

IMPEDANCE RESPONSE OF ALUMINA-  
SILICON CARBIDE WHISKER COMPOSITES

A Thesis Presented to the Academic Faculty

By

David Spencer Mebane

In Partial Fulfillment  
Of the Requirements for the Degree  
Master of Science in Materials Science and Engineering

Georgia Institute of Technology

November 2004

IMPEDANCE RESPONSE OF ALUMINA-  
SILICON CARBIDE WHISKER COMPOSITES

Approved by

Professor Rosario A. Gerhardt

Professor Arun M. Gokhale

Professor Hamid Garmestani

Date Approved

November 22<sup>nd</sup>, 2004

## DEDICATION

This work is dedicated to God, in whom all things are possible, and to my family, especially my mother, father, brother and sister, my wife, Rebecca LeeAnn Mebane and her family, and my daughter, Scarlet May Mebane, without all of whom I would not have completed this work.

## ACKNOWLEDGMENT

The author wishes to thank Advanced Composite Materials Corporation for providing the ceramic composite samples used in this study. The author also wishes to thank the National Science Foundation, who funded this work through grant number DMR-0076153. Most of all, I would like to thank Drs. Gokhale, Garmestani and especially Dr. Gerhardt for their invaluable help.

## TABLE OF CONTENTS

Acknowledgement .....	iv
List of Tables .....	vii
List of Figures .....	viii
List of Symbols .....	xii
Summary .....	xv
Chapter 1 Introduction.....	1
Part I Quantitative Stereology of Randomly and Preferentially Oriented Chopped Fiber Composites.....	4
Chapter 2 Bivariate Stereological Unfolding Procedure for Randomly Oriented Chopped Fibers or Whiskers.....	5
Introduction.....	5
Theory .....	7
Full Ellipses .....	9
Other Section Types .....	12
Final Expression .....	23
Discussion.....	24
Relation to Line Length .....	24
Simulation Results.....	25
Stability and Uniqueness of Solutions.....	26
Conclusion .....	28
Chapter 3 Trivariate, Stereological Length-Radius-Orientation Unfolding Derived and Applied to Alumina-Silicon Carbide Whisker Composites.....	30
Introduction.....	30
Theory .....	31
Unfolding Equations .....	31
Distribution Function .....	35
Application to SiCw-Al <sub>2</sub> O <sub>3</sub> Composites .....	39
Experimental Results and Discussion.....	42

Conclusion .....	47
Part II Non-Destructive Characterization of Ceramic Matrix Composites.....	48
Chapter 4 Interpreting Impedance Response of Silicon Carbide Whisker / Alumina Composites Through Microstructural Simulation.....	49
Introduction.....	49
Simulation.....	51
Intersection: Excluded Volume .....	52
Determining Effective Resistivity .....	55
Experimental.....	57
Results.....	58
Discussion.....	65
Dispersion Quality.....	65
Interfacial Phenomena.....	66
Conclusion .....	67
Chapter 5 Conclusion and Recommendations .....	68
References.....	70
Appendix Simulation Code .....	75

## LIST OF TABLES

<i>Number</i>	<i>Page</i>
Table 3.1, Distribution Parameters	43

## LIST OF FIGURES

<i>Number</i>	<i>Page</i>
Figure 2.1. The different types of elliptical sections found on the plane of polish: (a) full ellipses, (b) single-truncated ellipses and (c) double-truncated ellipses.	7
Figure 2.2. Definition of orientation angles with respect to the particle and the sectioning plane. $\beta$ does not enter into calculations and therefore does not appear in the text.	11
Figure 2.3. Depictions of the ‘box height’ for single-truncated sections at different angles to the sectioning plane, for sections between $r$ and $2r$ in length. The height used in equation 8 is twice the height shown in the figures, taking into account both ends of the cylinder.	15
Figure 2.4. Box heights for single-truncated sections with length greater than $2r$ and less than $\frac{1}{2}\sqrt{L^2 + 4r^2}$ .	16
Figure 2.5. Box heights for single-truncated sections greater than $\frac{1}{2}\sqrt{L^2 + 4r^2}$ and less than $\sqrt{L^2 + r^2}$ .	17
Figure 2.6. Box heights for single-truncated sections of length greater than $\sqrt{L^2 + r^2}$ .	18
Figure 2.7. Fraction of elliptical sections wherein $a < 2r$ for a monodisperse system at given $\alpha$ angles. The average over all angles is the average height of each curve.	19
Figure 2.8. Box heights for double-truncated sections.	22
Figure 2.9. Monte Carlo simulation correspondence to the unfolding equation.	26

Figure 2.10. The results of the unfolding equation compared to input three-dimensional whisker distributions showing different average lengths. The different line weights of part (b) correspond to the output of the unfolding equation using the same distribution in part (a). 28

Figure 3.1. Schematic of the hot pressed composites. Part (a) is a view perpendicular to the hot pressing direction, while view (b) is parallel to the hot pressing direction (indicated with an arrow). After Gerhardt and Ruh.<sup>27</sup> 33

Figure 3.2. Definition of the angle  $\delta$ . The figure shows an elliptical section occurring on the vertical plane. 34

Figure 3.3. SEM images of hot pressed SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub> composite, taken from the (a) vertical and (b) horizontal sections. 41

Figure 3.4. Fits of the distribution parameters to the experimental data taken from the vertical section, via the orientation unfolding equation (equation 3.3). Figures (a)-(d) refer to rows 1 through 4 of Table 3.1, respectively. In each figure, the abscissa values are  $\delta$  angle, in radians, and the ordinate values are cumulative number per unit area, in  $\mu\text{m}^2$ . 44

Figure 3.5. Fits of the distribution parameters to the experimental data taken from the horizontal section, via the aspect ratio unfolding equation. Figures (a)-(d) refer to rows 1 through 4 of Table 3.1, respectively. In each figure, the abscissa values are the length of elliptical sections, in  $\mu\text{m}$ , and the ordinate values are cumulative number per unit area, in  $\mu\text{m}^2$ . 45

Figure 3.6. Bivariate length-orientation distributions at constant radii. (a)-(d) refer to rows 1-4 of Table 3.1, respectively. 47

Figure 4.1. Depiction of the excluded volume surrounding a cylinder already placed in the matrix (left) with respect to a particle to be placed (shaded, right). $\theta$ is the angle between the particles. The figure shows the concept in two dimensions, although its extension to three dimensions is straightforward. From Balberg, <i>et al.</i> ; used by permission. <sup>36</sup>	53
Figure 4.2. Graphical rendering of a percolating cluster from the Monte Carlo simulation.	56
Figure 4.3. SEM images of Al <sub>2</sub> O <sub>3</sub> -SiC <sub>w</sub> composites used in this study. The micrographs show (a) a 20% SiC <sub>w</sub> sample taken from the plane perpendicular to the hot pressing direction (b) a 20% SiC <sub>w</sub> sample taken from the plane parallel to the hot pressing direction and (c) a 30% SiC <sub>w</sub> sample taken from the plane perpendicular to the hot pressing direction.	58
Figure 4.4. Fitted impedance spectra for (a) 10% (b) 20% and (c) 30% samples of SiC <sub>w</sub> -Al <sub>2</sub> O <sub>3</sub> composite. The high frequency semicircle occurs at frequencies too high to be shown in part (c), but it still exists.	59
Figure 4.5. Fitted spectra for (a) 8 and (b) 9% SiC <sub>w</sub> samples.	60
Figure 4.6. The equivalent circuit used to fit the spectra in Figures 4.4 and 5. In Figure 4.5, the response from the electrode is completely overtaken by the large resistivity of the bulk.	60
Figure 4.7. Complex resistivity (size-normalized) response from samples of 10% SiC <sub>w</sub> , but different thicknesses and contact areas.	61
Figure 4.8. Bulk composite conductivity plotted versus volume fraction for all measured specimens, from all batches.	62
Figure 4.9. Simulated percolating volume fraction versus volume fraction for different values of the shorting distance.	63

Figure 4.10. Bode plot (real impedance versus frequency) for a SiC-Al<sub>2</sub>O<sub>3</sub>-SiC interface, simulated as a parallel RC equivalent circuit.

64

Figure 4.11. Conductivities from a single batch plotted versus the connectivity parameter  $K$ , as determined through the simulation. The  $K$  values are normalized such that the value at 10% volume fraction overlaps with the average experimental value.

65

Figure 4.12. Schematic of the two different types of possible whisker interfaces, (a) SiC-Al<sub>2</sub>O<sub>3</sub>-SiC and (b) SiC-SiC.

67

## LIST OF SYMBOLS

$N_A$	Total number of sections per unit area on a sectioning plane
$a$	Length of an elliptical section
$f(a)$	Two-dimensional distribution of elliptical section lengths on a horizontal plane
$r$	Width of an elliptical section
$R$	Radius of a cylindrical particle
$L_{\min}$	Minimum length of a cylindrical particle based on the length of its two-dimensional section
$L$	Length of a cylindrical particle
$H(L,a)$	Number density of sections on a horizontal sectioning plane per unit plane area with length $a$ or greater and of length $L$
$\alpha$	Angle between a cylindrical particle and the horizontal sectioning plane
$F(L,\alpha)$	Three-dimensional density of cylindrical particles
$N_V$	Number of cylindrical particles per unit volume
$G(a)$	Number density of sections on the horizontal plane with length greater than $a$
$L_V$	Line length per unit volume
$Q_A$	Points per unit area
$\bar{L}$	Average particle length
$\beta$	Angle of rotation of a cylindrical particle with respect to the horizontal sectioning plane

$\gamma$	Angle between the cylindrical particle and the vertical sectioning plane
$\delta$	Angle of rotation of a cylindrical particle with respect to the vertical sectioning plane
$g(\delta)$	Number density of sections on the vertical plane with angle $\delta$ or higher
$s_L$	Lognormal shape parameter
$s_\alpha$	Normal scale parameter
$m$	Lognormal scale parameter
$c$	Correlation coefficient
$\Omega$	Angular distribution multiplier; related to average lengths of cylindrical particles
$N(\alpha)$	Angular distribution of cylindrical particles
$T_1, T_2$	Normalization factors
$Z'$	Real impedance
$R_s$	Resistance
$C$	Capacitance
$\omega$	Angular frequency
$j$	The imaginary unit
$\rho_w$	Resistivity of whiskers
$l$	Length of conducting paths
$ar$	Area of conducting paths
$M_K$	Kirchoff's matrix

$V$	Vector of node voltages
$V_A$	Vector of applied voltages
$\sigma$	Conductivity of the composite
$\sigma_w$	Conductivity of the whiskers
$K$	Connectivity factor

## SUMMARY

The impedance response of silicon carbide whisker-alumina composites is investigated utilizing novel stereological techniques along with a microstructural simulation. The stereological techniques developed allow for a measurement of the trivariate length, radius and orientation distribution of whiskers in the composite from measurements made on two-dimensional sectioning planes. The measured distributions are then utilized in a Monte Carlo simulation that predicts connectivity in the composite for a given volume fraction. It is assumed in the simulation that connectivity factors dominate the electrical response, not interfacial phenomena. The results of the simulation are compared with impedance spectra taken from real samples, and conclusions are drawn regarding the nature of the impedance response.

## CHAPTER 1

### INTRODUCTION

Structural ceramics are being used ever more frequently for industrial applications where high strength, high corrosion resistance or high service temperatures are required. And since the 1980's it has been known that the ceramic's Achilles heel – fracture toughness – can be mitigated by the use of anisotropic second phases. One material that is used extensively as a fracture toughness enhancer is silicon carbide whisker. Grown in a chemical vapor deposition or carbothermal reduction process, whiskers with diameters less than a micrometer and with high aspect ratios can be achieved.<sup>1</sup> The high aspect ratio performs well for crack bridging and deflection.<sup>2</sup> But the small size and perfect crystallinity of the whiskers brings a challenge to composite processing, in that the whiskers do not disperse evenly. This is unfortunate from a mechanical properties point of view, since clumping of the reinforcement phase reduces the fracture toughness enhancement relative to perfectly dispersed filler.<sup>3</sup> Moreover, the difficulty in producing high-quality dispersion and the dependence of fracture toughness on dispersion quality are such that producing consistently reliable manufactured products is difficult. This inevitably leads to high losses in manufacturing. With this background, it becomes clear that a non-destructive method for testing the dispersion quality of ceramic matrix composites is desirable.

Impedance-based testing is a good candidate technique, as it can be sensitive to microstructural changes. Silicon carbide, in particular, presents good potential opportunities for electrical testing, because of its high conductivity relative to most common ceramic matrix materials. But focusing on the electrical properties of composites leads into the realm of effective medium theory, for which an over-arching theoretical framework remains elusive. In particular, the phenomenon of percolation is still not very well understood from a general theoretical standpoint. In fact, percolation considerations dominate the electrical response of ceramic whisker composites in the range of filler volume fraction they are most often found in.

Computational and numerical approaches may hold the key to using electrically based non-destructive testing techniques for ceramic matrix composites. This work is a first attempt to bring computational techniques to bear on the problem, focusing on the alumina-silicon carbide whisker composite system. The computer simulation presented in Part II of this thesis is a Monte Carlo simulation that offers insights into the effect of connectivity on the electrical response through the comparison of simulation results with impedance-based tests.

But microstructural simulations can offer few insights into experimental results without solid microstructural characterization techniques. For that reason, Part I of this thesis picks up the decades-old problem of characterizing fibrous microstructure using stereology. Specifically, the lengths, diameters and orientations of the whiskers all affect connectivity. This implies a trivariate unfolding. In fact, a trivariate unfolding evolves in

Chapter 3, but is attained indirectly by route of a bivariate unfolding for random orientations that is the subject of Chapter 2.

With the critical characterization problem tackled, the analysis moves on to a discussion of simulation results with respect to experiment in Chapter 4. A central question to be addressed is the role of the matrix in conduction. Because the conductivity of alumina is several orders of magnitude smaller than that of silicon carbide, one may think of the matrix as unimportant to the conductivity of percolating  $\text{SiC}_w\text{-Al}_2\text{O}_3$  composites. But an A/C electrical signal means that the situation may not be so simple, and the dielectric properties of the matrix phase may also play a role. In addition, a question arises as to the influence of the kind and quality of the interface between contacting whiskers in the composite. Insight into this question holds relevance not only for the non-destructive characterization problem at hand, but the investigation of the critical behavior of percolation in general.

PART I

QUANTITATIVE STEREOLOGY OF  
RANDOMLY AND PREFERENTIALLY  
ORIENTED CHOPPED FIBER  
COMPOSITES

## CHAPTER 2

### BIVARIATE STEREOLOGICAL UNFOLDING PROCEDURE FOR RANDOMLY ORIENTED CHOPPED FIBERS OR WHISKERS

#### **Introduction**

Interest in the problem of determining the length of a collection of straight objects from a sample of reduced dimensionality goes at least as far back as the late 18<sup>th</sup> century, when Buffon presented his needle problem.<sup>4</sup> But in modern times Fullman showed that Buffon's famously simple result – the number of intersections of a needle with a straight line is proportional to the needle length – is insufficient to determine the lengths or number of long rods per unit volume of a three-dimensional composite.<sup>5</sup> In a 1953 paper, Fullman demonstrated that average measurements on the intersections between particles represented as straight cylinders and a sectioning plane (such as average lineal transverse and area) offer no information on the length of particles in the material.<sup>5</sup> DeHoff and Rhines later revisited Fullman's work, showing that in fact a determination of average cylinder length is possible through consideration of intersections of the cylinders' circular ends with the sectioning plane.<sup>6</sup> Thouless, Daghiesh and Evans modified DeHoff's approach, considering the aspect ratios of two-dimensional elliptical and partial elliptical sections as opposed to the intersections of the circular ends of the cylinders with the sectioning plane.<sup>7</sup>

All of the above-cited authors treat the estimation of aspect ratio and length parameters for monodisperse systems or average parameters for polydisperse systems. For perhaps most applications, knowledge of average size parameters is sufficient. However, many analyses require knowledge of the full distribution of aspect ratios. Three-dimensional unfolding procedures for several different particle shapes are available, including spheres,<sup>8</sup> discs<sup>9</sup> and prolate / oblate ellipsoids.<sup>10-12</sup> In reality, microstructures do not strictly hold to such ideal shape conventions. In the context of determining the size and shape distribution of inclusions with a high aspect ratio, the unfolding of prolate and oblate ellipsoids – addressed by Cruz-Orive and Benes<sup>10-12</sup> – may well serve the problem at hand. However, some microstructures – whisker composites, for example – conform more closely to the idealization of a distribution of cylinders. For these cases, a more straightforward solution is presented here.

This chapter presents a mathematically rigorous unfolding for length distributions of cylindrical particles from measurements made on a two-dimensional sectioning plane. The derivation below follows a generally established unfolding procedure – that is, it employs a particular technique for developing a mathematical expression to relate the three-dimensional distribution function of interest to a measurable two-dimensional distribution function. Gokhale's unfolding of a bivariate size-orientation distribution of discs served as a primary guide.<sup>9</sup> In addition, probabilistic and geometric relationships first established for the cylinder problem by Fullman, DeHoff and Thouless form a significant portion of the mathematical basis for the following analysis.<sup>5-7</sup>

## Theory

The first step in the derivation is an examination of the types of intersections made by straight fibers intersecting a sectioning plane. If we assume that the fibers take the shape of cylinders, then there are three different types of intersections, each elliptical in nature. There are fully elliptical sections, which occur when cylinders are cut through the middle, single-truncated ellipses, which occur when cylinders are cut at one end, and double-truncated ellipses, which occur when cylinders are cut down their entire length, such that both ends intersect the sectioning plane (see Figure 2.1). These three general types divide further into sections that contain the cylinder axis (for which the sectioning plane cuts through the axis) and those that do not.

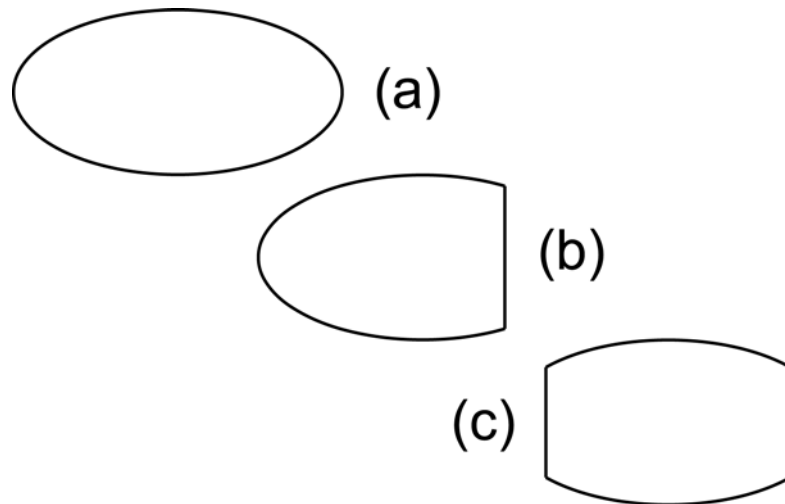


Figure 2.1. The different types of elliptical sections found on the plane of polish: (a) full ellipses, (b) single-truncated ellipses and (c) double-truncated ellipses.

The treatment of the general bivariate length-radius unfolding problem benefits from a simple initial limitation on the types of sections considered, effectively reducing the

problem to a univariate unfolding. Namely, limiting the analysis to those elliptical sections that contain the cylinder axis permits the unfolding of a bivariate length-radius distribution through an iterative unfolding of univariate distributions in length operated on sections of constant radius. (The full radius of any elliptical section that includes the particle axis appears as the minor axis of the ellipse.) This is an advantageous observation, since it greatly simplifies the following analysis. Furthermore, sorting out sections that contain the fiber axis is relatively straightforward, based on area considerations: the actual area of a partial elliptical section which is less than half of an ellipse will always be less than the value of  $\frac{\pi}{4}ab$ , where  $a$  is the longest feret and  $b$  is the shortest feret. This limitation also provides for an easy mathematical check to the rather complex calculations that follow, based on well-known concepts in stereology. The following analysis therefore pertains to particles of a known (constant) radius,  $R$ , but it may be extended to particles of any distribution of radii, if applied iteratively to different radius classes.

The most efficient mathematical treatment of the problem involves a separate consideration of each type of section, followed by a summation of the separate solutions into an overall solution covering all section types. In other words, if one denotes the number density of the three different types of sections (of length  $a$ ) as  $f_1(a)$ ,  $f_2(a)$  and  $f_3(a)$ , and the total number of each section type per unit area as  $N_{A1}$ ,  $N_{A2}$  and  $N_{A3}$ , then

$$N_{A1}f_1(a) + N_{A2}f_2(a) + N_{A3}f_3(a) = N_A f(a) \quad (2.1)$$

where  $N_A$  and  $f(a)$  represent the total number of sections per unit area and the number density of sections with size  $a$ , respectively. The different types of sections are discussed in the following paragraphs, starting with the full ellipse and moving through single-truncated (Figure 2.1b) and to double-truncated (Figure 2.1c) ellipses.

### Full Ellipses

The simplest treatment of the different section types is the full ellipse, which also becomes more prevalent as aspect ratios increase. Some important properties of the fully elliptical section:

- i. Its minor axis,  $r$ , is equal to the radius of the particle  $R$ .
- ii. The major and minor axes determine a minimum particle length,  $L_{min}$ .
- iii. The particle's angle of orientation with the sectioning plane,  $\alpha$ , is equal to the inverse sine of the ratio between minor and major axis.

According to condition i,

$$r = R \tag{2.2}$$

where  $R$  is the radius of the particle causing the elliptical section and  $r$  is the minor axis of the ellipse. Furthermore, condition ii leads to

$$L_{min} = \sqrt{a^2 - 4r^2} \tag{2.3}$$

where  $L_{min}$  is the minimum length of the particle sectioned and  $a$  the full major axis (double the normal major axis). Equation 2.3 arises from the sectioning plane's bisection

of a cylinder at its broadest point: across its entire length and width. A mathematical expression of condition iii follows below.

Considering all full ellipses on the sectioning plane, establish a function  $H_{fe}(L,a)dL$  as the number of fully elliptical sections on the plane per unit plane area with full major axis of  $a$  or greater, coming from particles of length between  $L$  and  $L + dL$ . Denoting  $F(L,\alpha)dLd\alpha$  as the distribution of particles with length between  $L$  and  $L + dL$ , and angle between the particle axis and the sectioning plane of  $\alpha$  to  $\alpha + d\alpha$ ,

$$H_{fe}(L, a)dL = \int_{\alpha_{\min}}^{\alpha_{\max}} N_v F(L, \alpha) \cos\alpha (L \sin\alpha - 2r \cos\alpha) d\alpha dL \quad (2.4)$$

Note that in equation 2.4, the term  $(L \sin\alpha - 2r \cos\alpha)$  arises as the height of the box built around the sectioning plane inside of which a particle center may be placed in order to cause a full ellipse section. Because  $H_{fe}$  is defined per unit area of sectioning plane, the width and length of the box are 1. The additional  $\cos\alpha$  term comes from the choice of convention for angles, with a latitudinal angle measured in the sectioning plane and the angle of inclination to the plane measured as shown in Figure 2.2. The angles  $\alpha_{\min}$  and  $\alpha_{\max}$  may be defined in terms of  $r$ ,  $L$  and  $a$  (as per condition iii above) as follows:

$$\alpha_{\max} = \arcsin \frac{2r}{a} \quad (2.5a)$$

$$\alpha_{\min} = \arctan \frac{2r}{L} \quad (2.5b)$$

Furthermore, because the orientations of the fibers are isotropic,  $F$  in equation 2.4 does not depend on  $\alpha$ .

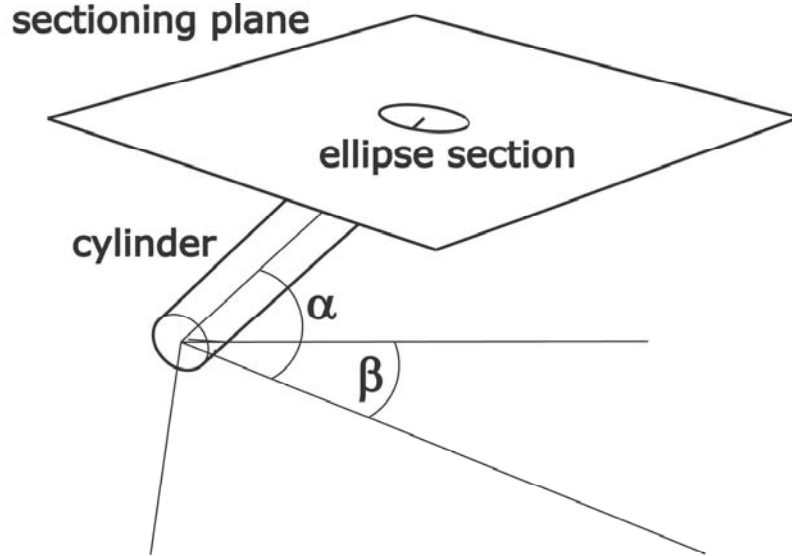


Figure 2.2. Definition of orientation angles with respect to the particle and the sectioning plane.  $\beta$  does not enter into calculations and therefore does not appear in the text.

The next step is to integrate the function  $H$  over all particle lengths. Using equation 2.3 to determine  $L_{\min}$ ,

$$G_{fe}(a) = \int_a^{a_{\max}} N_{A_{fe}} f_{fe}(a') da' = \int_{\sqrt{a^2 - 4r^2}}^{L_{\max}} H_{fe}(L, a) dL \quad (2.6)$$

Equation 2.6 leads to the full expression for the number of full ellipses on the plane with full major axis larger than an arbitrary section length  $a$ :

$$\int_a^{a_{\max}} N_A f_{je}(a') da' = \int_{\sqrt{a^2-4r^2}}^{L_{\max}} \int_{\tan^{-1} \frac{2r}{L}}^{\sin^{-1} \frac{2r}{a}} N_V F(L) \cos \alpha (L \sin \alpha - 2r \cos \alpha) d\alpha dL \quad (2.7)$$

### Other Section Types

It is clear from equation 2.7 that, at least in theory, one need only consider full ellipses appearing on the plane of polish to find the three-dimensional radius-length distribution. However, a full ellipse-only approach presents some practical problems. This is primarily due to the fact that to filter the full ellipses from other section types, some method of distinguishing full ellipses from truncated ones must be applicable to raw data of minimum and maximum ferets measured by an automatic image analysis program. Comparing feret lengths to areas suffers from the fact that the area of the section will be exactly equal to  $\frac{\pi}{4}ab$  both when the section is fully elliptical and when the section is exactly half an ellipse. Perimeter methods suffer from the fact that there is no closed form expression for the perimeter of an ellipse – only an infinite series – and approximations are not currently available for truncated ellipses.

However, the area-feret comparison may be used to sort ellipses that are greater than half ellipses from those that are not. This is because the product  $\frac{\pi}{4}ab$  is always less than the actual area of the ellipse for single-truncated ellipses that are more than half of an ellipse, leading to a simple comparison condition in a filter. Moreover, almost all double-truncated ellipses that include the ellipse's full minor axis (which occurs when the

sectioning plane cuts through a cylindrical particle's central axis) adhere to the same criterion.

An additional advantage of including only particles that reveal their full diameters on the sectioning plane is that it increases the amount of data available for the unfolding, which improves accuracy and stability.

### Single-Truncated Ellipses

Single-truncated ellipses that are greater than half ellipses (sections that include the particle axis) undergo a similar treatment as full ellipses, with the added condition that the distance from the particle center to the plane of polish as well as the angle of incidence will influence the length of the section  $a$ . As before, the minor axis is equal to the actual radius of the particle, and equation 2.2 applies. However, as depicted in Figures 2.3-6, the derivation of an expression for the number of single-truncated ellipses of section length greater than  $a$  arising from particles of length  $L$  to  $L + dL$  must take into account several different size classes for the section length  $a$ , relative to  $r$  and  $L$ . The reason for this is that the 'box height' used to calculate the number of sections longer than an arbitrary length  $a$  change as the particle rotates through the range of  $\alpha$ . But the manner in which the box height changes through the rotation depends on which of several length classes  $a$  falls into. This is a consequence of the particle's finite shape, and it may seem from the following analysis that a discontinuous final expression will result. However, all but one of the different length restrictions in  $a$  invert to length restrictions in  $L$ , expressed as integration limits. The only length class that will not invert to limits of integration may be safely disregarded in most cases.

The shortest size class is  $r \leq a \leq 2r$ , and the analysis that follows corresponds to Figure 2.3, which shows a side view of a cylindrical particle. (The perspective of Figures 2.3-6 and 2.8 is looking onto a plane perpendicular to the sectioning plane.) Bearing in mind that the cumulative function counts sections with a length greater than some arbitrary length  $a$ , it becomes clear that only a certain portion of the ‘box height’ (from above) for single-truncated ellipses will produce a section longer than  $a$  for certain large angles  $\alpha$  (Figure 2.3a). For other, smaller angles, a slice anywhere along the box height will produce a section of sufficient length (Figure 2.3b-c). The expression for the number of sections larger than  $a$ , coming from particles of length between  $L$  and  $L + dL$  for this size class is

$$\begin{aligned}
 H_{ste1}(L, a)dL = & \hspace{15em} (2.8) \\
 & \int_{\sin^{-1} \frac{r}{a}}^{\pi/2} N_V F(L) \cos \alpha (4r \cos \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL + \\
 & \int_{\tan^{-1} \frac{2r}{L}}^{\sin^{-1} \frac{r}{a}} N_V F(L) \cos \alpha (2r \cos \alpha) d\alpha dL + \\
 & \int_{\tan^{-1} \frac{r}{L}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (2L \sin \alpha - 2r \cos \alpha) d\alpha dL
 \end{aligned}$$

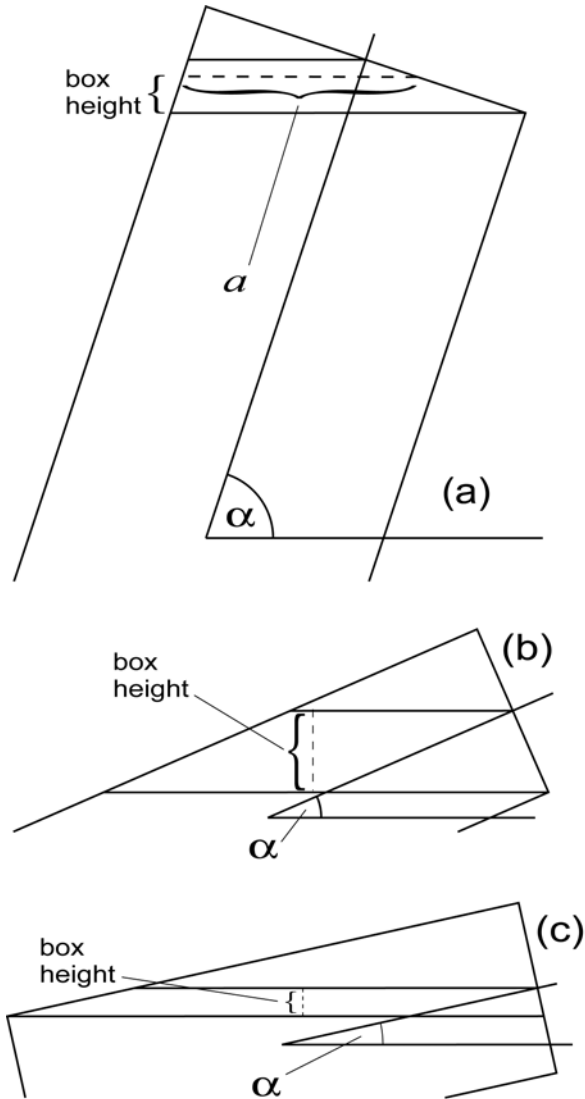


Figure 2.3. Depictions of the ‘box height’ for single-truncated sections at different angles to the sectioning plane, for sections between  $r$  and  $2r$  in length. The height used in equation 8 is twice the height shown in the figures, taking into account both ends of the cylinder.

The angular limits to the integral terms on the right-hand side of equation 2.8 correspond to the angular regions depicted in Figure 2.3. The next length class is

$2r \leq a \leq \frac{1}{2}\sqrt{L^2 + 4r^2}$  (Figure 2.4). An analysis similar to that used to derive equation 2.8

leads to

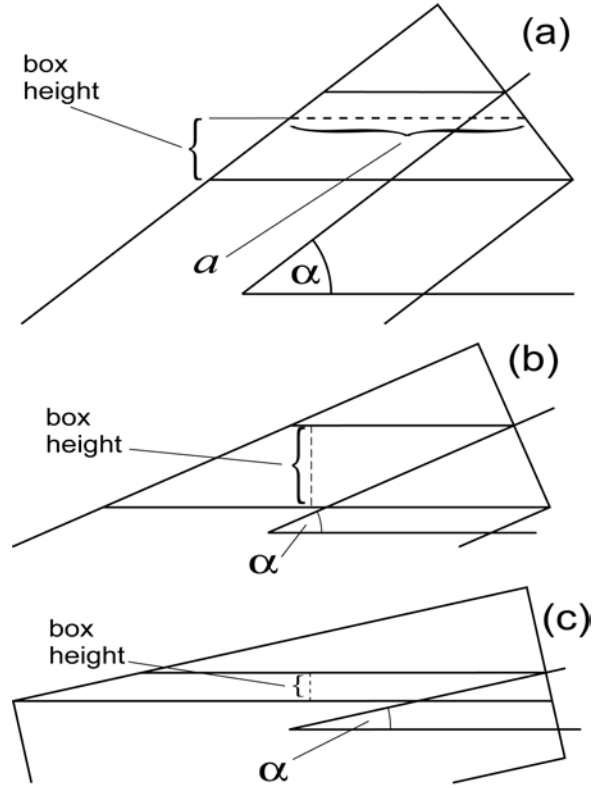


Figure 2.4. Box heights for single-truncated sections with length greater than  $2r$  and less than  $\frac{1}{2}\sqrt{L^2 + 4r^2}$ .

$$\begin{aligned}
 H_{ste2}(L, a)dL = & \hspace{15em} (2.9) \\
 & \int_{\sin^{-1} \frac{2r}{a}}^{\sin^{-1} \frac{r}{a}} N_V F(L) \cos \alpha (4r \cos \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL + \\
 & \int_{\sin^{-1} \frac{r}{a}}^{\sin^{-1} \frac{r}{a}} N_V F(L) \cos \alpha (2r \cos \alpha) d\alpha dL + \\
 & \int_{\tan^{-1} \frac{2r}{L}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (2L \sin \alpha - 2r \cos \alpha) d\alpha dL
 \end{aligned}$$

For  $\frac{1}{2}\sqrt{L^2 + 4r^2} \leq a \leq \sqrt{L^2 + r^2}$  (Figure 2.5),

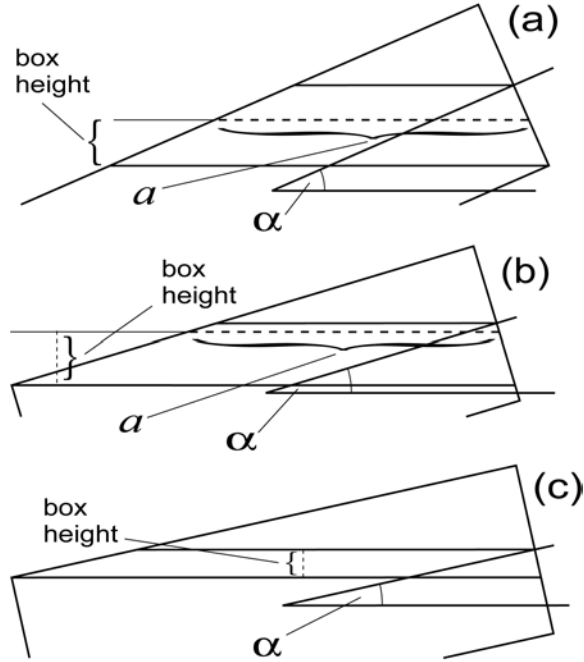


Figure 2.5. Box heights for single-truncated sections greater than  $\frac{1}{2}\sqrt{L^2 + 4r^2}$  and less than  $\sqrt{L^2 + r^2}$ .

$$\begin{aligned}
 H_{ste3}(L, a)dL = & \quad (2.10) \\
 & \int_{\tan^{-1} \frac{2r}{L}}^{\sin^{-1} \frac{2r}{a}} N_V F(L) \cos \alpha (4r \cos \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL + \\
 & \int_{\sin^{-1} \frac{r}{a}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (2L \sin \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL + \\
 & \int_{\tan^{-1} \frac{r}{L}}^{\sin^{-1} \frac{r}{a}} N_V F(L) \cos \alpha (2L \sin \alpha - 2r \cos \alpha) d\alpha dL
 \end{aligned}$$

For the  $\sqrt{L^2 + r^2} \leq a \leq \sqrt{L^2 + 4r^2}$  category (Figure 2.6), the portion of the box that corresponds to sections larger than  $a$  disappears before the angle reaches  $\tan^{-1} \frac{r}{L}$ :

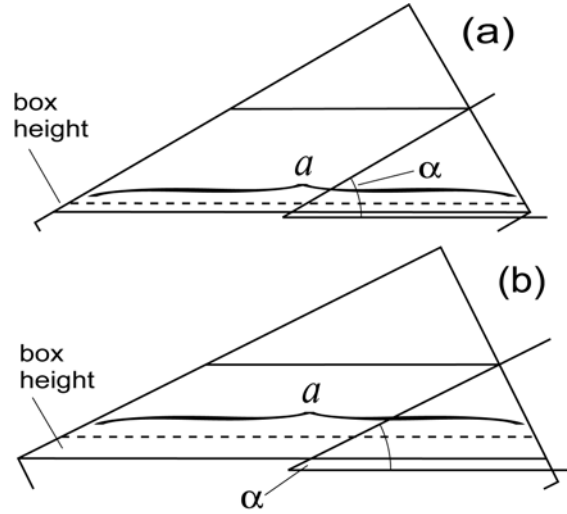


Figure 2.6. Box heights for single-truncated sections of length greater than  $\sqrt{L^2 + r^2}$ .

$$H_{ste4}(L, a)dL = \quad (2.11)$$

$$\int_{\sin^{-1} \frac{2r}{a}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (4r \cos \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL +$$

$$\int_{\cos^{-1} \frac{L}{a}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (2L \sin \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL$$

Put together, these expressions are continuous over the range of  $a$ , as demonstrated by the fact that adjoining expressions are equal at their common  $a$  values.

Determining the percentage of single-truncated sections that fall into the category  $r \leq a \leq 2r$  for unidisperse (single particle length  $L$ ) composites now becomes possible, using equation 2.8 and the expression for the total number of sections on the plane of polish (discussed below). Figure 2.7 shows the fraction of sections on the plane for psi angles between  $\pi/6$  and  $\pi/2$  for unidisperse systems of various aspect ratios. For aspect ratios

higher than 5, the fraction of sections with  $r \leq a \leq 2r$  stays below 10%. Therefore, for high aspect ratio systems we may safely leave out any consideration of this section type. This makes the analysis considerably easier from a theoretical standpoint, as otherwise the lack of a relationship between  $L$  and  $a$  in the length limits on  $a$  for the  $r \leq a \leq 2r$  length category means that final expression for the cumulative distribution would be discontinuous.

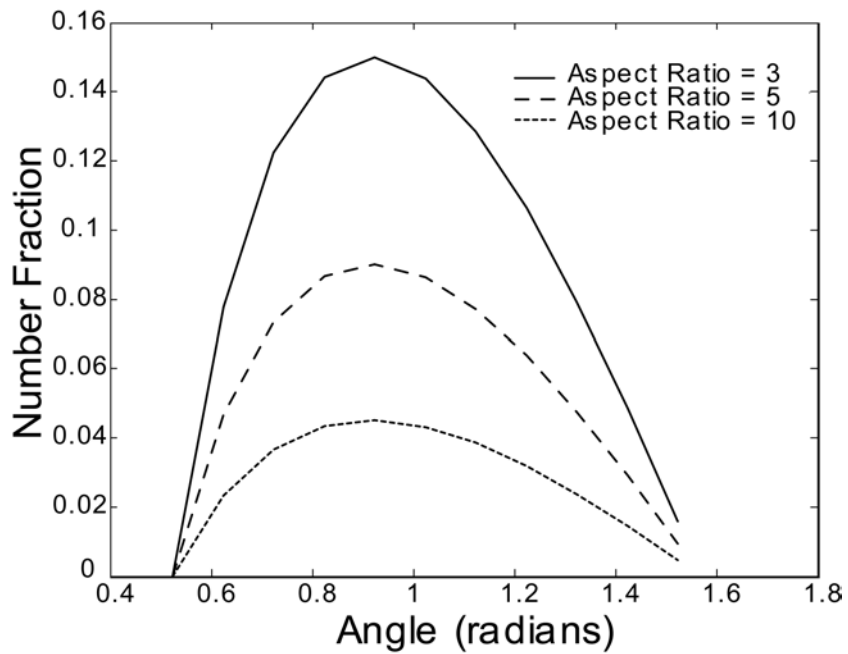


Figure 2.7. Fraction of elliptical sections wherein  $a < 2r$  for a monodisperse system at given  $\alpha$  angles. The average over all angles is the average height of each curve.

For single-truncated sections wherein  $a \geq 2r$ , the total number of sections greater than  $a$  for polydisperse systems follows by integrating equations 2.9-11 over the appropriate portions of  $L$  and summing. Integrating equation 2.9,

$$\begin{aligned}
& \int_a^{a_{\max}} N_A f_{ste2}(a') da' = \tag{2.12} \\
& \int_{2\sqrt{a^2-r^2}}^{L_{\max}} \int_{\sin^{-1}\frac{r}{a}}^{\sin^{-1}\frac{2r}{a}} N_V F(L) \cos \alpha (4r \cos \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL + \\
& \int_{2\sqrt{a^2-r^2}}^{L_{\max}} \int_{\tan^{-1}\frac{2r}{L}}^{\sin^{-1}\frac{r}{a}} N_V F(L) \cos \alpha (2r \cos \alpha) d\alpha dL + \\
& \int_{2\sqrt{a^2-r^2}}^{L_{\max}} \int_{\tan^{-1}\frac{r}{L}}^{\tan^{-1}\frac{2r}{L}} N_V F(L) \cos \alpha (2L \sin \alpha - 2r \cos \alpha) d\alpha dL
\end{aligned}$$

Integrating equation 2.10 gives

$$\begin{aligned}
& \int_a^{a_{\max}} N_A f_{ste3}(a') da' = \tag{2.13} \\
& \int_{\sqrt{a^2-r^2}}^{2\sqrt{a^2-r^2}} \int_{\tan^{-1}\frac{2r}{L}}^{\sin^{-1}\frac{2r}{a}} N_V F(L) \cos \alpha (4r \cos \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL + \\
& \int_{\sqrt{a^2-r^2}}^{2\sqrt{a^2-r^2}} \int_{\sin^{-1}\frac{r}{a}}^{\tan^{-1}\frac{2r}{L}} N_V F(L) \cos \alpha (2L \sin \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL + \\
& \int_{\sqrt{a^2-r^2}}^{2\sqrt{a^2-r^2}} \int_{\tan^{-1}\frac{r}{L}}^{\sin^{-1}\frac{r}{a}} N_V F(L) \cos \alpha (2L \sin \alpha - 2r \cos \alpha) d\alpha dL
\end{aligned}$$

and for equation 2.11,

$$\begin{aligned}
& \int_a^{a_{\max}} N_A f_{ste4}(a') da' = & (2.14) \\
& \int_{\sqrt{a^2-4r^2}}^{\sqrt{a^2-r^2}} \int_{\tan^{-1}\frac{2r}{L}}^{\sin^{-1}\frac{2r}{a}} N_V F(L) \cos \alpha (4r \cos \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL + \\
& \int_{\sqrt{a^2-4r^2}}^{\sqrt{a^2-r^2}} \int_{\cos^{-1}\frac{L}{a}}^{\tan^{-1}\frac{2r}{L}} N_V F(L) \cos \alpha (2L \sin \alpha - 2a \sin \alpha \cos \alpha) d\alpha dL
\end{aligned}$$

The full expression for all single-truncated sections then follows by adding equations 2.12-14:

$$f_{ste, a \geq 2r} = f_{ste2} + f_{ste3} + f_{ste4} \quad (2.15)$$

### Double-Truncated Ellipses

The analysis for double-truncated ellipses splits into sections in  $a$  length, similar to that for the single-truncated type. Figure 2.8 shows the box heights required for the derivation. A limit to box size does not arise, as the section length created is the same for a slice anywhere in the box. The different categories in  $a$  length arise as sections that are longer than the particle length establish a minimum  $\alpha$  that is greater than zero. Again, the changes in the box height (see Figure 2.8) as the particle rotates through the  $\alpha$  range dictates the use of multiple terms integrated over different ranges in  $\alpha$ . There are three length regimes in total: one for sections smaller than the length of the particle, and one apiece for the establishment of a new minimum angle in the high-angle regime (Figure 2.8a) and low-angle regime (Figure 2.8b). The three regimes lead to the following three equations for double-truncated ellipses:

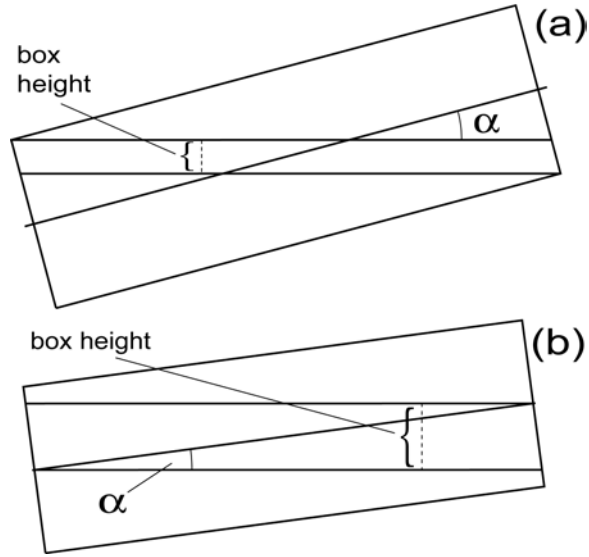


Figure 2.8. Box heights for double-truncated sections.

$$\int_a^{a_{\max}} N_A f_{dte1}(a') da' = \quad (2.16)$$

$$\int_a^{L_{\max}} \int_{\tan^{-1} \frac{r}{L}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (2r \cos \alpha - L \sin \alpha) d\alpha dL +$$

$$\int_a^{L_{\max}} \int_0^{\tan^{-1} \frac{r}{L}} N_V F(L) \cos \alpha (L \sin \alpha) d\alpha dL$$

$$\int_a^{a_{\max}} N_A f_{dte2}(a') da' = \quad (2.17)$$

$$\int_{\sqrt{a^2 - r^2}}^a \int_{\tan^{-1} \frac{r}{L}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (2r \cos \alpha - L \sin \alpha) d\alpha dL +$$

$$\int_{\sqrt{a^2 - r^2}}^a \int_{\cos^{-1} \frac{L}{a}}^{\tan^{-1} \frac{r}{L}} N_V F(L) \cos \alpha (L \sin \alpha) d\alpha dL$$

$$\int_a^{a_{\max}} N_A f_{dte3}(a') da' = \int_{\sqrt{a^2-4r^2}}^{\sqrt{a^2-r^2}} \int_{\cos^{-1}\frac{L}{a}}^{\tan^{-1}\frac{2r}{L}} N_V F(L) \cos \alpha (2r \cos \alpha - L \sin \alpha) d\alpha dL \quad (2.18)$$

Again, a full expression for double-truncated ellipses follows from adding equations 2.16-18.

### Final Expression

Summing over all section types, the total number of sections arising from full, single-truncated and double-truncated ellipses is

$$\int_a^{a_{\max}} N_A f(a') da' = \int_a^{a_{\max}} N_A [f_{fe}(a') + f_{ste}(a') + f_{dte}(a')] da' \quad (2.19)$$

Equation 2.19 corresponds to equation 2.1. One may, of course, simplify the expression somewhat by carrying out the integration in  $\alpha$ , or further manipulate the expression so as to produce a result for the number of sections found over a certain range in  $a$  and  $r$  (by subtracting cumulative equations of the desired limits in  $a$  and integrating the entire expression over the interval in  $r$ ). One may also produce an expression for the number density found at a given  $(r,a)$  by differentiating equation 2.19.

## Discussion

### Relation to Line Length

An expression for the total number of sections per unit area arises from an analogy between the polydisperse cylinder problem and the problem of determining the number of points per unit line length for lines of random orientations. The relevant equation is<sup>8</sup>

$$L_V = 2\langle Q_A \rangle \quad (2.20)$$

where  $L_V$  is the length per unit volume, and  $Q_A$  is the number of points per unit area on the sectioning plane. In relation to the concepts used in this paper,  $Q_A$  may be viewed as the number of intersections of particle axes with the sectioning plane. Since the preceding analysis treated only sections for which such an intersection occurs, we have

$$Q_A = N_A = \int_{a_{\min}}^{a_{\max}} N_A f(a') da' \quad (2.21)$$

If we further observe that  $L_V$  – the axis length per unit volume – is simply the average particle length times the number of particles per unit volume, then equation 20 becomes

$$N_A = \frac{1}{2} N_V \bar{L} \quad (2.22)$$

where  $\bar{L}$  is the average particle length.

To check the derived expression for the cumulative distribution of sections, evaluate the expression for each section type at the lowest possible  $a$  value. In other words, substitute  $2r$  for  $a$  in equation 2.7,  $r$  in equation 2.8 (then integrating from  $L = 0$  to  $L =$

$L_{\max}$ ), and  $2r$  in equations 2.16-18 (assuming that the minimum particle aspect ratio is 1).

Adding these terms together gives

$$\begin{aligned}
 & \int_{a_{\min}}^{a_{\max}} N_A f(a') da' = N_A = & (2.23) \\
 & \int_0^{L_{\max}} \int_{\tan^{-1} \frac{2r}{L}}^{\pi/2} N_V F(L) \cos \alpha (L \sin \alpha - 2r \cos \alpha) d\alpha dL + \\
 & \int_0^{L_{\max}} \int_{\tan^{-1} \frac{2r}{L}}^{\pi/2} N_V F(L) \cos \alpha (2r \cos \alpha) d\alpha dL + \\
 & \int_0^{L_{\max}} \int_{\tan^{-1} \frac{r}{L}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (2L \sin \alpha - 2r \cos \alpha) d\alpha dL + \\
 & \int_0^{L_{\max}} \int_{\tan^{-1} \frac{r}{L}}^{\tan^{-1} \frac{2r}{L}} N_V F(L) \cos \alpha (2r \cos \alpha - L \sin \alpha) d\alpha dL + \\
 & \int_0^{L_{\max}} \int_0^{\tan^{-1} \frac{r}{L}} N_V F(L) \cos \alpha (L \sin \alpha) d\alpha dL \\
 & = \int_0^{L_{\max}} \int_0^{\pi/2} N_V F(L) (L \sin \alpha \cos \alpha) d\alpha dL = \frac{1}{2} N_V \bar{L}
 \end{aligned}$$

### Simulation Results

A microstructural simulation further validated the equation for the cumulative distribution in  $a$  derived above. Figure 2.9 shows the results of the simulation. The agreement between the simulated and theoretical curve – the latter derived by inputting the three-dimensional  $F(L)$  distribution used in creation of the simulation space into the 2-dimensional cumulative distribution equation – is quite good.

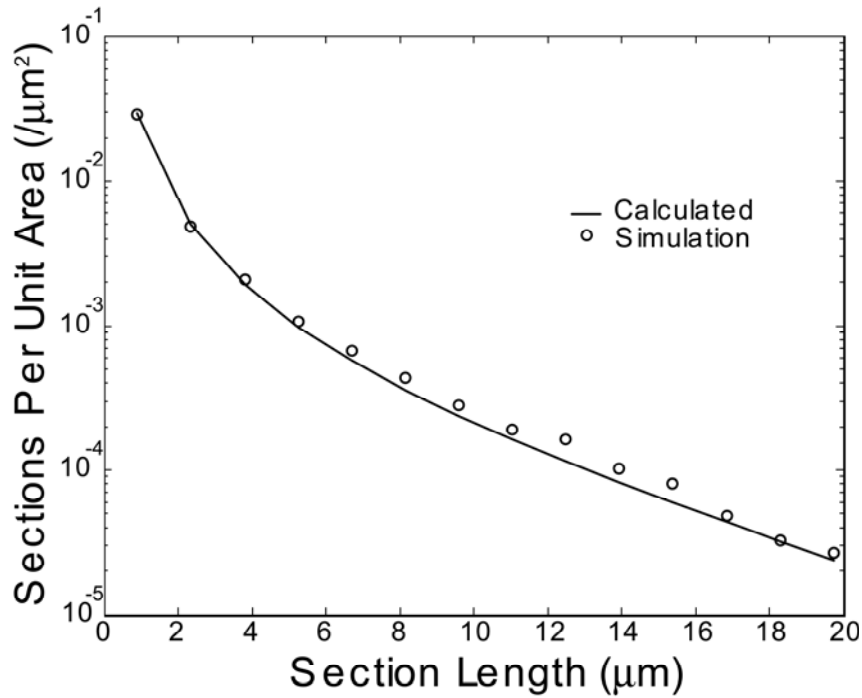


Figure 2.9. Monte Carlo simulation correspondence to the unfolding equation.

### Stability and Uniqueness of Solutions

Although an analytical solution to the problem represented by equations 2.9-19 was not attempted, any solution to the whole problem must take into account the problem's practical ill-posedness. A reasonable sample size is unlikely to produce a stable number of sections of large  $a$ . A straightforward numerical solution that extinguishes the instability involves assigning a functional form to  $F(L)$ . One may then simply use a non-linear least squares optimization. A likely candidate function for many systems is the lognormal function, as lognormal functions describe particle-size distributions resulting from natural processes.<sup>13-14</sup>

Thouless and co-authors suggested that the distribution of two-dimensional section lengths displays too little variation over different average particle aspect ratios to make a good practical determination of three-dimensional length parameters.<sup>7</sup> While Figure 2.10 does, in fact, show a moderate to small change in the curve of equation 2.19 versus  $a$  with average three-dimensional aspect ratio, it is the opinion of the present authors that the change is significant enough for practical purposes. Choosing a functional form for  $F(L)$  and solving the system through a non-linear least squares optimization method requires a first guess of distribution parameters (and  $N_v$ , since this must be solved for as well). Guessing close to the solution – possible if something is known about the fiber growth process or if inclusions have been measured qualitatively by some other method – will reduce the chances of an errant convergence.

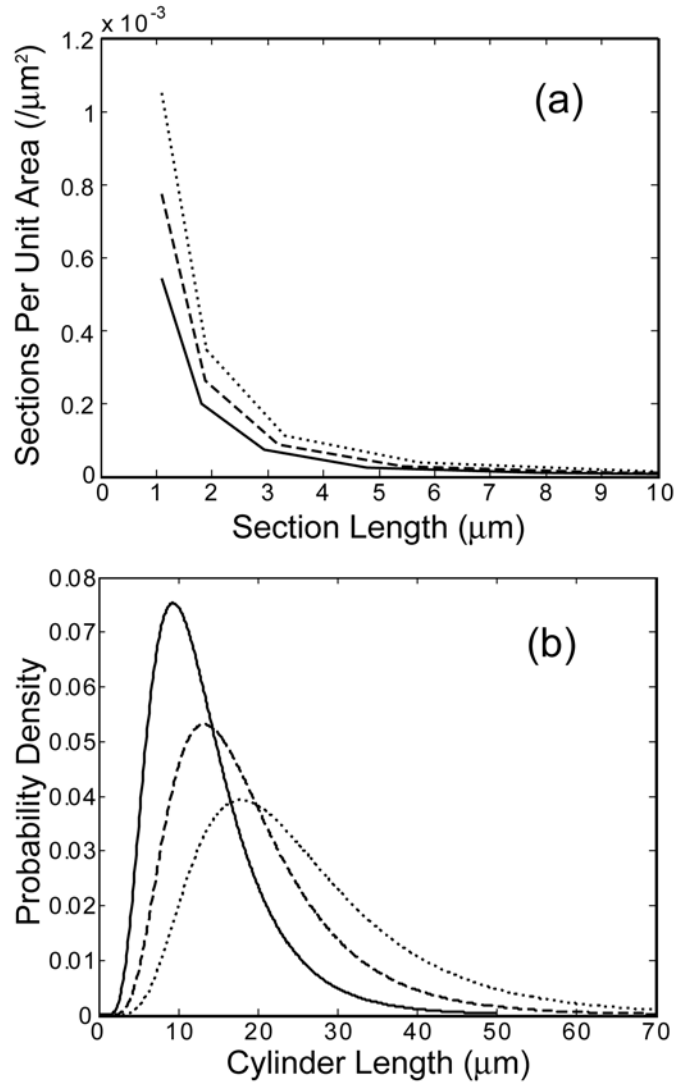


Figure 2.10. The results of the unfolding equation compared to input three-dimensional whisker distributions showing different average lengths. The different line weights of part (b) correspond to the output of the unfolding equation using the same distribution in part (a).

## Conclusion

A general equation relating the three-dimensional distribution of cylindrical particle lengths to the two-dimensional distribution of ellipse and partial ellipse lengths found on two-dimensional polished sections was derived. The equation was verified using

mathematical and computational techniques. The final equation is very complex, consisting of a number of integral terms, and practical applications of the equation may encounter stability problems. However, these stability problems can be alleviated through the designation of a functional form for the three-dimensional distribution, such as a lognormal distribution.

## CHAPTER 3

### TRIVARIATE, STEREOLOGICAL LENGTH-RADIUS-ORIENTATION UNFOLDING DERIVED AND APPLIED TO ALUMINA-SILICON CARBIDE WHISKER COMPOSITES

#### **Introduction**

It is well known that short fibers can improve the fracture toughness of structural ceramics through crack bridging, crack deflection and fiber pullout mechanisms.<sup>16-19</sup> In addition, the orientation and size distribution of fibers embedded in the ceramic matrix strongly affect the mechanical properties of the composite.<sup>2,20</sup> These facts naturally lead to an interest in microstructural characterization studies for fiber composites. The preponderance of these studies – especially with respect to the ceramic matrix composite – fall into the category of orientation characterization.<sup>21-27</sup> But the techniques laid out in these works do not produce a measurement of size (length), nor do they measure the correlation between size and orientation.

The size of the fibers deserves some attention in the context of composites with a preferred orientation. As cited above, the aspect ratios of the fibers influence mechanical properties, such as fracture toughness and strength (see especially reference 19). Aspect ratio also influences electrical properties, especially in ceramic matrix composites where the matrix material is much less conductive than the inclusions.<sup>28-29</sup> And the correlation

between length and orientation – apparent only when measuring both properties at once – influences macroscopic properties as well.

Multivariate stereological unfolding techniques reveal correlations between measured parameters. An overview of such techniques appears in the introduction to Chapter 2. Stereological unfolding procedures require a choice of idealized shapes to represent the inclusions; while this does not strictly conform to reality for most filler materials, the cylinder serves as a good approximation for chopped fibers and / or whiskers. This chapter develops a size-orientation unfolding for the cylindrical inclusions of an axisymmetric composite. It makes use of two data sets – that taken from the horizontal sectioning plane (perpendicular to the symmetry axis) and one from the vertical sectioning plane (any plane parallel to the symmetry axis). Two equations are derived – one for each sectioning plane – to describe the size and orientation of elliptical sections found on the planes of polish in terms of the three-dimensional size-orientation distribution. These equations are then numerically solved to yield a three-dimensional, trivariate size-orientation distribution. The procedure so derived is demonstrated on a common ceramic matrix composite, SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub>.

## **Theory**

### Unfolding Equations

The derivation of a bivariate length-radius unfolding procedure for polydisperse cylinders embedded randomly in a matrix phase is the subject of Chapter 2. The key result of the derivation is an equation that relates the cumulative distribution for the length of all of the different elliptical sections (see Figure 2.1) on a two-dimensional sectioning

plane (at constant radius) with the three-dimensional length distribution for cylindrical particles. The equation is too cumbersome to reprint here in full, but it is a collection of terms similar in form to the following equation, which describes the length distribution of full ellipses on the sectioning plane:

$$\int_a^{a_{\max}} N_{A,h} f_{fe}(a') da' = \int_{\sqrt{a^2-4r^2}}^{L_{\max}} \int_{\tan^{-1}\frac{2r}{L}}^{\sin^{-1}\frac{2r}{a}} N_V F(L) \cos\alpha (L \sin\alpha - 2r \cos\alpha) d\alpha dL \quad (2.7)$$

where  $a$  is the full major axis of the ellipse (twice the normal major axis),  $r$  is the minor axis of the ellipse and is equal to the particle radius,  $L$  is the particle length,  $\alpha$  is the angle of inclination between the particle and the sectioning plane (as per Figure 2.2),  $L_{\max}$  is the maximum length of particles in the section,  $N_V$  and  $N_{A,h}$  are the number of particles per unit volume and ellipses per unit area, respectively, and  $F$  and  $f_{fe}$  are the distribution functions for particle lengths and ellipse lengths. The full unfolding equation for which equation 2.7 forms one term may be applied iteratively to sections of half-ellipse or greater at constant radii to produce a bivariate length-radius unfolding. (See reference 15 for a representative application of the bivariate unfolding to a randomly oriented whisker composite.)

Equation 2.7 assumes that the angle of incidence to the sectioning plane,  $\alpha$ , is random and therefore the function  $F$  does not depend on it. However, it is well known that hot pressed whisker ceramic composites develop a preferred orientation (see Figure 3.1).<sup>19,29-30</sup> Equation 2.7 – and the full unfolding equation of which it is a part – remain valid if  $\alpha$

becomes nonrandom. The only change to the equation in this case is the addition of an  $\alpha$  term in the dependence of  $F$ . Equation 2.7 now becomes a double-integral equation over  $L$  and  $\alpha$ , and the solution  $F(L, \alpha)$  is the bivariate length-orientation distribution sought.

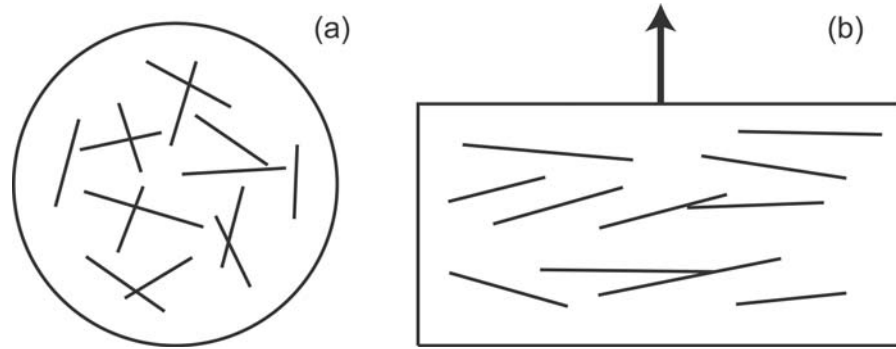


Figure 3.1. Schematic of the hot pressed composites. Part (a) is a view perpendicular to the hot pressing direction, while view (b) is parallel to the hot pressing direction (indicated with an arrow). After Gerhardt and Ruh.<sup>27</sup>

The univariate distribution of ellipse lengths on the sectioning plane cannot by itself uniquely determine a bivariate length-orientation distribution; more data and another unfolding equation are needed. Vertical sectioning planes – those planes that correspond to the view presented in Figure 3.1b – contain easily measurable information on the angle of incidence  $\alpha$  through measurements of the angle  $\delta$  that the elliptical sections on the vertical plane make with the horizontal (see Figure 3.2). However, measurement of the distribution of  $\delta$  angles on the vertical plane is not the same as a direct measurement of the distribution of  $\alpha$  angles – it is a convolution of the length distribution, the  $\alpha$  angle distribution and the uniform  $\beta$  distribution.

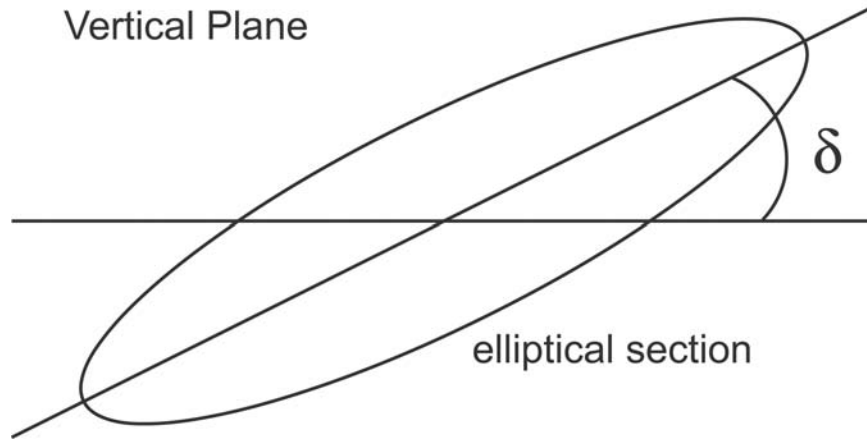


Figure 3.2. Definition of the angle  $\delta$ . The figure shows an elliptical section occurring on the vertical plane.

Derivation of an equation relating the distribution of  $\delta$  angles to the trivariate length-radius-orientation distribution  $F$  proceeds as follows. Two orientation angles – the preferred angle  $\alpha$  and the random angle  $\beta$  – fully describe the orientation of a particle in the composite. The three angles are related by

$$\tan \alpha = \tan \delta \sin \beta \quad (3.1)$$

In addition, the angle of incidence to the vertical plane,  $\gamma$  (defined as the angle  $\alpha$  but with respect to the vertical plane), can be expressed in terms of  $\alpha$  and  $\beta$ :

$$\sin \gamma = \cos \alpha \cos \beta \quad (3.2)$$

Following equation 3.1, any particle that intersects the vertical plane will create a section of in-plane angle greater than  $\delta$  if its  $\alpha$  angle is greater than  $\arctan(\tan \delta \sin \beta)$ . Considering only sections that are half-ellipse or greater, equation 3.2 determines the height of the box (per unit area) inside of which any particle of length  $L$  to  $L + dL$  and

orientation  $\alpha$  to  $\alpha + d\alpha$  and  $\beta$  to  $\beta + d\beta$  will intersect the vertical plane as  $L \cos \alpha \cos \beta$ . The total number of intersections per unit vertical plane area having in-plane angle greater than  $\delta$  then becomes

$$\int_{\delta}^{\pi/2} N_{A,v} g(\delta') d\delta' = \tag{3.3}$$

$$\frac{2}{\pi} \int_0^{L_{\max}} \int_0^{\pi/2} \int_{\tan^{-1}(\tan \delta \sin \beta)}^{\pi/2} N_V L F(L, \alpha) \cos^2 \alpha \cos \beta d\alpha d\beta dL$$

where  $g$  is the two dimensional angular distribution function on the vertical plane and  $N_{A,v}$  is the total number of sections per unit area on the vertical plane.

Checking the accuracy of equation 3.3 is straightforward. If particle orientations are random, the total number of sections on the vertical plane should be equal to  $\frac{1}{2} N_V \bar{L}$ , where  $\bar{L}$  is the average length of particles in the composite.<sup>5</sup> Removing the  $\alpha$  dependence from  $F$ , substituting 0 for  $\delta$  and carrying out the integration, equation 3.3 becomes

$$N_{A,v} = \tag{3.4}$$

$$\frac{2}{\pi} \int_0^{L_{\max}} N_V L F(L) dL \int_0^{\pi/2} \int_0^{\pi/2} \cos^2 \alpha \cos \beta d\alpha d\beta = \frac{1}{2} N_V \bar{L}$$

### Distribution Function

It is certainly tempting to search for a simultaneous inversion of equation 3.3 and the aspect ratio unfolding equation of which equation 2.7 forms a part. However, it proves more efficient to assume a functional form for  $F$  and solve for the parameters of the distribution for discrete radius values. Aside from obviating the need to analyze a very

complicated set of simultaneous integral equations, this method solves a stability problem and a potential uniqueness problem generated by the aspect ratio unfolding equation.<sup>18</sup> Additionally, the literature offers guidance on the form of the distribution function  $F$ .

Particles acquiring a size distribution through a natural process tend to form a lognormal distribution.<sup>13-14</sup> The orientation distribution of whiskers in hot-pressed SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub> composites has been shown to take the form of a truncated, half normal distribution on the interval  $\alpha \in \{0, \pi/2\}$  with a mean value at zero (for the definition of  $\alpha$  used here).<sup>23,25</sup> Taken together, these observations suggest a bivariate, lognormal-truncated half normal distribution as the functional form of  $F$ :

$$F(L, \alpha) = \frac{1}{L s_L \sqrt{2\pi(1-c^2)} \int_0^{\pi/2} \exp\left[-\frac{(\alpha')^2}{2s_\alpha^2}\right] \cos \alpha' d\alpha'} \times \dots \quad (3.5)$$

$$\dots \times \exp\left[\frac{-1}{2(1-c^2)} \left( \frac{\log^2\left(\frac{L}{m}\right)}{s_L^2} - \frac{2c \log\left(\frac{L}{m}\right)\alpha}{s_L s_\alpha} + \frac{\alpha^2}{s_\alpha^2} \right)\right]$$

In equation 3.5,  $m$  and  $s_\alpha$  are the scale parameters of the lognormal and normal distributions, respectively,  $s_L$  is the shape parameter of the lognormal distribution and  $c$  is the correlation coefficient.  $c$  falls between  $-1$  and  $+1$ , with positive values indicating a positive correlation between  $L$  and  $\alpha$ , and negative values the opposite. In the hot pressed SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub> composite, any correlation between the length of whiskers and the orientation angle should be negative. The optimum value of the parameter set  $(N_V, m, s_L, s_\alpha, c)$  with respect to the experimental data called for in the unfolding equations should form a unique solution to the overall bivariate unfolding at constant radii.

Using equation 3.5, one may also derive expressions for the distribution represented by the left-hand side of equation 3.3 in terms of the bivariate distribution function parameters. Re-writing equation 3.3,

$$\int_{\delta}^{\pi/2} N_{A,v} g(\delta') d\delta' = \tag{3.6}$$

$$\frac{2}{\pi} \int_0^{\pi/2} \int_{\tan^{-1}(\tan \delta \sin \beta)}^{\pi/2} \Omega N(\alpha) \cos^2 \alpha \cos \beta d\alpha d\beta$$

where  $\Omega$  and  $N$  are a multiplier and normalized distribution in  $\alpha$ , respectively, that can be fit to the experimental data represented by the left-hand side of the equation. Comparing with equation 3.3, we find that

$$\Omega N(\alpha) = \int_0^{L_{\max}} N_V L F(L, \alpha) dL \tag{3.7}$$

Analyzing equation 3.7 in light of the distribution function given in equation 3.5,

$$\Omega N(\alpha) = \tag{3.8}$$

$$\frac{N_V \exp\left[\frac{-\alpha^2}{2s_\alpha^2}\right]}{T_1} \int_0^{L_{\max}} \frac{1}{s_L \sqrt{2\pi(1-c^2)}} \exp\left\{\frac{-\left[\log\left(\frac{L}{m}\right) - \frac{s_L}{s_\alpha} c \alpha\right]}{2(1-c^2)s_L^2}\right\} dL$$

where

$$T_1 = \int_0^{\pi/2} \exp\left[\frac{-(\alpha')^2}{2s_\alpha^2}\right] \cos \alpha' d\alpha'$$

The expression under the integral sign in equation 3.8 takes the form of the mean value of a lognormal distribution. Equation 3.8 therefore simplifies to

$$\Omega N(\alpha) = \frac{N_V \exp\left[\frac{-\alpha^2}{2s_\alpha^2}\right]}{T_1} m \exp\left[\frac{s_L}{s_\alpha} \rho \alpha + \frac{s_L^2(1-c^2)}{2}\right] \quad (3.9)$$

Re-arranging,

$$\Omega N(\alpha) = \frac{N_V m \exp\left[\frac{s_L^2(1-c^2)}{2}\right]}{T_1} \exp\left[\frac{s_L}{s_\alpha} c \alpha - \frac{\alpha^2}{2s_\alpha^2}\right] \quad (3.10)$$

Completing the square in the right-hand brackets and simplifying,

$$\Omega N(\alpha) = \frac{N_V m \exp\left[\frac{s_L^2}{2}\right]}{T_1} \exp\left[\frac{-(\alpha - s_L s_\alpha c)^2}{2s_\alpha^2}\right] \quad (3.11)$$

Given the fact that  $N(\alpha)\cos(\alpha)$  is normalized,

$$N(\alpha) = \frac{1}{T_2} \exp\left[\frac{-(\alpha - s_L s_\alpha c)^2}{2s_\alpha^2}\right] \quad (3.12)$$

$$\Omega = \frac{T_2}{T_1} N_V m \exp\left(\frac{s_L^2}{2}\right) \quad (3.13)$$

where

$$T_2 = \int_0^{\pi/2} \exp\left[\frac{-(\alpha' - s_L s_\alpha c)^2}{2s_\alpha^2}\right] \cos \alpha' d\alpha'$$

Clearly,  $N$  is a normal distribution on the interval  $\{0, \pi/2\}$  with variance  $s_\alpha$  and mean  $s_L s_\alpha c$ . Since  $s_L s_\alpha$  is always positive and nonzero, the correlation coefficient is positive

when the mean is positive, negative when the mean is negative, and when the mean is at zero the length and orientation are uncorrelated. Also, when  $c = 0$ ,  $T_1 = T_2$  and from equations 3.12 & 13,

$$N(\alpha) = \frac{1}{T_1} \exp\left[\frac{-\alpha^2}{2s_\alpha^2}\right] \quad (3.14)$$

$$\Omega = N_V \bar{L} \quad (3.15)$$

Equations 3.14 and 15 imply that when the correlation coefficient is equal to zero, it is possible to unfold the orientation distribution from the angular distribution on the vertical plane alone.

If the correlation between the orientation and length is unknown or less than zero, equations 3.12 and 13 offer a method of splitting up the five-parameter fitting by first solving for  $\Omega$  and the mean and variance parameters of  $N$  from the vertical section, then using the equations to reduce the aspect ratio unfolding to a two parameter fitting. Computationally, this is much more efficient – especially if the experimental data is noisy, such that the optimization process must contend with several local minima. However, fitting five parameters to both unfolding equations simultaneously will always produce more accurate results than a sequential procedure.

### **Application to SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub> Composites**

Hot pressed SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub> composites were produced by Advanced Composite Materials Corporation. A 20% SiC<sub>w</sub> volume fraction sample was ground to 1 μm roughness with diamond paste, then polished with 0.02 μm colloidal silica suspension. The suspension

also chemically etched the sample through its mild alkalinity. The samples were then analyzed in a Leo Gemini scanning electron microscope using the InLens setting. The results of the grinding and polishing were quite good; the SEM micrographs (Figure 3.3) showed clear distinctions between SiC whisker and Al<sub>2</sub>O<sub>3</sub> matrix. Montage images of the sample were taken at 3000X (5-by-5 montage) for the plane perpendicular to the hot-pressing direction (the horizontal plane) and at 4750X (10-by-10 montage) for the vertical plane.

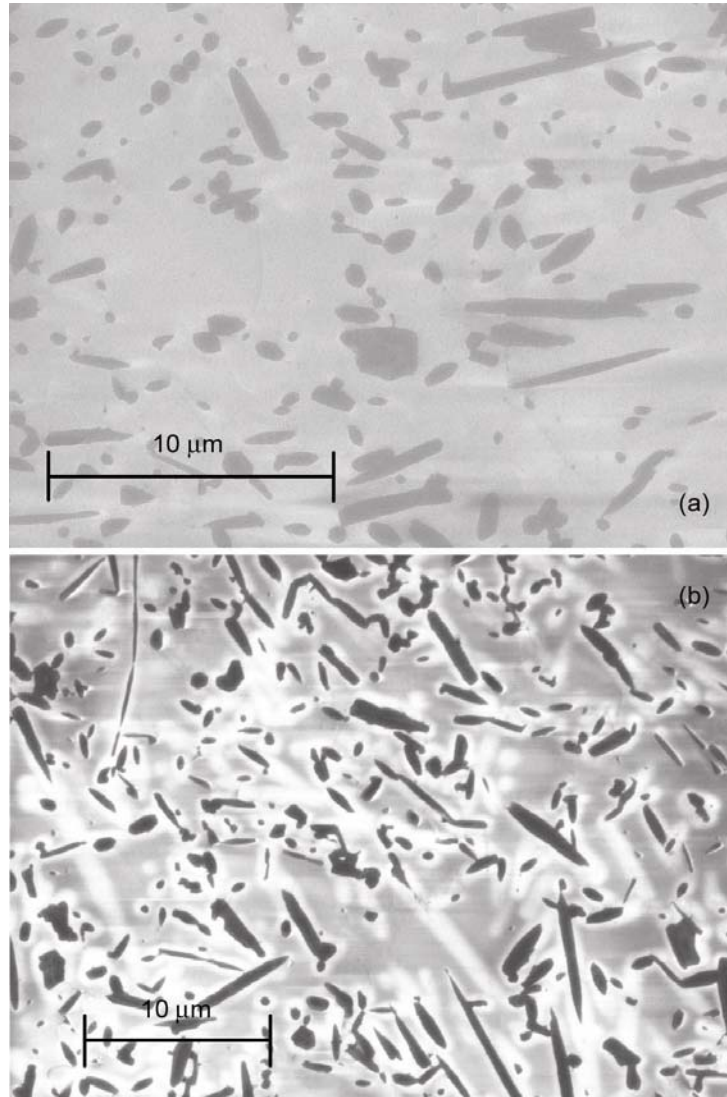


Figure 3.3. SEM images of hot pressed SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub> composite, taken from the (a) vertical and (b) horizontal sections.

The images were thresholded and analyzed using a Zeiss KS400 image analysis system. The polished surfaces revealed a significant amount of intersections between particles and clumps of SiC<sub>w</sub>. Most of the intersections were removed manually through KS400, while the remaining clumps and indistinguishable intersected particles were rejected. The final volume fraction analyzed was 12.96 % for the horizontal plane, and 12.46 % for the vertical plane. These volume fractions are very close, but there were

some slight corrections made to the experimental data of the two highest radius classes due to differing volume fractions.

Angular distributions were then measured on the vertical plane, and length distributions were measured on the horizontal plane (see Figure 3.1). The experimental data was then split into four radius categories of 0.225, 0.325, 0.425 and 0.525 microns. Approximately 11,300 particles were measured on the vertical section, while approximately 4,200 were measured on the horizontal section. The smallest number of sections fell into the largest radius class, in which there were 320 particles (spread over 17 histogram bins) in the vertical section and 77 particles (spread over 8 bins) in the horizontal section.

Instead of a simultaneous fitting of the distribution parameters using the two unfolding equations, the split procedure described in the previous section was used. The fits were made using Matlab and Mathematica optimization routines.

## **Experimental Results and Discussion**

The tabulated results for the distribution parameters at different radii appear in Table 3.1, along with the corresponding fits to the experimental data in Figures 3.4 and 5. The different parts of Figure 3.4 are the fits to the orientation unfolding equation at constant radii, while the different parts of Figure 3.5 are the fits to the length unfolding equation. Note the different scale in the different parts of the figures. The quality of the fits is fair, and could improve with more data and stronger optimization tools. The fits to the length data (horizontal plane) show a closer correlation than the orientation fits, but the shape of the orientation fitting curve clearly mimics the experimental data for all radius classes.

The close approximation of the shape of the fitted curve to that of the data in both length and orientation fitting regimes suggests that equation 3.5 is an accurate functional form for the length-orientation distribution of particles.

Table 3.1. Distribution parameters.

$R$ ( $\mu\text{m}$ )	$N_V$ ( $\mu\text{m}^{-2}$ )	$m$ ( $\mu\text{m}$ )	$\bar{L}$ ( $\mu\text{m}$ )	$s_L$	$s_\alpha$ (radians)	$c$
<b>0.225</b>	0.07	5.5	5.56	0.15	0.413	0
<b>0.325</b>	0.0205	8.5	8.89	0.3	0.5	0
<b>0.425</b>	0.005	12.25	12.81	0.3	0.4	-0.3
<b>0.525</b>	0.0007	22.2	22.82	0.234	0.376	-0.475

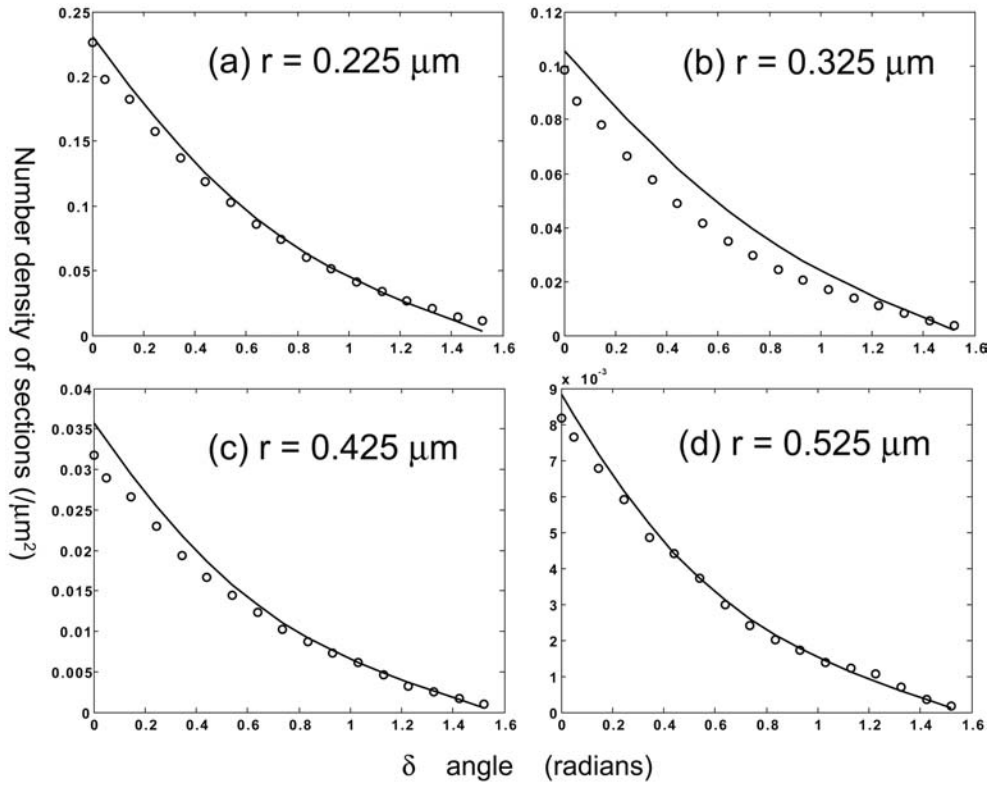


Figure 3.4. Fits of the distribution parameters to the experimental data taken from the vertical section, via the orientation unfolding equation (equation 3.3). Figures (a)-(d) refer to rows 1 through 4 of Table 3.1, respectively. In each figure, the abscissa values are  $\delta$  angle, in radians, and the ordinate values are cumulative number per unit area, in  $\mu\text{m}^{-2}$ .

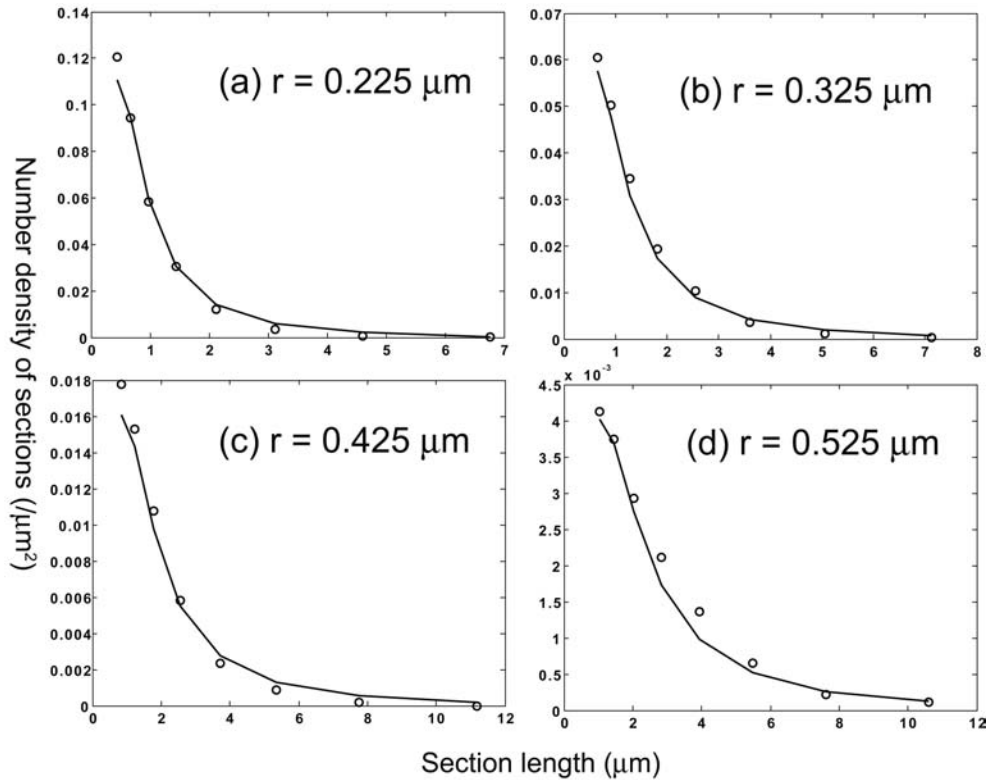


Figure 3.5. Fits of the distribution parameters to the experimental data taken from the horizontal section, via the aspect ratio unfolding equation. Figures (a)-(d) refer to rows 1 through 4 of Table 3.1, respectively. In each figure, the abscissa values are the length of elliptical sections, in  $\mu\text{m}$ , and the ordinate values are cumulative number per unit area, in  $\mu\text{m}^{-2}$ .

Substituting the parameters from Table 3.1 into equation 3.5 produced the bivariate histograms of Figure 3.6. Note that the parameter  $N_V$  acts as a scaling factor. The histograms of Figure 3.6 constitute the trivariate  $R$ - $L$ - $\alpha$  distribution sought. The properties of the calculated distribution generally fit expectations. By far the largest contribution to the overall distribution in terms of relative numbers of particles is the smallest radius class, at over 72.8% of all particles. In fact, the two smallest radius classes together (corresponding to particle diameters of 0.45 and 0.65 microns, respectively) constitute 94.1% of the total whisker population. This corresponds qualitatively to the manufacturer's observations of the as-grown whiskers, for which the smaller diameter

particles form the vast numerical majority. The average length for each radius class is  $\bar{L}_R = m_R \exp(0.5s_{L,R}^2)$ , where the subscript  $R$  indicates that the measurement is taken over a constant radius. This places the average aspect ratios of the particles in the range of approximately 12-22, steadily increasing as radii increase. The manufacturer reports the average aspect ratios of the as-grown whiskers as 14 and above (especially for the smaller radius particles), which matches well with the measured distribution considering the likely breakage during composite processing. Finally, correlation coefficients (furthest right column of Table 3.1) become steadily more negative as whisker radii increase, indicative of a stronger preferred orientation for the longer whiskers. This is also expected as the moment of inertia of the particles increases with particle length.

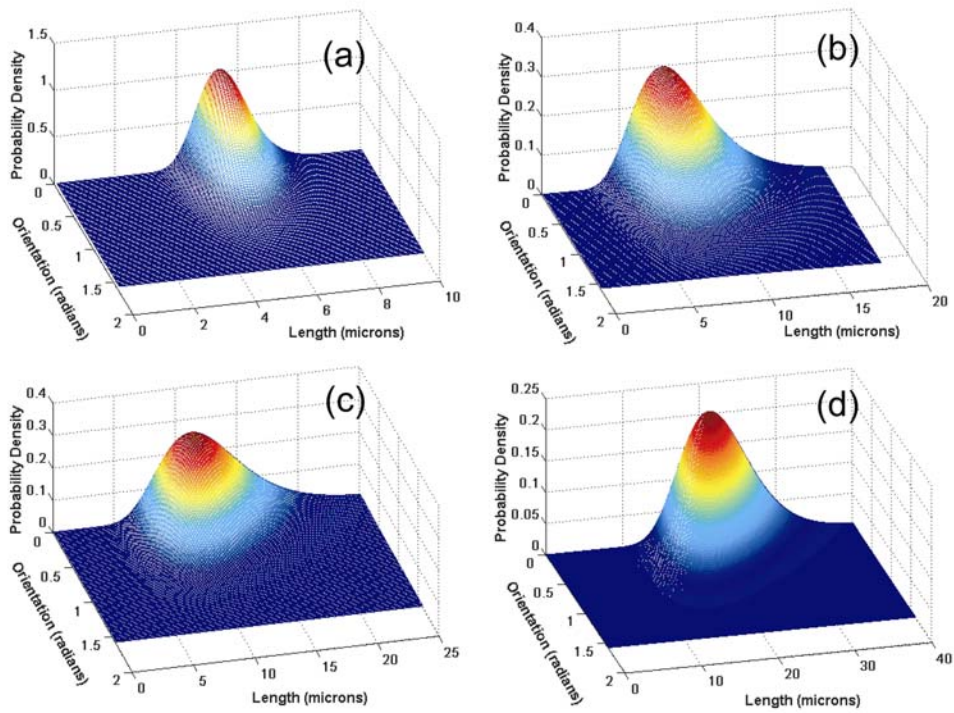


Figure 3.6. Bivariate length-orientation distributions at constant radii. (a)-(d) refer to rows 1-4 of Table 3.1, respectively.

## Conclusion

A stereological method of determining trivariate length-radius-orientation distributions for chopped fiber or whisker composites was developed and demonstrated for a hot-pressed 20% SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub> specimen. The technique relies on the identification of a functional form of the three-dimensional distribution. Such a distribution was proposed and validated for the demonstration material. A fit to the distribution was accomplished through a non-linear least squares optimization.

PART II

NON-DESTRUCTIVE  
CHARACTERIZATION OF CERAMIC  
MATRIX COMPOSITES

## CHAPTER 4

### INTERPRETING IMPEDANCE RESPONSE OF SILICON CARBIDE WHISKER / ALUMINA COMPOSITES THROUGH MICROSTRUCTURAL SIMULATION

#### **Introduction**

As discussed in the introduction to Chapter 3, the fibrous microstructure of anisotropic ceramic matrix composites – including the size, orientation and dispersion quality of the fibers – strongly affects the mechanical properties of the material, especially the fracture toughness.<sup>3,17-19</sup> The fact that these microstructural parameters also influence the electrical response of these composites opens the door to electrically based non-destructive testing techniques. In particular, impedance-based techniques offer a promising route due to the sensitivity of impedance response to microstructural changes.<sup>31</sup>

When discussing the electrical response of fiber composites – especially those for which the filler phase conductivity is much higher than that of the matrix phase – it is useful to distinguish between materials above and below the percolation threshold. For many materials, fracture toughness experiences a maximum at filler volume fractions above the material's percolation threshold.<sup>3</sup> Fiber connectivity will dominate the electrical response of these materials. Therefore, initial investigations toward finding links between fiber microstructure and electrical response should focus on the role of fiber connectivity.

Several studies on the orientation dependence of impedance response in percolating composites are available,<sup>29,31-33</sup> as are studies of chopped fiber composites at filler fractions below the percolation threshold.<sup>34-35</sup> It is clear from the available studies that orientation does have an effect on particle connectivity. However, quantitative analysis has heretofore not been undertaken. Additionally, it is clear that the aspect ratio of fibers also has an effect on particle connectivity – and probably a more critical effect than orientation in most cases.<sup>36-38</sup> However, the combined effect of aspect ratio and orientation on fiber connectivity has yet to be studied in a quantitative manner. But clearly, quantitative relationships linking fiber size, orientation, connectivity and conductivity are needed before electrical testing techniques can be applied as a non-destructive microstructural characterization technique to percolating composites.

The literature lacks straightforward, quantitative connections between fiber microstructure and connectivity. Most well-known analytical investigations focus on predictions of percolation thresholds.<sup>37,39-41</sup> Effective medium studies linking composite microstructure to conductivity are either generally based on simple cases (see Landauer for an excellent overview of these<sup>42</sup>) or empirically related to a simple microstructural parameter such as volume fraction.<sup>43</sup>

The underlying simplicity of this complex problem suggests that computational approaches might best serve to advance our understanding of the role connectivity plays in the conductivity of fiber composites. Previous authors used Monte Carlo simulations to predict percolation thresholds and simulate conducting pathways in two and three

dimensions for systems of rods or cylinders.<sup>36,38,44</sup> But probing the material with an A/C signal will require different assumptions from the DC case.

This chapter reports a Monte Carlo simulation modeling connectivity in  $\text{Al}_2\text{O}_3\text{-SiC}_w$  composites analyzed through impedance techniques. The simulation explicitly measures the effect of connectivity in terms of electrical transport through the filler phase. The simulation does not explicitly measure the effect of any contact resistance or transport through the matrix phase, although these effects are implicit in any results for the conductivity of the whisker phase. Still, investigating the composite through such a simulation represents the logical first step toward fully interpreting the impedance response.

### **Simulation**

The simulation is object-defined, using capped cylinders to represent the whisker phase. The Monte Carlo process then simply places cylinders – of sizes and orientations chosen from a given distribution – stochastically into the matrix. It uses a periodic boundary condition and places up to 35,000 particles into the sample space. The simulation thus produces perfectly dispersed microstructures – a deviation from reality that will be discussed below.

For each simulation run, 35,000 cylinders were placed in simulation space, except for the 30% runs, where processing time necessitated a reduction to 20,000 – 25,000 cylinders. The simulation calculates percolation with respect to the top and bottom of the simulation space only, as this corresponds to the direction of impedance measurement – parallel to the axis of symmetry / hot pressing direction (see Figure 3.1.). However, due to

the finite nature of the simulation space, an edge effect arises in terms of the determination of the percolating cluster. Specifically, a percolating cluster may “exit” the simulation space through a lateral boundary wall before achieving the full span from top to bottom. To mitigate this effect, the simulation spaces were flattened, minimizing the lateral boundary area with respect to the area of the top and bottom boundaries. A minimum distance between top and bottom boundaries coincided with the longest cylinder placed in the simulation space. Ratios between the lateral dimensions and the vertical dimension ranged from  $\sim 5$  for the 10% simulations to  $\sim 2$  in the 30% simulations. This variation is acceptable, as the edge effect is more prominent at volume fractions near the percolation threshold.

Immediately below follows a brief discussion of the simulation’s solution to two key problems: how to determine intersections between cylinders, and how to relate the resulting percolating clusters to the conductivity of the composite.

#### Intersection: Excluded Volume

Balberg, Anderson, Alexander and Wagner addressed the problem of determining the intersection of capped cylinders in three-dimensional space by defining the excluded volume.<sup>37</sup> The excluded volume is the volume around a given particle that the center of another particle (to be placed) cannot enter into without causing an intersection. (See Figure 4.1.) It is defined pair-wise: it depends on the angle between the two particles. The simulation therefore tests each particle located in the immediate vicinity of a particle to be placed to discover whether it falls within any of the previously placed particles’ excluded volumes.

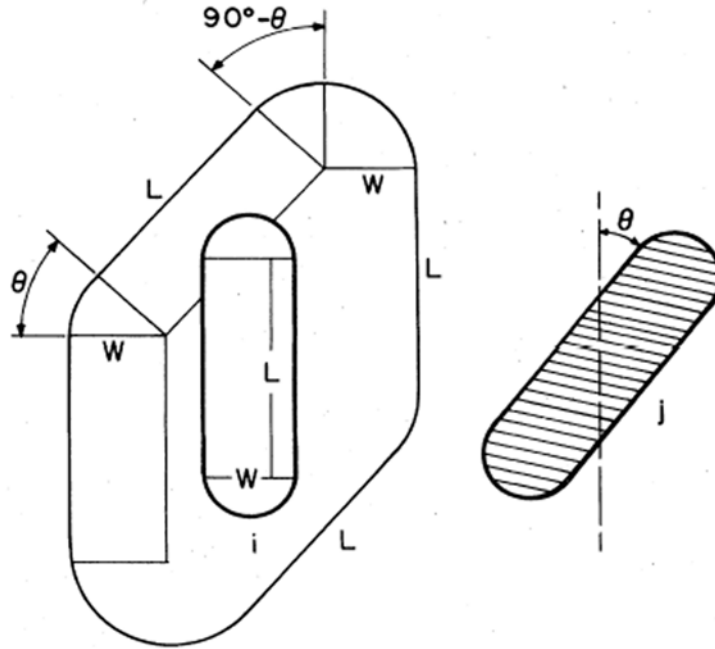


Figure 4.1. Depiction of the excluded volume surrounding a cylinder already placed in the matrix (left) with respect to a particle to be placed (shaded, right).  $\theta$  is the angle between the particles. The figure shows the concept in two dimensions, although its extension to three dimensions is straightforward. From Balberg, *et al.*; used by permission.<sup>37</sup>

A method of determining particle intersection leads to the question of allowable overlap. Clearly, in a mathematical sense, there must be some allowed overlap for percolation to be possible. This leads to either a “soft shell” model, wherein a particle falling into a shell around the outside of the excluded volume is allowed, or a “contact distance,” wherein a contact between two particles is assumed if the two particles are within a certain distance from one another. Deciding which of these two models to use depends on the physics of the problem. In the case of whisker composites probed with an A/C signal, the contact distance model (hereafter referred to as “shorting distance”) is perhaps more appropriate. This is because of the nature of the A/C signal. Although the matrix phase in this case is much more resistive than that of the filler (measurements

below give  $\sim 10^{12}$  ohm-cm for  $\text{Al}_2\text{O}_3$  while the literature reports anywhere from 0.2 – 4 ohm-cm for undoped SiC<sup>45</sup>), an A/C signal will still propagate between whiskers that are not otherwise in contact if the distance is short enough, and the frequency is high enough. This is because the matrix phase between the whiskers acts as a resistor and capacitor in parallel, or perhaps a series of such circuit representations if treating the interphase boundaries separately. The resistance of the circuit decreases with the distance between particles even as the capacitance increases. (The dielectric constant of  $\text{Al}_2\text{O}_3$  in the frequency range used in this study was measured at about 10). The real impedance for such an RC-circuit is given by

$$Z' = \frac{Rs}{1 + j\omega^2 Rs^2 C^2} \quad (4.1)$$

where  $\omega$  is the angular frequency,  $Rs$  is the resistance and  $C$  the capacitance. Equation 4.1 illustrates that the higher the frequency and the shorter the distance between whiskers, the more likely it is that the signal will be able to “short” through the matrix to the other whisker. The shorting distance can be determined by finding the value for which the simulated and experimental percolation thresholds coincide.

It is worth mentioning that the incorporation of a shorting distance into the model does not lead directly to an explicit consideration of matrix transport in the model. The assumption at work here is that the connectivity of the whiskers will dominate the impedance response at the expense of any response due to the nature of the contacts between particles. This assumption obviates the need to ask the simulation to measure any matrix-oriented effect: it is enough to know that two particles in close proximity to each

other are in electrical contact due to the signal shorting between the particles. This assumption will be tested, of course, by comparing the results of the simulation to experimental measurements, and is considered a key result of the following exercise.

### Determining Effective Resistivity

The simulation determines the percolating cluster in a straightforward fashion: by keeping track of particle intersections and determining which particles belong to clusters that reach the top and the bottom of the simulation space. (Figure 4.2 shows a graphical rendering of a percolating cluster.) Given the resistivity of the whiskers and the simulation output of the percolating cluster, the effective conductivity follows from applying Kirchoff's current law. The whiskers are imagined to be perfect cylindrical conductors whose resistance follows the simple geometric resistance rule

$$R_s = \rho_w \frac{l}{ar} \quad (4.2)$$

where  $\rho_w$  is the resistivity of the whisker,  $l$  is the distance between current carrying contacts, and  $ar$  is the area of the whisker. Kirchoff's current law states that the sum of the current entering any current carrying node is zero. In such a manner, a set of equations arise for the node voltages. Displayed in matrix form:

$$\underline{\underline{M}}_K \underline{V} = \underline{V}_A \quad (4.3)$$

where  $M_K$  is a matrix of resistances,  $V$  is the vector of node voltages to be solved for, and  $V_A$  is the vector of applied voltage, determined arbitrarily (usually one volt) which enters into those node equations that intersect the high-voltage surface. Solving equation 4.3

leads to a list of node voltages, from which the effective conductivity of the composite follows. Taking the dimensions of the simulation space into account, the equations can be reduced to

$$\sigma = K\sigma_w \quad (4.4)$$

where  $\sigma$  is the effective conductivity of the composite,  $\sigma_w$  is the conductivity of the whiskers (that will take both bulk and interfacial processes into account) and  $K$  is the connectivity factor, determined from the simulation through application of Kirchhoff's current law.

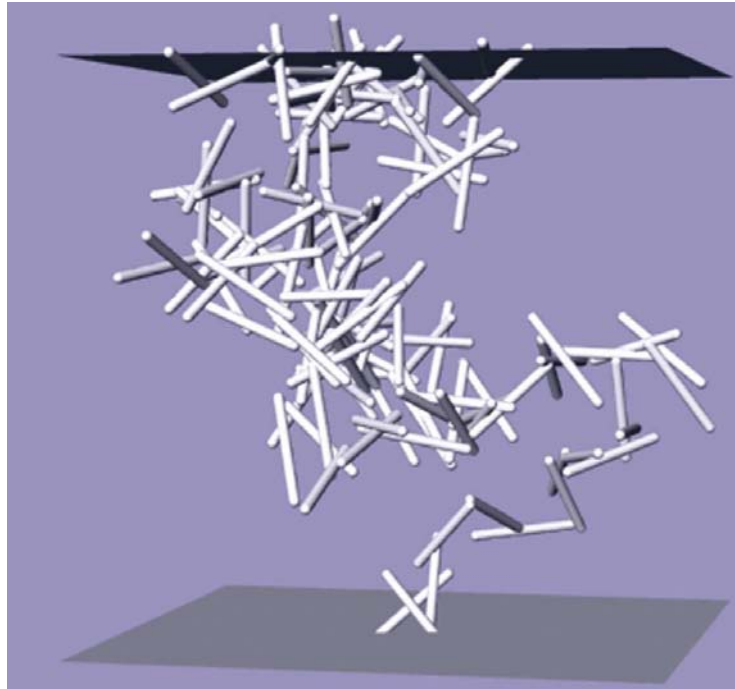


Figure 4.2. Graphical rendering of a percolating cluster from the Monte Carlo simulation.

## Experimental

Samples of hot pressed  $\text{Al}_2\text{O}_3\text{-SiC}_w$  composites were obtained from Advanced Composite Materials Corporation of Greer, SC. Each sample pellet was cylindrical in shape, approximately 8-8.5 mm thick and 28-29 mm in diameter. The hot pressing direction in each was parallel to the axis of the cylindrical pellet, imparting to each sample an axis of symmetry (see Figure 3.1). There were several different batches fabricated, and processing conditions possibly varied somewhat between the batches. Results arising from samples of different batches will be accounted for in the presentation of results below.

Pellets of various volume fractions of  $\text{SiC}_w$  (from 0 to 30%) were ground to 6  $\mu\text{m}$  smoothness, measured and painted on two circular faces with silver paint. After air drying, the impedance response was measured for each sample in a direction parallel to the hot pressing direction. A Solartron SI 1260 frequency response analyzer with a 1296 dielectric interface was used for samples of 9%  $\text{SiC}_w$  and less, and an HP 4192A was used for samples of 10%  $\text{SiC}_w$  and higher. The frequency range for each sweep was 0.1 Hz to 10 MHz. Additionally, some samples were sectioned to a thickness of 0.6-1.2 mm and a quarter of the electrode area and analyzed in an Agilent E4991A high-frequency analyzer, under a sweep of 1 MHz to 3.3 GHz.

After the impedance measurements were made, samples were sectioned both parallel and perpendicular to the hot-pressing direction. Montage images of the polished surfaces were taken in an SEM, and a stereological unfolding procedure was used to measure the trivariate length-radius-orientation distribution of whiskers in the composites. Figure 4.3

shows representative SEM images from 20 and 30% samples at low magnification. Figure 4.3 demonstrates the degree of whisker clumping in the composites, which the simulation does not account for.

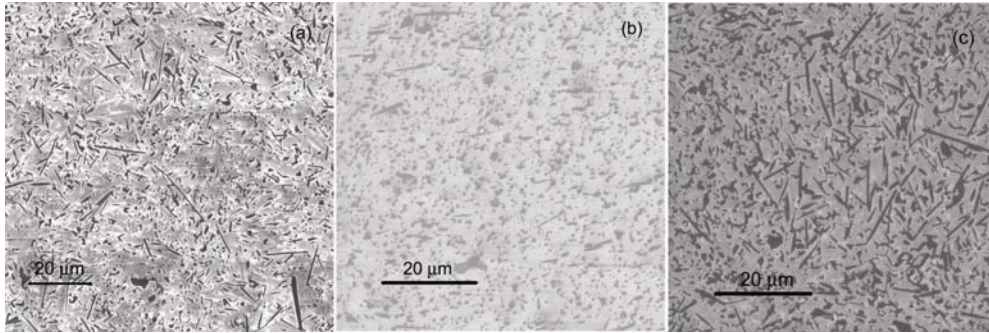


Figure 4.3. SEM images of Al<sub>2</sub>O<sub>3</sub>-SiC<sub>w</sub> composites used in this study. The micrographs show (a) a 20% SiC<sub>w</sub> sample taken from the plane perpendicular to the hot pressing direction (b) a 20% SiC<sub>w</sub> sample taken from the plane parallel to the hot pressing direction and (c) a 30% SiC<sub>w</sub> sample taken from the plane perpendicular to the hot pressing direction.

The details of the unfolding procedure appear in Chapters 2 and 3. Measurements taken on the 20% sample analyzed in Chapter 3 were taken as representative of the length-radius-orientation distribution for all 10, 20 and 30% samples of the same processing batch. While this last assumption will impart some error to the simulation output relative to experiment, the magnitude of the error will be less than the error due to the assumption of a perfect dispersion of whiskers, as shown below.

## Results

Figures 4.4 and 5 show representative impedance spectra for the composites. The spectra of Figure 4.4 are taken from the samples of 10% SiC<sub>w</sub> and above, while those of Figure 4.5 are taken from 8 and 9% SiC<sub>w</sub> samples. (Note the different scales in both

figures.) The equivalent circuit used to fit the spectra appears in Figure 4.6. These spectra clearly show that the critical behavior related to percolation of SiC<sub>w</sub> whiskers occurs in the range of 8-10% SiC<sub>w</sub>.

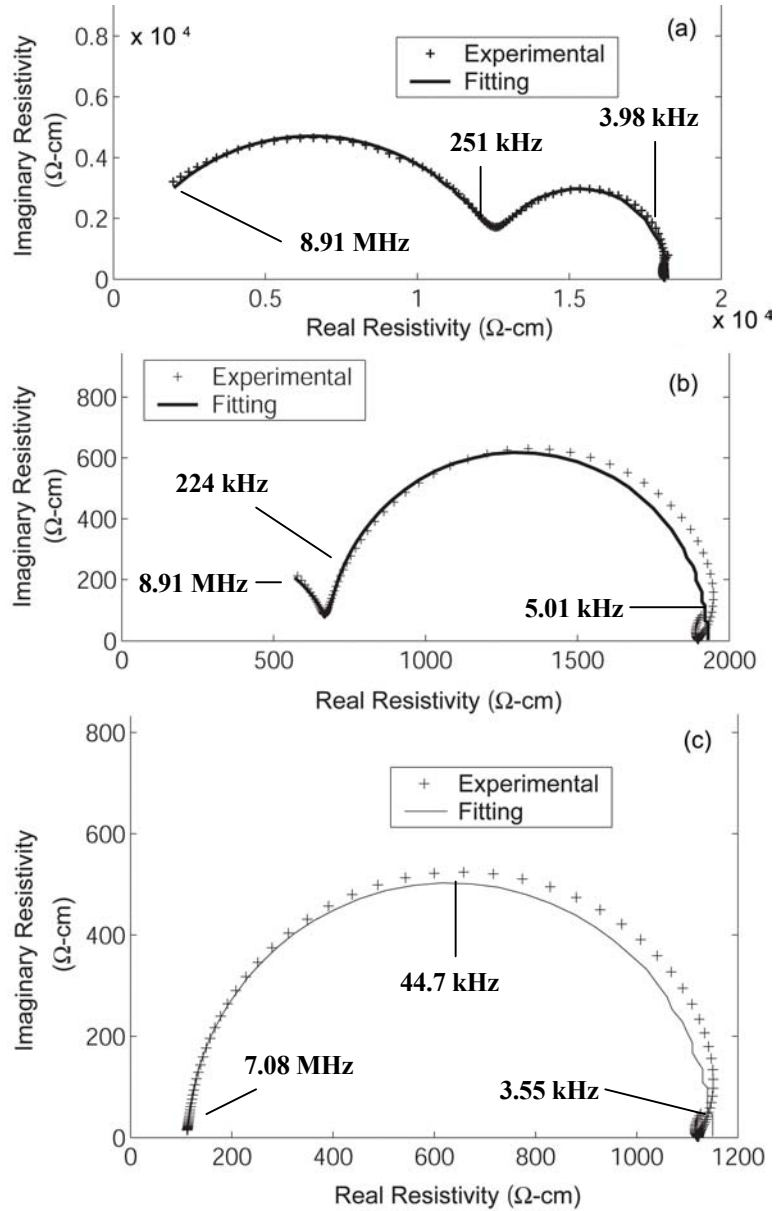


Figure 4.4. Fitted impedance spectra for (a) 10% (b) 20% and (c) 30% samples of SiC<sub>w</sub>-Al<sub>2</sub>O<sub>3</sub> composite. The high frequency semicircle occurs at frequencies too high to be shown in part (c), but it still exists.

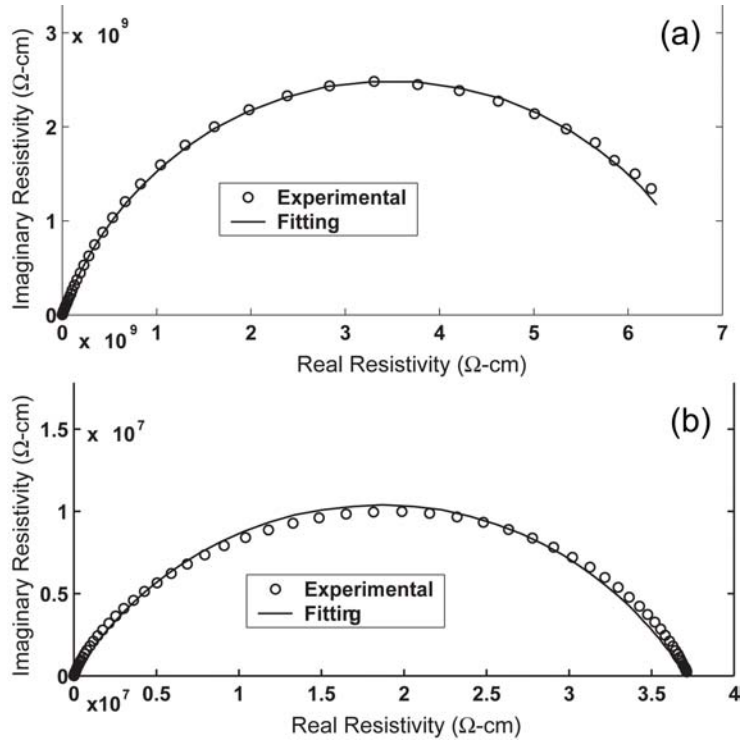


Figure 4.5. Fitted spectra for (a) 8 and (b) 9% SiC<sub>w</sub> samples.

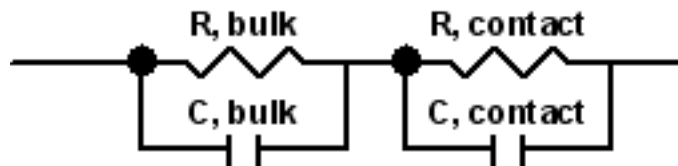


Figure 4.6. The equivalent circuit used to fit the spectra in Figures 4.4 and 5. In Figure 4.5, the response from the electrode is completely overtaken by the large resistivity of the bulk.

The impedance spectra display two capacitive features for volume fractions of 10% and above, but only a single capacitive feature for volume fractions below 10%. The perfect semi-circular shape of the low-frequency semi-circle, along with the calculated capacitance values (on the order of  $2-4 \times 10^{-8}$  F) indicate a macro-sized space charge capacitance with a smooth boundary. In addition, subsequent tests of samples with

different thicknesses and electrode areas revealed that the size of the low-frequency feature in the complex impedance plane (or the impedance of the process) does not depend on the thickness of the sample, but only depends on the size of the electrode area. Figure 4.7 illustrates this, showing size-normalized response from 10% volume fraction samples of different thicknesses and contact areas. The high-frequency semicircle changes little from sample to sample, while the low-frequency semicircle grows larger as the contact area becomes smaller. The blocking nature of the contact between Ag and SiC is almost certainly the cause of this prominent electrode impedance.<sup>46</sup> Volume fractions below 10% enter the critical range, where the non-electrode impedance overwhelms the electrode impedance, and only a single feature is visible in the spectra.

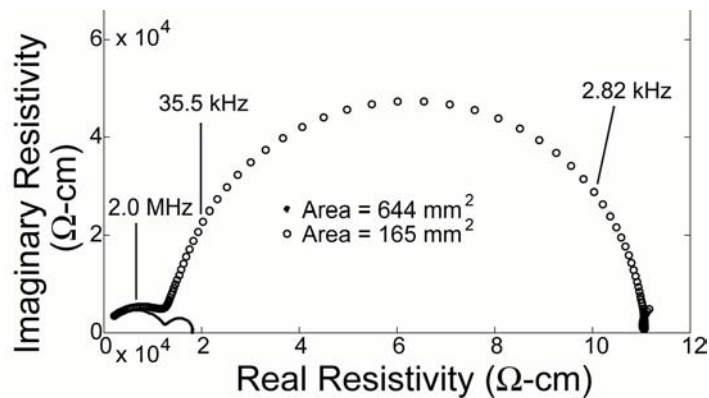


Figure 4.7. Complex resistivity (size-normalized) response from samples of 10% SiC<sub>w</sub>, but different thicknesses and contact areas.

The high-frequency process must therefore include contributions to the impedance from all of the charge-transport processes in the material – both bulk and interfacial. Its shape is that of a depressed semi-circle, indicating that it acts over a non-uniform geometry. Whether this feature represents a single type of resistive process (bulk or interfacial) dominating a much smaller one or a combination of two or more processes of

similar size and relaxation frequency is unclear. However, it is clear that this high-frequency feature represents the impedance response of the percolating whiskers, and therefore its intercept in the resistivity spectrum is the resistivity parameter of interest for this investigation.

It now becomes possible to use the resistivity data gathered from the impedance spectra to assign a value to the shorting distance parameter. Figure 4.8 shows a curve of conductivity versus volume fraction on a semi-log scale for all specimens measured. This curve clearly shows that the critical behavior occurs in the 8-10% range. The shapes of the various data points indicate which measurements belong to the same processing batch. This qualitatively shows the variations that different processing conditions create in the materials' conductivity, especially near the percolation threshold.

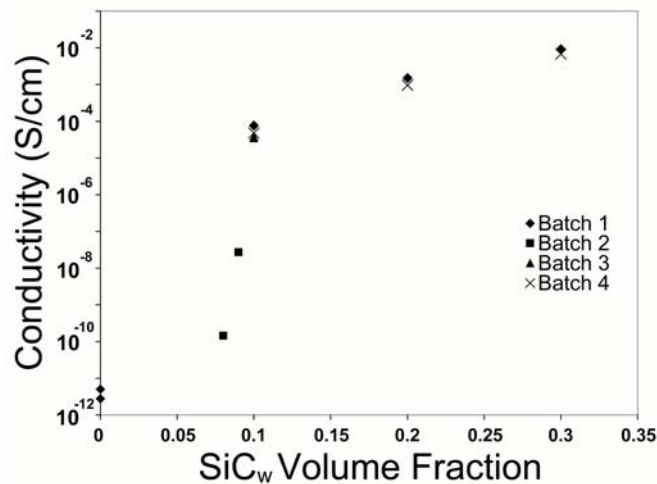


Figure 4.8. Bulk composite conductivity plotted versus volume fraction for all measured specimens, from all batches.

Figure 4.9 shows results from the simulation, in terms of percolating volume fraction versus total filler volume fraction for various values of the shorting distance, also plotted

on a semi-log scale. The percolating volume fraction is simply the total volume of whiskers determined to be active participants in percolating clusters divided by the total simulation volume. This parameter does not consider the effect of parallel versus series conduction, but it should clearly reveal the existence of critical behavior from a percolation standpoint. It is clear from the figure that the shorting distance for which the critical volume fraction is 9% is 0.15  $\mu\text{m}$ .

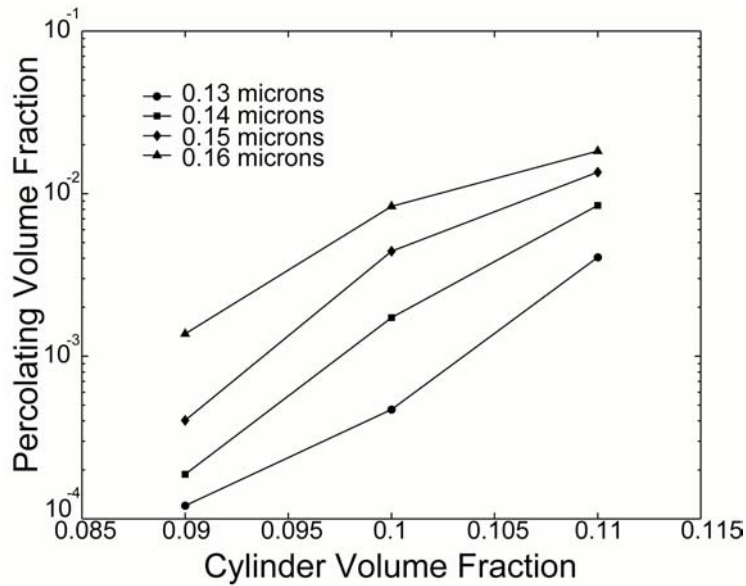


Figure 4.9. Simulated percolating volume fraction versus volume fraction for different values of the shorting distance.

To investigate the plausibility of a shorting distance of 0.15  $\mu\text{m}$ , an impedance simulation was performed on an RC-circuit designed to model the matrix phase between two whiskers that are 0.15  $\mu\text{m}$  (150 nm) apart. The capacitance and resistance of the circuit were calculated using the low-frequency relative permittivity ( $\sim 10$ ) and resistivity ( $\sim 10^{12}$   $\Omega\text{-cm}$ ) of  $\text{Al}_2\text{O}_3$  (which were measured over the frequency ranges used in this study), a conductor / capacitor thickness of 0.15  $\mu\text{m}$  and an area equal to the projected overlap between two  $\text{SiC}_w$  whiskers from the smallest diameter class oriented at an angle

of 45°. Figure 4.10 shows the results of the simulation as real impedance versus frequency. Note that in the frequency range of interest (approximately  $1 \times 10^5 - 1 \times 10^8$  Hz) the real impedance of the circuit drops below 10,000 ohms, reaching down into the single digits.

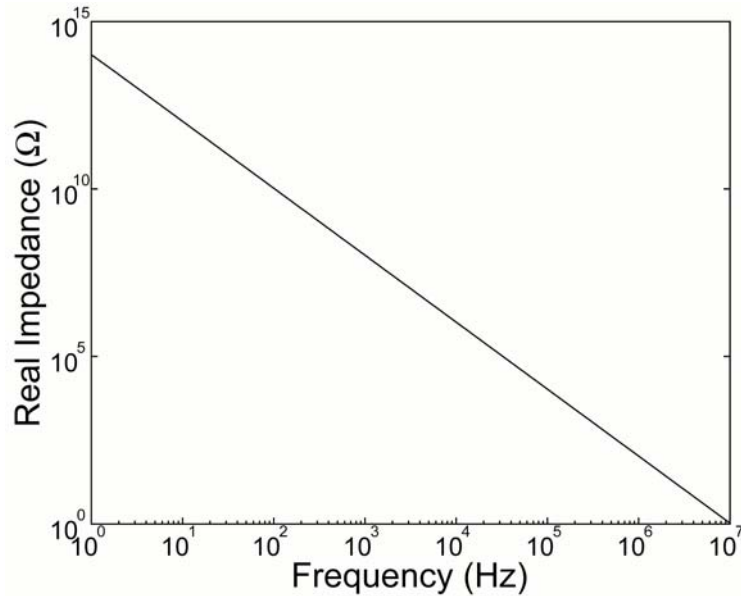


Figure 4.10. Bode plot (real impedance versus frequency) for a  $\text{SiC}_w\text{-Al}_2\text{O}_3\text{-SiC}_w$  interface, simulated as a parallel RC equivalent circuit.

Figure 4.11 shows the results of the simulation – expressed in terms of the parameter  $K$  in equation 4.4 – plotted against the experimentally determined conductivities. The analysis here is limited to the hyper-critical whisker volume fractions of the same batch (corresponding to the diamond points in Figure 4.8), and the simulation results for  $K$  are normalized to the average measured conductivity at 10% volume fraction. The shorting distance used in the simulation is the previously determined  $0.15 \mu\text{m}$ . Clearly, the trends in Figure 4.11 diverge significantly, with the simulation values rising much faster than the

measured conductivities at first, then leveling off to a slope more in line with (and actually slightly less steep than) the experimental curve at higher volume fractions.

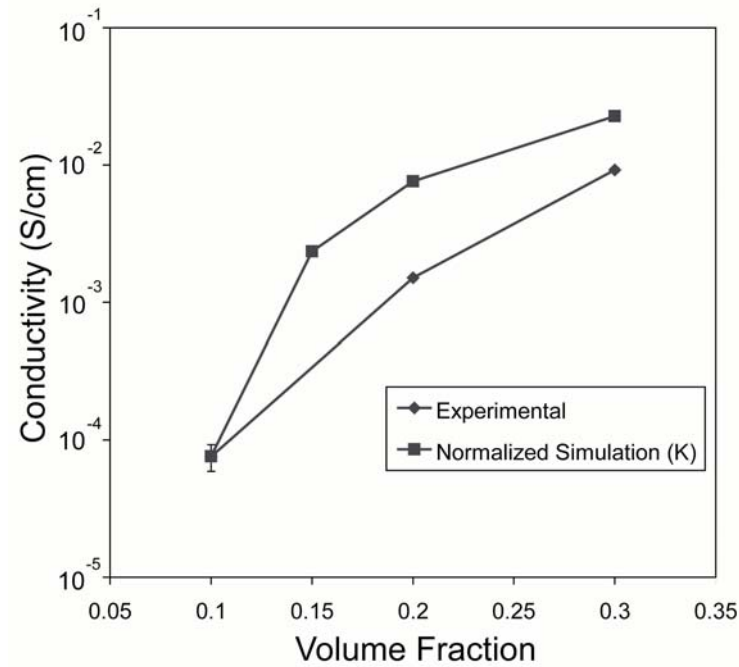


Figure 4.11. Conductivities from a single batch plotted versus the connectivity parameter  $K$ , as determined through the simulation. The  $K$  values are normalized such that the value at 10% volume fraction overlaps with the average experimental value.

## Discussion

### Dispersion Quality

The results shown in Figure 4.11 are not unexpected, given the assumptions of the simulation. The increased slope of the simulated curve relative to the experimental is due to the assumption of a perfect dispersion in the simulation. As seen in Figure 4.3, and reported in the literature for the  $\text{Al}_2\text{O}_3\text{-SiC}_w$  composite,<sup>3</sup> the real microstructure does not adhere to this idealization. Clumping of whiskers leads to fewer conducting paths through the material and therefore lower sample conductivity. If further developed, the difference

between the simulation and the experiment could form the basis for a non-destructive technique for the measurement of whisker dispersion quality in these materials.

### Interfacial Phenomena

The behavior shown in Figure 4.11 between 10 and 20% volume fraction seems to validate the approach used in the simulation with respect to the assumption of a constant shorting distance, as connectivity issues clearly dominate the response. However, the behavior between 20 and 30% raises questions about the assumption of a constant shorting distance. In this region the slopes of the two curves become almost identical, with the experimental curve actually rising somewhat more steeply than the simulated curve. Since the clumping of whiskers is still a significant factor at these volume fractions, another process other than the multiplication of conducting pathways must be responsible for this shift. A likely explanation is the fact that the simulation does not consider the possibility that the interfacial resistance between whiskers may change with the volume fraction.

Figure 4.12 shows a schematic of the two types of conducting paths between whiskers: direct  $\text{SiC}_w\text{-SiC}_w$  contacts and  $\text{SiC}_w\text{-Al}_2\text{O}_3\text{-SiC}_w$  contacts. As the volume fraction increases, the average nearest-neighbor distance between whiskers decreases and one expects the relative number of the latter type to gradually decrease with respect to the former. It is possible that the direct SiC-SiC interfaces are less resistive, leading to an accelerated increase in conductivity over that gained through the multiplication of conducting pathways alone. Future modeling efforts should include explicit consideration of the two different types of whisker interfaces.

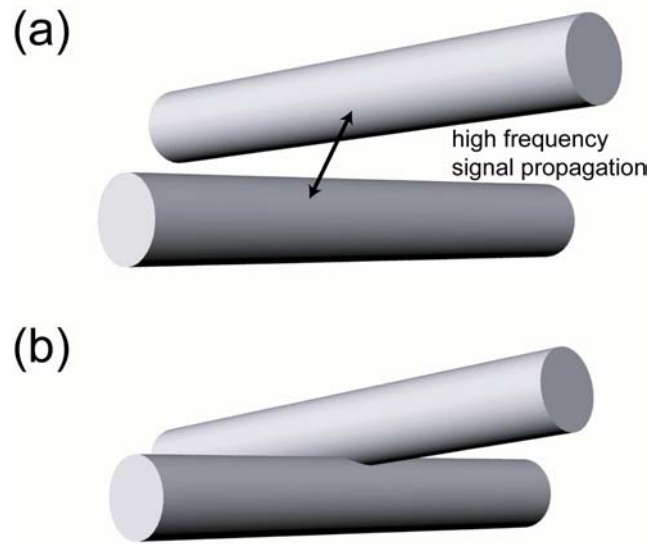


Figure 4.12. Schematic of the two different types of possible whisker interfaces, (a)  $\text{SiC}_w\text{-Al}_2\text{O}_3\text{-SiC}_w$  and (b)  $\text{SiC}_w\text{-SiC}_w$ .

## Conclusion

A Monte Carlo simulation was applied to the case of whisker connectivity in  $\text{Al}_2\text{O}_3\text{-SiC}_w$  composites. The results indicate a possible method for characterizing the dispersion quality of whiskers in these materials, by comparing impedance measurements to simulated predictions for perfectly dispersed microstructures. While the multiplication of conducting pathways was found to dominate the response near the percolation threshold, the interfacial resistance takes on increasing importance as volume fractions increase. Future non-destructive characterization methods must take both factors into account.

## CHAPTER 5

### CONCLUSION AND RECOMMENDATIONS

The problem of creating a non-destructive method of characterizing the dispersion quality of silicon carbide whisker-alumina and other anisotropic ceramic matrix composites through impedance techniques was substantially addressed – although not solved – in this work. Several things were established. First, it was established that the percolation behavior of the composite dominates the electrical response. This leads directly to the need for characterization of the size and orientation of whiskers, since the percolation of the whiskers depends on these factors. A destructive (stereological) method of making these measurements was derived and demonstrated. Finally, comparing a microstructural simulation to the measured spectra revealed that the type and quality of the interfaces between the whiskers play a significant role in the impedance response.

The comparisons between experimental spectra and the simulation revealed that impedance techniques do offer substantial information on the dispersion quality. While this correspondence was not quantified, future efforts and a more detailed model could achieve that goal. However, the quantification of dispersion quality for a given size and orientation distribution of whiskers may not be the ultimate end. It would be very interesting to discover through a future study – perhaps focused on the dielectric functions

measurable through impedance techniques but not examined here – whether the size and orientation of the whiskers itself may be non-destructively measurable, in addition to the dispersion quality. These questions are left to future investigations.

## REFERENCES

1. H.-S. Ahn and D.-J. Choi, "Fabrication of Silicon Carbide Whiskers and Whisker-Containing Composite Coatings Without Using a Metallic Catalyst," *Surf. Coat. Tech.*, **154**, 276-81 (2002).
2. F. Ye, Y. Zhou, C. Lei, J. M. Yang and L. T. Zhang, "Microstructure and Mechanical Properties of Barium Aluminosilicate Glass-Ceramic Matrix Composites Reinforced with SiC Whiskers," *J. Mater. Sci.*, **36**, 2575-80 (2001).
3. Y. Fu, Y. W. Gu and H. Du, "SiC Whisker Toughened Al<sub>2</sub>O<sub>3</sub>-(Ti, W)C Ceramic Matrix Composites," *Scripta Mater.*, **44**, 111-16 (2001).
4. M. E. Calhoun and P. R. Mouton, "Length Measurement: New Developments in Neurostereology and 3D Imagery," *J. Chem. Neu.*, **21**, 257-65 (2001).
5. R. L. Fullman, "Measurement of Particle Sizes in Opaque Bodies," *Trans. AIME*, **197**, 447-52 (1953).
6. R. T. DeHoff and F. N. Rhines, "Determination of Number of Particles per Unit Volume from Measurements Made on Random Plane Sections: The General Cylinder and the Ellipsoid," *Trans. Metall. Soc. AIME*, **221**, 975-82 (1961).
7. M. D. Thouless, B. J. Dagiiesh and A. G. Evans, "Determining the Shape of Cylindrical Second Phases by Two-Dimensional Sectioning," *Mater. Sci. Eng. A*, **102**, 57-68 (1988).
8. E. E. Underwood, *Quantitative Stereology*, Reading, MA: Addison Wesley, 1970.
9. A. M. Gokhale, "Estimation of Bivariate Size and Orientation Distribution of Microcracks," *Acta Mater.*, **44** [2] 475-85 (1996).
10. L.-M. Cruz Orive, "Particle Size-Shape Distributions: The General Spheroid Problem I. Mathematical Model," *J. Microsc.*, **107** [3] 235-53 (1976).
11. L.-M. Cruz Orive, "Particle Size-Shape Distributions: The General Spheroid Problem II. Stochastic Model and Practical Guide," *J. Microsc.*, **112** [2] 153-67 (1978).

12. V. Beneš, M. Jiruše and M. Slámová, "Stereological Unfolding of the Trivariate Size-Shape-Orientation of Spherical Particles with Application," *Acta Mater.*, **45** [3] 1105-13 (1997).
13. J. J. Saastamoinen, A. Tourunen, J. Hämäläinen, T. Hyppänen, M. Loschkin and A. Kettunen, "Analytical Solutions for Steady and Unsteady State Particle Size Distributions in FBC and CFBC Boilers for Non-Breaking Char Particles," *Combust. Flame*, **132**, 395-405 (2003).
14. B.J. McCoy, "A New Population Balance Model for Crystal Size Distributions: Reversible, Size-Dependent Growth and Dissolution," *J. Colloid Interface Sci.*, **240**, 139-49 (2001).
15. S. Lieberman, D. S. Mebane, A. M. Gokhale and R. A. Gerhardt, "First Application of a Novel Stereological Length-Radius Unfolding Procedure to Determine the Three Dimensional Bivariate Size and Shape Distribution of TiB Whiskers in Ti-6Al-4V-2.9B," accepted for publication in EPD Congress 2005, Edited by M. E. Schlesinger, The Minerals, Metals and Materials Society, 2005.
16. Y. Shi, X. Huang and D. Yan, "Synergistic Strengthening and Toughening of Zircon Ceramics by the Additions of SiC Whisker and 3Y-TZP Simultaneously," *J. Eur. Ceram. Soc.*, **17**, 1003-10 (1997).
17. A. Mukherjee and H. S. Rao, "FE-Modeling of the Toughening Mechanism in Whisker Reinforced Ceramic-Matrix-Composites," *Comput. Mater. Sci.*, **4**, 249-62 (1995).
18. P. F. Becher, "Microstructural Design of Toughened Ceramics," *J. Am. Ceram. Soc.*, **74** [2] 255-69 (1991).
19. K. T. Faber and A. G. Evans, "Crack Deflection Processes – I. Theory," *Acta Metall.*, **31** [4] 565-76 (1983).
20. P. F. Becher and G. C. Wei, "Toughening Behavior in SiC-Whisker-Reinforced Alumina," *J. Am. Ceram. Soc.*, **67** [12] C267-C269 (1984).
21. A. A. Zisman, "Rotation of Hard Particles in Deformed Matrix: 3D Simulation and Stereological Reconstruction from 2D Data," in Risø International Symposium on Materials Science Proceedings, vol. 25, Edited by C. Gundlach *et al.*, Risø National Laboratory, Roskilde, Denmark, 2004.

22. S. Toll and P.-O. Andersson, "Microstructural Characterization of Injection Moulded Composites Using Image Analysis," *Composites*, **22** [4] 298-306 (1991).
23. H.-J. Park, H.-E. Kim and D.-Y. Kim, "Evaluation of Whisker Alignment in Axisymmetric SiC<sub>w</sub>-Reinforced Al<sub>2</sub>O<sub>3</sub> Composite Materials," *J. Am. Ceram. Soc.*, **77** [11] 2828-32 (1994).
24. M. S. Sandlin, C. R. Peterson and K. J. Bowman, "Texture Measurement on Materials Containing Platelets Using Stereology," *J. Am. Ceram. Soc.*, **77** [8] 2127-31 (1994).
25. M. S. Sandlin, F. Lee and K. J. Bowman, "Simple Geometric Model for Assessing Whisker Orientation in Axisymmetric SiC-Whisker-Reinforced Composites," *J. Am. Ceram. Soc.*, **75** [6] 1522-28 (1992).
26. R. S. Bay and C. L. Tucker III, "Stereological Measurement and Error Estimates for Three-Dimensional Fiber Orientation," *Polym. Eng. Sci.* **32** [4] 240-53 (1992).
27. D. van Hille, S. Bengtsson and R. Warren, "Quantitative Metallographic Study of Fibre Morphology in a Short Alumina Fibre Reinforced Aluminum Alloy Matrix," *Compos. Sci. Technol.*, **35**, 195-206 (1989).
28. X. Cheng, A. M. Sastry and B. E. Layton, "Transport in Stochastic Fibrous Networks," *J. Eng. Mater. Technol.*, **123**, 12-19 (2001).
29. R. A. Gerhardt and R. Ruh, "Volume Fraction and Whisker Orientation Dependence of the Electrical Properties of SiC-Whisker-Reinforced Mullite Composites," *J. Am. Ceram. Soc.*, **84** [10] 2328-34 (2001).
30. S. Iio, M. Watanabe, M. Matsubara and Y. Matsuo, "Mechanical Properties of Alumina/Silicon Carbide Whisker Composites," *J. Am. Ceram. Soc.*, **72** [10] 1880-84 (1989).
31. R. A. Gerhardt, "Electrically Based Non-Destructive Microstructural Characterization of All Classes of Materials," pp. 93-104 in *Materials Research Society Symposium Proceedings*, Vol. 591, *Nondestructive Methods for Materials Characterization* Edited by G. Y. Baaklini, et al. Materials Research Society, Warrendale, Pa., 2000.

32. R. A. Gerhardt, J. Runyan, C. Sana, D. S. McLachlan and R. Ruh, "Electrical Properties of Boron Nitride Matrix Composites III, Observations near the Percolation Threshold in BN-B<sub>4</sub>C Composites," *J. Am. Ceram. Soc.*, **84** [10] 2335-42 (2001).
33. X. Wang and P. Xiao, "Nondestructive Characterisation of Alumina / Silicon Carbide Nanocomposites Using Impedance Spectroscopy," *J. Eur. Ceram. Soc.*, **20**, 2591-99 (2000).
34. A. D. Hixson, L. Y. Woo, M. A. Campo, T. O. Mason and E. J. Garboczi, "Intrinsic Conductivity of Short Conductive Fibers in Composites by Impedance Spectroscopy," *J. Electroceram.*, **7**, 189-95 (2001).
35. T. O. Mason, M. A. Campo, A. D. Hixson and L. Y. Woo, "Impedance Spectroscopy of Fiber-Reinforced Cement Composites," *Cem. Concr. Compos.*, **24**, 457-65 (2002).
36. E. A. Holm and M. J. Cima, "Two-Dimensional Whisker Percolation in Ceramic Matrix-Ceramic Whisker Composites," *J. Am. Ceram. Soc.*, **72** [2] 303-5 (1989).
37. I. Balberg, C. H. Anderson, S. Alexander and N. Wagner, "Excluded Volume and its Relation to the Onset of Percolation," *Phys. Rev. B*, **30** 3933-43 (1984).
38. X. Cheng, A. M. Sastry and B. E. Layton, "Transport in Stochastic Fibrous Networks," *J. Eng. Mater. Technol.*, **123**, 12-19 (2001).
39. A. L. R. Bug, S. A. Saffran and I. Webman, "Continuum Percolation of Rods," *Phys. Rev. Lett.*, **54** [13] 1412-15 (1985).
40. A. Celzard, E. McRae, C. Deleuze, M. Dufort, G. Furdin and J. F. Marêché, "Critical Concentration in Percolating Systems Containing a High-Aspect-Ratio Filler," *Phys. Rev. B*, **53** [10] 6209-14 (1996).
41. J.-R. deDreuzy, P. Davy and O. Bour, "Percolation Parameter and Percolation-Threshold Estimates for Three-Dimensional Random Ellipses with Widely Scattered Distributions of Eccentricity and Size," *Phys. Rev. E*, **62** [5] 5948-52 (2000).
42. R. Landauer, "Electrical Conductivity in Inhomogeneous Media," pp. 2-43 in AIP Conference Proceedings, vol. 40, *Electrical Transport and Optical Properties of Inhomogeneous Media* Edited by J. C. Garland and D. B. Tanner. American Institute of Physics, New York, 1978.

43. D. S. McLachlan, M. Blaskiewicz and R. E. Newnham, "Electrical Resistivity of Composites," *J. Am. Ceram. Soc.*, **73** [8] 2187-203 (1990).
44. I. Balberg and N. Binenbaum, "Cluster Structure and Conductivity of Three-Dimensional Continuum Systems," *Phys. Rev. A*, **31** [2] 1222-5 (1985).
45. G. L. Harris, H. S. Henry and A. Jackson, "Carrier Mobilities and Concentrations in SiC," pp. 63-8 in *Properties of Silicon Carbide* Edited by G. L. Harris. INSPEC, London, 1995.
46. J. R. Waldrup, R. W. Grant, Y. C. Wang and R. F. Davis, "Metal Schottky Barrier Contacts to Alpha 6H-SiC," *J. Appl. Phys.*, **72** 4757-60 (1992).

## APPENDIX

### SIMULATION CODE

The following is the microstructural simulation code. It was developed in the C programming language. The simulation performs many tasks, including creating the percolation / simulation space described in Chapter 4, creating a sectioning plane through the simulation space in order to check stereological equations (Chapters 2 and 3), and producing output that is renderable in the Rhinoceros drafting software (Figure 4.2.). As not all of these capabilities are required at any given time, some may be inactivated through the // or /\* - \*/ commands. The code as it appears below is in the form used for creation of the simulation spaces discussed in Chapter 4.

The code consists of three main sections. First there is the main code that loads the input distribution, chooses the particles (along with the particles used in the periodic boundary condition) calls the intersection and percolation functions and produces the output. The main part of the code also executes the sectioning plane simulation when that part of the code is active. The intersection function takes the length, radius, orientation and position of two particles as input and returns a yes or no answer as to whether any intersection between particles is allowed. If it is allowed, the function also returns the point of intersection. Finally, the percolation function determines the identity of the particles and particle intersections that participate in the percolating cluster. This

information may be used to build a Kirchoff's matrix to estimate conductivity as described in Chapter 4.

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
#include "erf.h"
#include "Rand_fast_ulong.h"
#include "Rand_53bit.h"

#define pi 3.14159265

//float endpoints[5000][6];
float intsectpts[110000][3]; // these three are a matrix of intersection

long int intsectnos[110000][2]; // particle numbers, intersection points
int intind, mind;

float centers[45000][3];
float angles[45000][4];

double r; // cylinder radius working var
float radlen[45000][2]; // this is radii and length
double l, gam; // cylinder length working var
double sh2, ss2, cs2, ch2;
double radav, gamav, lenav, sinalphav;
double Ovlp, Vi, Vil, tunn;

double percvol, tunnvol;
float botpercpt[20000][3], toppercpt[20000][3];
long int intnos[110000][2], intnos1[110000][2];
long int particles[45000], particles1[45000];
int bottom[20000], finish1[20000], temp1[35000];
int botpercenos[20000], toppercnos[20000], killcount=0;

float angles0[35000][4], radlen0[35000][2]; //centers0[10000][3];
float angtmp0[35000][4], radtmp0[35000][2], maxgam;

int intest(int i);
void perc(int Nw, double Lo, double Loxy);
```

```

main()
{

// Set up the random number generator

int i, j;

cout << "do you have a SimID? 1=yes";
cin >> j;

long SimID; // the generator returns the same set for the same SimID.

if (j==1)
    cin >> SimID;
else
    SimID = (unsigned)time(NULL); // gets SimID from the computer's clock.

seedMT(SimID);

// Finished. Now random53() returns 53-digit random # from 0 to 1.

int iternum, iter;
cout << "\nhow many iterations per step?\n";
cin >> iternum;

double Vf; // volume fraction
cout << "\nwhat is the initial volume fraction?\n";
cin >> Vf;

long Nworig, Nw; // # of particles
cout << "\ntotal number of particles per iteration?\n";
cin >> Nworig;

// double Ovh; // overlap %
// cout << "\nwhat is the average allowable overlap?";
// cin >> Ovlp;

tunn = 0.15;
double tunnr;
tunnr = tunn/2;

double percvolfrac[20];
int intrec[200], numit, numitt=0;

```

```

cout << "\nhow many steps in total?";
cin >> numit;

FILE *resultstream;
resultstream = fopen("results.txt", "w");

double Vw, Vo, Lo, Sw, Loos[50], Loxy, Lfact; // whisker volume, model volume
double percvoltot=0, radii;

// This next sequence uploads the parameters of the distribution.
// Uses the quasi-normal distribution as calculated below; parameters appear in dispar.txt.

float Rmin, Rmax, Lmin, Lmax;
float muR, mL, sgR, sgL, sgpsi, corl;
double F, distR, distGam, distPsi, Ftest;
double cosdist, cosdistest, psi, phi;
// float angles0[25000][4], radlen0[25000][2]; //centers0[10000][3];
// float angtmp0[25000][4], radtmp0[25000][2], maxgam;
int sortind;

FILE *radstream;
radstream = fopen("distpar.txt", "r");

fscanf(radstream, "%f%f%f%f\n\n", &Rmin, &Rmax, &Lmin, &Lmax);
fscanf(radstream, "%f%f%f%f%f%f", &muR, &mL, &sgR, &sgL, &sgpsi, &corl);

fclose (radstream);

// Now we move to the iterations of the Monte Carlo simulation. First, we define
// a few variables and open a few files we'll need.

FILE *sectstream;
sectstream = fopen("sections.txt", "w");

FILE *ctrstream;
FILE *oristream;
FILE *radlestream;
ctrstream = fopen("centers.txt", "w");
oristream = fopen("angles.txt", "w");
radlestream = fopen("sizes.txt", "w");

```

```

// FILE *rhinostream;
// rhinostream = fopen("rhino.txt", "w");

FILE *intstream;
intstream = fopen("intpoints.txt", "w");

FILE *int2stream;
int2stream = fopen("intnumbers.txt", "w");

FILE *lostream;
lostream = fopen("cubedges.txt", "w");

FILE *sterstream;
sterstream = fopen("stereology.txt", "w");

// These are edge-effect variables
int k, seq, gemm;
double axvect[3], t, endpt[2][3], edgcent[7][3];

double vects; // this is something we use to determine the average angle between cyls.

// these are variables we use in the particle insertion loops.
int m, intindtemp;
double phisum, phiav;

// Stereology variables
int n, sectcnt=0, jstonce=0, sectnum=0, sectz=0, ist,iy,ix;
float xsect,ysect,zsect,tx,ty,tz,txr,tyr,tzr,majx,majy,alphx,alphy,alphz,tstxz,tstxy;

// Variables for the histogram
int brklp, rbin, gambin, psibin;
float Hist[4][100][40], binliners[3][101], MaxHist;

// Note: angles defined about the positive y-axis: plus/minus pi/2 in phi (xy-plane)
// and psi (yz-plane).

int conthist;
cout << "\nHistogram or Continuous? 1=hist, other=cont";
cin >> conthist;

if (conthist == 1) {

    cout << "\nSize of Histogram? Format: r L psi";
    cin >> rbin >> gambin >> psibin;
}

```

```

// cout << "\n" << rbin << " " << gambin << " " << psibin;

FILE *histstream;
histstream = fopen("histogram2_2.txt","r");

cout << "\nopened";

MaxHist=0;
for (m=0; m<rbin; m++) {
  for (j=0; j<gambin; j++) {
    for (k=0; k<psibin; k++) {
      fscanf(histstream, "%f ", &Hist[m][j][k]);

      if (m==0)
        MaxHist = MaxHist + Hist[m][j][k];
    }
  }
}

fclose(histstream);

FILE *binstream;
binstream = fopen("binliners2_2.txt","r");

for (m=0; m<rbin+1; m++)
  fscanf(binstream, "%f ", &binliners[0][m]);

for (m=0; m<gambin+1; m++)
  fscanf(binstream, "%f ", &binliners[1][m]);

for (m=0; m<psibin+1; m++)
  fscanf(binstream, "%f ", &binliners[2][m]);

fclose(binstream);
}

iter=0;
for (iter=0; iter<iternum; iter++) { // simulation iterations

  Nw = Nworig;

  if (conthist != 1){

    i=0;
    for (i=0; i<Nw; i++) { // This gets us a list of particles for this iteration

```

```

r = random53()*(Rmax - Rmin) + Rmin;
gam = random53()*(Lmax - Lmin) + Lmin;

cosdistest = -1; // chooses provisional psi randomly, which means
while (cosdistest <= 0) { // distributed as cos(psi).
    psi = pi*random53() - (pi/2);
    cosdist = cos(psi);
    cosdistest = cosdist - random53();
}

//sgPsi = 100000;

//gam = 37.233*exp(-3.70264*psi*psi); // just for the test for L-bar

distR = 0; //(r - muR)/sgR;
distGam = (log(gam/mL))/sgL; //0; //(gam - muGam)/sgGam;
distPsi = 0; //psi/sgPsi;

F = (mL/(gam*exp(0.5*sgL*sgL)))*exp(-0.5*(distGam*distGam));

// code will truncate here. If it is needed to create a continuous distribution,
// will need to add code that uses the above probability, F, to choose particles
// according to the continuous distribution in a similar manner as below.

} // end for loop
} // end if
else {

for (i=0; i<Nw; i++) {

r = random53()*(binliners[0][rbin] - binliners[0][0]) + binliners[0][0];
gam = random53()*(binliners[1][gambin] - binliners[1][0]) + binliners[1][0];

cosdistest = -1; // chooses provisional psi randomly, which means
while (cosdistest <= 0) { // distributed as cos(psi).
    psi = pi*random53() - (pi/2);
    cosdist = cos(psi);
    cosdistest = cosdist - random53();
}

brklp=0;
for (m=0; m<rbin; m++) {
    for (j=0; j<gambin; j++) {
        for (k=0; k<psibin; k++) {

```

```

        if (r < binliners[0][m+1] && gam < binliners[1][j+1] && psi < binliners[2][k+1])
    {
        brklp = 1;
        break;
    }
    }
    if (brklp == 1)
        break;
    }
    if (brklp == 1)
        break;
    }
}

```

```
F = Hist[m][j][k];
```

```
Ftest = F - random53();
```

```

if (Ftest >= 0) {
    radlen0[i][0] = r + tunnr;
    radlen0[i][1] = gam;
    angles0[i][2] = sin(psi);
    angles0[i][3] = cosdist;

```

```
// Choose random phi
```

```
phi = pi*random53() - (pi/2);
```

```

angles0[i][0] = sin(phi);
angles0[i][1] = sqrt(1 - angles0[i][0]*angles0[i][0]);

```

```

}
else {
    i--;
    continue;
}

```

```
} // end histogram definition loop (i)
```

```
}// end else: Histogram definition
```

```

// Next step is to sort the particles by size, so that the biggest ones get placed
// first.

```

```

// First, make a temporary set of distribution vectors.

for (i=0; i<Nw; i++) {
    radtmp0[i][0] = radlen0[i][0];
    radtmp0[i][1] = radlen0[i][1];
    angtmp0[i][0] = angles0[i][0];
    angtmp0[i][1] = angles0[i][1];
    angtmp0[i][2] = angles0[i][2];
    angtmp0[i][3] = angles0[i][3];
}

// This is the sorting step.

for (i=0; i<Nw; i++) {

    maxgam = 0;

    for (j=0; j<Nw; j++) {

        if (radtmp0[j][1] > maxgam) {

            maxgam = radtmp0[j][1];
            sortind = j;

        }

    }

//   cout << "\n" << maxgam << " " << sortind;

    radlen0[i][0] = radtmp0[sortind][0];
    radlen0[i][1] = radtmp0[sortind][1];
    angles0[i][0] = angtmp0[sortind][0];
    angles0[i][1] = angtmp0[sortind][1];
    angles0[i][2] = angtmp0[sortind][2];
    angles0[i][3] = angtmp0[sortind][3];

    radtmp0[sortind][1] = 0;

}

for (i=0; i<Nw; i++)
    fprintf(sterstream, "%f%f%f%f\n", radlen0[i][0], radlen0[i][1], acos(angles0[i][3]),
    acos(angles0[i][1]));

```

```
// Now determine the total whisker volume, thereby determining the sample volume.
```

```
i=0;
Vw = 0;
for (i=0; i<Nw; i++) {

    radii = radlen0[i][0] - tunnr;
    Vw = Vw + pi*(radii*radii)*radlen0[i][1] + (4.0/3.0)*pi*(radii*radii*radii);

}
```

```
// Ov = 1 - Ovl;
Vo = Vw/Vf;
Lo = 40;
Lfact = sqrt(Vo/(Lo*Lo*Lo));
Loos[iter] = Lfact;
Loxy = Lfact*Lo;
```

```
// Note: indent changes to avoid right creep, but all of this code is still in iter loop.
```

```
k=0;
seq=0;
```

```
// Randomly choose & place particles
```

```
long int trynum = 0, tries=0;
intind = 0;    // this is an index that will help keep track of intersections
i=0;
m=0;
phisum=0;
for (i=0; i<Nw; i++)
{
    // choose random center

    centers[i][0] = Loxy*random53();
    centers[i][1] = Loxy*random53();
    centers[i][2] = Lo*random53();

    // choose random phi

    phi = pi*random53() - pi/2;

    angles0[seq][0] = sin(phi);
```

```

angles0[seq][1] = cos(phi);

sh2 = angles0[seq][0];
ch2 = angles0[seq][1];
ss2 = angles0[seq][2];
cs2 = angles0[seq][3];

// if tries is more than a certain amount, choose random psi

if (tries == 70000)
    trynum++;

if (tries > 0 && tries%70000 == 0) {

    radlen0[seq][1] = radlen0[seq][1]/2;
}

// This is the edge effect code
// First, find out what the coordinates of particle endpoints are.

axvect[0] = sh2*cs2;
axvect[1] = ch2*cs2;
axvect[2] = ss2;

t = radlen0[seq][1]/2 + radlen0[seq][0];

j=0;
for (j=0; j<3; j++) {
    endpt[0][j] = axvect[j]*t + centers[i][j];
    endpt[1][j] = -axvect[j]*t + centers[i][j];
}

// Now, test those endpoints against the six planes that define the sim space
// We keep track of how many, and automatically define an opposite center in
// 'edgecent'

j=0;
m=0;
for (j=0; j<3; j++) {
    if (endpt[0][j] < 0 || endpt [1][j] < 0) {
        k=0;
        for (k=0; k<3; k++) {
            if (k == j && j < 2)

```

```

    edgcent[m][k] = centers[i][k] + Loxy;
else if (k == j && j == 2)
    edgcent[m][k] = centers[i][k] + Lo;
else
    edgcent[m][k] = centers[i][k];
}

m++;

}

if ((endpt[0][j] > Loxy || endpt [1][j] > Loxy) && j < 2) {
    k=0;
    for (k=0; k<3; k++) {
        if (k == j)
            edgcent[m][k] = centers[i][k] - Loxy;
        else
            edgcent[m][k] = centers[i][k];
    }

    m++;
}

if ((endpt[0][j] > Lo || endpt [1][j] > Lo) && j == 2) {
    k=0;
    for (k=0; k<3; k++) {
        if (k == j)
            edgcent[m][k] = centers[i][k] - Lo;
        else
            edgcent[m][k] = centers[i][k];
    }

    m++;
}

if ((endpt[0][j] < 0 && endpt[1][j] > Loxy) || (endpt[0][j] > Loxy && endpt [1][j] < 0)
&& j < 2) {
    m=0;
    break;
}

if ((endpt[0][j] < 0 && endpt[1][j] > Lo) || (endpt[0][j] > Lo && endpt [1][j] < 0) &&
j == 2) {
    m=0;
    break;
}

```

```

}

// Now we must add more opposites to edgcent if the particle intersects
// 2 or 3 boundary planes.

if (m > 3) {
    m = 0;
}

if (m == 3) {
    j=0;
    for (j=0; j<3; j++) {
        edgcent[3][j] = edgcent[0][j] + edgcent[1][j] - centers[i][j];
        edgcent[4][j] = edgcent[0][j] + edgcent[2][j] - centers[i][j];
        edgcent[5][j] = edgcent[1][j] + edgcent[2][j] - centers[i][j];
        edgcent[6][j] = edgcent[0][j] + edgcent[1][j] + edgcent[2][j] - 2*centers[i][j];
        m = 7;
    }
}

if (m == 2) {
    j=0;
    for (j=0; j<3; j++) {
        edgcent[2][j] = edgcent[0][j] + edgcent[1][j] - centers[i][j];
        m = 3;
    }
}

// Now, if we have extra particles, we need to insert them into the centers,

if (m > 0) {
    j=0;
    for (j=0; j<m; j++) {
        centers[i+j+1][0] = edgcent[j][0];
        centers[i+j+1][1] = edgcent[j][1];
        centers[i+j+1][2] = edgcent[j][2];
    }
}

```

```

    }
}

k=i;
for (k=i; k<i+m+1; k++) {
    angles[k][0] = angles0[seq][0];
    angles[k][1] = angles0[seq][1];
    angles[k][2] = angles0[seq][2];
    angles[k][3] = angles0[seq][3];

    radlen[k][0] = radlen0[seq][0];
    radlen[k][1] = radlen0[seq][1];
}

// test for intersection

intindtemp = intind;

if (i > 0)
{
    k=0;
    for (k=0; k<m+1; k++) {

        gemm = i+k;
        j = intest(gemm);

        if (j == 1)
            break;
    }
}
else
    j=0;

if (j == 1)
{
    tries++;
    intind = intindtemp;
    i = i - 1;
    continue;
}

```

```

// Okay, now, because of the continue statement above, we know that if we are at this
// spot we should place the particle (or particles, if we have a successful edge particle)

Nw = Nw + m;
/*
k=0;
for (k=0; k<m+1; k++) {

    endpoints[i+k][0] = (radlen[i+k][1]/2)*sh2*cs2 + centers[i+k][0];
    endpoints[i+k][1] = (radlen[i+k][1]/2)*ch2*cs2 + centers[i+k][1];
    endpoints[i+k][2] = (radlen[i+k][1]/2)*ss2 + centers[i+k][2];
    endpoints[i+k][3] = (-radlen[i+k][1]/2)*sh2*cs2 + centers[i+k][0];
    endpoints[i+k][4] = (-radlen[i+k][1]/2)*ch2*cs2 + centers[i+k][1];
    endpoints[i+k][5] = (-radlen[i+k][1]/2)*ss2 + centers[i+k][2];
}
*/
cout << "\n" << i << " of " << Nw << ", iteration " << iter << " step " << numitt;

// centers0[seq][0] = centers[i][0]; // records the centers of the original
// centers0[seq][1] = centers[i][1]; // particles.
// centers0[seq][2] = centers[i][2];

i = i + m;
tries=0;
seq++;

} // particles are placed (end i loop)

// Now, we have to create a stereological section. It would be good to get a number of
// sections for each cube. Let's make it 2. Note: code still set for cubic space.

/*
n=0;
for (n=0; n<1; n++) {

    xssect = Lo*random53();
    yssect = Lo*random53();
    zssect = Lo*random53();

    i=0;
    for (i=0; i<Nw; i++) {

        alphx = asin(angles[i][0]*angles[i][3]);

```

```

alphy = asin(angles[i][1]*angles[i][3]);

alphx = fabs(alphx);
alphy = fabs(alphy);
alphz = acos(angles[i][3]);

tx = fabs(xsect - centers[i][0])/(sin(alphx));
ty = fabs(ysect - centers[i][1])/(sin(alphy));
tz = fabs(zsect - centers[i][2])/(sin(alphz));

txr = radlen[i][1]/2 - tx;
tyr = radlen[i][1]/2 - ty;
tzt = radlen[i][1]/2 - tz;

tstxz = (xsect - centers[i][0])*angles[i][2]*fabs(angles[i][0]/angles[i][0] +
centers[i][2]);
tstxy = (xsect - centers[i][0])*angles[i][3]*angles[i][1] + centers[i][1];

if (tstxz > Lo || tstxz < 0 || tstxy > Lo || tstxy < 0)
    continue;

if (tzt >= 0)
    sectz++;

// first the x's

majx = 0;

if (txr >= 0) {
    if (alphx <= atan(radlen[i][0]/radlen[i][1])) {
        majx = radlen[i][1]/cos(alphx);
        fprintf(sectstream, "%f%f%f\n", majx, radlen[i][0], angles[i][3]);
        sectcnt++;
//      cout << "\ndte1 " << alphx << " " << radlen0[i][1] << " " << centers0[i][0] << " "
<< txr;
    }
    else if (alphx <= atan(2*radlen[i][0]/radlen[i][1])) {
        if (txr <= radlen[i][1] - radlen[i][0]/tan(alphx)) {
            majx = txr*cos(alphx) + radlen[i][0]/sin(alphx);
            fprintf(sectstream, "%f%f%f\n", majx, radlen[i][0], angles[i][3]);
//      cout << "\nste1 " << alphx << " " << radlen0[i][1] << " " << centers0[i][0] << " "
<< txr;
            sectcnt++;
        }
    }
    else {
        majx = radlen[i][1]/cos(alphx);

```

```

        fprintf(sectstream, "%f%f%f\n", majx, radlen[i][0], angles[i][3]);
//      cout << "\ndte2 " << alphx << " " << radlen0[i][1] << " " << centers0[i][0] << " "
<< txr;
        sectcnt++;
    }
}
else {
    if (txr <= 2*radlen[i][0]/tan(alphx)) {
        majx = txr*cos(alphx) + radlen[i][0]/sin(alphx);
        fprintf(sectstream, "%f%f%f\n", majx, radlen[i][0], angles[i][3]);
//      cout << "\nste2 " << alphx << " " << radlen0[i][1] << " " << centers0[i][0] << " "
txr;
        sectcnt++;
    }
    else {
        majx = 2*radlen[i][0]/sin(alphx);
        fprintf(sectstream, "%f%f%f\n", majx, radlen[i][0], angles[i][3]);
//      cout << "\nfe " << alphx << " " << radlen0[i][1] << " " << centers0[i][0] << " "
<< txr;
        sectcnt++;
    }
}
}
}
}
*/
//  if (i == Nworig-1)
//      cout << "\nxsect = " << xsect;
/*
    if (majx < 0)
        cout << "\n" << xsect << " " << centers[i][0] << " " << angles[i][0] << " " <<
angles[i][1] << " " << angles[i][2] << " " << angles[i][3] << " " << radlen[i][0] << " " <<
radlen[i][1];

    // now the y's

    if (alphy <= atan(radlen[i][0]/radlen[i][1]))
        iy = 1;
    else if (alphy <= atan(2*radlen[i][0]/radlen[i][1]))
        iy = 2;
    else
        iy = 3;

    majy = 0;

    if (tyr >= 0) {
        switch (iy) {

```

```

case 1:
    majy = radlen[i][1]*cos(alphy);
    fprintf(sectstream, "%f %f %f\n", majy, radlen[i][0], angles[i][3]);
    sectcnt++;
    break;
case 2:
    if (tyr <= radlen[i][0]*tan(alphy)) {
        majy = tyr*cos(alphy) + radlen[i][0]/sin(alphy);
        fprintf(sectstream, "%f %f %f\n", majy, radlen[i][0], angles[i][3]);
        sectcnt++;
    }
    else {
        majy = radlen[i][1]*cos(alphy);
        fprintf(sectstream, "%f %f %f\n", majy, radlen[i][0], angles[i][3]);
        sectcnt++;
    }
    break;
case 3:
    if (tyr <= radlen[i][0]*tan(alphy)) {
        majy = tyr*cos(alphy) + radlen[i][0]/sin(alphy);
        fprintf(sectstream, "%f %f %f\n", majy, radlen[i][0], angles[i][3]);
        sectcnt++;
    }
    else {
        majy = 2*radlen[i][0]/sin(alphy);
        fprintf(sectstream, "%f %f %f\n", majy, radlen[i][0], angles[i][3]);
        sectcnt++;
    }
}
}
}

```

```
*/
```

```
// now, we have to make sure we only do this once per particle (we have repeats
// from the edge effect)
```

```
/*
```

```

ist=1;
jstone=0;
for (ist=1; ist<8; ist++) {
    if (angles[i][0] == angles[i+ist][0] && angles[i][2] == angles[i+ist][2]) {
        jstone++;
        //cout << "\n" << jstone;
    }
    else
        break;
}
}

```

```

    i = i + jstone;

    sectnum++;

    }// end stereology i loop (getting all particles tested against sect planes)
  }// end n loop (depends on how many sectioning planes we want for each realization
    // - usually just 1).
*/

// create a file of particle centers and a file of angles

k=0;
for (k=0; k<Nw; k++)
{
  fprintf(ctrstream, "%f%f%f\n", centers[k][0], centers[k][1], centers[k][2]);
  // if (centers[k][2] < -1000)
  //   cout << "\nbad = " << k;
  fprintf(ostream, "%f%f%f%f\n", angles[k][0], angles[k][1], angles[k][2],
angles[k][3]);
  fprintf(radlstream, "%f%f\n", radlen[k][0], radlen[k][1]);

  //   cout << "particle " << (k + 1) << " " << centers[k][0] << " " << centers[k][1] << " " <<
  //   centers[k][2] << " " << angles[k][0] << " " << angles[k][1] << "\n";
}

/*
// Now, create a file of rhinoceros commands

k = 0;
for (k=0; k<Nw; k++) {
  fprintf(rhinostream, "cylinder\n");
  //   fprintf(rhinostream, "w");
  fprintf(rhinostream, "%f", endpoints[k][0]);
  fprintf(rhinostream, ",");
  fprintf(rhinostream, "%f", endpoints[k][1]);
  fprintf(rhinostream, ",");
  fprintf(rhinostream, "%f\n", endpoints[k][2]);
  fprintf(rhinostream, "%f\n", radlen[k][0]);
  //   fprintf(rhinostream, "w");
  fprintf(rhinostream, "%f", endpoints[k][3]);
  fprintf(rhinostream, ",");
  fprintf(rhinostream, "%f", endpoints[k][4]);
  fprintf(rhinostream, ",");
  fprintf(rhinostream, "%f\n", endpoints[k][5]);
}

```

```

    fprintf(rhinostream, "sphere\n");
//   fprintf(rhinostream, "w");
    fprintf(rhinostream, "%f", endpoints[k][0]);
    fprintf(rhinostream, ",");
    fprintf(rhinostream, "%f", endpoints[k][1]);
    fprintf(rhinostream, ",");
    fprintf(rhinostream, "%f\n", endpoints[k][2]);
    fprintf(rhinostream, "%f\n", radlen[k][0]);

    fprintf(rhinostream, "sphere\n");
//   fprintf(rhinostream, "w");
    fprintf(rhinostream, "%f", endpoints[k][3]);
    fprintf(rhinostream, ",");
    fprintf(rhinostream, "%f", endpoints[k][4]);
    fprintf(rhinostream, ",");
    fprintf(rhinostream, "%f\n", endpoints[k][5]);
    fprintf(rhinostream, "%f\n", radlen[k][0]);

}
*/

// Now, create a file of intersection points and sample space sizes

k = 0;
for (k=0; k<intind; k++) {
    fprintf(intstream, "%f%f%f\n", intsectpts[k][0], intsectpts[k][1], intsectpts[k][2]);
}

k = 0;
for (k=0; k<intind; k++) {
    fprintf(int2stream, "%d %d\n", intsectnos[k][0], intsectnos[k][1]);
}

fprintf(lostream, "%f\n", Lfact);

// cout << "\n" << iter << " " << Nw << " " << Lo;

percvol=0;
perc(Nw,Lo,Loxy); // Now, we produce the percolation parameters.
cout << "\n Percvol " << percvol;

percvoltot = percvoltot + percvol;
percvolfrac[iter] = percvol/Vo;

```

```

if ((iter + 1 == iternum)) { // if the statement holds, we have finished the iterations
    // for this step, and will record our information in
    // results.txt before increasing the volume fraction
    // and repeating.

    i=0;
    for (i=0; i<iternum; i++) {
        fprintf(resultstream, "%g %g %g %g %d\n", Vf, percvolfrac[i], tunn, Loos[i], trynum);
    }

    Vf = Vf + .01;

    iter=-1;
    numitt++;

    // iternum = iternum - 4;

    if (numitt%5 == 0 && numitt > 0) {

        tunn = tunn + .01;
        tunnr = tunn/2;
        Vf = .07;
        //iternum = 20;

        // tunn = 0.01;
        // tunnr = 0.005;
    }

}

if (numitt == numit)
    break;

} // this is the end of the iter loop.

fclose(ctrstream);
fclose(oristream);
fclose(radlstream);
fclose(sectstream);

```

```

// fclose(rhinostream);
fclose(intstream);
fclose(int2stream);
fclose(lostream);
fclose(sterstream);

// cout << "\ntotal percolating volume = " << percvoltot << "\n";
// cout << "\n cube length = " << Lo;
cout << "\nSimID = " << SimID << "\n";
// cout << "sectcnt = " << sectcnt << " sectz = " << sectz;
fclose(resultstream);
return 0;
}

int intest(int i)
{
// recently placed particle (@ origin, along y-axis) is called 2, the other is 1.

double fx, fy, fz, x, y, z, x1, y1, z1, r1, r2, l1, l2;
double dist; // distance between particle centers
double testdist1, testdist2; // distance from particle centers to location of nearest pass to
the other line
double shortdist; // shortest distance between particles
double t; // parametric variable to find nearest distance between lines
int k=0, j=0, q=1;
double xint, yint, zint; // intersection points
double xint2, yint2; // working variables for intersection varification
double beta, cb, sb; // thrid rotation ange, sine and cosine.
double ss, shcs, chcs; // coefficients for rotated line
double sh1, ch1, cs1, ss1; // angles for particle 1.
double sa, ca; // sin and cos of single angle between particles.
double xdist, ydist, ytest1, ytest2; // exclusion volume geometry.
double xdisti, ydisti, ytest1i, ytest2i;
double testline; // test distance variable for exclusion volume measurements.
double Ov;
int logp1, logm1, ogip1, gicm1, logic1; // logic variables for exclusion volume.
int logp2, logm2, ogip2, ogim2, gicp2, gicm2, logic2;
int logp3, gicp3, logm3, gicm3, logic3;
int logp4, gicp4, logm4, gicm4, logic4;
int a,b; // +/-1, to use in calculating around exclusion volume

```

```

void rotate1(double *x, double *y, double *z, double cs, double ss, double ch, double
sh);
void rotate2(double *x, double *y, double *z, double cs, double ss, double ch, double sh,
double cb, double sb);
void intersectlist(int i, int k, double xint, double yint, double zint);

mind = 0;

for (k=0; k<i; k++)
{
// define x1, y1, and z1 as the center coordinates of
// the particle to be tested against (particle 1).

x1 = centers[k][0] - centers[i][0]; // move center particle 2 to the origin,
y1 = centers[k][1] - centers[i][1]; // and move particle 1's coordinates accordingly.
z1 = centers[k][2] - centers[i][2]; // effectively, we are moving the whole system over
// such that the most recently placed particle
// is centered at the origin.

// Create working variables for angles

sh1 = angles[k][0];
ch1 = angles[k][1];
ss1 = angles[k][2];
cs1 = angles[k][3];

r1 = radlen[k][0]; // now same for radius and length
r2 = radlen[i][0];

l1 = radlen[k][1];
l2 = radlen[i][1];

Ov = 1 - tunn/(r1 + r2);

// Do the first 'wildfire' distance test

double distest;
dist = sqrt(x1*x1 + y1*y1 + z1*z1);
distest = r1 + r2 + l1/2 + l2/2;

if (dist > distest)
continue;

```

```

// Rotate y-axis so that it lines up with particle 2.
// x,y,z of particle 1's center change accordingly.
// Important: must rotate through +phi2 then +psi2.

x = x1*ch2 - y1*sh2;
y = x1*cs2*sh2 + y1*cs2*ch2 + z1*ss2;
z = -x1*ss2*sh2 - y1*ss2*ch2 + z1*cs2;

// Now do second distance test

if (fabs(y) <= l2/2) {
    dist = sqrt(x*x + z*z);
    distest = (r1 + r2 + l1/2);
}

if (dist > distest)
    continue;

// Now we find the new angles between the particles.
// Or, if you prefer, the coefficients of t in the parametric equation of particle 1's
// axis line when particle 2 aligns with the y-axis.
// ss is the sin of psi, which is the psi angle used in rotating between particle 1 and 2.

shcs = (sh1*cs1*ch2 - ch1*cs1*sh2);
chcs = (sh1*cs1*cs2*sh2 + ch1*cs1*cs2*ch2 + ss1*ss2);
ss = (-sh1*cs1*ss2*sh2 - ch1*cs1*ss2*ch2 + ss1*cs2);

// cout << shcs << " " << chcs << " " << ss << "\n";

// Now, to start the testing for only those particles close enough as defined by the earlier
// wildfire test.

// first, test to see if lines are parallel

if ((shcs == 0) && (ss == 0))
{
// cout << "\nhey!";
Ov = 1 - tunn/(r1 + r2);
}

```

```

if (sqrt(x*x + z*z) > (r1 + r2))
    continue;

// this gets the intersection point

else if (sqrt(x*x + z*z) >= (r1 + r2)*Ov) {

    xint = x*(r2/(r2 + r1));
    yint = y/2;
    zint = z*(r2/(r2 + r1));

    rotate1(&xint, &yint, &zint, cs2, ss2, ch2, sh2);
    intersectlist(i, k, xint, yint, zint);

    mind = mind + 1;
    continue;
}
else {
    q=0;
    break;
}
}

// if not parallel, find the parametric variable for the point of shortest
// distance between lines created by cylinder axes

// this is a fast test that can eliminate
// the additional rotation to orient the exclusion volume.

/* t = -(x*shcs + z*ss)/(shcs*shcs + ss*ss);

// test to see if the point of nearest distance falls between both endpoints
// for both particles

testdist1 = sqrt(t*shcs*t*shcs + t*chcs*t*chcs + t*ss*t*ss);
testdist2 = fabs(t*chcs + y);

if ((testdist1 <= l1/2) && (testdist2 <= l2/2)) {

    shortdist = sqrt((t*shcs + x)*(t*shcs + x) + (t*ss + z)*(t*ss + z));

```

```

    if (shortdist > (r1 + r2)) {
//      if (i + k == 9)
//        cout << "shortdist" << k+1 << i+1 << ": " << shortdist << "\n";
//        cout << "t" << k+1 << i+1 << ": " << t << "\n";
        continue;
    }

    else if (shortdist >= (r1 + r2)*Ov) {

        yint = t*chcs + y;
        xint = (t*shcs + x)*(r2/(r1 + r2));
        zint = (t*ss + z)*(r2/(r1 + r2));

//      cout << "\nOriginal:" << xint << " " << yint << " " << zint << "\n";

        rotate1(&xint, &yint, &zint, cs2, ss2, ch2, sh2);
        intersectlist(i, k, xint, yint, zint);

        mind = mind + 1;
        continue;
    }
    else {
        q=0;
        break;
    }
}
*/
// Now we must treat
// all other cases

// find the value of beta, (or its sine and cosine), that angle through which one must
// rotate to get to the exclusion
// volume orientation

if ((shcs == 0) && (chcs == 0)) {
    beta = pi/2;
    cb = 0;
    sb = 1;
}
else if (shcs == 0) {
    switch (ss*chcs < 0){
    case 1:
        beta = pi/2;
        sb = 1;
        cb = 0;
        break;

```

```

    case 0:
        beta = -pi/2;
        sb = -1;
        cb = 0;
        break;
    }
}
else {
    beta = atan(-ss/shcs);

    if (shcs/chcs < 0)
        beta = beta - pi;

    cb = cos(beta);
    sb = sin(beta);
}

// now rotate the axes beta degrees about y and find the new center point
// of (1).

x1 = x*cb - z*sb;
y1 = y;
z1 = x*sb + z*cb;

// now find the angle between particles 1 and 2.

sa = sqrt((shcs*shcs + ss*ss)/(shcs*shcs + chcs*chcs + ss*ss));
ca = sqrt(1 - sa*sa);

// Okay, now we must develop our overlap parameter a bit. We will arrange it so that
// the pairwise 'inclusion volume' between particles is always constant - in other words,
// as the radii, lengths and angle between the two particles change, the allowable overlap
// (which we define as the fraction of the test parameter (r1 + r2) *less than unity* that
// we will allow) will change, but the total inclusion volume (the volume for which the
// placed particle center will cause an allowed overlap) will not. This should mimic
// a real sintering process, wherein the actual limit for overlap is driven by contact
// area. Calculating the contact area geometrically is very difficult.

// double Ovhn, Ov, a0, a1, a2, SplusT, ST, ST2, R, Q, D;

// then, calculate Ovhn and Ov for the current set of particles
/*
    a0 = -(1 + 0.75*(l1 + l2)/(r1 + r2) + (3.0/(2*pi))*l1*l2*sa/((r1 + r2)*(r1 + r2)) -
3*Vil*(l1 + l2)/(4*pi*(r1 + r2)*(r1 + r2)*(r1 + r2)));

```

```

a1 = (3.0/(2*pi))*l1*l2*sa/((r1 + r2)*(r1 + r2));
a2 = 0.75*(l1 + l2)/(r1 + r2);
R = (9*a2*a1 - 27*a0 - 2*a2*a2*a2)/54.0;
Q = (3*a1 - a2*a2)/9.0;
D = Q*Q*Q + R*R;
ST = pow((R*R + fabs(D)), (1.0/6.0));
ST2 = 2*cos((1.0/3.0)*atan2(sqrt(fabs(D)), R));
SplusT = ST*ST2;
Ovhn = -(1.0/3.0)*a2 + SplusT;
*/
// cout << "\n" << Ovhn;

// Ov = Ovhn;
// Ov = 1 - Ovlp;

// Now test to see if particle 1's center falls into the exclusion parallele-
// piped of particle 2.

fx = fabs(x1);
fy = fabs(y1);
fz = fabs(z1);

xdist = (l1/2)*sa + (r1 + r2);
ydist = (l2/2) + (r1 + r2)/sa;
ytest1 = ydist + xdist*ca/sa;
ytest2 = ydist - xdist*ca/sa;

// 1st make sure that the particle is not totally out of the 'edged' exclusion volume,
// beyond which overlap is impossible.

if (fz > (r1 + r2)) {
// cout << "fz " << k + 1 << " vs. " << i+1 << " = " << fz << "\n";
// q=0; //test
// break;
continue;
}

if (fx > xdist) {
// cout << "fx " << k + 1 << " vs. " << i+1 << " = " << fx << "\n";
// q=0; //test
// break;
continue;
}

```

```

if((x1*y1) > 0) {
  if (fy > ytest1) {
    continue;
  //   q=0; //test
  //   break;
  }
  if (fy > ydist && fx < (((fy - ydist)/(xdist*ca/sa))*xdist)) {
    continue;
  //   q=0; //test
  //   break;
  }
}
else {
  if (fy > ydist) {
  //   q=0; //test
  //   break;
    continue;
  }
  if (fy > ytest2 && fx > (xdist - ((fy - ytest2)/(xdist*ca/sa))*xdist)) {
    continue;
  //   q=0; //test
  //   break;
  }
}
}

```

// Now find out what the nature of the contact might be.

// First, get the dimensions of the inner parallelepiped

```

xdisti = xdist - (r1 + r2);
ydisti = ydist - (r1 + r2)/sa;
ytest2i = ydisti - xdisti*ca/sa;
ytest1i = ydisti + xdisti*ca/sa;

```

// go quadrant-wise like before, but this time, compute the intersection point using  
// the appropriate condition: end-end or end-side. (side-side already handled above)

// we want to handle both sides of the exclusion volume at once - lowering the testing  
// regime to four cases as opposed to 8. We can do this with some long logic statements  
// and some use of +/-1.

```

a = x1/fabs(x1);
b = y1/fabs(y1);

```

```
// p and m refers to positive x1/y1 or negative, where the switch leads to a change of regime
```

```
// logic for center along either vertical side (parallel to y-axis)  
// of the inner parallelepiped
```

```
logp1 = x1 >= xdisti;  
logm1 = x1 <= -xdisti;  
ogip1 = (y1 <= ytest1i) && (y1 >= -ytest2i);  
gicm1 = (y1 >= -ytest1i) && (y1 <= ytest2i);  
logic1 = (ogip1 && logp1) || (logm1 && gicm1);
```

```
// logic for center along either top or bottom side of the inner parallelepiped.
```

```
logp2 = y1 >= (ytest1i - (xdisti - x1)*ca/sa);  
logm2 = y1 <= (-ytest1i - (-xdisti - x1)*ca/sa);  
ogip2 = y1 <= (ytest1i + (xdisti - x1)*sa/ca);  
gicp2 = y1 >= (ytest2i + (-xdisti - x1)*sa/ca);  
ogim2 = y1 >= (-ytest1i + (-xdisti - x1)*sa/ca);  
gicm2 = y1 <= (-ytest2i + (xdisti - x1)*sa/ca);  
logic2 = (logp2 && ogip2 && gicp2) || (logm2 && ogim2 && gicm2);
```

```
// logic for center at top or bottom tips
```

```
logp3 = y1 > ytest1i;  
gicp3 = y1 > (ytest1i + (xdisti - x1)*sa/ca);  
logm3 = y1 < -ytest1i;  
gicm3 = y1 < (-ytest1i + (-xdisti - x1)*sa/ca);  
logic3 = (logp3 && gicp3) || (logm3 && gicm3);
```

```
// logic for side tips
```

```
logp4 = y1 < -ytest2i;  
logm4 = y1 > ytest2i;  
gicp4 = y1 > (-ytest2i + (xdisti - x1)*sa/ca);  
gicm4 = y1 < (ytest2i + (-xdisti - x1)*sa/ca);  
logic4 = (logp4 && gicp4) || (logm4 && gicm4);
```

```
if (logic1) {
```

```
testline = sqrt((x1 - a*(11/2)*sa)*(x1 - a*(11/2)*sa) + z1*z1);
```

```
if (testline < (r1 + r2)*Ov) {  
    q = 0;  
    break;  
}
```

```

    else if (testline > (r1 + r2)) {
        continue;
//    q=0; //test
//    break;
    }
    else {

        xint = (x1 - a*(l/2)*sa)*(r2/(r1 + r2));
        yint = y1 - a*(l/2)*ca;
        zint = z1*(r2/(r1 + r2));

// rotate2 first rotates axes about y -beta, then about x -psi2, then about z -psi2.
// the function calls rotate1.

        rotate2(&xint, &yint, &zint, cs2, ss2, ch2, sh2, cb, sb);
        intersectlist(i, k, xint, yint, zint);

//    float alpha = asin(sa); // test code
//    cout << "\nx1 = " << x1 << "\ny1 = " << y1 << "\nz1 = " << z1;
//    cout << "\nbeta = " << beta << "\nalpha = " << alpha; //test code

        mind = mind + 1;
        continue;
//    q=0; //test code
//    break;

    }
}
}
else if (logic2) {

// this is the intersection of particle 1's axis with the shortest line connecting the
// appropriate end of particle 2 with particle 1's axis. (in the 2-d projection to the
// xy plane)

        xint2 = (b*(l2/2) - y1 + (ca/sa)*x1)*sa*ca;
        yint2 = b*(l2/2)*ca*ca + y1*sa*sa - x1*sa*ca;

        testline = sqrt(xint2*xint2 + (yint2 - b*(l2/2))*(yint2 - b*(l2/2)) + z1*z1);

        if (testline < (r1 + r2)*Ov) {
            q = 0;
            break;
        }
        else if (testline > (r1 + r2)) {
            continue;

```

```

//    q=0; //test
//    break;
}
else {

    xint = xint2*(r2/(r1 + r2));
    yint = (yint2 - b*(l2/2))/2 + b*(l2/2);
    zint = z1*(r2/(r1 + r2));

// rotate2 first rotates axes about y -beta, then about x -psi2, then about z -psi2.
// the function calls rotate1.

    rotate2(&xint, &yint, &zint, cs2, ss2, ch2, sh2, cb, sb);
    intersectlist(i, k, xint, yint, zint);

//    float alpha = asin(sa); // test code
//    cout << "\nx1 = " << x1 << "\ny1 = " << y1 << "\nz1 = " << z1;
//    cout << "\nbeta = " << beta << "\nalpha = " << alpha; //test code

    mind = mind + 1;
    continue;

}
}
else if (logic3) {

//    cout << "\nok" << i;

    testline = sqrt((x1 - b*(l1/2)*sa)*(x1 - b*(l1/2)*sa) + (y1 - b*(l2/2) - b*(l1/2)*ca)*(y1
- b*(l2/2) - b*(l1/2)*ca) + z1*z1);

    if (testline < (r1 + r2)*Ov) {
        q = 0;
        break;
    }
    else if (testline > (r1 + r2)) {
        continue;
//    q=0; //test
//    break;
    }
    else {

        xint = (x1 - b*(l1/2)*sa)*(r2/(r1 + r2));
        yint = (y1 - b*(l1/2)*ca - b*(l2/2))/2 + b*(l2/2);
        zint = z1*(r2/(r1 + r2));

```

```

// rotate2 first rotates axes about y -beta, then about x -psi2, then about z -psi2.
// the function calls rotate1.

    rotate2(&xint, &yint, &zint, cs2, ss2, ch2, sh2, cb, sb);
    intersectlist(i, k, xint, yint, zint);

//    float alpha = asin(sa); // test code
//    cout << "\nx1 = " << x1 << "\ny1 = " << y1 << "\nz1 = " << z1;
//    cout << "\nbeta = " << beta << "\nalpha = " << alpha; //test code

    mind = mind + 1;
    continue;

}
}
else if (logic4) {

    testline = sqrt((x1 - a*(l1/2)*sa)*(x1 - a*(l1/2)*sa) + (y1 - a*(l1/2)*ca + a*(l2/2))*(y1
- a*(l1/2)*ca + a*(l2/2)) + z1*z1);

    if (testline < (r1 + r2)*Ov) {
        q = 0;
        break;
    }
    else if (testline > (r1 + r2)) {
        continue;
//    q=0; //test
//    break;
    }
    else {

        xint = (x1 - a*(l1/2)*sa)*(r2/(r1 + r2));
        yint = (y1 - a*(l1/2)*ca + a*(l2/2))/2 - a*(l2/2);
        zint = z1*(r2/(r1 + r2));

// rotate2 first rotates axes about y -beta, then about x -psi2, then about z -psi2.
// the function calls rotate1.

        rotate2(&xint, &yint, &zint, cs2, ss2, ch2, sh2, cb, sb);
        intersectlist(i, k, xint, yint, zint);

//    float alpha = asin(sa); // test code
//    cout << "\nx1 = " << x1 << "\ny1 = " << y1 << "\nz1 = " << z1;
//    cout << "\nbeta = " << beta << "\nalpha = " << alpha; //test code

        mind = mind + 1;

```

```

        continue;
    }
}

// Finally, if the axes of the particles overlap, such that the shortest distance between
// them goes from one axis to the next . . . . We can keep the logic simple because we
// already made sure, in previous logics, that the particle does not fall around the
// cylindrical / spherical edges & corners of the exclusion volume, (thus eliminating all
// particles too far away in the x and y directions) and also, before the
// progression, we excluded any particles that were too far away in the z-direction. So all
// we have left at this point are those that overlap in z, or are too close in x and y.

    else if (fz >= ((r1 + r2)*Ov)) {

        xint = 0;
        yint = y1 - (ca/sa)*x1;
        zint = z1/2;

        rotate2(&xint, &yint, &zint, cs2, ss2, ch2, sh2, cb, sb);
        intersectlist(i, k, xint, yint, zint);

        mind = mind + 1;
        continue;
    }

// This else makes sure that any particle coming within the inner parallelepiped gets
// tossed out (every possibility outside the inner parallelepiped has already been
// dealt with).
    else {
        q=0;
        break;
    }

// end of loop
}

if (q == 0)
    return 1;
else
// cout << "\n" << "shortdist" << k+1 << i+1 << ": " << shortdist << "\n";
    intind = intind + mind;
    return 0;
}

```

```

void rotate2(double *x, double *y, double *z, double cs, double ss, double ch, double sh,
double cb, double sb)
{
    void rotate1(double *x, double *y, double *z, double cs, double ss, double ch, double
sh);

    // this function rotates the axes -beta, -psi then -phi

    double x1, z1;
    x1 = *x;
    z1 = *z;

    *x = x1*cb + z1*sb;
    *z = -x1*sb + z1*cb;

    rotate1(x, y, z, cs, ss, ch, sh);

    return;
}

```

```

void rotate1(double *x, double *y, double *z, double cs, double ss, double ch, double sh)
{
    // This function rotates the axes -psi then -phi (defined as rotation about
// x-axis and z-axis, respectively), and gives new coordinates for the point passed to it
// if x, y & z are the coordinates of the point and cs, ss are cos and sin of psi and
// ch, sh are cos and sin of phi.

    double x1, y1, z1;
    x1 = *x;
    y1 = *y;
    z1 = *z;

    *x = x1*ch + y1*sh*cs - z1*sh*ss;
    *y = -x1*sh + y1*ch*cs - z1*ch*ss;
    *z = y1*ss + z1*cs;

    return;
}

```

```

// This function simply places intersection data into appropriate global variables

```

```

void intersectlist(int i, int k, double xint, double yint, double zint)
{
    intsectpts[intind + mind][0] = xint + centers[i][0];
    intsectpts[intind + mind][1] = yint + centers[i][1];
    intsectpts[intind + mind][2] = zint + centers[i][2];
    intsectnos[intind + mind][0] = i + 1;
    intsectnos[intind + mind][1] = k + 1;

    return;
}

// Here's the code that finds the percolating volume

void perc(int Nw, double Lo, double Loxy){

    // First, make a dummy variable for intersection numbers called intnos. You will need
    // this. Make another one called intpts for the points.

    cout << "\nintind = " << intind;

    // int intnos[15000][2], intnos1[15000][2];
    // int particles[11000], particles1[11000];
    // int bottom[10000], finish1[10000], temp1[16000], killcount=0;
    // double intpts[5000][3];

    int i=0;

    for (i=0; i<intind; i++) {
        intnos[i][0] = intsectnos[i][0];
        intnos[i][1] = intsectnos[i][1];

        // intpts[i][0] = intsectpts[i][0];
        // intpts[i][1] = intsectpts[i][1];
        // intpts[i][2] = intsectpts[i][2];

    }

    // Create particles

    i=0;
    for (i=0; i<Nw; i++)
        particles[i] = i + 1;

    // Eliminate contacts that fall outside of the cube.

```

```

i=0;
int logicint1, logicint2;
for (i=0; i<intind; i++) {
    logicint1 = (intsectpts[i][0] > Loxy) || (intsectpts[i][1] > Loxy) || (intsectpts[i][2] > Lo);
    logicint2 = (intsectpts[i][0] < 0) || (intsectpts[i][1] < 0) || (intsectpts[i][2] < 0);
    if (logicint1 || logicint2) {
        intnos[i][0] = -1;
        intnos[i][1] = -1;
    }
}

```

// Now, calculate which particles intersect the percolation start and finish, and put them  
// in the vectors bottom and finish1, respectively. Keep track of how many are in each.

```

i=0;
double testz;
int k=0, j=0;

```

// We're going to go ahead and create a dummy particles variable, to use in the first test.

```

int m=0;
for (m=0; m<Nw; m++)
    particles1[m] = particles[m];

```

// Find out which particles intercept the top and bottom.

```

for (i=0; i<Nw; i++) {

    testz = (radlen[i][1]/2)*fabs(angles[i][2]) + radlen[i][0];

    if (testz >= centers[i][2]) {
        bottom[k] = i + 1;
        k++;
        particles1[i] = -1;
    }

    if (testz >= Lo - centers[i][2]) {
        finish1[j] = i + 1;
        j++;
        particles1[i] = -1;
    }
}

```

```

// FILE *botstream;
// botstream = fopen("bottoms.txt", "a");
// fprintf(botstream, "%d\n", k);
// fclose(botstream);

// Now, go through and get rid of all particles that have only one contact (or none).
// But leave the particles that appear in finish1 and bottom

m=0;
int n=0, testone=0, breaktest1, breaktest2=1, testoneind;
int killcount1;
i=0;

while(breaktest2 > 0) {

    breaktest2 = 0;

    for (i=0; i<Nw; i++) {

        // Remember how all of the top and bottom particles are -1 in particles1?
        // This statement assures that they remain in particles through the test.

        if ((particles1[i] == -1) || (particles[i] == -1))
            continue;

        // If it's not a top or bottom particle, test to see if it has more than one contact.
        // If not, erase it and its only intersection. testone is the number
        // of intersections each particle has.

        n=0;
        for (n=0; n<intind; n++) {

            if ((i + 1 == intnos[n][0]) || (i + 1 == intnos[n][1])) {
                testoneind = n;
                testone++;
            }
        }

        if (testone == 1) {
            intnos[testoneind][0] = -1;
            intnos[testoneind][1] = -1;
        }

        // when all remaining interior particles have two or more intersections, breaktest2
        // stays at zero and the while loop ends.

```

```

    if (testone < 2) {
        particles[i] = -1;
        killcount++;

        breaktest2++;
    }

    testone = 0;
}
}

killcount1 = killcount;

m=0;
int onecount=0;
for (m=0; m<Nw; m++) {
    if (particles[m] == -1)
        onecount++;
}

// Now we need to find clusters and eliminate the ones that are stranded. In other words,
// we will eliminate clusters that do not contain both a particle in finish1 and one in
// bottom.

// particlestemp
m=0;
for (m=0; m<Nw; m++)
    particles1[m] = particles[m];
// particlestemp

int p, q, testwoind2, accumtwo, clustsiz;
int testwob, testwof, testwo, killcount2;
i=0;
for (i=0; i<Nw; i++) {

// if we found the particle's cluster earlier, we don't consider it.

    if ((particles[i] == -1) || (particles1[i] == -1))
        continue;

// These are the loops that actually build the cluster. The routine accumulates all
// contacts emanating out from the particle under consideration (i), and collects them
// in the vector temp1. Simultaneously, each intersection that is considered is blanked

```

```
// out in the dummy vector intnos1. The loop continues until all intersections for
// particles appearing in temp1 have been considered.
```

```
// intnostemp
m=0;
for (m=0; m<intind; m++) {
    intnos1[m][0] = intnos[m][0];
    intnos1[m][1] = intnos[m][1];
}
// intnostemp

temp1[0] = i + 1;
testwoind2=1;
accumtwo=0;

p=0;
for (p=0; p<testwoind2; p++) {

    m=0;
    for (m=0; m<intind; m++) {

        if (intnos1[m][0] == temp1[p]) {
            temp1[accumtwo + testwoind2] = intnos1[m][1];
            accumtwo++;
            intnos1[m][0] = -1;
            intnos1[m][1] = -1;
        }

        if (intnos1[m][1] == temp1[p]) {
            temp1[accumtwo + testwoind2] = intnos1[m][0];
            accumtwo++;
            intnos1[m][0] = -1;
            intnos1[m][1] = -1;
        }

    }

    if ((p == testwoind2 - 1) && (accumtwo != 0)) {
        testwoind2 = testwoind2 + accumtwo;
        accumtwo = 0;
    }

}

clustsiz = testwoind2;
```

```
// Now we must find out if the cluster percolates - ie, does it contain both
// a bottom and finish particle?
```

```
n=0;
testwob=0;
testwof=0;
for (n=0; n<clustsiz; n++) {

    q=0;
    for (q=0; q<k; q++) {
        if (temp1[n] == bottom[q])
            testwob++;
    }

    q=0;
    for (q=0; q<j; q++) {
        if (temp1[n] == finish1[q])
            testwof++;
    }

}

testwo = 0;
if ((testwob>0) && (testwof>0))
    testwo++;
```

```
// Now, testwo will be zero only if the cluster does not percolate. If this is the case,
// we want to eliminate all the particles in that cluster and their intersections.
// (remember, we have already blanked out the cluster's intersections in the dummy vector
// intnos1.) On the other hand, if it does percolate, we don't want to consider any
// particles in the cluster again. So we blank them out of our dummy particles vector.
```

```
if (testwo == 0) {

    n=0;
    for (n=0; n<clustsiz; n++) {
        if (particles[temp1[n] - 1] == -1)
            continue;
        else {
            particles[temp1[n] - 1] = -1;
            killcount++;
        }
    }
}
```

```

    m=0;
    for (m=0; m<intind; m++) {
        intnos[m][0] = intnos1[m][0];
        intnos[m][1] = intnos1[m][1];
    }
}
else {

    n=0;
    for (n=0; n<clustsiz; n++)
        particles1[temp1[n] - 1] = -1;

}

} // This ends the second test.

if (killcount == Nw) { // in other words, if we don't percolate
// cout << "\nkilled it";
    return;
}

// killcount2 = killcount - killcount1;

m=0;
int twocount = 0;
for (m=0; m<Nw; m++) {
    if (particles[m] == -1)
        twocount++;
}

/*

// OK, now we just have one more test to go through. We have to eliminate parts of the
// percolating clusters that don't conduct current, ie, stranded peninsulas on which the
// particles nonetheless all have two or more intersections, and therefore escaped the
// first test. (dead loops)

// We do this by examining those particles remaining in the system that have more than
// two contacts. Each dead loop must contact with the main cluster through a particle
// with at least three contacts. This means that if we consider each of these particles,
// temporarily eliminate all its contacts, the clusters surrounding it must touch bottom or
// finish. If a cluster so isolated doesn't do that, it's a dead loop.

// First, let's identify all of the particles that have more than two contacts. Put them

```

```

// in the vector threevect1.

i=0;
int threeint, threevect1[40000], threevect2[100], threevect3[100], threeind1=0;
int threeind2, testthree, testthree1, j1, k1;
for (i=0; i<Nw; i++) {

    if (particles[i] == -1)
        continue;

    m=0;
    threeint=0;
    for (m=0; m<intind; m++) {

        if (intnos[m][0] == i + 1)
            threeint++;

        if (intnos[m][1] == i + 1)
            threeint++;
    }

    if (threeint > 2) {
        threevect1[threeind1] = i + 1;
        threeind1++;
    }
}

// Next, go through the particles that have three or more intersections and find out which
// particles intersect it. Put them in the temporary vector threevect2. At the same time,
// keep track of all of the intersections for each threevect1 particle considered.

```

```

int threeindvect[100];
q=0;
for (q=0; q<threeind1; q++){

    if (threevect1[q] == -1)
        continue;

    m=0;
    threeind2=0;
    for (m=0; m<intind; m++) {

        if (intnos[m][0] == threevect1[q]) {
            threevect2[threeind2] = intnos[m][1];

```

```

    threeindvect[threeind2] = m;
    threeind2++;
}

if (intnos[m][1] == threevect1[q]) {
    threevect2[threeind2] = intnos[m][0];
    threeindvect[threeind2] = m;
    threeind2++;
}
}

```

// Now, go through all of the particles listed in threevect2 and build a cluster, first  
// blanking out the intersections between the particle considered from threevect1 and the  
// corresponding particles in threevect2. After  
// every iteration of the cluster build, check to see if we've hit finish1 or bottom.

```

n=0;
for (n=0; n<threeind2; n++){

    // intnostemp
    m=0;
    for (m=0; m<intind; m++) {
        intnos1[m][0] = intnos[m][0];
        intnos1[m][1] = intnos[m][1];
    }
    // intnostemp

    i=0;
    for (i=0; i<threeind2; i++) {
        intnos1[threeindvect[i]][0] = -1;
        intnos1[threeindvect[i]][1] = -1;
    }

    p=0;
    testwoind2=1;
    accumtwo=0;
    temp1[0] = threevect2[n];

    for (p=0; p<testwoind2; p++) {

// first, we test the cluster addition against bottom and finish1 here, even if it's a
// cluster of 1.

        i=p;

```

```

testthree1=0;
for (i=p; i<testwoind2; i++) {

    k1=0;
    for (k1=0; k1<k; k1++) {
        if (bottom[k1] == temp1[i]) {
            testthree1++;
            break;
        }
    }

    j1=0;
    for (j1=0; j1<j; j1++) {
        if (finish1[j1] == temp1[i]) {
            testthree1++;
            break;
        }
    }
}

```

// the only purpose of this is to allow a break back to the n loop if we hit top or bottom  
// during the above sequence

```

testthree=0;
if (testthree1 > 0) {
    testthree++;
    break;
}

```

// This loop builds out the cluster, getting every intersecting particle for those particles  
// already in temp1 and putting them in temp1 as well.

```

m=0;
for (m=0; m<intind; m++) {

    if (intnos1[m][0] == temp1[p]) {
        temp1[accumtwo + testwoind2] = intnos1[m][1];
        accumtwo++;
        intnos1[m][0] = -1;
        intnos1[m][1] = -1;
    }

    if (intnos1[m][1] == temp1[p]) {
        temp1[accumtwo + testwoind2] = intnos1[m][0];
    }
}

```

```

    accumtwo++;
    intnos1[m][0] = -1;
    intnos1[m][1] = -1;

}

}

if ((p == testwoind2 - 1) && (accumtwo != 0)) {
    testwoind2 = testwoind2 + accumtwo;
    accumtwo = 0;
}

} // p loop

```

// if the cluster escapes, we hit continue here. If it doesn't, the whole cluster is  
// removed from particles and intnos, and the next iteration in threevect2 continues.

```

if (testthree > 0) {
    continue;
}
else {

    p=0;
    for (p=0; p<testwoind2; p++) {

        if (particles[temp1[p] - 1] == -1)
            continue;

```

// The following ensures that we do not eliminate two particles that form a 'triangle' with  
// a main-cluster particle. In other words, the previous test eliminates loops that are  
// live but begin and end on the same particle. What we're doing here is to exempt the  
// shortest of these: loops consisting of two particles. This is a compromise; loops longer  
// than that could be live but would not carry much current because of the shorter  
// path available.

```

    m=0;
    testthree1=0;
    for (m=0; m<threeind2; m++) {

        if (temp1[p] == threevect2[m]) {
            testthree1++;
            break;
        }
    }
}

```

```

    if (testthree1 > 0)
        continue;

    particles[temp1[p] - 1] = -1;
    killcount++;

    m=0;
    for (m=0; m<intind; m++) {

        if (temp1[p] == intnos[m][0]) {
            intnos[m][0] = -1;
            intnos[m][1] = -1;
            break;
        }

        if (temp1[p] == intnos[m][1]) {
            intnos[m][0] = -1;
            intnos[m][1] = -1;
            break;
        }
    }

    } // finishes p-loop

} // finishes else

} // finishes n loop - which tests threect2.

cout << "\nperc: " << q << " of " << threind1;

} // finishes q loop - which tests threect1.

// Actually, just one more. We just need to repeat the first test.

// particlestemp
m=0;
for (m=0; m<Nw; m++)
    particles1[m] = particles[m];
// particlestemp

i=0;
for (i=0; i<k; i++)

```

```

particles1[bottom[i] - 1] = -1;

i=0;
for (i=0; i<j; i++)
    particles1[finish1[i] - 1] = -1;

testone=0;
breaktest2=1;
while(breaktest2 > 0) {

    breaktest2 = 0;

    i=0;
    for (i=0; i<Nw; i++) {

        if ((particles1[i] == -1) || (particles[i] == -1))
            continue;

// If it's not a top or bottom particle, test to see if it has more than one contact.
// If not, erase it and its only intersection. testone is analogous to the number
// of intersections each particle has.

        n=0;
        for (n=0; n<intind; n++) {

            if ((i + 1 == intnos[n][0]) || (i + 1 == intnos[n][1])) {
                testoneind = n;
                testone++;
            }
        }

        if (testone == 1) {
            intnos[testoneind][0] = -1;
            intnos[testoneind][1] = -1;
        }

// when all remaining interior particles have two or more intersections, breaktest2
// stays at zero and the while loop ends.

        if (testone < 2) {
            particles[i] = -1;
            killcount++;

            breaktest2++;
        }
    }
}

```

```

    testone = 0;
  }
}

m=0;
int finalkillcount=0;
for (m=0; m<Nw; m++) {
  if (particles[m] == -1)
    finalkillcount++;
}

// cout << "\n finalkillcount = " << finalkillcount;

*/

// Now, we have to go through and make sure that intpts is up to date, so that we know
// what all of the participating intersections are.
/*
m=0;
int intrem=0;
for (m=0; m<intind; m++) {
  if (intnos[m][0] == -1) {
    intpts[m][0] = -1;
    intpts[m][1] = -1;
    intpts[m][2] = -1;
  }
  else
    intrem++;
}

i=0;
int sumtest=0;
for (i=0; i<Nw; i++) {
  if (particles[i] != -1)
    sumtest++;
}

*/

// cout << "\nsumtest = " << sumtest;

// Now, we have a list of particles that participate in percolation, and their
// intersections. What we need to do is to rotate the axes about each participating

```

// particle, such that it lines up with the y-axis. Then we can easily get the percolating  
// volume of the particle. We collect all of the points intersecting a particular particle  
// in a vector (volvectpts), and then we rotate those points along with the particle.

```
int volvectnos[100], volvectind, bott, seq1=0, seq2=0; //particip[2000];
double volvectpts[100][3], t, x1, y1, z1, x2, y2, z2;
double volp1, volp2, percsum1=0, percsum2=0, perclen=0, lenp, lenp0, radii;
// double pointout[6][3], twal[6], vecto[3], lenpp;
// int pts[2], pntnum, j1=0;
i=0;
for (i=0; i<Nw; i++) {
```

```
    if (particles[i] == -1)
        continue;
```

```
    // First, test to see if the particle's in the bottom or top vectors, and put it's
    // intersection coordinates into volvectpts if it is.
```

```
    // Volvectpts is a list of intersection points for each particle.
```

```
    bott=0;
    volvectind=0;
    n=0;
    for (n=0; n<k; n++) {
        if (particles[i] == bottom[n]) {
            t = -centers[i][2]/angles[i][2];
            volvectpts[0][0] = t*angles[i][0]*angles[i][3] + centers[i][0];
            volvectpts[0][1] = t*angles[i][1]*angles[i][3] + centers[i][1];
            volvectpts[0][2] = 0;
            botpercno[seq1] = bottom[n];
            botpercpt[seq1][0] = volvectpts[0][0];
            botpercpt[seq1][1] = volvectpts[0][1];
            botpercpt[seq1][2] = volvectpts[0][2];
            seq1++;
            volvectind++;
            bott++;
```

```
        break;
    }
}
```

```
n=0;
for (n=0; n<j; n++) {
    if (particles[i] == finish1[n]) {
        t = (Lo - centers[i][2])/angles[i][2];
        volvectpts[volvectind][0] = t*angles[i][0]*angles[i][3] + centers[i][0];
```

```

    volvectpts[volvectind][1] = t*angles[i][1]*angles[i][3] + centers[i][1];
    volvectpts[volvectind][2] = Lo;
    toppercns[seq2] = finish1[n];
    toppercpt[seq2][0] = volvectpts[volvectind][0];
    toppercpt[seq2][1] = volvectpts[volvectind][1];
    toppercpt[seq2][2] = volvectpts[volvectind][2];
    seq2++;
    volvectind++;

    break;
}
}

// Now, fill in the rest of the intersections.

m=0;
for (m=0; m<intind; m++) {

    if ((intnos[m][0] == particles[i] || (intnos[m][1] == particles[i])) {
        volvectnos[volvectind] = m;
        volvectpts[volvectind][0] = intsectpts[m][0];
        volvectpts[volvectind][1] = intsectpts[m][1];
        volvectpts[volvectind][2] = intsectpts[m][2];
        volvectind++;
    }

}

    if (fabs(centers[i][0]) > 100000 || fabs(centers[i][1]) > 100000 || fabs(centers[i][2]) >
100000)
        cout << "\nhey! " << i;

// Now, we center each particle at the origin and line it up with the y-axis.

p=0;
for (p=0; p<volvectind; p++) {
    x1 = volvectpts[p][0] - centers[i][0];
    y1 = volvectpts[p][1] - centers[i][1];
    z1 = volvectpts[p][2] - centers[i][2];

    if (fabs(x1) > 100000 || fabs(y1) > 100000 || fabs(z1) > 100000) {
        for (m=0; m<volvectind; m++)
            cout << "\n " << m << " " << volvectpts[m][0] << " " << volvectpts[m][1] << " " <<
volvectpts[m][2];
    }
}

```

```

    volvectpts[p][0] = x1*angles[i][1] - y1*angles[i][0];
    volvectpts[p][1] = x1*angles[i][0]*angles[i][3] + y1*angles[i][1]*angles[i][3] +
z1*angles[i][2];
    volvectpts[p][2] = -x1*angles[i][0]*angles[i][2] - y1*angles[i][1]*angles[i][2] +
z1*angles[i][3];

}

// Now, we find the lowest and highest y values, and call the distance between them
// lenp.

p=0;
lenp=0;
for (p=0; p<volvectind; p++) {
    for (q=p+1; q<volvectind; q++) {

        x1 = volvectpts[p][0];
        y1 = volvectpts[p][1];
        z1 = volvectpts[p][2];

        x2 = volvectpts[q][0];
        y2 = volvectpts[q][1];
        z2 = volvectpts[q][2];

        lenp0 = fabs(y1 - y2); //sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) + (z1-z2)*(z1-z2));
//      cout << "\n" << x1 << " " << y1 << " " << z1 << " " << x2 << " " << y2 << " " <<
z2;
//      cout << "\nlenp0 " << lenp0;

        if (lenp0 > lenp)
            lenp = lenp0;

    }
}

//  cout << "\nlenp " << lenp;

    volp1 = lenp*pi*(radlen[i][0] - tunn/2)*(radlen[i][0] - tunn/2);

//  for (p=0; p<volvectind; p++)
//      cout << "\n volvectpts " << p << "= " << volvectpts[p][1];

//  cout << "\n" << lenp << " " << tunn << " " << volp1;

```

```

    percvol = percvol + volp1;
    perclen = perclen + lenp;

//   cout << "\nvolp = " << volp1;

//   particip[j1] = i + 1;
//   j1++;

} // end i loop - percolating volume of particles now calculated and summed in percvol.

// Now output all of the percolating intersections (including top & bottom intersections)
and
// the particle numbers that correspond to them.

FILE *pstream;
pstream = fopen("percstuff.txt", "w");

for (i=0; i<intind; i++) {

    if (intnos[i][0] == -1)
        continue;

    fprintf(pstream, "%d %d ", intnos[i][0], intnos[i][1]);
    fprintf(pstream, "%f %f %f\n", intsectpts[i][0], intsectpts[i][1], intsectpts[i][2]);

}

fclose(pstream);

FILE *btpstream;
btpstream = fopen("percbt.txt", "w");

for (i=0; i<seq1; i++) {

    fprintf(btpstream, "%d ", botpercinos[i]);
    fprintf(btpstream, "%f %f %f\n", botpercpt[i][0], botpercpt[i][1], botpercpt[i][2]);

}

for (i=0; i<seq2; i++) {

    fprintf(btpstream, "%d ", toppercinos[i]);
    fprintf(btpstream, "%f %f %f\n", toppercpt[i][0], toppercpt[i][1], toppercpt[i][2]);
}

```

```

}

fclose(btpstream);

/*
i=0;
int volcount=0, partcount;
for (i=0; i<Nw; i++) {

    radii = radlen[i][0] - tunn/2;

    if (particles[i] == -1)
        continue;

    if (angles[i][0] == angles[i-1][0] || angles[i][0] == angles[i+1][0]) {

//    cout << "\n hey";

        vecto[0] = angles[i][0]*angles[i][3];
        vecto[1] = angles[i][1]*angles[i][3];
        vecto[2] = angles[i][2];

        volcount=0;

        n=0;
        for (n=0; n<3; n++) {

            twal[n] = -centers[i][n]/vecto[n];
            twal[n+3] = (Lo - centers[i][n])/vecto[n];

        }

        lenpp = sqrt(vecto[0]*vecto[0] + vecto[1]*vecto[1] + vecto[2]*vecto[2]);

        if (angles[i][0] != angles[i-1][0]) {

            lenp = fabs(twal[0]);

            n=1;
            for (n=1; n<6; n++) {
                if (fabs(twal[n]) < lenp)
                    lenp = fabs(twal[n]);
            }

```

```

lenp = lenpp*lenp;

if (lenp < radlen[i][1]/2) {

    volp1 = pi*radii*radii*(lenp + radlen[i][1]/2) + (2.0/3.0)*pi*radii*radii*radii;
    volp2 = pi*radlen[i][0]*radlen[i][0]*(lenp + radlen[i][1]/2) +
(2.0/3.0)*pi*radlen[i][0]*radlen[i][0]*radlen[i][0];
    percsum1 = percsum1 + volp1;
    percsum2 = percsum2 + volp2;

    continue;
}
else {
    volp1 = pi*radii*radii*radlen[i][1] + (4.0/3.0)*pi*radii*radii*radii;
    percsum1 = percsum1 + volp1;
    volp2 = pi*radlen[i][0]*radlen[i][0]*radlen[i][1] +
(4.0/3.0)*pi*radlen[i][0]*radlen[i][0]*radlen[i][0];
    percsum2 = percsum2 + volp2;

    continue;
}
}

// particles outside the box

first = i;
while (1) {
    if (angles[first-1][0] == angles[first][0])
        first = first-1;
    else
        break;
}

n=0;
for (n=0; n<3; n++)
    vectout[n] = centers[first][n] - centers[i][n];

for (m=0; m<3; m++) {
    if (vectout[m] < 0) {
        n=0;
        for (n=0; n<3; n++)
            pointout[outnum][n] = twal[m]*vector[n];
        outnum++;
    }
    else if (vectout[m] > 0) {
        n=0;

```

```

    for (n=0; n<3; n++)
        pointout[outnum][n] = twal[m+3]*vector[n];
    outnum++;
}
}

n=0;
for (n=0; n<3; n++) {
    m=0;
    for (m=0; m<3; m++) {
        pointout[n][m] = twal[n]*vecto[m] + centers[i][m];
        pointout[n+3][m] = twal[n+3]*vecto[m] + centers[i][m];
        if (n == m) {
            pointout[n][m] = 0;
            pointout[n+3][m] = Lo;
        }
    }
}

n=0;
pntnum=0;
for (n=0; n<6; n++) {
    if (pointout[n][0] >= 0 && pointout[n][1] >= 0 && pointout[n][2] >= 0 &&
pointout[n][0] <= Lo && pointout[n][1] <= Lo && pointout[n][2] <= Lo) {
        pts[pntnum] = n;
        pntnum++;
    }
}

if (pntnum < 2) {
    lenp = sqrt((centers[i][0] - pointout[pts[0]][0])*(centers[i][0] - pointout[pts[0]][0]) +
(centers[i][1] - pointout[pts[0]][1])*(centers[i][1] - pointout[pts[0]][1]) + (centers[i][2] -
pointout[pts[0]][2])*(centers[i][2] - pointout[pts[0]][2]));

    if (lenp > radlen[i][1]/2){
        lenp = radlen[i][1]/2 - lenp;

        volp1 = pi*radii*radii*lenp + (2.0/3.0)*pi*radii*radii*radii;
        volp2 = pi*radlen[i][0]*radlen[i][0]*lenp +
(2.0/3.0)*pi*radlen[i][0]*radlen[i][0]*radlen[i][0];
        percsum1 = percsum1 + volp1;
        percsum2 = percsum2 + volp2;
    }

    continue;
}

```

```

    }

    lenp = sqrt((pointout[pts[1]][0] - pointout[pts[0]][0])*(pointout[pts[1]][0] -
pointout[pts[0]][0]) + (pointout[pts[1]][1] - pointout[pts[0]][1])*(pointout[pts[1]][1] -
pointout[pts[0]][1]) + (pointout[pts[1]][2] - pointout[pts[0]][2])*(pointout[pts[1]][2] -
pointout[pts[0]][2]));
    volp1 = pi*radii*radii*lenp;
    volp2 = pi*radlen[i][0]*radlen[i][0]*lenp;
    percsum1 = percsum1 + volp1;
    percsum2 = percsum2 + volp2;

    continue;

} // This ends the "edge particle" if statement.

*/
// This code is an alternate method of calculating percolating volume, which counts
// the volume of the entire particle.
/*
    volp1 = pi*radii*radii*radlen[i][1] + (4.0/3.0)*pi*radii*radii*radii;
    percsum1 = percsum1 + volp1;
    volp2 = pi*radlen[i][0]*radlen[i][0]*radlen[i][1] +
(4.0/3.0)*pi*radlen[i][0]*radlen[i][0]*radlen[i][0];
    percsum2 = percsum2 + volp2;
    volcount++;

    partcount=1;
    while(partcount) {

        if (angles[i+1][0] == angles[i][0])
            i++;
        else
            partcount=0;

    }

}

cout << "\n volcount = " << volcount;

percvol = percsum1;
tunnvol = percsum2 - percsum1; // this is what we're after!

// cout << "\npercvol = " << percvol;

```

```

*/
/*
FILE *rhinostream2;

rhinostream2 = fopen("rhinoperc.txt", "w");

k=0;
for (k=0; k<Nw; k++) {

    if (particles[k] == -1)
        continue;

    fprintf(rhinostream2, "cylinder\n");
//    fprintf(rhinostream2, "w");
    fprintf(rhinostream2, "%f", endpoints[k][0]);
    fprintf(rhinostream2, ",");
    fprintf(rhinostream2, "%f", endpoints[k][1]);
    fprintf(rhinostream2, ",");
    fprintf(rhinostream2, "%f\n", endpoints[k][2]);
    fprintf(rhinostream2, "%f\n", radlen[k][0]);
//    fprintf(rhinostream2, "w");
    fprintf(rhinostream2, "%f", endpoints[k][3]);
    fprintf(rhinostream2, ",");
    fprintf(rhinostream2, "%f", endpoints[k][4]);
    fprintf(rhinostream2, ",");
    fprintf(rhinostream2, "%f\n", endpoints[k][5]);

    fprintf(rhinostream2, "sphere\n");
//    fprintf(rhinostream2, "w");
    fprintf(rhinostream2, "%f", endpoints[k][0]);
    fprintf(rhinostream2, ",");
    fprintf(rhinostream2, "%f", endpoints[k][1]);
    fprintf(rhinostream2, ",");
    fprintf(rhinostream2, "%f\n", endpoints[k][2]);
    fprintf(rhinostream2, "%f\n", radlen[k][0]);

    fprintf(rhinostream2, "sphere\n");
//    fprintf(rhinostream2, "w");
    fprintf(rhinostream2, "%f", endpoints[k][3]);
    fprintf(rhinostream2, ",");
    fprintf(rhinostream2, "%f", endpoints[k][4]);
    fprintf(rhinostream2, ",");
    fprintf(rhinostream2, "%f\n", endpoints[k][5]);
    fprintf(rhinostream2, "%f\n", radlen[k][0]);
}

```

```
fprintf(rhinostream2, "plane3pt\n");  
fprintf(rhinostream2, "0,0,%f\n", Lo);  
fprintf(rhinostream2, "0,%f,%f\n", Loxy, Lo);  
fprintf(rhinostream2, "%f,%f,%f\n", Loxy, Loxy, Lo);
```

```
fprintf(rhinostream2, "plane3pt\n");  
fprintf(rhinostream2, "0,0,0\n");  
fprintf(rhinostream2, "0,%f,0\n", Loxy);  
fprintf(rhinostream2, "%f,%f,0\n", Loxy, Loxy);
```

```
fclose(rhinostream2);
```

```
*/  
return;  
}
```