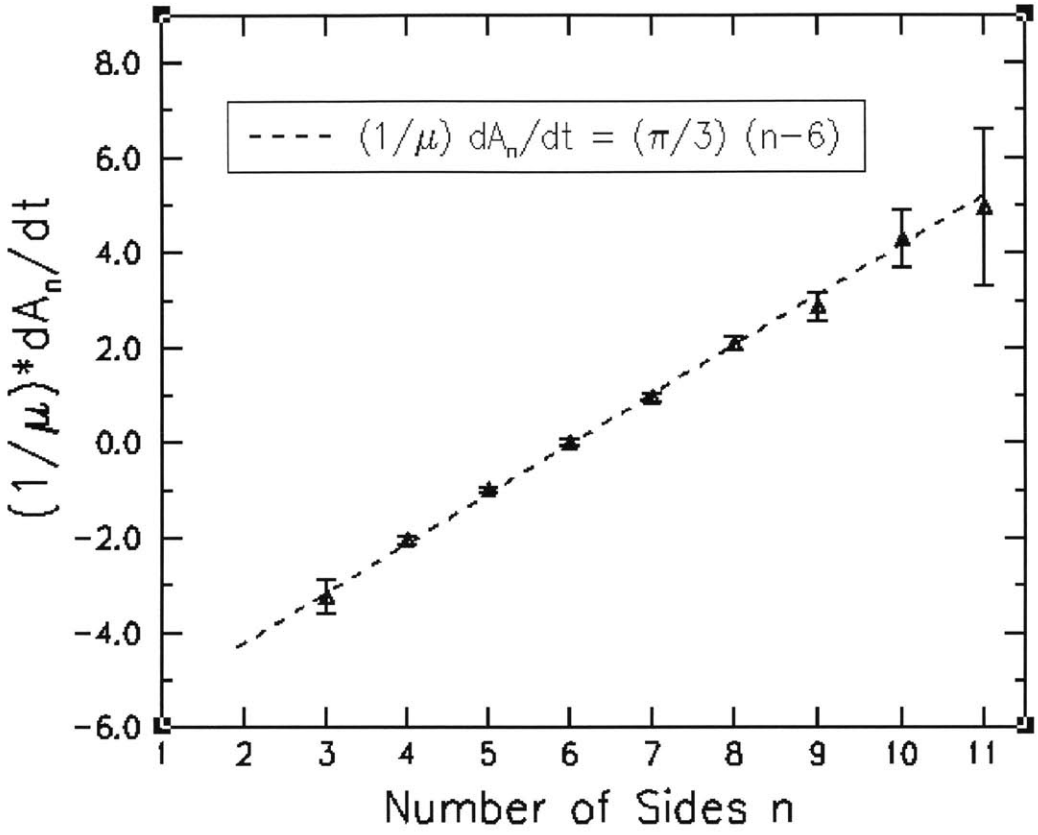


FIG. 2.5: Von Neumann's law, rate of growth of n-sided grains as a function of n. The error bars reflect a 95% confidence interval, assuming a normal distribution for the data.

further indicates that both the left and right hand sides of this relationship are well approximated by one (± 0.05). The variations of the Mullins topological constant M_1 (right hand side in equation (2.4)) are plotted in Fig. 2.6. Similarly, the rate of increase of the square of the average grain radius is related to the steady-state topology by

$$\frac{1}{2\mu}\frac{d<R>^2}{dt} = \frac{1}{3}\frac{<R>^2<R^2>}{<R^4>}\sum_{n=0}^{\infty}\alpha_n(n-6)\,. \qquad (2.5)$$

The kinetic/topological constant M_2 (right/left hand side of equation (2.5)) is found to be equal 0.14 ± 0.01, as seen in Fig. 2.4 and 2.6.

The number-of-sides-per-grain distribution in the steady state is plotted at three different times in Fig. 2.7. We find that the second moment for this distribution is 1.5 ± 0.1. Also, plotted in Fig. 2.7 are experimental results for a 2-D soap froth [Sta 93], for foam-like structures in monolayers of amphiphilic molecules at an air-water interface [Ber 90, Sti 90], and for selected other computer simulations [Sro 84, Gil 96, Fan 97, Nag 92]. These matching plots suggest a universality of the self-similar steady state structure.

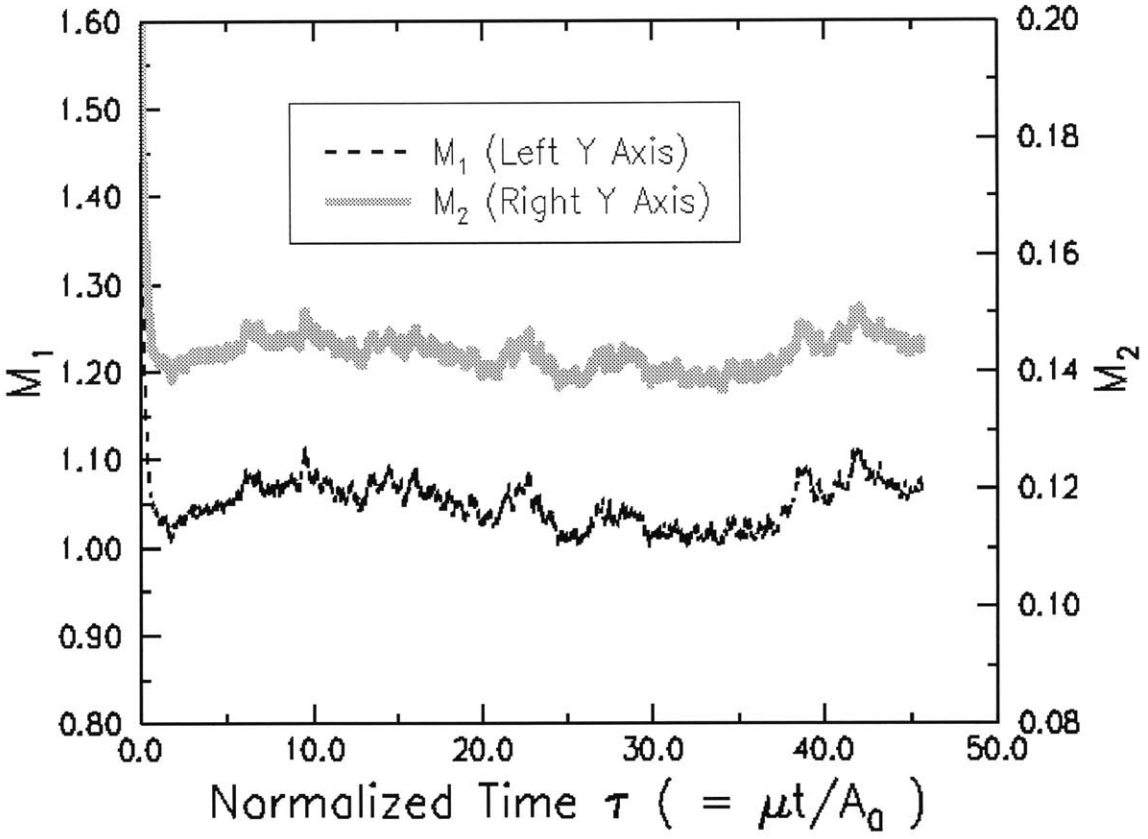


FIG. 2.6: Self-similarity. Evolution of the kinetic/topological constants M_1 defined by equation (2.4) (left axis) and M_2 defined by (2.5) (right axis).

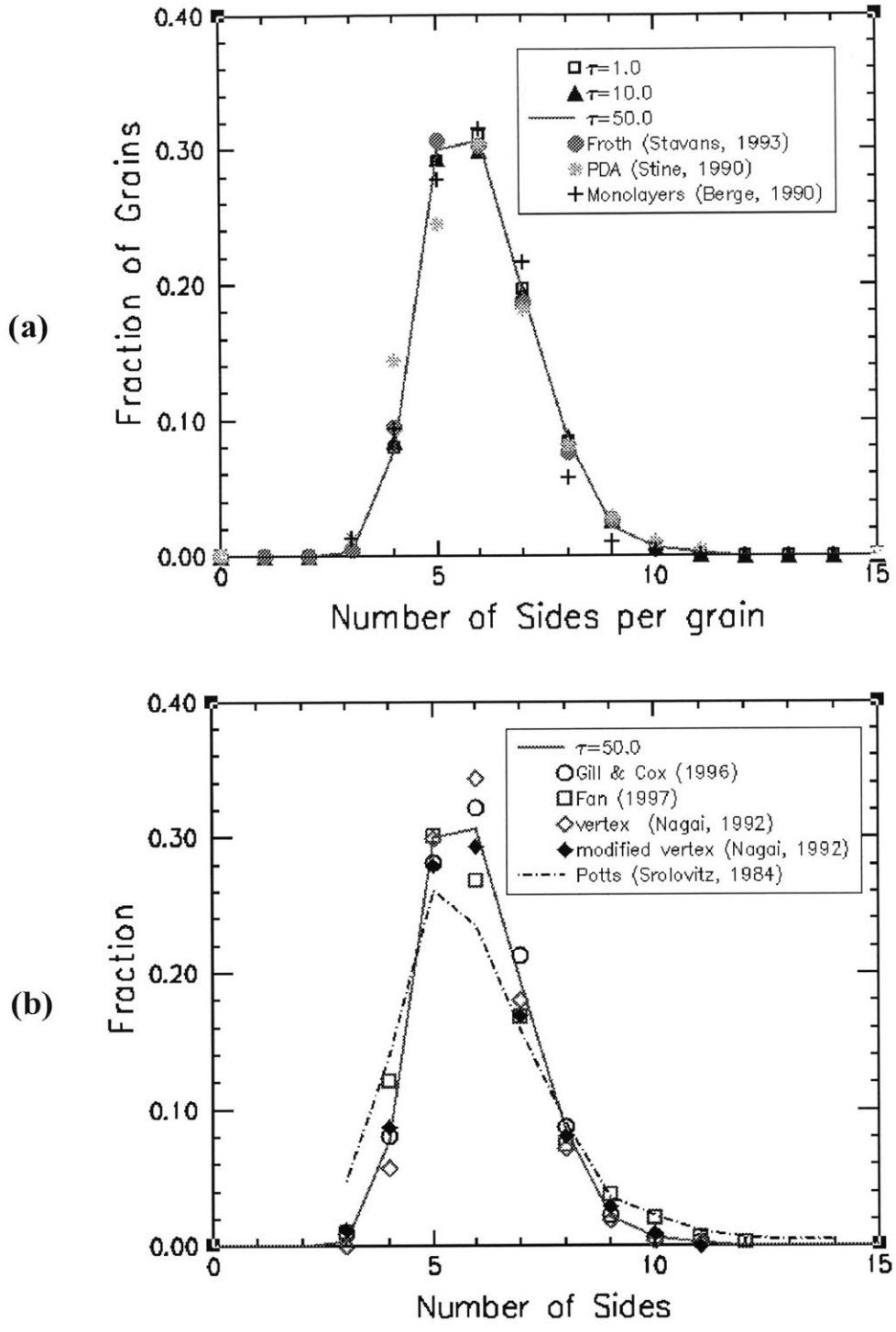


FIG 2.7: The number-of-sides distribution at different normalized times: **(a)** shown with experimental data for a soap froth [Sta 93] and for molecular monolayers [Sti 90, Ber 90], and, **(b)** shown with results from other simulations [Sro 84, Gil 96, Fan_2 97, Nag 92].

2.3 Grain Size Distribution

When we define the grain size as $d = (4A/\pi)^{1/2}$, we find that in the steady state, the distribution of relative grain sizes ($d/\langle d \rangle$) is constant. We find that this steady-state grain size distribution is better fit by a Weibull distribution function than the lognormal, Rayleigh, or Gamma distribution functions previously discussed in the literature [Sta 93, Gla 92, Fra 94]. The Weibull distribution function has two free parameters, $\alpha(t)$ and $\beta(t)$, and may be expressed as the partial distribution function, f(d, t) with:

$$f(d,t) = \frac{\beta}{\alpha^{\beta}} d^{\beta-1} \exp\left(-\left(\frac{d}{\alpha(t)}\right)^{\beta}\right), \tag{2.6}$$

or as the cumulative distribution function, F(d, t) (the fraction of grains with a diameter less than d, at time t) as follows:

$$F(d,t) = 1 - \exp\left(-\left(\frac{d}{\alpha(t)}\right)^{\beta}\right) \tag{2.7}$$

A Weibull plot of the distribution is a plot of ln(d) as a function of ln(-(ln(1-F)) which would give a straight line with a slope $1/\beta$ and an intercept $\ln(\alpha)$ for an exact Weibull distribution function (Fig. 2.8).

The Rayleigh distribution is a special case of the Weibull distribution in which the parameter β is equal to 2. This distribution arises from the analytic model first proposed by Louat [Lou 74] in which the grain-size distribution is presumed to evolve by random size fluctuations of individual grains. Srolovitz et al. [Sro 84] and Fan and Chen [Fan 97] both found that the Rayleigh distribution provided a moderately good fit to their simulation results.

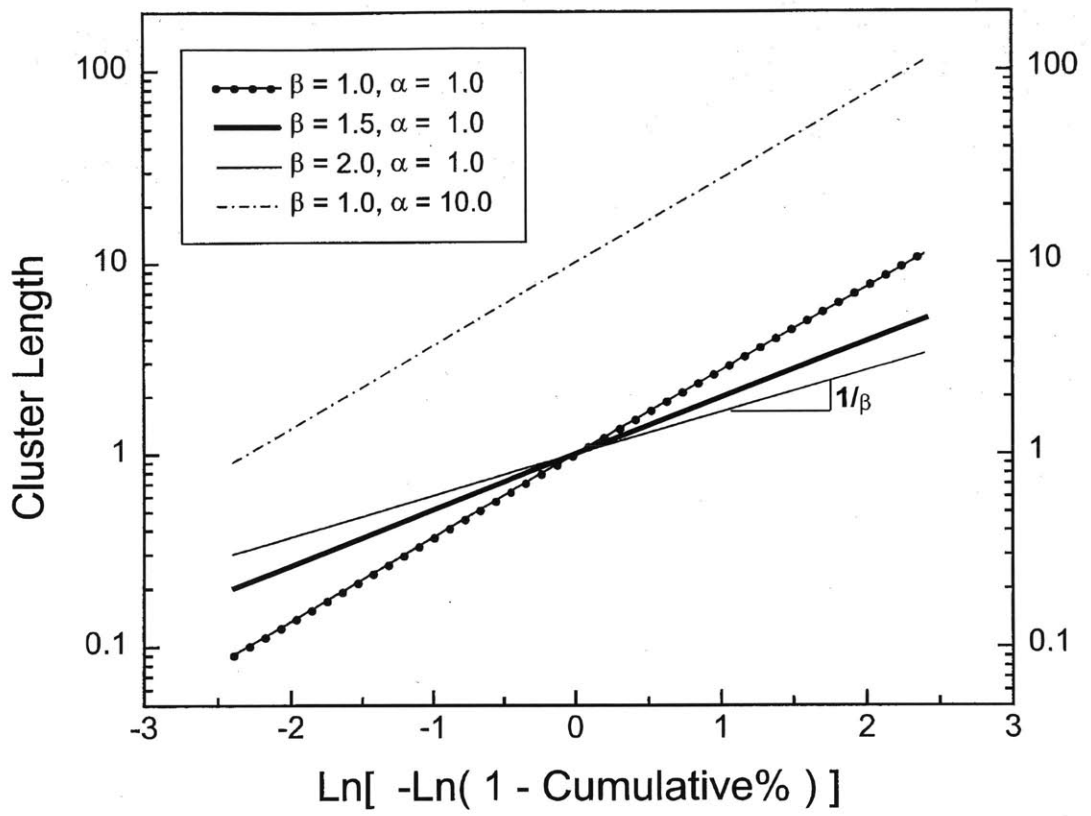


FIG. 2.8: Weibull plot template. A Weibull distribution characterized by (α, β) would plot here as a straight line with a slope $1/\beta$ and an intercept $\ln(\alpha)$.

Another distribution that Srolovitz et al. and Fan et al. used for comparison was the lognormal distribution, which also has two free parameters. The lognormal probability density function, f_{LN}, is given as

$$f_{LN}(d,t) = \frac{1}{\sigma d\sqrt{2\pi}} \exp\left(-\left(\frac{1}{\sqrt{2}\sigma} \ln\left(\frac{d}{\delta} \right) \right)^2 \right), \tag{2.8}$$

where both δ and σ can be taken to be free parameters, though δ is often taken to be the median grain size. Both Srolovitz et al. and Fan and Chen found that their simulations were not well fit by the lognormal distribution, although experiments on thin films generally are found to be well fit by this distribution function [Pal 87, Tra 88, Wu 91]. We have previously attributed the lognormal character of experimental distributions to

other factors not considered in the idealized grain growth simulation, such as boundary pinning due to surface grooving [Fro 90], or the inhibition of boundary migration due to solute drag [Fro 94].

A third distribution function occasionally discussed in the literature [Mar 96, Vaz 88] is the Gamma distribution function which may be given with two parameters as

$$f_{\Gamma}(d,t) = \frac{1}{\Gamma(\beta)\alpha^{\beta}} d^{\beta-1} \exp\left(-\left(\frac{d}{\alpha(t)}\right)\right). \tag{2.9}$$

This function provides a good fit to the diameter distributions for the cellular structure of the Voronoi polygons of random points, which is produced by simultaneous nucleation and growth to impingement from randomly placed nuclei. However it does not provide a good fit for the steady-state grain structure.

Figure 2.9 is a Weibull plot of the grain-diameter distribution at different values of the normalized time τ. The goodness-of-fit to the Weibull distribution is qualitatively demonstrated by plotting, as in Fig. 2.8, ln(d) against ln(-ln(1-F)). The figure shows good fits of the Weibull functions at any time after the steady-state is reached. Furthermore, and consistent with the existence of a steady-state, the slope $1/\beta$ which is a dimensionless measure of the spread in the distribution, seems to remain constant. The intercept, $\ln(\alpha)$ (or α, if the y-axis is logarithmic), increases with time, as expected since it is proportional to the average grain diameter.

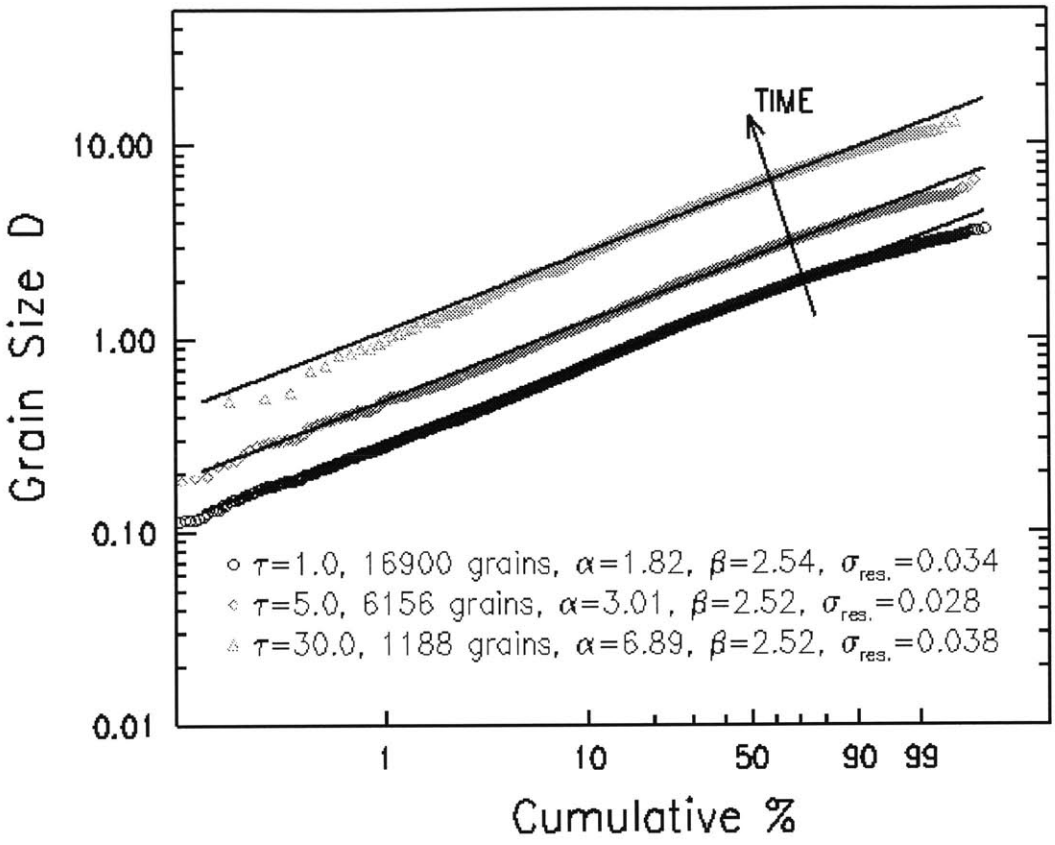


FIG 2.9: Weibull plot of the distribution of the grain-diameters d at different values of the normalized time τ. β, the inverse of the slope, remains constant with time, which is consistent with a steady-state self-similar structure.

Figure 2.10 shows the time dependence of the parameters α and β for the distribution function that best fits the grain-diameter distribution. Within a 4% variation, β remains constant and equal to 5/2. The dependence of α^2 on time is linear, which is the expected result for the observed linear time dependence of the average grain area and the constant value of β. For a Weibull distribution,

$$< d >= \alpha \Gamma\left(1+\frac{1}{\beta}\right), \tag{2.10}$$

and,

$$< d^2 >= \alpha^2 \Gamma\left(1+\frac{2}{\beta}\right), \tag{2.11}$$

where Γ is the gamma function, so that the distribution function for d/<d> is a Weibull function with $\alpha = 1/\Gamma(1+1/\beta)$ and $\beta = 5/2$, and equivalently, since d^2 scales with the area A, A/<A> is a Weibull function with $\alpha_A = 1/\Gamma(1+1/\beta_A)$ and $\beta_A = 5/4$. It is worth noting that given relations (2.2) and (2.11), the slope of the linear dependence of α^2 on time should be $4\mu/[\pi\Gamma(1+2/\beta)] \sim 1.36\mu$, which is verified in Fig. 2.10 with a deviation smaller than 3%.

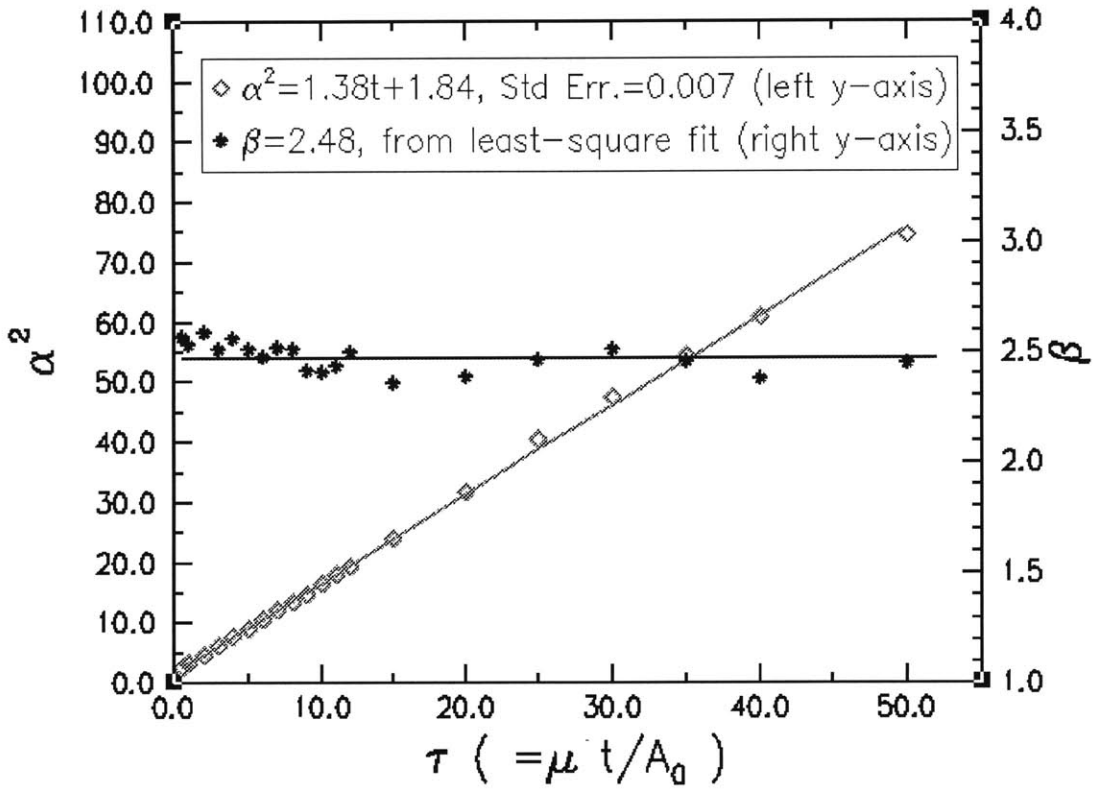


FIG 2.10: Evolution with time of the Weibull parameters for the grain-diameter distribution. Within statistical variations, β remains constant and equal to 2.5, and α^2 increases linearly with time t, with a slope 1.38.

Figure 2.11 shows a Weibull plot for the cumulative normalized grain-diameter d/<d> distribution at different values of the normalized time τ. Figure 2.11-a shows our results along with lines indicating the best-fitting Rayleigh and lognormal functions. Also shown in this figure are distributions from the experiments mentioned earlier, and, in Fig.

2.11-b, distributions from selected simulations. Alternatively, in Fig. 2.12, we plot the normalized grain size distribution on a linear scale.

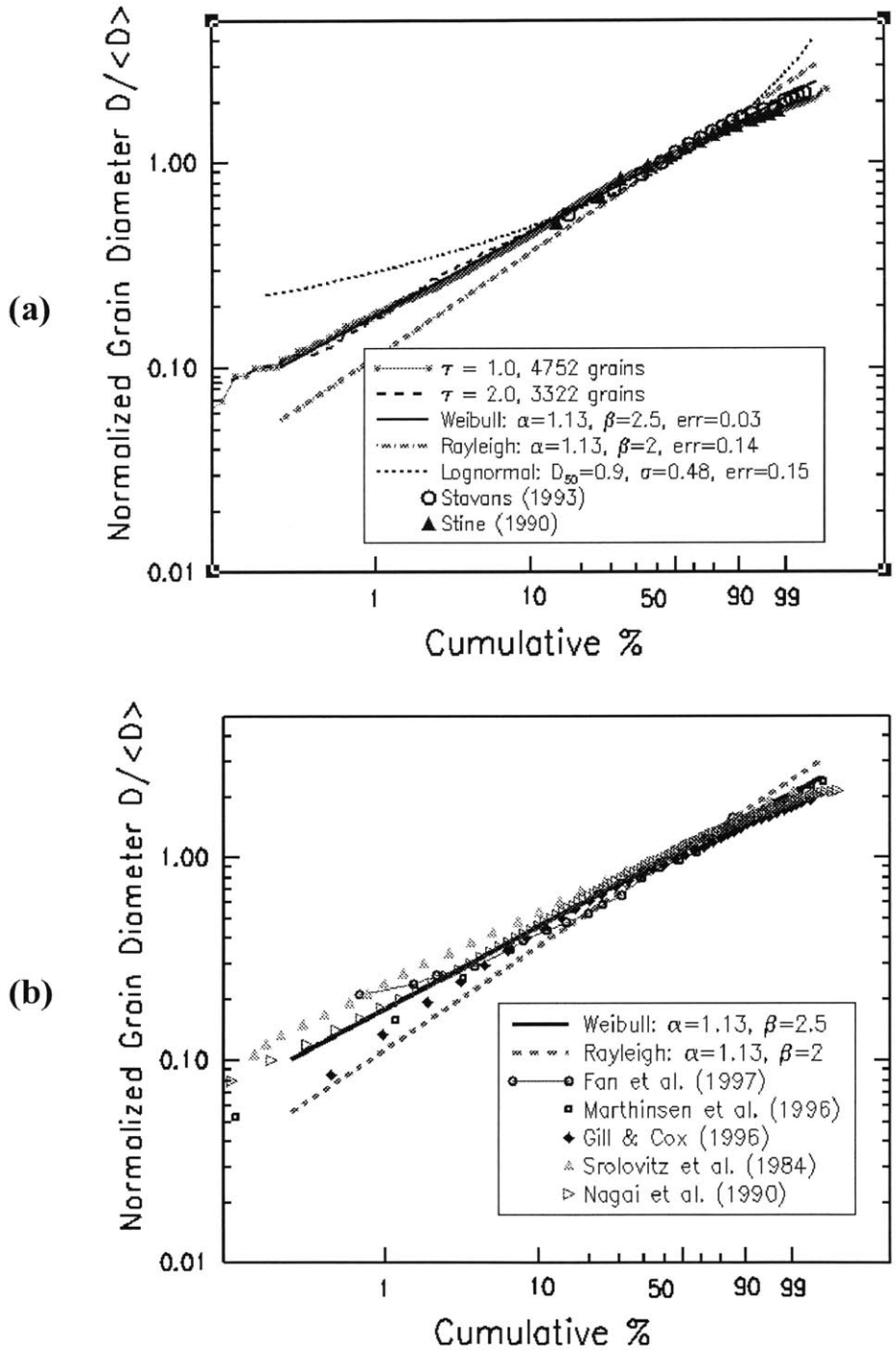
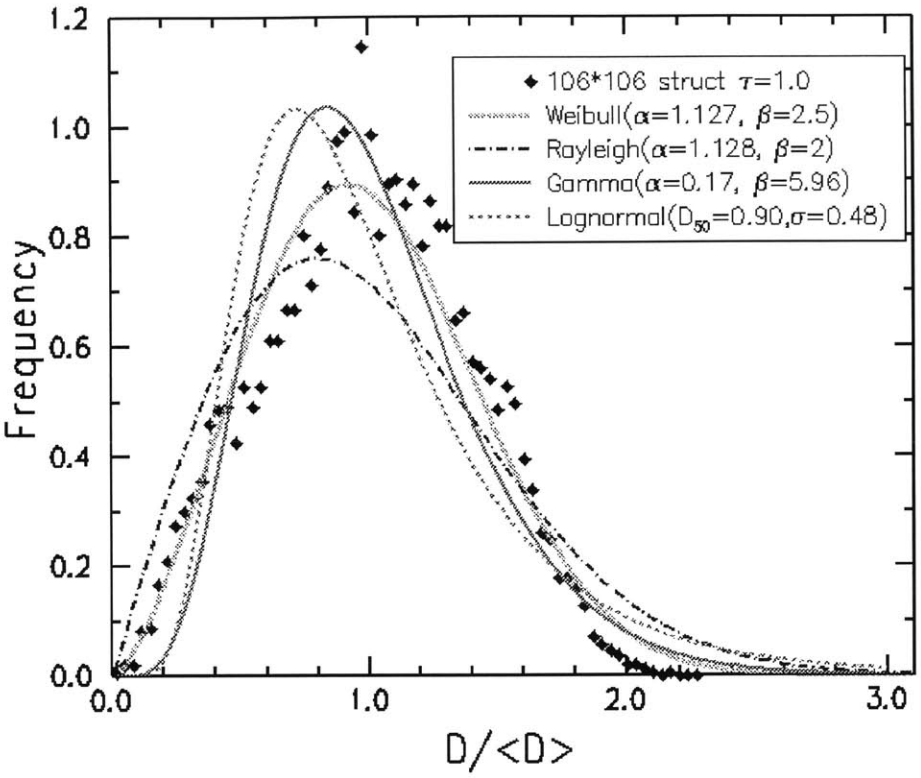


FIG. 2.11: Weibull plots of steady-state normalized grain size distributions: **(a)** for an array of grains of size 106 x 106, at $\tau = 1.0$ and $\tau = 2.0$, shown with plots of best-fitting Rayleigh and lognormal functions and with experimental results for a froth [Sta 93] and a molecular monolayer [Sti 90], and **(b)** for selected simulation results [Sro 84, Nag 90, Gil 96, Mar 96, Fan 97].

(a)



(b)

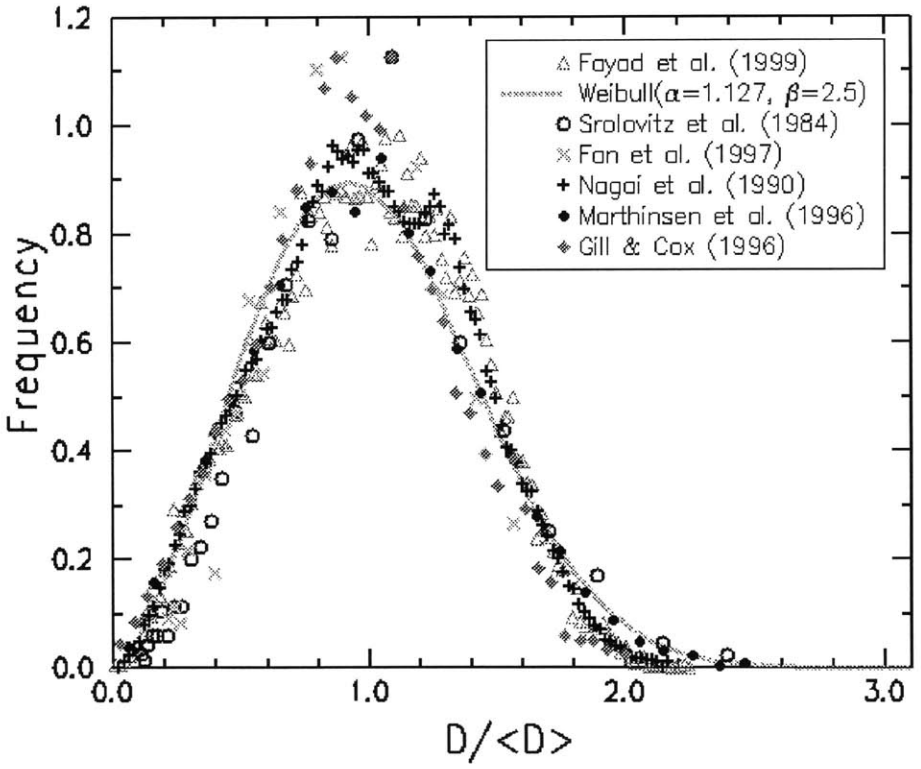


FIG. 2.12: Linear plot of steady-state grain size distribution for an array of grains of size 106x 106, at τ = 1.0, shown **(a)** with plots of best-fitting analytic distribution functions, and **(b)** with selected simulation results [Nag 90, Gil 96, Mar 96, Fan 97].

We have also fit our simulation data to other distribution functions: the lognormal, Rayleigh, and Gamma distribution functions. Figure 2.13 shows a comparative analysis of the various distributions, given by the sums of the residuals squared. When δ is used as an adjustable parameter, the errors in the fits for Lognormal and Rayleigh distribution functions are similar, despite the fact that a lognormal distribution has 2 adjustable parameters, while the Rayleigh distribution function, a special case of the Weibull distribution function with $\beta = 2$, has only one. (If δ is taken to be the median grain size, this fit of the lognormal distribution is worse). The Weibull distribution with $\beta = 2.5$, on the other hand, provides, a consistently better fit, with a deviation in the logarithm of d of about 4%. Similar comparisons were carried between Weibull and Gamma distribution functions and show that the latter, is also inadequate, with an error that is comparable to the ones observed for Rayleigh and lognormal fits. It should be noted that we have reached similar conclusions when fitting these standard distribution functions to the grain area data as well. The Weibull area distribution function, with a deviation in ln(A) of about 5%, provides a better fit than the other distributions mentioned above.

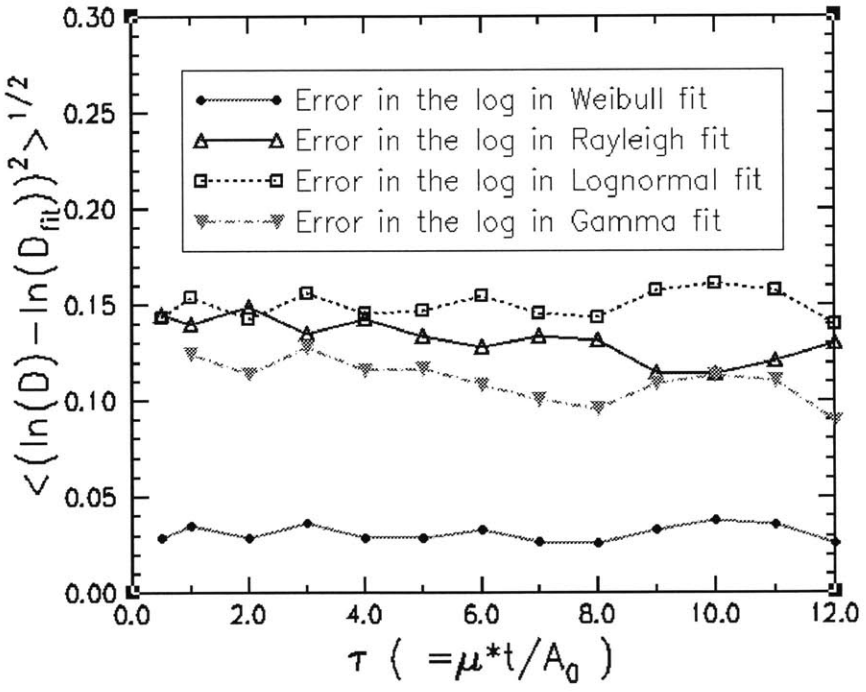


FIG. 2.13: Comparison of the errors in the fit of Weibull, lognormal and Rayleigh distribution functions for the grain size distribution for a simulation with a 200 x 200 starting array (at $\tau = 0$), after a steady state was achieved.

2.4 Correlation between Grain Size and Topology

Figure 2.14 shows, during the steady-state regime, the number of sides per grain of a given size d (within an infinitezimal variation), $N_{|d}$, as a function of the reduced grain size (d/<d>). Recently, Mullins [Mul 98] extended an analysis first developed by Hunderi and Ryum [Hun 80, Viñ 98] to establish the following relation between the reduced grain size distribution and the average number of sides for a given size, $<N_{|d}>$:

$$< N_{|d} >= 6 + 6M_2 x\left(x - \frac{2}{f(x)} \int_x^{\infty} f(x')dx' \right), \qquad (2.12)$$

where x = d/<d> and f(x) is the distribution function of the normalized grain size x. The kinetic parameter M_2 is the one defined by equation (2.5). Using, alternatively, the Rayleigh distribution function, and the Weibull distribution function, equation (2.12) provides a simple analytic functional dependence of $<N_{|d}>$ on the reduced grain size x. Figure 2.15 shows a comparison of the simulation and the equations resulting from the models. In the range of values of x for which f is appreciable (greater than a few percent), the Weibull-distribution-derived curve matches the simulation data well, and matches it better than the corresponding Rayleigh curve.

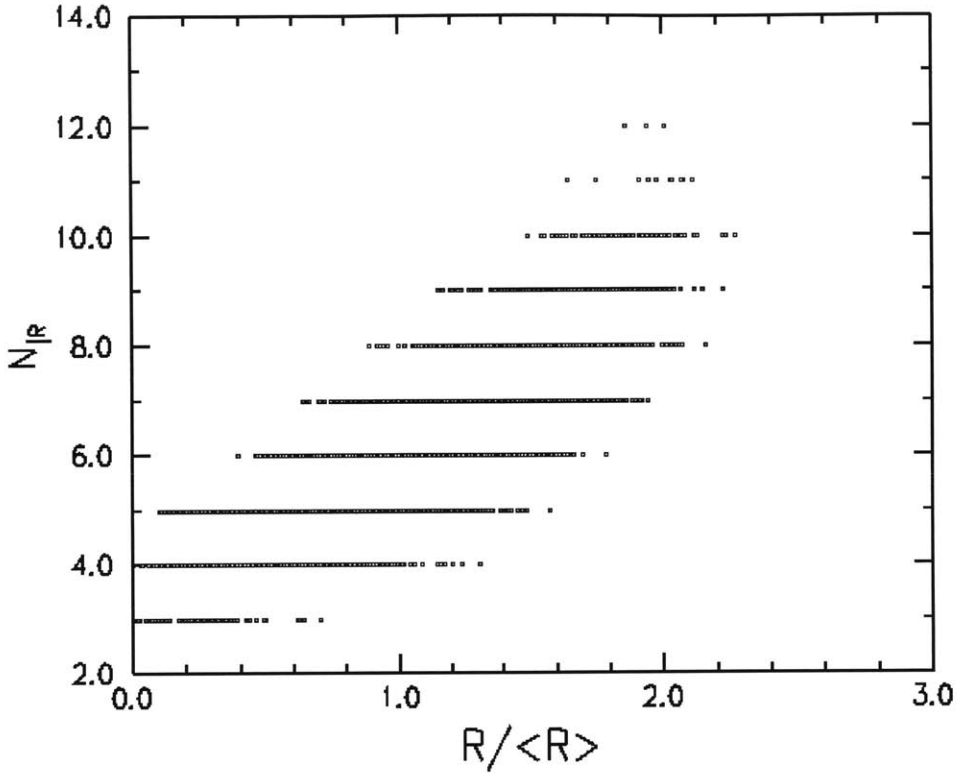


FIG 2.14: The number of sides per grain against the reduced grain size in the steady-state regime.

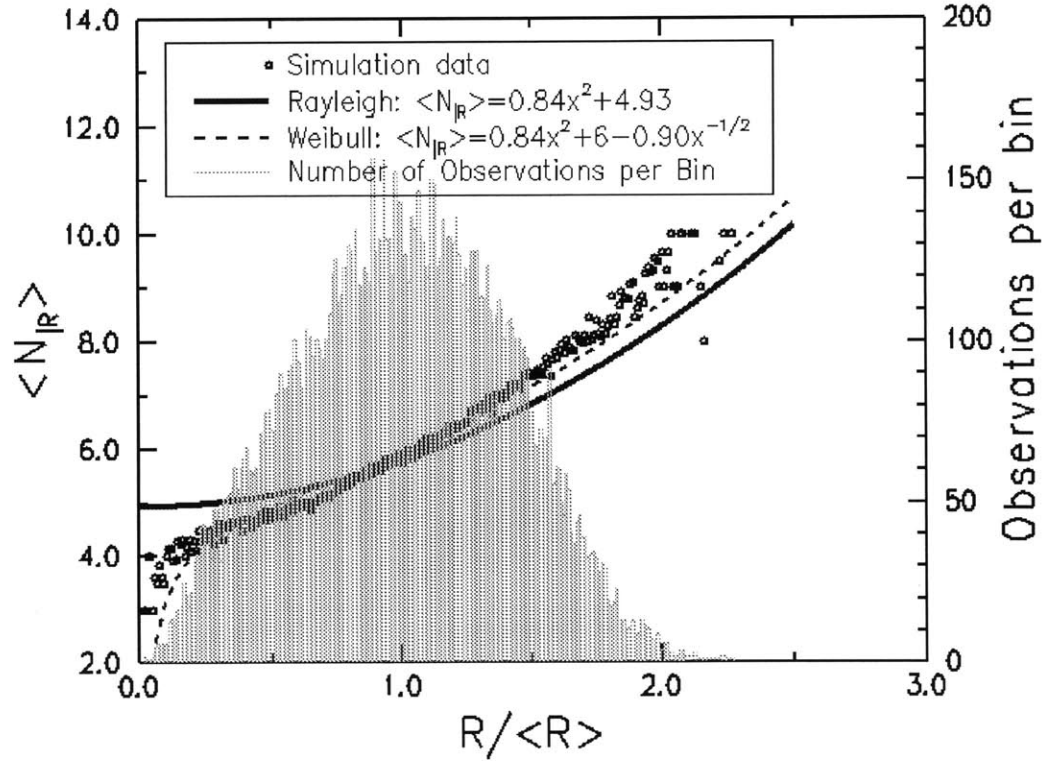


FIG 2.15: Dependence of the average number of sides per grain of a given size on the reduced grain size.

2.5 Conclusions

In summary, we have further characterized results obtained using a simulation of 2D grain growth, carrying out larger simulations leading to better defined statistical characteristics. We have confirmed that a scaling state is reached in which the average grain area increases linearly with time, and the normalized-grain-area and the number-of-sides-per-grain distributions are time invariant. We have further demonstrated that the constant rate of change of the average grain area is equal to the grain boundary mobility constant. We have also found that the steady state grain size distribution we find using our simulation technique, as well as those reported in experiments on simple model systems as well as for very different simulation techniques, are all very well fit by a Weibull function with the parameter $\beta = 5/2$, and are better fit by this function than the lognormal, Gamma or Rayleigh functions. These results suggest that the true scaling state for simple 2D grain growth is revealed in these experiments and simulations, and provide new simple functional description for the steady state distribution of grain sizes.

Chapter 3

As-Patterned Interconnect Grain Structure Model

3.1 Introduction

Electromigration-induced failure times and failure-time statistics in metal interconnects are governed by their grain structure and their grain structure statistics. As discussed in Chapter 1, interconnect lines with near-bamboo structures have a mixture of polygranular clusters separated by one or more grains that span the width of the line, which we refer to as bamboo segments. In these interconnects, electromigration-induced diffusion occurs primarily along grain boundaries in the polygranular clusters, while it is expected that in the bamboo segments, diffusion occurs primarily along the Al-oxide interface [Joo 94]. Because grain boundary diffusivities are orders of magnitude higher than diffusivities in bamboo segments, electromigration-induced failure time statistics will strongly depend on the length distributions of polygranular clusters and bamboo segments. As the ratio of the line width w to the median grain size D_{50} of the initial metal film decreases, polygranular clusters become shorter and an increase in the median time to electromigration-induced failure is observed, along with an increase in the deviation in the times to failure [Cho 89], as shown in Fig. 3.1.

Our goal is to predict the effects of interconnect grain structure on the statistics of electromigration-induced failure. This ability will allow optimization of manufacturing processes to maximize interconnect reliability. Our approach is to use our grain growth

simulator to construct an analytic model for interconnect grain structures. The predicted spatial dependence of the type of diffusion path, and the corresponding spatial variation in the diffusivity, can then be used with the electromigration simulator MIT/EmSim [Par 99] to predict lifetime information. This gives us a tool to investigate the relationship between grain structure and interconnect reliability.

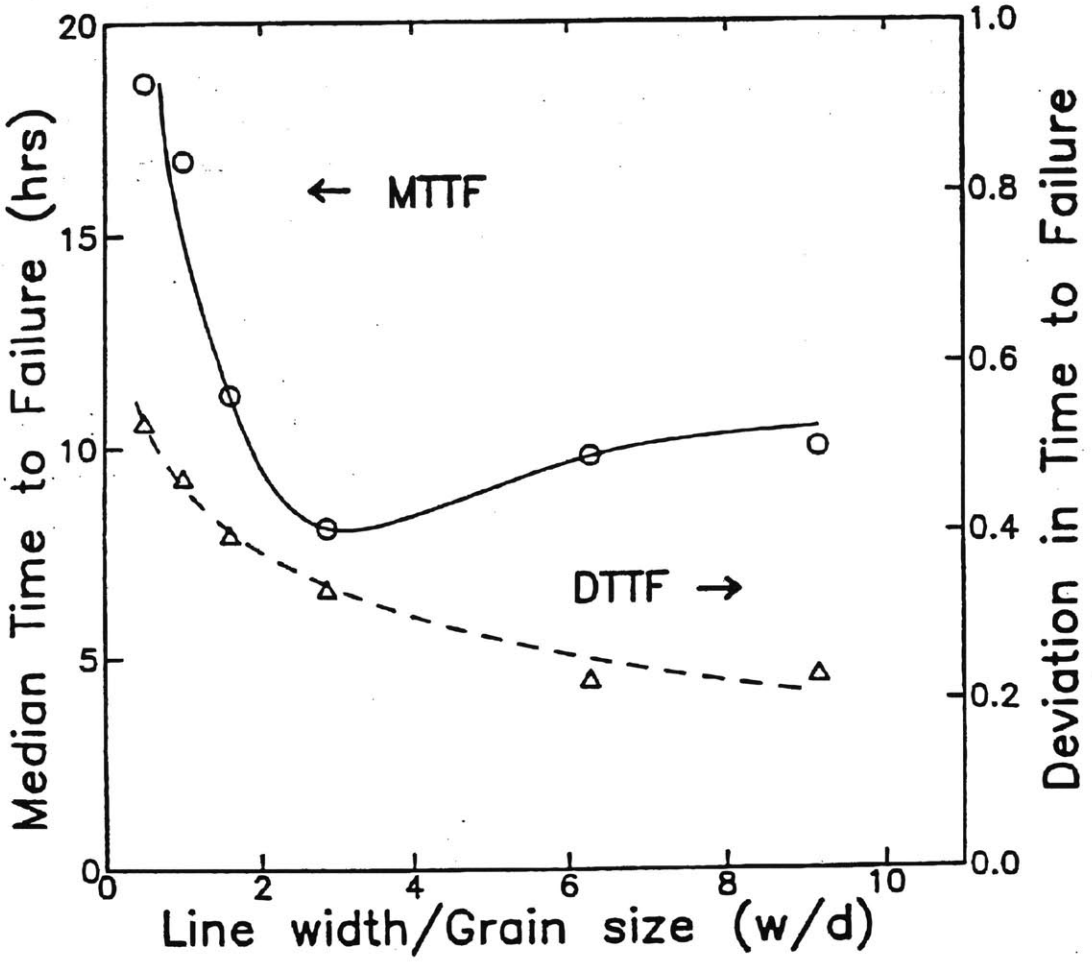
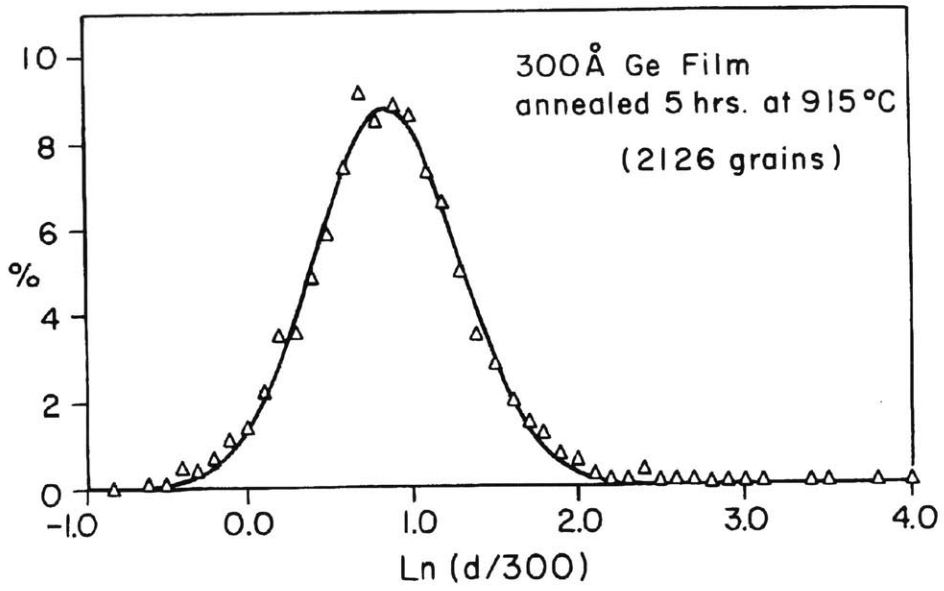


FIG. 3.1: Microstructure dependence of electromigration-induced failure times. The median time to failure (left axis) and the deviation in time to failure (right axis) are plotted as a function of w/D_{50} [Cho 89] where w is the line width and D_{50} is the median grain size in the film.

3.2 Thin Film Grain Structure

In the previous chapter, ideal capillarity-driven grain growth was shown to yield, after transients, a universal steady-state structure that constantly evolves in a self-similar fashion. In real thin films, however, a variety of drag forces lead to significantly different structures where grain growth stagnation is observed when the average grain size approaches 2 or 3 times the thickness of the film [Bec 49, Pal 87]. Such non-ideal effects include surface grooving at the grain boundaries [Mul 58, Fro 90] and solute drag [Fro 94]. The resulting grain structures typically have lognormal grain size distributions [Pal 87, Tra 88], as can be seen in Fig. 3.2. Using the grain growth simulation outlined in Chapter 2, it was possible to include grain boundary grooving effects [Fro 90] by assuming that boundaries with local curvatures smaller than a critical value κ_{cr}, are unable to migrate. This procedure yields fully stagnant simulated grain structures, which closely resemble the ones observed experimentally. The stagnant structures have a lognormal grain size distribution with an average grain size proportional to the inverse of the critical curvature. Such a structure, along with a lognormal plot of the resulting grain diameter distribution, is depicted in Fig. 3.3.

(a)



(b)

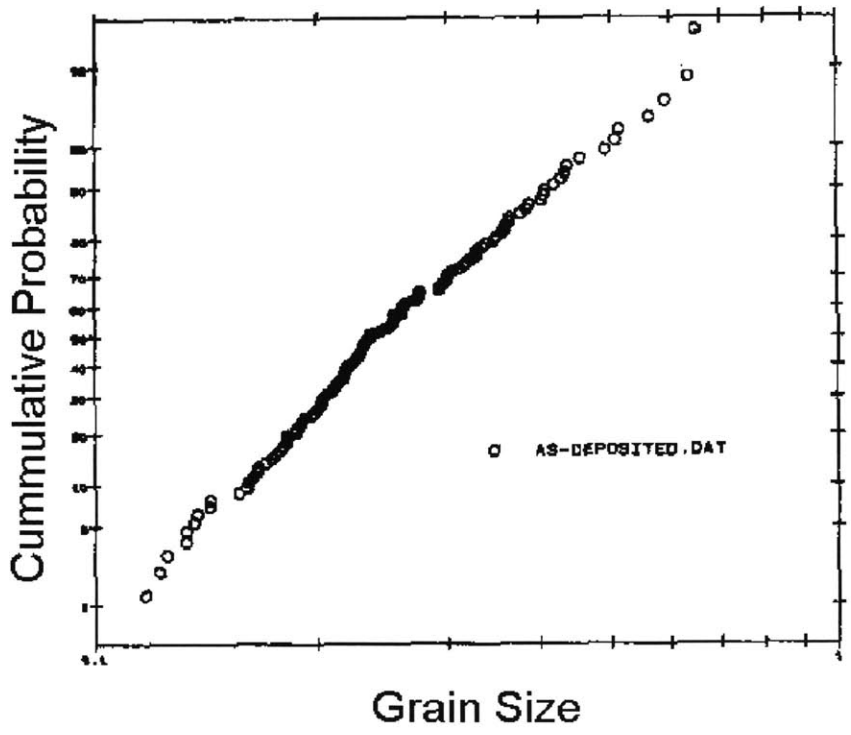
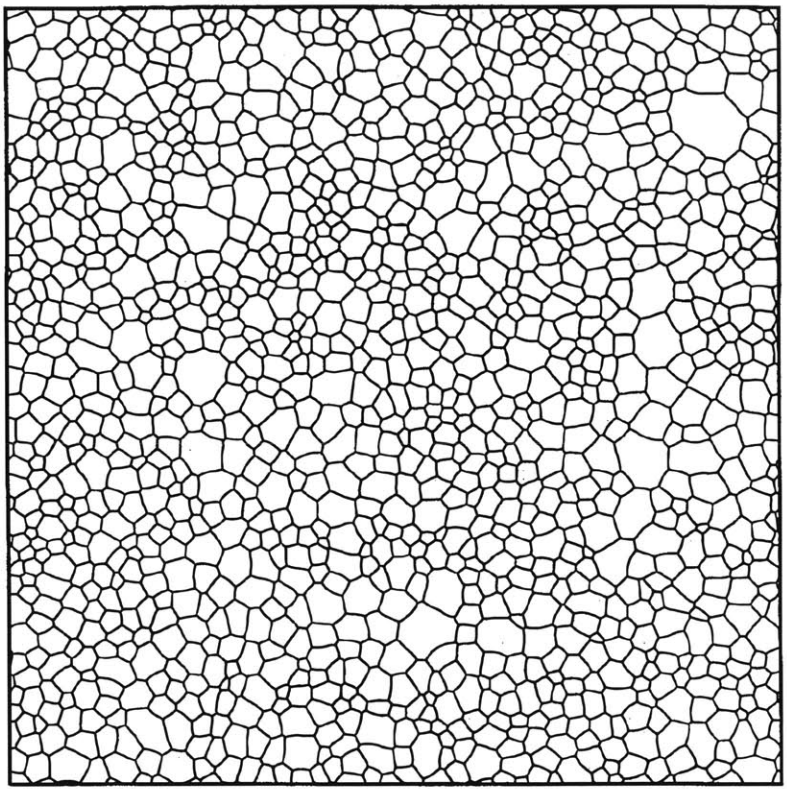


FIG. 3.2: Grain size distribution in metal thin films. **(a)** A linear plot of the probability density function obtained for Germanium [Pal 87]. The solid line is the best fitting lognormal distribution function; **(b)** a lognormal plot of the distribution obtained for Aluminum [Tra 88].

(a)



(b)

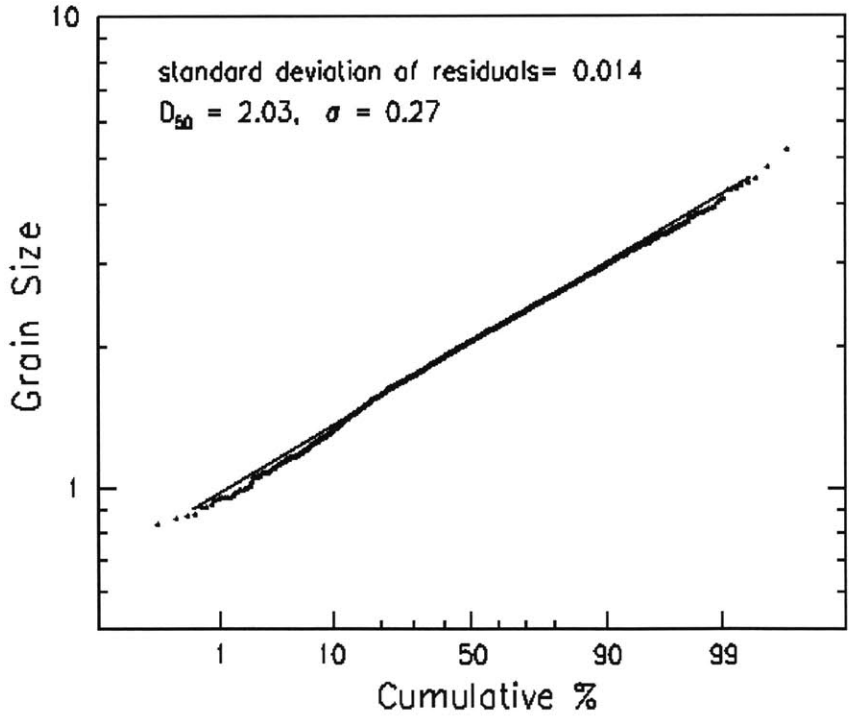


FIG. 3.3: **(a)** Simulation of a thin film grain structure that has reached full stagnation. **(b)** Lognormal plot of the corresponding grain size distribution.

3.3 Grain Structure Characteristics of Interconnects

We have used the *etch* [Wal 91] feature of the grain growth simulation to generate interconnect lines with different widths. Figure 3.4 shows a stagnant film structure with a lognormal grain size distribution, patterned into strips of different widths. As depicted in the figure, the grain structure of the simulated interconnect strips depends on the ratio of the line width w to the median grain size D_{50} of the film. As w/D_{50} decreases, the nature of the lines changes from a fully polygranular structure ($w/D_{50}>2$) to a near-bamboo structure, characterized by polygranular clusters separated by bamboo segments.

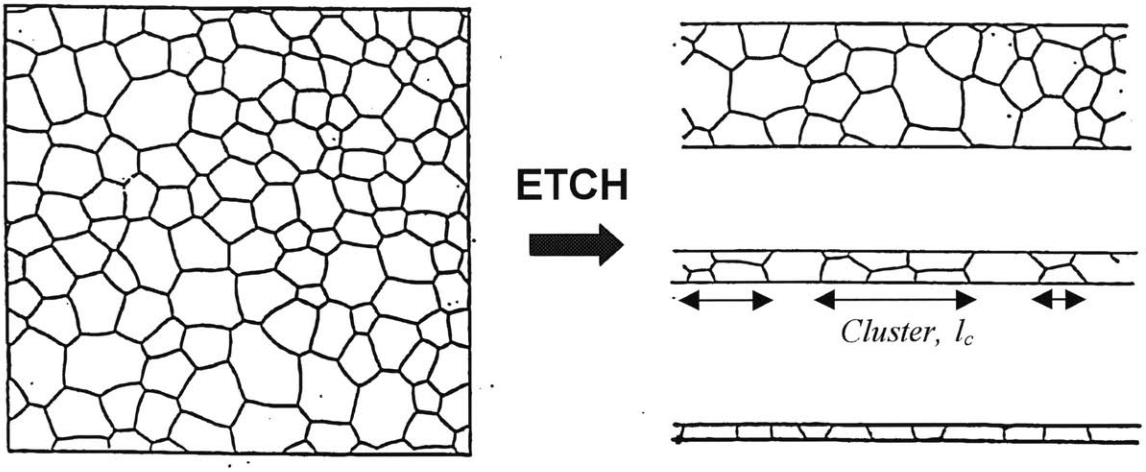


FIG. 3.4: As-patterned grain structure of interconnects with different widths.

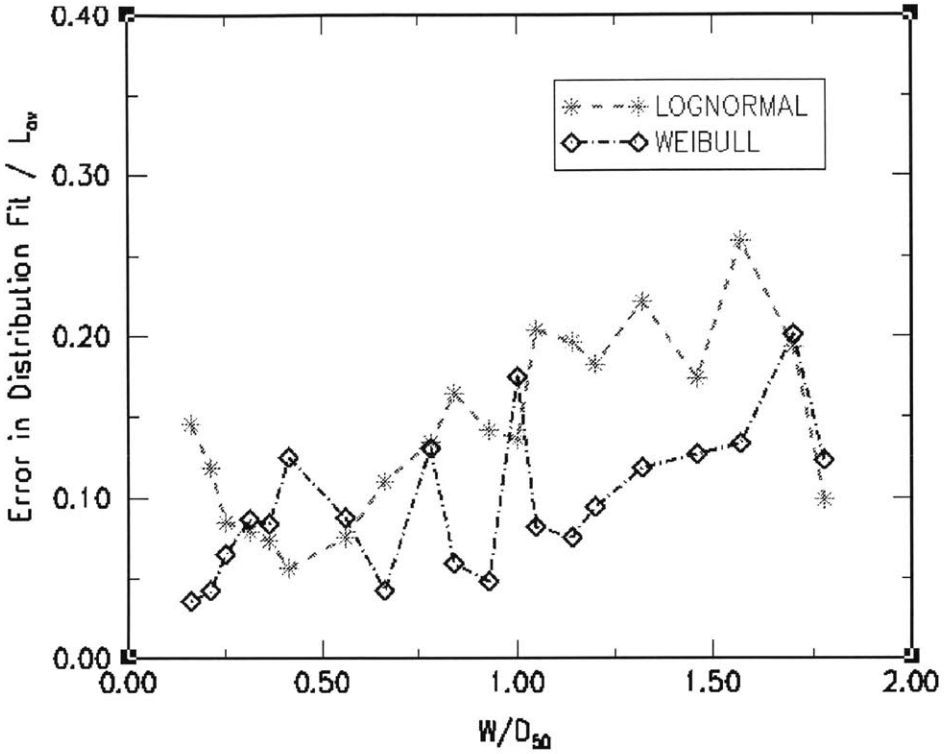
Previous work suggested that the polygranular cluster length distribution in as-patterned near-bamboo lines is well described by lognormal distributions with a tail for long clusters well fit by an exponential distribution [Joo_2 94]. We used our grain growth simulator to generate lines for a wide range of values of the ratio w/D_{50}, and extracted data on polygranular cluster lengths as well as bamboo segment lengths. We compared the fits of lognormal distribution functions to this data with fits of Weibull distribution functions. A Weibull distribution function is, as seen in the previous chapter, characterized by a cumulative distribution function F of the form:

$$F(L)=1-e^{-(L/\alpha)^{\beta}}\ , \qquad (3.1)$$

where α and β are treated as independent fitting parameters. It should be noted that the special case $\beta=1$ gives the exponential distribution with a mean α, which motivated the choice of this particular distribution for the fitting. Figure 3.5 depicts a comparison of the normalized errors in the fittings between Weibull and lognormal distributions for both polygranular cluster lengths and bamboo segment lengths. The normalized error is the square root of the average value of the square of the differences between the fitting lengths and the data, normalized by the average value of the data. The figure shows, that for the wide range of w/D_{50}, $0.1<w/D_{50}<2.0$, both polygranular clusters and bamboo segments are better fit by a Weibull distribution than a lognormal distribution. This is not in contradiction with the fact, as previously mentioned, that the exponential distribution, which is a special Weibull distribution function describes the tail of the polygranular cluster length distribution well. It is possible to make Weibull plots of the clusters length distribution as depicted in Fig. 3.6.

The importance of the above results stems from the fact that it becomes possible, if we know the dependence of the Weibull fitting parameters α and β on the key structural variable w/D_{50}, not only to predict key statistical descriptors of the structure of a line, but also to generate realistic interconnect structures, assuming no correlation between clusters. Figure 3.7 is a plot of the w/D_{50}-dependence of these parameters (α is normalized by D_{50}). The parameter β, for both polygranular clusters and bamboo segments, has values that are not very far from 1 and can be described by a linear dependence on w/D_{50}. The w/D_{50} dependence of α in the case of the polygranular clusters is exponential, and in the case of bamboo segments, a function of the form $a(w/D_{50}) + b/(w/D_{50})$ provides a good fit. These functions are consistent with the behavior expected in the extreme cases $w/D_{50}<<1$ and $w/D_{50}>>1$.

(a)



(b)

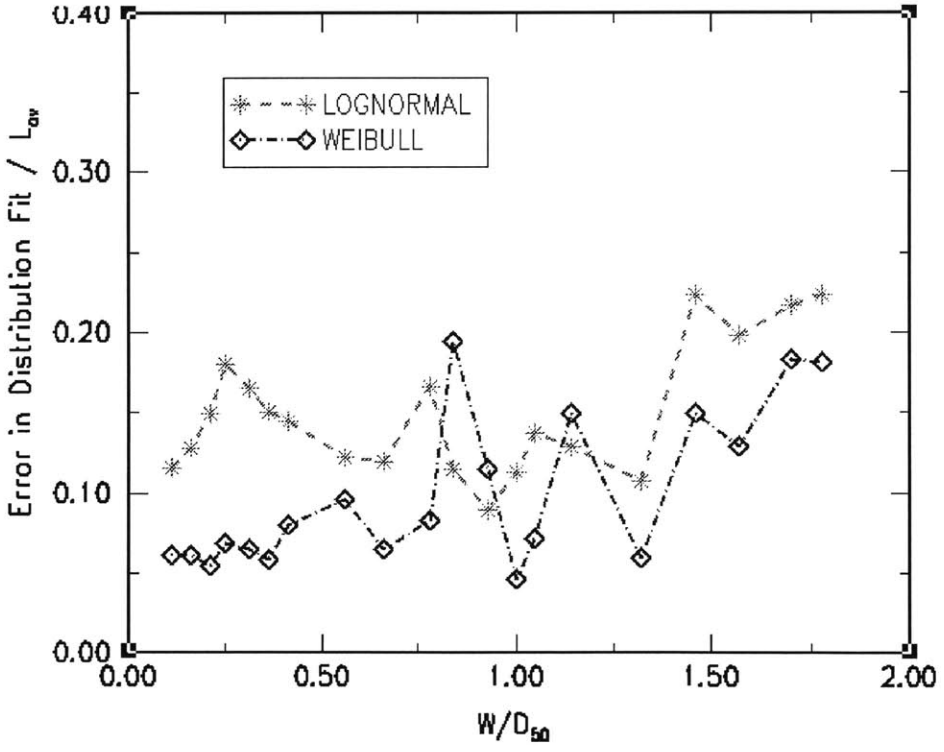


FIG. 3.5: Comparison of the errors from the best fits to Weibull and lognormal distributions **(a)** in the case of polygranular clusters, and **(b)** in the case of bamboo segments.

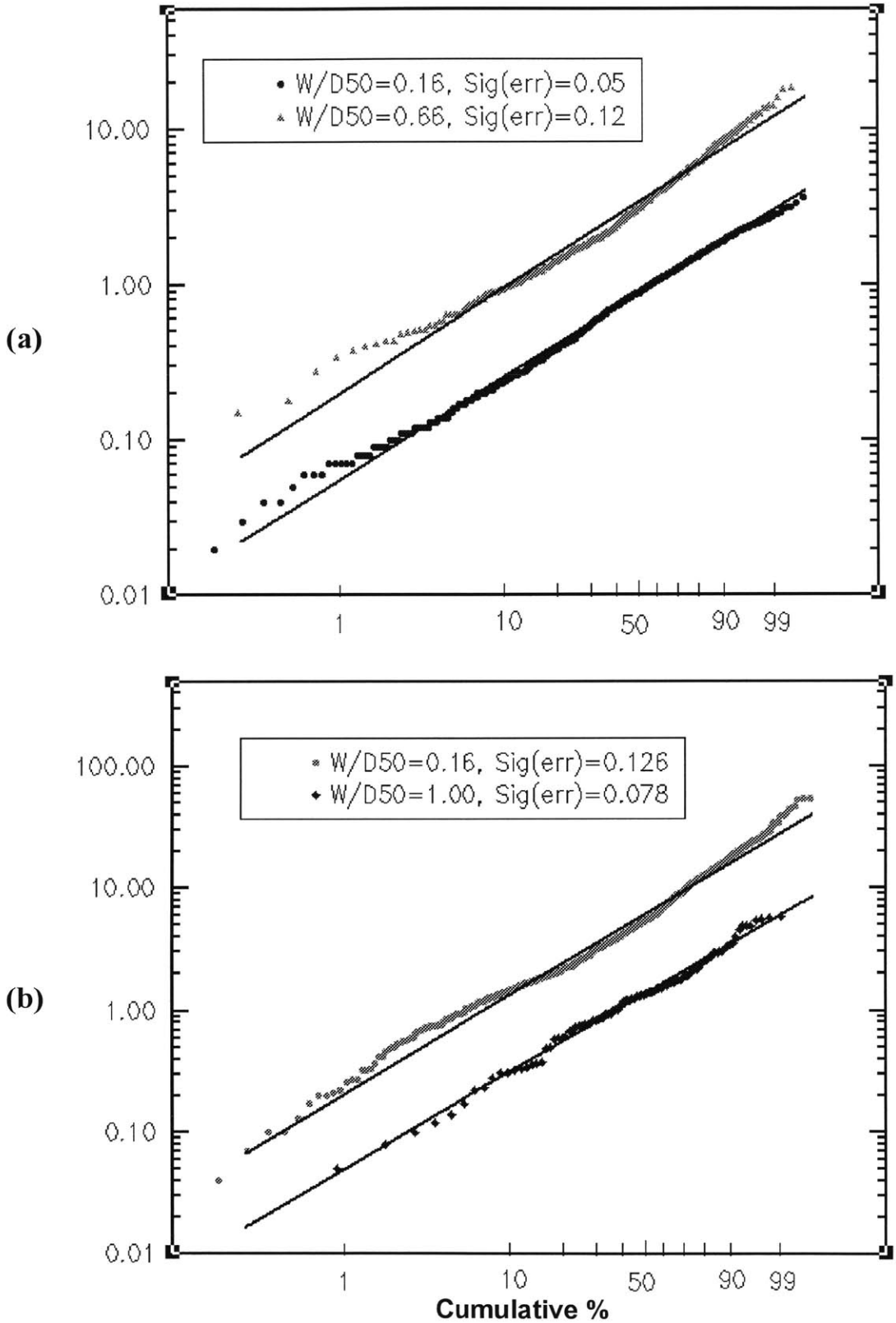
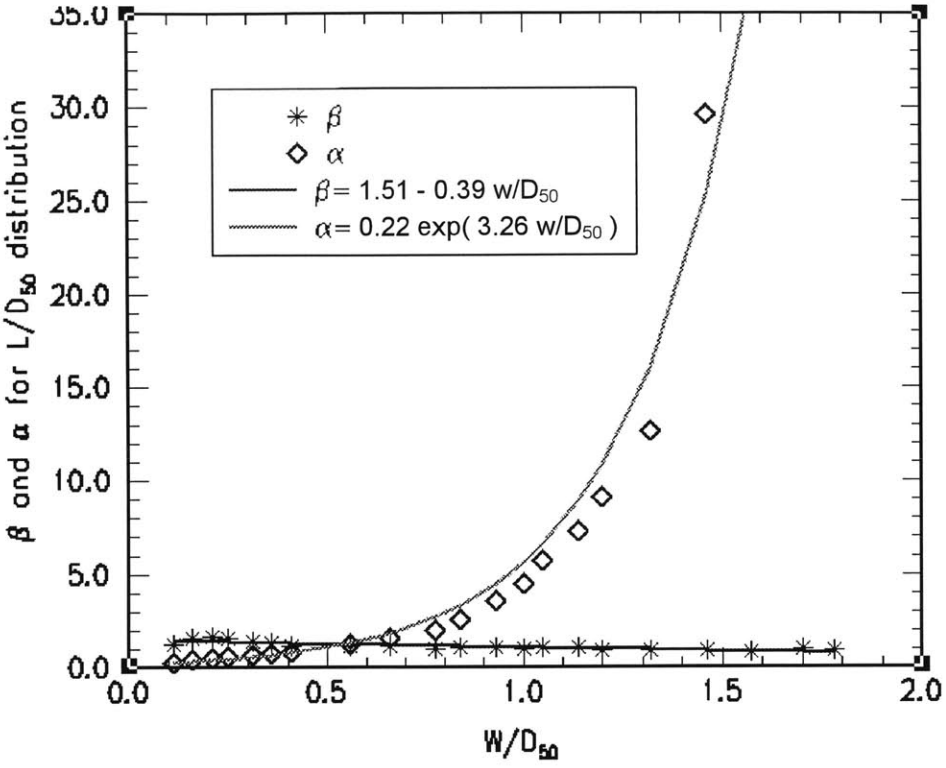


FIG. 3.6: Weibull plots for **(a)** polygranular cluster length distributions, and **(b)** bamboo segment length distributions.

(a)



(b)

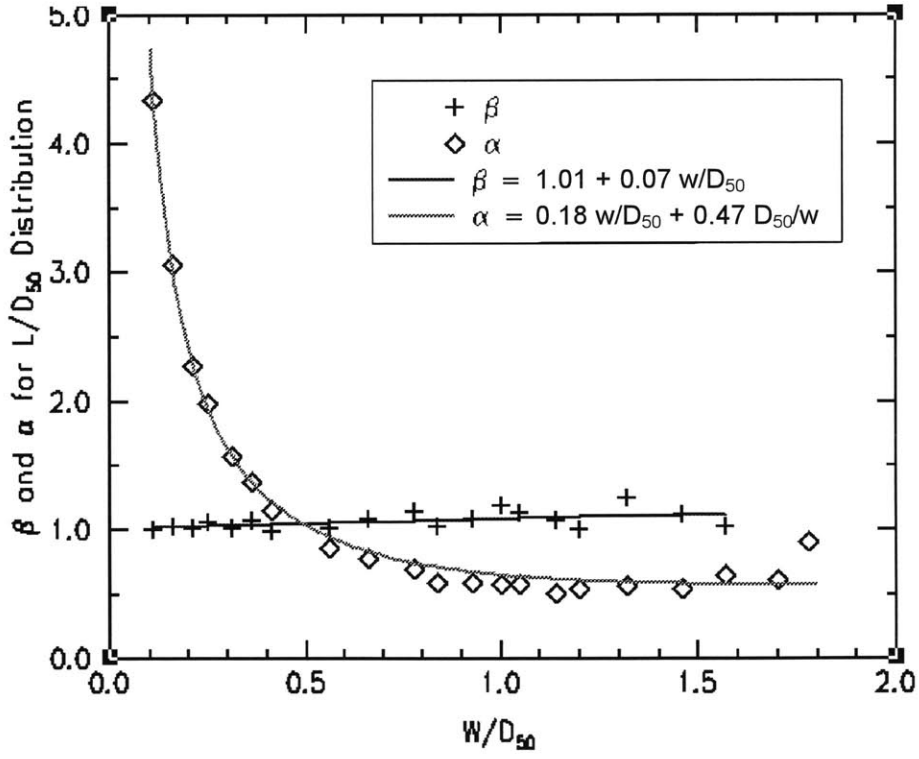


FIG. 3.7: w/D_{50} dependence of the Weibull parameters for **(a)** polygranular clusters, and **(b)** bamboo segments.

Using equations and the data of Fig. 3.7, it is possible to evaluate statistical cluster information such as the average cluster length L_{av}, the total number of clusters in a line N_{tot}, the total fraction of polygranular cluster length in a line L_{ctot}/L_{tot} and the maximum cluster length L_{max}, as a function of the total line length L_{tot} and w/D_{50}. It can be shown that:

$$\frac{L_{av}}{D_{50}} = \frac{\alpha}{\beta}\Gamma\left(\frac{1}{\beta}\right), \tag{3.2}$$

$$\frac{N_{tot}}{L_{tot}/D_{50}} = \left[\frac{\alpha_c}{\beta_c}\Gamma\left(\frac{1}{\beta_c}\right) + \frac{\alpha_b}{\beta_b}\Gamma\left(\frac{1}{\beta_b}\right)\right]^{-1}, \tag{3.3}$$

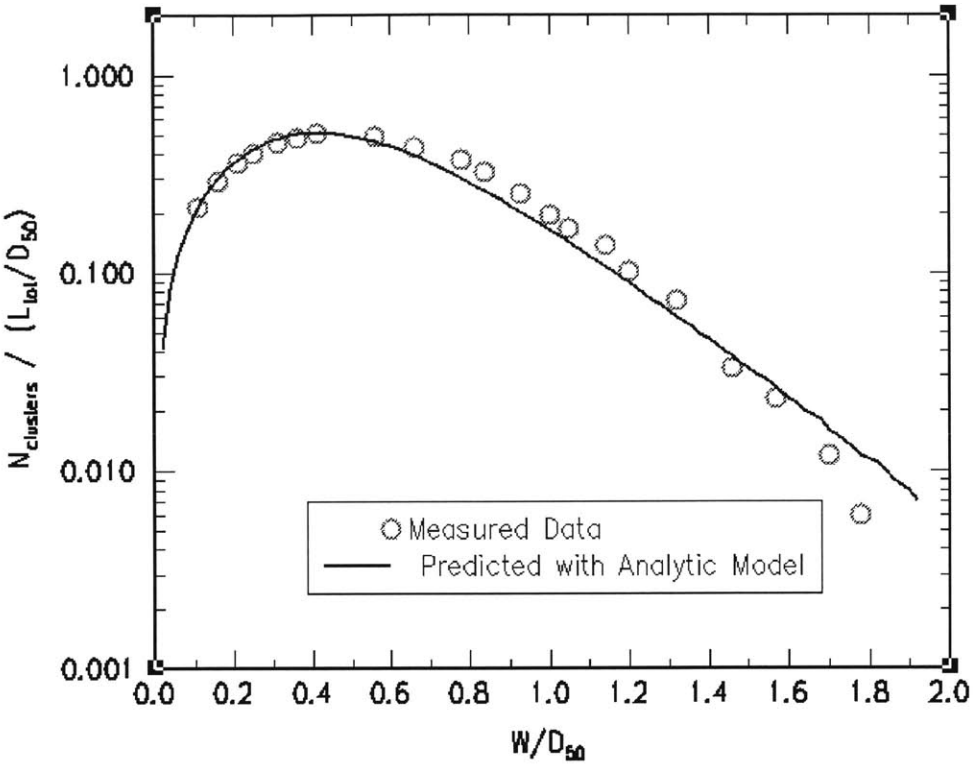
$$\frac{L_{ctot}}{L_{tot}} = \left[\frac{\alpha_c}{\beta_c}\Gamma\left(\frac{1}{\beta_c}\right) + \frac{\alpha_b}{\beta_b}\Gamma\left(\frac{1}{\beta_b}\right)\right]^{-1} \frac{\alpha_c}{\beta_c}\Gamma\left(\frac{1}{\beta_c}\right), \tag{3.4}$$

and

$$\frac{L_{\max}}{D_{50}} = \alpha\left[\ln N_{tot}\right]^{\frac{1}{\beta}}, \tag{3.5}$$

where the subscripts c and b refer to polygranular clusters and bamboo segments, respectively. Equations (3.2) and (3.5) are valid for polygranular clusters and bamboo segments using the corresponding parameters, respectively. An equation symmetric to (3.4), where b and c are switched, is valid for the bamboo length fraction. Figure 3.8 illustrates the results obtained with the analytic model for the total polygranular cluster length and the total number of clusters in a line. The figure shows that the models provide good descriptions of the data measured from the simulation. Figure 3.9 (respectively 3.10) validates the model against the simulation data for both the average and the maximum polygranular cluster length (respectively the average and maximum bamboo segment length) as a function of w/D_{50}.

(a)



(b)

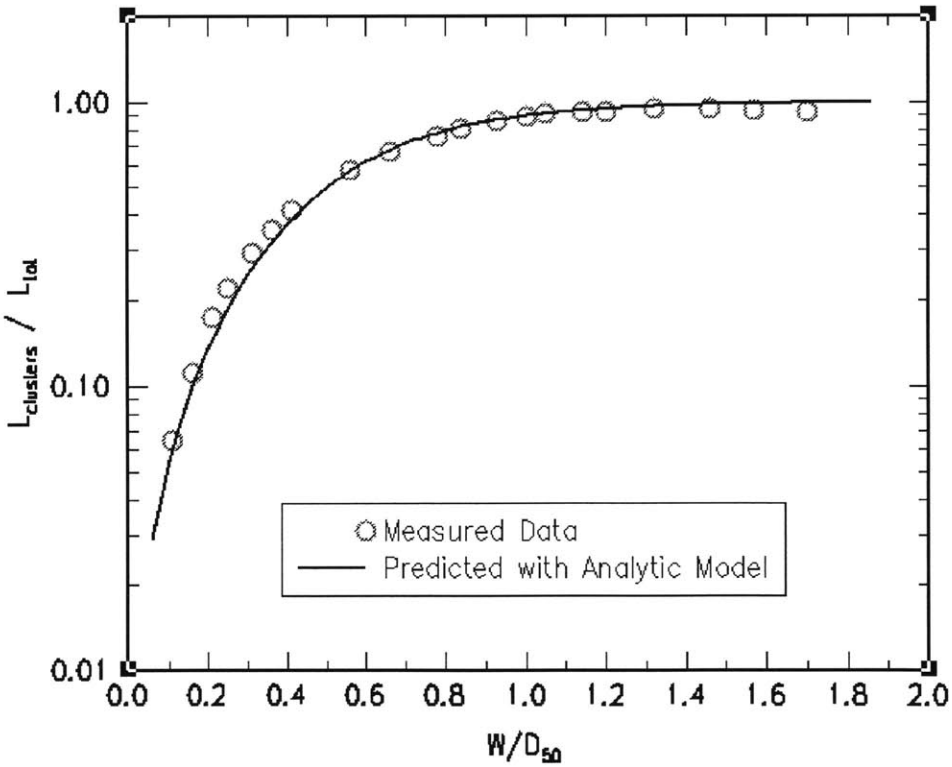


FIG. 3.8: w/D_{50} dependence of **(a)** the total number of polygranular clusters in a line, and **(b)** the total polygranular length fraction of the total line length.

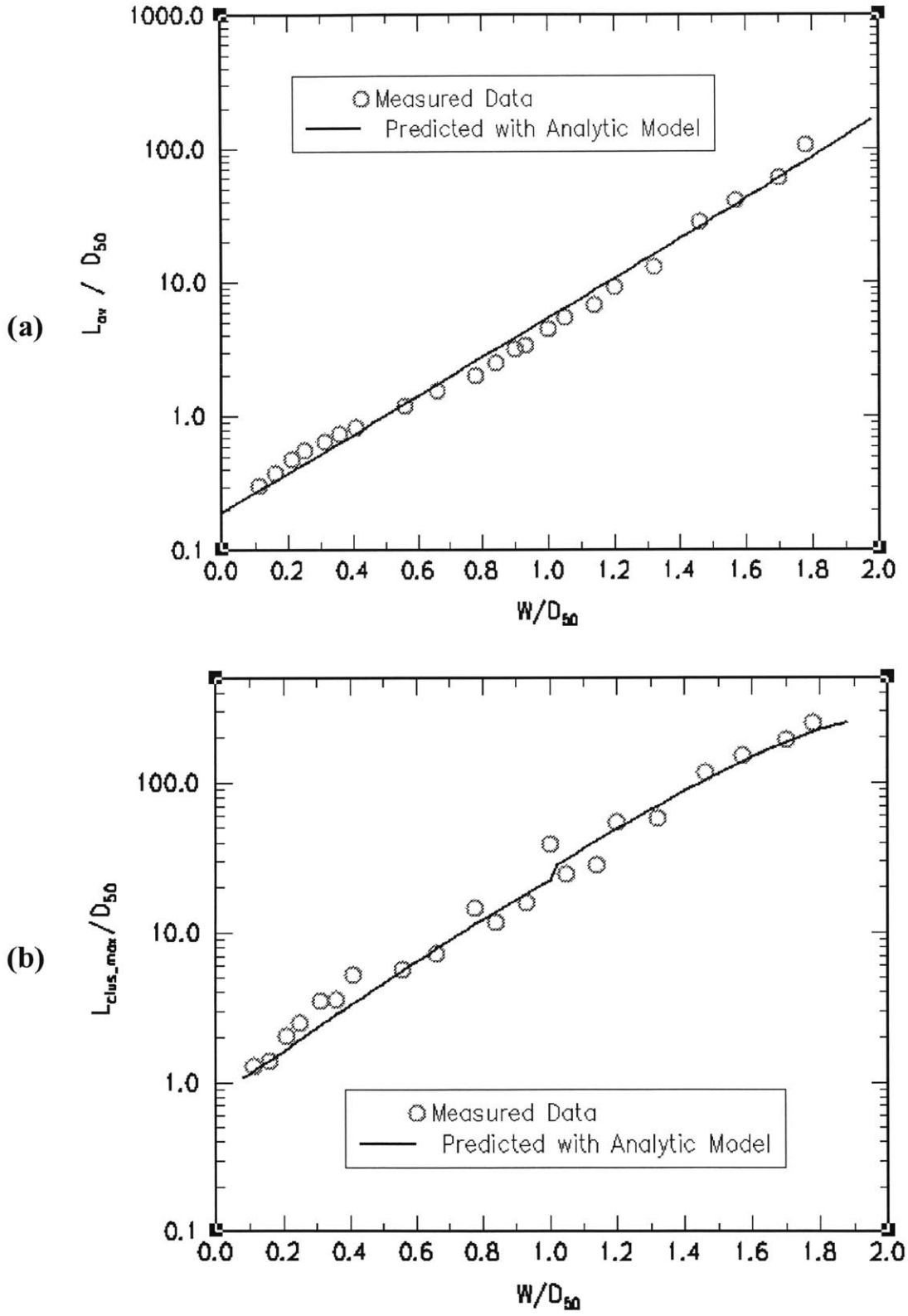
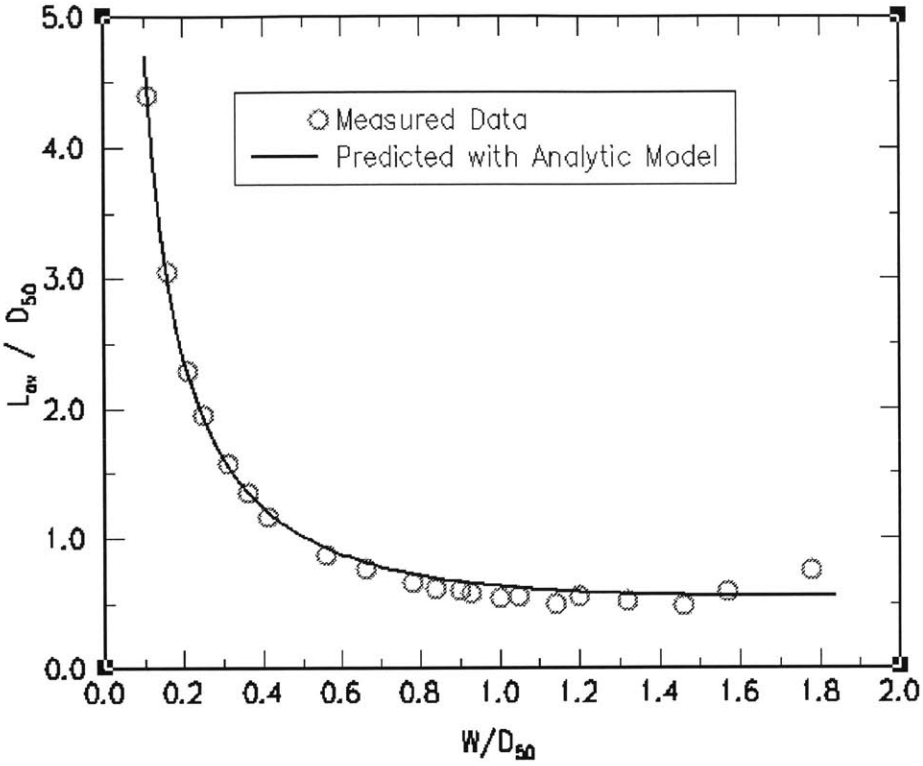


FIG. 3.9: w/D_{50} dependence of **(a)** the average polygranular cluster length in a line; and **(b)** the maximum polygranular cluster length (this plot is adjusted to account for the fact that the total line length is not the same for all the simulation data points).

(a)



(b)

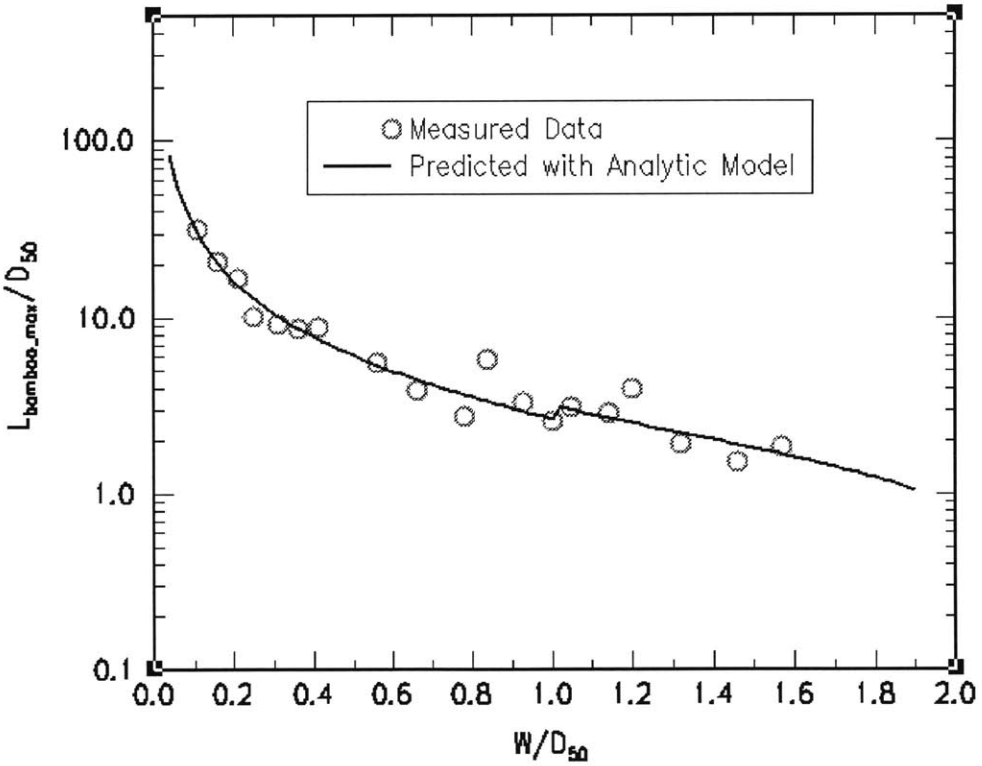
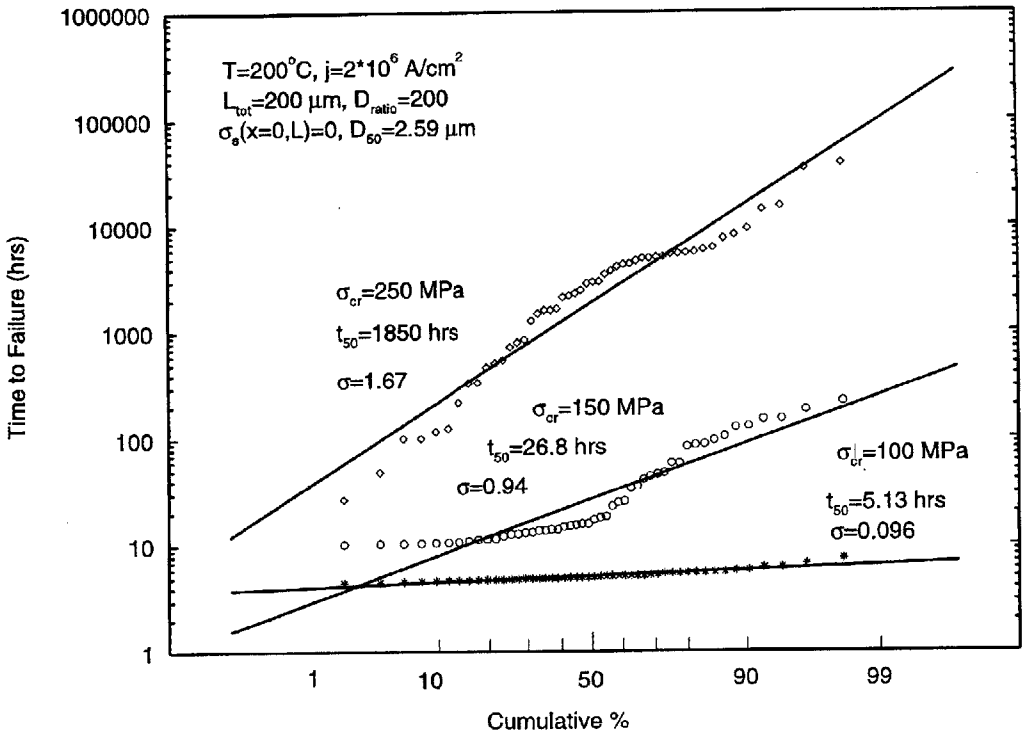


FIG. 3.10: w/D_{50} dependence of **(a)** the average bamboo segment length in a line; and **(b)** the maximum bamboo segment length (this plot is adjusted to account for the fact that the total line length is not the same for all the simulation data points).

3.4 Electromigration Simulation Using the Grain Structure Model

We have seen that polygranular cluster and bamboo segment length distributions in as-patterned interconnects are well fit by Weibull distribution functions. We used this observation in conjunction with the w/D_{50}-dependence of the fitting parameters to construct an analytic model describing the statistics of interconnect grain structures. Based on this model, we have developed a program that provides microstructural statistical information to the user, based on input parameters such as the initial film median grain size and the line width. This program, *Cluster2.0*, is available at http://web.mit.edu/cthomp/www. We also used the analytic models to generate interconnect lines with realistic microstructures, and used the resulting corresponding diffusivity profiles as input to our electromigration simulation program MIT/EmSim (http://nirvana.mit.edu/emsim) to determine electromigration-induced failure time statistics. MIT/EmSim [Demo 98, Kno 95, Kno 97] is based on the Korhonen 1D stress evolution model [Kor 93] as outlined in Chapter 1. We here assume uniform diffusivities within a polygranular cluster, with a diffusivity that is 200 times larger than the bamboo diffusivity. A plot of the times to failure obtained for a population of 50 lines with $w/D_{50} = 0.5$, generated using the analytic models, is shown in Fig. 3.11, along with the same type of statistics for line structures generated directly using the grain growth simulation [Kno 97] (these simulations are based on a void-nucleation failure criterion). The predicted failure statistics are essentially the same for the two methods.

(a)



(b)

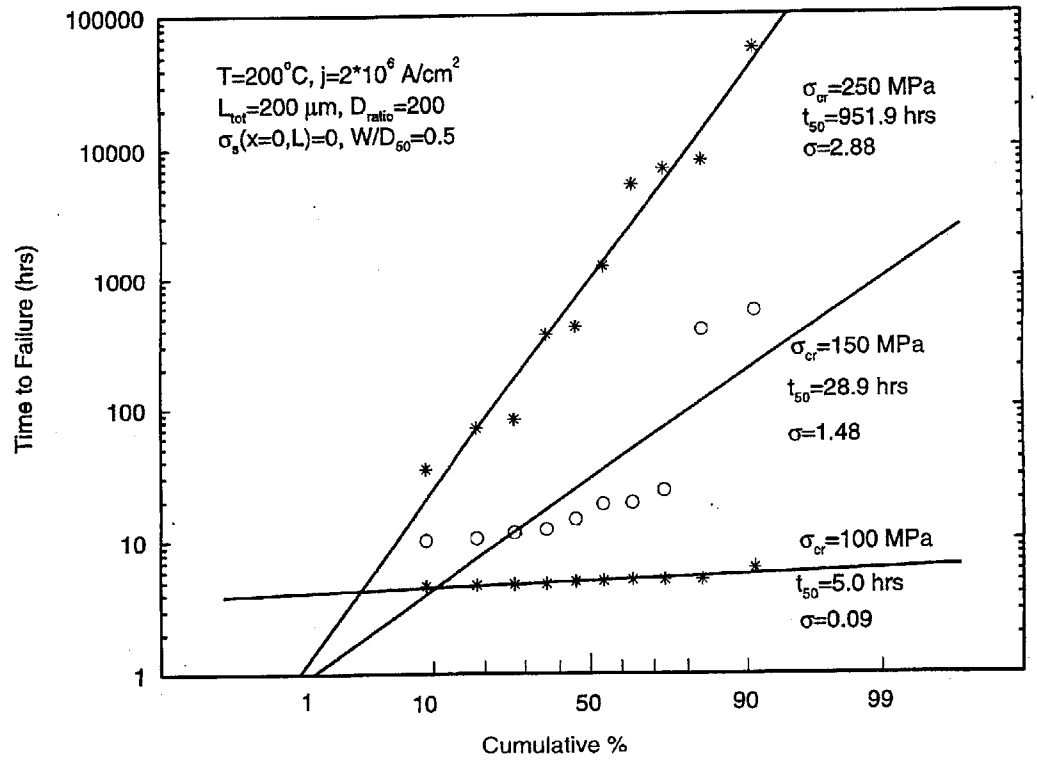


FIG. 3.11: Failure time statistics for **(a)** a population of 50 lines generated using Weibull distribution functions for polygranular and bamboo segment lengths, and **(b)** a population of 10 lines generated with the grain growth simulation.

3.5 Summary

The development of grain structures in polycrystalline films with lognormally distributed grain sizes was simulated, and an extensive characterization of polygranular cluster and bamboo segment length statistics was carried out. These statistics were characterized as a function of line-width to initial-grain-size ratios in as-patterned strips. Among the important findings is that polygranular cluster and bamboo segment length distributions for as-patterned lines are best fit by Weibull distribution functions instead of lognormal or exponential functions, as has been previously assumed. We report analytic formulae describing grain structure statistics that can be used in reliability calculations and simulations. We have carried out structure-sensitive electromigration simulations to show that the predicted failure statistics are essentially the same for grain structures generated using grain growth simulations and grain structures generated using our new analytic models. Analytic models provide computationally efficient means of determining the lifetime variations associated with grain-structure variations.

In the following chapter, we report on the post-patterning annealing-induced time evolution of the interconnect grain structure statistics.

Chapter 4

Analytic Model for Interconnect Grain Structure Evolution During Normal Grain Growth

4.1 Introduction

It has been well established in experiments that the geometry and process-dependent microstructures of on-chip integrated-circuit Al-based interconnects have a direct impact on their electromigration-limited reliability [Vai 80, Kin 80, Kwo 88, Cho 89]. In the previous chapter, we analyzed and modeled the effects of geometry on the grain structure statistics of interconnects. In this chapter, we will investigate the interconnect microstructure evolution due to thermal processing.

Interconnects can have polygranular structures for which there are continuous grain boundary paths along the length of the interconnect (see the first snapshot in Fig. 4.1). This is likely when the median in-plane grain size of the film from which an interconnect is patterned, D_{50}, is smaller than the line width, w. Post-patterning annealing can lead to grain growth that results in bamboo structures for which all grain boundary planes are normal to the interconnect length (see the last figure in Fig. 4.1). At intermediate stages, interconnects can have near-bamboo structures for which polygranular clusters with grain boundaries along the interconnect length are separated by one or more grains which span the width of the line (see Fig. 4.1). Lengths of line in which one or more neighboring grains span the line-width constitute bamboo segments.

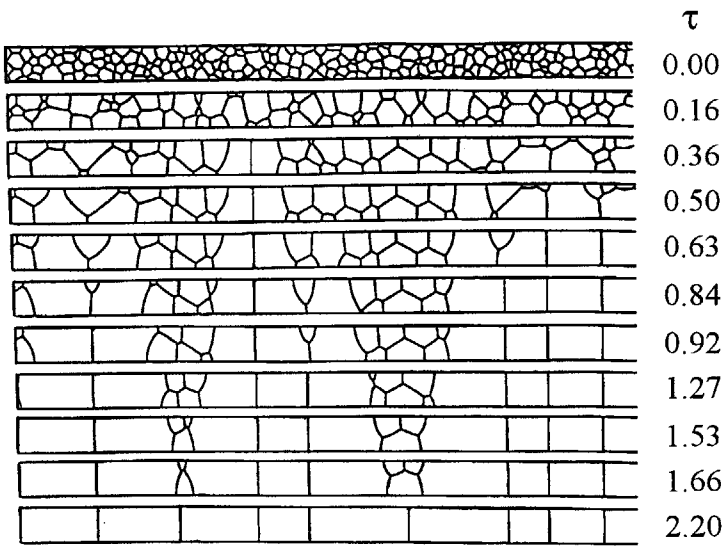


FIG. 4.1: Evolution of the grain structure within a strip with $w/D_{50}=3.0$. w is the line width and D_{50} is the median grain size in the film the line is etched from. $\tau=\mu t/w^2$ is the normalized dimensionless time.

In interconnects with near-bamboo structures, electromigration-induced diffusion occurs much faster along grain boundaries in the polygranular segments than in the bamboo segments, where diffusion occurs primarily along the Al-oxide or Al-intermetallic interfaces [Joo 97, Sri 98]. Because grain boundary diffusivities are orders of magnitude higher than diffusivities in bamboo segments, the magnitudes and statistics of electromigration-induced failure times and statistics depend strongly on the distributions of the lengths of polygranular clusters and bamboo segments [Kno 95, Kno 97]. As the ratio of the line-width to the median grain size of the initial metal film decreases, the polygranular clusters in as-patterned films become shorter and an increase in the median time to electromigration-induced failure is observed, along with an increase in the deviation in the time to failure [Cho 89, Kno 95, Kno 97]. However, if such lines are annealed, their reliability can be improved by orders of magnitude due to post-patterning grain growth [Tho_2 93, Kan 97]. This improvement is related to the change in microstructure outlined above, during which polycrystalline segments shrink through formation of new spanning grains in the cluster interiors, and through motion of the boundaries at the edge of the clusters. The goal of this chapter is to use computer simulations of this process [Wal_2 91, Wal 92] to develop an analytic model for the evolution from polygranular to bamboo structures. Such compact analytic models are

needed for simulations of the effects of processing on the rates of electromigration-induced damage and on interconnect reliability.

In the following sections we will present a Johnson-Mehl-Avrami analysis in which the formation of a line-width-spanning grain is treated as a nucleation event and the shrinkage of polygranular segments corresponds to growth of bamboo segments. We will also develop a differential model which allows line-geometry-sensitive prediction of the evolution of the polygranular-segment-length distribution during normal grain growth. Finally, we will use data generated with a 2D grain-growth simulation to validate our analytic models.

4.2 Nucleation and Growth Johnson-Mehl-Avrami Analysis

Figure 4.1 depicts the transformation in which we are interested. As grain growth proceeds, a few grains grow to span the width of the line, forming bamboo grains and corresponding to "nucleation" of bamboo segments. The boundaries shared with grains in polycrystalline segments continue to move and the bamboo grains continue to grow unless two growing bamboo grains meet. From this perspective, the kinetics of the bamboo segment formation can be treated using a Johnson-Mehl-Avrami (JMA) analysis of 1D nucleation and growth [Chr 75]. Taking L to be the total line length, L_c the total length of polygranular segments in the line, and L_b the total length of bamboo segments (L_b=L-L_c) with an initial value L_{b0}, we can apply the JMA analysis to the portion of the line, of length L-L_{b0}, initially available for nucleation and growth. We define the extended bamboo length L_b^e as the bamboo length expected to be created when the effects of impingement of growing bamboo segments are ignored. If n(t) is the rate of bamboo nucleation per unit time and untransformed length L_c, so that nL_cdt new nuclei form between t and t+dt, v(t) the velocity at which bamboo segments grow, and t_0 the time at which the transformation starts, then, following the JMA analysis,

$$L_b^e = \int_{t_0}^{t} (L - L_{b0}) \cdot l_\tau(t) \cdot n(\tau) d\tau \text{ , with } l_\tau(t) = \int_\tau^t v(\tau') d\tau' . \qquad (4.1)$$

$l_\tau(t)$ is the length at time t of a bamboo segment nucleated at time τ and t_0 is the initial time. The actual bamboo length dL_b created in an interval dt is only a fraction $(L-L_b)/(L-L_{b0})$ of the extended one:

$$dL_b = \frac{L - L_b}{L - L_{b0}} dL_b^e, \qquad (4.2)$$

which after integration gives:

$$1 - \underline{L}_b = (1 - \underline{L}_{b0}) \ exp\left(- \frac{L_b^e}{L - L_{b0}} \right), \qquad (4.3)$$

where $\underline{L}_b = L_b/L$ and $\underline{L}_{b0} = L_{b0}/L$ are the bamboo length fractions, initially and at time t. When the initial condition is characterized by a value of $\underline{L}_{b0}$ different from zero, and an initial number of bamboo segments (or, equivalently, number of polygranular clusters), N_0 (not equal to zero), the nucleation rate n(t) is given by:

$$n(t) = \frac{N_0}{L - L_{b0}} \delta(t - t_0) + a(t) , \qquad (4.4)$$

where $\delta(t)$ is the zero-centered Dirac distribution and a(t) $[L-L_b(t)]$ dt is the number of bamboo nucleation events occurring between $t > t_0$ and t+dt. The first term in the expression of n(t) accounts for the growth of the bamboo sections present initially. In the special case that the nucleation rate "a", as well as the growth rate "v" are constant, (4.1), (4.3) and (4.4) lead to:

$$1 - \underline{L}_b = (1 - \underline{L}_{b0}) exp\left(- \frac{N_0}{L - L_{b0}} v(t - t_0) - \frac{1}{2} va(t - t_0)^2 \right). \qquad (4.5)$$

It is also possible to evaluate the evolution of the number of bamboo grains N_b (N_b is larger, and generally, strictly larger, than the number of bamboo segments and the number of polygranular clusters, N). Using a similar analysis, we obtain:

$$N_b = N_{b0} + \int_{t_0}^{t} (L - L_b) a(\tau) d\tau , \tag{4.6}$$

where N_{b0} is the initial number of bamboo grains. The JMA analysis allows characterization of the evolution of the total length-fraction of bamboo or cluster regions, but does not allow characterization of the evolution of the statistical characteristics of individual cluster lengths. In the following section, we will present a geometry-sensitive model that can accomplish this task.

4.3 Direct Differential Model for Cluster Statistics Evolution

The guide to the development of our analytic model is a front-tracking 2D simulation of boundary-curvature-driven grain growth [Fro 88, Fro_2 88, Fro 90]. The velocity of points on grain boundaries is taken to be proportional to the local curvature κ, such that $v = \mu \kappa$ where the mobility term μ is the temperature-dependent product of the grain boundary mobility and the grain boundary energy. At grain boundary triple junctions, a local force balance is enforced so that grain boundaries meet at 120°. Normal 2D grain growth in thin film or cellular structures has been investigated elsewhere [Fro 88, Fro_2 88, Fro 90, Fay 99]. It has been shown that 2D normal grain growth leads to a uniquely defined grain structure, evolving in a geometrically and statistically self-similar fashion, with an average grain area increasing linearly with both time t and mobility μ (assuming that μ is uniform and constant). The transition to this steady state regime occurs soon after growth has been initiated, typically before the grains have undergone a 20% increase in area.

2D normal grain growth in strips can be simulated by requiring the contact angles of grain boundaries with strip edges to be 90^0 [Wal$_2$ 91]. This leads to the evolution of a bamboo structure as illustrated in Fig. 4.1. Two strips of different width etched from a film with a pre-etching grain size smaller than the line widths will undergo similar grain structure evolution (their initial structure is similar to a 2D cellular structure). Differences in grain structure evolution in strips of different widths can be accounted for through kinetic and geometric scaling. As suggested by Fig. 4.1, the evolution towards bamboo is governed by three phases. During a first *incubation* phase, grains in the initially polygranular structure grow to a size comparable to the strip width. As the evolution proceeds, some grains grow large enough to span the width of the strip, creating sections of the strip with a bamboo structure. This is the *nucleation* period, during which bamboo sections continue to form, increasing the number of polygranular clusters, until these polygranular clusters separating the bamboo segments achieve geometrically stable configurations with grains having four sides within the strip interior and one side defined by the strip edge. The structure then undergoes a *growth-dominated* evolution during which almost no nucleation is observed, with the evolution occurring exclusively at the boundaries between grain clusters and bamboo segments, with the bamboo segments growing to consume the polycrystalline clusters. Given the growth conditions discussed previously and the fact that in polygranular lines the initial structures have an average area that is negligible compared to the square of the width w, the duration of these phases will be proportional to w^2/μ. We can therefore define a dimensionless time $\tau=\mu t/w^2$ that unifies the kinetic analysis for strips with different widths. The grain structures of two lines with different widths will be statistically identical at a given τ, after accounting for geometric magnification. It is therefore also possible to define reduced dimensionless variables to account for geometric scaling in the evolution of: the number of bamboo grains in a line N_b ($\underline{N}_b = N_b.w/L$), the total cluster length in a line L_c ($\underline{L}_c = L_c/L$), the total number of clusters in a line N ($\underline{N} = N.w/L$), the average cluster length $l_{av} = L/N$ ($\underline{l}_{av} = l_{av}/w = \underline{L}_c/\underline{N}$), and the average bamboo length l_b ($\underline{l}_b = l_b/w$). All of these geometric parameters scale as a function of the reduced time $\tau=\mu t/w^2$. The evolution of these dimensionless variables is expected to be independent of line geometry, provided the initial width is larger than the pre-etching grain size.

In previous work [Wal 92], Walton et al. analyzed the kinetics of the transformation of the structure of a polygranular strip to a bamboo structure. The three phases of evolution were identified, and it was argued that bamboo "nucleation" occurred randomly within the polygranular regions, leading to an exponential distribution of polygranular cluster lengths. Walton et al. then focused on the growth-dominated phase to show that it was characterized by an exponential decay of N and L_c, as well as by a constant value for the average cluster length in the strip. This behavior is consistent with an exponential distribution for the length of clusters, and with cluster shrinkage at a constant rate v [Wal 92]. Although it is true that the growth-dominated phase is the one that governs the kinetics of the bamboo transformation through the shrinkage and elimination of polygranular clusters, knowledge of the rate of polygranular cluster shrinkage alone does not allow complete characterization of the evolution of the grain structure statistics. To accomplish this, one also needs to know the cluster statistics (number and length) at the beginning of the growth-dominated phase. This, in turn, depends on what happens during the nucleation phase.

To capture the essentials of the structural evolution during both the nucleation and growth-dominated phases, we will first show that at any time during the evolution, the grain structure statistics are well described by exponential distributions. We will then develop a simple analytic model of the cluster length and number evolution that allows the prediction of cluster and bamboo length statistics at any point in time during post-patterning annealing.

Figure 4.2 shows an exponential plot of the polygranular cluster length distribution for a strip with a linewidth-to-initial-median-grain size ratio $w/D_{50}=1.0$, at many different times during all phases of the evolution. When plotting the cluster length l_c as a function of $-\ln(1-F(l_c))$, where $F(l_c)$ is the proportion of clusters shorter than l_c, data that falls on a straight line is fit by an exponential distribution function. Figure 4.2 shows that the polygranular cluster length distribution is well fit by an exponential distribution function at all times. The lines overlap for $\tau > 0.5$, which indicates a constant average cluster length in this regime.

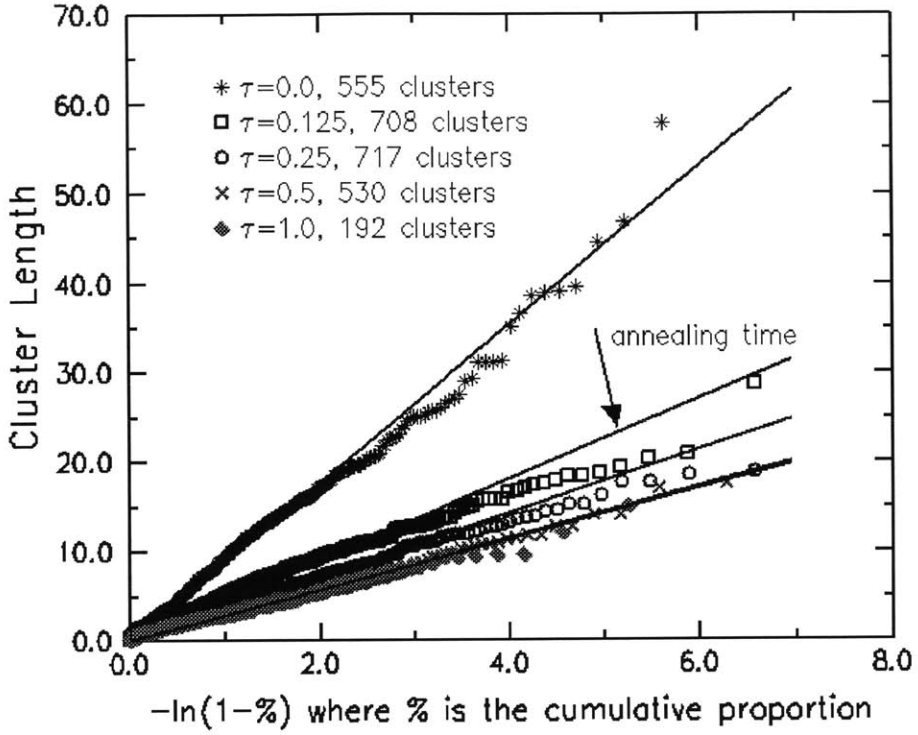


FIG. 4.2: Exponential plot of cluster length distributions at various times during simulated evolution of the grain structure in a thin-film strip with $w/D_{50}=1.0$. Solid lines represent the best-fitting exponential distributions. The plot is consistent with the structure reaching a steady-state in which the average cluster length is constant.

If we consider that during annealing-induced evolution, the polygranular cluster-length distribution is, and remains, exponential, the problem of predicting the structure statistics is reduced to the determination of the evolution with time of both the average cluster length $l_{av}(t)$ and the total number of clusters N(t), or equivalently, one of these variables and the total cluster length $L_c(t)$. Assuming that the effect of bamboo nucleation on the total cluster length is negligible, only cluster shrinkage will account for the variations in L_c, so that:

$$\frac{dL_c}{dt} = -v(t)N(t), \tag{4.7}$$

where v(t) is the average value of the rate of polygranular cluster shrinkage at time t. The variation in the total number of clusters is caused by: the increase due to bamboo

nucleation inside a polygranular cluster, or, equivalently, cluster splitting events, and the decrease following cluster disappearance by shrinkage. If a(t) is the rate of splitting events per unit cluster length at time t, as defined previously, and v(t) is the rate at which clusters shrink, then assuming an exponential distribution of cluster length $f_t(l_c) = (1/l_{av}(t)) \exp(-l_c/l_{av}(t))$, the number of clusters that disappear by shrinkage between t and t+dt can be evaluated to be $v(t)N(t)dt/l_{av}(t)$, which leads to the second evolution equation:

$$\frac{dN}{dt} = a(t)L_c(t) - v(t)\frac{N^2(t)}{L_c(t)}. \tag{4.8}$$

These equations can be recast in terms of the reduced dimensionless variables previously defined and two dimensionless parameters, $\underline{v} = (w/\mu)v$ and $\underline{a} = (w^3/\mu)a$. At this point, knowledge of the initial conditions $\underline{L_c}(0)$ and $\underline{N}(0)$ and the profiles of $\underline{a}(\tau)$ and $\underline{v}(\tau)$ allows solving equations (4.7) and (4.8) to determine $L_c(t)$ and N(t). It is important to note that, as mentioned earlier in the section, the evolution of the normalized variables in the case $w/D_{50} >> 1$ is independent of line geometry, which implies that $\underline{a}$ and $\underline{v}$ depend on geometry only through τ. This, in turn, proves that the rate of cluster-splitting "a" is proportional to μ/w3 and the shrinkage velocity v is proportional to μ/w.

The rate of cluster shrinkage has been investigated previously [Wal 92]. In the growth-dominated regime, most polygranular clusters are bound by pairs of 4 and 5-sided grains (counting the strip edge as a side) with a series of 5-sided edge-grains between them. The Mullins-von Neumann law can be used to show that the rate of cluster shrinkage is constant and proportional to 1/w:

$$v = 2\frac{1}{w}\frac{dA_{4side\_edge}}{dt} = \frac{2\pi}{3}\frac{\mu}{w}, \tag{4.9}$$

where it should be noted that when using the Mullins-von Neumann analysis for the rate of shrinkage of individual grains in strips, the strip edge counts as 2 sides [Wal 91]. The

time evolution of the normalized velocity as defined in equation (4.7) is depicted, for several line widths, in Fig. 4.3.

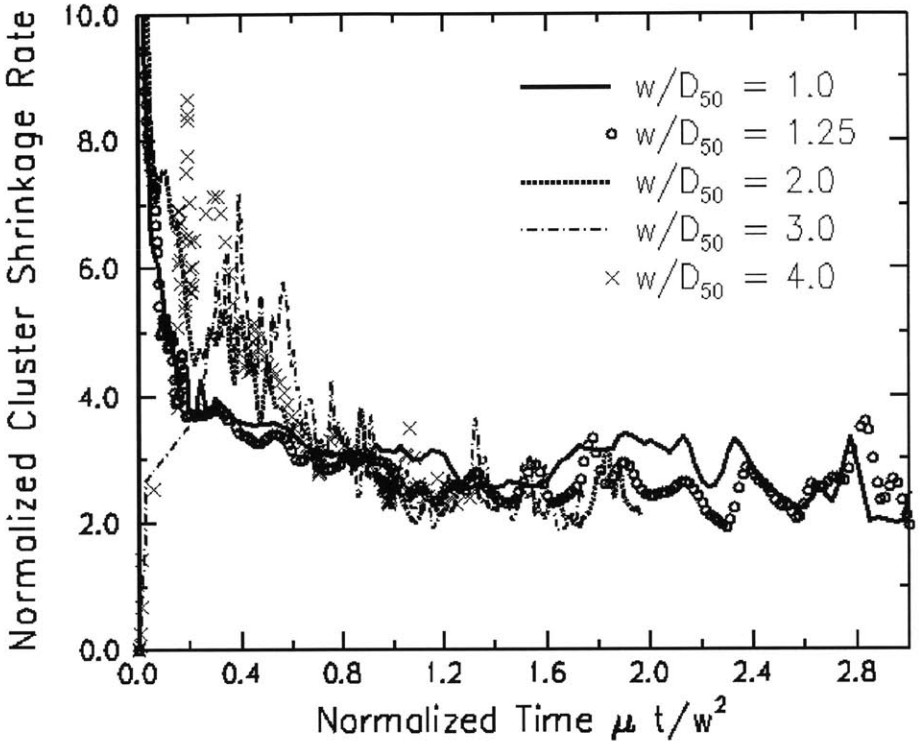


FIG. 4.3: Evolution of the normalized cluster shrinkage rate $\underline{v}=-(1/\underline{N})d\underline{L}_c/d\tau$ with $\tau=\mu t/w^2$. The plots coincide for initially polygranular lines ($w/D_{50}>2$), confirming that the shrinkage rate is proportional to μ/w.

The plot confirms that the average shrinkage rate is constant in the steady-state regime, and that all curves overlap confirms that this rate is proportional to μ/w. The average value of the shrinkage rate in the steady-state exceeds the predictions of equation (4.9) by about 20%, a difference that is accounted for by the high velocities associated with bamboo nucleation at the edge of a cluster. At the early nucleation-dominated stage, the values of "v" are variable and higher. This is expected since, following equation (4.7), it is "v" and not "a" that accommodates the topology-driven cluster length variations due to random nucleation events.

An analytic assessment of the rate of cluster splitting per unit time and unit length, a(t), is more complex. This rate is expected to depend on the average grain size at a given time, as well as on the deviation in the grain size, since it is the number of grains that are bigger than a certain width-related threshold that affects the number of nucleation events. However, as can be seen in Fig. 4.4, and will be discussed in the next section, simulations indicate that significant cluster splitting through nucleation occurs only during the *nucleation* period, and is essentially absent during the steady state growth regime. This behavior is expected since during the steady state phase, clusters have stable geometries which are immune to an internal bamboo nucleation. Therefore a relatively good approximation would be to take a(t) to be constant during the nucleation period in the time interval $[t_0, t_1]$, and zero at other times. This is confirmed by the data in Fig. 4.4, for the evolution of the simulated normalized bamboo nucleation rate $(1/\underline{L}_c)d\underline{N}_b/d\tau$, an over-estimate of the normalized cluster splitting rate $\underline{a}=(w^3/\mu)a$, as a function of $\tau = \mu t/w^2$. The plots coincide within statistical variations for initially polygranular lines ($w/D_{50}>2$), confirming that the nucleation rate is proportional to μ/w^3.

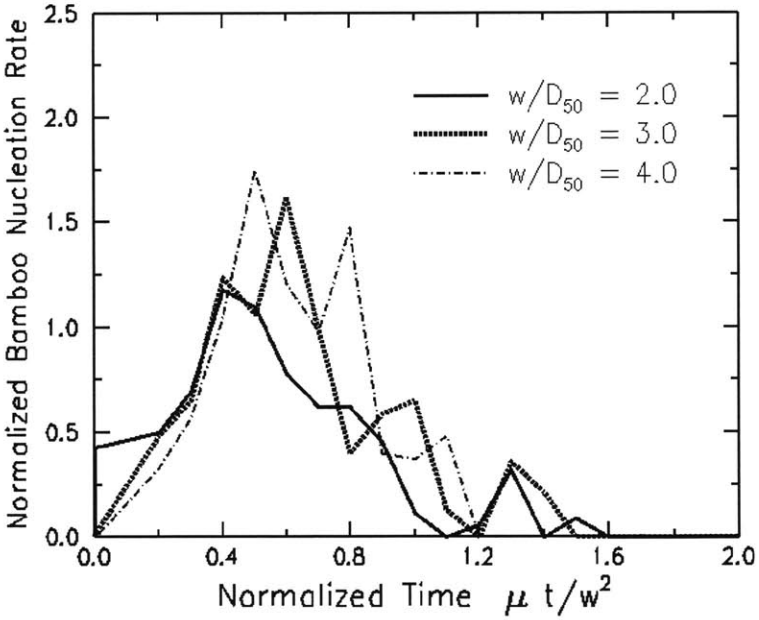


FIG. 4.4: Evolution of the normalized bamboo nucleation rate $(1/\underline{L}_c)d\underline{N}_b/d\tau$ with $\tau=\mu t/w^2$. These plots have been obtained after averaging highly variable instantaneous rates over longer times. The plots coincide within statistical variations showing that bamboo nucleation is only significant during the nucleation phase. They coincide within statistical variations for initially polygranular lines ($w/D_{50}>2$), confirming that the nucleation rate is proportional to μ/w^3.

Using, as a simplification, a cluster-splitting rate a(t) such as the one defined above, and a constant shrinkage velocity, the coupled equations (4.7) and (4.8) can be solved analytically to obtain:

for $t_0 < t < t_1$,

$$L_c = L_{c0} \exp\left(-\frac{1}{l_{av0}} v(t - t_0) - \frac{1}{2} va(t - t_0)^2 \right) \tag{4.10}$$

and

$$l_{av}(t) = \left(\frac{1}{l_{av0}} + a(t - t_0) \right)^{-1}, \tag{4.11}$$

and, for $t > t_1$,

$$L_c = L_c(t_1) \exp\left(-\frac{1}{l_{av1}} v(t - t_1) \right) \tag{4.12}$$

and

$$l_{av}(t) = l_{av}(t_1). \tag{4.13}$$

That the average cluster length reaches a constant value (l_{av} reaching a constant value means also that l_{av} reaches a constant value proportional to w) is in agreement with Walton's results [Wal$_2$ 91]. We also note, as a final remark before discussing simulation results, that equation (4.5) obtained through the JMA analysis and equation (4.10) above are identical, as would be expected for nucleation at a constant rate and growth at a constant velocity.

4.4 Simulation Results and Discussion

We have used the grain growth simulation technique discussed in references [Kno 95], [Kno 97] and [Wal 91] to generate grain structures in strips with different widths, and compared the evolution of the polycrystalline cluster statistics observed using the simulation with predictions made using the analytic model derived above. Table 4.1 shows values for various error-minimizing parameters obtained using the simulation.

Table 4.1: The parameters for simulations of grain structure evolution in lines with different widths w that minimize the error $e = \langle \log^2(\underline{L}_e/\underline{L}_{e\_fit})\rangle + \langle \log^2(\underline{l}_{av}/\underline{l}_{av\_fit})\rangle + \langle \log^2(\underline{N}/\underline{N}_{fit})\rangle$.

w/D_{50}	$\tau_0 = \mu t_0 / w^2$	$\tau_1 = \mu t_1 / w^2$	$\underline{v} = v\, w/\mu$	$\underline{a} = a\, w^3/\mu$.	Error
7.05	0.16	0.75	3.96	0.74	0.026
5.81	0.16	0.75	3.84	0.66	0.020
5.0	0.18	0.75	3.88	0.64	0.022
4.0	0.15	0.75	3.50	0.80	0.013
3.0	0.12	0.75	3.76	0.84	0.019
2.0	0.0	0.75	3.38	0.60	0.018
1.25	0.0	0.45	3.34	0.78	0.024
1.0	0.0	0.40	3.84	0.70	0.035
0.75	0.0	0.35	4.44	0.52	0.055
0.5	0.0	0.35	4.92	0.28	0.096

The simulation shows that during 2D normal grain growth in strips, the incubation period t_0 is approximately $0.15w^2/\mu$ for wide, initially polygranular lines ($w/D_{50} \geq 3.0$). The incubation time decreases rapidly with strip width, and is already zero for $w/D_{50} \leq 2.0$. The nucleation period lasts until $t_1 \approx 0.75\ w^2/\mu$ for all polygranular lines. The fact that $\underline{v}$ and $\underline{a}$ have constant values ($\underline{v} \approx 3.8 \pm 0.2$ and $\underline{a} \approx 0.7 \pm 0.1$) for all polygranular lines, regardless of the line width ($w/D_{50} \geq 3.0$) in the growth-dominated regime validates the assumptions of the analytic model. The value of $\underline{v}$ is consistent with the Mullins- von Neumann analysis described above after accounting for the high velocities associated with bamboo nucleation events. Figures 4.5, 4.6, and 4.7 show the evolution of $\underline{L}$, $\underline{N}$, and $\underline{l}_{av}$ for both simulation data and model predictions in lines with different widths. We note the exponential decay of $\underline{L}$ and $\underline{N}$ during the steady state phase ($\tau > \tau_1$), while $\underline{l}_{av}$ reaches a constant value of about 2 (corresponding to an average cluster length of twice the line

width). The coincidence of the curves for the wide lines ($w/D_{50} \geq 3.0$) is a validation of the geometric scaling analysis presented earlier.

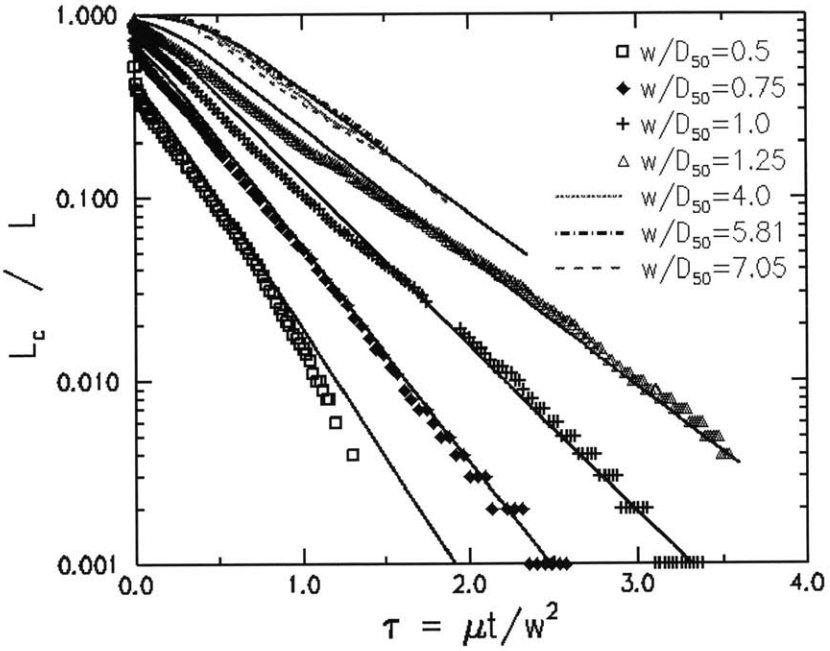


FIG. 4.5: Evolution of the normalized total cluster length $\underline{L}_c = L_c/L$ with $\tau = \mu t/w^2$. The plots coincide for initially polygranular lines ($w/D_{50} > 2$). Solid lines represent the evolution predicted using the analytic model.

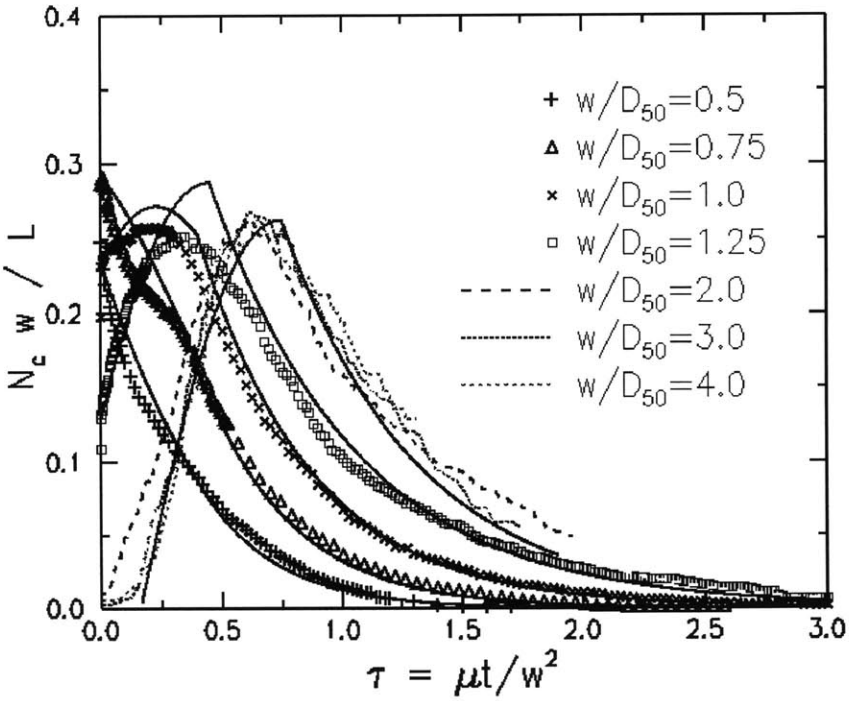


FIG. 4.6: Simulated evolution of the normalized number of clusters $\underline{N} = Nw/L$ with $\tau = \mu t/w^2$. The plots overlap for initially polygranular lines. Solid lines represent the predictions of the analytic model.

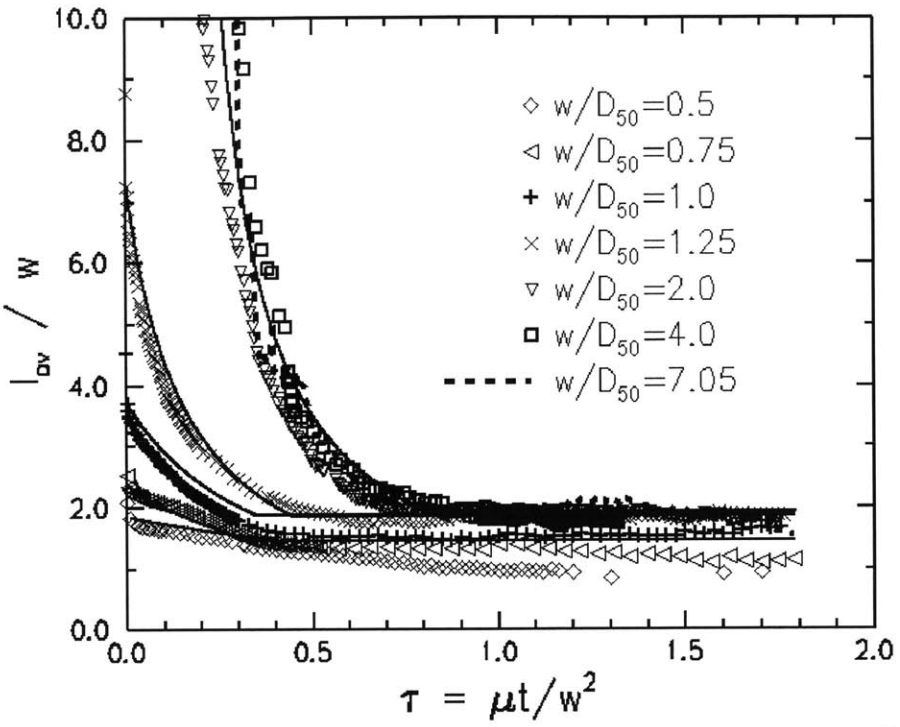


FIG. 4.7: Simulated evolution of the normalized average cluster length $l_{av}=l_{av}/w$ with $\tau=\mu t/w^2$. Solid lines represent the predictions of the analytic model.

The time to reach 90% bamboo structure (in terms of length fraction) for wide lines is about 1.85 w^2/μ, and the final bamboo structure has grains with an average length of about 2.1 times the line width. Figure 4.8 shows that the distribution of the final bamboo grain lengths normalized by the line width is width-independent for initially polygranular lines, as expected with geometric scaling, and is well fit by a lognormal distribution. Figure 4.9 shows the normalized total number of bamboo nucleation events $(N_b-N_{b0})w/L$ as a function of w/D_{50}, demonstrating that the total number of bamboo nucleation events is proportional to L/D_{50} for initially near-bamboo lines ($w/D_{50}<2.0$), and L/w for initially polygranular lines ($w/D_{50}>2.0$), the latter result being consistent with the geometric scaling discussed earlier.

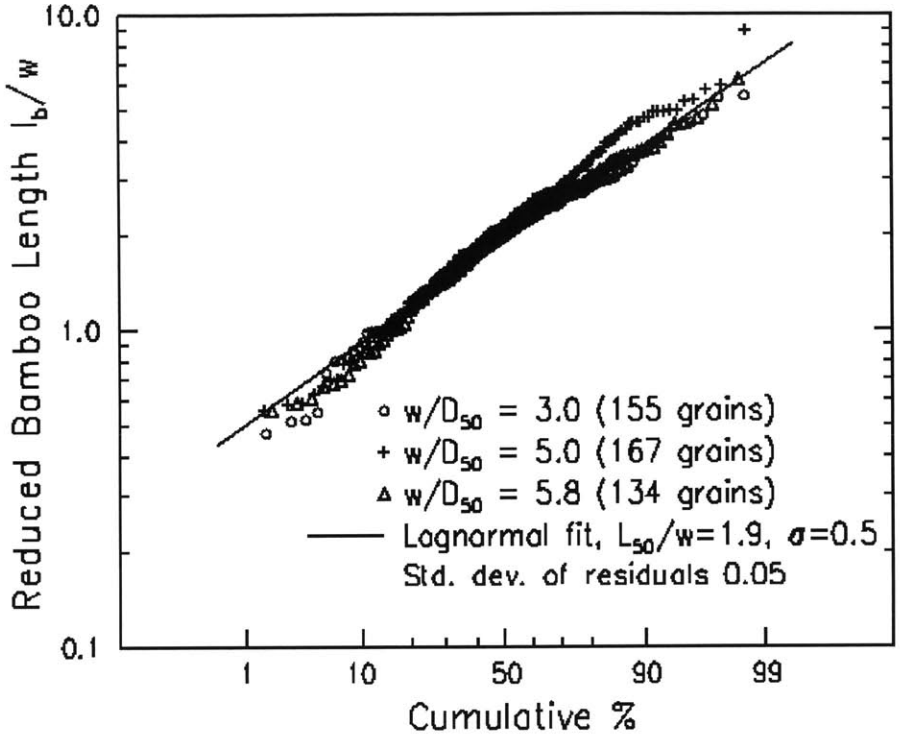


FIG. 4.8: Lognormal plot of normalized bamboo grain length l_b/w distributions in the bamboo structures, resulting from simulated prolonged annealing of polygranular strips with different widths. The overlapping curves for different values of w/D_{50} show that the distribution of the values of l_b/w is line-width-independent. The solid line represents the best-fitting lognormal distribution.

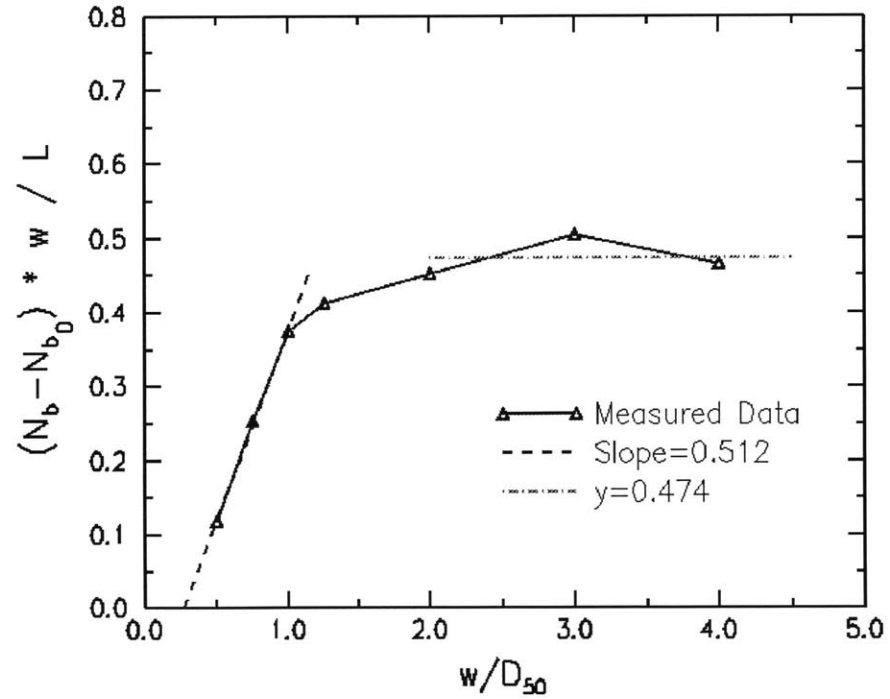


FIG. 4.9: Normalized total number of bamboo nucleation events as a function of w/D_{50}.

4.5 Summary

We have presented a geometry-sensitive analytic model for the evolution of the grain structures of initially polygranular thin film strips to bamboo grain structures. The model is in agreement with a Johnson-Mehl-Avrami analysis of the transformation. The model predicts the line-geometry dependence of the two transformation parameters, the bamboo nucleation rate and the polygranular cluster shrinkage velocity. The model also allows prediction of the time and geometry dependence of the polygranular segment length distribution. For initially near-bamboo lines ($w/D_{50}<2.0$), the model's predictions are still in agreement with the simulation, but the geometry dependence of the nucleation rate and the cluster shrinkage velocity is not the same as for initially polygranular lines. Evolution from the near-bamboo regime can be analyzed using parameters derived from simulations [Fay 98]. With these compact analytic models it is possible to generate appropriately varying grain structures for simulations of interconnect reliability, eliminating the need for time-consuming multiple simulations of grain structure evolution.

In the next chapter, we will use the geometry-sensitive bamboo-interconnect grain structure statistics model described here to investigate the effects of grain-orientation-dependent surface diffusivities on the reliability of bamboo interconnects.

Chapter 5

Modeling Texture Effects on Electromigration-Limited Reliability in Bamboo Interconnects

5.1 Introduction

Post-patterning annealing of polygranular interconnects can result in grain growth leading to purely bamboo structures for which EM lifetimes are governed solely by surface diffusion [Joo 97, Sri 98]. In this case, the variability in EM lifetimes observed in experiments on Al-based or Cu-based interconnects [Hu_2 95, Hu 97] is not well understood, and is presumed to arise from grain-structure-related variations in surface diffusivities.

In the following, we use the 2D grain-growth-simulation-based model, developed in Chapter 4, for the width-dependent bamboo grain length distributions in bamboo structures resulting from post-patterning annealing of polygranular interconnects (see Fig. 4.1), to generate lines with realistic bamboo microstructures. We then simulate the effects of grain-structure-related variations in the surface diffusivities by assuming that surface diffusivity varies with grain orientation in accordance with a Read-Shockley model [Rea 54]. We finally show, using an EM simulation [Par 99], along with published values for surface diffusion in Cu [Cho 62], that a grain-orientation-dependent diffusivity is a likely mechanism for the lifetime variations observed experimentally in Al-based and Cu-based interconnects with bamboo grain structures [Hu 95].

5.2 Bamboo Grain Structure Statistics

Using the 2D simulation of curvature-driven grain growth, GGSim [Fro 88, Fro_2 88, Fro 90], we have simulated the development of bamboo structures in polygranular interconnect strips with different widths. This transformation is depicted in Fig. 4.1. We have shown elsewhere (see Chapter 2 or [Fay 99]) that 2D normal grain growth in thin films leads to a uniquely defined grain structure, evolving in a geometrically and statistically self-similar fashion, with an average grain area increasing linearly with both time and the grain boundary mobility μ (assuming μ is constant and uniform). Under these conditions, the grain structure evolution in lines of width w, etched from films with an average grain area much smaller than w^2, will obey geometric and kinetic scaling (see Chapter 4 or [Wal 92, Fay 00]): the duration of the transformation will scale with w^2/μ and the resulting bamboo grain structure statistics in lines with different widths will only differ due to geometric magnification. In particular, we expect the distribution of the bamboo grain lengths normalized by the line width, (L/w), to be width-independent. Figure 4.8 shows that such a distribution is well fit by a lognormal distribution function. When plotting ln(L/w) as a function of Φ^{-1}(F(L/w)), where F(L/w) is the proportion of grains with normalized length smaller than L/w and Φ is the Gaussian function, data that falls on a straight line fits a lognormal distribution. For the simulated lines with different widths, the curves overlap, as expected with the geometric scaling outlined above. The unique lognormal distribution function that fits these results is characterized by a median value of 1.9 and a lognormal deviation $\sigma = 0.5$. The average bamboo grain length associated with such a distribution is about twice (2.1 times) the line width.

Normal grain growth in 3D systems leads to similar results to the ones observed in the 2D simulation. Fortes et al. investigated the case of the coarsening of soap froths inside cylindrical glass tubes with different diameters [For 98]. They reported a diameter-independent distribution for the bamboo grain length normalized by the tube diameter. The average bamboo length was found to be very close to the tube diameter. In Chapter 7 (see also [Fay_2 00]), we will present similar evidence in the case of tubes

with rectangular cross sections where we found the average bamboo length to be approximately 1.5w (w is the larger side in the cross section).

5.3 Simulation of Texture Effects on Reliability

Annealing-induced grain structure evolution in metal interconnects is not only governed by curvature-driven growth. It also depends on a number of other factors, such as elastic anisotropy [Car 96], variable grain boundary energy [Fro$_2$ 94], surface grooving or other boundary pinning effects [Fro 90, Fro 94]. However, the 2D normal growth simulation and the model to be presented here provide a good first order evaluation of the resulting bamboo grain structure. We can use the compact model discussed above (based on the generation of structures using the fitted lognormal distribution function) or use GGSim directly to generate appropriately varying grain lengths for simulations of electromigration in lines with bamboo structures. Our electromigration simulation tool MIT/EmSim [Par 99] is based on the Korhonen 1D stress evolution model [Kor 93]. Current-driven atomic diffusion creates, at sites of atomic flux divergences where high diffusion segments meet low diffusion segments, an accumulation or depletion of atoms resulting in a compressive or tensile hydrostatic stress, respectively. If the increasing stresses exceed a critical limit, the interconnect fails due to hillock formation or voiding in the metal. Microstructure-induced diffusivity variations are input to this simulator, along with the current density and temperature, to enable the calculation of the atomic fluxes and stress evolution in the interconnect. Times to failure can be calculated as the time to reach a critical stress or critical resistance change.

To simulate the effects of grain-structure-related variations in the surface diffusion, which is the dominant diffusion mechanism in the case of bamboo interconnection [Joo 97, Sri 98], we assumed that the surface diffusivity of Cu varies with grain orientation in accordance with a Read-Shockley model [Rea 54], and fitted published values for surface diffusion on (100) and (111) Cu surfaces [Cho 62], which are expected to be the minimum and the maximum diffusivities, respectively. D_{111} and D_{100} were evaluated

with Arrhenius relationships with activation energies of 2.10 eV and 2.26 eV respectively, and pre-exponential factors of 1.8 m^2/s and 4.0 m^2/s respectively [Cho 62]. This description can also be implemented in MIT/EmSim so that surface diffusivities can be assigned to bamboo grains, assuming a random grain orientation, according to a Read-Shockley function, as depicted in Fig. 5.1.

When simulations of the reliability of bamboo structures were carried out using these models, results such as those shown in Fig. 5.2 were obtained. Figure 5.2 is a lognormal plot of EM-induced failure times in a population of 10 simulated bamboo lines with a width of 1.4 μm and a length of 300 μm. A current density of $2x10^6$ A/cm^2 and a temperature of 392°C were used. The lifetime variations observed in the simulations are very similar to those observed in experiments [Hu 97]. Assuming a critical stress for failure of 300 MPa, we obtained a lognormal deviation in the times to failure of about 0.3. This important result supports the expectation that lifetime variations in lines for which surface diffusion is dominant (e.g. Cu-lines or Al-lines with bamboo structures), are the result of grain-structure related variations in surface diffusivities, and the associated atomic flux divergences.

5.4 Conclusion

We have presented a model for grain length statistics in bamboo interconnects with structures that have evolved from an initially polygranular structure. The average grain length scales with the line width, and the deviation in normalized length is independent of line-width. Assuming grain-orientation-dependent surface diffusivities allowed us to successfully simulate the experimentally observed variations in EM-lifetimes in bamboo Cu interconnects.

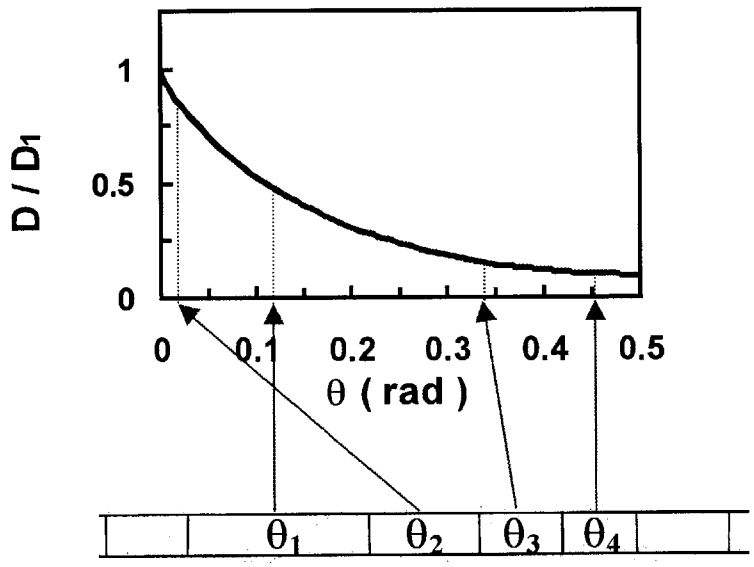


FIG. 5.1: Distribution of surface diffusivities as a function of bamboo grain orientation. The grain orientation is assigned randomly, and the diffusivity for a given orientation θ is given by $D=D_{fac}(1+ A\ \theta\ \ln(\theta) - B\ \theta)$ where $D_{fac} = 2.2 \times 10^{-16}$ m^2/s at 392°C, A = 1.8, and B=0.55.

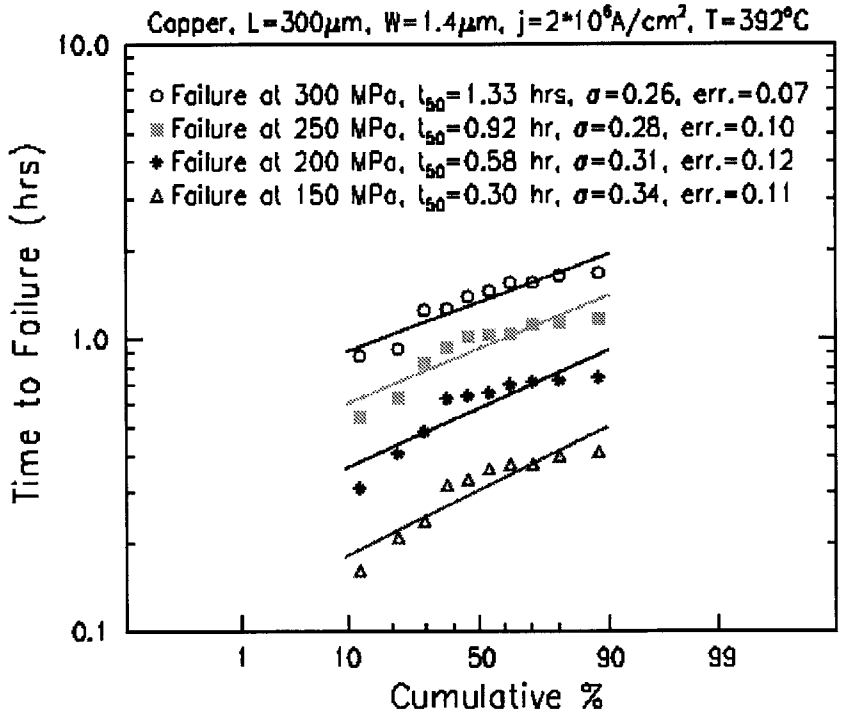


FIG. 5.2: Lognormal plot showing the simulated variations in lifetimes resulting from surface diffusivity variations in lines with bamboo grain structures.

Chapter 6

Process-Sensitive Tools for Interconnect Reliability Prediction

6.1 Introduction

In the previous three chapters, interconnect grain structure was analyzed, leading to the development of process-sensitive geometry-dependent analytic models for microstructure and microstructural evolution in straight, junction-free interconnect lines. However, interconnects can have different geometries and are not always straight lines. In order to generate realistic structures, we have extended GGSim, the grain growth simulation program discussed in [Fro 88, Fro 90, Wal 91, Wal$_2$ 91, Wal 92, Fay 97, Fay 99] by developing a *pattern* program capable of simulating the etching of interconnect structures with general geometrical configurations, as well as further annealing-induced grain structure evolution. This powerful simulation capability will be presented in the next section along with a number of direct applications.

In section 6.3, we will show how the improved grain growth simulation capabilities can be used in conjunction with electromigration simulations to determine the reliability of interconnect trees and wide interconnect lines with fully-polygranular structures, in addition to the reliability of near-bamboo straight-line interconnects.

In section 6.4, we present a tool, EmSimGen, that generates process and geometry-sensitive interconnect structures for electromigration simulations, based on the analytic models developed in chapters 3, 4 and 5.

6.2 PATTERN: A Tool for Generation of Interconnects with General Geometries

The grain-growth simulation program GGSim has been briefly discussed in previous chapters. We outline here the main pre-existing components and functions, and present the additions brought to it in the context of this thesis research. For a more complete review, we refer the reader to [Fro 88, Fro 90, Wal 91, Wal$_2$ 91, Wal 92, Fay 97, Fay 99].

The simulation package is made up of six computer programs with approximately 20,000 lines of C code. The programs *nucleate, impinge,* and *init* generate the nucleation and growth to impingement structures used to initiate grain growth simulations. The patterning of continuous film structures into thin-film strips is performed by *etch*. The patterning of continuous film structures into more generally shaped patterns is handled by the new routine *pattern*. The program *anneal* simulates grain growth in both continuous films and thin film strips and has been extended through this research to handle grain growth in patterned structures with arbitrary shapes.

The *nucleate* and *impinge* programs generate the Johnson-Mehl [Joh 39] continuous nucleation structure. This structure results from the nucleation and growth-to-impingement of grains on a two-dimensional surface. Here, grain nucleation (at random sites) continues at the same time that the growth of previously nucleated grains occurs. In other words, grain nucleation occurs continuously throughout the growth-to-impingement process. Throughout this process, the probability of grain nucleation within a unit of untransformed area remains constant. In the fully impinged structure, each grain boundary is a segment of a hyperbola. We generate the Johnson-Mehl structure via a three-step process. First, *nucleate* establishes a collection of nucleation sites, shown as

dots in Fig. 6.1. The program *impinge* then determines the positions of the grain-boundary triple junctions and specifies arrays of points to represent the grain boundaries. We refer to the points describing the grain-boundary triple junctions as triple points. The points representing the grain boundaries are called segment points. A third program, *init*, is then used to convert the bare-bones representation of the grain structure generated by *nucleate* and *impinge* into the linked data structure recognized by *etch*, *pattern*, and *anneal*. Periodic boundary conditions are used within all of the five previously developed programs in the simulation package. The Johnson-Mehl structure of Fig. 6.1 clearly demonstrates these boundary conditions. Here, grains located at the array's lower edge are duplicated at the upper edge, those at the left edge are also found at the right edge, etc. The resulting 2D grain structure may be envisioned as lying on the surface of a torus.

The *etch* program is designed to convert an entire continuous-film grain structure into a single strip structure [Wal 91]. Figure 6.2 shows an example of lines used to perform this task. Each line in the figure represents both the top edge of the lower strip slice and the bottom edge of the upper slice. During the patterning process, a grain boundary which traverses the etch line is broken into two segments. Each of the two segments is terminated at a new triple point inserted on the appropriate strip edge. Notice that due to periodic boundary conditions, the strip actually wraps back on itself, forming a closed loop, or band. To operate the *etch* program, one specifies the input structure and the desired number of slices. The input structure may either be the nucleation and growth-to-impingement structure, created by the sequence *nucleate-impinge-init*, or a structure resulting from a period of continuous-film grain growth, as produced by *anneal*. The width of the strip, w, is determined by the number of slices n: $w = a/\sqrt{n^2+1}$, where a is the side length of the square simulation array. Also, since the total area of the film is conserved, the total strip length l is $l = a^2/w = a\sqrt{n^2+1}$

The *pattern* program, which I have written and incorporated into the simulation package, provides the capability of patterning the film with any general shape polygon

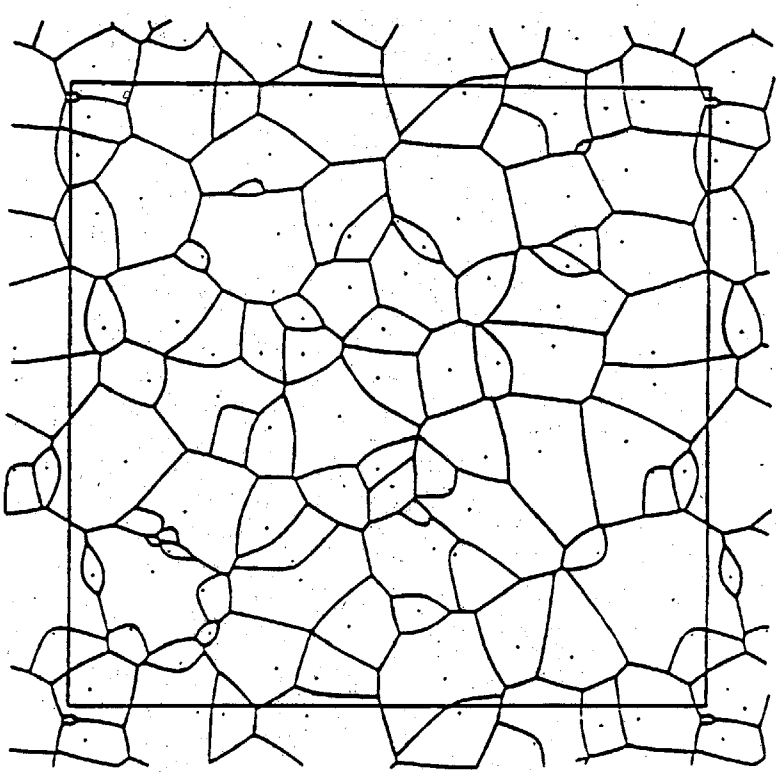


FIG. 6.1: The Johnson-Mehl structure generated by the programs *nucleate* and *impinge*. The dots represent grain nucleation sites [Wal 91] .

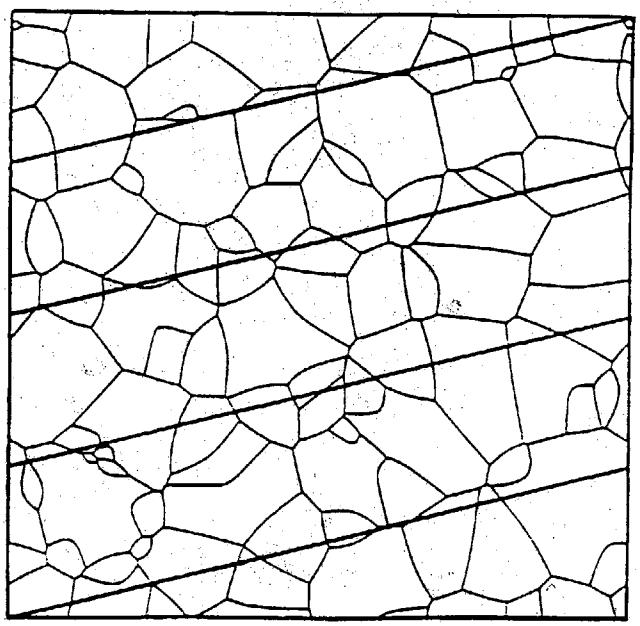


FIG. 6.2: Patterning lines used by the *etch* program. These patterning lines will convert the continuous film into a single, thin-film band [Wal 91].

(including non-convex polygons) [Fay 97]. A direct generalization of *etch* to achieve this goal was impossible because *etch* uses the periodic boundary conditions of the film and also produces a stripped structure with the same periodic boundary conditions. Here, as in real interconnect designs, the pattern is general and does not have any periodic boundary conditions, and therefore the periodic structure of the film cannot be used to simplify the patterning algorithm. Figure 6.3 shows the result of the patterning of a film with two commonly found interconnect elements.

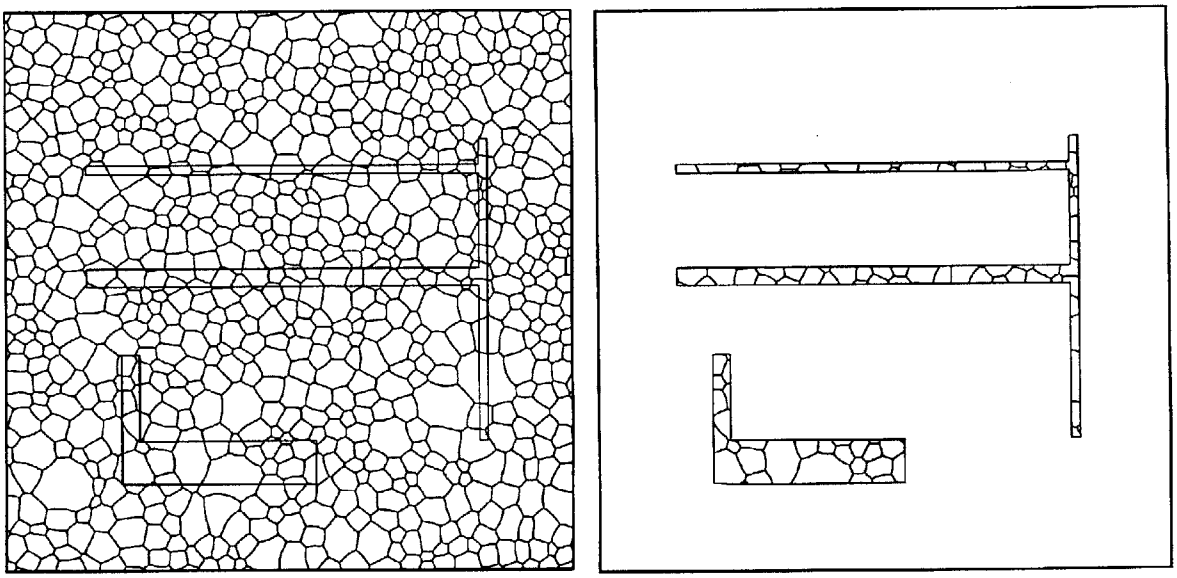


FIG. 6.3: The patterning of 2 different interconnect structures from the same film. In **(a)** the total structure is kept after patterning and post-patterning evolution can be simulated in the internal structures as well as the external one. In **(b)**, evolution of only the internal structures is simulated.

The program begins by finding all the intersection points between grain boundaries and the edges of the polygon to be patterned. These intersection points are then sorted on the edges of the pattern in a counter-clock-wise order. The process of incorporating these new points as well as the points representing the corners of the pattern as part of the linked data structure then begins. It involves starting at one intersection point and walking from one intersection point to the following in a counter-clockwise way on the polygon edges, making the new edge triple point incorporations, creating new grains due

to the splitting by the edge, and updating the links of the data-structure. Eventually, one returns to the intersection point of departure, having transformed the continuous film into two structures, the inner pattern, and the outer one (which also could be an interesting feature to analyze). In fact, one reason for keeping the outer structure is to make only the minimum necessary changes in the initial data structure. In addition, the program allows patterning from the same film, one or more interconnect features, as can be seen in Fig. 6.3. The simulation package represents the grain structure in such a way that each boundary point and triple point is linked to points which are its nearest neighbors. Using this linking information, it is possible to move from one point to its neighboring point, then to a point neighboring this second point, etc. With this process, one can walk along the linked grain boundary network. In Fig. 6.4, we show a close-up view of the patterning process.

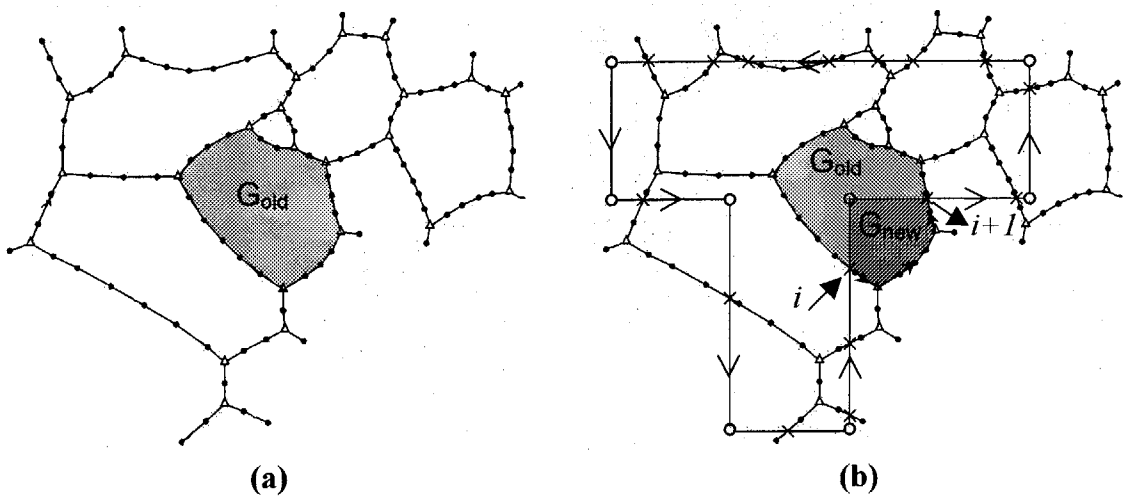


FIG. 6.4: Schematic illustration of the patterning algorithm in the program *pattern*. **(a)** continuous structure; **(b)** patterned structure. For each intersection point with the edge, two new triple points are inserted and linked into the data structure and a new grain is created. The corners of the pattern are inserted as stagnant segment points.

At an intersection point (i) between the grain boundary and the edge, two new edge triple points are inserted, one internal (as part of the inner structure), and one external (as part of the outer structure). As part of the insertion process, the links between boundary

points straddling the etch line are broken, and then reformed with the new appropriate edge triple point. In addition, the newly created edge triple points are linked to the two previous ones, thereby defining segments of the pattern side-wall. After this is done, the program walks around the grains in a counter-clock-wise way until it reaches the next intersection point (i+1). The grain split in two by the edge portion [i, i+1] is identified and a new label for the created external grain is inserted (the internal grain keeps its original label). Grain neighbors are also updated in a consistent way. The corners of the pattern are inserted in the data structure as new stagnant segment points. The insertion process is continued in this manner until the program returns to the starting point of the process. The particular case of a lense, which is a grain with only two grain boundaries, presents more complicated technical issues because of the nature of the data structure itself. This case has been dealt with, but since its interest is purely technical, it has been omitted here, along with many other technical details.

The pre-existing *anneal* program models 2D grain growth in both continuous films and thin film strips resulting from *etch*. After nucleation and growth-to-impingement, a thin film may evolve through 2D grain growth. Grain growth is a thermally activated process through which the average grain size of a material increases. In the case of thin films, the driving forces for such a phenomenon are, primarily, capillarity (grain boundary energy), surface/substrate interface energies anisotropy and strain energy density anisotropy. In the case of idealized, capillarity-driven grain growth, grain boundaries are assumed to move at a velocity, v, proportional to their local curvature κ, according to $v = \mu \kappa$, where μ is a mobility constant. The boundaries are also assumed to meet at triple junctions to form 120° angles, consistent with the assumption that all boundaries have uniform energy. In the simulation, each boundary is, as we have seen, represented by an array of points. Grain growth is simulated by alternately moving the points describing the grain boundaries and repositioning the grain boundary triple junctions. However, idealized 2D grain growth is rarely observed experimentally. Instead, grain growth is found to stagnate when the average grain size approaches 2 or 3 times the thickness of the film [Bec 49, Pal 87]. Mullins [Mul 58] has proposed that this stagnation is due to grain boundary grooving at the intersections of the grain boundaries

with the film surfaces. Under certain conditions, these grooves will trap or pin migrating boundaries. This effect has been included in the existing grain growth simulator by assuming that boundaries with curvature below a certain critical curvature, κ_{crit}, are unable to migrate. This procedure has permitted us to generate fully stagnant structures which closely resemble those observed experimentally. These stagnant grain structures have a lognormal grain size distribution with an average grain size proportional to the inverse of the critical curvature.

Similar considerations of grain boundary energy effects and grain free-surface-energy-induced grooving effects, have permitted the modeling of grain growth in etched infinite lines obtained with the *etch* program. In idealized 2D grain growth, grain boundaries meet the side-wall of a strip at an angle of 90°. Grooving at the strip side-walls will cause boundaries to meet at angles slightly different from 90°. Modeling the motion of grain boundaries at the strip-walls involves the definition of two critical angles between the grain boundary and the normal to the side-wall [Wal 91], the static-groove critical angle θ_0, which determines if a static grain boundary has enough energy to start moving ($\theta>\theta_0$), and the dynamic-groove critical angle θ_c, which is smaller than θ_0, and which determines if a boundary with steady state motion will continue to move ($\theta>\theta_c$).

The grain boundary motion, handled by the simulation program in the routine *anneal*, only simulates grain growth in semi-infinite strips with periodic boundary conditions. This is a limiting factor when one needs to assess the effects of post-patterning annealing of interconnect elements with arbitrary shapes. We have completed the necessary modifications to the existing program, in order to successfully model grain growth in these cases of great practical importance. A substantial modification concerns the state of an edge triple point approaching a corner of the patterned structure. Such a situation is not encountered in the case of infinite strips with periodic boundary conditions. Modeling this effect has been done by repositioning the edge triple point on the subsequent edge as soon as the driving force it is subjected to is enough to get it to the corner. This is depicted in Fig. 6.5.

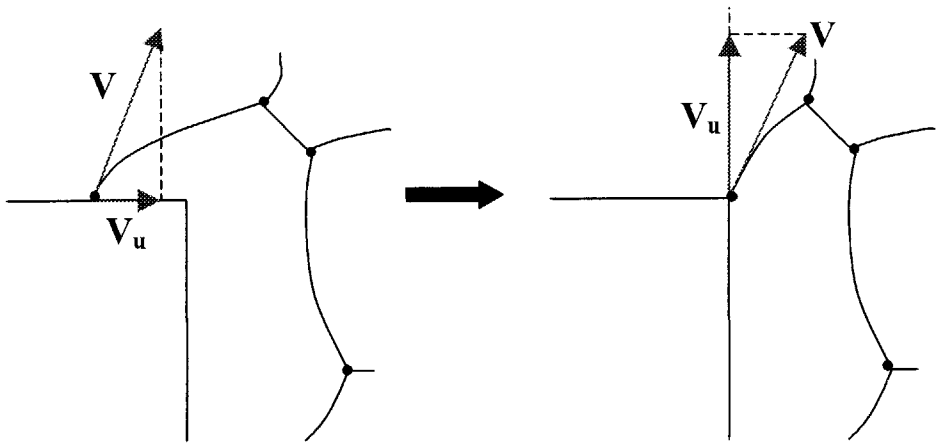
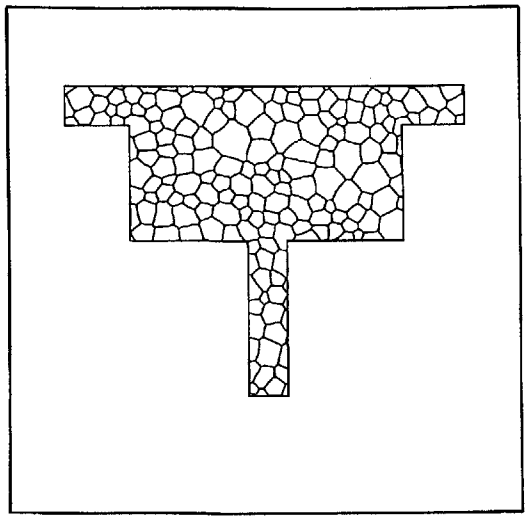


FIG. 6.5: The case in which a grain boundary is close to one of the pattern's corners. If the driving force is enough for the grain boundary intersecting the edge to hit the corner in the next time step, the grain boundary intersection point is relocated at the corner, and linked to the consecutive edge. The intersection of the boundary with the corner may thus be stabilized.

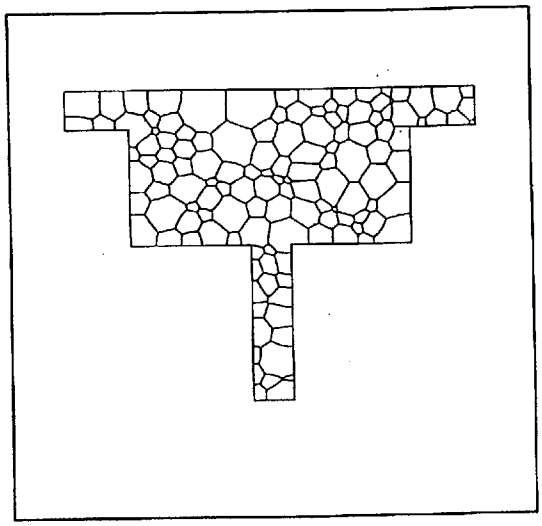
Figure 6.6 shows the results obtained on a complex tree structure. The simulation was done with the critical curvature set to zero. The modifications to the grain growth simulation package described in this section are listed in Appendix B.

We have tested the validity of our program by simulating the evolution within one of the simplest geometries, the *square*. As depicted in Fig. 6.7, grain growth within the square eventually leads to a stagnant structure which may or not contain any triple point junctions. We can show that for a given square size, there is a value of the critical radius defining stagnation above which we wouldn't expect to observe any triple junctions within the square when stagnation is reached. Assuming boundaries are arcs of circles, it can be easily shown that this value of the critical radius is precisely the square side. Figure 6.8 plots the proportion of polygranular squares (squares with at least one internal triple junction) at stagnation as a function of the product of the critical curvature by the edge length in populations of 10 squares. This figure shows that the simulation confirms the expectations, in a statistical sense.

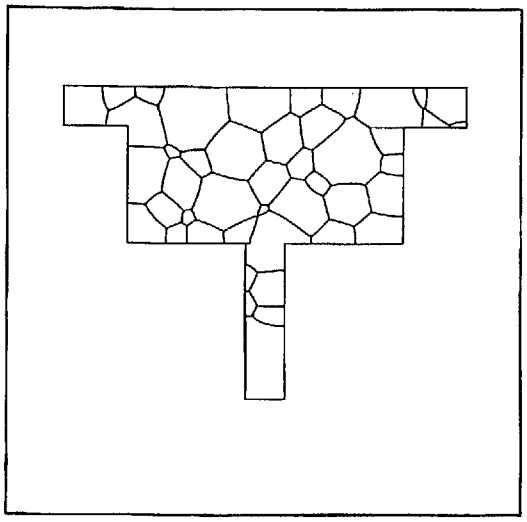
Recently Hau et al. have used the newly developed simulation capability to simulate the effects of scanned laser annealing on grain structure evolution in rectangular interconnects [Hau 99].



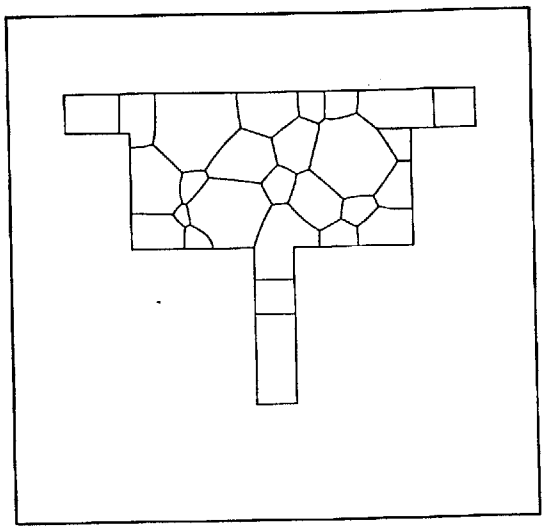
$\tau = 19.3$



$\tau = 40.0$



$\tau = 55.0$



$\tau = 70.0$

FIG. 6.6: Post-patterning annealing induced grain structure evolution in a complex tree structure. In this particular case, the critical curvature defining the grain boundary stagnation condition was set to 0. τ is the normalized simulation time.

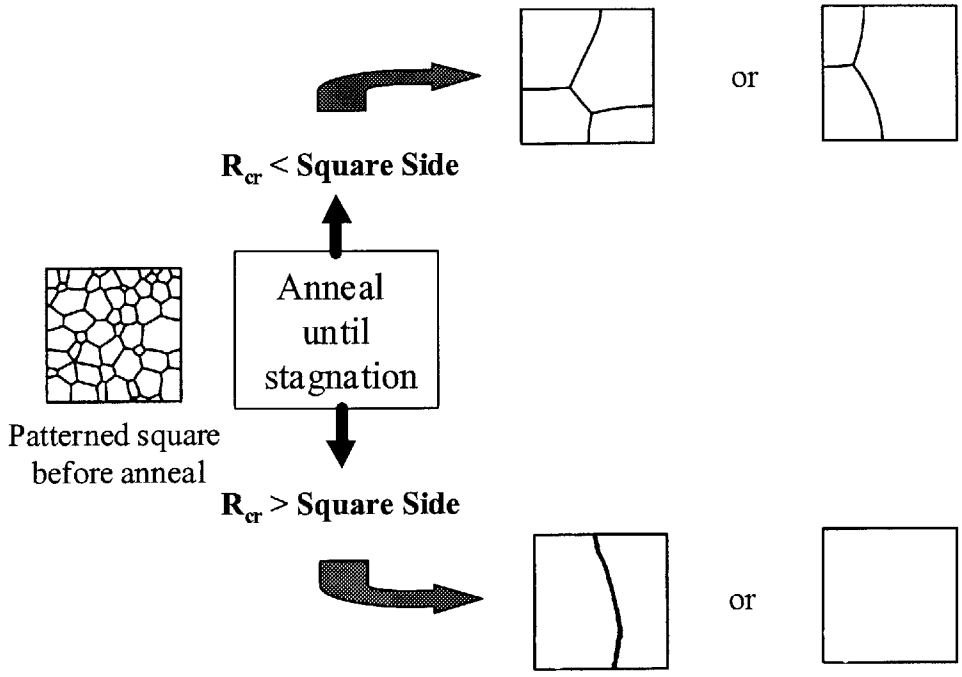


FIG. 6.7: Structural characterization of post-patterned annealed square structures. When the square side is larger than the critical radius defining grain growth stagnation, squares are likely to have a polygranular grain structure at stagnation. As the square side decreases, the probability of evolving to a state with no triple junctions becomes larger, and becomes almost certain when the square side is smaller than the critical radius for stagnation.

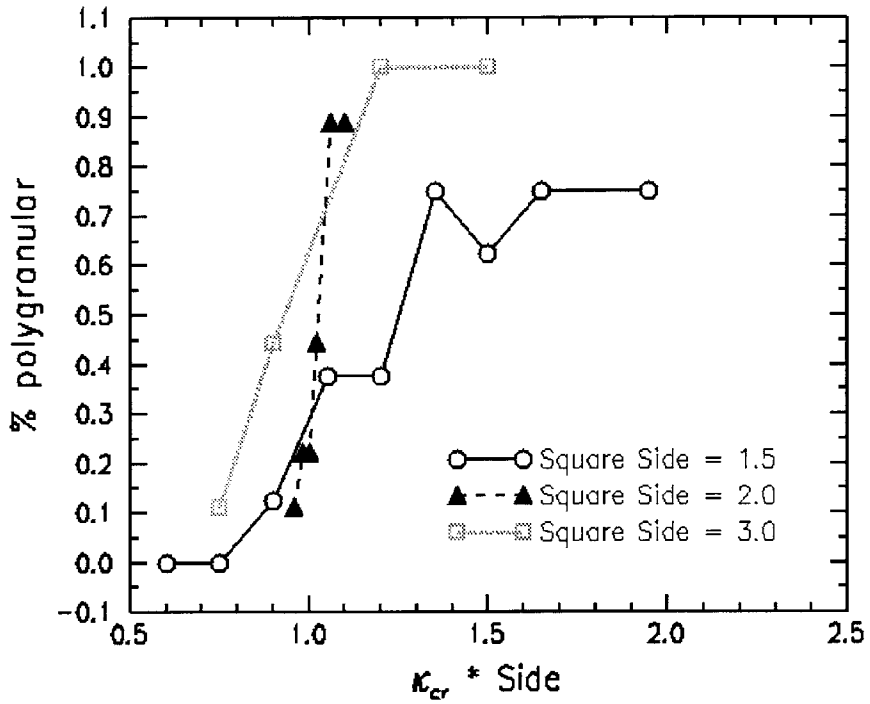


FIG. 6.8: Microstructures of squares after grain growth stagnation, as a function of the product of the square side length and the critical curvature defining stagnation.

They were able to successfully reproduce experimental results obtained for Al interconnects, and demonstrated that this novel procedure can lead to bamboo structures with grains much longer than the ones obtained through conventional scanning techniques.

6.3 PolySeg: A Tool for Generation of Process and Geometry-Sensitive Interconnect Microstructures Based on Grain Growth Simulations

Grain growth simulations provide efficient means of producing properly statistically varying realistic microstructures for electromigration simulations and interconnect reliability predictions. Electromigration simulations, such as MIT/EmSim [Par 99], allow, through stress calculations, the prediction of interconnect lifetimes at service conditions as well as testing conditions. More specifically, the one-dimensional stress evolution model given by equations (1.4), (1.5), and (1.6), is used to calculate the stress profile along the interconnect length and its evolution with time. A crucial requirement for this calculation is the detail of the transport mechanism, or, in other terms, the diffusivity profile along the interconnect length.

In previous work, Knowlton et al. treated grain structure effects by assigning one effective diffusivity to regions composed of polygranular clusters with grain boundary diffusion paths along the length of the line, and another orders of magnitude lower effective diffusivity was assigned to regions of the line composed of bamboo grains without high-diffusivity grain boundary paths [Kno 97]. This allowed simulation of the effects of grain structure statistics on lifetime statistics for lines with near-bamboo grain structures, as a function of line length and width [Kno 97]. However, this does not allow a statistical treatment of lifetimes for wide lines with continuous grain boundary paths along the interconnect length.

To treat the statistical effects of continuously connected grain structures (as found in lines with widths significantly greater than the average grain size), we have modified

MIT/EmSim and its grain structure input file to treat effective diffusivities of any value, which vary continuously along the length of the lines. The magnitude of the effective diffusivity as a function of position along the length of the line can be set by summing the effects of diffusion along all the available paths along the line length. This can include the effective diffusivities of multiple grain boundaries as well as the effects of surface/interface diffusion, leading to the following effective diffusivity D(x):

$$D(x) \approx D_{gb} \frac{\delta}{w}\left(\sum_{gb's} \cos^2 \theta(x)\right) + \frac{2\delta}{w} D_i + D_L, \tag{6.1}$$

where D_{gb}, D_i, and D_L refer to grain boundary, interface, and lattice diffusivities, respectively. $\theta(x)$ is the inclination of a grain boundary with respect to the direction of the line (or current), assuming a 2D structure. Application of this approach to a 2D grain structure obtained using our grain growth simulator GGSim is shown in Fig. 6.9 for a near-bamboo line.

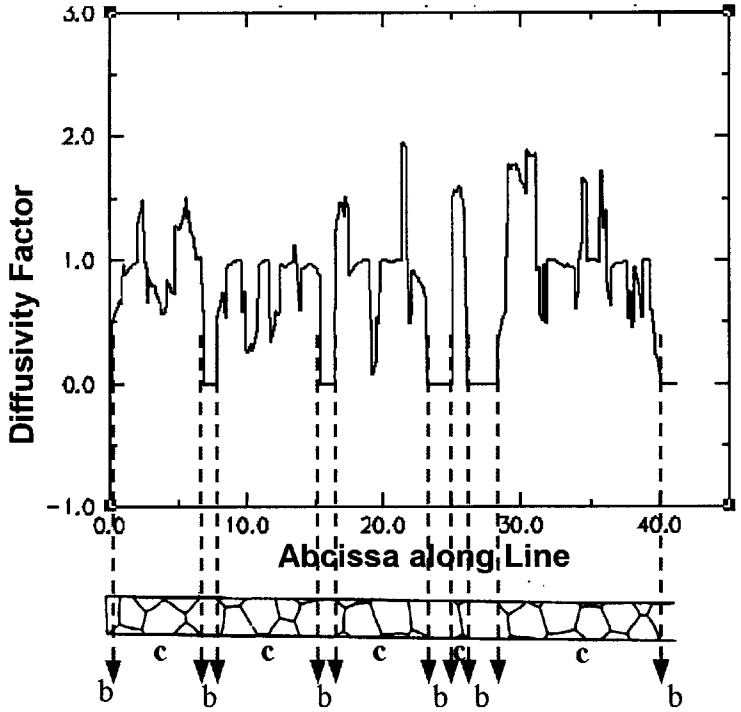


FIG. 6.9: Determination of the effective diffusivity as a function of position, for simulation of the reliability of wide polycrystalline and near-bamboo interconnect lines. The letters b and c denote bamboo and polygranular regions, respectively.

Fig. 6.9 shows in particular that while the variation in diffusivity within a polygranular region of the line is within the same order of magnitude, the difference in diffusivity between polygranular regions and bamboo regions is several orders of magnitude. This is consistent with Knowlton's treatment, and shows that his approximation is well justified in the case of near-bamboo lines. In wide polygranular lines however, it is the variation in diffusivity within the polygranular structure that gives rise to the grain-structure-induced statistical variations in electromigration lifetimes. Through the treatment of interconnect microstructures discussed above, we were able to simulate line-width effects on the statistics of reliability of wider lines, and reproduce the well known variations in the median time to failure as a function of line width [Vai 80, Vai 81], or line-width-to-grain-size ratio [Cho 89] (see Fig. 3.1), as shown in Fig. 6.10. This approach can be readily extended to treat variable grain-boundary diffusivities and grain-orientation-related variations in surface diffusivities. It can also be extended to treat 3D grain structures.

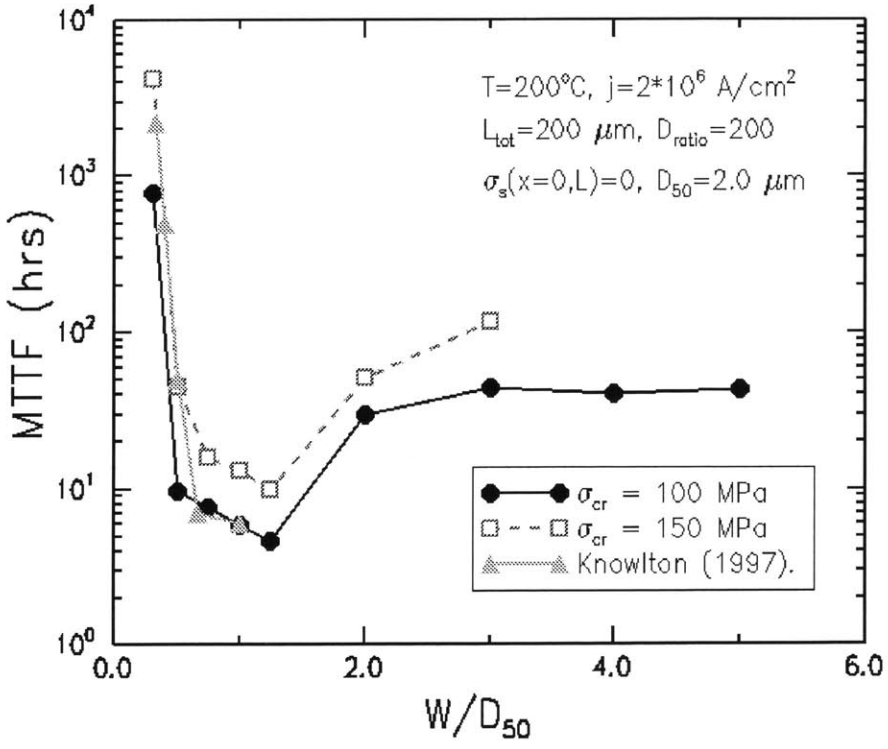


FIG. 6.10: Simulated median times to failure (MTTF) as a function of line-width-to-median-grain-size ratio w/D_{50}.

PolySeg is the name we chose for the microstructure extraction feature of GGSim. Appendix C lists a copy of the C programs that make up PolySeg, which include three main sub-routines. The sub-routine *polyseg* (for simplification, the whole extraction program shares the name of this main subroutine) assesses the line microstructure in terms of polygranular segments with a high, uniform diffusivity and bamboo segments with a smaller diffusivity, as has been done in the past [Kno 97]. The sub-routine *diff_info* extracts the line's microstructure following the technique outlined above and in accordance with (6.1). Finally, *polyseg_tree*, is a program that allows the same grain structure interpretation for a tree-interconnect as *polyseg* allows for lines. Tree-interconnects, which are the fundamental reliability units of an IC, are interconnect units composed of two or more metal strands of different widths and lengths in electrical contact with each other on a single level of metallization. Generation of such interconnects, and simulation of their annealing-induced grain structure evolution are accomplished in GGSim by using the newly developed program *pattern* and the routine *anneal*, extended in the context of this thesis specifically for that objective.

It becomes possible under these circumstances, to simulate the reliability of interconnect trees as well as interconnect lines. The technique is based on the use of three computational tools: ERNI, GGSim, and MIT/EmSim [Demo 98]. ERNI is a circuit-level reliability analysis tool which is used along with a circuit layout tool (MAJIC) to subdivide metal levels into interconnect trees. These trees are binned according to predetermined categories: feature size, current density, and direction of current for example. Based on this information, and given the processing conditions used in fabricating these interconnects, GGSim can, in its recent version, simulate the process of grain growth as a function of materials, stress, boundary mobility, and thermal history. The microstructure at service conditions (or other conditions), is then assessed using PolySeg, which provides the input to MIT/EmSim. In turn, MIT/EmSim simulates the electromigration-induced stress evolution, given the initial conditions, boundary conditions and transport profiles. Failure times can be extracted according to a number of failure mechanisms [Par 99] and interconnect reliability is predicted by simulating large populations with statistically varying microstructures.

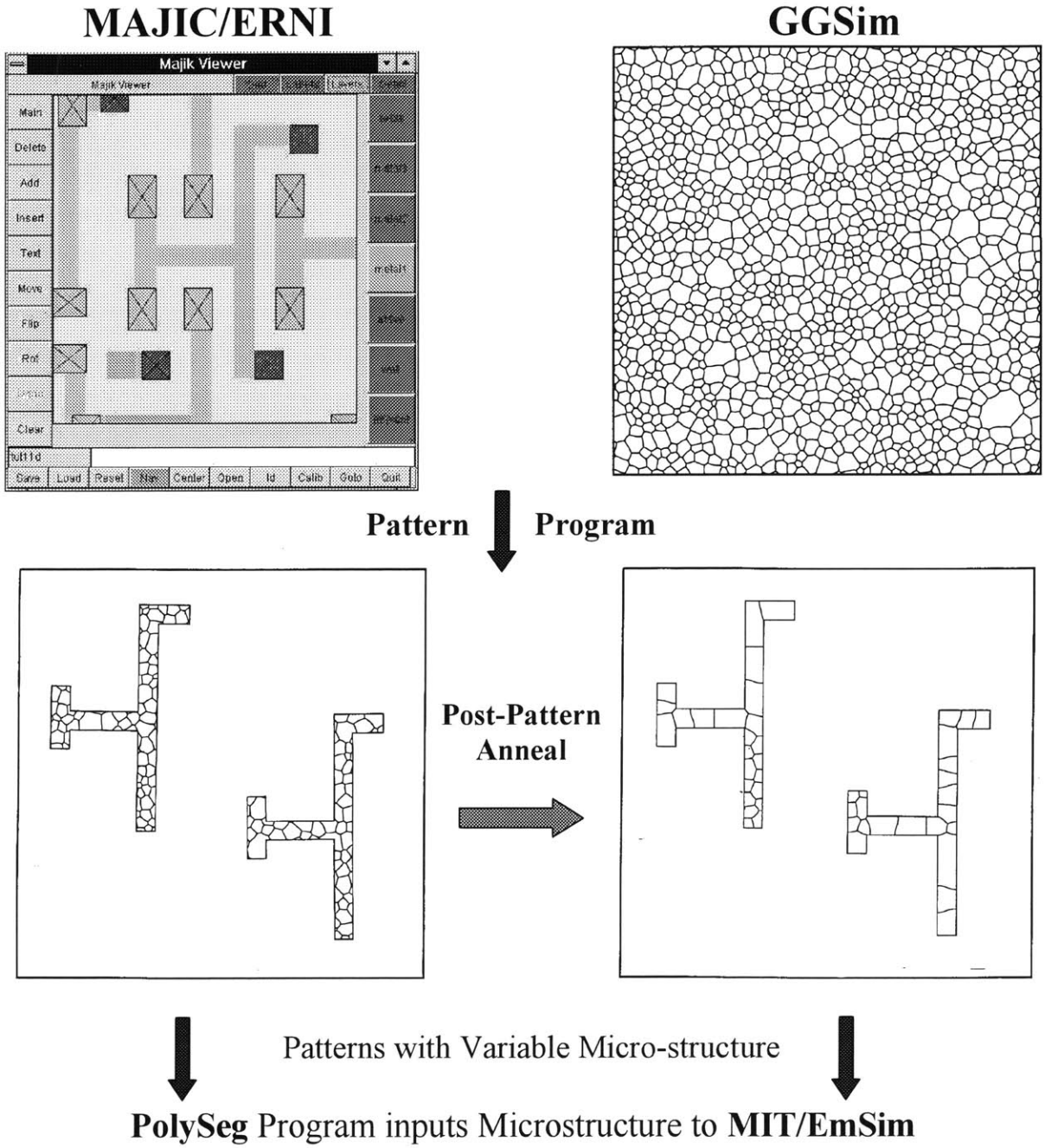


FIG. 6.11: Generation of interconnect structures for electromigration simulations and interconnect reliability estimation: ERNI extracts geometry and current information, GGSim provides realistic thin film microstructures, and *pattern* is used to etch populations of interconnects with the geometry of interest and appropriately variable microstructures. Effects of post-patterning annealing can also be simulated using the modified *anneal*. Finally, PolySeg is used to input microstructures into the electromigration simulator MIT/EmSim for failure time predictions and reliability estimations.

Figure 6.11 depicts the full process for a specific interconnect unit with four strands and 3 junctions, and Fig. 6.12 is a lognormal plot of the failure times obtained in a population

of ten such units. These units were generated by etching a film with a lognormal grain size distribution (consistent with experimental observations) at different places, and therefore have appropriately varying microstructures. Failure times are plotted for the as-patterned interconnects, as well as the stagnant structures obtained with a prolonged post-pattern-annealing sequence.

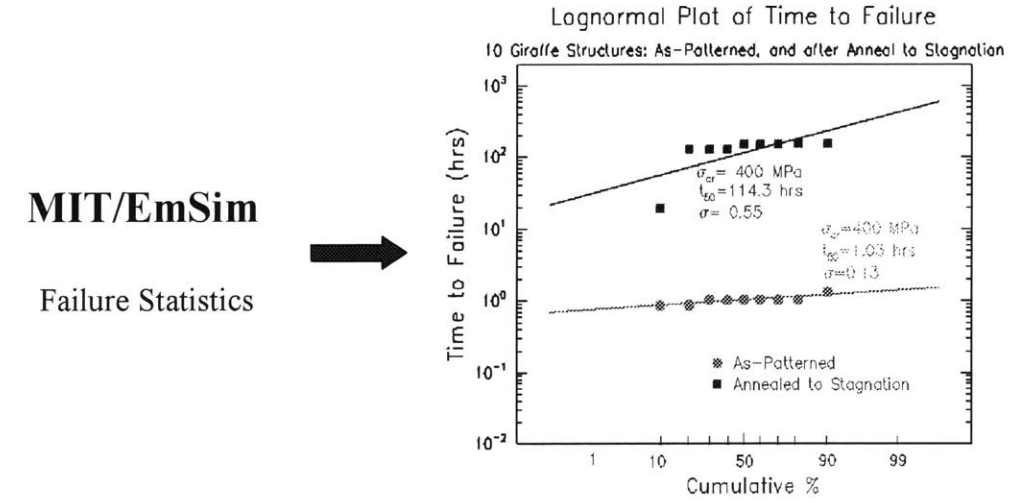


FIG. 6.12: MIT/EmSim is used to evaluate the failure times in a population of ten interconnect structures with the same geometry, both before and after a post-pattern anneal to stagnation. The interconnect microstructures were generated using *pattern* and *post-pattern-annealing* as indicated in Fig. 6.11.

6.4 EmSimGen: A Tool for Generation of Process and Geometry-Sensitive Interconnect Microstructures Based on Analytic Models

Using grain growth simulations allows generation of large populations of interconnect structures suitable for electromigration reliability estimations. Such simulations are clearly less costly and less time-consuming than traditional experimental testing procedures. However, these simulations still require significant computing power and the generation of very large populations is limited by both time and the computer storage capacity. In chapters 3, 4, and 5, we have used grain growth simulations and experiments to model the microstructures of linear interconnects, with patterning and after post-patterning-annealing. The compact analytic models developed in chapters 3, 4

and 5 can be used for generating very large interconnect populations with statistically varying grain structures which depend on the line features and the processing conditions. A sample of results of this methodology is shown, in the case of as-patterned lines in figure 3.11, and show the consistency of the model with the grain growth simulation. The program EmSimGen, which generates microstructures suitable for input for MIT/EmSim, based on the analytic models, is listed in Appendix D. It features the capacity to generate populations of linear-interconnects with varying grain structures in the cases mentioned above and analyzed in chapter 3 (as-patterned interconnects), chapter 4 (post-pattern annealed interconnects), and chapter 5 (bamboo interconnects), along with the case of 3D linear interconnects that will be discussed in detail in Chapter 7.

6.5 Conclusions

Be it through simulation techniques, or through the development of compact analytic models, we are now able to efficiently generate large populations of realistic grain structures for interconnects with arbitrary geometries, and processed under arbitrary conditions. When used with electromigration simulation techniques, and circuit extraction tools, this procedure allows circuit-level and process-sensitive reliability estimations during the IC design and layout processes, ultimately leading to more reliable, better-performing and less-costly circuits.

Chapter 7

Normal 3D Grain Growth in Rectangular Prisms

7.1 Introduction

In Chapter 4, the processing-dependent grain structure of interconnects was analyzed and successfully modeled, assuming normal 2D grain growth in the interconnect strip. The treatment is, however, limited to the case for which the line's aspect ratio (the ratio of the line width w to the line thickness h) is much smaller than 1.0, which justifies the 2D approximation. In the general case, grain structure evolution in the interconnect is three-dimensional. The grain structure evolution can still, however, be analyzed in terms of bamboo grains nucleating within polygranular segments. A polygranular structure can be three-dimensional, with a continuous triple line (line which segments are the intersection of three grains) inside the volume (see Fig. 7.1 (a)), or columnar (2D) with through-thickness, vertical grain boundaries (see Fig. 7.1 (b)). Interconnects can have fully-polygranular structures for which there are continuous grain boundary paths along the length of the interconnect (see Fig. 7.1 (a) and (b) as well as first snapshot in Fig. 7.2). Post-patterning annealing can lead to grain growth that results in bamboo structures for which all grain boundary planes are normal to the interconnect length (see Fig. 7.1 (c)). At intermediate stages, interconnects can have near-bamboo structures for which polygranular clusters with grain boundaries along the interconnect length are separated by one or more grains which span the width and thickness of the line (see Fig. 7.2).

Lengths of line in which one or more neighboring grains span the line width and thickness constitute bamboo segments.

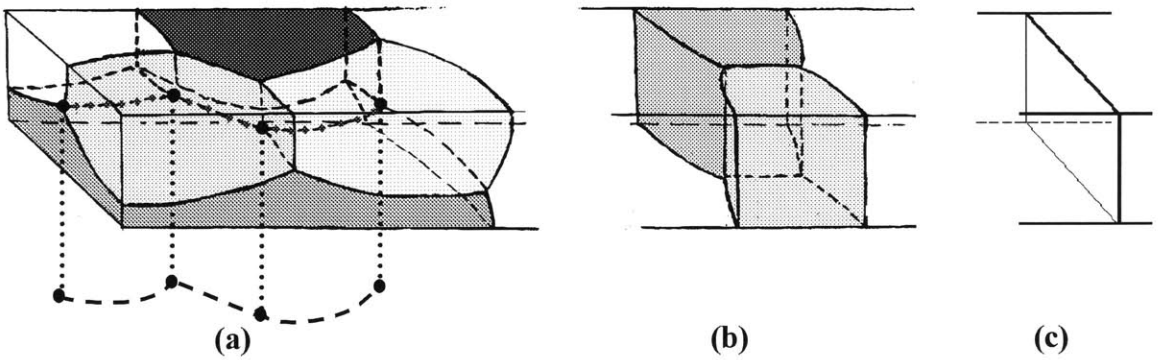


FIG 7.1: Grain or cell structure in a volume of a 3D prism. **(a)** a 3D grain or cell structure, with a continuous triple line (intersection of three grains) in the interior of the volume; **(b)** a 2D structure, with columnar grain boundaries; **(c)** a 1D or bamboo structure.

Aside from simulations, froth experiments have traditionally provided an efficient and relatively simple means of studying the evolution of cellular structures as an analog to grain growth [Gla 92, Sta 93]. The evolution of 2D froth structures is driven by the minimization of boundary curvature as in the case of ideal 2D grain growth in thin films, and results from experiments on froth have been shown to be in detailed agreement with simulations of grain growth [Gla 92, Fay 99]. In this chapter, we will describe an experiment in which we have observed the evolution of soap froths to model 3D grain growth in long rectangular prisms. A conceptually similar experiment was carried out by Fortes et al. on long cylindrical tubes [For 98]. The characteristics of froth evolution observed in the case of cylinders, which we also studied, are significantly different from those observed for rectangular prisms in a number of ways, as will be demonstrated and discussed later. In another experiment, Rosa et al. studied 2D soap froth evolution in tori with rectangular cross-sections [Ros 98]. As will also be discussed in this chapter, our results are in agreement with those of Rosa and Fortes *after* 3D cell structures evolve to 2D cell structures in prisms with significantly different widths and thicknesses.

The goal of this chapter is to determine the geometry-sensitive kinetics of the transformation from polygranular to bamboo structures through 3D cellular evolution, and by analogy, grain growth in rectangular prisms in cases for which the aspect ratio or the cross-section width-to-thickness ratio w/h, is close to as well as larger than 1, and to develop an analytic model for the evolution from polygranular to bamboo structures. Such compact analytic models are needed for simulations of the effects of processing on the rates and statistics of electromigration-induced damage in interconnects.

7.2 Experimental Procedures

A soap solution (5% by volume Palmolive brand liquid detergent, 5% by volume Glycerin and the rest water) was colored for better visibility with a few drops of ink and injected into prismatic tubes with rectangular cross-sections to create fully-polygranular networks of grains (see the first snapshot in Fig. 7.2). The tubes were sealed after having added an absorbing piece of paper towel at each extremity. The tubes were drained continuously (every 2 to 3 hours) ensuring a permanently "dry" network with thin Plateau borders [Sta 90]. Two populations of 17 tubes with lengths of 42 cm and one rectangular cross-sectional dimension of 0.75 cm (h) were used, one with a width in the cross-section $w = 1.5$ h, and one with $w = 1.0$ h. The initial grain structures were fully polygranular with an average grain diameter d_0 such that $h/d_0 = 2.3 \pm 0.1$. The time evolution of the froths was followed by regularly (every 2 to 3 hours) making photocopies of the four different faces of the prismatic tubes.

7.3 Experimental Results

Figure 7.2 shows the evolution on two adjacent faces for a tube for which w/h=1.5. The evolution in the volume can be re-constructed by following and correlating the evolution on these two 2D-faces, which is similar to what is observed in 2D strips undergoing normal grain growth [Wal 91, Wal 92, Fay 00].

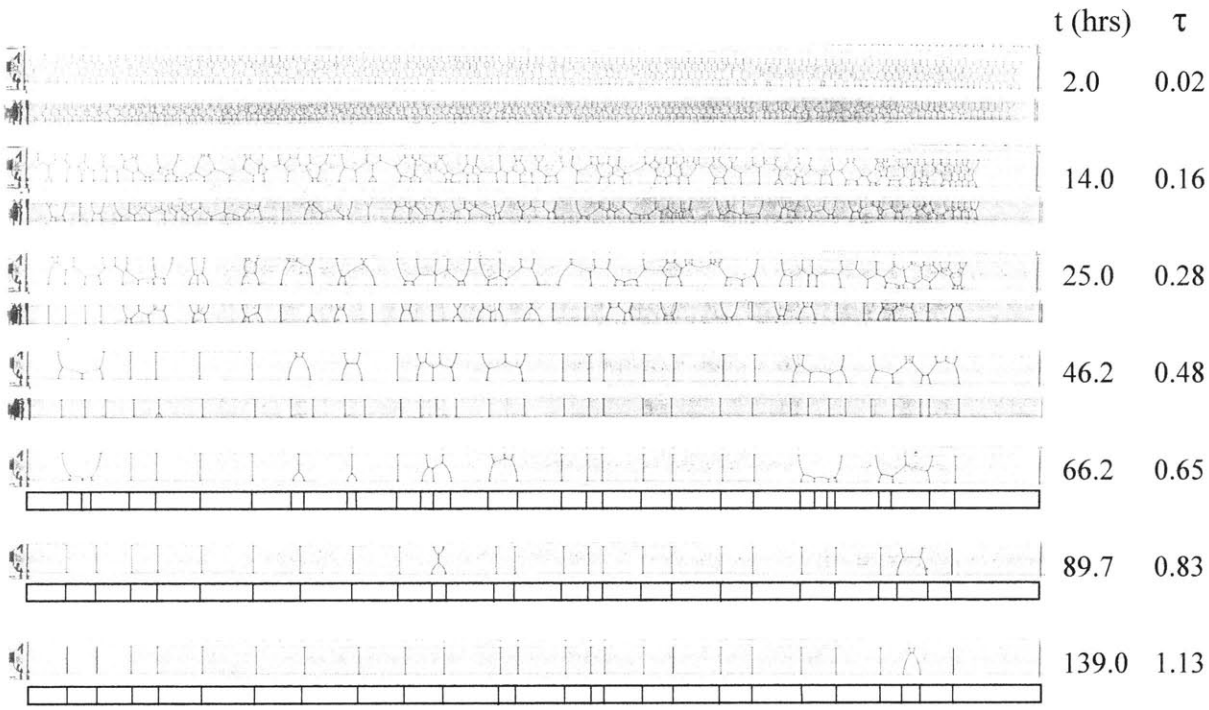


FIG. 7.2: Grain structure evolution of a 3D soap froth network in a rectangular prism with aspect ratio w/h = 1.5. For each time t, the top view corresponds to the wider face and the bottom view to the adjacent, narrower face. τ is the normalized time given by $d\tau = \mu(t)dt/w^2$, where $\mu(t)$ is the average grain boundary mobility at time t.

Initially, the structure is polygranular in both the width and thickness directions. During an initial *incubation* phase, grains in the initially polygranular structure grow to an in-plane size comparable to the prism cross-section's larger dimension. As the evolution proceeds, some grains grow large enough to span the entire cross-section of the line, creating sections of the line with a bamboo structure. We define this as the *nucleation* period, during which bamboo sections continue to form, increasing the number of polygranular clusters until these polygranular clusters separating the bamboo segments achieve geometrically stable configurations with grains having three or four sides within the strip interior and one side defined by the strip edge when looking at the strip from a view corresponding to the larger dimension (see for example Fig. 7.1 (b)). The structure then undergoes a *growth-dominated* evolution during which many fewer bamboo-segment nucleation events are observed, with the evolution occurring

A parallel evolution process, occurring simultaneously with the bamboo nucleation and growth process described above, is the conversion of the 3D polygranular segments to 2D polygranular segments, the latter having all boundary normals lying in one of the two possible planes. We note, before we investigate it further, that for w/h = 1.5, this conversion occurs early, during the incubation period and the initial part of the nucleation period, resulting in a columnar structure with boundary normals perpendicular to the thickness direction (in Fig. 7.2, for $\tau > 0.28$, we see a near-bamboo structure on the wider face and a fully-bamboo structure on the narrower face). For w/h = 1.0, the evolution from 3D to 2D structures continues through all three evolution phases, with 2D clusters appearing with equal probability on all prism faces.

To carry out a quantitative analysis of the kinetics of the grain structure evolution and of the 3D-2D conversion process, an important kinetic parameter to assess is the cell boundary mobility μ. For 2D normal grain growth in strips, it can be shown, using a Mullins-von Neumann analysis [Neu 52, Mul 56], that for a grain of area A:

$$\frac{dA}{dt} = \mu \frac{\pi}{3}(n-6), \tag{7.1}$$

where n is the number of grain sides, and where a side shared with the edge of the strip counts twice (due to the fact that the grain boundary meets the edge at 90° for normal ideal growth or soap froth growth) [Wal 91]. By taking measurements of the rate of area variations for columnar grains with different numbers of sides, we first confirmed equation (7.1), and then used it to quantitatively assess the cell boundary mobility μ. Figure 7.3 shows the evolution of the average mobility with time. The figure shows that the mobility remains relatively uniform with its standard deviation not exceeding 25% of its average value at any given time. It also shows, however, that during the first 150 hours, which as we will see account for more than 90% of the evolution to bamboo in both cases considered, the mobility drops to about 50% of its initial value of approximately 1.5 mm^2/hr. This mobility decay, too large to be neglected, is due to imperfect experimental conditions, associated with froth drying. The average mobility as

a function of time is well fit by a slowly decaying exponential function, as can be seen in Fig. 7.3.

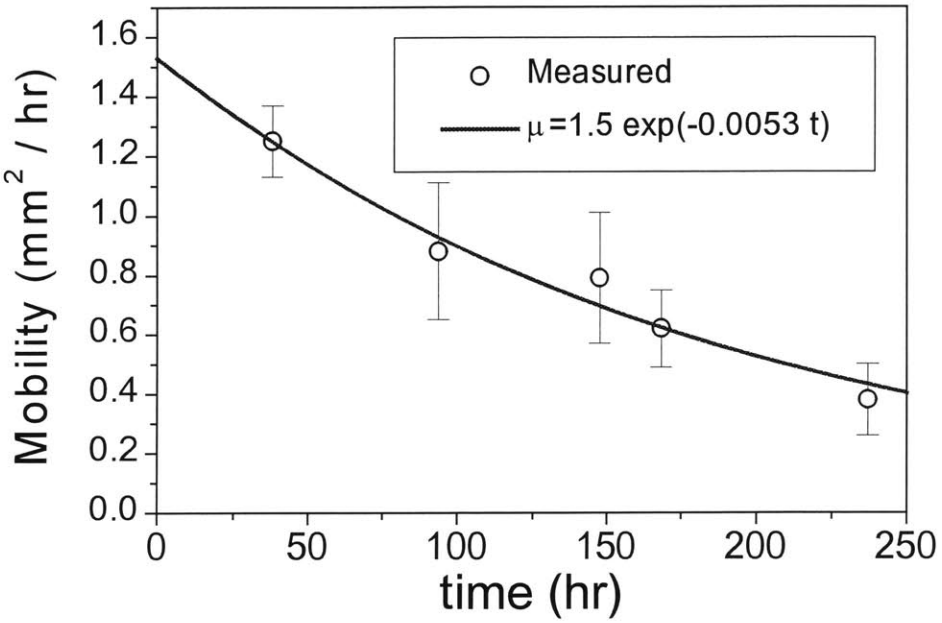


FIG. 7.3: The average cell boundary mobility as a function of time. Error bars represent a total variation of twice the standard deviation. The solid line represents the best fit to an exponentially decaying function.

Ideal (normal) 2D grain growth in thin film or cellular structures has been investigated elsewhere [Fro 88, Fro_2 88, Fro 90, Fay 99]. It has been shown that 2D normal grain growth leads to a uniquely defined grain structure, evolving in a geometrically and statistically self-similar fashion, with an average grain area increasing at a constant rate essentially equal to the mobility μ (assuming that μ is uniform for all boundaries and constant in time). In the case where $\mu(t)$ is uniform but not constant, it is expected that the same result holds with the instantaneous rate of increase of the average area proportional to $\mu(t)$. Ideal normal 3D grain growth is similarly expected to lead to a grain structure evolving with an average volume increasing as $t^{3/2}$ [Mul 86, Kup 00]. This, in turn, implies that the average area of the grain faces increases linearly with time when grain boundary mobility is uniform and constant. In the case where $\mu(t)$ is uniform, but not constant, the same result expected in 2D for the average grain area is expected to hold in 3D for the grain-face area. Under these circumstances, it is expected that, when

looking at one view of a 3D prism, the incremental increase between t and t+dt in the average grain area dA is proportional to $\mu(t)dt$. A naturally defined normalized time allowing the analysis of the grain structure evolution kinetics is therefore τ, given by $d\tau = \mu(t)dt/w^2$ (or equivalently $\tau = \int_0^t \mu(u)du/w^2$), where w is the largest of the two dimensions in the cross-section. The same definition was adopted in the 2D case to unify the kinetic description of the bamboo transformation in strips of different widths [Wal 91, Wal 92, Fay 00].

Figure 7.4 shows cell structure evolution maps obtained in the two cases w/h=1.5 and w/h=1.0. In each case, the figure allows assessment, as a function of the normalized time τ, of the fraction of the total prism length composed of bamboo segments L_b/L (with L being the total line length given by L = 17 prisms x 42 cm = 714 cm). Bamboo lengths are measured in the experiment by assuming that a segment is bamboo, if and only if, it is seen in the two adjacent views to be formed of one or more neighboring-grains spanning the width of the prism. The plots in Fig. 7.4 also show the fraction of the prism length that has a 3D froth structure, L_{3D}/L, and the fraction that is columnar or 2D, L_{col}/L. These lengths are related by: $L_{3D}+L_{col} = L_c = L-L_b$, where L_c is the total polygranular length. Immediate conclusions drawn from analysis of Fig. 7.4 are, first, that the polygranular cluster length decays exponentially as a function of τ in both cases, and second, that the conversion from 3D to 2D occurs early during the evolution in the w/h=1.5 case (at τ=0.4, $L_{3D}/L \approx 0.1$ and $L_c/L \approx 0.7$), leading to further evolution in a quasi-2D mode. In the w/h=1.0 case, the conversion from 3D to 2D clusters occurs throughout the evolution to bamboo structures. The plots in Fig. 7.4 also suggest that L_{3D}/L decays exponentially with τ, except towards the end of the transformation when the effect of the creation of 3D structures through a pinch-off of 2D columnar cells is not negligible.

As discussed in the introduction, process and geometry-sensitive electromigration simulations require knowledge of the evolution of the geometry-dependent polygranular cluster length distribution. In previous work on grain structure evolution in 2D strips [Wal 91, Wal 92, Fay 00], it has been demonstrated that, if nucleation of bamboo segments occurs randomly along the strip length, the resulting polygranular cluster length

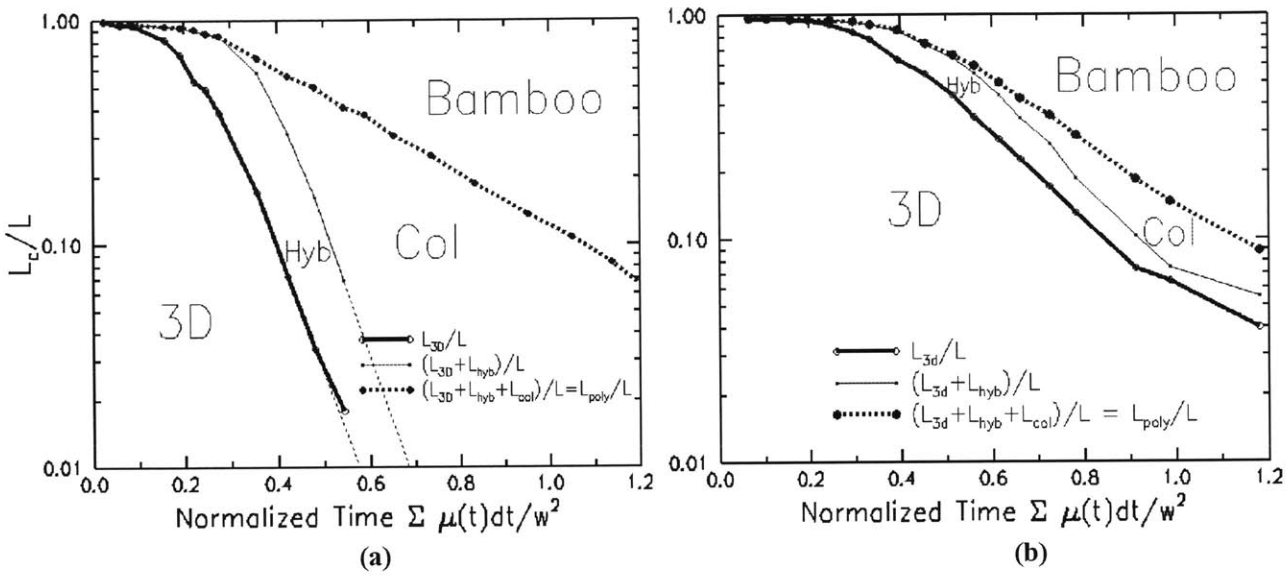


FIG. 7.4: Experimentally obtained grain structure evolution maps for: **(a)** w/h = 1.5; **(b)** w/h = 1.0.

distribution is exponential. It was also shown that the rate of bamboo-segment nucleation per unit time and unit of untransformed length is proportional to μ/w^3, and is negligible in the growth-dominated steady-state. The cluster shrinkage velocity was demonstrated to reach a constant steady-state value proportional to μ/w (assuming constant and uniform μ). This was shown to lead to a time-invariant, steady-state exponential cluster length distribution with an average cluster length proportional to the strip width, and a cluster length fraction decaying exponentially with $\tau=\mu t/w^2$. These results can be expected in our experiments on froth evolution in prisms in the w/h=1.5 case, given the previous argument that the evolution in that case is quasi two-dimensional. The purpose of the following discussion is to show that, in fact, these results hold in both cases when w/h is larger than 1.0 and when w/h is close to 1.0. This is true even when $\mu(t)$ is only uniform and not necessarily constant, provided we use the normalized time which scales with the time-dependent-mobility defined earlier ($\tau = \int_0^t \mu(u)du\ /w^2$).

Figure 7.5 shows exponential plots of the polygranular cluster length distribution for the two cases w/h=1.5 and w/h=1.0, at different times during all phases of the evolution.

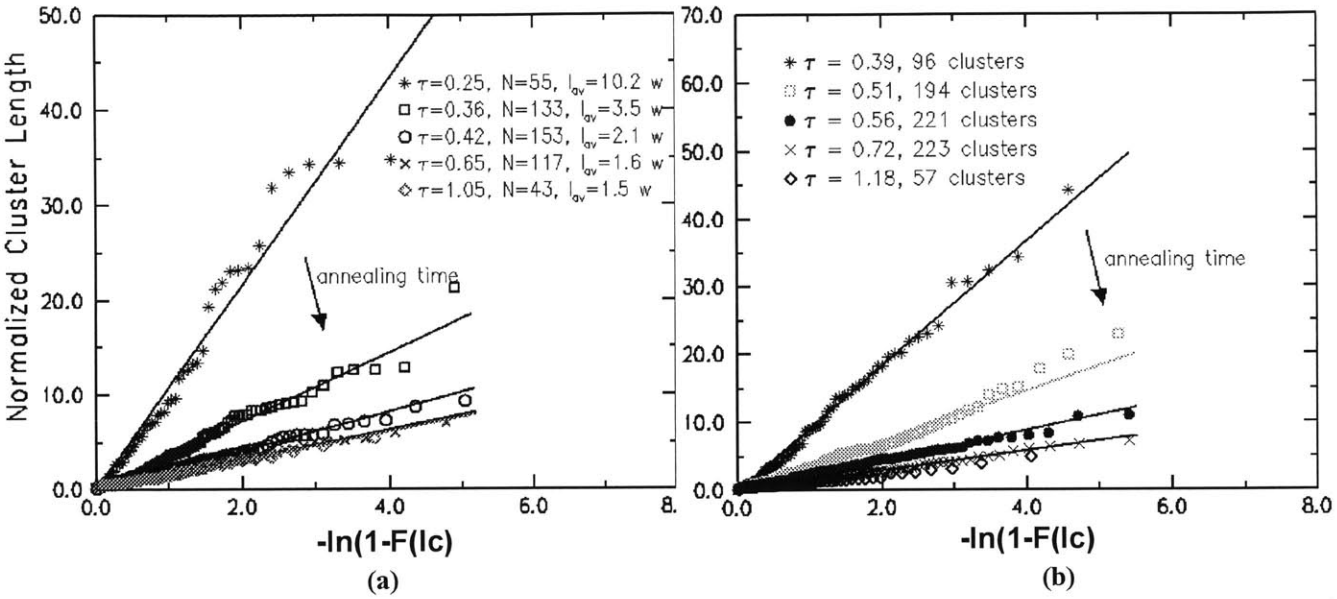


FIG. 7.5: Exponential plots of cluster length distributions at various times during the evolution of cell structures with: **(a)** w/h=1.5; **(b)** w/h = 1.0. $F(l_c)$ is the proportion of clusters shorter than l_c. Solid lines represent the best-fitting exponential distributions. These results demonstrate that the structure reaches a steady-state in which the average cluster length is constant.

When plotting the individual cluster lengths l_c as a function of $-\ln(1-F(l_c))$, (where $F(l_c)$ is the fraction of clusters shorter than l_c), data falling on a straight line is fit by an exponential distribution function with a mean value equal to the line's slope. Figure 7.5 shows that the polygranular cluster length distribution is well fit by an exponential distribution function at all times in all cases. The lines overlap for $\tau > 0.6$ in the case w/h = 1.5, and for $\tau > 0.7$ in the case w/h = 1.0, which indicates a constant average cluster length l_{av} of about 1.5w in this regime for both cases. This result demonstrated by Fig. 7.5 can be rationalized in the following way: whether it is a 2D or a 3D process, random nucleation of bamboo segments leads to grain structures having polygranular-segment-lengths with exponential distributions. That the distribution evolves to a constant limiting steady-state distribution is a consequence of the fact that bamboo-segment nucleation stops in the growth-dominated phase (see Fig. 7.2 in particular for which, after $\tau = 0.48$, no new bamboo segments, or equivalently, no new polygranular clusters are created) and is demonstrated analytically in Appendix A. It is also proved in the appendix that during the steady-state evolution the number of polygranular clusters (or

equivalently the total cluster length fraction, since the average length is constant) decays exponentially as $\exp(-(1/l_{av})\int_0^t v(u)du)$, which is of the form $\exp(-\alpha\tau)$, since l_{av} is proportional to w and, as we will see in the next section, v, the polygranular cluster shrinkage velocity, is proportional to $\mu(t)/w$.

7.4 Analytic Model

If we consider that during annealing-induced evolution, the polygranular cluster-length distribution remains exponential, the problem of predicting the structure statistics is reduced to the determination of the evolution with time of both the average cluster length $l_{av}(t)$ and the total number of clusters N(t), or equivalently, one of these variables and the total cluster length $L_c(t) = l_{av}(t)N(t)$. Assuming that the effect of bamboo nucleation on the total cluster length is negligible, only cluster shrinkage will account for the variations in L_c, so that:

$$\frac{dL_c}{dt} = -v(t)N(t), \tag{7.2}$$

where v(t) is the average value of the rate of polygranular cluster shrinkage at time t. The variation in the total number of clusters is caused by: the increase due to bamboo nucleation inside a polygranular cluster (or, equivalently, cluster-splitting events), and the decrease following cluster disappearance by shrinkage. If a(t) is the rate of cluster splitting events per unit cluster length at time t and v(t) is the rate at which clusters shrink, then assuming an exponential distribution of cluster lengths $f_t(l_c) = (1/l_{av}(t))\exp(-l_c/l_{av}(t))$, the number of clusters that disappear by shrinkage between t and t+dt is the number of clusters shorter than v(t)dt, i.e., $v(t)N(t)dt/l_{av}(t)$, which leads to the second evolution equation:

$$\frac{dN}{dt} = a(t)L_c(t) - v(t)\frac{N^2(t)}{L_c(t)}. \tag{7.3}$$

Equations (7.2) and (7.3) can be recast in terms of reduced dimensionless variables, $\underline{N}$ = Nw/L and $\underline{L}_c$ = L_c/L, which account for geometric scaling, and two dimensionless parameters, $\underline{v}$ = (w/μ)v and $\underline{a}$ = (w^3 /μ)a, which account for geometric and kinetic scaling of the rates of cluster shrinkage and cluster splitting. At this point, knowledge of the initial conditions $\underline{L}_c(0)$ and $\underline{N}(0)$ and the profiles of $\underline{a}(\tau)$ and $\underline{v}(\tau)$ allows solution of equations (7.2) and (7.3) to determine $L_c(t)$ and N(t). It is important to note that we expect the evolution of the normalized variables in the case w/h ≥ 1.5 to be independent of line geometry, which implies that $\underline{a}$ and $\underline{v}$ depend on geometry only through τ (Fayad et al. 2000). Similarly, for a case in which w/h is equal to a certain fixed value close to 1.0, we expect the evolution of the normalized variables to be independent of geometric magnification. This, in turn, requires that the rate of cluster-splitting "a" be proportional to μ(t)/w^3 and the shrinkage velocity "v" be proportional to μ(t)/w, both when w/h is close to and larger than 1.0. We will show, in fact, that the values obtained in the two cases w/h = 1.5 and w/h = 1.0 are in the same range, which will allow the description of the evolution of the grain structure statistics with a single 3D geometry-sensitive model.

The rate of 2D cluster shrinkage has been previously investigated [Wal 92, Fay 00]. Columnar polygranular clusters in the growth-dominated regime are usually bound by pairs of 4 and 5-sided grains (counting the strip edge as a side) with a series of 5-sided edge-grains between them (see Fig. 7.1 (b)). The Mullins-von Neumann law can be used to show that the rate of cluster shrinkage is constant and proportional to 1/w such that

$$v = 2\frac{1}{w}\frac{dA_{4side\_edge}}{dt} = \frac{2\pi}{3}\frac{\mu}{w}, \tag{7.4}$$

where, as before, it should be noted that when using the Mullins-von Neumann analysis for the rate of shrinkage of individual grains in strips, the strip edge counts as 2 sides [Wal 91]. The time evolution of the normalized velocity as defined in equation (7.2) is

depicted in Fig. 7.6, for the cases w/h = 1.5 and w/h = 1.0. In the latter case, we have also plotted the normalized shrinkage velocity obtained when looking only at one face of the prism.

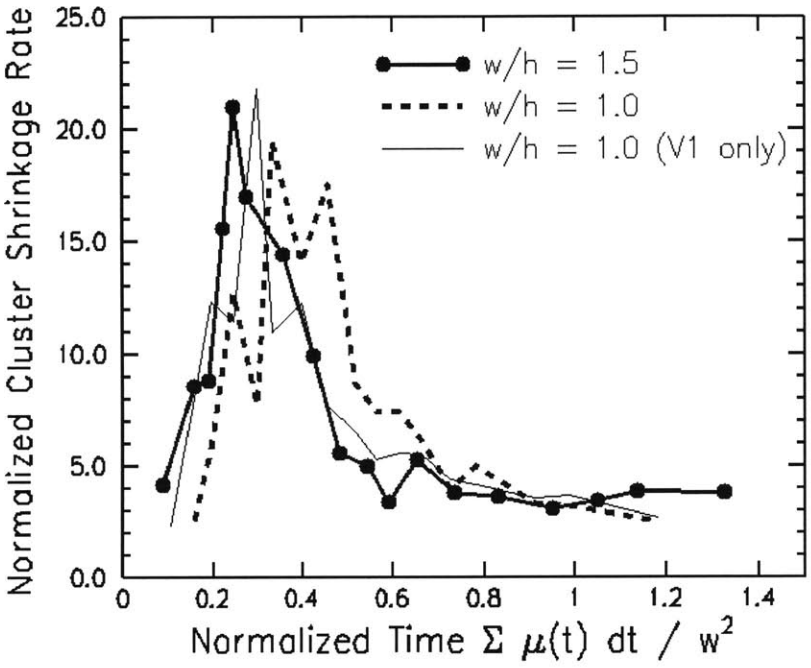


FIG. 7.6: Evolution of the normalized cluster shrinkage rate $\underline{v}=-(1/\underline{N})d\underline{L}_c/d\tau$ with $\tau=\int \mu(t)dt/w^2$.

The plot confirms that the average shrinkage rate is constant in the steady-state regime. Within statistical variations, the curves for w/h=1.5 and w/h=1.0-restricted-to-one-view overlap, and are simply shifted in τ with respect to the curve obtained for the w/h = 1.0 case. This confirms that the shrinkage rate is proportional to $\mu(t)/w$ and shows that the velocity evolution for the two cases differs only by a normalized-time delay ($\Delta\tau \approx 0.15$). The average value of the shrinkage rate in the steady-state exceeds the predictions of equation (7.4) by about 40%. This difference is caused by the high velocities associated with bamboo nucleation at the edge of a cluster as well as the high velocities of the disappearing 3D clusters created by pinch-offs of shrinking columnar-clusters. At the early, nucleation-dominated stage, the values of "v" are variable and higher, which is expected since, following equation (7.2), it is "v" and not "a" that

accommodates the topology-driven cluster length variations associated with random nucleation events.

An analytic assessment of the rate of cluster-splitting per unit time and unit length, a(t), is more complex. This rate is expected to depend on the average grain size at a given time, as well as on the deviation in the grain size, since it is the number of grains that are bigger than a certain width and thickness-related threshold that affects the number of nucleation events. However, as can be seen in Fig. 7.2, and will be discussed in the next section, simulations indicate that significant cluster splitting through nucleation occurs only during the *nucleation* period, and is essentially absent during the steady-state growth regime. This behavior is expected since during the steady-state phase, clusters have stable geometries which are immune to an internal bamboo nucleation event. Therefore, it is a reasonable approximation to take a(t) to be constant during the nucleation period in the time interval $[t_0, t_1]$, and zero at other times. This is confirmed by the data for the evolution of the simulated normalized bamboo-grain nucleation rate $(1/\underline{L}_c)d\underline{N}_b/d\tau$, an over-estimate of the normalized cluster-splitting rate (or equivalently, the bamboo-segment nucleation rate) $\underline{a}=(w^3/\mu(t))a$, as a function of τ shown in Fig. 7.7. N_b is the total number of bamboo grains and $\underline{N}_b = N_b w/L$ is the corresponding normalized number. Similar to the evolution of the shrinkage velocity, plots of the bamboo nucleation rate coincide within statistical variations for w/h=1.5 and w/h=1.0-restricted-to-one-view, and are shifted by $\Delta\tau \approx 0.15$ with respect to the w/h=1.0 case. This confirms that the nucleation rate is proportional to $\mu(t)/w^3$, and that its values are independent of the prism's cross-sectional aspect ratio. It should be noted that the normalized-time delay characterizing the difference in the evolution between the two cases considered is not unexpected. For a bamboo grain to nucleate, a grain has to grow to span both the thickness and width of the line, two quasi-random events which would be expected to occur at similar times when w/h is close to 1.0. When the aspect ratio is large (w/h = 1.5 case) however, the structure first converts to a 2D columnar structure early during the evolution (in terms of a time normalized by the square of the largest dimension, w) and a bamboo grain nucleation event occurs as soon as the width of the line is spanned by a growing grain.

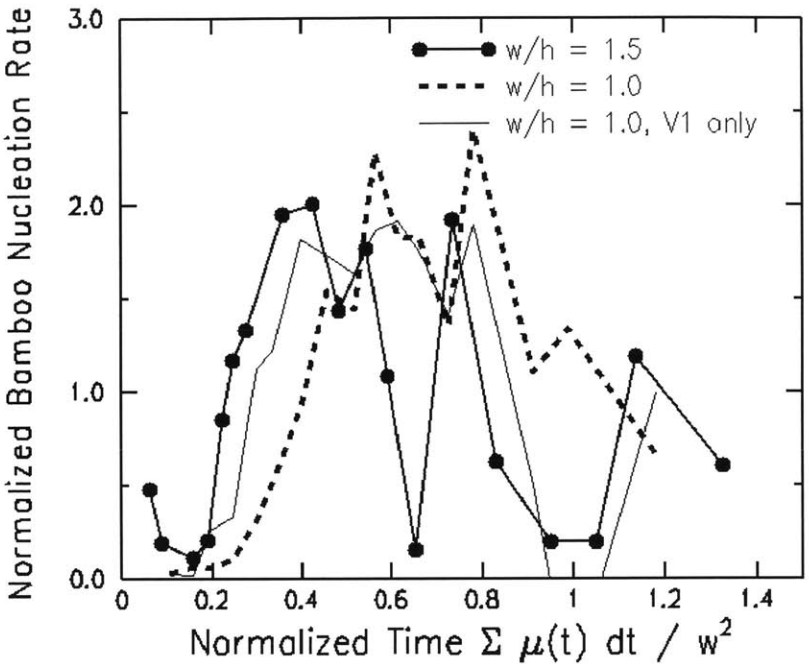


Fig. 7.7: Evolution of the normalized bamboo nucleation rate $\underline{a}=(1/\underline{L}_c)d\underline{N}_b/d\tau$ with normalized time. The plots for the two aspect ratios differ statistically only by a delay time, confirming that the nucleation rate is proportional to μ/w^3.

Using, as a simplification, a cluster-splitting rate a(t) such as the one defined above (taking $\underline{a}$ to be constant in a normalized-time interval $[\tau_0, \tau_1]$, and zero at other times), and a constant normalized shrinkage velocity, the coupled equations (7.2) and (7.3) can be solved analytically to obtain:

$$\underline{L}_c(\tau) = \underline{L}_{c0}\, exp\left(-\frac{1}{\underline{l}_{av0}}\underline{v}\cdot(\tau-\tau_0) - \frac{1}{2}\underline{v}\cdot\underline{a}\cdot(\tau-\tau_0)^2 \right) \tag{7.5}$$

and

$$\underline{l}_{av}(\tau) = \left(\frac{1}{\underline{l}_{av0}} + \underline{a}\cdot(\tau-\tau_0) \right)^{-1} \tag{7.6}$$

for $\tau_0 < \tau < \tau_1$, and

$$\underline{L}_c(\tau) = \underline{L}_c(\tau_1)\, exp\left(-\frac{1}{\underline{l}_{av1}}\underline{v}\cdot(\tau-\tau_1) \right) \tag{7.7}$$

and

$$\underline{l}_{av}(\tau) = \underline{l}_{av}(\tau_1) \tag{7.8}$$

for $\tau > \tau_1$, where $\underline{l}_{av} = l_{av}/w$. The equations above generalize the ones obtained in [Fay 00] to model the effect of variable, time-dependent grain-boundary mobility. That the average cluster length reaches a constant value ($\underline{l}_{av}$ reaching a constant value means also that l_{av} reaches a constant value proportional to w) is in agreement with previous results for 2D systems [Wal 91, Fay 00]. We also note, as a final remark before discussing these results, that equation (7.7) is consistent with predictions from the analysis presented in Appendix A.

7.5 Results and Discussion

We have used a soap froth experiment to generate cell structures in prisms with different cross-sectional aspect ratios, and compared the evolution of the polygranular cluster statistics observed in experiments with predictions made using the analytic model derived above. Table 7.1 shows values for various error-minimizing parameters obtained for the experimental results.

Table 7.1: Parameters for grain structure evolution in prisms with different aspect ratios w/h that minimize the error $e = \langle \log^2(\underline{L}_c/\underline{L}_{c\_fit}) \rangle + \langle \log^2(\underline{l}_{av}/\underline{l}_{av\_fit}) \rangle + \langle \log^2(\underline{N}/\underline{N}_{fit}) \rangle$.

w/h	τ_0	τ_1	$\underline{v} = v\, w/\mu$	$\underline{a} = a\, w^3/\mu$.	Error
1.5	0.15	0.65	5.96	1.06	0.03
1.0	0.30	0.80	5.92	1.24	0.02

The results shown in Table 7.1 demonstrate that during 3D normal grain growth in rectangular prisms, the incubation period t_0 is approximately $0.15w^2/\mu$ (if μ were constant) for lines with aspect ratios $w/h \geq 1.5$, and is longer when w/h is closer to 1.0 ($\tau_0 = 0.3$ for the case w/h=1.0). The nucleation period lasts until $t_1 \approx 0.65\ w^2/\mu$ (assuming a

constant μ) for w/h $\geq$ 1.5 and becomes longer as the aspect ratio approaches 1.0 (with a maximum delay given by $\Delta\tau \approx 0.15$ for the case w/h=1.0). The values obtained for the shrinkage and splitting rates are similar for both cases ($v \approx 5.9 \pm 0.2$ and $a \approx 1.1 \pm 0.1$). The value of "v" is higher than what is observed in the growth-dominated regime and than what is expected from the Mullins-von Neumann analysis described above, even after accounting for the high velocities associated with bamboo nucleation events. This is the case because the reported value includes averaged effects of the high velocities initially observed during the nucleation period, along with the constant steady-state value (see Fig. 7.6). As will be shown below, this limitation, which could be overcome by allowing a more complex time-dependence of the shrinkage velocity in the model, does not significantly alter the predictive capacities of the analytic model.

The fact, observed in Fig. 7.6 and 7.7, that the time dependencies of $\underline{v}$ and $\underline{a}$ for prisms with the two aspect ratios considered differ only by a time delay, within statistical variations, is demonstrated by the time evolution of the variables plotted in Fig. 7.8, 7.9, and 7.10 (showing the evolution of $\underline{L}$, $\underline{N}$, and $\underline{l}_{av}$ respectively) for both experimental data and model predictions. This result confirms the validity of the analytic model given by equations (7.2) and (7.3). We also note that the values of $\underline{L}$ and $\underline{N}$ decay exponentially during the steady state phase ($\tau > \tau_1$), while $\underline{l}_{av}$ reaches a constant value of about 1.5 ± 0.2 (corresponding to an average cluster length of 1.5 times the line width). Figure 7.11 shows the normalized total number of bamboo grain nucleation events $N_b w/L$ as a function of normalized time. This data demonstrates the same time delay for the w/h=1.5 case and the w/h=1.0 case. The final bamboo structure has grains with an average length of about 1.5 times the line width in the two cases, in good agreement with results presented by Rosa and Fortes on bamboo structures resulting from the evolution of a 2D liquid foam [Ros 98]. In contrast, in the case of cylindrical tubes, Fortes et al. reported on a final bamboo structure with an average grain length very close to the tube diameter [For 98]. This discrepancy is likely to be related to the isotropic nature of growth in the cylinders, leading to more bamboo grain nucleation events.

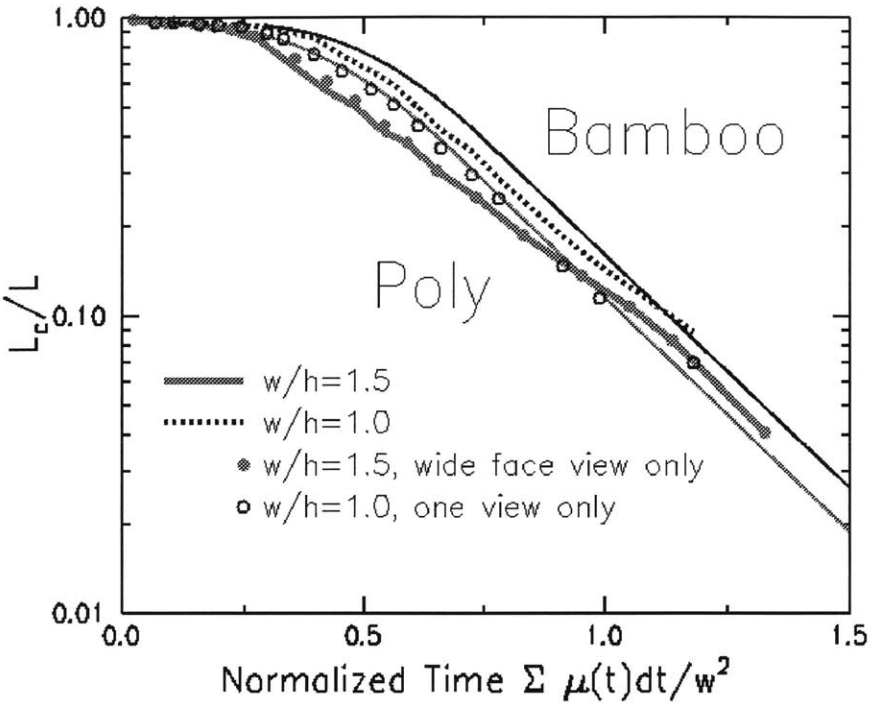


Fig. 7.8: Evolution of the normalized total cluster length $\underline{L_c}=L_c/L$ with τ. Solid lines represent the evolution predicted using the analytic model developed in the text.

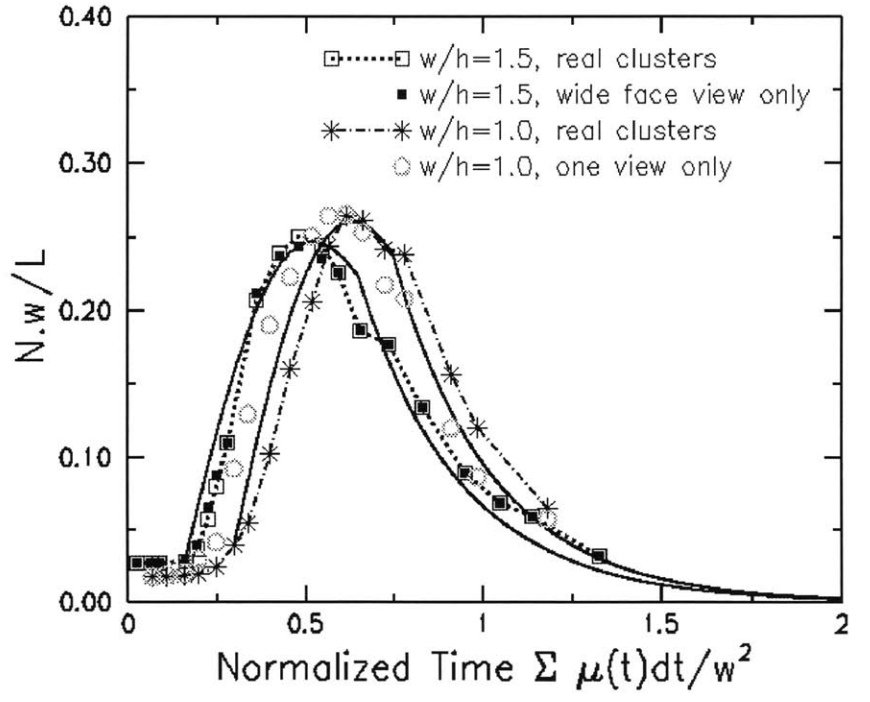


Fig. 7.9: Evolution of the normalized number of clusters $\underline{N}=Nw/L$ with τ. Solid lines represent the predictions of the analytic model.

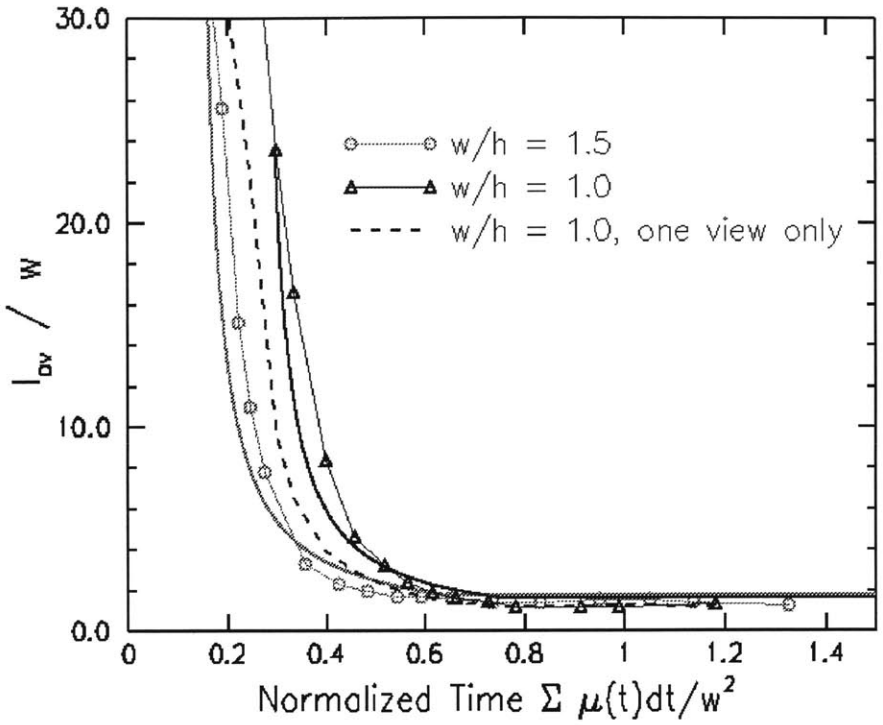


Fig. 7.10: Evolution of the normalized average cluster length $\underline{l}_{av}=l_{av}/w$ with τ. Solid lines represent the predictions of the analytic model.

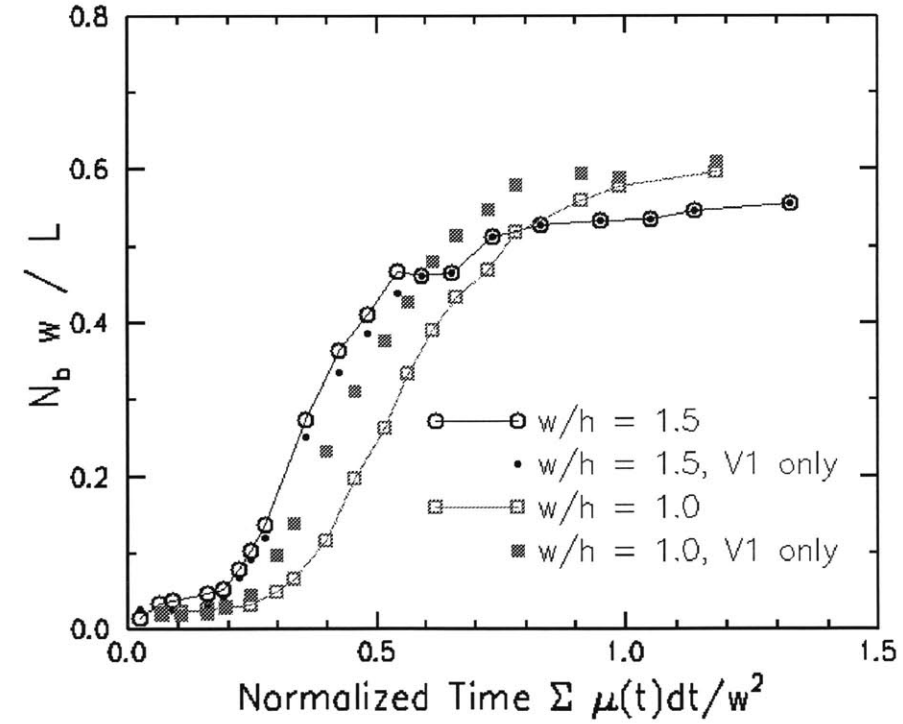


Fig. 7.11: Evolution of the normalized number of bamboo grains for the two different aspect ratios studied in experiments.

Recently, Kuprat et al. have extended their 3D grain growth simulation tool [Kup 00] to simulate growth in rectangular prisms. Preliminary results show that the bamboo structures obtained using this front-tracking simulation are in agreement with the results presented here [Kup 00]. Figure 7.12 shows that the distribution of the final bamboo grain lengths normalized by the line width is width-independent for initially polygranular lines, as expected with geometric scaling, and is well fit by a lognormal distribution function.

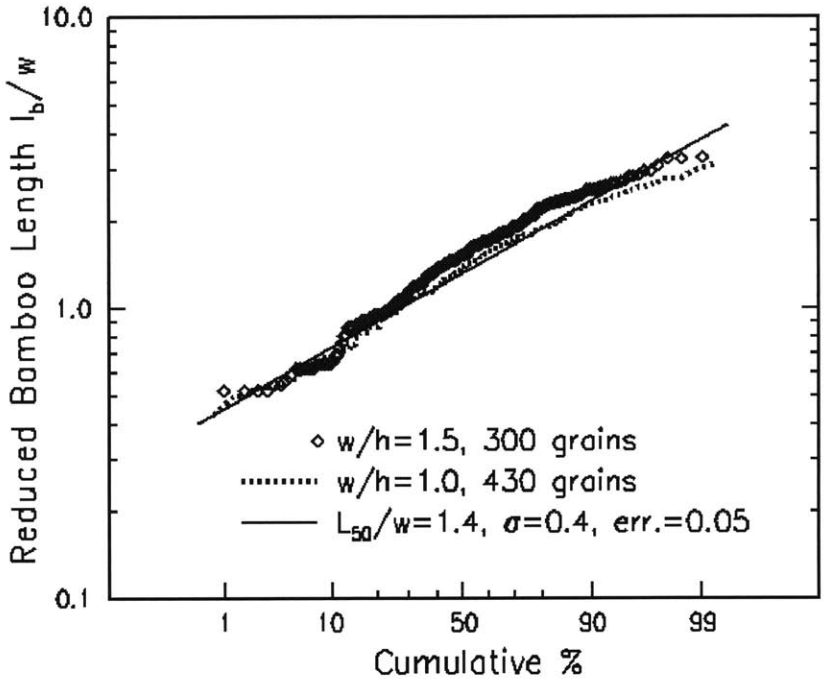


Fig. 7.12: Lognormal plot of the normalized bamboo grain length l_b/w distributions in the bamboo structures resulting from prolonged evolution of initially polygranular cellular structures in prisms with different aspect ratios. The overlapping curves for different values of w/h show that the distribution of l_b/w is aspect-ratio-independent and identical for initially 2-dimensional or 3-dimensonal structures. The Solid line represents the best-fitting lognormal distribution.

Using the analytic model, the optimal values obtained for the parameters, and an interpolation (e.g. linear) of the incubation time between the two values obtained for the quasi-2D case w/h=1.5, and the extreme 3D case w/h=1.0, it becomes possible to predict the evolution of the grain structure statistics during 3D normal grain growth, for a prism with any aspect ratio w/h (including the extreme case w/h = 1.0).

7.6 Summary

We have carried out experiments on 3D froth evolution in rectangular prisms. By analyzing results from experiments we have also developed a geometry-sensitive analytic model for the evolution of 3D grain structures to bamboo grain structures in rectangular prisms. The model allows prediction of the prism geometry dependence of the two transformation parameters, the bamboo-segment nucleation rate and the polygranular cluster shrinkage velocity. The model also allows prediction of the time and geometry dependence of the polygranular segment length distribution. With compact analytic models it is possible to generate appropriately varying grain structures for simulations of interconnect reliability.

Chapter 8

Summary and Future Research

8.1 Summary of Results

Electromigration-induced failure of interconnects remains a major concern in assessing and controlling the reliability of modern integrated circuits. Knowledge of the grain-structure-dependent details of the transport mechanisms is required for the prediction of failure time statistics. This thesis accomplished this goal through different means, for two categories of interconnects: straight-line interconnect strips and interconnects with general tree-shaped geometries.

First, analytic models were developed for geometry-sensitive and process-sensitive evolution of the grain structure statistics in interconnects with 2D (columnar) or 3D (non-columnar) structures. The models were validated through comparisons with experiments on froths and through comparisons with grain growth simulations. These models can be used to efficiently generate very large populations of realistic interconnect structures suitable for EM simulations and reliability predictions. The models were also used in the case of fully bamboo structures in conjunction with EM simulations to show that grain-orientation dependent surface diffusivities are a likely cause for variations in the failure times of interconnects with bamboo structures.

Second, the grain growth simulation GGSim was modified to simulate patterning and subsequent grain growth in 2D interconnects with general tree geometries. The tool, PolySeg, was generalized to extract from GGSim interconnect grain structures of trees and of wide polygranular lines. PolySeg was also modified to account for the effects of diffusion along all the available grain boundary paths along the line length. Using this modified version of PolySeg, median failure times were predicted for a wide range of values of w/D_{50}, covering both near-bamboo lines and wide, fully polygranular lines, and the results were found to be in agreement with experiments. With these modified versions of GGSim and PolySeg, large populations of geometrically equivalent trees with appropriately varying microstructures can be generated for electromigration simulations and reliability analyses.

8.2 Future Research

Be it through the development of new simulation techniques or the more appealing, because more efficient, building of analytic models, this thesis contributes to the accurate description of interconnect microstructures and of the way in which microstructures affect interconnect reliability. However, the results obtained on these two fronts can still be generalized to cover less restrictive assumptions.

Although the analytic model for grain structure evolution in linear interconnects is powerful because it includes the 3D effects present when thickness and width are comparable, as well as the effects of a non-constant grain boundary mobility, it only handles the effects of uniform, purely curvature-driven, grain growth. Interconnect structures can have variable grain boundary energies and mobilities [Fro_2 94]. Non-ideal effects also include the effects of surface grooving at the grain boundaries [Mul 58, Fro 90], solute drag [Fro 94], variable surface/interface energies [Car 96], and strain-energy density effects [Car 96]. The grain growth simulation GGSim allows simulation of such effects and can, in principle, be used to extend the current analytic models in order to incorporate some or possibly all of these effects. The models can also be confirmed

through an adequate set of experiments on real metal interconnects, and eventually modified to account for specific non-ideal details observed. However, this is beyond the scope of this research, the objective of which was, in part, to develop basic analytic models for interconnect grain structure evolution to provide the basis for more advanced models that account for non-idealities.

The analytic models developed here, which allow the determination of the length distributions of the high-diffusivity polygranular clusters, and the low-diffusivity bamboo segments are crucial to the determination of the lifetimes and lifetime variations in Aluminum-based interconnects, which presently are still the most widely used metallizations. More recently, Copper-based metallizations, which are more appealing, because of the higher conductivity for copper, have been successfully used in the making of integrated circuits [IEEE 93, Awa 95, Hu 98, Hu 99]. Copper is characterized by a higher melting temperature than Aluminum, which results in lattice and grain boundary diffusivities that are three orders of magnitude lower than lattice and grain boundary diffusivities for Al at typical testing temperatures (~500K). This would lead to longer failure times for Cu, by the same orders of magnitude than for Al, assuming similar diffusion mechanisms in both cases. However, the experimentally observed difference in lifetimes of only one to two orders of magnitudes higher in the case of Cu compared with Al indicates that grain boundary diffusion is not the main failure mechanism in the currently used Cu damascene structures. In fact, experimental results on Cu interconnects with different microstructures, where large voids were observed along the interface between the copper and the top SiN diffusion barrier, indicate that it is interface diffusion, due to the bad adherence of Cu to the SiN layer, that is the principal electromigration diffusion path in this case. In experiments on Cu structures sandwiched with a top and bottom Ta layer, Hu et al. observed that electromigration lifetimes were independent of line thickness and increased linearly with line width, which supports that diffusion was occurring along the sidewall surfaces of the lines [Hu 97]. Surface or interface diffusion can therefore account for the failure times being not as high as they would be if grain boundaries were the paths with highest diffusivity. A more adherent interface, probably using a different barrier than SiN, would, in the case of damascene

structures, not only increase the failure times in Cu-based interconnects, but also restore grain boundary diffusion as the fastest diffusion path, leading to failure mechanisms similar to the ones observed in Al, where polygranular clusters have a higher diffusivity than bamboo segments. Ultimately, the accurate assessment of failure time statistics would require for Cu, as for Al, knowledge of the grain structure statistics, which this thesis provides, through analytic models, but also, when models are not available, through newly developed simulation capabilities.

On the simulation front, we have demonstrated a set of tools which make it possible to predict, at the design level, process-sensitive failure characteristics and reliability. These have been discussed in previous chapters and are summarized in Fig. 1.5. In particular, the tools *pattern*, *post-pattern-anneal*, and PolySeg, which have been created to generate and assess grain structures in complex tree geometries. This tool could be used, as GGSim was used in the case of straight lines, to establish analytic models for the development of bamboo structures in trees. Such a treatment should include the grain structure statistics at tree junctions and at tree shoulders. Depending on the geometry, tree junctions may in particular stabilize triple points leading to even fully evolved structures which are still partially polygranular. Experiments with 3D soap froths might also be undertaken to assess 3D effects on grain structure evolution in trees, in a similar approach to what was done here for linear structures. In parallel, 3D front-tracking simulations of grain growth, can provide flexible tools for comparison with froths experiments, and ultimately to the development of elaborate analytic models, in the cases in which the 2D simulation does not apply. We have been collaborating with Kuprat et al. at the Los Alamos National Laboratory on the development of a 3D front-tracking simulation of grain growth, Grain3D [Kup 00, Kup$_2$ 00]. Preliminary results have qualitatively demonstrated the validity of the simulation model, but further developments are needed in order to ensure the reliability of the simulation for 3D grain structure evolution predictions.

We have successfully captured with simple analytic models as well as elaborate simulations the physics of microstructure evolution in complex patterned thin-film

structures. In particular, we have developed an array of models and simulations that can be used to investigate the impact of geometry and process history on microstructure evolution, and ultimately on EM-induced failure statistics. We have also developed a methodology for model development that can be further extended to account for 3D effects on grain structure evolution in even more complex interconnect structures. Incorporation of the existing new models for grain structure and grain structure evolution into one global reliability prediction tool is only pending the completion of the development of ERNI, the circuit-level interconnect geometry/current extraction tool. Once achieved, this will allow the determination of the simulated reliability of a whole circuit with a push of a button!

Appendix A

Invariance of Exponential Distributions by Uniform Shrinkage

Lemma: *Let $(S)_i$ be a population of segments with length $(l)_i$ that initially, at t=0, has an exponential distribution with an average value l_{av}. Assume that l_i can only shrink, and only uniformly: $\forall\ t \geq 0$, $\exists\ v(t) \geq 0$, $\forall\ i$, $dl_i/dt = -v(t) \leq 0$. Then, the length distribution of segments with positive lengths remains unchanged over time. Moreover, the number of these segments decays as $\exp(-\int_0^t v(u)du / l_{av})$.*

Proof: Take $l_t = \int_0^t v(u)du$, and N_0 the initial number of segments. The initial distribution of segment lengths is given by $f_0(x) = \exp(-x/l_{av})/l_{av}$. With these notations, exactly $N_t = N_0 \int_{l_t}^{+\infty} f_0(x)dx = N_0\, exp(-l_t / l_{av})$ are positive-length segments at time t which already proves the last part of the lemma. In addition, the number of segments that have a length at time t larger than a positive length l is exactly the number of segments at time 0 with length larger than $l+l_t$: $N_0 \int_{l+l_t}^{+\infty} f_0(x)dx$ $= N_0\, exp(-(l+l_t)/l_{av}) = N_t\, exp(-l/l_{av})$. This means that, at time t, the fraction of the population with positive lengths that have a length longer than l is $exp(-l/l_{av})$ which is characteristic of an exponential distribution with average l_{av}.

Appendix B

New Features of GGSim: Patterning and Post-Patterning Annealing of Arbitrary Shapes

B.1 Usage of the Programs

The grain growth simulation program is a result of several years of research and development efforts [Fro 88, Fro 90, Wal 91, Wal_2 91, Wal 92, Fro 94, Fay 97, Fay 99]. In the context of this thesis, this program has been extended to handle the patterning of general shapes from a continuous film structure, and to simulate the effects of post-patterning annealing grain structure evolution caused by annealing of structures with these shapes.

GGSim's different simulation capabilities, which are written in C code, can be run using a command file in which the user specifies the different tasks he wishes to be performed. These different tasks have distinct integer flags that make up the initial entry in each line of the command file. They are mapped in the program *run.c* in which, depending on the value of the task's flag (the flags actually correspond to different "Case" statements in the file *run.c*), the program calls the appropriate GGSim procedures to be performed. These capabilities include, but are not limited to, simulating nucleation and impingement of a thin film structure (Case 100 and 200), grain growth in such films (Case 7), etching linear strips from a continuous film (Case 60), and outputing files for graphic visualization (Case 9). We have added the capability of patterning one or many

arbitrary shapes (Case 70) and have extended the programs involved in Case 7 (mainly *numerics.c*) and Case 9 (*graphics.c*) to simulate further grain structure evolution in such patterns and to allow visualization of the results.

A specific shape to be patterned has to be entered in a *.dat* file as an array of coordinates of the pattern edges listed in a counter-clockwise order. This file should be in the directory where the continuous film is stored. An example of such a file, "giraffe.dat", which was the pattern used in the tree interconnect shape of Fig. 6.11, is listed below:

```
12.0 24.0
15.0 24.0
15.0 28.0
26.0 28.0
26.0 11.0
34.0 11.0
34.0 14.0
29.0 14.0
29.0 47.0
26.0 47.0
26.0 31.0
15.0 31.0
15.0 34.0
12.0 34.0
```

A typical command file, "anneal_giraffe.com", allowing the generation of a continuous film structure, the patterning of this structure using "giraffe.dat" along with further annealing-induced evolution is listed below:

```
0 0 is a flag indicating a comment line, to be disregarded by the code
0 Test command file for "stripped" version of gg_sim
0
0 Command line: ~/gg_sim_stripped/source/grain_growth > anneal anneal_giraffe
0
0 Annealing sequence
0
0 5 -> output directory
0 6 -> input directory
0
100 65_65_1 will nucleate and impinge 65*65 structure in dir: 65_65_1
200 65_65_1_init
0 the previous will format the structure into one suitable for further tasks
1  0.001      assign a mobility constant
2  0.080      tooclose
3  0.4        assign a critical curvature for stagnation
5  65_65_1_k0.4_stag output_dir
6  65_65_1_init input_dir
```

```
7  20.0 anneals the sructure until τ=20.0, or stagnation (if sooner)
5  65_65_1_k0.4_stag_giraffe output_dir
6  65_65_1_k0.4_stag input_dir
70 1 giraffe 1 (1 for internal, 0 for ext, default for total structure)
0 the previous line indicates that there is only one pattern to be etched,
0 that the name of that pattern is giraffe.dat, and that the structure external
0 to the pattern is to be disregarded.
0 70 2 giraffe1 giraffe2 would pattern both giraffe1 giraffe2 from the film
9 0   performs a graphic dump.
7 25.0 anneals the patterned structure until τ=25.0, or stagnation (if sooner)
9 0    performs a graphic dump
11     save the latest structure in the latest declared output directory
12     exit simulation
```

The program is run by typing the command line "anneal anneal_giraffe" in the directory where the executable *anneal* is located.

B.2 List of Code

```
/* pattern.c, Walid Fayad, March 97, update for master copy 29Aug97 */


/*****************************************************************************/
/*****************************************************************************/
/* Outline for the patterning routine
   Big picture:

   1) Get the etch points coordinates
   2) Get a list of all segment and triple points in the film
   3) For all etch edges,
   4)       For all the segment and triple points,
   5)            Compute intersection
   6)            If intersection exists,
   7)                  Store x and y coordinates of the intersection pt.
   8) For all intersection points,
   9) Create 2 new edge type triple points & Update linkages of the structure
consistently
  10) The corners of the pattern are added to the structure as stagnant segment
points
*/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include <time.h>
#include<sys/types.h>
#include<sys/times.h>
#include "defines.h"
#include "struct.h"

#define MAX_INTERSECT_POINTS 5000
```

```
#define CORNER_MAX 200
#define SAFETY 2.49

typedef struct {
  float x,y;
}Corner;

typedef struct {
        float a,b;
       int vertical;
        float xmax, xmin, ymax, ymin;
       }Edge;

typedef struct {
        float x, y;
        int index;
       int go;
       }Data;



/* gb_pattern procedure declaration */
int pattern_main(FILE *, char*, char**, char*, int, int);
int read_pattern(char*, Corner *, Edge *);
int collect_data(Data*, int*);
float min(float, float);  /* finds the minimum of 2 number */
float max(float, float);  /* finds the maximum of 2 number */
int intersect(Data[], int, Edge[], int, Data[], Corner[]);
           /* finds the intersections of the film with the overlayed structure
*/
void sort(Data *, int, int (*) (Data, Data));
int by_x(Data, Data);
int by_opp_x(Data, Data);
int by_y(Data, Data);
int by_opp_y(Data, Data);
int CCW(Data, Data, Data);
int intersection (Data, Data, Data, Data);
int inside(Data, Corner[], int);
              /* checks if a given point is inside or outside the geometry */
int walk_to_j(int,int,int,Data*,Data*);
void update_struct(Data[], Data*, int, Corner[], int, int);
int keep_in_or_out(int, int*, int);


/* The followings are used in the updating routine in gb_pattern */
int walk_ccw(int,int);
int PREVIOUS_walk_ccw(int,int);
int get_index(int, int);
void get_tp1_tp2(int, int *, int *, int);
float cross(float [], float []);
int cycle_forward(int);
int cycle_back(int);
int find(int,Data*,Data*,Corner*,int);
int commun(int,int,int); /* gives the commun neighbor for 3 triple points */
int init_times2();


/* gb_support */
extern int questat(int control);
extern int compact();
extern int put_triple(int t);
```

```
/* External Variables Declaration */
extern segment_point *segment;
extern triple_point  *triple;
extern grain_struct  *grain;
extern int    pnt_stat, grain_max, segment_max, triple_max, strip, n_regions;
extern float  xsize, ysize, xsized2, ysized2;
extern float  xlo, xhi, ylo, yhi;
extern struct tms cpu_start,cpu_current;
extern time_t current_time,start_time,time_diff;
Corner corner[MAX_PATTERNS][CORNER_MAX];

int pattern_main(FILE * fcommands, char* fn_input, char** pat_name, char*
fn_output,
              int flag_in_or_out, int number_pats)
{

  char text[200];
  char fn_pattern[200];
  char out_name[80] = CORNERS;

  Edge edge[CORNER_MAX];
  Data *data;
  Data  new[MAX_INTERSECT_POINTS];
  int ncorner[MAX_PATTERNS];
  int segmtpts;
  int count;
  int newpoints=1; /* new points is the number of new edgepoints obtained */
  int before0=0;
     /* before0 will be the index of the triple point preceding triple[0]*/

  /* Initialize the cpu time counters. */
  init_times2();

  /* Set some flags for the grain growth utility routines. */
  pnt_stat = 2;


  /* Verify that structure is continuous and can be patterned */
  if(strip==1)
    {
      printf("ERROR GB_PATTERN: Input Structure is a Strip. Cannot Be
Patterned");
      return(0);
    }

  fprintf(fcommands,"0 PATTERN:\n0\n");
  current_time = time(&current_time);
  fprintf(fcommands,"0 Simulation begun: %s",ctime(&current_time));

  /* LOOP ON NUMBER OF PATTERNS */
  for(count=0;count<number_pats;count++)
    {
      *fn_pattern = 0;
      strcat(fn_pattern,DATA_DIR);
      strcat(fn_pattern,fn_input);
      strcat(fn_pattern,"/");
      printf("\npat%s",pat_name[count]);

      strcat(fn_pattern,pat_name[count]);
      printf("\nHELLOO\n");
      strcat(fn_pattern,".dat");
      /* Read the Pattern Info */
```

```
      ncorner[count] = read_pattern(fn_pattern, corner[count], edge);/* reads
in the pattern info*/

/*Fill the array data with all the points (segments and triples of the film
structure */
/*First allocate memory for the array data that is going to carry all the
structure  triple points and segment points coordinates */
      if( (data=malloc(2*(triple_max+segment_max)*sizeof(Data))) == NULL )
       {
         printf("cannot allocate memory for array data ");
         exit(-1);
       }

      segmtpts = collect_data(data, &before0);
      printf("\n\n\nBEFORE triple_max = %d, segment_max=%d, grain_max=%d\n\n",
            triple_max,segment_max,grain_max);

      /*
       pintf("*****************   USED TRIPLES  *************************\n");
       for(triplept = triple[0].forward;
       triplept != 0;
       triplept = triple[triplept].forward)
       printf("triplept=%d,  used=%d x=%.3f y=%.3f\n",triplept,
triple[triplept].used,
       triple[triplept].x, triple[triplept].y);
      */


      /*  printf("\n\n\n\n\n\n\n\n\n*****************   UNUSED TRIPLES
***************\n");
         for(triplept = triple[triple_max-1].forward;
         triplept != triple_max-1;
         triplept = triple[triplept].forward)
         printf("triplept=%d,  used=%d x=%.2f, y=%.2f\n",triplept,
triple[triplept].used,
         triple[triplept].x, triple[triplept].y);
         */

      /* Get the intersection points */
      newpoints = intersect(data, segmtpts, edge, ncorner[count], new,
corner[count]);
      printf("\nThe patterning creates %d intersection points\n", newpoints);

      update_struct(new,data,newpoints,corner[count],ncorner[count],before0);

      printf("\n\n\n\n\nAFTER UPDATE, triple_max = %d, segment_max=%d,
grain_max=%d\n\n",
            triple_max,segment_max,grain_max);

      /* Set strip to be the number of corners of the last pattern */
      strip = ncorner[count];   /* In "etch.c" strip=1 */

      /* free data */
      free(data);

   } /* END OF LOOP ON NUMBER OF PATTERNS */


  /* TRASH THE EXTERNAL STRUCTURE (flag_in_or_out==1) or THE INTERNAL ONES
(0),if flag<0 IGNORE*/
  if(flag_in_or_out>=0)
  {
    if(keep_in_or_out(flag_in_or_out, ncorner, number_pats)==0) return(2);
```

```
    if(flag_in_or_out==1) printf("Keeping only INTERNAL structures.\n\n");
    if(flag_in_or_out==0) printf("Keeping only EXTERNAL structure.\n\n");
  }

  /* Compact, then write out the new structure. */
  printf("Compact().\n");
  if (compact() != 1) return(2);
  printf("\n\n\n\n\nAFTER COMPACT(), triple_max = %d, segment_max=%d,
grain_max=%d\n\n",
        triple_max,segment_max,grain_max);

  /* Print out the queues. */
  printf("Questat().\n");
  if (questat(2) != 1) return(2);

  /* Finally Copy the Pattern File to the Output Directory */

 printf("Copying the pattern file to the output directory...\n");
 sprintf(text, "cp %s %s%s/%s.dat", fn_pattern, DATA_DIR, fn_output, out_name);
 system(text);

 printf("\n\n--------------------------------\n");
 printf("Pattern terminated successfully. \n\n\n");

 return(1);
}



/*-------------------------------------------------------------------------
----*/
/* This function reads in the pattern coords into an array: corner[], and an
array edge[] */
int read_pattern(char * fn_pattern, Corner* corner, Edge* edge)
{
  FILE * fp10;
  int i=0, status, ncorner;
  float dx, dy, aa;

  if((fp10=fopen(fn_pattern,"r"))==NULL)
    {
      printf("ERROR,  can't open %s \n",fn_pattern);
      exit(-1);
    }
 printf("The pattern is given by:\n");

 while( ((status = fscanf(fp10,"%f", &(corner[i].x))) != EOF) && (i<CORNER_MAX)
)
     {
       fscanf(fp10,"%f", &(corner[i].y));
       printf(" %.2f %.2f \n",corner[i].x,corner[i].y);
       if(corner[i].x<SAFETY || corner[i].x>xsize-SAFETY
         || corner[i].y<SAFETY || corner[i].y>ysize-SAFETY)
        { printf("Pattern trespasses the SAFETY area\n"); exit(-1); }
       i++;
       /* The corners are supposed to be entered in a counter clockwise way,
for a
         geometry that is homotopious to a circle */
     }
  fclose(fp10);

  ncorner = i;      /* Stores the number of corners */
```

```
  /* make up the edges array */

  /* duplicate the first corner as the last */

  corner[ncorner].x = corner[0].x;
  corner[ncorner].y = corner[0].y;

  /*  cycle through all the other corners */
  for(i=0;  i<ncorner;  i++)
    {
      dx = corner[i+1].x - corner[i].x;
      if( fabs(dx) < toosmall7 )  edge[i].vertical = 1;
      else
       {
         edge[i].vertical = 0;
         dy = corner[i+1].y - corner[i].y;
         aa = dy/dx;
         edge[i].b = corner[i].y - (corner[i].x * aa);
         edge[i].a = aa;
       }
      edge[i].xmax = max(corner[i].x, corner[i+1].x);
      edge[i].xmin = min(corner[i].x, corner[i+1].x);
      edge[i].ymax = max(corner[i].y, corner[i+1].y);
      edge[i].ymin = min(corner[i].y, corner[i+1].y);

    }

return(ncorner);
}

/*-------------------------------------------------------------------------
----------------*/
/* This function fills data[] with all the points (segments and triples) of the
film structure */
int collect_data(Data data[], int* before0)
{
  /*FILE * fish;*/
  int i, triplept, segmt, segmtpts=0;
  float dx, dy;


  /** Cycle through all triple points, to check all segments.  To avoid
   **  checking segments twice, only check those segments which are linked
   **  such that the forward pointer cooresponds to link[0].
   **/
  for(triplept = triple[0].forward,segmtpts = 0;
      triplept != 0;
      triplept = triple[triplept].forward)

    {
      if (triple[triplept].forward == 0) (*before0) = triplept;


      /*  Loop on the 3 boundary's of triple point */
      for(i=0; i<3; i++)    /* Only search the segments once, forward link. */

       if (triple[triplept].neighbor[i] > 0)

         {
           data[segmtpts].x = triple[triplept].x;
           data[segmtpts].y = triple[triplept].y;

           data[segmtpts].index = -triplept;
```

```
          /* This will keep track of the triple point encountered */

          data[segmtpts].go = 1;
          /* This flag indicates when it is not nul that the next data point
             will be actually sequential to this one in the film structure */

          segmtpts++;

          /* Now to obtain segment points, if any exist, loop over the
             segment points along this boundary until triple point is
             reached. */
          segmt = triple[triplept].segment[i];
          while ((segmt > 0) && (segmtpts < 2*(triple_max+segment_max)))
            {
             data[segmtpts].x = segment[segmt].x;
             data[segmtpts].y = segment[segmt].y;

             data[segmtpts].index = segmt;
             /* This will keep track of the segment point encountered, the sign
                is to distinguish between segment points and triple points */

             data[segmtpts].go = 1;


             segmt = segment[segmt].link[0];
             segmtpts++;
            }

          /* Opposite triple point reached when segmt becomes negative */
           segmt = -segmt;

           data[segmtpts].x = triple[segmt].x;
           data[segmtpts].y = triple[segmt].y;
           data[segmtpts].index = -segmt;
           data[segmtpts].go = 0; /*meaning that the next point collected is not
sequential*/

           segmtpts++;


       }  /* End of if, end of the loop over the neighbors of triple points */

      if ( segmtpts == 2*(triple_max+segment_max) )
       {
         printf("ERROR(gb_pattern): function collect_data() reads too many ");
         printf("segment points from the data structures.\n");
         exit(-1);
       }



    }  /*  End of the loop over triple points, data collected */


   /*  Collect the x, y coordinates of the segments points
     encountered along the boundary along with the segment index,
     flipping their x, y coordinates by xsize or ysize as necessary
     First make sure that triple point coordinates are
     wrapped similarly to segment points.  In the past all
     the segment points were unwrapped.
     In this version, all segment point coordinates are kept
     wrapped similarly to avoid this excess computation.
   */
```

```
  for(i=1; i<segmtpts; i++)
    {
      dx = data[i].x - data[i-1].x;
      dy = data[i].y - data[i-1].y;

      if (dx > xsized2 || dx < -xsized2)
       {
         printf("In collect_data(), |data[i].x-data[i-1].x| > xsized2 for
i=%d.\n",i);
         data[i-1].go = 0;
       }
      if (dy > ysized2 || dy < -ysized2)
       {
         printf("In collect_data(), |data[i].y-data[i-1].y| > ysized2 for
i=%d.\n",i);
         data[i-1].go = 0;
       }
    }


  /* Store data in file called data */
  /*if((fish = fopen("data","w"))==NULL)
    {
    printf("Can't open the file data to put in the data array info");
    exit(-1);
    }

    for(i=0;i<segmtpts;i++)
    {
    fprintf(fish,"%d  index=%d  x=%.2f  y=%.2f go=%d\n
",i,data[i].index,data[i].x,data[i].y,
            data[i].go); */
  /*if (inside(data[i],corner,ncorner))
    fprintf(fish,"Inside: %d \n",(inside(data[i],corner,ncorner)));
    else fprintf(fish,"Outside: %d \n",(inside(data[i],corner,ncorner)));*/
  /* }
     fclose(fish);
     */
  return(segmtpts);
}

/*-------------------------------------------------------------------------*/
/* This function gives the intersection between the two structures and stores
   in a file called intersections_struct_name_pattern_name.dat */
int intersect(Data data[],   /* The film structure data */
            int segmtpts,     /* The number of points stored (size of data) */
            Edge edge[],   /* The overlayed structure data */
            int ncorner,
            Data new[],
            Corner corner[])
{
  /*FILE * fish;
  char intersections[300] = DATA_DIR;*/
  int i, j, k, ki, t;
  float answerx, answery;
  Data anew[MAX_INTERSECT_POINTS];

  k=0;

  for(i=0; i<ncorner; i++)
    {
      ki = 0;
```

```
for(j=0; j<segmtpts-1; j++)

{
  if(data[j].go==1)

    if( (data[j].x>=edge[i].xmin || data[j+1].x>=edge[i].xmin)
       && (data[j].x<=edge[i].xmax || data[j+1].x<=edge[i].xmax) )

            if(   (data[j].y>=edge[i].ymin || data[j+1].y>=edge[i].ymin)
        && (data[j].y<=edge[i].ymax || data[j+1].y<=edge[i].ymax) )


       /* one point is inside the boundaries, and we shoud check for
          intersections */

       if(edge[i].vertical==0)   /* edge not vertical */
         {
           if(fabs(edge[i].a*(data[j+1].x-data[j].x)-(data[j+1].y-
                                                 data[j].y))
              >toosmall7)   /* lines not parallel */

             {
             answerx = (data[j].y*data[j+1].x-data[j+1].y*data[j].x-
                       edge[i].b*(data[j+1].x-data[j].x)) /
                         (edge[i].a*(data[j+1].x-data[j].x)-
                          (data[j+1].y-data[j].y));

             answery = edge[i].a*answerx + edge[i].b;

             if(answerx>=edge[i].xmin && answerx<=edge[i].xmax &&
                answery>=edge[i].ymin && answery<=edge[i].ymax &&
                min(data[j].x,data[j+1].x)<=answerx &&
                answerx<=max(data[j].x,data[j+1].x) &&
                min(data[j].y,data[j+1].y)<=answery &&
                answery<=max(data[j].y,data[j+1].y) )

               {
                 anew[ki].x = answerx;
                 anew[ki].y = answery;
                 anew[ki].index = j;
                 /* {data[j], data[j+1]} is the segment
                    that gave birth to this point */

                 anew[ki].go = (i+1) % ncorner;
                 /* gives the index of the next corner */

                 ki = ki+1;
             /*ki is exactly the number of intersections on edge[i]*/
               }
             }
         }

              else  /* edge vertical */
         if(fabs(data[j+1].x-data[j].x)>toosmall7) /* seg not vertical */
           {
             answerx = edge[i].xmin;
             answery = data[j].y  +  (answerx-data[j].x)*
              (data[j+1].y-data[j].y)/(data[j+1].x-data[j].x);

             if(answery>=edge[i].ymin && answery<=edge[i].ymax &&
               min(data[j].x,data[j+1].x)<=answerx &&
               answerx<=max(data[j].x,data[j+1].x) &&
```

```
                    min(data[j].y,data[j+1].y)<=answery &&
                    answery<=max(data[j].y,data[j+1].y) )
                   {
                     anew[ki].x = answerx;
                     anew[ki].y = answery;
                     anew[ki].index = j;
                     /* {data[j],data[j+1]} is the segment
                        that gave birth to this point */

                     anew[ki].go = (i+1) % ncorner;
                     /* gives the index of the coming corner on the edge */

                     ki = ki+1;
                   }
                 }
      }

      if(edge[i].vertical==0)
       {
         if(corner[i].x<corner[i+1].x) sort(anew,ki,by_x);
         else                          sort(anew,ki,by_opp_x);
       }
      else
       {
         if(corner[i].y<corner[i+1].y) sort(anew,ki,by_y);
         else                          sort(anew,ki,by_opp_y);
       }

      for(t=0; t<ki; t++)
       new[k+t] = anew[t];

      k = k + ki;  /* k is the total number of intersection points */
    }

  /* Duplicate the first intersection point as the last */
  new[k] = new[0];


  /* Now store the results in a file called intersections... */
  /*strcat(intersections, "intersections_");
  strcat(intersections, struct_name);
  strcat(intersections, "_");
  strcat(intersections, corners);
  strcat(intersections, ".dat");
  if((fish = fopen(intersections,"w"))==NULL)
    {
      printf("Can't open the file %s ",intersections);
      exit(-1);
    }

  fprintf(fish,"\n\n                          Results of the Patterning
\n\n\n\n");
  fprintf(fish," With a pattern given by the following points (entered in a
counter");
  fprintf(fish,"clockwise way)\n\n\n");
  for(i=0;i<ncorner;i++)
    fprintf(fish,"  corner[%d]: x=%.2f y=%.2f \n\n", i, corner[i].x,
corner[i].y);


  fprintf(fish," The total number of intersection points is %d \n\n",k);
  fprintf(fish," The following data represents the coordinates of the
intersection \n");
```

```
  fprintf(fish,"points as well as the indexes of the extremities of the
segment\n");
  fprintf(fish,"intersecting the pattern at that point, the sign is + for a
segment\n");
  fprintf(fish,"point and - for a triple point\n\n\n");

  for(i=0;i<k;i++)
    {  j = new[i].index;
       fprintf(fish,"new[%d]: x=%.2f, y=%.2f, start@ %d (x=%.3f, y=%.3f),
end@%d (x=%.3f, y=%.3f)\n", i,
               new[i].x, new[i].y, data[j].index,data[j].x, data[j].y,
data[j+1].index,data[j+1].x, data[j+1].y);
     }

  fclose(fish);
  */

  return(k);

}


/*-------------------------------------------------------------------------*/

/* This function gives the minimum of 2 numbers */
float min(float x1, float x2)
{
  float min;
  if(x1 <= x2) min=x1;
  else min=x2;

  return(min);
}


/*-------------------------------------------------------------------------*/

/* This function gives the maximum of 2 numbers */
float max(float x1, float x2)
{
  float max;
  if(x1 >= x2) max=x1;
  else max=x2;

  return(max);
}



/*-------------------------------------------------------------------------*/

/* This functions sorts an array of Data according to a certain key */
void sort(Data * e, int size, int (*key) (Data, Data) )
{
  int pos, index;
  Data value;

  for(pos=1; pos<size; pos++)
    {
      value = e[pos];

      for(index=pos; index>0 && key(e[index-1],value); index--)
       e[index]=e[index-1];
```

```
      e[index] = value;


    }

}

/*-------------------------------------------------------------------------*/

/* the four following functions specify the type of keys needed for sorting */

int by_x(Data d1,Data d2)
{
  return( d1.x > d2.x);
}

int by_opp_x(Data d1,Data d2)
{
  return( d1.x < d2.x);
}

int by_y(Data d1,Data d2)
{
  return( d1.y > d2.y);
}


int by_opp_y(Data d1,Data d2)
{
  return( d1.y < d2.y);
}

/*-------------------------------------------------------------------------
*/
/* The following function updates the different structure arrays,
   triple[], segment[] and grain[], adding the edge intersection points
   as new triple points (duplicating them for the internal pattern and
   the external pattern), the corners of the pattern as new stagnant segment
   points, and creating new grains if a grain is split into different parts
   as a result of the patterning. */


void update_struct(Data * new,
                   Data * data,
                   int newpoints,
                   Corner * corner,
                   int ncorner,
                   int before0)

{

  int i, j, k, mu, nu, index, p, diff;
  int old_p1, p1, p2, muprime, nuprime;
  int newgrain = grain_max; /* will hold the index of newly created grains */
  int flag_full_turn, initial_p2;
  int alpha = triple_max-1;
  int alphaseg = segment_max-1;
  int newseg = segment_max;
  Data temp_data;

  /* First,allocate memory for the 2*newpoints new triple points and the
     2*ncorner new segment points to be created */
```

```
  if ((triple=(triple_point *)
realloc(triple,(triple_max+2*newpoints+1)*sizeof(triple_point))) == NULL)
    {
      printf("ERROR(gb_pattern): Couldn't allocate memory for new triple
points.\n");
      exit(-1);
    }

  if ((grain=(grain_struct *)
realloc(grain,(grain_max+newpoints)*sizeof(grain_struct))) == NULL)
    {
      printf("ERROR(gb_pattern): Couldn't allocate memory for new grains \n");
      exit(-1);
    }

  if ((segment=(segment_point *)
realloc(segment,(segment_max+2*ncorner+1)*sizeof(segment_point))) == NULL)
    {
      printf("ERROR(gb_pattern): Couldn't allocate memory for new segment
points.\n");
      exit(-1);
    }

  /* The strategy is to create new triple points by opening the ring of triple
points
     by modifying the forward pointer of triple[before0],
     then add a first set of INTERIOR triple points indexed from alpha+1 until
     alpha+newpoints,and a second set of EXTERIOR triple points indexed from
     alpha+1+newpoints until alpha+2newpoints. */

  /* now break the used triple points boucle */
  triple[before0].forward = alpha+1;  /* instead of 0 */
  triple[alpha+1].backward = before0;
  triple[0].backward = alpha+2*newpoints; /* instead of before0 */
  triple[alpha+2*newpoints].forward = 0;

  /* now break the not-used triple points boucle */
  triple[alpha+2*newpoints+1].forward = triple_max-1;
  triple[alpha+2*newpoints+1].backward = triple[alpha].backward;
  triple[alpha+2*newpoints+1].type = 0;
  triple[triple[alpha].backward].forward = alpha+2*newpoints+1;
  triple[alpha].backward = alpha+2*newpoints+1;

  /* now break the not-used segment points boucle */
  segment[alphaseg+2*ncorner+1].link[0] = alphaseg;
  segment[alphaseg+2*ncorner+1].link[1] = segment[alphaseg].link[1];
  segment[alphaseg+2*ncorner+1].stop = -1;
  segment[segment[alphaseg].link[1]].link[0] = alphaseg+2*ncorner+1;
  segment[alphaseg].link[1] = alphaseg+2*ncorner+1;


  /* set boundary links for new triple points */
  triple[alpha+1].segment[0] = alphaseg + ncorner;
  triple[alpha+1+newpoints].segment[1] = alphaseg + 2*ncorner;

  /* set the boundary links for the new seg points */
  segment[alphaseg+ncorner].link[0] = -(alpha+1);
  segment[alphaseg+2*ncorner].link[0] = -(alpha+1+newpoints);

   /* Set the initial internal triple point's neighbor[2] */
  index = find(newpoints-1,data,new,corner,ncorner);
  triple[alpha+1].neighbor[2] =
```

```
    inside(data[new[0].index],corner,ncorner) ? -index : index;

   /* Now, loop over all new edge points and do the updating, considering the 2
     different cases of the existence or not of corners between 2 consequent
edge
     triple points. */
  for(i=0;i<newpoints;i++)
     {

      /* Set the label of the internal new edge triple point to create */
      k = alpha + 1 + i;

      /* Create 2 new triple points, one external and one internal */
      /* set the coordinates */
      triple[k].x = new[i].x;
      triple[k].y = new[i].y;

      triple[k+newpoints].x = new[i].x;
      triple[k+newpoints].y = new[i].y;


      /* set the forward and backward pointers */
      triple[k].forward = k+1;
      if(k<alpha+newpoints) triple[k+newpoints].forward = k+newpoints+1;

      if(k>alpha+1) triple[k].backward = k-1;
      triple[k+newpoints].backward = k+newpoints-1;

      printf("\n\n\nProcessing newpoint number: %d ...\n",i);

      /* Find the grain index we're splitting in 2 */
      index=find(i,data,new,corner,ncorner);


      /* Keep the original grain label for the INTERNAL grain, and let its
triple
       point indicator become the new INTERNAL edge triple point to be
created.*/
      grain[index].triple = k;
      grain[index].segment = 1;


      /*Create a new EXTERNAL grain, with the new EXTERNAL edge triple point
       (to be created) as its triple point pointer */
      printf("\n The index of the current grain we're walking is %d\n",index);
      printf("\n The index of the new grain to create is %d\n",newgrain);

      grain[newgrain].triple = k+newpoints;
      grain[newgrain].segment = 2;

      /* set segment pointers and grain pointers */

      triple[k].neighbor[0] = -DOWNEDGE; /* minus due to orientation */
      triple[k+newpoints].neighbor[0] = UPEDGE;

      triple[k].neighbor[1] = index;
      if(k<alpha+newpoints) triple[k+newpoints+1].neighbor[1] = -newgrain;

      /* Compute inside and outside triple/segment pts near new point */
      j   = new[i].index;
      mu  = data[j].index;
      nu  = data[j+1].index;
```

```
      muprime = cycle_back(mu);
      nuprime = cycle_forward(nu);

      /* Updating the new outside grain's boundary to have "newgrain" as a
neighbor */
      if (inside(data[j],corner,ncorner))
       {
         printf("Starting point is INSIDE muprime=p1\n");
         p1 = muprime;
         p2 = nuprime;
       }
      else
       {
         printf("Starting point is OUTSIDE muprime=p2\n");
         p1 = nuprime;
         p2 = muprime;
       }

      /* Case of a lense split in 2 by the edge */
      if(triple[p1].lense == p2)
       {
         /* it's not a lense anymore: */
         triple[p1].lense = 0;
         triple[p2].lense = 0;
         /*printf("Lense Exterminated \n");*/

         for(p=0;p<3;p++)
           {
             if (triple[p2].neighbor[p] ==  index)
              triple[p2].neighbor[p] =  newgrain;
             if (triple[p2].neighbor[p] == -index)
              triple[p2].neighbor[p] = -newgrain;
           }
       }

      else /* General case , No lense split, regular grain */
       {


         /* Updating boundary's triple points neighbor */
         printf("-----------\n");
         printf("start p1 = %d (%.3f,%.3f)\n",p1,triple[p1].x,triple[p1].y);
               printf("start p2 = %d
(%.3f,%.3f)\n",p2,triple[p2].x,triple[p2].y);
         printf("-----------\n");


         initial_p2 = p2;
         flag_full_turn = 0;
          temp_data.x = triple[p2].x; temp_data.y = triple[p2].y;

         if(!inside(temp_data,corner,ncorner) || triple[p2].neighbor[0]==UPEDGE)
           while(!flag_full_turn)
             {

              for(p=0;p<3;p++)
                {
                  if (triple[p2].neighbor[p] ==  index)
                    triple[p2].neighbor[p] =  newgrain;
                  if (triple[p2].neighbor[p] == -index)
                    triple[p2].neighbor[p] = -newgrain;
                }
```

```
            old_p1=p1;
            p1=p2;
            p2=walk_ccw(old_p1,p2);

            if (p2==initial_p2 || p2==alpha+1+newpoints ||
walk_to_j(p1,p2,i,new,data))
              flag_full_turn = 1;

            printf("-when p1=%d p2=%d walk=%d\n",p1, p2,
walk_to_j(p1,p2,i,new,data));
            printf("----------\n");
            printf("new p1 = %d (%.3f,%.3f)\n",p1,triple[p1].x,triple[p1].y);
            printf("new p2 = %d (%.3f,%.3f)\n",p2,triple[p2].x,triple[p2].y);

           }

      }


/*************************************************************************/
     /* Now seperate cases of existence or not of corners between triple[k]
and
       its successor triple point and create new stagnant segment points in the
       second case */
     if( (diff=  ((new[i+1].go-new[i].go)+ncorner) % ncorner) != 0)
       /* diff is the number of intermediate corners */

      {
        for(p=0;p<diff;p++)
          {
            /* create internal stagnant corner-segment points */
            segment[newseg+p].x = corner[new[i].go+p].x;
            segment[newseg+p].y = corner[new[i].go+p].y;
            segment[newseg+p].stop = 1; /* stagnant segment point */


            /* create external stagnant corner-segment points */
            segment[newseg+p+ncorner].x = corner[new[i].go+p].x;
            segment[newseg+p+ncorner].y = corner[new[i].go+p].y;
            segment[newseg+p+ncorner].stop = 1; /* stagnant segment point */



            /* Now create the forward and backward pointers for those points */
            if(p<diff-1) segment[newseg+p].link[0] = newseg+p+1;
            else if(i<newpoints-1) segment[newseg+p].link[0] = -(k+1);

            if(p>0) segment[newseg+p].link[1] = newseg+p-1;
            else segment[newseg+p].link[1] = -k;

            if(p<diff-1) segment[newseg+p+ncorner].link[0] = newseg+p+1+ncorner
;
            else if(i<newpoints-1)
             segment[newseg+p+ncorner].link[0] = -(k+1+newpoints)  ;

            if(p>0) segment[newseg+p+ncorner].link[1] = newseg+p-1+ncorner;
            else segment[newseg+p+ncorner].link[1] = -(k+newpoints);

          }

        /* update triple in that case */
        triple[k+newpoints].segment[0] = newseg+ncorner;
        if(k<alpha+newpoints) triple[k+1].segment[0] = newseg+diff-1;
```

```
        triple[k].segment[1] = newseg;
        if(k<alpha+newpoints)
             triple[k+newpoints+1].segment[1] = newseg+diff-1+ncorner;

        newseg = newseg + diff;
      }


     else
      {
        triple[k+newpoints].segment[0] = -(k+1+newpoints);
        triple[k+1].segment[0] = -k;
         /* minus because triple point, and note that in this case k is
obligatory
           less than alpha+newpoints*/


        triple[k].segment[1] = -(k+1);
        triple[k+newpoints+1].segment[1] = -(k+newpoints);

      }

/*************************************************************************/


     /* Now update the links of the points (segment or triple points) in the
       neighborhood of the 2 new edge triple points (ext and int), also
determine
       triple[k].seg[2] and triple[k].neighbor[2]*/

     if(i<newpoints-1)
      triple[k+1].neighbor[2] =
        inside(data[new[i+1].index],corner,ncorner) ? -index : index;
           /* that has been split at previous step */


     if(inside(data[j],corner,ncorner))
      {

        triple[k].segment[2] = +mu;
            /* if positive segment point, else triple */


        triple[k+newpoints].neighbor[2] = +newgrain;

        triple[k+newpoints].segment[2] = nu;

        /* if positive segment point, else triple */


        /* Now also update the segment links for triple points or seg pts
linked
           to triple[k]*/

        if (mu>0) segment[mu].link[0] = -k; /* data[j] seg point */


        if (mu<0)  /* data[j] trip point */
          for(p=0;p<3;p++)
            if (triple[-mu].segment[p] == nu) triple[-mu].segment[p] = -k;
```

```
        if (nu>0) segment[nu].link[1] = -(k+newpoints); /* data[j+1] seg point
*/

        if (nu<0) /* data[j+1] trip point */
          for(p=0;p<3;p++)
            {
            if (triple[-nu].segment[p]==mu)
              triple[-nu].segment[p] = -(newpoints+k);
            if (triple[-nu].neighbor[p]==index)
              triple[-nu].neighbor[p] = newgrain;
            if (triple[-nu].neighbor[p]==-index)
              triple[-nu].neighbor[p] = -newgrain;
            }
      }

      else /* meaning data[j] outside */
      {

        triple[k].segment[2] = nu;
                      /* if positive segment point, else triple */


        triple[k+newpoints].neighbor[2] = -newgrain;
        triple[k+newpoints].segment[2] = mu;
        /* if positive segment point, else triple */

        /* Now also update the segment links for triple or seg points linked to
           triple[k]*/

        if (mu>0) segment[mu].link[0] = -(k+newpoints);
                                        /* mu>0 means data[j] seg pt */
        if (mu<0)  /* data[j] trip point */
          for(p=0;p<3;p++)
            {
            if (triple[-mu].segment[p]==nu)
              triple[-mu].segment[p] = -(k+newpoints);
            if (triple[-mu].neighbor[p]==index)
              triple[-mu].neighbor[p] = newgrain;
            if (triple[-mu].neighbor[p]==-index)
              triple[-mu].neighbor[p] = -newgrain;
            }


        if(nu>0) segment[nu].link[1] = -k;  /* nu>0 means  data[j+1] seg point
*/
        if (nu<0)  /* data[j+1] trip point */
          for(p=0;p<3;p++)
            if (triple[-nu].segment[p] == mu) triple[-nu].segment[p] = -(k);

      }

   /* Set the different triple point flags */


      /* checking for lenses in the new structure will be done
         after the end of the loop.  For now, set all lense
       flags to be 0.*/

      triple[k].lense = 0;
      triple[k+newpoints].lense = 0;
```

```
      triple[k].xold = triple[k].x;
      triple[k+newpoints].xold = triple[k+newpoints].x;

      triple[k].yold = triple[k].y;
      triple[k+newpoints].yold = triple[k+newpoints].y;

      triple[k].stop = 0;
      triple[k+newpoints].stop = 0;

      triple[k].used = 1;
      triple[k+newpoints].used = 1;

      triple[k].type = 1;
      triple[k+newpoints].type = 1;

      triple[k].conv =0;
      triple[k+newpoints].conv =0;

      triple[k].reor0 =0;
      triple[k+newpoints].reor0 =0;

      triple[k].reor1 =0;
      triple[k+newpoints].reor1 =0;

      triple[k].reor2 =0;
      triple[k+newpoints].reor2 =0;

      triple[k].moved =0;
      triple[k+newpoints].moved =0;

      newgrain = newgrain+1; /* IN THE CASE OF NO CORNERS BETWEEN NEW[I] and
NEW[I+1]
                             ONLY ONE EXTRA GRAIN IS CREATED */
    }

  /* Set the initial external triple point's neighbor[1] */
  triple[alpha+newpoints+1].neighbor[1] = -(newgrain-1);

  /* Check for newly created LENSES */

  /* First, the inside structure */
  for(i=0;i<newpoints-1;i++)
    {
      k = alpha+1+i;
      if(cycle_forward(triple[k].segment[2])==k+1 ||
        cycle_back(triple[k].segment[2])==k+1)
       {
         triple[k].lense = k+1;
         triple[k+1].lense = k;
         if(pnt_stat>1)
           printf("\n The couple (%d,%d) represents a new Internal
LENSE\n",k,k+1);
       }
    }

  if(cycle_forward(triple[alpha+newpoints].segment[2])==alpha+1 ||
     cycle_back(triple[alpha+newpoints].segment[2])==alpha+1)
    {
      triple[alpha+newpoints].lense = alpha+1;
      triple[alpha+1].lense = alpha+newpoints;
      if(pnt_stat>1)
       printf("\n The couple (%d,%d) represents a new Internal LENSE\n",
```

```
            alpha+1,alpha+newpoints);
    }

  /* Second, the outside structure */
  for(i=0;i<newpoints-1;i++)
    {
      k = alpha+1+newpoints+i;
      if(cycle_forward(triple[k].segment[2])==k+1 ||
        cycle_back(triple[k].segment[2])==k+1)
       {
         triple[k].lense = k+1;
         triple[k+1].lense = k;
         if(pnt_stat>1)
           printf("\n The couple (%d,%d) represents a new External
LENSE\n",k,k+1);
       }
    }

  if(cycle_forward(triple[alpha+2*newpoints].segment[2])==alpha+1+newpoints ||
     cycle_back(triple[alpha+2*newpoints].segment[2])==alpha+1+newpoints)
    {
      triple[alpha+2*newpoints].lense = alpha+1+newpoints;
      triple[alpha+1+newpoints].lense = alpha+2*newpoints;
      if(pnt_stat>1)
       printf("\n The couple (%d,%d) represents a new External lense\n",
              alpha+1+newpoints,alpha+2*newpoints);
    }



  /* Now store the 3 new structure arrays in 3 new structure files */
  triple_max = triple_max + 2*newpoints+1;   /* 1 is added for linking the un-
used triples */
  segment_max = segment_max + 2*ncorner+1;
  grain_max = grain_max + newpoints;

}

/*-------------------------------------------------------------------------*/
/* This set of functions checks to see if a data point is inside or outside the
shape */

int CCW(Data p0, Data p1,Data p2)
{
 float dx1,dx2,dy1,dy2;
 dx1=p1.x-p0.x;dy1=p1.y-p0.y;
 dx2=p2.x-p0.x;dy2=p2.y-p0.y;
 if (dx1*dy2>dy1*dx2) return 1;
 if (dx1*dy2<dy1*dx2) return -1;
 if ((dx1*dx2<0) || (dy1*dy2<0)) return -1;
 if ((dx1*dx1+dy1*dy1)<(dx2*dx2+dy2*dy2)) return 1;

 return 0;
}

int intersection (Data p1, Data p2, Data p3, Data p4)
{
 return((CCW(p1,p2,p3)
        *CCW(p1,p2,p4))<=0)
     &&((CCW(p3,p4,p1)
        *CCW(p3,p4,p2))<=0);
}
```

```
int inside (Data p, Corner * poly, int size)
{
 float P=3.141529;

 int n=size;
 int count=0;
 Data  p1, p2, p3, p4;
 int ii, i;
 int jj=0;
 p1.x=p.x;
 p1.y=p.y;

 p2.x=p1.x + 20000*cos(P/6.0);

 p2.y=p1.y + 20000*sin(P/6.0);

 for (i=1; i<=n; i++)
   {
    ii=i-(i/n)*n; /*printf("ii=%d\n",ii);*/

    p3.x=poly[ii].x;
    p3.y=poly[ii].y;
    p4.x=p3.x;
    p4.y=p3.y;
    if (!intersection(p1,p2,p3,p4))
        {
         p4.x=poly[jj].x;
         p4.y=poly[jj].y;
        jj=i;
         if (intersection(p1,p2,p3,p4)) count++;
        }
  }
 return (count & 1);
}

/*-------------------------------------------------------------------------*/
/* This function gives, for a given new edge-triple point new[i], the label
   of the grain devided in two by the segment (new[i], new[i+1]) */
int find(int i, Data* data, Data* new,Corner* corner, int ncorner)
{
  int j,mu,nu,muprime,nuprime,third,p,x=0,y=0;
  int a[6] = {0,0,0,0,0,0};


  j = new[i].index;
  mu = data[j].index;
  nu = data[j+1].index;
  muprime = cycle_back(mu);
  nuprime = cycle_forward(nu);


  /* First, deal with the case of a LENSE split in two by the edge */
  if(triple[muprime].lense == nuprime)
    {

      /* Walks from muprime to nuprime, records grain neigbhors */
      for(x=0, p=0;p<3;p++,x++)
       if ((cycle_forward(triple[muprime].segment[p])==nuprime) ||
            (cycle_back(triple[muprime].segment[p])==nuprime))

         a[x]=abs(triple[muprime].neighbor[p]);

      /* Walks from nuprime to muprime, records grain neigbhors */
```

```
      for(x=0, p=0;p<3;p++,x++)
       if ((cycle_forward(triple[nuprime].segment[p])==muprime) ||
           (cycle_back(triple[nuprime].segment[p])==muprime))
         a[x+3]=abs(triple[nuprime].neighbor[p]);

      /* Return grain neighbor in common from both walks above */
      for(x=0; x<3; x++)
       for(y=0; y<3; y++)
         if ((a[x] == a[y+3]) &&
             (a[x] != 0))
           return(a[x]);

      printf("ERROR: find() cannot find common lense grain label\n");
      exit(0);

    }

  /* Else, not the first side of a LENSE split in 2, regular case */
  if(inside(data[j],corner,ncorner)) third = walk_ccw(muprime,nuprime);

  else third = walk_ccw(nuprime,muprime);

  return(commun(muprime,nuprime,third));
}

/*-------------------------------------------------------------------------*/
/* This function is to walk on a grain counter clockwise, it gives for 2
   consecutive triple points start and n the third consecutive triple point
   when walking counter-clockwise on the grain boundaries.
   In the case of a LENSE between start and n, this function returns the
   other triple point n is linked to.
   In the case of a LENSE between n and another triple point, this function
   returns the label of that triple point */
int PREVIOUS_walk_ccw(int start,int n)
{
  extern triple_point * triple;
  float v0[3], v1[3], v2[3];
  int j, index, tp1, tp2, tp3, p;

  j = start;
  index = n;

  tp1 = get_index(index, 0);
  tp2 = get_index(index, 1);
  tp3 = get_index(index, 2);

  get_tp1_tp2(j, &tp1, &tp2, tp3);

  /* Treat first the case of a lense which occurs in these cases:
     if (tp1 = j) or (tp2 = j), then there is a lense between
     j and index.
     if (tp1=tp2 != j) then there is a lense between index and tp1 */
  /*printf("\n &&&tp1=%d, tp2=%d, start=%d index=%d \n",tp1,tp2,start,index);*/
  if (tp1==j) return(tp2);
  if(tp2==j) return(tp1);

  if(tp1==tp2) return(tp1);

  /* Else,  */
  /* Now create 3 vectors from 3 segments emanating from triple point */
  /* index.  These vectors will be crossed for the purpose of */
  /* determining which direction is ccw */
```

```
  for(p=0;p<3;p++)
    if(cycle_forward(triple[index].segment[p])==j ||
       cycle_back(triple[index].segment[p])==j)
      j = triple[index].segment[p];

  if(j<0)
    {
      j=-j;
      v0[1] = triple[j].x - triple[index].x;
      v0[2] = triple[j].y - triple[index].y;
    }
  else
    {
      v0[1] = segment[j].x - triple[index].x;
      v0[2] = segment[j].y - triple[index].y;
    }

  v1[1] = triple[tp1].x - triple[index].x;
  v1[2] = triple[tp1].y - triple[index].y;
  v2[1] = triple[tp2].x - triple[index].x;
  v2[2] = triple[tp2].y - triple[index].y;

  j = index;   /* THIS LINE OF CODE IS NOT NECESSARY */


  if (cross(v0, v1) <= 0)
    {
      if(cross(v0, v2) > 0) index = tp1;
      else
       if(cross(v1, v2) >= 0) index = tp2;
       else index = tp1;
    }


  else
    {
      if(cross(v0, v2) < 0) index = tp2;
      else
       if(cross(v1, v2) >= 0)  index = tp2;
       else index = tp1;
    }


  /*printf("walk_ccw(%d, %d)=%d\n",start,n,index);*/
  return(index);
}


int get_index(int i, int n)
{

  if (triple[i].neighbor[n] > 0)
    return(cycle_forward(triple[i].segment[n]));
  else
    return(cycle_back(triple[i].segment[n]));

}


void get_tp1_tp2(int i, int *t1, int *t2, int t3)
{

  /* printf("\nstart, tp1 tp2 tp3  are %d %d %d %d", i, *t1, *t2, t3);*/
```

```
  if( *t1==*t2 || *t1==t3 || *t2==t3 )
    {
   /* printf("\n Function get_tp1_tp2(), some arguments are identical\n");
      printf(" There is a LENSE \n");*/
    }


  if(*t1==*t2==t3)
    {
      printf("\n Error in function get_tp1_tp2()");
      printf("\n all arguments are identical\n");
      exit(-1);
    }


  if (*t1 == i)
    {
      *t1 = *t2;
      *t2 = t3;
    }
  else if (*t2 == i)
    *t2 = t3;
  else if (t3 != i)
    {
      printf("\nERROR in function get_tp1_tp2() \n");
      printf("j, tp1, tp2, or tp3 are incorrect \n");
      printf("j, t1, t2, t3 are %i %i %i %i\n", i, *t1, *t2, t3);
      exit(-1);
    }
}

/*-------------------------------------------------------------------------*/
/* This function is to walk on a grain counter clockwise, it gives for 2
   consecutive triple points start and n the third consecutive triple point
   when walking counter-clockwise on the grain boundaries.
   In the case of a LENSE between start and n, this function returns the
   other triple point n is linked to.
   In the case of a LENSE between n and another triple point, this function
   returns the label of that triple point */
int walk_ccw(int start,int n)
{
  extern triple_point * triple;
  float v[3][3];
  int j, index, tp[3], sp[3], p;

  j = start;
  index = n;

  for(p=0;p<3;p++)
    {
      tp[p] = get_index(index, p);
      sp[p] = triple[index].segment[p];
    }


/* Find which of the tp's is j and change them and the sp's to get
   tp[0]=j and sp[0] accordingly */
  if(j==tp[1]) {tp[1]=tp[0]; sp[1]=sp[0]; sp[0]=triple[index].segment[1];
tp[0]=j;}
  else if(j==tp[2]) {tp[2]=tp[0]; sp[2]=sp[0]; sp[0]=triple[index].segment[2];
tp[0]=j; }
```

```
  else if(j!=tp[0])
    {
      printf("ERROR in walk_ccw(); input triples not linked\n");
      exit(-1);
    }


/* First, treat the case of a lense between start(j) and n(index),
   in which case jump over it */
  if(j==tp[1])
    {
      if(triple[j].lense!=index)
       {
         printf("in walk_ccw(), didn't show existence of a lense, and it
should.\n");
         exit(-1);
       }
      return(tp[2]);
    }

  else if(j==tp[2])
    {
      if(triple[j].lense!=index)
       {
         printf("in walk_ccw(), didn't show existence of a lense, and it
should.\n");
         exit(-1);
       }
      return(tp[2]);
    }


  /* Treat the case of a lense between n(index) and tp[1], in which case
tp[1]==tp[2] */
  else if(tp[1]==tp[2]) return(tp[1]);


  /* Else,  */
  /* Now create 3 vectors from 3 segments emanating from triple point */
  /* index.  These vectors will be crossed for the purpose of */
  /* determining which direction is ccw */

  for(p=0;p<3;p++)
  if(sp[p]<0)
    {
      sp[p]=-sp[p];
      v[p][1] = triple[sp[p]].x - triple[index].x;
      v[p][2] = triple[sp[p]].y - triple[index].y;
    }
  else
    {
      v[p][1] = segment[sp[p]].x - triple[index].x;
      v[p][2] = segment[sp[p]].y - triple[index].y;
    }



  if (cross(v[0], v[1]) <= 0)
    {
      if(cross(v[0], v[2]) > 0) index = tp[1];
      else
       if(cross(v[1], v[2]) >= 0) index = tp[2];
       else index = tp[1];
```

```
    }


  else
    {
      if(cross(v[0], v[2]) < 0) index = tp[2];
      else
       if(cross(v[1], v[2]) >= 0)  index = tp[2];
       else index = tp[1];
    }


  /*printf("walk_ccw(%d, %d)=%d\n",start,n,index);*/
  return(index);
}

float cross(float vec1[], float vec2[])
{
  float crossproduct;

  crossproduct = vec1[1]*vec2[2] - vec1[2]*vec2[1];
  return(crossproduct);
}

int cycle_forward(int segnum)
{
  if (segnum < 0)  {
    segnum = -segnum;
    return (segnum);

  }
  else  {
    while (segment[segnum].link[0] > 0)
      segnum = segment[segnum].link[0];
    return (-segment[segnum].link[0]);
  }
}


int cycle_back(int segnum)
{
  if (segnum < 0)  {
    segnum = -segnum;
    return (segnum);
  }
  else  {
    while (segment[segnum].link[1] > 0)
      segnum = segment[segnum].link[1];
    return (-segment[segnum].link[1]);
  }
}

/* This function gives the grain label commun to 3 given triple points */
int commun(int t1, int t2, int t3)
{
  int i;

  for(i=0;i<3;i++)
    if ( ((abs(triple[t1].neighbor[i])==abs(triple[t2].neighbor[0])) ||
         (abs(triple[t1].neighbor[i])==abs(triple[t2].neighbor[1])) ||
          (abs(triple[t1].neighbor[i])==abs(triple[t2].neighbor[2])))
          &&
        ((abs(triple[t1].neighbor[i])==abs(triple[t3].neighbor[0])) ||
```

```
        (abs(triple[t1].neighbor[i])==abs(triple[t3].neighbor[1])) ||
        (abs(triple[t1].neighbor[i])==abs(triple[t3].neighbor[2])))
      )
     return(abs(triple[t1].neighbor[i]));

  printf("\n Error gb_pattern, there is no commun grain neighbor to %d, %d, %d
\n", t1, t2, t3);

  printf("t1 neighbors: ");
  printf("%d ",abs(triple[t1].neighbor[0]));
  printf("%d ",abs(triple[t1].neighbor[1]));
  printf("%d ",abs(triple[t1].neighbor[2]));

  printf(", t2 neighbors: ");
  printf("%d ",abs(triple[t2].neighbor[0]));
  printf("%d ",abs(triple[t2].neighbor[1]));
  printf("%d ",abs(triple[t2].neighbor[2]));

  printf(", t3 neighbors: ");
  printf("%d ",abs(triple[t3].neighbor[0]));
  printf("%d ",abs(triple[t3].neighbor[1]));
  printf("%d\n",abs(triple[t3].neighbor[2]));
  exit(-1);
}

int walk_to_j(int p1, int p2, int i, Data *new, Data* data)
{
  int walk_index;
  int p;

  for(p=0;p<3;p++)
    if (cycle_forward(triple[p1].segment[p]) == p2)    /*from p1 to p2*/
      for(walk_index=triple[p1].segment[p];
        1 == 1;
        walk_index = segment[walk_index].link[0])  /* walk to next segment */
      {
        printf("** %d **",walk_index);

        if(walk_index == -p2) return(0);
        else if(walk_index==data[new[i+1].index].index ||
              walk_index==data[new[i+1].index+1].index)  return(1);
        else if(walk_index < 0) return(0);
      }

  for(p=0;p<3;p++)
    if (cycle_back(triple[p1].segment[p]) == p2) /* from p1 to p2 */
      for(walk_index=triple[p1].segment[p];
        1 == 1;
        walk_index = segment[walk_index].link[1])  /* walk to next segment */
      {
        printf("&& %d &&",walk_index);
        if(walk_index == -p2) return(0);
        else if(walk_index==data[new[i+1].index].index ||
              walk_index==data[new[i+1].index+1].index)  return(1);
        else if(walk_index < 0) return(0);
      }

  return(0);

}

/*
This routine initializes the variables which keep track of CPU
```

```
time used by the run.  Also keeps track of total elasped time.
*/
int init_times2()
{
  extern struct tms cpu_start,cpu_current;
  extern time_t current_time,start_time,time_diff;

  times(&cpu_start);
  start_time = time(&start_time);

  return(1);
}


/* This routine cycles through the triple points, trashing the inner or outer
struct
   dependent on the value of flag: 1 we keep inside, 0 for external struct */
/* returns 1 if ok
**       0 if error*/
int keep_in_or_out(int flag, int *ncorner, int number_pats)
{
  Data temp;
  int triplept, old;
  int flag_count;
  int count;
  int j;

  triplept = triple[0].forward;


  if(flag==1)
    {
      while(triplept != 0)
       {
         temp.x = triple[triplept].x;
         temp.y = triple[triplept].y;
         flag_count=0;

         for(count=0;count<number_pats;count++)
           if( (!inside(temp,corner[count],ncorner[count]) ||
               abs(triple[triplept].neighbor[0])==UPEDGE)
              && (abs(triple[triplept].neighbor[0])!=DOWNEDGE) ) flag_count++;

         if(flag_count==number_pats)
           {
             old = triplept;
             triplept = triple[triplept].forward;
             if(put_triple(old)==0) return(0);

             /* Also, make sure to put the triple point descriptor to 0 for any
grain associated
                with old */
             for (j=0;j<grain_max;j++)
              if (grain[j].triple == old)
                grain[j].triple = 0;
           }
         else triplept = triple[triplept].forward;
       }
    }

  else if(flag==0)
    {
      while(triplept != 0)
```

```
      {
        temp.x = triple[triplept].x;
        temp.y = triple[triplept].y;
        flag_count=0;

        for(count=0;count<number_pats;count++)
          if( inside(temp,corner[count],ncorner[count]) ||
             abs(triple[triplept].neighbor[0])==DOWNEDGE )
            flag_count=1;

        if(abs(triple[triplept].neighbor[0])==UPEDGE)  flag_count=0;
        if(flag_count==1)
          {
            old = triplept;
            triplept = triple[triplept].forward;
            if(put_triple(old)==0) return(0);

            /* Also, make sure to put the triple point descriptor to 0 for any
grain associated
              with old */
            for (j=0;j<grain_max;j++)
             if (grain[j].triple == old)
               grain[j].triple = 0;
          }
        else triplept = triple[triplept].forward;
      }
    }


  return(1);
}
```

Appendix C

PolySeg Microstructure Extraction Program

C.1 Program Components and Usage

PolySeg is designed to extract the microstructure obtained with GGSim using the *etch* or *pattern* programs and evolved using *anneal* and map it into a diffusivity information file suitable for input to the electyromigration simulation MIT/EmSim. It features three capabilities. The sub-routine *polyseg* (for simplification, we called the whole extraction program after this main subroutine) assesses the line microstructure in terms of polygranular segments with a high, uniform diffusivities and bamboo segments with a smaller diffusivity, as has been done in the past [Kno 97]. The sub-routine *diff_info* extracts the line's microstructure following the technique outlined above and in accordance with (6.1). Finally, *polyseg_tree*, is a program that allows the same grain structure interpretation for a tree-interconnect as *polyseg* allows for lines.

The routines *polyseg* and *diff_info* are available in stand-alone versions but have also been incorporated, through modifications of the file *statistics.c*, into GGSim as Case 18 and Case 180, respectively. Running them can be performed by including the task number in a command file as described in Appendix B, and will produce the output files named "cluster2.dat" containing the cluster length distribution information and "diffdata" which serves as the diffusivity information input file of MIT/EmSim. In contrast,

polyseg_tree is available only as a stand-alone program. A proper usage of polyseg_tree is demonstrated in the following command line:

```
polyseg_tree pattern_dir diffdata1
```

where "pattern_dir" refers to the directory where the patterned structure, obtained using GGSim, is located, and "diffdata1" is the name we wish to give to the output file which will serve as an input to MIT/EmSim. When this step is achieved, MIT/EmSim can be executed by running a simple command line such as:

```
emsim119i tree.inp geometry diffdata1 ttf.dat
```

where "tree.inp" and "geometry" are other MIT/EmSim input files which can either be created by the user or produced by ERNI, the circuit layout extraction tool. We refer the reader to [ems 98] for more details. "ttf.dat" is the name of the file output by MIT/EmSim containing the time to failure information.

C.2 List of Code

The following is a listing of *polyseg*, the subroutine of statistics.c.

```
/*******************************WALID DEC 97*******************************/

/* Outputs all Clusters and Bamboo lenghts in the form of 2 arrays to the file
CLUSTER2, and the file DIFFDATA (for input into MIT/EmSim), WALID DEC 97 */
int polyseg(char *filename)
{
  void rotate(int, float*, float*);
  void calc_seg_data(int, polygranular_seg*, float*, float*, float*, float*);
  float check_ccw(int, float*, float*);
  float check_cw(int, float*, float*);
  int abs_val(int);


  FILE  *fcluster;

  polygranular_seg* cluster;
  float *xmin, *xmax, *xtrip, *ytrip;
  float theta, linewidth, ltot;
  int i=0, j, begin=-1, cnum=1;
```

```
int segmt, forward, trip_op, n_vb;

/* Open the cluster output file. */
if ( (fcluster=fopen(filename,"a")) == NULL)
{ printf("ERROR(gb_statistics): POLYSEG can't open %s.\n",filename);
  return(-1); }

theta = atan(1.0 / (float) n_regions);
linewidth = (xsize * cos(theta)) / (float) n_regions;
ltot = (xsize * xsize) / linewidth;


fprintf(fcluster,"%.3f %f %f", tau, linewidth, ltot);

fprintf(fdiffdata,"STRAND1\n");
fprintf(fdiffdata,"%.4f\n",  diff_ratio);




/* Allocate memory for the cluster temporary arrays. */
if ( (xmin=malloc(grain_max*sizeof(float))) == NULL)
{ printf("ERROR(gb_statistics): Couldn't allocate memory for xmin[].\n");
  return(-1); }
if ( (xmax=malloc(grain_max*sizeof(float))) == NULL)
{ printf("ERROR(gb_statistics): Couldn't allocate memory for ymin[].\n");
  return(-1); }
if ( (xtrip=malloc(triple_max*sizeof(float))) == NULL)
{ printf("ERROR(gb_statistics): Couldn't allocate memory for xtrip[].\n");
  return(-1); }
if ( (ytrip=malloc(triple_max*sizeof(float))) == NULL)
{ printf("ERROR(gb_statistics): Couldn't allocate memory for ytrip[].\n");
  return(-1); }
/* xtrip[i] and ytrip[i] will contain the rotated triple[i].x and .y
   in the stripped line. */

if ( (cluster=malloc(grain_max*sizeof(polygranular_seg))) == NULL)
{ printf("ERROR(gb_statistics): Couldn't allocate memory for cluster.\n");
  return(-1); }

for (i = 0; i < grain_max; i++)  {
  xmin[i] = -9.9;
  xmax[i] = -9.9;
}

for(i=1;i<triple_max;i++)
  if(triple[i].used) rotate(i, xtrip, ytrip);  /* condition added by walid */

/* WALID FEB 98 ADDED THIS FOLLOWING PORTION TO MONITOR THE NUMBER OF VERTICAL
BOUNDARIES
   which can be related to the NUMBER of SPLITTINGS */
n_vb=0;
{
  for(i=triple[0].forward;i!=0;i=triple[i].forward)
    if(edge(i)==-1) /* Bottom Edge triple */
     {
       segmt = triple[i].segment[2];
       if ( triple[i].neighbor[2] > 0 )
         forward = 0;  else forward = 1;
       while (segmt > 0)
```

```
            segmt = segment[segmt].link[forward];
          trip_op = -segmt;

          if( edge(trip_op)==1 )  /* Top Edge triple */
            {
              n_vb++;

               /* Temporary, print abcissa of the bamboo boundary, DANGEROUS
                 because this file is intended for other info...WALID

              fprintf(fcluster, " %.3f\n ", xtrip[i]);
               */
            }
        }
    }

    /* Output the Number of Vertical Boundaries at this time tau */
    fprintf(fcluster, " %d\n ", n_vb);
    /* WALID, OVER */



    /* Find the first listed bottom edge triple point which is also the */
    /* beginning of a cluster in the triple pt file.  This will then become */
    /* the first point in a loop over all bottom edge triple points. */

    printf("Starting search for first edge triple pt\n");

    for (i = 1; begin < 0; i++)
      {
        if (abs_val(triple[i].neighbor[0]) == 6543210     /* Triple pt is on
bottom edge */
          && triple[i].used) /* WALID */
         {
           xmin[cnum] = check_ccw(i, xtrip, ytrip);
           if (xmin[cnum] > -9.0)
             {
               begin = i;
               i--;
             }
         }

        if (i == triple_max) /* This line is fully bamboo */
         {
           begin = 0;
           i--;
         }
      }

    i = abs_val(triple[i].segment[1]);

    printf("Starting loop to find cluster characteristics\n");

    while ((begin > 0) && (i != begin))
      {
        if (xmin[cnum] < -9.0)       /* if minimum has not yet been found */
         xmin[cnum] = check_ccw(i, xtrip, ytrip);

        xmax[cnum] = check_cw(i, xtrip, ytrip);     /* Always check for xmax */

        if (xmax[cnum] > -9.0)   /* xmax found */
         {
           if (cnum > 1)
```

```
          calc_seg_data(cnum, cluster, xmax, xmin, xtrip, ytrip);
        /* xmaxlast unknown if cnum = 1 */

        cnum++;
      }

     i = abs_val(triple[i].segment[1]);

   }


 /* Now calculate the cluster data for the first cluster */

 xmax[0] = xmax[cnum - 1];
 calc_seg_data(1, cluster, xmax, xmin, xtrip, ytrip);

 for (j = 1; j < cnum; j++)  {
   if(cluster[j].length>600) /* little check to remove possible wrong data */
     { cluster[j].length=0.0;
     printf("PROBLEM j=%d, xmin=%.3f, xmax=%.3f\n", j, xmin[j], xmax[j]);}
   /*WALID HERE*/


   if (xmin[j] > xmax[j])
     {
     cluster[j].length = 0.0;
     }
   /*ltot_clus = ltot_clus + cluster[j].length;*/
 }
 /*
 printf("linewidth = %f\n", linewidth);
 printf("total length = %f\n", ltot);

 printf("With d50=%.2f\n",d50);
 printf("w/d50 = %.2f\n", linewidth/d50);
 printf("ltot/d50 = %f\n\n", ltot/d50);

 printf("number of clusters %d\n", cnum-1);
 printf("number of clusters/(ltot/d50) = %.3f\n",
        ((double)(cnum-1))/(ltot/d50));
 printf("ltot_clus=%.3f\n", ltot_clus);
 printf("ltot_clus/ltot = %.3f\n",ltot_clus/ltot);
 printf("Av_clus_length/d50 = %.3f\n", (ltot_clus/(cnum-1))/d50);
 */

 for (j = 1; j < cnum; j++)
   {
    fprintf(fcluster, "%.2f %.2f\n", cluster[j].length, cluster[j].spacing);
    fprintf(fdiffdata, "1.0 %.2f %.2f\n", xmin[j], xmax[j]);
   }

 fprintf(fcluster, "-1\n");
 fprintf(fdiffdata, "-1\n");

 return(1);
}
```

The following is the list of the subroutines used by *polyseg*.

```
void rotate(int i, float* xtrip, float* ytrip)
{

  int abs_val(int);

  float x, y, ypivot, amtan, xtan, dx, dy;
  float linewidth, theta, slope, yint, inc;
  float  r, phi, diff, del;
  int region;

  region = 1;

  x = triple[i].x;
  y = triple[i].y;

  /* Make sure that all x and y values lie in the 'box' defined by */
  /* amax (the length of a side of the 'box') */

  while (x > xsize)
    x -= xsize;
  while (y > xsize)
    y -= xsize;
  while (x < 0)
    x += xsize;
  while (y < 0)
    y += xsize;


  linewidth = (xsize * cos(theta)) / (float) n_regions;
  theta = atan(1.0 / (float) n_regions);

  slope = 1.0 / (float) n_regions;
  inc = xsize / (float) n_regions;
  amtan = xsize / tan(theta);
  xtan = x / tan(theta);


  yint = y - x*slope;

  /* The if part of the next if..then..else statement determines which */
  /* pivot point to use when rotating the x and y co-ordinates of non- */
  /* edge triple points */

  if ((abs_val(triple[i].neighbor[0]) != 6543210) &&
(abs_val(triple[i].neighbor[0]) != 6543212))
    if (yint < 0)  {
      if (y + xtan > amtan)  {      /* Small slice in first region */
        ypivot = 0.0;
        x -= xsize;
      }
      else  {                        /* Large slice in last region */
        ypivot = inc * ( (float) n_regions - 1.0);
        y += xsize;
        region = n_regions;
      }
    }
    else  {
      while (yint > inc * (float) region)   /* Determine relevant region */
        region++;
```

```
      if ( y + xtan > amtan + inc * ( (float) region - 1.0))  {
        ypivot = inc * (float) region;
        x -= xsize;              /* In small slice, so increment region */
        region++;

      }
      else
        ypivot = inc * ( (float) region - 1.0); /* In big part of region */
    }

  /* The else part of the if..then..else statement determines ypivot */
  /* for triple points on the line edge */

  else  {

    diff = yint - (inc * ( (float) region - 1.0));
    while (fabs(diff) > 0.0005)  {
      region++;
      diff = yint - (inc * ( (float) region - 1.0));
    }

    ypivot = inc * ( (float) region - 1.0);

    if (region == 1)  {
      if (y + xtan > amtan)  {      /* Small segment on first line */
        if (abs_val(triple[i].neighbor[0]) == 6543212)   /* Top edge point */
          x -= xsize;
      }
      else  {                      /* Large segment on first line */
        if (abs_val(triple[i].neighbor[0]) == 6543212)  {   /* Top edge point
*/
          y += xsize;
          ypivot = xsize - inc;
          region = n_regions;
        }
      }
    }
    else  {
      if ( y + xtan > amtan + inc * ( (float) region - 1.0)) {
        if (abs_val(triple[i].neighbor[0]) == 6543212)    /* Top edge point */
          x -= xsize;
      }
      else  {       /* Large segment on nth region */
        if (abs_val(triple[i].neighbor[0]) == 6543212)  {    /* Top edge point
*/
          ypivot -= inc;
          region--;
        }
      }
    }
  }


  dx = x;
  dy = y - ypivot;
  r = sqrt(pow(dx,2.0) + pow(dy,2.0));
  if (x < 0)  {
    dx = -dx;
     phi = atan(dy / dx);
     phi = PI - atan(dy / dx);
  }
  else
    phi = atan(dy / dx);
```

```
  phi -= theta;
  del = (sqrt(pow(xsize,2.0) + pow(inc,2.0))) * ( (float) region - 1.0);
  xtrip[i] = r * cos(phi) + del;

  ytrip[i] = r * sin(phi) /*  - linewidth * ( (float) region - 1.0) */;
  if (abs_val(triple[i].neighbor[0]) == 6543210)
    ytrip[i] = 0.0;
  else if (abs_val(triple[i].neighbor[0]) == 6543212)
    ytrip[i] = linewidth;

}



void rotate_seg(int i, float* xseg, float* yseg)
{

  int abs_val(int);

  float x, y, ypivot, amtan, xtan, dx, dy;
  float linewidth, theta, slope, yint, inc;
  float  r, phi, del;
  int region;

  region = 1;

  x = segment[i].x;
  y = segment[i].y;

  /* Make sure that all x and y values lie in the 'box' defined by */
  /* amax (the length of a side of the 'box') */

  while (x > xsize)
    x -= xsize;
  while (y > xsize)
    y -= xsize;
  while (x < 0)
    x += xsize;
  while (y < 0)
    y += xsize;


  linewidth = (xsize * cos(theta)) / (float) n_regions;
  theta = atan(1.0 / (float) n_regions);

  slope = 1.0 / (float) n_regions;
  inc = xsize / (float) n_regions;
  amtan = xsize / tan(theta);
  xtan = x / tan(theta);


  yint = y - x*slope;

  /* The if part of the next if..then..else statement determines which */
  /* pivot point to use when rotating the x and y co-ordinates */

    if (yint < 0)  {
      if (y + xtan > amtan)  {      /* Small slice in first region */
        ypivot = 0.0;
        x -= xsize;
      }
      else  {                        /* Large slice in last region */
```

```
        ypivot = inc * ( (float) n_regions - 1.0);
        y += xsize;
        region = n_regions;
      }
    }
    else  {
      while (yint > inc * (float) region)   /* Determine relevant region */
        region++;

      if ( y + xtan > amtan + inc * ( (float) region - 1.0))  {
        ypivot = inc * (float) region;
        x -= xsize;               /* In small slice, so increment region */
        region++;

      }
      else
        ypivot = inc * ( (float) region - 1.0); /* In big part of region */
    }


  dx = x;
  dy = y - ypivot;
  r = sqrt(pow(dx,2.0) + pow(dy,2.0));
  if (x < 0)  {
    dx = -dx;
     phi = atan(dy / dx);
     phi = PI - atan(dy / dx);
  }
  else
    phi = atan(dy / dx);

  phi -= theta;
  del = (sqrt(pow(xsize,2.0) + pow(inc,2.0))) * ( (float) region - 1.0);
  *xseg = r * cos(phi) + del;

  *yseg = r * sin(phi) /*  - linewidth * ( (float) region - 1.0) */;

}



float check_ccw(int j, float* xtrip, float* ytrip)
{
  int abs_val(int);
  int get_index(int, int);
  void get_tp1_tp2(int, int *, int *, int);
  float cross(float [], float []);

  float v0[3], v1[3], v2[3];
  float xmin;
  int start, index, tp1, tp2, tp3;
  float ltot;


  ltot = xsize * sqrt( 1.0 + (float) n_regions*n_regions);

  start = j;
  index = get_index(j, 2);

  if (triple[index].type != 1)  {
    while (triple[index].type != 1)  {
      tp1 = get_index(index, 0);
```

```
      tp2 = get_index(index, 1);
      tp3 = get_index(index, 2);
      get_tp1_tp2(j, &tp1, &tp2, tp3);

      /* Now create 3 vectors from 3 segments emanating from triple point */
      /* index.  These vectors will be crossed for the purpose of */
      /* determining which direction is ccw */

      v0[1] = xtrip[j] - xtrip[index];
      v0[2] = ytrip[j] - ytrip[index];
      v1[1] = xtrip[tp1] - xtrip[index];
      v1[2] = ytrip[tp1] - ytrip[index];
      v2[1] = xtrip[tp2] - xtrip[index];
      v2[2] = ytrip[tp2] - ytrip[index];

      /* Deal with the wrapping at the end of the line */
      if(v0[1]>ltot/2) v0[1] = v0[1]-ltot;
      if(v1[1]>ltot/2) v1[1] = v1[1]-ltot;
      if(v2[1]>ltot/2) v2[1] = v2[1]-ltot;

      if(v0[1]<(-ltot/2)) v0[1] = v0[1]+ltot;
      if(v1[1]<(-ltot/2)) v1[1] = v1[1]+ltot;
      if(v2[1]<(-ltot/2)) v2[1] = v2[1]+ltot;



      if (cross(v0, v1) < 0)  {
        j = index;
        index = tp1;
      }
      else if (cross(v0, v1) > 0)  {
        j = index;
        index = tp2;
      }
      else  {
        printf("ERROR - 180 degree anlge at a triple junction\n");
       printf("triple[index].x, .y  = %f %f\n", xtrip[index], ytrip[index]);
       printf("triple[j].x, .y  = %f %f\n", xtrip[j], ytrip[j]);
       printf("triple[tp1].x, .y = %f %f\n", xtrip[tp1], ytrip[tp1]);
       printf("triple[tp2].x, .y = %f %f\n", xtrip[tp2], ytrip[tp2]);
       printf("index, j, tp1, tp2 are %i %i %i %i\n", index, j, tp1, tp2);
       }
    }
    if (abs_val(triple[index].neighbor[0]) == 6543212)  {  /* Top edge triple
point */
      if (xtrip[start] < xtrip[index])
        xmin = xtrip[start];
      else
        xmin = xtrip[index];
    }
    else            /* This edge point is not the beginning of a cluster */
      xmin = -9.9;
  }
  else  {                       /* Bamboo segment */
    xmin = -9.9;
  }
  return(xmin);
}



float check_cw(int j, float* xtrip, float* ytrip)
{
```

```
int abs_val(int);
int get_index(int, int);
void get_tp1_tp2(int, int *, int *, int);
float cross(float [], float []);

float v0[3], v1[3], v2[3];
float xmax;
int start, index, tp1, tp2, tp3;
float ltot;

ltot = xsize * sqrt( 1.0 + (float) n_regions*n_regions);
start = j;

index = get_index(j, 2);
if (triple[index].type != 1)
  {
    while (triple[index].type != 1)
     {
       tp1 = get_index(index, 0);
       tp2 = get_index(index, 1);
       tp3 = get_index(index, 2);
       get_tp1_tp2(j, &tp1, &tp2, tp3);

       /* Now create 3 vectors from 3 segments emanating from triple point */
       /* index.  These vectors will be crossed for the purpose of */
       /* determining which direction is ccw */

       v0[1] = xtrip[j] - xtrip[index];
       v0[2] = ytrip[j] - ytrip[index];
       v1[1] = xtrip[tp1] - xtrip[index];
       v1[2] = ytrip[tp1] - ytrip[index];
       v2[1] = xtrip[tp2] - xtrip[index];
       v2[2] = ytrip[tp2] - ytrip[index];

        /* Deal with the wrapping at the end of the line */
        if(v0[1]>ltot/2) v0[1] = v0[1]-ltot;
       if(v1[1]>ltot/2) v1[1] = v1[1]-ltot;
       if(v2[1]>ltot/2) v2[1] = v2[1]-ltot;

       if(v0[1]<(-ltot/2)) v0[1] = v0[1]+ltot;
       if(v1[1]<(-ltot/2)) v1[1] = v1[1]+ltot;
       if(v2[1]<(-ltot/2)) v2[1] = v2[1]+ltot;



       if (cross(v0, v1) > 0)  {
         j = index;
         index = tp1;
       }
       else if (cross(v0, v1) < 0)  {
         j = index;
         index = tp2;
       }
       else  {
         printf("ERROR - 180 degree anlge at a triple junction\n");
         printf("triple[index].x, .y  = %f %f\n", xtrip[index], ytrip[index]);
         printf("triple[j].x, .y  = %f %f\n", xtrip[j], ytrip[j]);
         printf("triple[tp1].x, .y = %f %f\n", xtrip[tp1], ytrip[tp1]);
         printf("triple[tp2].x, .y = %f %f\n", xtrip[tp2], ytrip[tp2]);
         printf("index, j, tp1, tp2 are %i %i %i %i\n", index, j, tp1, tp2);
       }
```

```
      }


      if (abs_val(triple[index].neighbor[0]) == 6543212)     /* Top edge triple
point */
       if (xtrip[start] > xtrip[index])
         xmax = xtrip[start];
       else
         xmax = xtrip[index];

      else             /* This edge point is not the beginning of a cluster */
       xmax = -9.9;



    }

  else  xmax = -9.9;  /* bamboo segment */

  return(xmax);
}

void calc_seg_data(int cl_num, polygranular_seg* cluster, float* xmax, float*
xmin,
                   float* xtrip, float* ytrip)
{

  cluster[cl_num].edgenum = 0;
  cluster[cl_num].intnum = 0;

  cluster[cl_num].length = xmax[cl_num] - xmin[cl_num];
  if (cl_num == 1)
    cluster[cl_num].spacing = 0.0;
  else
    cluster[cl_num].spacing = xmin[cl_num] - xmax[cl_num - 1];
  /*
  for (i = 1; i <= triple_max; i++)  {
    if ((xtrip[i] >= xmin[cl_num]) && (xtrip[i] <= xmax[cl_num]))  {
      if ((ytrip[i] == 0.0) || (ytrip[i] == linewidth))
        cluster[cl_num].edgenum++;
      else
        cluster[cl_num].intnum++;
    }
  }
  cluster[cl_num].tot = cluster[cl_num].edgenum + cluster[cl_num].intnum;
  */   /* WALID COMMENTED THIS PORTION */
}



int abs_val(int z)
{
  if (z < 0)
    return(-z);
  else
    return(z);
}
```

The following is a listing of *diff_info*, the subroutine of statistics.c.

```
/*******************************WALID OCT 98*********************************/

/* Outputs all Diffusivity info in the form of "x_start x_end diff" to the file
DIFFDATA, WALID OCT 98 */

int diff_info(char *filename)
{
  void rotate(int, float*, float*);
  void rotate_seg(int, float*, float*);

  int abs_val(int);


  FILE  *fdiffdata;

  float *xtrip, *ytrip;

  float *diffusivity;
  float diff_ratio = 0.005;
  float step=0.1, square_cos;
  int n_cells;
  float begin, end, x_begin, y_begin, r_begin, x_end, y_end, r_end, temp;
  int  n_begin, n_end;

  float theta, linewidth, ltot;
  int i=0, j;

  int segmt, triplept, segmtpts;
  int boundaries_count=0;

  /* Open the diffdata output file. */
  if ( (fdiffdata=fopen(filename,"w")) == NULL)
  { printf("ERROR(gb_statistics): DIFF_INFO can't open %s.\n",filename);
    return(-1); }

  theta = atan(1.0 / (float) n_regions);
  linewidth = (xsize * cos(theta)) / (float) n_regions;
  ltot = (xsize * xsize) / linewidth;

  n_cells = (int)(ltot/step) + 1;
  printf("ltot/step=%f   n_cells=%d \n", ltot/step, n_cells);


  /*  fprintf(fdiffdata,"#%.3f %f %f\n", tau, linewidth, ltot);*/
  fprintf(fdiffdata,"STRAND1\n");
  fprintf(fdiffdata,"%.4f\n",  diff_ratio);


  /* Allocate memory for the cluster temporary arrays. */

  if ( (xtrip=malloc(triple_max*sizeof(float))) == NULL)
  { printf("ERROR(gb_statistics): Couldn't allocate memory for xtrip[].\n");
    return(-1); }
  if ( (ytrip=malloc(triple_max*sizeof(float))) == NULL)
  { printf("ERROR(gb_statistics): Couldn't allocate memory for ytrip[].\n");
    return(-1); }
  /* xtrip[i] and ytrip[i] will contain the rotated triple[i].x and .y
     in the stripped line. */
```

```
  if ( (diffusivity=malloc((n_cells+1)*sizeof(float))) == NULL)
  { printf("ERROR(gb_statistics): Couldn't allocate memory for
diffusivity.\n");
    return(-1); }

  for (i = 0; i <= n_cells; i++)  {
    diffusivity[i] = diff_ratio;
  }


  for(i=1; i<triple_max; i++)
    if(triple[i].used) rotate(i, xtrip, ytrip);  /* condition added by walid */


 /** Cycle through all triple points, to check all segments.  To avoid
   **  checking segments twice, only check those segments which are linked
   **  such that the forward pointer cooresponds to link[0].
   **/
  for(triplept = triple[0].forward, segmtpts = 0;
      triplept != 0;
      triplept = triple[triplept].forward)

    {

      /*  Loop on the 3 boundaries of triple point */
      for(i=0; i<3; i++)    /* Only search the segments once, forward link. */

       if ( triple[triplept].neighbor[i] > 0 )
         if( (abs(edge(triplept))!=1) || (i==2)  ) /* not an edge boundary */
         {
            boundaries_count++;

            x_begin = xtrip[triplept];
            y_begin = ytrip[triplept];

          segmtpts++;

          /* Now to obtain segment points, if any exist, loop over the
             segment points along this boundary until triple point is
             reached. */
          segmt = triple[triplept].segment[i];


          while ((segmt > 0) && (segmtpts < 2*(triple_max+segment_max)))
            {
            x_end = segment[segmt].x;
            y_end = segment[segmt].y;
               rotate_seg(segmt, &x_end, &y_end);
               if(x_end<0) {
              printf("ERROR(gb_statistics) in DIFF_INFO x negative\n");
              return(-1); }


            if(x_begin <= x_end) { begin=x_begin; end=x_end; }
            else { begin=x_end; end=x_begin; }

            /* be ware of line wrapping */
            if( (end-begin)>ltot/2.0 )
              {
                temp = begin;
                begin = end;
                end = temp + ltot;
              }
```

```
            n_begin = (int) (begin/step);   r_begin = (begin/step) - n_begin;
            n_end = (int) (end/step);  r_end = (end/step) - n_end;


               /* UPDATE DIFFUSIVITY ARRAY */

            /* if (n_begin<n_end) {
               diffusivity[n_begin] += (1.0-r_begin);
            for(j=n_begin+1; j<n_end; j++) diffusivity[j] += 1.0;
               diffusivity[n_end] += r_end;
            }
            else if (n_begin==n_end)  diffusivity[n_end] += (r_end-r_begin);
            else { printf("Error gb_statistics, n_begin>n_end\n"); return(-1);
}
            */ /* old way of doing it */

            square_cos = (end-begin)*(end-begin)/( (end-begin)*(end-begin)
                                                  +(y_end-y_begin)*(y_end-
y_begin) );
            if (n_begin<n_end)
              for(j=n_begin+1; j<=n_end; j++)
                diffusivity[j] += square_cos;

            else if (n_begin != n_end)
              { printf("Error gb_statistics, n_begin>n_end\n"); return(-1); }

            segmt = segment[segmt].link[0];
            segmtpts++;

            x_begin = x_end;
               y_begin = y_end;

           }


         /* Opposite triple point reached when segmt becomes negative */
         segmt = -segmt;

         x_end = xtrip[segmt];
         y_end = ytrip[segmt];

         if(x_end<0) {
           printf("ERROR(gb_statistics) in DIFF_INFO x negative\n");
           return(-1); }


         if(x_begin <= x_end) { begin=x_begin; end=x_end; }
         else { begin=x_end; end=x_begin; }

         /* be ware of line wrapping */
         if( (end-begin)>ltot/2.0 )
           {
            temp = begin;
            begin = end;
            end = temp + ltot;
           }


         n_begin = (int) (begin/step);   r_begin = (begin/step) - n_begin;
         n_end = (int) (end/step);  r_end = (end/step) - n_end;
```

```
            /* UPDATE DIFFUSIVITY ARRAY */
            /*if (n_begin<n_end) {
              diffusivity[n_begin] += (1.0-r_begin);
              for(j=n_begin+1; j<n_end; j++) diffusivity[j] += 1.0;
              diffusivity[n_end] += r_end;
            }
            else if (n_begin==n_end)  diffusivity[n_end] += (r_end-r_begin);
            else { printf("Error gb_statistics, n_begin>n_end\n"); return(-1); }
            */  /* old way */

            square_cos = (end-begin)*(end-begin)/( (end-begin)*(end-begin)
                                                   +(y_end-y_begin)*(y_end-
y_begin) );
            if (n_begin<n_end)
              for(j=n_begin+1; j<=n_end; j++)
               diffusivity[j] += square_cos;

            else if (n_begin != n_end)
              { printf("Error gb_statistics, n_begin>n_end\n"); return(-1); }


            segmtpts++;

          }

      }

  diffusivity[0] = diffusivity[n_cells-1];


  printf("Number of grain boundaries in strip = %d\n", boundaries_count);

  for (j = 0; j*step <= ltot; j++)
    fprintf(fdiffdata, "%.3f %.2f %.2f\n",  diffusivity[j], j*step,
(j+1)*step);

  fprintf(fdiffdata, "-1\n");
  fclose(fdiffdata);

  return(1);
}
```

The following is a listing of the tree extraction program *polyseg_tree.c*.

```
/* This program will accept output files from the grain growth simulator
   for a tree (or more eventually) patterned with pattern and generate input
   for emsim concerning geometrical and microstructural infos.


   It is called by typing:
   polyseg [patterned_film_struct_dir_name] diffdata_filename x_offset y_offset

   It will create 3 new files in the patterned_film_struct_dir:
   cluster_data.dat
   bamboo _data.dat
   diffdata_filename
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "polyseg.h"


FILE *fpsegment, *fptriple, *fpstrands;    /*referring to files to read from*/
FILE *fpcluster, *fpbamboo, *fpdiffdata;   /*referring to files to write to*/


char segment[MAXLENGTH] = DATA_DIR;
char triple[MAXLENGTH] = DATA_DIR;
char strands[MAXLENGTH] = DATA_DIR;


char cluster_data[MAXLENGTH] = DATA_DIR;
char bamboo_data[MAXLENGTH] = DATA_DIR;
char diffdata[MAXLENGTH] = DATA_DIR;
char fn_corners[MAXLENGTH] = DATA_DIR;

int trip_max, newpoints;
float DRATIO=1.0, DBAMB=0.005;
float linewidth, ltot, amax;
float xmin[MAXCLUSTERS];
float xmax[MAXCLUSTERS];
float tot_line_clength = 0.0;

triple_pt trip[MAXTRIPLES];
segment_pt seg[MAXSEGS];
polygranular_seg cluster[MAXCLUSTERS];

box_struct strand[MAXSTRANDS];
box_struct junction[MAXJUNCTIONS];


Corner corner[MAXCORNERS];
Corner box[4];
Corner point;
char **ext;


void open_files(int);
void close_files(void);
void calc_seg_data(int, int);
```

```
float check_ccw(int, int, int);
void sort(polygranular_seg *, int, int (*) (polygranular_seg, polygranular_seg)
);
int by_length(polygranular_seg, polygranular_seg);
float check_cw(int, int, int);
int read_in_data(void);
int abs_val(int);
void read_strands(int *, int *);

/* this set of functions is to find if a point is inside a polygon */
int CCW(Corner,Corner,Corner);
int intersection (Corner, Corner, Corner, Corner);
int inside (Corner, Corner*, int);

main(int argc, char *argv[])
{


  float cluster_length_step;
  float cluster_max=0.0, cluster_min;

  int begin = -1, j, i, k, l=0, cnum = 0, num_trips, initial, number, status,
horiz_flag;
  int num_strands, num_junctions, cluster_junc;
  float small_y, small_x;
  int stop_flag = 0;
  float dx=0.0, dy=0.0;

  ext = argv;

  if(argc==4)
    { dx= (float) atof(argv[2]); dy= (float) atof(argv[3]); }

  else if(argc!=2) { printf("ERROR IN CALLING POLYSEG_TREE\n"); exit(-1); }


  /*printf("dx=%.3f, dy=%.3f\n", dx, dy);*/


  open_files(1);                    /* Open all required files */


  num_trips = read_in_data();      /* Read in triple pt and segment pt data */



  read_strands(&num_strands, &num_junctions);


  printf("num_strands=%d, num_junctions=%d.\n\n", num_strands, num_junctions);


  fprintf(fpdiffdata,"OFFSETS \n%f %f\n",dx, dy);



  /* PROCESS THE STRANDS
*****************************************************************/
  for(j=1;j<=num_strands;j++)
  {
    printf("Processing strand #%d...\n",j);
    fprintf(fpdiffdata,"STRAND%d\n",j);
```

```
    fprintf(fpdiffdata, "%d\n", strand[j].horiz);
    /* flag indicating to Vab wether strand is horizontal or vertical */

    fprintf(fpdiffdata,"%.3f\n",DBAMB);
    cnum = 0;  /* initialize cnum for this strand */

    for (i = 0; i < 9999; i++)
      {
       xmin[i] = -9.9;
       xmax[i] = -9.9;
      }       /* initialize cluster info */

    /* FIRST, PICK THE STARTING EDGE TRIPLE POINT */

    if(strand[j].horiz==1)  /* case of a HORIZONTAL strand */
      {
       printf("Strand %d is horizontal\n", j);
       horiz_flag = 1;
       small_x = strand[j].xmax;
       for (i = 0; i<num_trips; i++)
         {
           if (trip[i].real==1 && trip[i].type==1 && trip[i].y>=strand[j].ymin
                && trip[i].y<strand[j].ymax && trip[i].x>=strand[j].xmin &&
              trip[i].x<strand[j].xmax)

             if(trip[i].x<small_x) { small_x = trip[i].x; initial=i; }
         }
      }
    else if(strand[j].horiz==0)  /* case of a VERTICAL strand */
      {
       printf("Strand %d is vertical\n", j);
       horiz_flag = 0;
       small_y = strand[j].ymax;
       for (i = 0; i<num_trips; i++)
         {
           if (trip[i].real==1 && trip[i].type==1 && trip[i].x<=strand[j].xmax
                && trip[i].x>strand[j].xmin && trip[i].y>=strand[j].ymin &&
              trip[i].y<strand[j].ymax)

             if(trip[i].y<small_y) { small_y = trip[i].y; initial=i; }
         }
      }

    else { printf("ERROR IN THE INPUT FILE FORMAT.");  exit(-1); }


    i = initial;
    stop_flag = 0;

    printf(" strand%d, horiz=%d, initial=%d, x=%.2f, y=%.2f, type=%d,
real=%d\n\n",
          j, strand[j].horiz, initial, trip[i].x, trip[i].y, trip[i].type,
trip[i].real);



    /* NOW PROCESS THE STRAND */
    while( stop_flag==0  && i>0)
      {

       /* check for beginning of a cluster */
       if (xmin[cnum] < -9.0)          /* if minimum has not yet been found */
         xmin[cnum] = check_ccw(i, horiz_flag, j);
```

```
      /*printf("cnum=%d, xmin[cnum]=%f \n\n", cnum, xmin[cnum]);*/


      /* check for ending of cluster */
      xmax[cnum] = check_cw(i, horiz_flag, j);         /* always check for xmax
*/
      /*printf("cnum=%d, xmax[cnum]=%f \n\n", cnum, xmax[cnum]); */


          if (xmax[cnum] > -9.0)    /* xmax found */
            {
             calc_seg_data(num_trips, cnum);
             cnum++;
            }

          i = -trip[i].s2;

          if(i>0)
            {
             if(horiz_flag==1)
               {if(trip[i].y!=strand[j].ymin || trip[i].x>strand[j].xmax
                   || trip[i].x<strand[j].xmin) stop_flag=1;}
             else
               {if(trip[i].x!=strand[j].xmax || trip[i].y>strand[j].ymax
                   || trip[i].y<strand[j].ymin) stop_flag=1;}
            }

      }



    /* Length of FIRST and LAST bamboo segments (could be 0) */
    if(horiz_flag==1){
         cluster[0].spacing = xmin[0] - strand[j].xmin;
         cluster[cnum].spacing = strand[j].xmax - xmax[cnum-1];
       }
    else
      {
       cluster[0].spacing = xmin[0] - strand[j].ymin;
       cluster[cnum].spacing = strand[j].ymax - xmax[cnum-1];
      }



    cluster_min = cluster[0].length;

       for (i = 0; i < cnum; i++)
         {
           if (xmin[i] > xmax[i])
             cluster[i].length = 0.0;


           if ((xmin[i] > 0.0) && (xmax[i] < 500.0))
             {
              tot_line_clength += cluster[i].length;

              if (cluster[i].length > cluster_max)
                cluster_max = cluster[i].length;

              if (cluster[i].length < cluster_min)
                cluster_min = cluster[i].length;
             }
```

```
          /*fprintf(fpcluster, "%i %.2f %.2f ", i, xmin[i], xmax[i]);
            fprintf(fpcluster, "%.2f\n", cluster[i].length);
            fprintf(fpcluster, "  %.2f\n", cluster[i].spacing);
            fprintf(fpcluster, "%i %i", cluster[i].intnum, cluster[i].edgenum);
            fprintf(fpcluster, " %i\n", cluster[i].tot);
            */

        }


      /* sort(cluster,cnum,by_length);*/

      for(i=0; i<cnum;i++)
        {
          if(cluster[i].length>=0 && cluster[i].length<=strand[j].length)
            fprintf(fpcluster, "%.2f\n", cluster[i].length);
          if(xmin[i]>=0 && xmin[i]<xmax[i])
            fprintf(fpdiffdata, "%.1f %.1f %.1f\n", DRATIO, xmin[i], xmax[i]);
                                    /* begining and end of each cluster */
        }
      /* end instruction for strand j */

      fprintf(fpdiffdata, "-1\n");


      for(i=0; i<cnum+1;i++)
        if(cluster[i].spacing>=0) fprintf(fpbamboo, "%.2f\n",
cluster[i].spacing);

      fprintf(fpcluster,"finished strand %d.\n\n", j);
      fprintf(fpbamboo, "finished strand %d.\n\n", j);
  }

  /*************************************************************************/




  /**** PROCESS THE JUNCTION ***************************************************/
  for(j=1;j<=num_junctions;j++)
    {
      printf("Processing junction #%d.\n",j);
      fprintf(fpdiffdata,"JUNCTION%d\n", j);
      cluster_junc = 0;

      box[0].x=junction[j].xmin; box[0].y=junction[j].ymin;
      box[1].x=junction[j].xmax; box[1].y=junction[j].ymin;
      box[2].x=junction[j].xmax; box[2].y=junction[j].ymax;
      box[3].x=junction[j].xmin; box[3].y=junction[j].ymax;



      for(i=0; i<num_trips; i++)
       if(trip[i].real==1)    /* only consider actually USED (real) triple
points */
         {
           point.x = trip[i].x; point.y = trip[i].y;
           if( inside(point, box, 4) && trip[i].type==0 )
             {
              /*printf("trip %d with x=%.3f y=%.3f real=%d  inside junc%d\n",
                     i, trip[i].x, trip[i].y, trip[i].real, j);*/
              cluster_junc = 1;
```

```
            }
          }

      if(cluster_junc)
       fprintf(fpdiffdata,"%.1f %.1f\n", DRATIO, DRATIO);   /* CLUSTER JUNCTION
*/
      else
       fprintf(fpdiffdata,"%.3f %.3f\n", DBAMB, DBAMB);   /* BAMBOO JUNCTION */
    }

/*****************************************************************************/

  close_files();

  printf("tot_line_clength for all lines = %f\n", tot_line_clength);
  printf("cluster_max for all lines = %f\n", cluster_max);
  printf("\n\nPOLYSEG_TREE terminated successfully.\n");




}

/* This functions sorts an array of Clusters according to a certain key */
void sort(polygranular_seg * e, int size, int (*key) (polygranular_seg,
polygranular_seg) )
{
  int pos, index;
  polygranular_seg value;

  for(pos=1; pos<size; pos++)
    {
      value = e[pos];

      for(index=pos; index>0 && key(e[index-1],value); index--)
       e[index]=e[index-1];

      e[index] = value;


    }
}

int by_length(polygranular_seg d1,polygranular_seg d2)
{
  return( d1.length > d2.length);
}


void open_files(int i)
{
  printf("open_files()\n");


  strcat(segment, ext[1]);
  strcat(segment, "/segment.dat");

  strcat(triple, ext[1]);
  strcat(triple, "/triple.dat");


  strcat(strands, ext[1]);
  strcat(strands, "/strands.dat");
```

```
  strcat(cluster_data, ext[1]);
  strcat(cluster_data, "/cluster_data.dat");

  strcat(bamboo_data, ext[1]);
  strcat(bamboo_data, "/bamboo_data.dat");

  strcat(diffdata, ext[1]);
  strcat(diffdata, "/diffdata");   /*strcat(diffdata, "/");  strcat(diffdata,
ext[2]);*/

  fpcluster = fopen(cluster_data, "w");
  if (fpcluster == NULL)
    printf("FILE ERROR - unable to open file type %s/n", cluster_data);

  fpbamboo = fopen(bamboo_data, "w");
  if (fpbamboo == NULL)
    printf("FILE ERROR - unable to open file type %s/n", bamboo_data);


  fpdiffdata = fopen(diffdata, "w");
  if (fpdiffdata == NULL)
    printf("FILE ERROR - unable to open file type %s/n", diffdata);


  fpsegment = fopen(segment, "r");
  if (fpsegment == NULL)
    printf("FILE ERROR - unable to open file type %s/n", segment);

  fptriple = fopen(triple, "r");
  if (fptriple == NULL)
    printf("FILE ERROR - unable to open file type %s/n", triple);


  fpstrands = fopen(strands, "r");
  if (fpstrands == NULL)
    printf("FILE ERROR - unable to open file type %s/n", strands);
}


int read_in_data(void)
{
  int read_trip_entry(int);

  char hstring[MAXLENGTH];
  int i = 0, flag = 0, tnum;

  printf("read_in_data()\n");


  /* Read in triple point information */

  flag = read_trip_entry(0);

  for (i = 1; flag > 0; i++)
    {
      flag = read_trip_entry(i);
    }

  tnum = i - 1;

  /* Read in segment point information */
```

```
  i = 0;
  while(fscanf(fpsegment, "%i %i %i",&seg[i].a,&seg[i].b,&seg[i].real) != EOF)
    {
      fscanf(fpsegment, "%f %f", &seg[i].x, &seg[i].y);
      i++;
    }


  return(tnum);
}



int read_trip_entry(int i)
{
  int d1, d2, dummy1, dummy2;


  if (fscanf(fptriple, "%i %i", &trip[i].a, &trip[i].b) != EOF)
    {
      fscanf(fptriple, "%i %i %i %i", &d1, &d2, &trip[i].real, &trip[i].type);
      fscanf(fptriple, "%i %i %i", &trip[i].s1, &trip[i].s2, &trip[i].s3);
      fscanf(fptriple, "%i %i %i", &trip[i].g1, &trip[i].g2, &trip[i].g3);
      fscanf(fptriple, "%f %f", &trip[i].x, &trip[i].y);
      fscanf(fptriple, "%f %f", &dummy1, &dummy2);

      return (1);
    }
  else
    return (-1);
}


void read_strands(int *num_strands, int *num_junctions)
{
  int num_str=0, num_jun=0;
  int num, i;
  char keyword[100];
  float x[4], y[4];


  printf("read_strands()\n");
  while (fscanf(fpstrands, "%s %i", keyword, &num) != EOF)
    {
      if (strcmp(keyword,"STRAND")==0)
       {
         for(i=0;i<4;i++)
           fscanf(fpstrands, "%f %f", &x[i], &y[i]);
         *num_strands = num;

         if(x[0]==x[1]) /* HORIZONTAL STRAND */
           {
             strand[num].horiz=1;
             if(x[1]<x[2]) { strand[num].xmin=x[1]; strand[num].xmax=x[2];
                             strand[num].ymin=y[1]; strand[num].ymax=y[0]; }
             else { strand[num].xmin=x[2]; strand[num].xmax=x[1];
                    strand[num].ymin=y[0]; strand[num].ymax=y[1]; }
             strand[num].length=strand[num].xmax-strand[num].xmin;
           }

         else if(y[0]==y[1]) /* VERTICAL STRAND */
           {
```

```
            strand[num].horiz=0;
            if(y[1]<y[2]) { strand[num].ymin=y[1]; strand[num].ymax=y[2];
                            strand[num].xmin=x[0]; strand[num].xmax=x[1]; }
            else { strand[num].ymin=y[2]; strand[num].ymax=y[1];
                   strand[num].xmin=x[2]; strand[num].xmax=x[0]; }
            strand[num].length=strand[num].ymax-strand[num].ymin;
          }


       }
       else if (strcmp(keyword,"JUNCTION")==0)
       {
         for(i=0;i<4;i++)
           fscanf(fpstrands, "%f %f", &x[i], &y[i]);
         *num_junctions = num;

         if(x[0]==x[1]) /* HORIZONTAL JUNCTION */
           {
             junction[num].horiz=1;
             if(x[1]<x[2]) { junction[num].xmin=x[1]; junction[num].xmax=x[2];
                             junction[num].ymin=y[1]; junction[num].ymax=y[0]; }
             else { junction[num].xmin=x[2]; junction[num].xmax=x[1];
                    junction[num].ymin=y[0]; junction[num].ymax=y[1]; }
           }

         else if(y[0]==y[1]) /* VERTICAL JUNCTION */
           {
             junction[num].horiz=0;
             if(y[1]<y[2]) { junction[num].ymin=y[1]; junction[num].ymax=y[2];
                             junction[num].xmin=x[0]; junction[num].xmax=x[1];
}
             else { junction[num].ymin=y[2]; junction[num].ymax=y[1];
                    junction[num].xmin=x[2]; junction[num].xmax=x[0]; }
           }



       }
       else
       {
         printf("ERROR IN THE FORMAT OF INPUT FILE strands.dat\n");
         exit(-1);
       }

    }
}




float check_ccw(int j, int horiz_flag, int strand_num)
{
  int get_index(int, int);
  void get_tp1_tp2(int, int *, int *, int);
  float cross(float [], float []);

  float v0[3], v1[3], v2[3];
  float xmin;
  int start, index, tp1, tp2, tp3, p;

  start = j;
  index = get_index(j, 3);
```

```
  if (trip[index].type != 1)
    {
      while (trip[index].type != 1)
       {
         tp1 = get_index(index, 1);
         tp2 = get_index(index, 2);
         tp3 = get_index(index, 3);

         get_tp1_tp2(j, &tp1, &tp2, tp3);


      /* Now create 3 vectors from 3 segments emanating from triple point */
      /* index.  These vectors will be crossed for the purpose of */
      /* determining which direction is ccw */

         v0[1] = trip[j].x - trip[index].x;
         v0[2] = trip[j].y - trip[index].y;
         v1[1] = trip[tp1].x - trip[index].x;
         v1[2] = trip[tp1].y - trip[index].y;
         v2[1] = trip[tp2].x - trip[index].x;
         v2[2] = trip[tp2].y - trip[index].y;


         if (cross(v0, v1) < 0)
           {
             j = index;
             index = tp1;
           }
         else if (cross(v0, v1) > 0)
           {
             j = index;
             index = tp2;
           }
         else if (j==tp1)
           {
             j = index;
             index = tp2;
           }
         else if (j==tp2)
           {
             j = index;
             index = tp1;
           }

         else
           {

             printf("ERROR - 180 degree anlge at a triple junction\n");
             printf("trip[index].x, .y  = %f %f\n", trip[index].x,
trip[index].y);
             printf("trip[j].x, .y  = %f %f\n", trip[j].x, trip[j].y);
             printf("trip[tp1].x, .y = %f %f\n", trip[tp1].x, trip[tp1].y);
             printf("trip[tp2].x, .y = %f %f\n", trip[tp2].x, trip[tp2].y);
             printf("index, j, tp1, tp2 are %i %i %i %i\n", index, j, tp1, tp2);
             printf("Program Aborted when calling check_ccw\n");
             exit(-1);
           }

       } /* end of while */
```

```
      if(horiz_flag==1)       /* HORIZONTAL STRAND */
       {
         if(trip[index].x<=strand[strand_num].xmin)
           return(strand[strand_num].xmin);         /* begining of strand */

         if (trip[index].y > trip[start].y)
           {  /* Top edge triple point */
             if (trip[start].x < trip[index].x)
              xmin = trip[start].x;
             else
              xmin = trip[index].x;
           }
         else             /* This edge point is not the beginning of a cluster */
           xmin = -9.9;
       }

      else                     /* VERTICAL STRAND */
       {
         if(trip[index].y<=strand[strand_num].ymin)
           return(strand[strand_num].ymin);          /* begining of strand */


         if (trip[index].x < trip[start].x)
           {  /* Left edge triple point */
             if (trip[start].y < trip[index].y)
              xmin = trip[start].y;
             else
              xmin = trip[index].y;
           }
         else             /* This edge point is not the beginning of a cluster */
           xmin = -9.9;
       }


    }
  else
    xmin = -9.9;      /* Bamboo segment */




  return(xmin);
}



float check_cw(int j, int horiz_flag, int strand_num)
{
  int get_index(int, int);
  void get_tp1_tp2(int, int *, int *, int);
  float cross(float [], float []);

  float v0[3], v1[3], v2[3];
  float xmax;
  int start, index, tp1, tp2, tp3;

  start = j;

  index = get_index(j, 3);
  if (trip[index].type != 1)
    {
```

```
      while (trip[index].type != 1)
       {
         tp1 = get_index(index, 1);
         tp2 = get_index(index, 2);
         tp3 = get_index(index, 3);
         get_tp1_tp2(j, &tp1, &tp2, tp3);



      /* Now create 3 vectors from 3 segments emanating from triple point */
      /* index.  These vectors will be crossed for the purpose of */
      /* determining which direction is ccw */

         v0[1] = trip[j].x - trip[index].x;
         v0[2] = trip[j].y - trip[index].y;
         v1[1] = trip[tp1].x - trip[index].x;
         v1[2] = trip[tp1].y - trip[index].y;
         v2[1] = trip[tp2].x - trip[index].x;
         v2[2] = trip[tp2].y - trip[index].y;

         if (cross(v0, v1) > 0)
           {
             j = index;
             index = tp1;
           }
         else if (cross(v0, v1) < 0)
           {
             j = index;
             index = tp2;
           }
         else if (j==tp1)
           {
             j = index;
             index = tp2;
           }
         else if (j==tp2)
           {
             j = index;
             index = tp1;
           }

         else
           {
             printf("ERROR - 180 degree anlge at a triple junction\n");
             printf("trip[index].x, .y  = %f %f\n", trip[index].x,
trip[index].y);
             printf("trip[j].x, .y  = %f %f\n", trip[j].x, trip[j].y);
             printf("trip[tp1].x, .y = %f %f\n", trip[tp1].x, trip[tp1].y);
             printf("trip[tp2].x, .y = %f %f\n", trip[tp2].x, trip[tp2].y);
             printf("index, j, tp1, tp2 are %i %i %i %i\n", index, j, tp1, tp2);
             printf("Program Aborted when calling check_cw\n");
             exit(-1);
           }

       } /* End of while */

      if(horiz_flag==1)        /* HORIZONTAL LINE */
       {
         if(trip[index].x>=strand[strand_num].xmax)/* ||
trip[index].y>strand[strand_num.ymax]
            || trip[index].y<strand[strand_num.ymin])*/
           return(strand[strand_num].xmax);        /* end of strand */
```

```
        if (trip[index].y > trip[start].y)
          {  /* Top edge triple point */
            if (trip[start].x > trip[index].x)
             xmax = trip[start].x;
            else
             xmax = trip[index].x;


          }
        else            /* This edge point is not the beginning of a cluster */
          xmax = -9.9;
      }


     else       /* VERTICAL LINE */
      {
        if(trip[index].y>=strand[strand_num].ymax)/* ||
trip[index].x>strand[strand_num.xmax]
           || trip[index].x<strand[strand_num.xmin])*/
          return(strand[strand_num].ymax);     /* end of strand */


        if (trip[index].x < trip[start].x)
          {  /* Left edge triple point */
            if (trip[start].y > trip[index].y)
             xmax = trip[start].y;
            else
             xmax = trip[index].y;


          }
        else            /* This edge point is not the beginning of a cluster */
          xmax = -9.9;
      }


    }
  else                          /* Bamboo segment */
    xmax = -9.9;

  return(xmax);

}



int get_index(int i, int n)
{
  int cycle_forward(int);
  int cycle_back(int);

  if (n == 1)
    if (trip[i].g1 > 0)
      return(cycle_forward(trip[i].s1));
    else
      return(cycle_back(trip[i].s1));

  else if (n == 2)
    if (trip[i].g2 > 0)
      return(cycle_forward(trip[i].s2));
    else
      return(cycle_back(trip[i].s2));
```

```
  else
    if (trip[i].g3 > 0)
      return(cycle_forward(trip[i].s3));
    else
      return(cycle_back(trip[i].s3));

}

int cycle_forward(int segnum)
{
  if (segnum < 0)  {
    segnum = -segnum;
    return (segnum);
  }
  else  {
    while (seg[segnum].a > 0)
      segnum = seg[segnum].a;
    return (abs(seg[segnum].a));
  }
}


int cycle_back(int segnum)
{
  if (segnum < 0)  {
    segnum = -segnum;
    return (segnum);
  }
  else  {
    while (seg[segnum].b > 0)
      segnum = seg[segnum].b;
    return (abs(seg[segnum].b));
  }
}

void get_tp1_tp2(int i, int *t1, int *t2, int t3)
{
  /* first some error cases */
  if(*t1!=i && *t2!=i && t3!=i)
    {
      printf("ERROR in function get_tp1_tp2() \n");
      printf("j, tp1, tp2, or tp3 are incorrect \n");
      printf("j, t1, t2, t3 are %i %i %i %i\n", i, *t1, *t2, t3);
      printf("Program aborted when calling get_tp1_tp2()\n");
      exit(-1);
    }

   if(*t1==*t2==t3)
    {
      printf("\n Error in function get_tp1_tp2()");
      printf("\n all arguments are identical\n");
      printf("Program aborted when calling get_tp1_tp2()\n");
      exit(-1);
    }



  /* Case of a lense */
  if( *t1==*t2 || *t1==t3 || *t2==t3 )
    {
      printf("\n Function get_tp1_tp2(), some arguments are identical\n");
      printf(" There is a LENSE \n");
```

```
      if(*t1==*t2)
       *t2 = t3;
    }



  /* Now the regular case */
  else if (*t1 == i)
    {
      *t1 = *t2;
      *t2 = t3;
    }
  else if (*t2 == i)
    *t2 = t3;

}



float cross(float vec1[], float vec2[])
{
  float crossproduct;

  crossproduct = vec1[1]*vec2[2] - vec1[2]*vec2[1];
  return(crossproduct);
}



void calc_seg_data(int tot_trips, int cl_num)
{
  int i;

  cluster[cl_num].edgenum = 0;
  cluster[cl_num].intnum = 0;

  cluster[cl_num].length = xmax[cl_num] - xmin[cl_num];
  if (cl_num == 0)
    cluster[cl_num].spacing = 0.0;
  else
    cluster[cl_num].spacing = xmin[cl_num] - xmax[cl_num - 1];

  /* for (i = 1; i <= tot_trips; i++)  {
    if ((trip[i].x >= xmin[cl_num]) && (trip[i].x <= xmax[cl_num]))  {
      if ((trip[i].y == 0.0) || (trip[i].y == linewidth))
        cluster[cl_num].edgenum++;
      else
        cluster[cl_num].intnum++;
    }
  }
  cluster[cl_num].tot = cluster[cl_num].edgenum + cluster[cl_num].intnum;*/


}



void close_files(void)
{
  printf("close_files()\n");
```

```
  fclose(fpsegment);
  fclose(fptriple);

  fclose(fpdiffdata);
  fclose(fpcluster);
  fclose(fpbamboo);
  fclose(fpstrands);

}
```

```
/*-------------------------------------------------------------------------*/
/* This set of functions checks to see if a data point is inside or outside the
shape */

int CCW(Corner p0, Corner p1,Corner p2)
{
 float dx1,dx2,dy1,dy2;
 dx1=p1.x-p0.x;dy1=p1.y-p0.y;
 dx2=p2.x-p0.x;dy2=p2.y-p0.y;
 if (dx1*dy2>dy1*dx2) return 1;
 if (dx1*dy2<dy1*dx2) return -1;
 if ((dx1*dx2<0) || (dy1*dy2<0)) return -1;
 if ((dx1*dx1+dy1*dy1)<(dx2*dx2+dy2*dy2)) return 1;

 return 0;
}

int intersection (Corner p1, Corner p2, Corner p3, Corner p4)
{
 return((CCW(p1,p2,p3)
        *CCW(p1,p2,p4))<=0)
     &&((CCW(p3,p4,p1)
        *CCW(p3,p4,p2))<=0);
}

int inside (Corner p, Corner * poly, int size)
{
 float P=3.141529;

 int n=size;
 int count=0;
 Corner  p1, p2, p3, p4;
 int ii, i;
 int jj=0;
 p1.x=p.x;
 p1.y=p.y;

 p2.x=p1.x + 20000*cos(P/6.0);

 p2.y=p1.y + 20000*sin(P/6.0);

 for (i=1; i<=n; i++)
   {
    ii=i-(i/n)*n; /*printf("ii=%d\n",ii);*/

    p3.x=poly[ii].x;
    p3.y=poly[ii].y;
    p4.x=p3.x;
```

```
    p4.y=p3.y;
    if (!intersection(p1,p2,p3,p4))
        {
         p4.x=poly[jj].x;
         p4.y=poly[jj].y;
        jj=i;
         if (intersection(p1,p2,p3,p4)) count++;
        }
  }
 return (count & 1);
}
/*--------------------------------------------------------------------------*/
```

Listing of the file polyseg.h

```
/*--------------------------------------------------------------------------*/

#define PI 3.14159
#define MAXNAME 200
#define MAXLENGTH 200
#define MAXTRIPLES 100000
#define MAXSEGS 500000
#define MAXCLUSTERS 100000
#define NUMBER_PATTERNS 100
#define MAXCORNERS 100

#define MAXSTRANDS 100
#define MAXJUNCTIONS 100
#define      DATA_DIR     "/user4/walid/gg_sim_stripped/io/"

typedef struct {
  int a, b;
  int real, type;
  int s1, s2, s3;
  int g1, g2, g3;
  float x,y;
}triple_pt;

typedef struct  {
  int a, b, real;
  float x, y;
}segment_pt;

typedef struct  {
  float length, spacing;
  int intnum, edgenum, tot;
} polygranular_seg;


typedef struct  {
  float x;
  float y;
} Corner;


typedef struct  {
  float xmin, ymin;
  float xmax, ymax;
  int horiz;
  float length;
} box_struct;
```

Appendix D

EmSimGen Microstructure Generation Program

D.1 Program Components and Usage

This program generates input for the EM simulation based on the analytic models developed in Chapters 3, 4, 5, and 7. It produces the file "diffdata" containing the details of the microstructure of a line given the line geometrical definition in terms of thickness h, width w, and length L. The program is called by typing:

```
emsim_gen [L] [w] [h] [D50] diffdata [case_number] [t] [T]
```

where D_{50} is the median grain size in the original film, t is the duration of the annealing time, and T the temperature at which the annealing is considered. Values of the integer "case_number" can be 1, 2 or 3. A value of 1 would lead to the generation of an as-patterned interconnect structure in accordance with the model developed in Chapter 3. A value of 2 leads to the generation of an interconnect structure that has undergone post-patterning annealing until time t at the temperature T. The model used there is the one developed in Chapter 4, if $h << w$, or Chapter 7, when 3D effects cannot be neglected. Finally, if "case_number" is set to 3, bamboo-structures with variable grain surface diffusivities will be generated, in accordance with the developments in Chapter 5. The file "diffdata" output by this program can then be used by MIT/EmSim for electromigration stress calculations and failure time predictions as outlined in Appendix C.

D.2 List of Code

The following is a listing of *emsim_gen.c*, the subroutine of statistics.c.

```
/* OCT 1999, WALID FAYAD
   This program generates input for the EM simulation. It produces the
   file diffdata containing the details of the microstructure of a
   line. diffdata can be input into MIT/EmSim for failure time predictions
*/

   The program is called by typing:
   emsim_gen  [L] [W] [h] [D50] diffdata [case_number] [time] [T]

   where case_number is a flag chosen from 3 possible values:

   case_number = 1
   ---------------
   In this case lines are assumed to have both polygranular clusters and bamboo
   segments. These lines have a WEIBULL distribution for both CLUSTER LENGTHS
   and BAMBOO LENGTHS following the analytic model presented in Spring MRS 98.



   case_number = 2
   ---------------
   Given time t and temperature T, the program produces the structure resulting
   from an anneal until time t, at temperature T, of an initially polygranular
   line.



   case_number = 3
   ---------------
   In this case, lines are assumed to have undergone a long anneal that led
   them from an initial polygranular state to a fully bamboo state. We here
   assume a Weibull distribution of bamboo grain length with α=2.34W and
   β=2.6 which is very close to the lognormal distribution with median 1.9W
   and lognormal deviation 0.5 that was used in the text. We also assume a
   diffusivity dependent on the grains orientation.
   The variation in diffusivity could be as high as an order of magnitude.
   Note that the input of D50 is irrelevant in this case.



   The program outputs 1 new file: <output file name> (diffdata)

*/



#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>


#define MAX_MEASURES 100
```

```
#define MAX_CLUSTERS 100000
#define MAX_GRAINS 100000

#define PI 3.14159

#define DATA_DIR  "/walid/vab/"

FILE * fpout;

char direc[300]=DATA_DIR;


void generate(double, double, double, double, int, time_t, double *, double *,
double,
              char*, double);
double poly(double*, int, double);

void generate_bamboo(double, double, int, time_t,
                     double *, double *, double, char*, double);

void main(int argc, char **argv)
{
  double L, W, D50, h, ww, h_w, Al_clus, Bet_clus, Al_bamb, Bet_bamb;
  int N=1;
  int status, i=0, num, j, case_flag;
  double length[MAX_CLUSTERS], x[MAX_CLUSTERS];
  time_t seed;

  float time, temperature, mu, tau, tau0, tau1, temp;
  float aa, vv, lav_bar, Lbar, Nbar, Lbar1, lav_bar1;
  float mu0 = 4.26;   /* [m2/s], Aluminum assumed pre-exp factor for boundary
mobility */
  float Q = 0.871; /* [eV], Al grain boundary mobility activation energy */
  float k_bolt = 1.3806e-23;
  float ee=1.602e-19;

  double A=exp(-1.530206), B=3.261326;
  /* alpha_clus(w/D50) = A*exp(B*w/D50) */

   double Ab=0.18, Bb=0.47;
  /* alpha_bamb = Ab*(w/D50) + Bb*D50/w) */

  double alB[6]= {7.48886945,-38.788237,89.3081093,-100.356792,54.0000068,-
11.0969354};
  /* stores the 5th degree polynom fitting alpha_bamb(w/d50): Not used in this
model */

  double betC[2]= {1.507935,-0.3894744};
  /* stores the polynom (linear) fitting beta_clus(w/d50) */

  double betB[2]= {1.010636,0.07394694};
  /* stores the polynom (linear) fitting beta_bamb(w/d50) */


  /* specific to case 3 */
  double Al_bamb_grain, Bet_bamb_grain, Diff_factor=15.0;
  double  diffusivity[MAX_GRAINS];


  /* Check error in calling the program */
  if( (argc!=7)  && (argc!=9) )
    {
      printf("Error in calling emsim_gen_plus. \n");
```

```
      exit(-1);
    }

  /* Read in the input information */
  sscanf(argv[1],"%lf",&L);
  sscanf(argv[2],"%lf",&W);

  sscanf(argv[3],"%lf",&h);
  sscanf(argv[4],"%lf",&D50);

  sscanf(argv[6],"%d",&case_flag);

  printf("case_flag=%d\n",case_flag);


  ww = W/D50;

  if(case_flag==1) /* as-patterned, near-bamboo lines, with Wibull
distributions */
    {

      if(ww>2.0)
       {
         printf("W/D50 exceeds range of results accuracy, try with
W/D50<=2.0.\n");
         exit(-1);
       }


      /* Now evaluate distribution parameters according to their functional
dependence
        on W/D50 as what is used in nweib_res.c */

      Al_clus = A*exp(B*ww);
      Al_bamb = Ab*ww + Bb/ww;

      Bet_clus = poly(betC, 1, ww);
      Bet_bamb = poly(betB, 1, ww);



      /* Generate a Line */
      generate(Al_clus, Bet_clus, Al_bamb, Bet_bamb, N, seed, x, length, L,
argv[5], D50);
    }

  else if(case_flag==2)
    {
      h_w = h/W;
      if(h_w>1.0) { temp=h; h=W; W=temp; }

      sscanf(argv[7],"%f", &time);
      sscanf(argv[8],"%f", &temperature);

      mu = mu0*exp(-Q*ee/(k_bolt*temperature));
      tau = mu*time/(W*W);
      aa = 1.1;
      vv = 5.9;

      if(h_w<0.66) tau0=0.15;
      else tau0=0.4545*h_w-0.1545;   /* extrapolation between the two values
experimentally obtained:
```

```
                                    for h/w=1.0 tau0=0.3, for h/w=0.66 tau0=0.15
*/

      tau1 = tau0 + 0.5;

      if(tau<tau0) /* Structure is still fully polygranular */
       {

         strcat(direc,argv[5]);
         /* Open the output file that is going to contain the lines infos */
         if((fpout=fopen(direc,"w"))==NULL)
           {
             printf("ERROR,  can't open %s \n",direc);
             exit(-1);
           }


         /* Store results for this line in output file */
         fprintf(fpout,"OFFSETS\n");
         fprintf(fpout,"0.000000 0.000000\n\n");

         fprintf(fpout,"STRAND1\n");
         fprintf(fpout,"0.005 \n"); /* THIS LINE CONTAINS DIFFUSIVITY OF BAMB
AND DIFF OF CLUS */
         fprintf(fpout,"1.0 0.0 %.11f\n", L);
         fprintf(fpout,"-1\n\n\n\n\n\n\n\n\n\n\n");
         fclose(fpout);
       }

      else if(tau<tau1)
       {

         /* Now evaluate distribution parameters based on exponential
distributions */
         Bet_clus = 1.0;
         Bet_bamb = 1.0;

         Al_clus = W*1/( (W/L) + aa*(tau-tau0) );

         lav_bar = Al_clus/W;
          Lbar = exp(-(W/L)*vv*(tau-tau0)-0.5*vv*aa*(tau-tau0)*(tau-tau0));
          Nbar = Lbar/lav_bar;


         Al_bamb = W*(1-Lbar)/Nbar;


         /* Generate a Line */
         generate(Al_clus, Bet_clus, Al_bamb, Bet_bamb, N, seed, x, length, L,
argv[5], 1.0);

       }
      else if(tau>tau1);
      {

       /* Now evaluate distribution parameters based on exponential
distributions */
       Bet_clus = 1.0;
       Bet_bamb = 1.0;

       Al_clus = W*1/( (W/L) + aa*(tau1-tau0) );

       lav_bar = Al_clus/W;
```

```
      Lbar1 = exp(-(W/L)*vv*(tau1-tau0)-0.5*vv*aa*(tau1-tau0)*(tau1-tau0));
      lav_bar1 = 1/( (W/L) + aa*(tau1-tau0) );
      Lbar = Lbar1*exp(-(1/lav_bar1)*vv*(tau-tau1));
      Nbar = Lbar/lav_bar;

      Al_bamb = W*(1-Lbar)/Nbar;

      /* Generate a Line */
      generate(Al_clus, Bet_clus, Al_bamb, Bet_bamb, N, seed, x, length, L,
argv[5], 1.0);

     }


   }   /* end of case_flag==2 */


  else if(case_flag==3)
    {

     Al_bamb_grain = 2.34*W;
     Bet_bamb_grain = 2.60;   /* corresponds to Weibull distribution  with
                                 alpha=2.34W and beta=2.60.
                                 This gives a distribution very close to
                                 the lognormal distribution with median 1.9W
                                 and lognormal deviation 0.5 which was used in
                                 the text.
                              */


      generate_bamboo(Al_bamb_grain, Bet_bamb_grain, N, seed, x, diffusivity,
L, argv[5], Diff_factor);

    }


}


double poly(double *p, int n, double x)
{

  int i;
  double pol;

  pol = p[0];

  if(n>=1)
    for(i=1;i<=n;i++)
      pol = pol + p[i]*pow(x,i);


  return(pol);
}


void generate(double alc, double betc, double alb, double betb, int N, time_t
seed,
              double * x, double * length, double L, char* out, double D50)
{
 int i=0, j, k, flag=1;
 double u, y, bamb_length, av;
```

```
char direc[300]=DATA_DIR;

/* strcat(direc,DATA_DIR); */

/* strcat(direc,"emsim/");*/
/*  strcat(direc,"emsim/vabttf/W1.29nemsim_genD2.58/"); */


strcat(direc,out);

printf("filname is %s\n\n", direc);

/* Open the output file that is going to contain the lines infos */
if((fpout=fopen(direc,"w"))==NULL)
  {
    printf("ERROR,  can't open %s \n",direc);
    exit(-1);
  }

if( (seed = time(&seed)) == -1 )
  {
    printf("Error in assessing the seed, program aborted\n");
    exit(-1);
  }


srand(seed);
if(seed % 2 == 1)  flag = -flag;


for(k=0; k<N; k++)
  {

    /* if flag is positive, start with a cluster */
    if(flag==1)
      {
       x[0] = 0;

       while(0==0)
         {
           u = (double) rand() / RAND_MAX;
           length[i] = D50*alc*exp( (1/betc)*log(-log(1-u)) );

           if(x[i]+length[i]>L)
             {
              length[i] = L-x[i];
              break;
             }

           u =  (double) rand() / RAND_MAX;
           bamb_length = D50*alb*exp( (1/betb)*log(-log(1-u)) );

           if( x[i]+length[i]+bamb_length > L) break;

           x[i+1] = x[i] + length[i] + bamb_length;
           i = i+1;
         }
       /* if i is still 0 then there is 1 cluster */
      }


    /* if flag is negative, start with a bamboo */
    if(flag==-1)
```

```
        {
        u = (double) rand() / RAND_MAX;
        bamb_length = D50*alb*exp( (1/betb)*log(-log(1-u)) );
        if(bamb_length<L)
          {
            x[0]=bamb_length;

            while(0==0)
              {
              u =  (double) rand() / RAND_MAX;
              length[i] = D50*alc*exp( (1/betc)*log(-log(1-u)) );
              if( x[i]+length[i] > L)
                {
                  length[i] = L-x[i];
                  break;
                }

              u = (double) rand() / RAND_MAX;
              bamb_length = D50*alb*exp( (1/betb)*log(-log(1-u)) );
              if( x[i]+length[i]+bamb_length > L) break;

              x[i+1] = x[i] + length[i] + bamb_length;
              i = i+1;
              }
          }
        /* if i is still 0 then all bamboo */

        }

      /* for(j=0; j<i+1; j++)
        {
        av += length[j];
        printf(" %lf  %lf\n", x[j], x[j]+length[j]);
        }
      av = av / (i+1);
      printf(" Ltot=%lf nclus=%d lastbamb=%lf\n", L, i, bamb_length);
      printf(" average clus_length=%lf\n",av);
      */

      /* Store results for this line in output file */
      fprintf(fpout,"OFFSETS\n");
      fprintf(fpout,"0.000000 0.000000\n\n");

      fprintf(fpout,"STRAND1\n");
      fprintf(fpout,"0.005 \n"); /* THIS LINE CONTAINS DIFFUSIVITY OF BAMB AND
DIFF OF CLUS */
      for(j=0;length[j]>0;j++)
        fprintf(fpout,"1.0 %.11f %.11f\n", x[j], x[j]+length[j]);

      fprintf(fpout,"-1\n\n\n\n\n\n\n\n\n\n\n");

      flag = -flag;
    }

}



void generate_bamboo(double al, double bet, int N, time_t seed,
                     double * x, double * diffusivity, double L, char* out, double
D_factor)
```

```
{
 int i=0, j, k;
 double u, y, bamb_length, av;
 double theta, length;


 char direc[300]=DATA_DIR;
 strcat(direc,out);

 /* printf("filname is %s\n\n", direc); */

 /* Open the output file that is going to contain the lines infos */
 if((fpout=fopen(direc,"w"))==NULL)
   {
     printf("ERROR,  can't open %s \n",direc);
     exit(-1);
   }


 if( (seed = time(&seed)) == -1 )
   {
     printf("Error in assessing the seed, program aborted\n");
     exit(-1);
   }


 srand(seed);


 for(k=0; k<N; k++)
   {


     x[0] = 0;

     while(0==0)
       {
/* Orientation effect, assign diffusivity for the grain to be generated */
        theta =  (double) rand() / RAND_MAX;
        theta = PI*theta/6.0;
        diffusivity[i] = D_factor*(1 + 1.8*theta*log(theta) - 0.554*theta);

         /* Generate the x of the end of grain */
        u = (double) rand() / RAND_MAX;
        length = al*exp( (1/bet)*log(-log(1-u)) );




        if(x[i]+length<=L)
          x[i+1] = x[i] + length;
        else
          {
            x[i+1] = L;
            break;
          }

        i=i+1;
       }
     /* if i is still 0 then there is 1 cluster */

     /* Store results for this line in output file */
     fprintf(fpout,"OFFSETS\n");
```

```
      fprintf(fpout,"0.000000 0.000000\n\n");

      fprintf(fpout,"STRAND1\n");
      fprintf(fpout,"0.005 \n"); /* THIS LINE CONTAINS DIFFUSIVITY OF BAMB AND
DIFF OF CLUS */
      for(j=0;j<i+1;j++)
        fprintf(fpout,"%.2f %.11f %.11f\n", diffusivity[j], x[j], x[j+1]);

      fprintf(fpout,"-1\n\n\n\n\n\n\n\n\n\n\n");

    }

}
```

References

[Abb 92] G. Abbruzzesse and P. Brosso, ed., "Grain Growth in Polycrystalline Materials", *Proc. of the 1st Intl. Conf. on Grain Growth*, Transtec Publications (1992).

[Awa 95] N. Awaya, K. Ohno, and Y. Arita, *J. Electrochem Soc.* **142**, 3173 (1995).

[Bec 49] P. A. Beck, M. L. Holtzworth, and P. R. Sperry, *trans. AIME*, **180**, 163 (1949).

[Ber 69] L. Berenbaum and R. Rosenbergh, "Electromigration Damage of Grain Boundary Triple Points in Al Thin Films", *Thin Solid Films* **4**, 187 (1969).

[Ber 90] B. Berge, A. Simon, and A. Libchaber, *Physical Review A* **41**, 6893 (1990).

[Bla 67] J. R. Black, "Mass Transport of Aluminum by Momentum Exchange with Conducting Electrons", 6^{th} *Int. Reliability Phys. Symp.*, 148 (1967).

[Ble 66] I. A. Blech and H. Sello, "The Failure of Thin Aluminum Current-Carrying Strips on Oxidized Silicon", *Physics of Failures in Electronics*, **5**, ed. by T. S. Shilliday and J. Vaccaro, Rome Air Development Center, 496 (1966).

[Ble 76] I. A. Blech and C. Herring, "Stress Generation by Electromigration", *Appl. Phys. Lett.* **29**, 131 (1976).

[Car 96] R. Carel, C. V. Thompson, and H. J. Frost, "Computer Simulation of Strain Energy Effects vs Surface and Interface Energy effects on Grain Growth in Thin Films", *Acta Mat.* **44**, 2479 (1996).

[Cho 62] J. Y. Choi and P. G. Shewmon, *Trans. AIME* **224**, 589 (1962).

[Cho 89] J. Cho and C. V. Thompson, "Grain Size Dependence of Electromigration-Induced Failures in Narrow Interconnects", *Appl. Phys. Lett.* **54**, 2577 (1989).

[Chr 75] J. W. Christian, *The Theory of Transformations in Metals and Alloys*, second edition, Pergamon (1975).

[Demo 98] *http://nirvana.mit.edu/darpademo/all3tools.html*

[Ems 98] *http://nirvana.mit.edu/emsim*

[Fan 97] D. Fan and L.-Q. Chen, *Acta Mat.* **45**, 611 (1997).

[Fan$_2$ 97] D. Fan, C. Geng, and L.-Q. Chen, *Acta Mat.* **45**, 1115 (1997).

[Fay 97] W. R. Fayad, "Simulation of the Effect of Microstructure on Electromigration-Induced Failure of Interconnects", M. S. Thesis, MIT, 1997.

[Fay 98] W. Fayad, V. Andleigh, C. V. Thompson, and H. J. Frost, "Grain Structure Statistics in As-Patterned and Annealed Interconnects", *MRS Symp. Proc.* **516**, 159 (1998).

[Fay 99] W. Fayad, C. V. Thompson, and H. J. Frost, "Steady-State Grain Size Distributions Resulting from Grain Growth in Two Dimensions", *Scripta Mat.* **40**, 1199 (1999).

[Fay 00] W. R. Fayad, M. J. Kobrinsky, and C. V. Thompson, "An Analytic Model for the Development of Bamboo Microstructures in Thin Film Strips Undergoing Normal Grain Growth", submitted for publication (January 2000).

[Fay$_2$ 00] W. R. Fayad, M. J. Kobrinsky, and C. V. Thompson, "Normal 3D Grain Growth in Rectangular Prisms", submitted for publication (February 2000).

[Fra 94] V. Fradkov and D. Udler, *Advances in Physics* **43**, 739 (1994).

[Fro 82] H. J. Frost and M. F. Ashby, *Deformation-Mechanism Maps: the Plasticity and Creep of Metals and Ceramics*, first edition, Pergamon Press, Oxford, NY, 1982.

[Fro 85] H. J. Frost and C.V. Thompson, *Acta Metall.* **35**, 529 (1985).

[Fro 88] H. J. Frost, C. V. Thompson, C. L. Howe, and J. Whang, *Scripta Met.* **22**, 65 (1988).

[Fro$_2$ 88] H. J. Frost and C. V. Thompson, *J. Electron. Mater.* **17**, 447 (1988).

[Fro 90] H. J. Frost, C. V. Thompson, and D. T. Walton, "Simulation of Thin Film Grain Structures: I. Grain Growth Stagnation", *Acta Met. et Mat.* **38**, 1455 (1990).

[Fro 94] H. J. Frost, Y. Hayashi, C. V. Thompson, and D. T. Walton, "The Effect of Solute Drag on Grain Growth in Thin Films", *MRS Symposium Proceedings* **317**, 431 (1994).

[Fro_2 94] H. J. Frost, Y. Hayashi, C. V. Thompson, and D. T. Walton, "Grain Growth in Thin Films with Variable Grain Boundary Energies", *MRS Symposium Proceedings* **317**, 485 (1994).

[For 98] M. A. Fortes, M. Emilia Rosa, S. Findlay, and M. Guedes, *Phil. Mag. A* **77**, 257 (1998).

[Gil 96] S. P. A. Gill and A. C. F. Cocks, *Acta Mat.* **44**, 4777 (1996).

[Gla 92] J. A. Glazier and D. Weaire, *J. Phys. Condens. Matter* **4** 1867 (1992).

[Hau 99] C. S. Hau-Riege and C. V. Thompson, "Microstructural Evolution Induced by Scanned Laser Annealing in Al Interconnects", *Appl. Phys. Lett.* **75**, 1464 (1999).

[Her 50] C. Herring, "Diffusional Viscosity of a Polycrystalline Solid", *J. Appl. Phys.* **21**, 437 (1950).

[Heu 78] F. M. d'Heurle and P.S. Ho, "Electromigration in Thin Films", in J.M. Poate, K.N. Tu and J.W. Mayer, eds., *Thin Films-Interdiffusion and Reactions*, p. 243 (Wiley, New York, 1978).

[Hu 95] C-K. Hu, K. P. Robdell, T. D. Sullivan, K. Y. Lee, and D. P. Bouldin, "Electromigration and Stress-Induced Voiding in Fine Al and Al-Alloy Thin-Film Lines", *IBM Journal of Research and Development* **39**, 465 (1995).

[Hu_2 95] C-K. Hu and B. Luther, *Mater. Chem. Phys.* **41**, 1 (1995).

[Hu 97] C-K. Hu, K. Y. Lee, L. Gignac, and R. Carruthers, *Thin Solid Films* **308-309**, 443 (1997).

[Hu 98] C-K. Hu and J. M. E. Harper, *Mater. Chem. Phys.* **52**, 5 (1998).

[Hu 99] C-K. Hu, R. Rosenberg, and K. Y. Lee, *Appl. Phys. Lett.* **74**, 2945 (1999).

[Hun 61] H. B. Huntington and A. R. Grone, "Current Induced Marker Motion in Gold Wires", *J. Phys. Chem. Solids* **20**, 76 (1961).

[Hun 80] O. Hunderi and N. Ryum, "The Kinetics of Normal Grain Growth", *J. Mat. Science* **15**, 1104 (1980),

[Joh 39] W. A. Johnson and R. F. Mehl, *Trans. AIME* **135**, 416 (1939).

[Joo 94] Y-C. Joo and C. V. Thompson, "Electromigration Lifetimes of Single Crystal Aluminum Lines with Different Crystallographic Orientations", *MRS Symp. Proc.***338**, 319 (1994).

[Joo_2 94] Y-C. Joo and C. V. Thompson, "Analytic Model for the Grain Structure of Near-Bamboo Interconnects", *J. Appl. Phys.* **76**, 7339 (1994).

[Joo 97] Y-C. Joo and C. V. Thompson, "Electromigration-Induced Transgranular Failure Mechanisms in Single-Crystal Aluminum Interconnects", *J. Appl. Phys.* **81**, 6062 (1997).

[Kan 97] S. H. Kang and J.W. Morris, *J. Appl. Phys.* **82**, 196 (1997).

[Kin 80] E. Kinsbron, "A Model for the Width Dependence of Electromigration Lifetimes in Aluminum Thin-Film Stripes", *Appl. Phys. Lett.* **36**, 968 (1980).

[Kno 95] B. D. Knowlton, J. J. Clement, R. I. Frank, and C. V. Thompson, "Coupled Stress Evolution in Polygranular Clusters and Bamboo Segments in Near-Bamboo Interconnects", *MRS Symp. Proc.* **391**, 189 (1995).

[Kno 97] B. D. Knowlton, J. J. Clement, and C. V. Thompson, "Simulation of the Effects of Grain Structure and Grain Growth on Electromigration and the Reliability of Interconnects", *J. Appl. Phys.* **81**, 6073 (1997).

[Kor 93] M. A. Korhonen, P. Borgesen, K. N. Tu, and C-Y. Li, "Stress Evolution Due to Electromigration in Confined Metal Lines", *J. Appl. Phys.* **73**, 3790 (1993).

[Kor_2 93] M. A. Korhonen, P. Borgesen, D.D. Brown, and C.-Y. Li, "Microstructure Based Model of Electromigration Damage in Confined Line Metallizations in the Presence of Thermally Induced Stresses", *J. Appl. Phs.*, **74**, 4995 (1993).

[Kor 95] M. A. Korhonen, T. Liu, D. D. Brown, and C-Y. Li, "Stress-Voiding and Electromigration in Multilevel Interconnects", *MRS Symp. Proc.* **391**, 411 (1995).

[Kup 00] A. Kuprat, N. Carlson, G. Straub, W. R. Fayad, and C. V. Thompson, Unpublished Results.

[Kup$_2$ 00] A. Kuprat, "Modeling Microstructure Evolution Using Gradient-Weighted Moving Finite Elements", Los Alamos National Laboratory Report LA-UR-98-4879, also accepted for publication in *SIAM J. Sci. Comp.* (2000).

[Kwo 88] T. Kwok and P. S. Ho, in *Diffusion Phenomena in Thin Films*, p. 369 (Noyes Publication, Park Ridge, NJ, 1988).

[Li 93] J. Li, R. Blewer, and J. W. Mayer, "Copper-Based Metallization for ULSI Applications", *MRS Bulletin* **XVIII**, No. 6, 18 (1993).

[Lou 74] N. P. Louat, *Acta Met.* **22**, 721 (1974).

[IEEE 93] *Proceedings of the IEEE VLSI Multilevel Interconnections Conference.* Santa Clara, CA, June 1993.

[Mar 96] K. Marthinsen, O. Hunderi, and N. Ryum, *Acta Mat.* **44**, 1681 (1996).

[Mul 56] W. W. Mullins, *J. Appl. Phys.* **27**, 900 (1956).

[Mul 58] W. W. Mullins, "The Effect of Thermal Grooving on Grain Boundary Motion", *Acta Met.* **6**, 414 (1958).

[Mul 86] W. W. Mullins, *J. Appl. Phys.* **59**, 1341 (1986).

[Mul 98] W. W. Mullins, "Grain Growth of Uniform Boundaries with Scaling", *Acta Mat.* **46**, 6219 (1998).

[Nag 90] T. Nagai, S. Ohta, S. Kawasaki, and T. Okuzono, *Phase Transitions* **28**, (1990).

[Nag 92] T. Nagai, S. Ohta, and K. Kawazaki, *Materials Science Forum* **94-96**, 313 (1992).

[Neu 52] J. von Neumann, in *Metal Interfaces*, p. 108 (American Society for Metals, Cleveland, Ohio, 1952).

[Neu 72] G. Neumann, G. M. Neumann, *Surface Self-Diffusion of Metals*, Bay Village, Ohio, Diffusion Information Center, 1972.

[Pal 87] J. E. Palmer, C. V. Thompson, and H. I. Smith, "Grain Growth and Grain Size Distributions in Germanium Thin Films", *J. Appl. Phys.* **62**, 2492 (1987).

[Par 99] Y-J. Park, V. K. Andleigh, and C. V. Thompson, "Simulations of Stress Evolution and the Current-Density Scaling of Electromigration-Induced Failure Times in Pure and Alloyed Interconnects", *Journal of Applied Physics* **85**, 3546 (1999).

[Pra 83] D. Pramanick and A. N. Saxena, "VLSI Metallization Using Aluminum and its Alloys", *Solid State Technology*, Jan., 127 (1983), and Mar., 131 (1983).

[Rea 54] W. T. Read, Jr., and W. Shockley, Chapter 2 in *Dislocations in Metals*, p. 37, ed. by J. S. Koehler, F. Seitz, W. T. Read, Jr., and E. Orawan (Inst. of Metals, Div. AIME, NY, 1954).

[Ros 98] M. Emilia Rosa, M. A. Fortes, *Europhys. Lett.* **41**, 577 (1998).

[Sro 84] D. J. Srolovitz, M. P. Anderson, P. S. Sahni, and G. S. Grest, *Acta Met.* **32**, 793 (1984).

[Sri 98] V. T. Srikar and C. V. Thompson, *Appl. Phys. Lett.* **72**, 2677 (1998).

[Sta 90] J. Stavans, *Phys. Rev. A* **42**, 5049 (1990).

[Sta 93] J. Stavans, *Rep. Prog. Phys.* **56**, 733 (1993).

[Sti 90] K. Stine, S. Rauseo, B. Moore, J. Wise, and C. Knobler, *Physical Review A* **41**, 6884 (1990).

[Tho 93] C. V. Thompson and J. R. Lloyd, "Electromigration and IC Interconnects", *MRS Bulletin* **XVIII**, 19 (1993).

[Tho_2 93] C. V. Thompson and H. Kahn, *J. Electron. Mater.* **22**, 581 (1993).

[Tra 88] B. M. Tracy, P.W. Davies, D. Fanger, and P. Gartman, Microstructural Science for Thin Film Metallizations in Electronic Applications, TMS, 157-167, Warrendale, PA (1988).

[Vai 80] S. Vaidya, T. T. Sheng, and A. K. Sinha, *Appl. Phys. Lett.* **36**, 464 (1980).

[Vai 81] S. Vaidya, and A. K. Sinha, *Thin Solid Films* **75**, 253 (1981).

[Vaz 88] M. F. Vaz and M.A. Fortes, *Scripta Metall.* **22**, 35 (1988).

[Viñ 97] J. Viñals and W. W. Mullins, "Self-Similarity and Coarsening of Three Dimensional Particles on a One or Two Dimensional Matrix", *J. Appl. Phys.* **83**, 621 (1997).

[Wal 91] D. T. Walton, M.S. Thesis, Thayer School of Engineering, Dartmouth College (1991).

[Wal_2 91] D. T. Walton, H.J. Frost, and C. V. Thompson, *MRS Symp. Proc.* **225**, 219 (1991).

[Wal 92] D. T. Walton, H. J. Frost, and C. V. Thompson, "Development of Near-Bamboo and Bamboo Microstructures in Thin-Film Strips", *Appl. Phys. Lett.* **61**, 40 (1992).

[Wea 84] D. Weaire and N. Rivier, *Contemporary Physics* **25**, 59 (1984).

[Wu 91] K. Wu, W. Baerg, and P. Jupiter, *Appl. Phys. Lett.* **58**, 1299 (1991).

[Yos 95] H. Yoshinaga, T. Watanabe, and N. Takahashi, ed., "Grain Growth in Polycrystalline Materials II", Proc. of the 2nd Intl. Conf. on Grain Growth, Transtec Publications (1995).