Fractal Vasculature and Vascular Network Growth Modeling in Normal and Tumor Tissue

by

Yuval Gazit

B.Sc., Tel-Aviv University (1991)

Submitted to the Harvard-M.I.T. Division of Health Sciences and Technology
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Medical Physics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author .....
Harvard-M.I.T. Division of Health Sciences and Technology
March 19, 1996

Certified by .....
Rakesh K. Jain
Andrew Werk Cook Professor of Tumor Biology
Thesis Supervisor

Certified by .....
Laurence T. Baxter
Assistant Professor of Radiation Oncology
Thesis Supervisor

Accepted by .....
Richard J. Cohen
Chairman, Thesis Committee

Accepted by .....
A. Gray
Co-Director, Harvard-MIT Division of Health Sciences and Technology

Fractal Vasculature and Vascular Network Growth Modeling in Normal and Tumor Tissue

by

Yuval Gazit

Submitted to the Harvard-M.I.T. Division of Health Sciences and Technology
on March 19, 1996, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Medical Physics

Abstract

Tumor vascular networks are different from normal vascular networks, but the mechanisms underlying these differences are not known. Understanding these mechanisms may be key to improving the efficacy of the treatment of solid tumors. We studied the scale-invariant behavior of two-dimensional normal and tumor vascular networks grown using a murine dorsal chamber preparation and imaged with an intravital microscopy station. Our studies show that several types of fractal dimensions can quantitatively distinguish different types of vascular networks. We find that vascular networks exhibit three types of fractal behavior. Tumor networks display percolation-like scaling, representing the first evidence for a biological growth process whose key determinants are local substrate properties. Normal networks belong to one of two other classes of fractal structures: normal arteriovenous networks display diffusion-limited scaling, and normal capillary networks are compact (space-filling) structures.

An apparent contradiction arises between the accepted view that angiogenesis is controlled by diffusion and the observation that normal capillary networks are not diffusion-limited structures. A growth model is developed to determine potential mechanisms responsible for the compact shape of normal capillary networks. The model suggests that in normal angiogenesis it is an autocrine mechanism that is key to the formation of a space-filling capillary bed. The growth model is extended to tumor networks by suggesting that substrate inhomogeneity, most likely due the effect of the extracellular matrix in tumor tissue, is responsible for tumor networks' observed fractal properties. The growth model is explored over a range of parameter values.

A growth model is also advanced for the formation of arteriovenous networks. Although arteriovenous networks display diffusion-limited scaling, they are known to form by the remodeling of a pre-existing compact capillary network, where some capillaries get larger while others are resorbed. The growth model shows that a shear stress based remodeling rule for terminal vessels leads to selective resorption of capillaries, transforming a space-filling capillary mesh into a tree-like network.

The percolation-like nature of tumor vasculature is shown to have important transport implications. It may explain why tumor networks have elevated resistance to blood flow compared to normal networks. Furthermore, by elucidating the scaling of avascular spaces and vessel tortuousity, it is shown that percolation-like tumor networks possess inherent architectural obstacles to delivery of diffusible substances to tumor tissue.

Thesis Supervisor: Rakesh K. Jain
Title: Andrew Werk Cook Professor of Tumor Biology

Thesis Supervisor: Laurence T. Baxter
Title: Assistant Professor of Radiation Oncology

Acknowledgments

My thesis research would not have been possible without the support of many individuals. However, I would like to dedicate this work to two dear people who made this research slightly *im*possible: my daughter, Ronie, and my son, Yonatan.

Many people deserve whole-hearted thanks for making this thesis a reality. First and foremost, my thesis supervisors, Rakesh Jain and Larry Baxter, who supported me unwaveringly throughout my three years at the Steele Laboratory for Tumor Biology. Rakesh and Larry taught me a great deal about conducting scientific research, yet also demonstrated that research can have a warm personal side too. Other people at the Steele Laboratory also contributed to this research. Michael Leunig, David Berk, Dai Fukumura, and Nina Safabakhsh were instrumental in the experimental stage, facilitating the acquisition of the dorsal chamber data. Their experimental acumen was key to the success of this research. Jim Baish from Bucknell University, who visited the lab for a year, became an invaluable collaborator and a close friend. Jim's penetrating insight and comments as well as the occasional piece of child-rearing advice helped advance this research at a brisk pace. Hera Lichtenbeld and Gabriel Helmlinger contributed their time and expertise to facilitate the in vitro experiments. I am grateful for their enthusiasm and good will. Fan Yuan, Bob Melder, Sybill Patan, and others assisted this research through the questions and comments they generated at lab meetings and through their keen interest in the results and implications of this research.

Outside the lab, I am much indebted to Richard Cohen, who served as my HST academic advisor and as my thesis committee chairman, and provided me with precious advice and comments. Gene Stanley from Boston University, who also served on my thesis committee, was a source of unending support and encouragement. I found his passion and enthusiasm very contagious.

Other people in the M.I.T. and Harvard communities with whom I had more limited contacts, but were still a source of learning and inspiration include: Roger Mark, HST's retiring co-director; Toyoichi Tanaka, my Physics departmental advisor; Justin Pearlman, my research supervisor at the NMR Center; Farrish Jenkins, the legendary anatomy professor; and Jim McIntyre and Simon Powell, radiation oncologists at MGH.

The administrative side of my graduate studies was surprisingly smooth due to the devoted services of an extraordinary staff. On the HST side: Patty Cunningham, Keiko Oh, Fred Bowman, Sally Mokalled, Ron Smith, Chris Newell, Carol Campbell, and the late Gloria McAvenia. On the MGH side: Carol Lyons and Phyllis McNally, the omnipotent divas of the Steele Laboratory.

And last but most certainly not least, my wife Inna, who silently shared the burden while raising two children and pursuing a career of her own. It was research that brought us together, and I feel truly blessed to have met her. Knowing that Inna was by side helped me overcome even the most daunting obstacles. My love and thanks to you, Inna.

ישמע חכם ויוסף לקח
ונבון תחבלות יקנה

(משלי א', ה')
(Proverbs 1:5)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The formation of new blood vessels, or angiogenesis, is a key process in the growth and metastasis of tumors [33]. If a tumor is unable to generate its own blood supply it will not increase in size beyond a diameter of approximately a millimeter. While our understanding of the process of angiogenesis has been greatly enhanced on the molecular level over the last two decades [33], there has been little progress in understanding the angiogenic process on the organ and tissue levels. In particular, it is unclear why tumor vascular networks look so different from normal vascular networks although presumably the same growth factors and inhibitors are involved in their formation. Little is known about the determinants of vascular network formation and architecture — the way in which vessels are arranged and interconnected. This knowledge is important in understanding the differences between physiologic and pathophysiologic angiogenesis, and in designing interventions that modify angiogenesis. Furthermore, this knowledge is essential for characterizing flow and transport in tumor vascular networks so that various therapeutic agents (e.g., drugs, heat, antibodies), which have proved effective in vitro, can be better delivered to their target in vivo [61, 62].

1.2 Current State

Three major areas of deficiency are noted when trying to study and compare tumor and normal vascular networks:

- To date, there has been no general network description scheme used to study and compare vascular networks. Commonly used schemes [106] are either limited to a particular network topology [29], or to organ-specific [26] or tumor-specific networks [76] and cannot be utilized to describe a wide range of networks.

- While the visual qualitative differences between normal and tumor vascular networks can be readily appreciated, there is little quantitative knowledge about the architectural differences between the two types.
- Currently there is no comprehensive growth model that is able to reproduce the spatial characteristics of tumor and normal vascular networks.

1.3 Long-Term Goal

In view of the above, the goal of this research was to develop a vascular network growth model based on experimental observations, that would facilitate the understanding of the dynamics and determinants of network growth, shape and architecture, and elucidate the characteristics of flow and transport in normal and tumor vascular networks.

1.4 Hypotheses

In order to achieve the long term goal, the following hypotheses were postulated and tested:

1. A general fractal-based non-deterministic network description scheme can quantitatively differentiate normal from tumor vascular networks.
2. Tumor and normal vascular networks correspond to different statistical growth structures based on fractal measurements.
3. The determinants of tumor vascular network architecture are different from those of normal networks.

1.5 Organization

This report is divided into the following chapters:

Chapter 2 provides a thorough background on the relevant facets of the three major disciplines this work encompasses: angiogenesis, network analysis, and fractal analysis. It describes the current state of knowledge and points out the gaps and limitations which this work seeks to overcome.

Chapter 3 discusses the experimental methods used in this research. It includes a description of the animal preparations, the image acquisition and analysis process, the fractal dimension calculation algorithms, and the programming platforms used for modeling.

Chapter 4 describes the fractal dimension measurement results. The statistical significance of the results is calculated, and the implications for growth modeling are discussed.

Chapter 5 is dedicated to the formulation of a normal capillary network growth model. This model helps to identify the key determinants of normal vascular network formation.

Chapter 6 is dedicated to the formulation of a tumor vascular network growth model. This model helps to identify the key determinants of tumor vascular network formation.

Chapter 7 is dedicated to the problem arteriovenous network formation. A vascular resorption model, the missing element in Laplacian arteriovenous growth models, is developed and discussed.

Chapter 8 analyzes the transport implications of the scale-invariant properties of tumor vascular networks. These include elevated geometric resistance in tumors and inadequate delivery of diffusible substances to tumor tissue.

Chapter 9 discusses future directions for extending and applying the research and results discussed in this thesis.

Publications

Some of the results detailed in the following chapters have now been published. The full citations are listed in bibliography References [5, 6, 7, 41, 42, 43, 44, 45, 46].

Chapter 2

Background

This thesis encompasses several traditionally disparate disciplines. In order to permit proper evaluation of this report's contents, some background material will be given in each of these disciplines. First, I will describe the relevant state-of-the-art knowledge in the field of angiogenesis. Second, I will provide a brief review of the traditional methods of vascular network analysis and outline their shortcomings. I will follow with a summary of fractal descriptors that can be potentially applied to vascular networks. Last, I will present a short summary of fractal growth processes. With this background, I hope that the significance of the findings of this research will be clearly understood.

2.1 Angiogenesis

Angiogenesis, the formation of new blood vessels, is essential to reproduction, development, and wound repair. Under normal conditions angiogenesis is highly regulated, activated for short periods and then fully inhibited. There are, however, disease states that are characterized by persistent and unregulated neovascularization. Unregulated angiogenesis occurs most commonly in neoplastic disease, and it is now well-established that tumor growth and metastasis are angiogenesis-dependent. Other disease states that are characterized by unregulated angiogenesis include arthritis, psoriasis, hemangiomas, and approximately 20 ocular diseases. My research focused on tumor angiogenesis.

During the last decade the field of angiogenesis has advanced considerably. At least 25 endogenous molecules stimulating or suppressing angiogenesis have been identified [30]. While much information has been gathered about the biochemical and structural properties of these molecules, there is only a hazy conception regarding the ways these molecules mediate angiogenesis in vivo and how they are regulated in both normal and tumor tissue. Furthermore, the interrelations between the different molecules and the interrelations between possible angiogenic pathways are virtually unknown. Since most angiogenesis research has concentrated on the identification of single effector molecules, it is less than surprising that there are only dim ideas regarding the mechanisms which determine the

shape and architecture of a whole network of blood vessels.

In the following paragraphs I will outline some basic experimental observations which are germane to this research, and describe the current biological concepts regarding the expression of the angiogenic phenotype in tumors.

2.1.1 Experimental Observations

Most neovascularization studies [20, 33, 88] have shown that vascular growth occurs on the capillary or post-capillary venule level. One study, dealing with tumor neovascularization in the rat, proposed indirect evidence for angiogenesis on the terminal arteriole level [53]. Generally, however, there is a consensus that only remodeling of existing vessels occurs on the artery-vein level (see Chapter 7).

The dependence of tumors on angiogenesis has been observed in a multitude of experimental models. A tumor's ability to neovascularize is not necessarily connected to its mitotic capability and hence a pre-vascular phase is recognized in tumor development. Lesions at this phase are spherical or flat, grow at a slow linear rate, and rarely metastasize. When neovascularization occurs, the tumor enters a vascular phase which is characterized by rapid exponential growth and increased metastatic potential. The tumor cells prefer to grow around blood vessels, forming cylindrical outshoots into the surrounding tissue. This "preference" is not necessarily connected to the nutrient supply furnished by blood, since this observation has also been made in vitro where there was no blood flow in capillaries [100]. The hyperpermeability of the new vessels coupled with the absence of lymphatics lead to an elevation in interstitial fluid pressure (IFP) in tumors. Observed avascular necrotic centers in growing tumors are generally attributed to compression and occlusion of vessels due to growing cancer cells and constantly generated matrix. There are also avascular regions in tumors which never become vascularized, with sizes of up to several hundred microns. It appears that once a threshold number of cells have switched to the angiogenic phenotype, sufficient neovascularization occurs so that the whole tumor population can expand [33]. It should further be mentioned that there are cases in which an angiogenic capability in and by itself is not sufficient for the progression of a solid tumor, since there are benign tumors which are highly vascularized (e.g., adrenal adenomas, hemangiomas).

2.1.2 How is Angiogenesis Expressed?

In the course of investigating the molecular promoters of angiogenesis it became clear that the situation was much more complex than the simple picture of a few molecules secreted by tumor cells which directly influence vascular endothelial cells. There are both inhibiting and activating angiogenic molecules which normally act in concert to maintain a quiescent microvascular network, where the turnover rate of endothelial cells is measured in thousands of days. Even in tumors it has been observed that both proteases and their inhibitors are secreted simultaneously, and that the

balance between them regulates the level of extracellular proteolysis, thus promoting or suppressing angiogenesis. During periods of regulated angiogenesis, as in wound healing, vascular endothelial cells can undergo rapid proliferation, and their turnover is then measured in days, but this process is also rapidly inhibited at the appropriate time. In tumor angiogenesis, the angiogenic phenotype may present in a variety of ways, listed below, that point out to several qualitative classes of processes. Not all of these processes occur in all tumors: some are more common and general, and some occur only in specific tumor types.

Angiogenic molecules effect angiogenesis in various ways:

- Direct mitogenic and/or chemotactic effect on vascular endothelial cells.
- Mitogenic and/or chemotactic effect on other cell types (fibroblasts, macrophages, mast cells) which then produce angiogenic factors.
- Indirect effects on the endothelium (e.g., increase in permeability) and the extracellular matrix (ECM) which initiate a cascade of events leading to angiogenesis.

To complicate the experimental assessment of the effects of these molecules, some molecules may exhibit conflicting effects, depending on the route of administration (e.g., TNF-α is angiogenic when injected extravascularly, but may cause tumor necrosis if injected intravascularly). A list of the eleven best known angiogenic polypeptides and their effects appears in Table 2.1. A diagram adapted from Reference [36] showing a simplified model of angiogenic phenotype expression in tumors is shown in Figure 2-1. The variety and complexity of actions and interactions even in this simple model is evidently great. Nevertheless, several important qualitative features should be noted. First, no matter which molecules are involved, an initial diffusive stage is always present. Second, the diffusing molecules may affect the endothelial cells directly, or by a series of mediators. The possibility of various mediating effects is key to understanding why a simple diffusive model is not sufficient to explain the formation of vascular networks. Third, the complex tumor angiogenesis activation process, as evidenced in Figure 2-1, does not necessarily contain any novel mechanisms which do not occur in normally regulated angiogenesis. Although there are presumably no growth factors which are unique to tumor angiogenesis, it is certain that the processes described in Figure 2-1 can be significantly influenced by changes in the microenvironment of the growing vessels.

As we stand entangled in the myriad of molecular and cellular events involved in angiogenesis and given the large gaps of knowledge even on these levels, it is clear that if one were to take a bottom-to-top approach to unmasking the behavior of vascular networks on the tissue and organ levels, one should expect to spend an extremely long time[1] untangling these mysteries. It is rather simplistic to think that the determinants of vascular network formation can be unlocked by the characterization of one molecule or another.

[1]Longer, even, than the time it takes to complete a Ph.D. thesis.

Table 2.1: Angiogenic Polypeptides and Their Actions (after References [33, 35])

Molecule	**Effects**	**Other Properties**
aFGF & bFGF	Mitogenic and chemotactic for vascular endothelial cells, fibroblasts, and smooth muscle cells.	Biological activities mediated by heparin or heparin-like molecules. bFGF is sequestered in ECM around blood vessels. Both FGFs are cell-associated and are normally not secreted.
Angiogenin	Does not appear to be mitogenic or chemotactic to vascular endothelial cells.	Present in normal cells and some neoplastic cells. Angiogenin is a secreted protein. Mechanism of angiogenic action not understood.
TGF-α	Mitogenic for vascular endothelial cells, fibroblasts, and epithelial cells.	Secreted by macrophages and some tumors such as sarcomas.
TGF-β	Inhibits growth of many cell types including endothelial cells in vitro, but induces angiogenesis in vivo.	Secreted by many cells in an inactive form. Activated by heat, acid, proteases. Angiogenetic activity may be mediated by macrophages. May have a bifunctional effect on angiogenesis depending on local tissue density of macrophages. Appears to be involved in wound healing, inflammation and differentiation of mesenchymal tissues.
TNF-α	Inhibits endothelial cell proliferation in vitro, but induces angiogenesis in vivo. Chemotactic for endothelial cells.	Secreted by tumor cells as well as by activated macrophages. Thought to be one of the major angiogenic molecules of macrophages.
PD-ECGF	Mitogenic for endothelial cells.	Thought to act physiologically as a maintenance factor for vascular endothelium.
VEGF	Mitogenic for endothelial cells. Increases vascular permeability.	Isolated from various neoplastic cells.
GM-CSF	Endothelial mitogen in vitro.	Granulocyte colony-stimulating factor. Directly augments tumor growth.
PGF	Endothelial mitogen in vitro.	Placental growth factor.
IL-8	Endothelial mitogen in vitro.	Directly augments tumor growth.
HGF	Endothelial mitogen in vitro.	Hepatocyte growth factor.

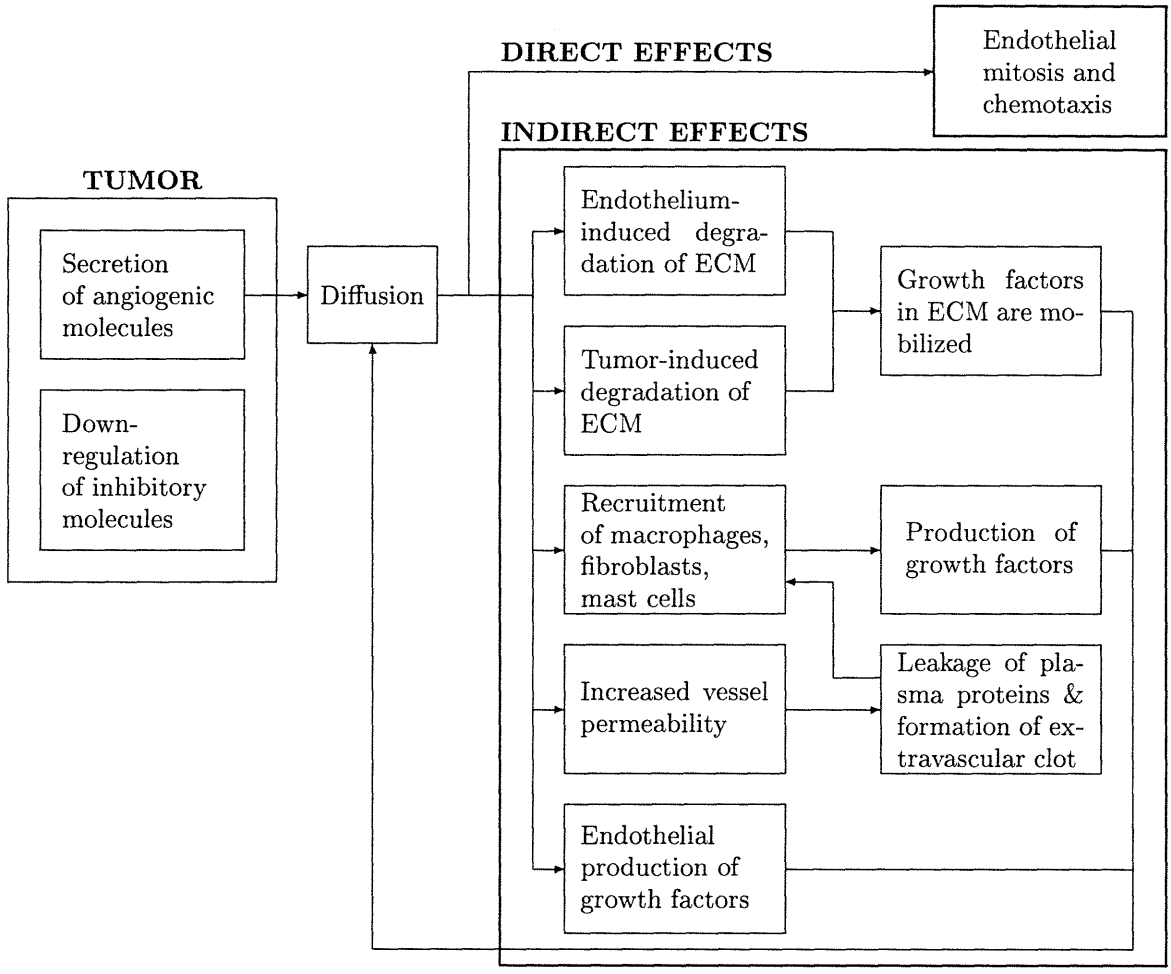

Figure 2-1: Mechanisms Involved in Tumor Angiogenesis

2.2 Network Analysis

Since the question of the determinants of vascular network shape formation cannot be readily answered by considering angiogenesis on the molecular and cellular level, a top-to-bottom approach may prove more practical. Using such an approach, analysis of the properties of the whole network could possibly shed new light on the mechanisms which underlie its formation. This would be beneficial in two ways: in creating a basis for studying the transport properties of vascular networks by facilitating the creation of realistic network models; and in pointing out the key qualitative pathways and cascades in the expression of the angiogenic phenotype. To this end, an overview of "traditional" deterministic network analysis schemes will be given. In delineating their shortcomings, I will lay the ground for the introduction of a non-deterministic (fractal) network analysis scheme.

The most important task in any attempt at network analysis is to develop a description scheme, which functions as the "language" by which the network's features are described. Such a scheme must be self-consistent so that a unique description is obtained for every network analyzed. Such a scheme should also be generalizable and portable, so that it can be applied to different types of networks. Historically, network description schemes were first developed as a tool in geological research, and were then adopted and expanded by researchers of microcirculation. The researchers were primarily interested in the flow behavior of the systems under study, and thus the descriptors that were developed were not geared towards characterizing the spatial behavior of networks. These descriptors were very useful for designing computer models which reproduced the flow behavior of the experimentally observed networks, but were not concerned with reproducing the spatial behavior of those networks. Only with the advent of fractal geometry did a descriptive tool become available for spatial characterization.

Deterministic network description schemes seek to characterize two major facets of network construction: network topology and network geometry. The topology of the network deals with the way in which vessels are connected to each other, while the geometry of the network deals with the geometric parameters that characterize each vessel in the network (diameter, length, branching angle, taper, cross-sectional shape). The simplest networks to describe are vascular trees. The situation becomes much more difficult if one tries to describe other types of networks. We therefore end up in a situation where network description schemes [106] are either limited to a particular network topology [29], or are limited to organ-specific [26] or tumor-specific networks [76] and cannot be utilized to describe a wide range of networks.

2.2.1 Characterizing Vascular Trees

Two basic methods are used to characterize the vascular topology of trees. The two methods originate from the different ways in which vessel generations can be ordered. The first method to be

employed for microvasculature description classified the vessels according to a centrifugal scheme. The first order was given to the largest vessel in the network, and at each bifurcation the vessels were assigned the same or the next order. This method relied on geometric characteristics of the vessels (diameter, branching angle) to determine the order. In this respect, it is rather arbitrary and impairs the capability to compare the topological characteristics of different networks. Later on, the centripetal method of ordering was applied to vessel networks [29]. In this method, known as the Strahler method, the first order is assigned to the terminal vessels of the network. Subsequently, when two vessels of the same order join together, a next order vessel is formed, and when two vessels of different orders are joined together, the higher order of the two is retained. This method has a major advantage over the centrifugal scheme, since it does not employ any geometric information in classifying the vessels, and therefore the geometric characteristics of the network can be studied independently. Such studies reveal constant branching ratio, diameter ratio and length ratio between different pairs of vessels of consecutive orders [106]. These experimental findings are also known as Horton's laws. Thus, we can define ratios based on the whole network as:

$$R = \frac{1}{K-1} \sum_{k=1}^{K-1} \frac{q_k}{q_{k+1}} \tag{2.1}$$

where K is the highest order in the network, and q is the quantity measured (number of vessels, length, or diameter). The values of these ratios are not independent of each other [67]. The nature of this method allows comparison between different trees, and thus makes it a useful tool for building computer models.

There have been many extensions of the basic features of the ordering schemes outlined above. One such extension [79], for example, used the centrifugal scheme but avoided the reliance on geometric characteristics by incrementing the vessel order automatically at each bifurcation. On one hand, such a method is advantageous for studying the vessels connecting the arteriolar and the venous trees, which are topologically similar in the classic Strahler ordering scheme. On the other hand, it does not elucidate the general geometric properties as the Strahler scheme does.

2.2.2 Characterizing Vascular Arcades

Arcades, or loops, do not lend themselves to such useful general schemes as trees do. In one study of arterial arcades [26], an attempt was made to draw a list of variables that would characterize the arcade, such as the number of feeder arterioles, the number of bifurcations inside or on the perimeter, etc. The independent variables were identified and measurements were made. Although this provided a scheme for quantifying the arcade, there was very little geometric information that could be drawn from such an analysis. Furthermore, it is not a useful scheme for modeling, since it averages out the variances in the data, and it is not generalizable to other types of networks.

There has been an attempt to characterize the complex tumor microvasculature [76] using an adaptation of a centrifugal scheme. The major flaw in such an approach is, again, the reliance on apriori geometric information to characterize the topology, thus inserting a non-negligible arbitrariness into the results, and making them very hard to compare with theoretical models or with other experimental results.

Graph theory provides a convenient way to describe pure topological characteristics of any arrangement of vessels. Although a detailed analysis of vascular topology can be made [132], virtually no geometric or spatial information can be drawn from such an analysis.

Other mathematical representations of vascular networks have been suggested. Such representations [19] inevitably fall into the trap that a generally applicable deterministic method requires so much information about each and every vessel that it cannot be practically applied to the analysis and modeling of large networks.

In summary, this synopsis of the attempts to characterize vessel networks leads to a conclusion that because of the complexity and variability of the structures involved, any method which is based on a vessel-by-vessel analysis is bound to fail. This is especially true when trying to study the erratic architecture of tumor networks. Perhaps the only plausible approach is a non-deterministic analysis of the structure-as-a-whole, which is the topic that will be addressed next.

2.2.3 Fractal Description of Networks

In view of the limitations of the traditional network description schemes, the approach which this research took was a non-deterministic one, where the statistical properties of the network-as-a-whole were described, and vessel-by-vessel analysis was disposed with. Fractal analysis, which relies on the scale-invariant properties of the analyzed objects, offers such an approach (a general introduction to fractals can be found in References [27, 85]). Previous fractal analyses of vascular networks have been mostly limited to measurement of the fractal dimension of the whole network in the natural two-dimensional compartment of the retinal vasculature [28, 72, 82]. As I will next describe, more measurements can be made, from which valuable information and insight can be gained. This is important because different statistical growth processes lead to structures with different fractal dimensions, and by measuring different dimensions one may be able to accurately associate a given structure with a certain statistical growth process (e.g., the fractal dimension d_f of a diffusion-limited aggregate and that of a percolation cluster are the same when both are embedded in three dimensions, but they have a different minimum-path dimension d_{min}).

It should be mentioned that there are various algorithms for measuring fractal dimension. While these algorithms will be discussed in detail in section 3.4, it is important to note that they can yield slightly different results. Measurements of fractal dimension can be applied to carefully defined subsets of the network. The fractal dimensions that can potentially be measured for vascular networks

are:

- The dimension of the network-as-a-whole d_f; this number characterizes the overall scaling behavior of the structure and gives an idea of how "dense" the network is.
- The dimension of the minimum-path between two opposite sides of the network d_{min}; this number provides an idea of the efficiency of propagation through the network without revisiting sites [126]. In biological terms it is a measure of the scale-invariant tortuousity of the vessels.

As mentioned above, the benefit of using fractal measurements for characterizing vascular networks does not lie only in the meaningful fractal dimensions that can be measured, but also in the fact that specific values correspond to specific statistical growth processes, which can then be incorporated into growth models. The models' attributes can then be correlated with the biological processes underlying angiogenesis. A discussion of fractal growth processes follows in section 2.3.

As a final note, there is another benefit to measuring fractal dimensions in view of the traditional methods: Horton's ratios can be derived from fractal dimension measurements instead of the tedious vessel-by-vessel analysis. To show this derivation, I will use the concept of the diameter exponent Δ that was introduced by Mandelbrot [85]. This diameter (or radius) exponent appears in the equation relating the radii of vessels at a bifurcation:

$$r_0^{\Delta} = r_1^{\Delta} + r_2^{\Delta} \tag{2.2}$$

The diameter exponent is equal to the fractal dimension d_f for self-similar trees [85]. With Equation 2.2 in mind, and with the assumption that at a bifurcation the difference between vessel orders can be 1 at most[2], it follows that $\sum r_i^{\Delta}$ is constant for each branching order i and therefore:

$$\sum_{j=1}^{n_i} r_{i,j}^{\Delta} = \sum_{j=1}^{n_k} r_{k,j}^{\Delta} \tag{2.3}$$

where k, i are any two branching orders, and n_k, n_i are the numbers of vessels in each order, respectively. Taking into account Horton's laws one can then write:

$$\sum_{j=1}^{n_i} r_{i,j}^{\Delta} = R_B^{k-i} \sum_{j=1}^{n_k} \left(\frac{r_{k,j}}{R_D^{k-i}} \right)^{\Delta} \tag{2.4}$$

where R_B and R_D are the braching ratio and the diameter ratio, respectively.

[2]In practice, this assumption is only partially valid. Studies of the pig coronary arterial tree [65] and the dog pulmonary venous tree [40] have shown that some bifurcations violate this assumption. It should be noted that these studies used a diameter-dependent vessel ordering scheme and not the classic Strahler scheme which is independent of geometric parameters.

Combining equations 2.3 and 2.4 it follows that:

$$R_B \cdot R_D^{-\Delta} = 1 \tag{2.5}$$

and therefore[3]:

$$d_f = \frac{\ln R_B}{\ln R_D} \tag{2.6}$$

Other authors have further shown that [110]:

$$\frac{d_f}{d_{min}} = \frac{\ln R_B}{\ln R_L} \tag{2.7}$$

2.3 Fractal Growth Processes

Fractal growth processes reflect non-equilibrium growth phenomena which lead to the formation of scale-invariant structures. While the physical mechanisms that govern the growth may vary from one structure to another, these physical mechanisms can usually be correlated with more general statistical growth rules which group these growth phenomena into four broad classes described below. Generally, each process in each class is defined by a set of statistical growth rules that reflect growth on a square lattice by the addition of identical particles. I will rely on the assumption that if one observes a physical structure whose fractal properties are similar to properties of structures formed by a certain fractal growth process, one can then gain intuition as to the physical mechanisms underlying the growth. With this intuition one can then construct realistic growth models of these structures.

Local Growth Processes

If the statistical growth rule for a particular process depends only on the immediate environment of the site where growth occurs, the growth process is considered local [131]. Examples of such processes are spreading percolation (e.g., as manifested in epidemic spread or flame propagation), invasion percolation (e.g., as manifested by one fluid displacing another in a porous medium). Percolation structures are generally characterized by loops and voids of many length scales and a fractal dimension of $d_f \approx 1.9$ when embedded in two dimensions. To date, no biological growth process has been found to be adequately modeled by a local growth process.

[3]Equation 2.6 can be verified experimentally for the bronchial tree, which, by virtue of its function, is the only biological tree close to being space-filling. For such a tree we would expect d_f to be close to 3. Indeed, published values for R_B and R_D in the bronchial tree of 4 species [54] concur with our expectation. These values yield a mean d_f of 2.885 in human, 2.831 in dog, 2.809 in rat, and 2.903 in hamster bronchial tree.

Laplacian Growth Processes

Growth processes which are governed by the spatial distribution of a Laplacian field-like quantity (which is inherently non-local) are called Laplacian or "diffusion-limited" [131] (since the probability field of finding a diffusing particle at a given point is a Laplacian quantity). Other such quantities include temperature or electric potential fields. The spatial behavior of these quantities obeys the Laplace equation with moving boundary conditions. Examples of such processes include diffusion-limited aggregation [133] (e.g., as an archetype for bacteria colony formation or viscous fingering), diffusion-limited deposition (e.g., as manifested in electrodeposition), and their many variations. Diffusion-limited aggregation has been found to be a good model for biological growth processes such as neurite growth [18] and bacterial colony growth [10, 87]. Diffusion-limited structures are generally characterized by a tree-like morphology with no loops and a fractal dimension of $d_f \approx 1.7$ when embedded in two dimensions.

Self-Affine and Compact Growth Processes

Not all objects have a non-trivial fractal dimension. There are many growth processes which produce space-filling structures whose dimension is equal to the Euclidean embedding dimension. These structures are known as "compact". In some of these structures the surface (or interface) is fractal. Irreversible growth processes rarely result in smooth interfaces. If the scaling properties of such an interface are not isotropic it is called "self-affine". The first model of self-affine interface growth, the Eden model, was actually proposed in conjunction with tumor growth modeling [131]. Other such processes include ballistic aggregation and deposition (e.g., in vapor deposition on a cold substrate). By virtue of its space-filling nature, compact growth implies that there are no variations in local or non-local properties that influence the growth perimeter.

Cluster-Cluster Growth Processes

There are many physical processes where diffusion-limited aggregates (or clusters) are allowed to diffuse themselves. Such processes can modeled by cluster-cluster aggregation (CCA). CCA well represents the physical mechanisms in an actual system of aggregating particles (e.g., smoke particles), unlike DLA which is more of a general paradigm of Laplacian growth [131]. It is therefore not immediately germane to the problem at hand.

Several qualifying statements should be made regarding the above list:

- There are many variations of these archetypical processes which involve modifications to the growth rules and are too numerous to list. A statistical growth process should be modified

according to the scaling properties of the observed objects, so that it can reproduce these properties.

- While many growth processes are closely related or may display some similar fractal characteristics, there have been extensive studies which have yielded more esoteric exponents that can differentiate among the growth processes [126]. It is then the analyst's task to assign physical meaning to these exponents.
- These archetypical processes are idealizations. They approximate reality only to a limited extent for two reasons:
  - Because of the complexity and variety of physical mechanisms involved.
  - Because of the inherent randomness in the physical mechanisms and the models themselves.

It is useful to think of these statistical growth processes as limiting cases of different physical mechanisms. For example, if one considers flow through a porous medium where both capillary forces and viscous forces are at play, one can see that in the limit where capillary forces are dominant the invasion percolation model holds, and in the limit where viscous forces are dominant the DLA model holds [75]. I will show that vascular networks also display different domains of scale invariant behavior which correspond to different classes of fractal growth processes.

Chapter 3

Methods

This chapter describes the experimental methods used to study vascular networks in this research. It will encompass the procedures implemented in growing the vascular networks, the imaging procedures, and the analysis procedures. The in vitro experiments are described separately in Appendix D.

3.1 Dorsal Window Preparation

Two-dimensional vascular networks were generated using the murine dorsal skinfold chamber preparation (Figure 3-1) [73, 78]. The experiments were performed in three mouse strains: C3H, T-cell deficient nude, and severe combined immunodeficient (SCID) mice. The latter two serve as convenient animal models to grow human tumor xenografts without rejection. The mice were bred and maintained in a pathogen-free environment in our laboratory. In an initial surgical procedure, two symmetrical titanium frames (total weight 3.2 g) which are mirror images of each other (Workshop, Department of Radiation Oncology, Massachusetts General Hospital) were implanted in the back of the mouse so as to sandwich the extended double layer of skin. One layer of skin was then microsurgically removed in a circular area of approximately 15 mm in diameter, and the remaining layer consisting of epidermis, subcutaneous tissue, and thin striated skin muscle was then covered with a coverslip incorporated into one of the frames. Following implantation of the transparent chamber, the animals were allowed to recover for 24 hours [78].

For the study of tumor vascular networks, human colorectal adenocarcinoma LS174T was used in SCID and nude mouse preparations. In addition, three murine tumor cell lines were used in C3H mice: Sa1 murine sarcoma, SCC7 murine squamous cell carcinoma, and MCaIV murine mammary carcinoma[1]. The choice of these tumor cell lines was due to previous experience with them in our laboratory and to the existence of published descriptions of their pathophysiologic behavior [1, 78].

[1]Near the completion of this study another tumor line, Shionogi (SC115) murine mammary carcinoma, was briefly used in SCID mice. See Appendix C for further details.

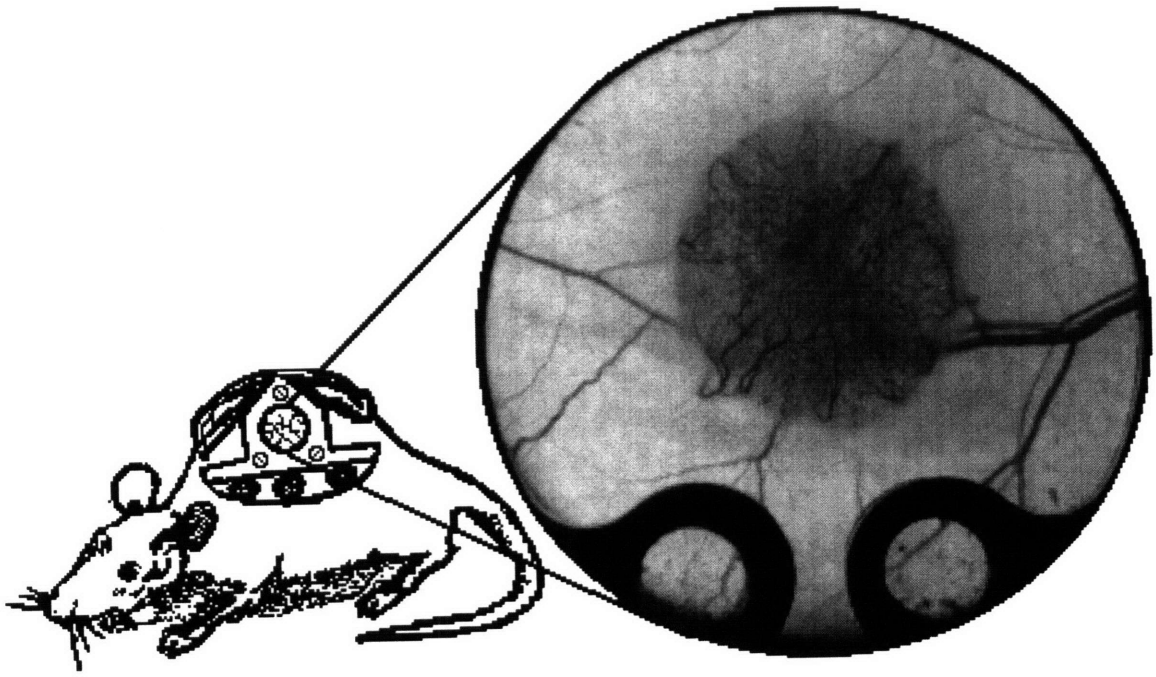

Figure 3-1: Dorsal window preparation in a nude mouse. This chamber was implanted with the LS174T human adenocarcinoma cell line. The tumor is now fully vascularized. The blood in the tumor gives it its dark appearance in relation to the underlying tissue.

The coverslip of the chamber was removed, 2 μl of a dense tumor cell suspension from cell culture ($\sim 2 \cdot 10^5$ cells) were inoculated onto the upper tissue layer of the chamber preparation (striated skin muscle), and the coverslip was replaced. The relatively small volume of tumor cells was used so as to avoid disseminated growth of tumors in the whole chamber.

For the study of normal vascular networks two preparations were used. One preparation consisted of the dorsal chamber with no implanted material. This preparation was used to study the normal subcutaneous vascular networks. In a second preparation, bone-implanted nude mice chamber preparations [77] were used. After sacrifice of newborn nude mice, both femora of each animal were blunt dissected in 10 ml Hanks balanced salt solution (HBSS, Gibco Laboratories, Grand Island, NY) at room temperature and cleaned from soft tissue. Afterwards, femora were transferred to a second dish containing fresh HBSS at room temperature. Only femora without signs of tissue damage (fracture, cartilage damage) as verified under a stereotactic microscope were used for implantation. The coverslip of the chamber was removed, a fresh femur, after 60 minutes of ischemia, was implanted onto the upper tissue layer of the chamber, and the coverslip was replaced [77].

3.2 Image Acquisition

Images of dorsal chamber vascular networks in nude and C3H mice were acquired when the tumors were approximately 4 mm in diameter (7-16 days old). The mice were anesthetized (s.c. injection of

ketamine hydrochloride, 0.1 mg, and xylazine, 0.01 mg, per g body weight) and were then positioned in a polycarbonate tube of 25 mm diameter. The tube was then directed so that the chamber was placed on the stage of an upright microscope (Zeiss Universal; Thornwood, NY) and imaged. Brightfield images were directed to an intensified CCD camera (model 2400; Hamamatsu, Japan). In the intravital microscopy station outside the pathogen-free colony, the video signal was digitized using a PC-AT computer (IBM; Boca Raton, FL) equipped with a DT-2851 image processing board (Data Translation; Marlborough, MA). In the intravital microscopy station inside the pathogen-free colony, the video signal was recorded on a videocassette recorder (model AG-6500; Panasonic; Secaucus, NJ). Images were then digitized using a videocassette player (model SVO-9500MD; Sony, Japan), a time-base corrector (model TBC-IV with ES-2200 expansion system; D.P.S., Florence, KY), and a Macintosh IIfx (Apple Computer; Cupertino, CA) equipped with a DT-2255 image processing board (Data Translation; Marlborough, MA). For contrast enhancement of some normal subcutaneous vascular networks, a bolus of 0.1 ml of fluorescein isothiocyanate-labeled dextran (FITC-dextran, M_r 150000; 5 mg/100 μl of 0.9% NaCl; Sigma, St. Louis, MO) was injected into the tail vein of the mice approximately 5 minutes prior to image acquisition. Epiillumination was achieved by a 100W mercury lamp (model 770; Opti-Quip, Highland Mills, NY) using an excitation filter (485–505 nm), a dichroic mirror (510 nm) and a barrier filter (530 nm).

3.3 Image Analysis

Image analysis was performed on a Macintosh LC III (Apple Computer; Cupertino, CA) using NIH-Image software (Wayne Rasband, N.I.H.; Bethesda, MD). Raw grayscale images were first processed to maximize contrast. Because the background illumination was markedly uneven, background leveling was achieved by performing grayscale dilation and then smoothing with a Gaussian kernel [111]. The image was subsequently divided into a series of parts. In each part thresholding was performed and the vascular network was traced out. Network tracings were then converted into binary images and skeletonized automatically to prevent artifacts in the subsequent fractal analysis [18]. The image analysis process is summed up in Figure 3-2. Each skeletonized image was verified by projecting it onto the raw grayscale image to assure that no vessels were excluded.

3.4 Fractal Dimension Calculation

The fractal dimension (d_f) of the whole network was measured using three different algorithms. The algorithms were verified against known fractal and whole-dimensional structures (see subsection 3.4.4). The algorithms were implemented using the THINK Pascal programming language (Symantec; Cupertino, CA) on a Macintosh LC III (Apple Computer; Cupertino, CA).

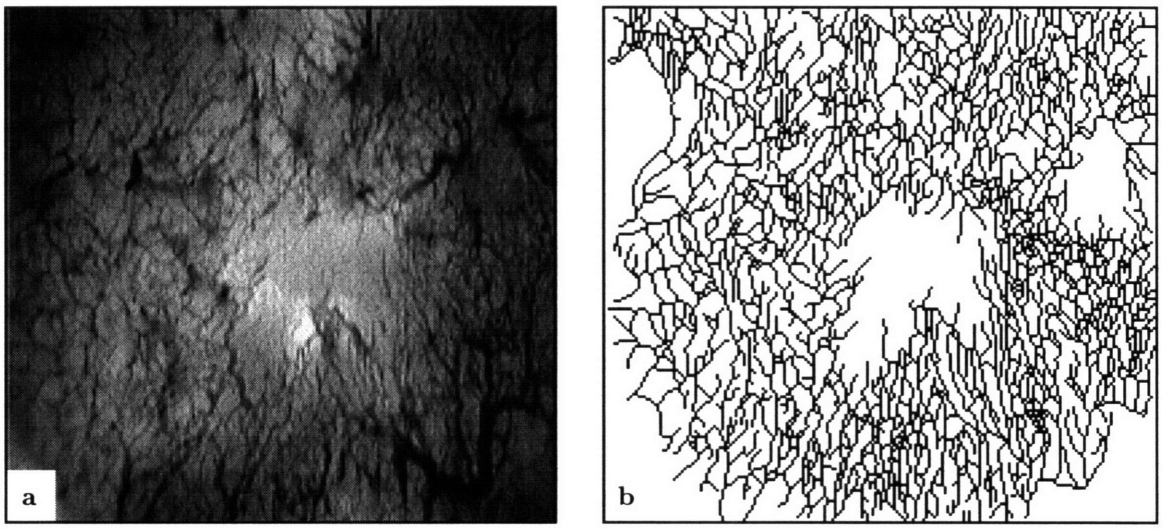

Figure 3-2: The image analysis process: **(a)** A raw grayscale image of LS174T tumor network in a nude mouse. **(b)** The subsequent binary skeletonized image. The latter image is the input for the fractal dimension calculation program.

3.4.1 Box-Counting Algorithm

In general, if F is a fractal subset of $\Re^n$ and $N_\delta(F)$ is the smallest number of hypercubes of size δ which can cover F, then the box dimension of F is defined as [27]:

$$d_{box} = \lim_{\delta \to 0} \frac{\log N_\delta(F)}{-\log \delta} \tag{3.1}$$

In practice this means that if one tiles a fractal object, embedded in two dimensions, with boxes of side ϵ and counts the minimum number of boxes $N(\epsilon)$ occupied by the object, a power law will be observed:

$$N(\epsilon) \propto \epsilon^D \tag{3.2}$$

over the range in which the object is scale-invariant. When a logarithmic plot of $N(\epsilon)$ vs. ϵ is obtained, the fractal dimension is then derived as the absolute value of the slope ($d_{box} = |D|$) of the straight line fit. The implementation of this algorithm iterated over all possible tiling configurations with an ϵ-mesh to find the configuration with minimum boxes. The source code used to implement the box-counting algorithm is listed in section A.2.

3.4.2 Sandbox Algorithm

One can select a pixel i on the fractal, surround it with boxes of increasing side ℓ and count the number of occupied pixels $M_i(\ell)$ in the box. If one repeats this procedure for n different points on

the fractal ($i = 1, \ldots, n$), one can obtain the mean number of occupied pixels in a box of side ℓ [14]:

$$M(\ell) = \frac{1}{n} \sum_{i=1}^{n} M_i(\ell) \tag{3.3}$$

The scale-invariant nature of the object implies that a power law will be observed:

$$M(\ell) \propto \ell^D \tag{3.4}$$

When a logarithmic plot of $M(\ell)$ vs. ℓ is obtained, the fractal dimension is then derived as the slope ($d_{sand} = D$) of the straight line fit. Formally speaking, this algorithm is similar to the box-counting algorithm only if there is no mass-multifractality. Otherwise, it is important to average over many points (Equation 3.3). For a single point, if there is mass-multifractality, then $d_{box} = D_0$ and $d_{sand} = D_{-\infty}$ (the indices 0 & $-\infty$ refer to the order of the generalized dimension [131]). In my implementation of the sandbox algorithm, I averaged over all points on the fractal that were within more than half the maximum box-length ℓ_{max} from the nearest border of the region-of-interest (ROI). The maximum box-length ℓ_{max} is defined as twice the distance from the center of mass to the nearest border of the ROI. This choice of points circumvented artifacts due to boundary effects. The source code used to implement the sandbox algorithm is listed in section A.3.

3.4.3 Correlation Algorithm

The two-point correlation function $c(r)$ of a fractal object of dimension d_f depends on r as [131]:

$$c(r) \sim r^{d_f - d_e} \tag{3.5}$$

where d_e is the Euclidean embedding dimension. Capitalizing on this property, the algorithm chose a pixel i on the object and counted the number of occupied pixels $M_i(r)$ within a spherical shell of radius r and width $0.1r$. The same calculation was repeated for n occupied pixels ($i = 1, \ldots, n$) and then normalized by n and by the total number of pixels (occupied and unoccupied) within each spherical shell $N_{pixels}(r \rightarrow 1.1r)$ to estimate the correlation function:

$$c(r) \sim \frac{1}{n \cdot N_{pixels}(r \rightarrow 1.1r)} \sum_{i=1}^{n} M_i(r) \tag{3.6}$$

Center pixels were chosen according to the same criterion as for the sandbox algorithm to avoid effects due to the boundary region. When a logarithmic plot of $c(r)$ vs. r is obtained the fractal dimension is derived from the slope D of the straight line fit: $d_{corr} = D + 2$. The source code used to implement the correlation algorithm is listed in section A.4.

Table 3.1: Verification of Fractal Dimension Algorithms

Structure	Theoretical Dimension	Box Dimension	Sandbox Dimension	Correlation Dimension
1-D Cantor Set	$\ln 2 \div \ln 3 = 0.63$	0.65 ± 0.02	0.64 ± 0.02	0.58 ± 0.25
2-D Cantor Set	$1 + (\ln 2 \div \ln 3) = 1.63$	1.63 ± 0.03	1.63 ± 0.02	1.63 ± 0.05
Koch Curve	$\ln 4 \div \ln 3 = 1.26$	1.28 ± 0.01	1.27 ± 0.00	1.24 ± 0.03
Sierpinski Gasket	$\ln 3 \div \ln 2 = 1.58$	1.60 ± 0.00	1.59 ± 0.00	1.56 ± 0.02
Sierpinski Carpet	$\ln 8 \div \ln 3 = 1.89$	1.87 ± 0.01	1.88 ± 0.00	1.87 ± 0.01
Straight Line	1.00	1.00 ± 0.00	1.00 ± 0.00	0.93 ± 0.08
Circle – Frame	1.00	1.01 ± 0.00	1.00 ± 0.00	1.01 ± 0.02
Circle – Filled	2.00	1.98 ± 0.00	2.00 ± 0.00	2.00 ± 0.00
Square – Frame	1.00	1.01 ± 0.00	1.02 ± 0.00	0.95 ± 0.04
Square – Filled	2.00	2.00 ± 0.00	2.00 ± 0.00	2.00 ± 0.00

3.4.4 Algorithm Verification

In order to verify that the source code implementing the three algorithms above is free of bugs, the algorithms were tested on a variety of fractal and whole-dimensional structures. The results of these tests are presented in Table 3.1. The results show that the box-counting and sandbox algorithms, as implemented in the source code listed in sections A.2 and A.3, are accurate and robust. The correlation algorithm, as implemented in the source code listed in section A.4 is less accurate than the other two in most of the cases.

3.4.5 Minimum Path

In addition to measuring the fractal dimension of the whole network, the fractal dimension of the minimum path d_{min} between two opposite sides of the network [126] was measured. In order to find the minimum path, all the points on one edge of the ROI were first marked and assigned a value of 1. Their nearest neighbors were then identified, and assigned a value of 2. The procedure continued in a similar fashion, marking the nearest neighbors in each subsequent iteration with an integer value incremented by 1. Once a value was assigned to a point, it could not be reassigned. The iterations continued until a point of the opposite edge of the ROI was reached. Then the program worked backwards from that point to find points with decreasing value, thus marking the minimum path. An example of a minimum path can be seen in Figure 3-3. The fractal dimension of this minimum path was measured by a modified box-counting algorithm. The "boxes" were rendered as straight lines, whose origin and end must fall on the minimum path. In this respect, the fractal dimension measurements were similar to the Richardson method used to measure coastlines [84]. In order to assure the accuracy of the results, the algorithm iterated over all possible configuration for covering the minimum path. The configuration utilizing the minimum number of lines was then chosen for the calculation. The calculation was performed as described in subsection 3.4.1. The source code used to identify the minimum path and calculate its fractal dimension is listed in section A.5.

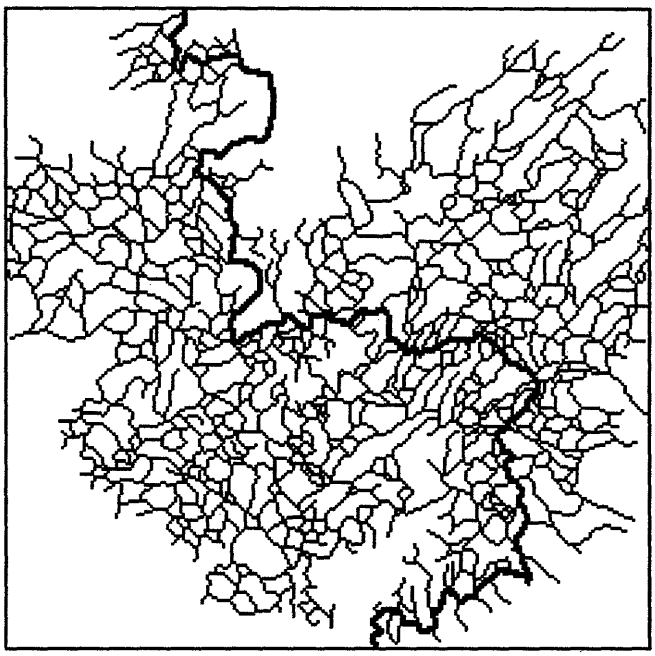

Figure 3-3: An example of the minimum path in tumor vasculature. The vascular network shown is from SCC7 tumor implanted in C3H mouse. The minimum path appears in bold. The fractal dimension of this minimum path is 1.16 ± 0.01.

3.5 Statistical Significance Tests

In cases where the statistical significance of the difference between two unpaired groups of observations needed to be measured, two different unpaired comparison tests were used:

- The unpaired t-test, which compares the means of two groups and determines the likelihood of the observed difference occurring by chance, under the hypothesis that the means of the two groups are equal. This likelihood is reported as a p-value. This test assumes that the observations in each group are normally distributed.

- The Mann-Whitney U-test, which is the nonparametric version of the unpaired t-test. The Mann-Whitney U-test does not make any assumptions about the underlying distribution of the observations in the two groups. It tests the hypothesis that the distributions underlying the two groups are the same. The result is also reported as p-value.

3.6 Computer Modeling

Computer modeling of tumor and normal capillary growth was performed using two different platforms. Initial modeling efforts were carried out using the THINK Pascal programming language (Symantec; Cupertino, CA) on a Macintosh LC III (Apple Computer; Cupertino, CA). This programming environment offered a user-friendly interface which provided instant visual feedback about

the model results, albeit at the expense of computing time. Once the model was verified to be bug-free (to a satisfactory degree of certainty), it was reprogrammed using the FORTRAN 77 programming language on a much faster UNIX platform.

The random number generator used in the modeling was of the congruential multiplicative type:

$$x_i \equiv cx_{i-1} \bmod (2^{31} - 1) \tag{3.7}$$

with a multiplier $c = 950706375$ and with shuffling. This specific generator stood up to extensive empirical tests [32]. It was employed via the FORTRAN routine package IMSL STAT/LIBRARY (IMSL; Houston, TX).

Computer modeling of arteriovenous network growth was performed using the MATLAB (MathWorks; Natick, MA) matrix manipulation software on various UNIX platforms provided by MIT's Project Athena.

3.7 Plotting

The plots in this document were generated using the public domain plotting package Gnuplot. The interpolations were generated with the MATLAB package using a cubic smoothing spline. The smoothing parameter s_p was in the range [0.999,1]. For $s_p = 0$, one obtains the least-squares straight line fit. On the other extreme, for $s_p = 1$, one obtains the "natural" cubic spline interpolant.

Chapter 4

Fractal Dimensions of Vascular Networks

Fractal dimensions were measured in a variety of normal and tumor vascular networks grown in the two-dimensional murine dorsal chamber preparation described in section 3.1. Results of these measurements (using the box-counting and sandbox algorithms) are summarized in Figure 4-1. A skeletonized image typical of each class of vascular networks discussed below is shown in Figure 4-2. Results of measurements using the correlation algorithm are listed separately in Appendix B, since benchmark tests showed that this algorithm is less accurate than either the box-counting or sandbox algorithms (see subsection 3.4.4).

4.1 Normal Networks

Normal networks can be broadly divided into two categories: arteriovenous networks and capillary networks. Arteries and veins are different from capillaries in several ways. Histologically, arteries and

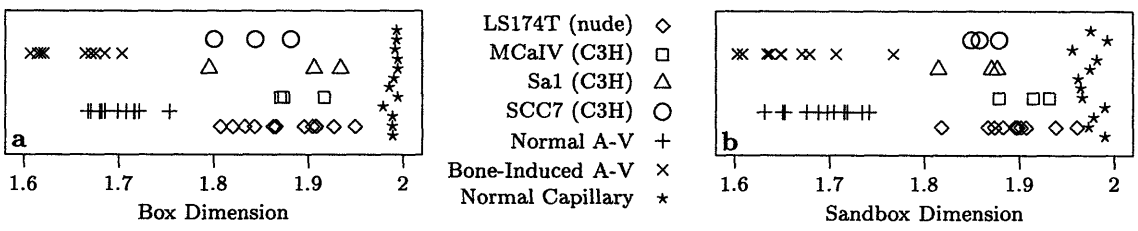

Figure 4-1: Fractal dimensions of the observed vascular networks. **(a)** As measured with the box-counting algorithm. **(b)** As measured with the sandbox algorithm.

veins contain smooth muscle cells and an adventitial layer, whereas capillaries do not [63]. Geometrically, arteries and veins are of consistently larger diameter than capillaries. Topologically, arteries and veins usually form a tree-like structure, whereas capillaries form a mesh-like structure [106]. The dorsal window provides a convenient preparation in which the arteriovenous and capillary networks can be easily differentiated since the arteriovenous network resides in a focal plane different from the striated skin muscle capillary network connected to it.

4.1.1 Arteriovenous Networks

Measurements of normal subcutaneous arteriovenous networks yielded values of $d_{box}=1.70\pm0.03$ and $d_{sand}=1.70\pm0.03$ in nude mice ($n = 12$); and $d_{box}=1.72\pm0.02$ and $d_{sand}=1.71\pm0.03$ in SCID mice ($n = 23$). The minimum-path dimension measured was $d_{min}=0.99\pm0.02$ in nude mice and $d_{min}=1.00\pm0.01$ in SCID mice. The normal subcutaneous arteriovenous networks were quiescent and were not growing at the time of measurement. In order to verify these results in a growing normal vascular network, the fractal dimension in bone-induced arteriovenous networks [77] was measured. In this preparation, the networks grew on a topologically two-dimensional surface (the bone surface). However, since this surface is not two-dimensional in the Euclidean sense, the field of view in which the network was in focus was more limited than in the standard dorsal window. Measurements of bone-induced arteriovenous networks ($n = 10$) yielded $d_{box}=1.65\pm0.04$ and $d_{sand}=1.66\pm0.05$. The minimum-path dimension was $d_{min}=1.00\pm0.01$.

4.1.2 Capillary Networks

All capillary networks were imaged with the aid of a fluorescent contrast material (see section 3.2), since the relatively low number of red blood cells in each of these thin vessels could not provide enough contrast. Only intact quiescent networks where the vessels were not leaky could be imaged with this method. Measurements of normal striated skin muscle capillary networks in nude mice ($n = 12$) yielded $d_{box}=1.99\pm0.00$ and $d_{sand}=1.97\pm0.01$. The minimum-path dimension was $d_{min}=1.00\pm0.02$.

4.2 Tumor Vascular Networks

The vascularization of tumors is a dynamic process. In the first stages of vascularization, the new vessels are not fully formed. This condition produces microhemorrhages in the tumor, which make the observation of the new vasculature virtually impossible [78]. In addition, the rate of vascularization may vary significantly between tumors. Taking into account that tumor size in the vascular phase of tumor growth is directly dependent on the level of vascularization, it was decided to study tumors of comparable sizes (approximately 4 mm in diameter). Tumors from the same cell line

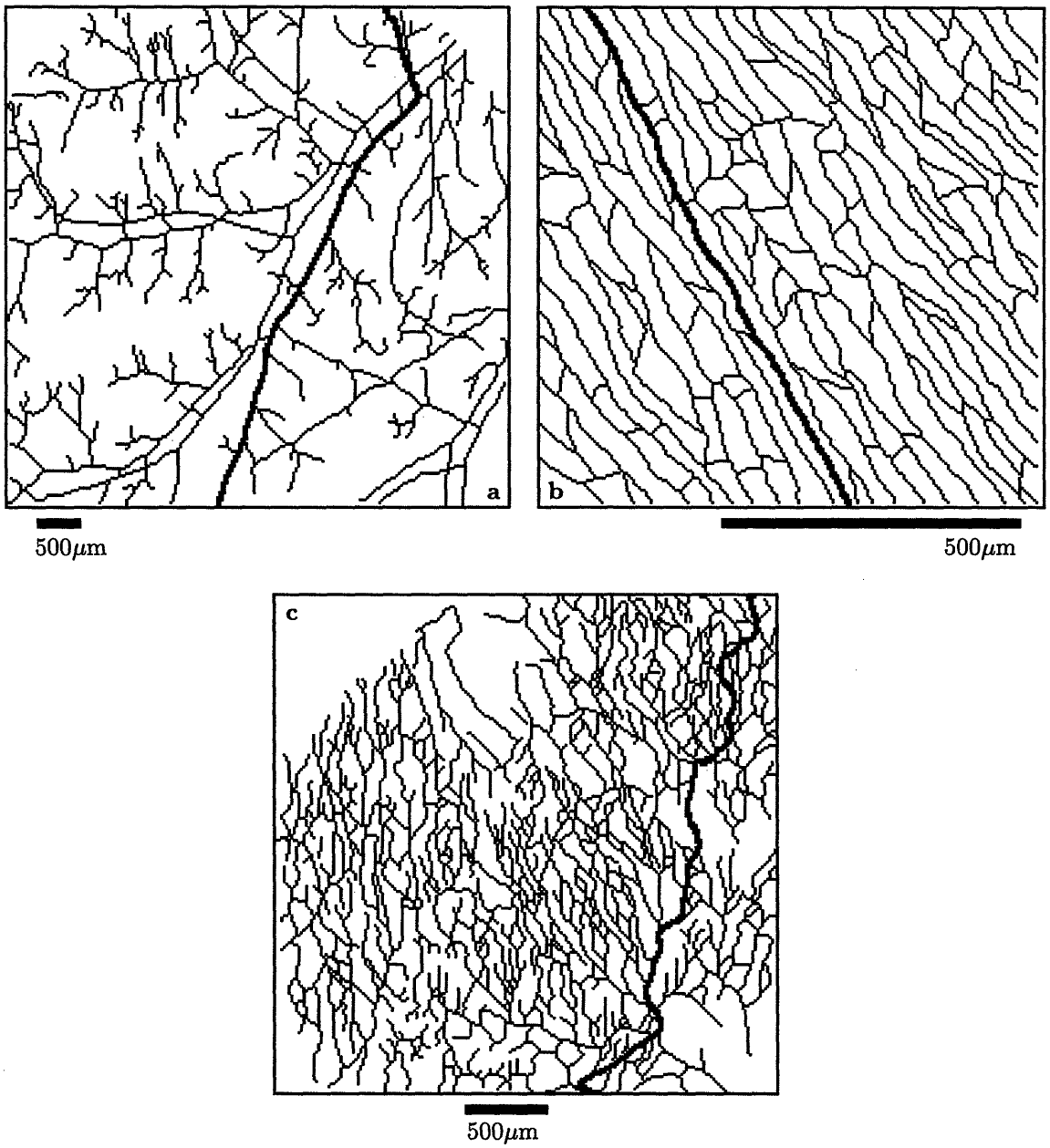

Figure 4-2: Typical skeletonized images of the three observed classes of vascular networks. **(a)** Normal subcutaneous arteriovenous network in a SCID mouse; **(b)** Normal subcutaneous capillary network in a nude mouse; **(c)** LS174T tumor network in a nude mouse. The minimum path is in bold.

Table 4.1: Fractal Dimensions of LS174T Tumor Networks Over Time

Dimension	day 10	day 14	day 18	day 22
Box-Counting	1.69 ± 0.07	1.81 ± 0.05	1.83 ± 0.02	1.84 ± 0.04
Sandbox	1.72 ± 0.06	1.84 ± 0.03	1.85 ± 0.02	1.85 ± 0.03
Minimum Path	1.10 ± 0.03	1.08 ± 0.02	1.09 ± 0.02	1.09 ± 0.03

reached this size in approximately the same time[1]. In a separate study, one type of tumor (LS174T implanted in SCID mice) was followed for a period of 12 days, measuring the fractal dimensions every 4 days. The duration of the experiment was limited by the viability of the dorsal chamber.

In tumors, unlike normal tissue, arteries, veins and capillaries cannot be distinctly classified [60]. The vessels' geometric parameters can vary dramatically, regardless of topological parameters or histological characteristics [68]. Therefore, in tumors the fractal dimension was measured for all observed vessels taken together.

4.2.1 Tumors of Fixed Size

Measurements of human adenocarcinoma LS174T vascular networks ($n = 12$) in nude mice yielded $d_{box} = 1.88 \pm 0.04$ and $d_{sand} = 1.89 \pm 0.04$. The minimum-path dimension was $d_{min} = 1.10 \pm 0.04$. To test the generalizability of these results, fractal dimensions were measured in C3H mice implanted with three different murine tumor cell lines ($n = 3$ for each). The results of these measurements are summarized in Figure 4-1. One can see that the additional results are consistent with the results for the LS174T tumor networks.

4.2.2 Tumor Networks Over Time

Fractal dimension measurements of human adenocarcinoma LS174T vascular networks ($n = 7$) were performed on images acquired on days 10, 14, 18 and 22 after tumor cell inoculation (only on day 10 does the optical quality of the tumor become sufficient for these measurements). The measurements are summarized in Table 4.1 and presented graphically in Figure 4-3.

These results show that approximately on day 14 after inoculation the fractal dimension of the whole network reaches a "steady-state" value. The reason for the initial period of rising fractal dimension lies in the fact the the inoculated tumor cell mass lacks any vessels, so only around day 14 does the tumor become "fully" vascularized (afterwards vascularization follows tumor growth). This explanation is consistent with the observation that tumor vascular density in the same animals also reached a "steady-state" value at approximately day 14 [78]. The mean fractal dimension values in Table 4.1 are slightly lower than the mean values reported in LS174T tumor networks in nude mice (see subsection 4.2.1). This is most likely due to the lower quality of images acquired using the

[1]Even if the tissue environment was exactly the same in all mice, there would be inevitable variations due to the inaccuracy in measuring the initial amount of tumor cell inoculate.

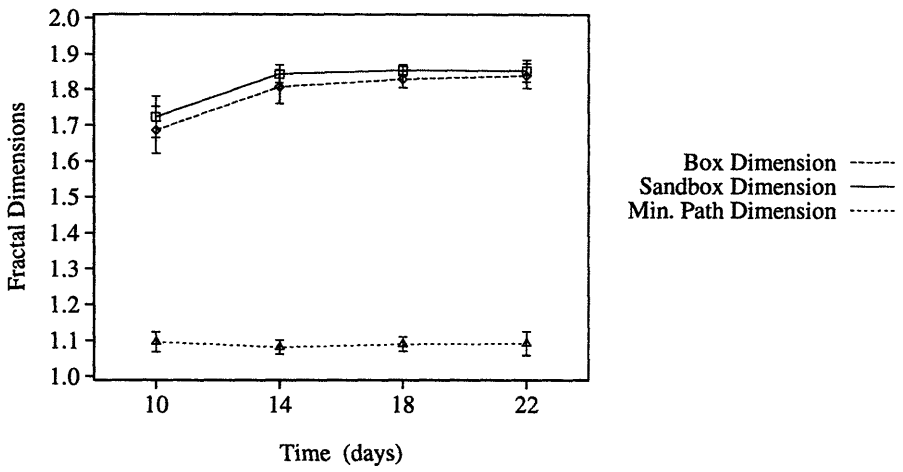

Figure 4-3: Fractal dimensions of LS174T tumor networks over time. Each point represents the mean and standard deviation of measurements in 7 different animals.

analog image acquisition system neccessary for following the same animal over time (see section 3.2).

The minimum-path dimension, however, remained statistically unchanged during the whole measurement period, and it was nearly identical to the minimum path dimension measured in tumor networks in nude mice (see subsection 4.2.1). Since the minimum-path dimension does not reflect the degree of overall vascularization but rather the scale-invariant tortuousity of single vascular paths, the constant minimum-path dimension seems to imply that the determinant of the elevated minimum-path dimension (e.g., substrate heterogeneity — see Chapter 6) remained largely unchanged throughout the measurement period.

As this thesis was nearing completion, a regressing tumor model (Shionogi mammary carcinoma in SCID mice) was being developed in our lab. Preliminary results of fractal dimensions during tumor regression ($n = 2$) are described in Appendix C.

4.3 Verification of Fractal Range

As detailed in section 3.4 the fractal dimension is measured by fitting a straight line to a logarithmic plot of a measure of mass[2] as a function of ruler size ($\ln r$). However, in nature there is only a limited range over which this linear relation holds. Figure 4-4a shows a typical logarithmic plot of measurement results obtained by the box-counting algorithm. Two gross observations can be made:

- For large r we see that $N(r)$ is constant, due to the finite size of the image.
- For small r we see that the slope $\frac{dN(r)}{dr}$ is no longer constant and gradually decreases. This is due to finite vessel-to-vessel distances (the average distance between capillaries is rarely below 50 μm).

[2]This can be a direct measure of mass as in the sandbox method ($\ln M_r$), or an indirect measure as in the box-counting method ($\ln N_{box}$).

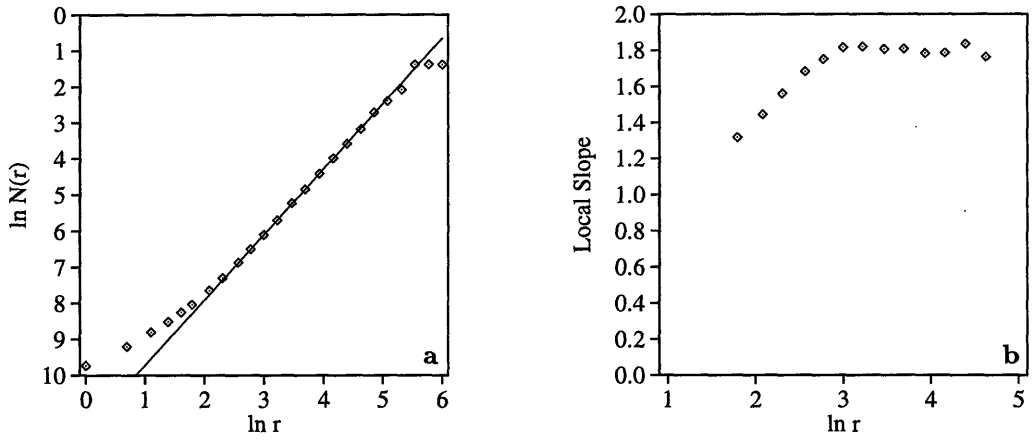

Figure 4-4: The plots used for determination of fractal dimension and range. **(a)** Logarithmic plot of number of occupied boxes ($\ln N(r)$) as a function of box-size ($\ln r$) and its linear fit; **(b)** Plot of the local slope as a function of box-size ($\ln r$). The local slope was calculated for each series of five consecutive points. These plots were derived from analysis of the SCC7 vascular network shown in Figure 3-3. The slope of the linear fit yields a box-counting dimension of $d_{box} = 1.81 \pm 0.01$.

These observations underscore the importance of verifying the fractal range (i.e., the linear range of the logarithmic plot).

The fractal range can be verified by plotting the local slope of the plot against the ruler size ($\ln r$), as shown in Figure 4-4b. The local slope can be calculated by fitting two or more consecutive points to a straight line. In Figure 4-4b five consecutive points were used to calculate the local slope. The flat part of the plot delineates the fractal range. Furthermore, this plot is a useful tool for uncovering slow crossover phenomena[3].

In the vascular networks observed, fractal ranges were invariably limited by capillary-to-capillary distances, giving a lower cutoff of approximately 50 μm. For normal capillary networks the theoretical upper cutoff would be the macroscopic length scale of the tissue, since capillary networks must be space-filling. In practice, this upper cutoff is determined by the image size.

For normal arteriovenous networks, one would expect a crossover from DLA-like scaling to compact scaling, because maximal artery-vein distances are determined by the distance that blood can travel through the capillary network before it is depleted of oxygen. This distance is on the order of several millimeters in skeletal muscle [105, 123], similar to the size of the whole image. Hence, no crossover was observed in the arteriovenous data.

For tumor networks, there is no good intuition for the upper bound of the fractal range. The measurements show that tumors are fractal below approximately 900 μm, yet no crossover phenomena was observed for tumors, because above 900 μm the finite image size introduces non-negligible

[3]For example, if the plot would never flatten, but would slowly slope upwards toward a value of 2.0, we could say that there is crossover to a compact structure. However, such crossover behavior was not observed.

artifacts into the results.

4.4 Statistical Significance of Results

The separation of vascular networks into three classes of scale invariant behavior is clearly evident in Figure 4-1. Nevertheless, this separation should be formalized using appropriate unpaired statistical comparison tests. Two unpaired comparison tests were used for significance testing: the unpaired t-test and the Mann-Whitney U-test (described in section 3.5).

4.4.1 Between and Within-Class Separation

The data from Figure 4-1 were divided into five pairs of groups. The first three pairs were used to measure between-class separation and consisted of all possible pairings of the groups:

- Arteriovenous networks — normal subcutaneous and bone-induced networks in nude mice ($n = 22$).
- Tumor networks — tumor networks in nude and C3H mice ($n = 21$).
- Capillary networks — normal capillary networks in nude mice ($n = 12$).

The last two pairs were used to measure within-class separation within the artriovenous network class and within the tumor network class:

- Bone-induced arteriovenous networks in nude mice ($n = 10$) vs. subcutaneous arteriovenous networks in nude mice ($n = 12$).
- LS174T tumor networks in nude mice ($n = 12$) vs. various tumor networks in C3H mice ($n = 9$).

All tests were performed for three different fractal dimensions: box-counting dimension, sandbox dimension, and minimum-path dimension. The results are summarized in Table 4.2.

Between-Class Separation

The p-values in Table 4.2 show that both the box-counting dimension and the sandbox dimension clearly separate the three classes of two-dimensional vascular networks observed: normal arteriovenous networks, normal capillary networks, and tumor networks. The minimum-path dimension is useful in distinguishing tumor networks from normal networks. Normal arteriovenous and capillary networks, however, have statistically similar minimum-path dimensions.

Table 4.2: Vascular Network Class Separation Based on Fractal Measurements

Network Classes	Box-Counting		Sandbox		Minimum-Path	
	M-W	t-test	M-W	t-test	M-W	t-test
Tumor vs. Capillary	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$
Tumor vs. A-V	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$
A-V vs. Capillary	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p = 0.8854$	$p = 0.9681$
Bone A-V vs. S.C. A-V	$p = 0.0240$	$p = 0.0006$	$p = 0.0479$	$p = 0.0788$	$p = 0.6444$	$p = 0.7128$
Tumor (nude) vs. Tumor (C3H)	$p = 0.8312$	$p = 0.6966$	$p = 0.1356$	$p = 0.2372$	$p = 0.1769$	$p = 0.1443$

Within-Class Separation

The p-values in Table 4.2 show that the box-counting dimensions of bone-induced and subcutaneous arteriovenous networks are not statistically similar, and that their sandbox dimensions are marginally similar. The mean differences of 0.05 for d_{box} and 0.03 for d_{sand}, are most likely due to the difficulty of keeping all arteries and veins on the curved bone surface in focus (see subsection 4.1.1). The minimum-path dimensions of these two types of arteriovenous networks are statistically similar.

Fractal dimensions for tumor networks in nude mice (LS174T tumors) and tumor networks in C3H mice (MCaIV, SCC7 and Sa1 tumors) are statistically similar. This observation suggests that the fractal dimensions for two-dimensional tumor networks are relatively independent of host and tumor type.

4.4.2 Effect of Imaging Method

A separate series of measurements was performed in normal arteriovenous networks in SCID mice. The imaged networks were divided into two groups. The first group ($n = 12$) was imaged using transmitted light microscopy, and the second group ($n = 11$) was imaged using fluorescence microscopy (see section 3.2). The fractal dimensions of the two groups were compared statistically. The comparison results appear in Table 4.3. These results show that there are no significant differences in fractal dimensions due to the use of transmitted light or fluorescence microscopy[4].

[4]It should be noted that measurements of vessel diameter are known to be sensitive to imaging method [81]. These differences are presumably due to the plasmatic zone between erythrocytes and the vessel wall, and hence should have no effect on fractal dimension measurements.

Table 4.3: Effect of Imaging Method on Fractal Dimension

Imaging Methods	Box-Counting		Sandbox		Minimum-Path	
	M-W	t-test	M-W	t-test	M-W	t-test
Trans. Light vs. Fluorescence	$p = 0.9999$	$p = 0.8546$	$p = 0.2423$	$p = 0.4690$	$p = 0.1962$	$p = 0.1028$

4.5 Discussion

The separation of vascular networks into three classes based on fractal measurements has important implications, because these classes correspond to different statistical growth processes (see section 2.3).

4.5.1 Normal Vascular Networks

The measurements in normal arteriovenous networks are in concert with the fractal dimensions of two-dimensional diffusion-limited aggregates ($d_f = 1.71$ and $d_{min} = 1.0$) [91, 131]. These results agree with previously published fractal measurements of retinal arterial and venous networks[5] [28, 72, 82], the retinal vasculature being a routinely observed naturally occurring two-dimensional network.

On the face of it, these results seem consistent with the accepted view of the angiogenic process, where growth factors initially diffuse from low-nutrient or hypoxic regions and induce growth. Under such circumstances, the diffusion-limited behavior of the arteriovenous network is quite understandable. This explanation, however, is fatally flawed because angiogenesis does not seem to occur on the artery-vein level. Rather, vascular growth occurs at the capillary or post-capillary level [20, 33, 88].

This vascular growth (on the capillary and post-capillary level) does not, however, form diffusion-limited structures. Neovascularization studies [2, 20, 88] have shown that newly formed capillaries grow in a compact mesh and not in a tree-like structure, contrary to the expected structure in diffusion-limited growth. Our measurements corroborated these observations by showing that the normal subcutaneous striated muscle capillary bed is also a compact structure. These observations

[5]There is one account [23] which claims that retinal vessels display percolation-like scaling. That account, however, was not peer-reviewed and used only the correlation method for measuring fractal dimension. Another account [72] claimed that at high resolution (small r) there is a crossover to another fractal domain with $d_f \approx 1.2$. This is most likely due to a misinterpretation of the logarithmic plot (such as the one in Figure 4-4a). This misinterpretation can be avoided by examining the local slope, as shown in Figure 4-4b. The authors of that particular study did not take this extra precaution.

In a paper on the modeling of arterial networks as minimum energy dissipation vascular systems [128], it was claimed that arterial networks are space-filling, and that fractal dimensions of $d_f < 2$ are a crossover effect. This claim was based on the assumption that distances between the ends of adjacent terminal arterioles are equal for all terminal arterioles in a network. Experimental evidence indicates that this assumption is clearly wrong [105]. The distance between the ends of adjacent terminal arterioles is bounded from above by the distances which erythrocytes can travel before their oxygen is depleted in the capillary network (typically several millimeters), and from below by the capillary spacing (typically 50 μm). My measurements were all carried in the range bounded by those values. Clearly, on the whole-organ scale the arteriovenous network can be viewed as space-filling.

are also supported by many published images of normal capillary beds including lung capillary bed [83], colon capillary bed [124], ocular capillary beds [37, 70, 112], cardiac and skeletal muscle capillary beds [107], and the chorioallantoic capillary bed [16].

Clearly, we need to explain what causes the capillary networks to attain a compact structure. How can we settle this apparent contradiction between the commonly accepted hypothesis that growth factor diffusion initiates the angiogenic process and the observation that newly growing capillaries form a compact structure? This issue will be dealt with in Chapter 5.

4.5.2 Tumor Vascular Networks

The fractal measurements in tumor vasculature show that the tumor vessels do not form a compact structure, but are consistent with measurements for the critical percolation cluster ($d_f = 1.896$ and $d_{min} = 1.13$) [50, 127]. This observation represents the first evidence for a biological growth process whose key determinants are local properties. Furthermore, measurements of fractal dimension over time indicate that the percolation-like nature of tumor vascular networks is not a transient phenomenon. Tumor vascular networks, like percolation clusters, are characterized by loops and avascular areas of many different length scales (see Figures 3-2, 3-3, 4-2c, and References [1, 68, 76]). Percolation being a local growth process [131] suggests that there exists some local substrate property which determines tumor capillary growth. Three biological observations suggest that this local property can be hypothesized to be matrix inhomogeneity. First, tumor tissue does not possess the regular periodic geometry formed by cells in normal tissue, and has different phenotypic subpopulations. Second, extracellular matrix (ECM) is a larger and more heterogeneous component of tumor tissue compared to normal tissue [49, 59, 99, 129]. Third, "blood channels", blood pathways through tissue not lined by endothelium, have been observed in tumors [60]. Such channels can be viewed as evidence of percolation phenomena through the ECM. This hypothesis and its compatibility with the normal vascular growth model will be explored in Chapter 6.

Chapter 5

Modeling Normal Capillary Network Growth

In subsection 4.5.1 I detailed the fundamental problem in modeling normal capillary network growth. The problem lies in the apparent contradiction between the commonly accepted hypothesis that growth factor diffusion initiates the angiogenic process and the observation that the newly growing capillaries form a compact structure. While diffusion is inevitably a component in the process of vascular growth (see Figure 2-1), a compact structure is no longer diffusion-limited because the diffusion field is "masked", so the growth becomes insensitive to variations in the diffusion gradient.

We therefore need a new mechanism that causes growth factor concentration to increase throughout the growth perimeter, thus "masking" the diffusion field and promoting uniform growth. Conceptually, the source of growth factors near the growth perimeter could either be the hypoxic tissue [93, 121] or the growing structure itself. If the source is solely the hypoxic tissue, high growth factor concentrations near the growth perimeter could be achieved if the rate of growth factor reception or removal at the growth perimeter is slow compared to the diffusion rate. In biological terms, this can be achieved by the existence of low-affinity receptors, for example. In physical terms, it is tantamount to a *low interaction probability* between growth factors and the growing structure. If, however, the main source of growth factors near the growth perimeter is the growing structure itself, then at each growth site we have a *local amplification* of growth factor which then propagates to neighboring sites. In biological terms, local amplification of growth factor levels can be achieved by autocrine or paracrine release of growth factors [34, 35], or by the release of growth factors sequestered in the endothelial basement membrane [21, 35]. In physical terms, it is tantamount to a process where a growth event is accompanied by a uniform increase of growth probability in sites neighboring the growth site.

The above hypotheses for capillary network formation were compared using a growth model.

5.1 Basic Growth Model

The basic growth model incorporated diffusion of growth factor and vascular growth in response to growth factor reception, in the presence of either a low interaction probability (i.e., the "low affinity hypothesis") or local amplification (i.e., the "autocrine hypothesis"). The growth model was implemented according to the following rules:

- Growth begins at a single central seed.
- Growth factor "particles" diffuse from points removed at least some minimum distance from the structure.
- When a "particle" hits the growing structure it is taken up with a preset probability p_i.
- All uptaken particles are recorded as "hits" within a fixed time period.
- At the end of the period growth occurs at all hit sites.
- If there is local amplification, F additional particles are released at each growth site. F is the "local amplification factor".
- There is a fixed growth factor production rate per unit area of "hypoxic" tissue.
- The model is implemented on a 128×128 square lattice with periodic boundary conditions, unless otherwise noted.

The source code for the model, including the parameters used in all later enhancements of the model, is listed in section A.6.

5.2 Local Amplification vs. Low Interaction Probability

The model presented in section 5.1 was used to examine whether the local amplification mechanism or the low interaction probability mechanism could account for the compact structure of normal capillary networks. The results are shown in Figure 5-1. One can observe that for $p_i = 1$ and no local amplification the classic diffusion-limited structure is obtained. As p_i is decreased the structure becomes more compact and regular. When local amplification is activated, a compact structure is obtained for $F > 1$. We can therefore conclude that either of these mechanisms, if strong enough, can lead to the formation of a compact capillary network.

We can assess which of the two mechanisms is a more likely determinant of the compact structure by examining two parameters as a function of fractal dimension, as seen in Figure 5-2 (the closer the fractal dimension is to 2.0, the closer the model network is to being compact).

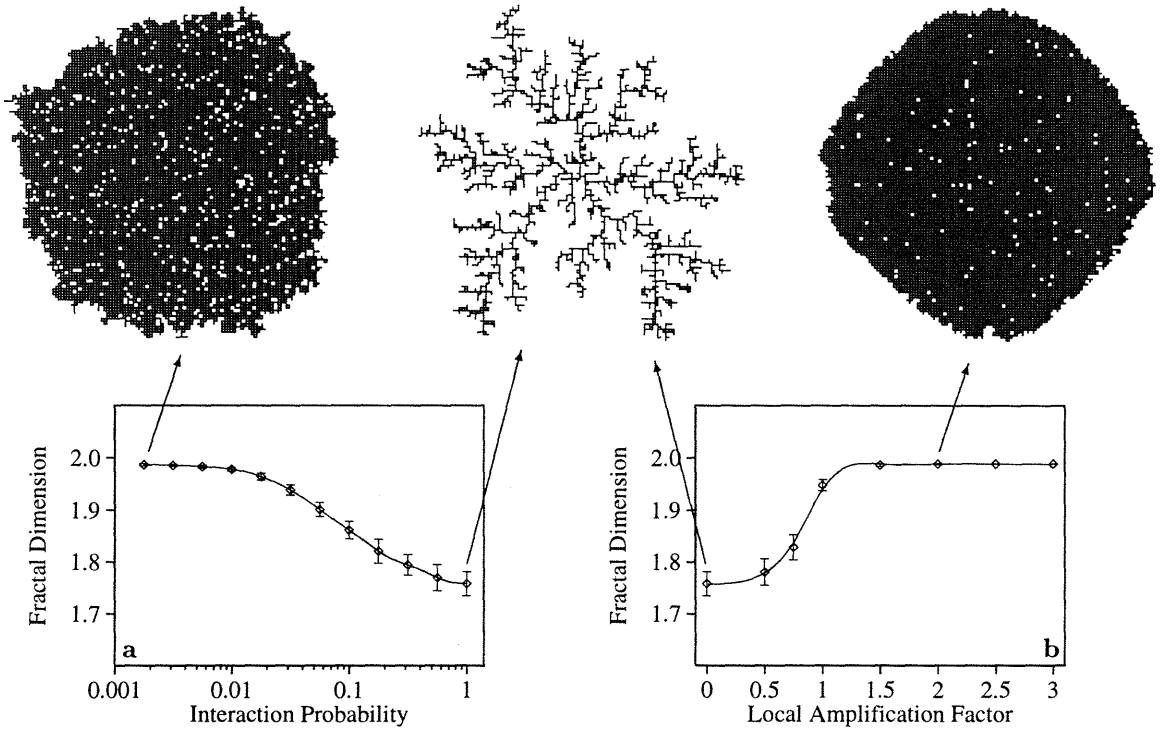

Figure 5-1: Effects of **(a)** low interaction probability and **(b)** local amplification on fractal dimension.

The first parameter, the growth time per unit mass, is obtained by dividing the total growth time (in arbitrary model time units) by the number of occupied lattice sites in the final structure. It is motivated by the reasoning that there is an advantage in growing blood vessels relatively fast towards the hypoxic region.

The second parameter, the growth efficiency, is motivated by the reasoning that in order for growth to be efficient, the hypoxic tissue should produce as little growth factor as possible to induce the formation of new capillaries. The growth efficiency is obtained by dividing the total number of growth factor particles originating beyond the network by the number of occupied lattice sites in the final structure.

Although at first thought the two parameters may seem to work in opposite directions, they do not. Figure 5-2a shows that when the interaction probability is lowered, growth time per unit mass does not change significantly; but when local amplification is incorporated, the growth time per unit mass for a compact structure decreases by an order of magnitude. Figure 5-2b shows that with local amplification, compact growth efficiency increases by more than an order of magnitude, compared to no significant change in growth efficiency when low interaction probability is incorporated.

Another approach to comparing the two mechanisms is to measure the percentage of unoccupied lattice sites within the structure perimeter. This value is an indicator of the number of poorly vascularized areas which a given mechanism would create. For compact structures with similar

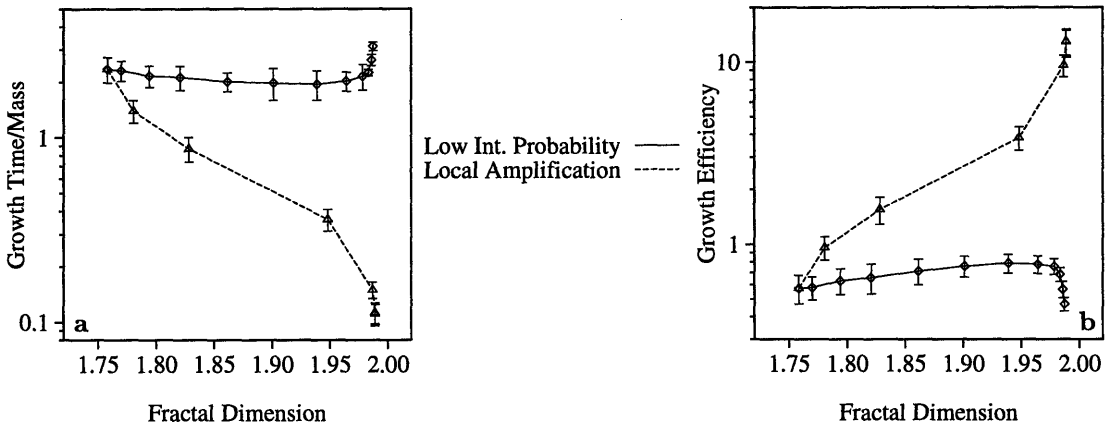

Figure 5-2: Comparison of the local amplification and the low interaction probability mechanisms: **(a)** growth time per unit mass and **(b)** growth efficiency. Clearly, growth with the local amplification mechanism is both faster and more efficient as the fractal dimension nears 2.0.

fractal dimensions ($d_f \approx 1.99$), the percentage of lattice sites left unoccupied with the low interaction probability is approximately 5%, and with the local amplification mechanism less than 1%.

In view of these results, it seems reasonable to suggest that local amplification, corresponding to the biological autocrine mechanisms of growth factor release, is the key determinant of the observed compact shape of normal capillary networks.

This suggestion is corroborated by several pieces of experimental evidence. A recent study of rabbit corneal endothelial cells [69], provided evidence showing the existence of an autocrine growth mechanism utilizing PDGF. A previous study showed that capillary endothelial cells express bFGF, and raised the possibility that the formation of new capillaries is induced by the endothelial cells themselves [113]. Furthermore, a study of low-affinity bFGF binding sites in a variety of cells ranging from baby hamster kidney cells to bovine capillary endothelial cells [96], showed that these binding sites are most likely cell-associated heparin-like molecules and that only high-affinity binding sites stimulated plasminogen activator production by bovine capillary endothelial cells, thus suggesting that only the high-affinity binding sites are receptors mediating growth stimulus.

In an attempt to corroborate experimentally the existence of an autocrine mechanism for endothelial cell growth, we performed a series of in vitro experiments. In these experiments bFGF was administered to endothelial cells in a culture dish either in a single dose, which created a uniform concentration in the dish, or by controlled release, which presumably created a gradient in the dish. If an autocrine mechanism was at work, one would expect the growth patterns in both cases to look similar and have little directional dependence. Indeed, this was the outcome (see Appendix D for details). Recently, however, a much more elegant in vitro experiment was published [48], which showed the existence of an autocrine mechanism for angiogenesis. In that experiment, mouse aortic endothelial cells were stably transfected with a retroviral expression vector harboring a human

bFGF cDNA. The transfected cells showed marked sprouting activity in vitro, forming networks of cord-like structures. Remarkably, this activity was suppressed by administration of anti-bFGF antibodies, thus revealing the autocrine nature of the angiogenic process [48].

5.3 Model Units and Reality

The model proposed in section 5.1 seems to do a good job in explaining the observed shape of capillary networks. However, it still remains to be proven that model parameters and results are consistent with physiological behavior when translated into "real-life" units. The following calculations are presented for this purpose.

The lattice constant a of the model lattice, representing the minimal vessel-to-vessel distance, can be reasonably assumed to be in the lower range of naturally observed capillary-to-capillary distances [123]:

$$a \approx 50\mu\text{m} \tag{5.1}$$

The diffusion constant in tissue of a growth factor such as VEGF (molecular weight ≈ 45000) can be approximated as [12]:

$$D \approx 5 \cdot 10^{-7}\,\text{cm}^2/\text{sec} \tag{5.2}$$

If the length unit is taken to be the lattice constant a, then D becomes:

$$D \approx 2 \cdot 10^{-2}\,a^2/\text{sec} \tag{5.3}$$

Therefore, the time it takes a particle to "explore" a unit lattice cell of area a^2 is on the order of:

$$\sqrt{\frac{a^2}{D}} \approx 50\,\text{secs} \tag{5.4}$$

In the model, 1 unit of model-time (umt) equals the time it takes to advance a particle one lattice constant a. In real units. this would therefore be on the same order as the time in Equation 5.4. Hence:

$$1\,\text{umt} \approx 50\,\text{secs} \tag{5.5}$$

The model lattice is approximately 100×100 units, or in real lengths (Equation 5.1): $5 \times 5\,\text{mm}^2$. The time t_0 it takes to grow a vascular network of such dimensions can be reasonably estimated as [77, 78, 119, 134]:

$$t_0 \approx 10^6\,\text{secs} \tag{5.6}$$

which is equivalent to 11.5 days. In a compact structure (as capillary networks are), virtually every

site is occupied. Therefore, the growth time per unit mass (or area), in real time, would be:

$$g_{real} \approx \frac{t_0}{100 \times 100} = 10\,\text{secs}/a^2 \tag{5.7}$$

In Figure 5-2a we saw that the model yielded a value of:

$$g_{model} \approx 0.1\,\text{umt}/a^2 \tag{5.8}$$

Are these two values comparable? Indeed they are. In Equation 5.5 we defined 1 umt as approximately 50 secs. Therefore:

$$g_{model} \approx 5\,\text{secs}/a^2 \tag{5.9}$$

which is of the same magnitude as g_{real} (Equation 5.7). Keeping in mind the approximations made in the process, it is notable that the two values for g are so close.

In summary, we see that the model and its parameter estimates produce results which are in concordance with physiological values. This concord lends further credence to conclusions drawn from the model.

5.4 Model Evaluation

The model offered in this chapter rests on two pillars. First, it attempts to reproduce the observed scale-invariant properties of capillary networks. Second, it is based on accepted physiological phenomena, such as growth factor diffusion and autocrine growth factor release mechanisms. Most previously reported models have failed to incorporate either one or both of these elements [9].

Virtually all previously proposed models have ignored the apparent paradox between the compact nature of the capillary network and the supposedly key role of growth factor diffusion in angiogenesis [9]. Some models [47, 71] have gone as far as assuming that capillaries grow in a tree-like branching network, an assumption that is clearly contradicted by most experimental evidence. Arterial and venous networks indeed display a morphology reminiscent of diffusion-limited processes, but these networks form by the remodeling of existing vessels and not by formation of new ones (see Chapter 7).

The model presented in this chapter does not assume a specific mechanism of capillary tube formation. Some models have been based on a particular mechanism, such as sprouting [8], although there is another mechanism — intussusception — which may be involved in capillary tube formation [15, 103]. The model proposed here is equally valid when either sprouting or intussusception is assumed as the mechanism of capillary tube formation.

This model is also unique in offering a clear distinction between normal and tumor capillary

network formation (see Chapter 6). Other models tended to blur this distinction [8]. This was due to the fact that until the present study, the unique scale-invariant properties of tumor networks were not known, and there seemed to be no need for different growth models for normal and tumor capillary networks. This model allows, for the first time, to differentiate the key determinants of normal and tumor capillary network formation.

Clearly, this model provides information about only a small subset of the properties of newly forming capillary networks. Other models, based on ad-hoc growth rules, have tried to reproduce other properties such as branching angle distribution [66]. Since this model is founded on basic physiologic phenomena and does not delve into the minutiae of the physiologic mechanisms involved, it is limited in the scope of the properties it can predict or reproduce without resorting to ad-hoc assumptions or rules. However, the model fully meets the goals of this research to uncover the key determinants of vascular network formation. The analysis in the preceding sections has shown that when growth factor diffusion is combined with an autocrine mechanism of growth factor release by endothelial cells, a compact capillary network is obtained. Furthermore, the analysis has shown that an autocrine mechanism is a relatively efficient and fast mechanism for capillary growth, and that the model's growth rates are in order-of-magnitude agreement with "real-life" growth rates.

Chapter 6

Modeling Tumor Network Growth

In subsection 4.5.2 we saw that the measured fractal dimensions of tumor vascular networks are consistent with the fractal dimensions of the critical percolation cluster. Percolation being a local growth process [131], suggested that there exists some local substrate property which determines tumor capillary growth. Biological observations suggest that this local property can be hypothesized to be extracellular matrix inhomogeneity (see subsection 4.5.2). In order to examine this hypothesis, the previous growth model (see section 5.1) was modified so that a randomly selected subset of all lattice sites became inaccessible to growth.

6.1 Growth Model Modification

In the modified version of the growth model, each lattice site is assigned a random number R in the range [0,1]. Growth occurs at all particle reception sites where $R<T$. T is a preset number in the range [0,1] and represents the fraction of lattice sites which are accessible to growth (T is also called the "accessibility", and reported as a percent). If no site with $R<T$ is available for growth in a given cycle, growth occurs at the site with the lowest R ($R>T$). The local amplification mechanism is taken into account as described in section 5.1.

6.2 Model Results

The model results are shown in Figure 6-1. Two important points are evident in these results.

First, we see that as the threshold T drops and more sites become inaccessible to growth, the transition zone between the compact phase and diffusion-limited phase becomes larger and more blurred. Furthermore, an increasing local amplification factor (F) is required to achieve a given fractal dimension as T decreases. Theoretically, below the site-percolation threshold for a square lattice ($T<0.6$), a compact structure cannot be achieved even as $F\rightarrow\infty$. In reality, F is finite and

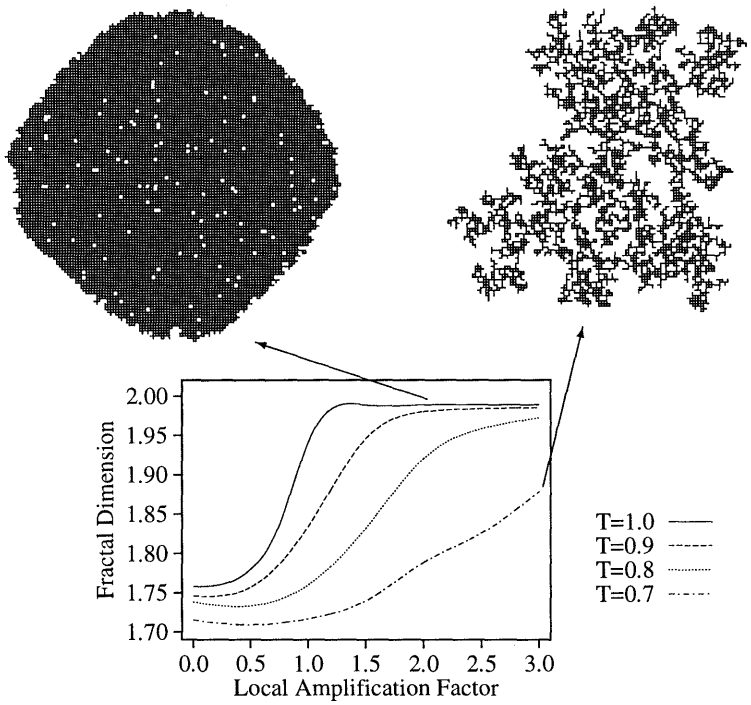

Figure 6-1: Effect of substrate inhomogeneity on fractal dimension. The different curves represent different degrees of inhomogeneity as represented by the accessibility parameter T. The lines represent cubic splines with a smoothing parameter $s_p = 0.9995$ (see section 3.7). Error bars were omitted for clarity. Note that the line with $T=1$ is the same line as in Figure 5-1b.

not every lattice site is necessarily explored at each growth cycle so that for $F = 3$, for example, $T \leq 0.7$ is sufficient to ensure a percolation-cluster type of vasculature.

Second, we see that without local amplification, the structure remains diffusion limited with $d_f \approx 1.7$, relatively insensitive to variations in the accessibility T. This observation further emphasizes the possible importance of the local amplification (i.e., autocrine) mechanisms in normal and tumor vascular network formation. Figure 6-1 demonstrates that in the presence of local amplification ($F = 3.0$) and 30% inaccessibility, a non-compact structure reminiscent of tumor vasculature was obtained.

6.3 Correlation with Biological Observations

The model results are quite striking in showing that the modification of a single parameter in the normal growth model leads to the formation of pathological-looking networks, similar in appearance and fractal characteristics to tumor vascular networks. In subsection 4.5.2 the hypothesis was advanced that this parameter (substrate inhomogeneity) is associated with matrix properties in tumors. The question now arises whether there exists any experimental evidence which demonstrates that changes in the extracellular matrix (ECM) can indeed modify angiogenesis. The answer is

positive.

It is well-known that when endothelial cells are cultured in a collagen gel they reorganize into a network of capillary-like tubes [94]. Recent in vitro studies of endothelium–ECM interactions have shown that ECM molecules have the ability to induce endothelial cell differentiation and growth by chemical and mechanical means [22, 58]. Chemically, the binding of transmembrane integrin receptors induces these effects. Mechanically, the effects are mediated by the resistance to cell-generated loads applied to the integrin receptors. One such study showed that as fibronectin concentrations in culture are increased, endothelial cell proliferation rate increases too [57] (fibronectin is a constituent of the ECM). Another study demonstrated that in vitro capillary tube formation can be modified by the application of anti-integrin antibodies [39].

Recent in vivo studies on the control of angiogenesis by the extracellular matrix also support the model. In one study [135] the local composition of the ECM was manipulated by varying the GM3:GD3 ganglioside ratio in the rabbit cornea. The study showed that these variations could either stimulate or repress angiogenesis. Another study [13] showed that by administering an antagonist to integrin $\alpha_v\beta_3$, the most promiscuous of the integrin family, ECM interaction with endothelial cells was inhibited and angiogenesis repressed.

6.4 Conclusions

The main conclusion from the model is that substrate inhomogeneity may be a key determinant of tumor vascular network formation. The studies cited in section 6.3 show this conclusion to be consistent with known properties of the ECM. This conclusion implies that in order to modify tumor vasculature significantly, the underlying substrate properties must also be modified. However, modification of the ECM may have two effects with uncertain therapeutic implications. On the one hand, if ECM inhomogeneity is reduced, vascular network formation will be more uniform, and the extent of the avascular areas will diminish. This may improve nutrient supply to tumor cells, but it will also assure a more homogeneous delivery of therapeutic agents. On the other hand, if ECM inhomogeneity is increased, the extent of the avascular areas will be greater, more tumor cells will be deprived of nutrients, but the delivery of therapeutic agents to tumor cells may be hampered. These contradictory effects may be part of the reason why tumors display an enormous variability in their responses to therapeutic interventions.

While it is uncertain whether this model's conclusions can be extended to other forms of pathological angiogenesis, it should be noted that some of the architectural features of tumor vasculature (e.g., increased avascular areas) are also present in other states of abnormal neovascularization such as diabetic retinopathy [3].

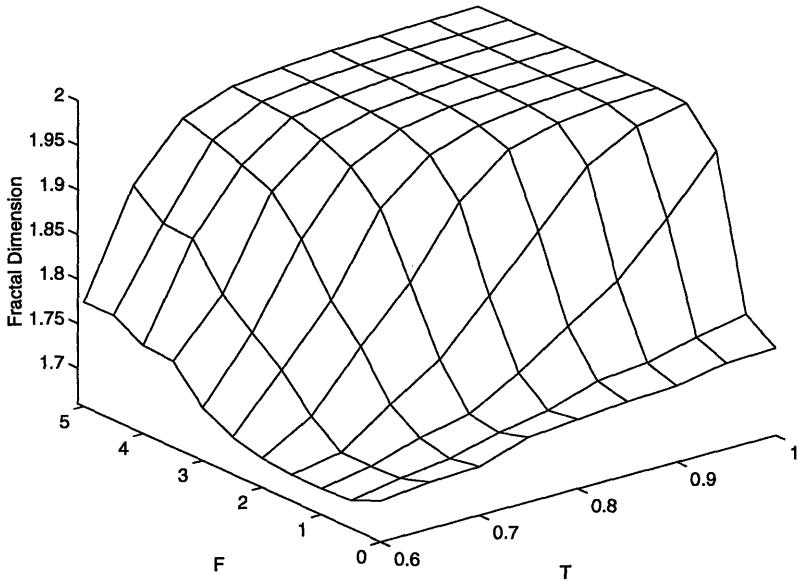

Figure 6-2: Fractal dimension as a function of model parameters. The horizontal axis represents the level of substrate inhomogeneity as expressed by the substrate threshold parameter T. The vertical axis represents the local amplification factor F.

6.5 Evaluation and Comparison with "Classic" Percolation

The growth model described in this chapter differs from "classic" percolation growth models such as spreading percolation and invasion percolation [131] in several important aspects. In traditional percolation models every perimeter site is a potential candidate for growth. In the model described in this chapter, some perimeter sites may be excluded from consideration in a particular growth cycle because they did not receive any growth factor "particles". This means that the threshold parameter T in this model is not equivalent to the threshold probability p in percolation [127]. An effective threshold probability p can be associated with this model, and it is generally less than T, unless we can assure that every perimeter site is explored (i.e., when the local amplification factor F is high, since $\lim_{F\to\infty} T = p$).

The interplay of the local amplification factor F and the substrate inhomogeneity T provides a large domain in which percolation-like scaling can be observed. Figure 6-2 shows a surface plot of fractal dimension as a function of substrate inhomogeneity (T) and local amplification (F). Two trends are evident in this plot. First, the fractal dimension increases as inhomogeneity decreases (i.e., as $T \to 1.0$). This trend is expected from well-known percolation properties [127]. Second, the fractal dimension increases as the local amplification factor F increases. This trend can be understood by noting that an increase in F is tantamount to an increase in the effective threshold probability p.

By calculating the contour lines which correspond to the domains of scale invariance described in

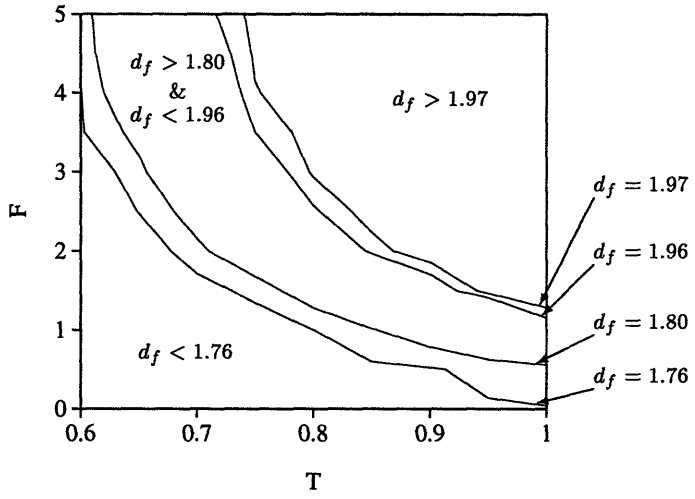

Figure 6-3: Fractal domains as a function of model parameters. The horizontal axis represents the level of substrate inhomogeneity as expressed by the threshold parameter T. The vertical axis represents the local amplification factor F. The contour lines separate this two-dimensional space into three morphological domains: compact ($d_f > 1.97$), tumor-percolation ($1.80 < d_f < 1.96$) and tree-like ($d_f < 1.76$). The contour line values were chosen at two standard deviations from the experimentally observed values (see Chapter 4).

Chapter 4, one can see how this model helps explain the large variations observed in tumor networks. Figure 6-3 shows contour lines drawn at two standard deviations from the observed fractal dimensions in normal capillary, normal arteriovenous, and tumor networks (see Chapter 4). These contour lines divide the T-F space into three domains with thin transition zones between them. Figure 6-3 illustrates that a normal capillary network (i.e., compact) morphology can be achieved even when the substrate is not perfectly homogeneous, provided the local amplification mechanism is strong enough to push p well above the percolation threshold. Furthermore, it illustrates that within the tumor network morphology domain small variations in local amplification or substrate inhomogeneity can lead to relatively large variations in fractal dimensions. This observation provides a possible explanation as to the reason why the standard deviation of fractal dimension measurements in tumor networks is 4 times that of normal capillary networks (see Chapter 4).

The variability in tumor network characteristics can be further explored by plotting the minimum-path dimension as a function of substrate inhomogeneity (T) and local amplification (F), as shown in Figure 6-4. Two trends are evident in this plot. First, the minimum-path dimension decreases as inhomogeneity decreases (i.e., as $T \rightarrow 1.0$). Second, the minimum-path dimension decreases as the local amplification factor F increases. These trends can be understood by noting that either an increase in F or an increase in T makes more sites effectively available for growth, thus reducing the average length of a path between any two points. Figure 6-5 shows a contour line corresponding to

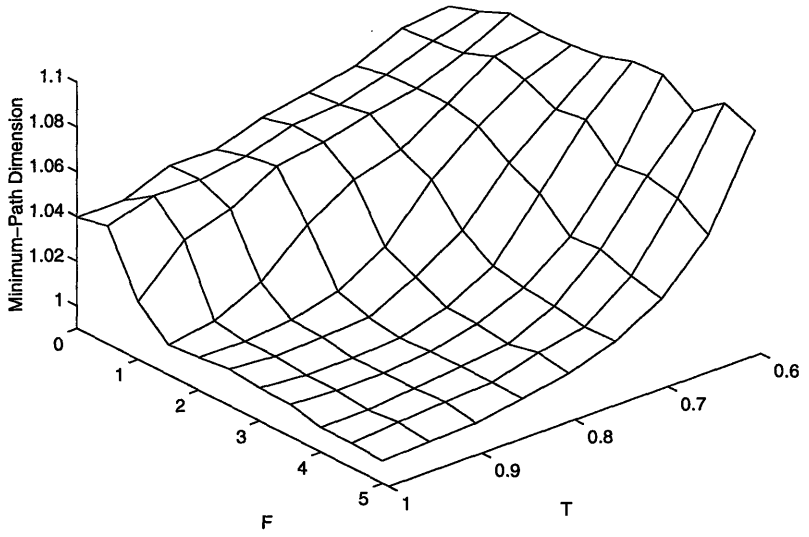

Figure 6-4: Minimum-path dimension as a function of model parameters. The horizontal axis represents the level of substrate inhomogeneity as expressed by the threshold parameter T. The vertical axis represents the local amplification factor F. Note that the T and F axes are reversed in comparison with Figure 6-2.

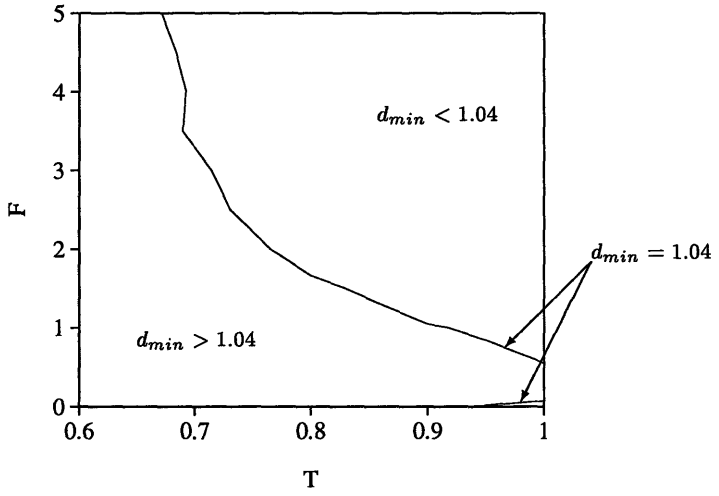

Figure 6-5: Minimum-path domains as a function of model parameters. The horizontal axis represents the level of substrate inhomogeneity as expressed by the threshold parameter T. The vertical axis represents the local amplification factor F. The contour lines separate this two-dimensional space into two morphological domains: straight ($d_{min} < 1.04$) and tortuous ($d_{min} > 1.04$). The contour line value was chosen at two standard deviations from the experimentally observed value for normal networks (see subsection 4.1.2).

$d_{min} = 1.04$, a value which is two standard deviations above the observed d_{min} of normal capillary networks. This contour line separates the T-F space into two morphological domains — a "straight" vessel domain and a "tortuous" vessel domain. Furthermore, this contour line splits the tumor network domain in Figure 6-3 into two nearly equal parts. By doing so it provides an explanation to the observation that the minimum-path dimension variance in tumors is 4 times that of normal networks (see Chapter 4). Figure 6-5 also points out that a large part of the domain with $d_f < 1.76$ in Figure 6-3 does not correspond to "classic" DLA structures. While a tree-like morphology is often observed in this domain, the structures tend to be much more tortuous than DLA clusters. The significance of this domain for the formation of capillary networks is rather limited, however, because tree-like capillary networks are very rare in nature [89].

It should be noted that the contours in Figures 6-3 and 6-5 are not exact, since the fractal dimensions of the model networks were calculated with a standard error of 2% (or less). Although the position of the contour lines may shift slightly, the qualitative results and conclusions are not affected.

Chapter 7

Modeling Arterial and Venous Network Growth

In subsection 4.1.1 we saw that the fractal dimensions of arteriovenous networks are consistent with the fractal dimensions of DLA clusters. As discussed in subsection 4.5.1, these results may seem consistent with the previously accepted view of the angiogenic process where growth factors initially diffuse from low-nutrient or hypoxic regions and induce growth. Under such circumstances, the diffusion-limited behavior of the arteriovenous network is quite understandable. This explanation, however, is flawed because angiogenesis does not seem to occur on the artery-vein level. Arteries and veins form by remodeling of existing capillaries.

However, if one thinks of the capillary mesh as a regular lattice (which it often appears to be — see, for example, Figure 2 in Reference [124]), one can view artery-vein formation as a remodeling process following Laplacian growth rules that occurs on a fully-occupied pre-existing capillary lattice. Indeed, experimental evidence has shown that arterialization occurs in the capillaries which are directly connected to the tips of the terminal arterioles [108] and not at random sites in the capillary mesh. If this is the case, one needs to explain what is the the biological equivalent of the Laplacian field which governs the remodeling process. If one assumes, as the biological evidence suggests, that artery-vein formation is a slow process, which occurs after a capillary network has largely formed, then one cannot assume that arteries and veins respond to the same growth factors as do newly forming capillaries. There could be, however, certain growth factors which are potent mitogens for smooth muscle cells and not for endothelial cells. Therefore, one possibility for the Laplacian field is a diffusion field of such a growth factor, set up by the secretion of these growth factors in low-flow regions. DLA-based vascular growth models published to date [51, 71], have all implicitly assumed such a diffusion field.

There is also another possibility for the source of gradient-sensitive growth of arteries and veins. It

has been observed in many microvascular studies that the steepest pressure gradients in the vascular system occur in the terminal arterioles[1] [80, 136]. This observation may lead us to postulate that if the remodeling rate (length/time) is dependent on the pressure gradient in the vessel, we could have a situation analogous to the process of viscous fingering[2] [101]. Yet regardless of the biological source of the Laplacian field associated with the ramified structure of arteriovenous networks, a fundamental problem remains. When a compact capillary mesh remodels into a vascular tree, some vessels must be resorbed, since a vascular tree is different from a vascular mesh not only geometrically (e.g., vessel diameters) but also topologically (i.e., in its connectedness). All DLA-based growth models fail on this point because they ignore the issue of capillary resorption.

In this chapter I would like to present a possible mechanism for vessel resorption, based on acceptable biological assumptions. I do not intend to claim that this is the actual physiological mechanism, since there is no currently known way to validate such a claim. I simply intend to show that using a general assumption about the role of shear stress in vessel remodeling, it is possible to develop a novel (yet simple) model, under which a perturbation in shear stress can lead to divergent behavior of vessel diameter.

7.1 Role of Stress in Vessel Remodeling

The hypothesis that vessel growth and resorption may be linked to stress in the vessel has been advanced in the past, on the basis of three empirical observations [38]. First, transport through cell membranes depends on the strain in the membranes. Second, the behavior of actin and actin-myosin bridges is strain-dependent. Third, chemical reaction rates are dependent on pressure, stress and strain. In this context, Fung [38] postulated that there should be a homeostatic stress state, and that deviations from this state could lead to either growth or resorption (depending on direction and magnitude of stress change), subsequently returning the stress level to its homeostatic state. Recent molecular biologic studies [55, 98, 102, 122] lend further credence to the hypothesis that vessel growth may be stress dependent, by showing upregulation of growth factors, adhesion molecules, and other bioactive compounds in response to shear stress changes. There has also been one study showing that changes in circumferential wall stress could be responsible for arterial tree remodeling [109]. In the latter report, a model was proposed in which input pressure changes were shown to cause terminal arteriole formation or rarefaction by assuming a stress-dependent growth rule. In addition, a plethora of in vivo experimental evidence shows that hypertension can lead to vessel rarefaction [120].

[1] Theoretically, it can be shown that in any vascular tree which obeys Murray's law [97], $Q \propto r^3$, the pressure gradient ∇P in a vessel is inversely proportional to the vessel radius r: $\nabla P \propto r^{-1}$.

[2] Viscous fingering occurs when a less viscous fluid is injected into a more viscous fluid under conditions in which the viscous forces alone determine the flow. Under such conditions a fingered interface emerges. It can be shown that the situation is analogous to Laplacian growth because the interface velocity is proportional to the interface pressure gradient ($\vec{v} \propto \nabla P$).

The existence of a constant homeostatic shear stress level in a vascular tree can also be derived [117] from Murray's law [97]. Murray's law states that if one minimizes the energy required to maintain flow in a given vessel segment, then the volumetric flow Q, obeys:

$$Q = kr^3 \tag{7.1}$$

where r is the vessel radius, and k is a constant depending on blood viscosity and endothelium metabolism. The average velocity in the vessel therefore obeys $\bar{v} \propto r$. If one assumes Poiseuille flow, then the velocity profile is given by:

$$v(x) = v_{max}\left[1 - \frac{(r-x)^2}{r^2}\right] \tag{7.2}$$

where x is the distance from the vessel wall. The shear stress at the wall ($x = 0$) is therefore:

$$\tau \propto \left.\frac{dv}{dx}\right|_{x=0} = \frac{2v_{max}}{r} \tag{7.3}$$

but since in Poiseuille flow $v_{max} = 2\bar{v}$, we see that the shear stress in all vessels obeying Murray's law is the same. Experimentally, many arterial systems obey Murray's law [118].

While the role of stress in vessel remodeling seems important, to date no model has been proposed for stress-based capillary network remodeling.

7.2 Shear-Stress Based Arterialization Model

Using the most general assumptions and trying to avoid ad-hoc growth rules, an arterialization model for a capillary lattice will now be advanced. In general, the model operates on a graph whose connectivity matrix is known. All capillary segments connecting adjacent nodes (i.e., lattice sites) are assumed to be of equal length L. Initially, all segments are also of equal diameter d. The conductance C_{ij} of a capillary segment between nodes i and j, assuming Poiseuille flow [80] and fixed viscosity (η) is:

$$C_{ij} = \frac{\pi d_{ij}^4}{128\eta L} \tag{7.4}$$

Designating P_i as the pressure at node i, the flow Q_{ij} between nodes i and j is then obtained as:

$$Q_{ij} = C_{ij}(P_j - P_i) \tag{7.5}$$

The wall shear stress in segment ij is then given as (adapting Equation 7.3):

$$\tau = \eta \left.\frac{dv}{dx}\right|_{x=0} = \eta\frac{4v_{max}}{d} \tag{7.6}$$

Since in Poiseuille flow $v_{max} = 2\bar{v}$, and $\bar{v}$ is defined as the volumetric flow divided by the vessel cross-section, one obtains:

$$\tau_{ij} = \frac{32\eta}{\pi} \frac{Q_{ij}}{d_{ij}^3} \tag{7.7}$$

When the capillary lattice is in its initial state, the shear stresses τ_{ij} are calculated and assumed to represent the homeostatic state of each vessel segment ij. Subsequently, a perturbation is introduced. In this model, the perturbation was chosen to be a diameter change in one or more of the vessels[3]. The new shear stresses are calculated, and "eligible" vessel diameters are modified according to a shear stress remodeling rule (see Equation 7.8 below). A capillary can be "eligible" for remodeling only if it is directly connected to a previously modified vessel.

At the heart of the model is a time-dependent shear stress remodeling rule. Looking at a single vessel, the homeostatic shear stress level will be designated as $\tau(0)$. The time-dependent diameter $d(t)$ of the vessel is assumed to obey the difference equation:

$$\frac{\Delta d}{\Delta t} = sd(t) \frac{\tau(t) - \tau(0)}{\tau(0)} \tag{7.8}$$

where s is a parameter representing the sensitivity of relative diameter changes to relative shear stress changes. Equation 7.8 simply states that the relative rate of diameter change is directly proportional to the relative deviation from the homeostatic shear stress level. At each time step Δt, the shear stress is calculated for all vessels in the network, and the diameter change Δd is calculated for each "eligible" vessel. The process progresses in time until all "eligible" vessel diameters reach a steady state value. If the steady state diameter drops below $0.1d(0)$, it is set to zero[4]. Depending on the specific capillary network, a new generation of "eligible" vessels may be defined and the whole remodeling process repeated. In all simulations input pressure was fixed at $P_{input} = 1$ and output pressure was fixed at $P_{output} = 0$ (it was assumed that the system is linear in ΔP). In order to simplify calculations the fixed vessel length was set at $L = \pi/128\eta$ in the appropriate units without loss of generality. Initial capillary diameters were chosen as $d(0) = 1$, unless otherwise noted. To avoid numerical instabilities, the time step Δt was chosen small enough to obtain a "smooth" response. The simulations were implemented using the MATLAB matrix manipulation software. The model routines are listed in section A.8.

[3] Any other perturbation which induces asymmetric shear stress changes would be adequate. Such perturbations include length change and pressure or flow changes (if the feeder vessels are not identical).

[4] A diameter drop to $0.1d(0)$ represents a 10000-fold decrease in conductance. For all effective purposes, there will be no flow through this capillary, and it can be assumed resorbed.

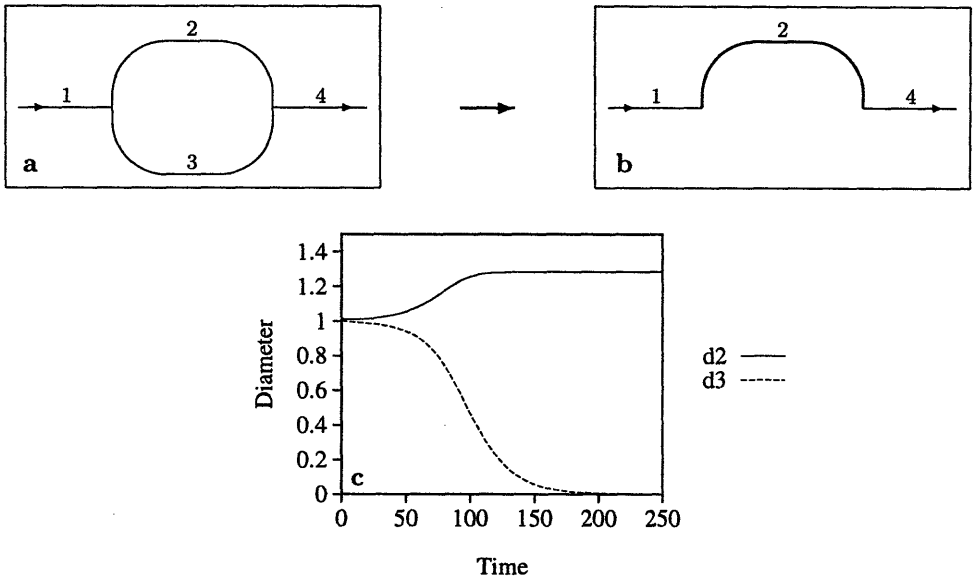

Figure 7-1: Self-loop resorption: **(a)** The initial self-loop with all diameters being unity; **(b)** The remodeled structure, after the diameter of vessel 2 was initially perturbed to 1.01, shows that vessel 3 has been resorbed; **(c)** The time course of the diameter changes. Note that d_2 reaches a steady state before d_3 because when d_3 is small enough it no longer has a significant effect on the flow behavior of the structure. The sensitivity parameter was $s = 0.05$ in this simulation.

7.3 Model Results

In the following sections, I will present several examples showing that this arterialization model produces the desired results. I will start with the simplest example — a self-loop (a loop with 2 nodes on its perimeter). I will then go on to a loop with 3 nodes on its perimeter. Subsequently, I will explore the model on a capillary lattice. I will show that in all cases selective vessel resorption occurs. This resorption remodels a mesh into a tree-like structure.

7.3.1 Self-Loop Resorption

A self-loop, the simplest of all vascular loop structures, is shown in Figure 7-1a. Such a loop has only two nodes on its perimeter. The loop shown in Figure 7-1a has an input vessel (vessel 1), an output vessel (vessel 4) and two loop vessels (vessels 2 and 3). All vessels start off with a unity diameter, and only vessels 2 and 3 are allowed to remodel. Figure 7-1b shows the resulting vasculature after vessel 2 is perturbed and its diameter increased to 1.01. The time course of the diameter changes of vessels 2 and 3 is shown in Figure 7-1c. A similar result is obtained if vessel 3 is the one perturbed and its diameter initially decreased.

Self-loop resorption is not a purely theoretical exercise. Self-loops occur occasionally in vasculature. Furthermore, self-loop resorption may be a key process in the formation of rare tree-like capillary networks seen in some marsupials [89].

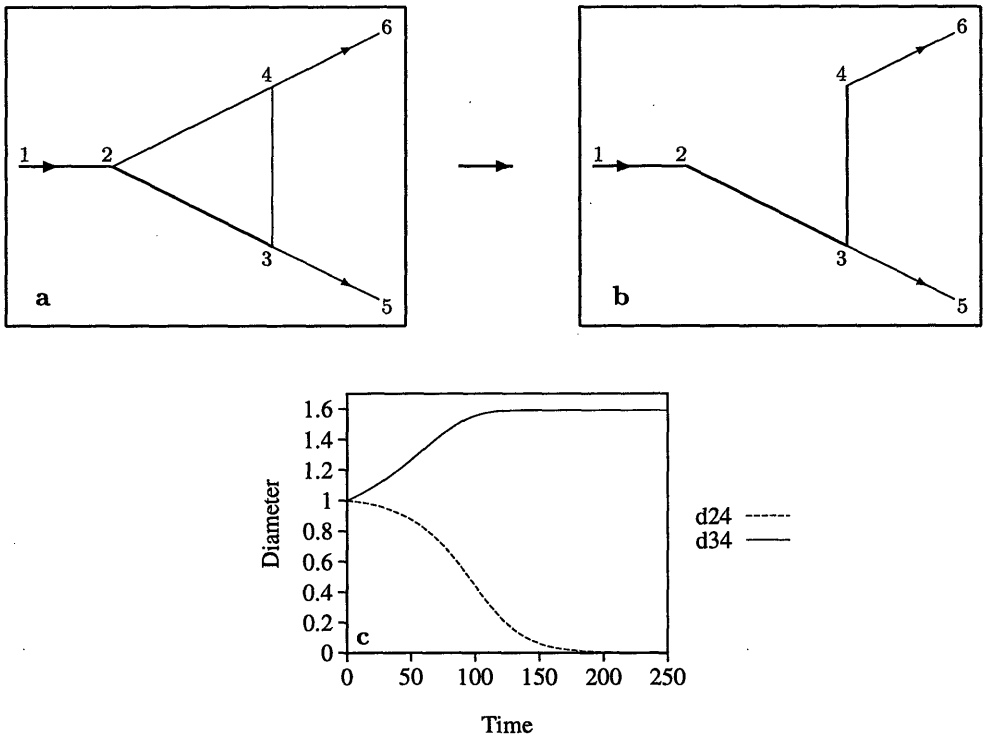

Figure 7-2: Complex loop resorption: **(a)** The initial structure with $d_{12} = 1.50$, $d_{23} = 1.30$, and all other diameters being unity; **(b)** The remodeled structure, after d_{23} was initially perturbed to 1.33, shows that vessel 24 has been resorbed; **(c)** The time course of the diameter changes. The sensitivity parameter was $s = 0.05$ in this simulation.

7.3.2 Complex Loop Resorption

Figure 7-2a shows a loop with three nodes on its perimeter (called a "complex" loop), representing a typical situation during the arterialization process. Segments 12 and 23 are already arterialized. For the arterialization to proceed, either capillary 24 or capillary 34 has to be resorbed. The perturbation in this case is an increase in the diameter of arteriole 23 (from 1.30 to 1.33). Only capillaries 24 and 34 are allowed to remodel. The input and output pressures were fixed so $P_1 = 1$ and $P_5 = P_6 = 0$. Figure 7-2b shows the resulting vasculature after vessel diameters d_{24} and d_{34} reach a steady state. The time course of the diameter changes is shown in Figure 7-2c.

The choice of the initial perturbation is important in determining which capillary is resorbed. In the network depicted in Figure 7-2a, a perturbation that slightly increases the diameter of capillary 24 would result in the resorption of capillary 34. In general, however, the key observation is that in this model a small perturbation tends to be amplified, resulting in the resorption of vessels. The choice of sensitivity parameter s was observed to affect only the rate at which the vessels reach their steady-state diameters.

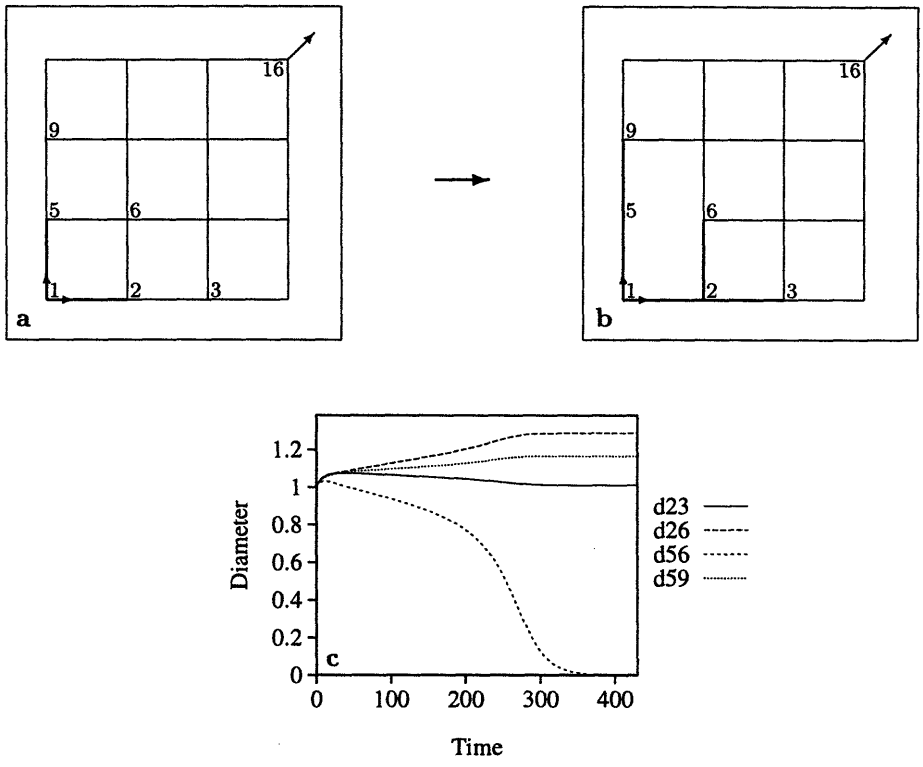

Figure 7-3: Loop resorption on a 3×3 square lattice: **(a)** The initial lattice with all diameters being unity except $d_{12} = 1.21$ and $d_{15} = 1.19$; **(b)** The remodeled structure, showing that vessel 56 has been resorbed; **(c)** The time course of the diameter changes. The sensitivity parameter was $s = 0.05$ in this simulation.

7.3.3 Resorption on a Lattice

In order to assess whether the proposed model yields the desired results in a network consisting of multiple loops, simulations were performed on a square capillary lattice. The input was selected at the lower left corner and output at the upper right corner. In these simulations, the perturbation used was an asymmetrical increase in the diameters of the two vessels connected to the lower left corner (node 1) to 1.21 and 1.19.

Figure 7-3 presents the results obtained on a lattice of 3×3 segments, where remodeling was allowed for one generation. The input and output pressures were fixed so $P_1 = 1$ and $P_{16} = 0$. All vessels directly connected to the perturbed vessels were allowed to remodel (i.e., capillaries 23,26,56,59). Clearly, for a tree structure to develop, either capillary 26 or capillary 56 must be resorbed. Figure 7-3b shows the resulting vasculature after vessel diameters reached a steady state. The time course of the diameter changes is shown in Figure 7-3c. As seen, vessel 56 is resorbed, and the tree structure gains another generation.

Figure 7-4 presents the results obtained on a lattice of 5×5 segments, where remodeling was performed for three generations. The input and output pressures were fixed so $P_1 = 1$ and $P_{36} = 0$. In each generation, only capillaries directly connected to previously remodeled vessels were allowed

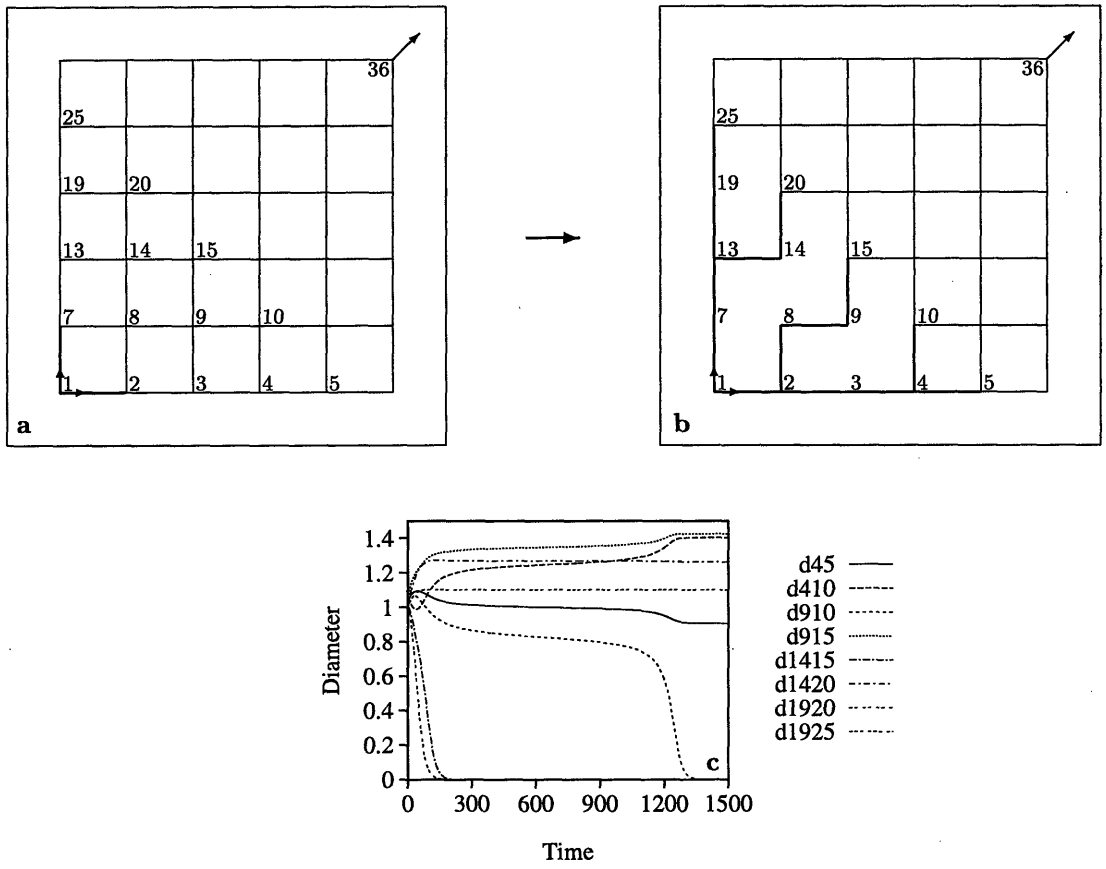

Figure 7-4: Three generations of remodeling on a 5×5 square lattice: **(a)** The initial lattice with all diameters being unity except $d_{12} = 1.21$ and $d_{17} = 1.19$; **(b)** The remodeled structure, after three generations of remodeling showing that several vessels have been resorbed; **(c)** The time course of the diameter changes of the third generation. The sensitivity parameter was $s = 0.05$ in this simulation.

to remodel. Before the third generation remodels, nodes 10, 15, and 20 each have two inflow vessels and two outflow vessels. In order for the tree structure to continue into its third generation, one of the inflow vessels into each of these nodes has to be resorbed. Figure 7-4b shows the resulting vasculature after the third generation vessel diameters reach a steady state. The time course of the diameter changes during the third generation remodeling is shown in Figure 7-4c. It is clear that only those vessels which were superfluous for the formation a tree-like topology were resorbed. It is also important to note that the time needed to reach a steady state in the third generation is approximately 4 times longer than the time needed to reach a steady state in the first generation (compare Figure 7-4c with Figure 7-3c — the different lattice size has negligible effect on first generation remodeling time).

Similar resorption patterns were obtained when the model was implemented on a hexagonal lattice. While a hexagonal lattice does not seem to differ from a square lattice in the potential of certain capillary segments to be resorbed, it seems to be less sensitive to small perturbations to the

symmetry of the lattice. For example, a perturbation of feeder vessel diameters to 1.19 and 1.21 (as in the square lattice case) could not elicit a resorption response. However, a perturbation of feeder vessel diameters to 1.01 and 1.21 elicited a multi-generation resorption of vessels (on a hexagonal lattice of 4×4 hexagons). Since a hexagonal lattice is probably more representative of the connectivity properties of real capillary beds, the above observation insinuates that the connectivity properties of the whole network may temper the divergent nature of stress-induced remodeling.

7.4 Model Evaluation

The results from the simple arterialization model described above contain two salient elements. First, they show that a shear-stress based remodeling rule can result in vessel resorption (and vessel enlargement). Second, they show that this resorption is selective in the sense that only vessels which form connections that are superfluous for tree topology are resorbed. Selective vessel resorption is a key element which must accompany any Laplacian remodeling process in order for the arterial network to achieve a tree structure. This model assumed an "eligibility" criterion, which stated that only vessels connected to previously modified vessels could be remodeled. In real life, vessel "eligibility" could possibly be established by the Laplacian field itself. For example, if a large pressure gradient in a vessel was to cause the smooth muscle cells on the vessel to proliferate and migrate, then arterial growth would occur mostly at the terminal arteriole level.

A model which combines Laplacian growth and stress-based remodeling is easy to conceptualize. In essence, the previous model needs to be implemented on a larger lattice and the eligibility criterion made dependent on the Laplacian field gradient. A larger lattice is needed for two reasons. First, it is impossible to accurately simulate Laplacian growth on a 5×5 lattice. Second, on a 5×5 lattice the total length of all arterial segments is comparable to the total length of all capillary segments. While this situation may be appropriate in simulations of the local mechanism of stress-based remodeling, it is inappropriate in simulations of the global behavior of gradient-sensitive remodeling. This is due to the fact that in real life, the total length of arteries is small compared to the total length of capillary segments. Since capillaries form a mesh and arteries form a tree, the pressure behavior of the capillary bed will be relatively unaffected by upstream changes (the effects of such changes will be felt mostly in the terminal branches of the arterial tree). A situation in which total artery length is small compared to total capillary length can only be achieved on a large lattice, where the remodeling of a few capillaries will not drastically alter the properties of the capillary bed as a whole. Unfortunately, such a large lattice (at least 500×500) requires prohibitive amounts of computing time. At each remodeling generation, at least several hundred time steps are required to achieve a steady-state. At each time step, the flows and pressures must be calculated in the whole network. A 500×500 lattice would have $2.5 \cdot 10^5$ nodes, necessitating the simultaneous solution of this number

of equations (for pressures and flows) *at each time step.* This computation requirement was beyond the resources available for this research. It will be left, sadly, for some unspecified future time.

When a simulation on a larger lattice is conducted, it will also be able to address another issue which the model raises. Some readers may find it troublesome that the vascular system in this model bears the inherent instabilities shown in section 7.3. If a small perturbation can cause profound changes in a vascular network, then the network will fail to provide a stable blood supply to the tissue. This problem is partly addressed by the "eligibility" criterion, which restricts the number of vessels that can be remodeled at a given time. In addition, simulations on square and hexagonal lattices point to a *global* effect which may act to stabilize the *local* instabilities introduced by stress-based remodeling. In subsection 7.3.3 it was observed that as the number of vessels that are allowed to remodel simultaneously increases, the remodeling process takes significantly longer to reach a steady state. Furthermore, it was observed, that a hexagonal lattice was less sensitive to small initial perturbations than a square lattice. Experimental evidence of integration of biological signals by capillary networks [125] may suggest that in a large capillary network multiple perturbations are needed to induce arteriolar response. These observations may lead one to hypothesize the existence of a global stabilizing effect dependent on network size and connectivity. However, a solid confirmation of this effect can only be performed with large lattice simulations.

7.5 Venous Network Formation

The model discussed in previous sections explicitly assumed that the remodeling was occurring from the inflow direction, thus simulating the process of arterialization. There is no mathematical obstacle to extending this sort of remodeling to the outflow side of the network as well, since the mass conservation equations ($\sum_j Q_{ij} = 0$ at each node i) involved in solving for the flow (and hence the shear stress) hold true if the direction of flow is reversed. Thus, one may argue that a branching venous network may form by the same stress-dependent mechanism.

While capillary resorption leading to the formation of post-capillary venules may occur by a mechanism similar to that leading to the formation of terminal arterioles, the source of the "eligibility" criterion in this case is less obvious. If the "eligibility" criterion is gradient-sensitivity, as postulated in the arterial case, then the source of the Laplacian field is more obscure in the venous case, because the pressure gradients throughout the venous side are relatively low. A possible solution to this problem lies in the observation that the histological difference between capillary and venule is less pronounced than the difference between capillary and arteriole [63], although, on average, venules have larger lumen diameters than arterioles. This may lead us to suspect that resorption of capillaries leading to the formation of venules may have a passive component to it. Such a component would manifest itself in the enlargement of diameter in response to increased

flow without an active mechanism (e.g., stress-based) for vessel resorption. In this scenario, some vessels would be resorbed when a neighboring vessel becomes large enough to occlude them or to draw enough flow away from them (by virtue of increased conductance) so as to collapse them. Due to the limited and controversial knowledge about the mechanical nature of vessel collapse and occlusion, this hypothesis should be regarded as highly speculative. Unfortunately, there have been no comprehensive studies of the dynamics of concomitant arterial and venous network formation. Such a study is necessary to determine whether the same time constants are involved and whether the same resorption patterns are observed. Until then, the model proposed in this chapter can apply to both arterial and venous network formation.

Chapter 8

Transport Implications

The transport of diffusible substances in tumors is a key process in the growth and treatment of solid tumors [61, 62]. For example, adequate oxygen supply is critical for tumor growth but also for successful radiation therapy. The scale-invariant behavior of vascular networks uncovered in Chapter 4 leads to important insights about the transport characteristics of tumors. It will be shown that by using percolation-like scaling as a paradigm for tumor vasculature, important experimental observations can now be explained.

Historically, most transport models have relied on drastic simplifications of vascular architecture. The most commonly used model, the Krogh cylinder, is based on the assumption of uniformly spaced straight parallel vessels [104]. The Krogh cylinder model is appropriate in the case of a vasculature which displays compact scaling behavior, such as the normal capillary bed. Indeed, one of this model's most profound impacts has been in the study of oxygen exchange in skeletal muscle [104], where capillaries can be accurately approximated to be parallel and arranged on a two-dimensional hexagonal lattice when viewed in cross section [11]. However, in this study it has already been shown that tumor vasculature does not display compact scaling but rather percolation-like scaling (section 4.2). Tumors are known for tortuous vessels and for avascular areas of many different sizes [1, 60, 68, 76], thus violating the assumptions necessary for the application of the Krogh cylinder model. Given the unavailability of a general vascular architecture model in tumors, there have been some attempts to quantify the transport characteristics of tumors by using reconstructions of small regions of tumor vasculature [114]. While these studies were important in demonstrating that the Krogh cylinder model was indeed inappropriate for quantifying the transport characteristics of tumors, their conclusions could not be extended to other tumor networks because the studies did not include a quantitative model of vascular architecture. By using the findings of this research, I will show that a general paradigm can be developed for assessing the transport behavior of scale-invariant vascular networks.

8.1 Geometric Resistance

The percolation-like ($d_f \approx 1.9$) behavior of tumor vasculature helps explain a long-standing paradox regarding the geometric resistance of tumors. Experimentally, it has been observed that tumors display elevated geometric resistance to blood flow when compared to normal tissues of similar weight [115]. However, it has also been observed that tumor vessels increase in diameter as the tumor grows. The mean diameter d of tumor vessels can often be more than 100% larger than that of normal vessels [52]. Since the geometric resistance is proportional to d^{-4}, we can see that, ceteris paribus, the geometric resistance of tumors would have to decrease in comparison to normal tissue. One can argue that in tumors one usually sees a decrease in the vascular density when compared to normal tissue. However, if one assumes that the vascular density is uniformly decreased in the tumor, then in order to counteract a 100% increase in vessel diameter to obtain elevated geometric resistance, the vascular density would have to decrease by a factor larger than 16. Published figures about vascular density in tumors show that it is typically 2-4 times less than normal tissue [25, 78]. Therefore, the source of the elevated geometric resistance remained shrouded in mystery.

The observed scale-invariant properties of tumor vasculature undermine the assumption that vascular density is uniformly reduced in tumors. Since tumors behave like percolation clusters, the large body of work dealing with the electrical resistance of random resistor networks [127] can be applied to explain the aforementioned paradox. In general, the resistance z_0 of a percolation cluster above the percolation threshold is known to obey the power law:

$$z_0 \propto (p - p_c)^{-\mu} \tag{8.1}$$

where p represents the occupancy level, p_c is the percolation threshold, and μ is a positive number. In two dimensions $\mu \approx 1.3$ [127]. We see that close to the percolation threshold ($d_f \approx 1.9$) the resistance diverges, consistent with the observation for tumor networks.

Given the similarity between tumor networks and percolation clusters, one can also cite two intuitive reasons for the elevated resistance in tumor networks. First, the connectivity properties of percolation networks are such that there may be a few flow paths that will carry a disproportionately large part of the flow (technically known as "singly connected bonds" [126]). In contrast, in a compact network the symmetry of the structure assures a fairly homogeneous flow distribution. Second, the flow paths in tumor networks tend, on average, to be longer than flow paths in normal networks, as shown by the elevated minimum path dimension d_{min} in tumor networks (Section 4.2). Since the geometric resistance is directly proportional to the length of the flow path, a tumor vessel with elevated d_{min} will have higher geometric resistance compared to a normal vessel, provided that both vessels are of equal diameter and connect equally distant points.

8.2 Scaling of Avascular Regions

The percolation-like ($d_f \approx 1.9$) structure of tumor vasculature has important implications for transport of diffusible substances in tumors. The number of vascular regions N_{vasc} of length scale L scales as:

$$N_{vasc} = \left(\frac{L_0}{L}\right)^{d_f} \tag{8.2}$$

where L_0 is the upper boundary of the fractal scaling range. The number of all regions (both vascular and avascular) N_{all} of length scale L scales as:

$$N_{all} = \left(\frac{L_0}{L}\right)^{D} \tag{8.3}$$

where D is the embedding Euclidean dimension ($D = 2$ in our case). Therefore, the fraction of tissue f_{avasc} consisting of avascular areas of length scale L scales as:

$$f_{avasc} \equiv \frac{N_{all} - N_{vasc}}{N_{all}} = 1 - \left(\frac{L_0}{L}\right)^{d_f - D} \tag{8.4}$$

This implies that for structures with $d_f \approx 1.9$ and $D = 2$ there exist a few large avascular areas and many smaller avascular areas. The number of cells in the few large avascular areas is not negligible, however. For the observed value of $L_0 \approx 900$ μm and a single-cell length scale of 10 μm, it can be calculated that 39% of all cells will lie in avascular areas of length scale $L > 200$ μm. In a three-dimensional percolation-like structure ($d_f \approx 2.5$ and $D = 3$), 59% of the cells lie in avascular areas of the same length scale. These percentage figures should not be regarded as more than order-of-magnitude estimates, but they point to the observation that the relatively scarce but large avascular areas have the most clinical significance, since they are the areas most prone to resist drug treatment or to be hypoxic and resist radiation treatment.

The relationship stated in Equation 8.4 can be demonstrated by measuring the distance of each point in the tissue to the nearest vessel. Figure 8-1 shows the cumulative distribution of the fraction of tissue at or above a given distance from the nearest vessel (the fraction at or above 1 distance unit (approximately 6 μm) was normalized to be 1.0). The distribution is shown for three representative normal subcutaneous striated muscle capillary networks and three representative LS174T tumor vascular networks. It is evident that the distance distribution in tumors has a much longer tail than that in normal tissue. The long tail reflects the existence of the large avascular areas in tumors. The differences among tumor distributions reflect the variability in size and number of the largest avascular areas. The normal vasculature distributions, however, show little variability. This is due to the relatively uniform intervessel spacing observed in normal tissue. The distributions shown in Figure 8-1 are very similar to those derived from invasion percolation clusters [7].

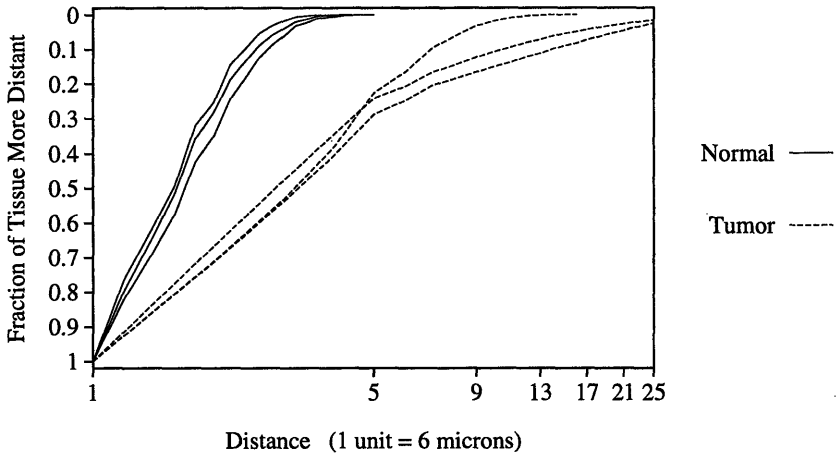

Figure 8-1: Fraction of tissue more distant from nearest vessel as measured from images of normal subcutaneous muscle capillary networks and LS174T tumor networks. The long tail of the tumor distributions reflects the existence of large avascular regions in tumor tissue. The variability in tumor distributions reflects the variability in the size and number of the largest avascular spaces. Note that the horizontal axis is logarithmic.

8.3 Oxygen Transport

Examination of oxygen transport is a useful paradigm in explaining the general transport properties of diffusible substances in tumors. Like many other nutrients or drugs, oxygen is transported via convection in blood and then diffusion from the blood into the extravascular space. Experimentally, tumors are known to display large variations in oxygenation measurements [130]. Hypoxic and anoxic regions are frequently found in tumors [64], but rarely in normal tissue.

The spacing between vessels is an important determinant of oxygen levels in tissue. Under steady diffusion in one dimension, the maximum distance d_{max} that oxygen can diffuse from blood vessels, assuming uniform tissue oxygen solubility α and uniform tissue oxygen consumption M_0, is given by [114]:

$$d_{max} = \sqrt{\frac{2D\alpha P_v}{M_0}} \tag{8.5}$$

where D is the diffusivity and P_v is the partial oxygen pressure in the vessel. Experimentally derived values for $D\alpha$, P_v and M_0 in tumors show that d_{max} can range from 41 μm to 183 μm [114]. In normal tissue, the avascular spaces are relatively uniform in size and the intervessel distances are well below $2d_{max}$. In tumors, however, the large avascular spaces discussed in section 8.2 can lead to the creation of hypoxic regions.

The effects of the percolation-like nature of tumor networks on oxygen delivery are not limited to the creation of hypoxic regions in the large avascular spaces. The increased resistance of tumor networks reduces flow to the tumor as a whole, thus aggravating hypoxic conditions. The existence

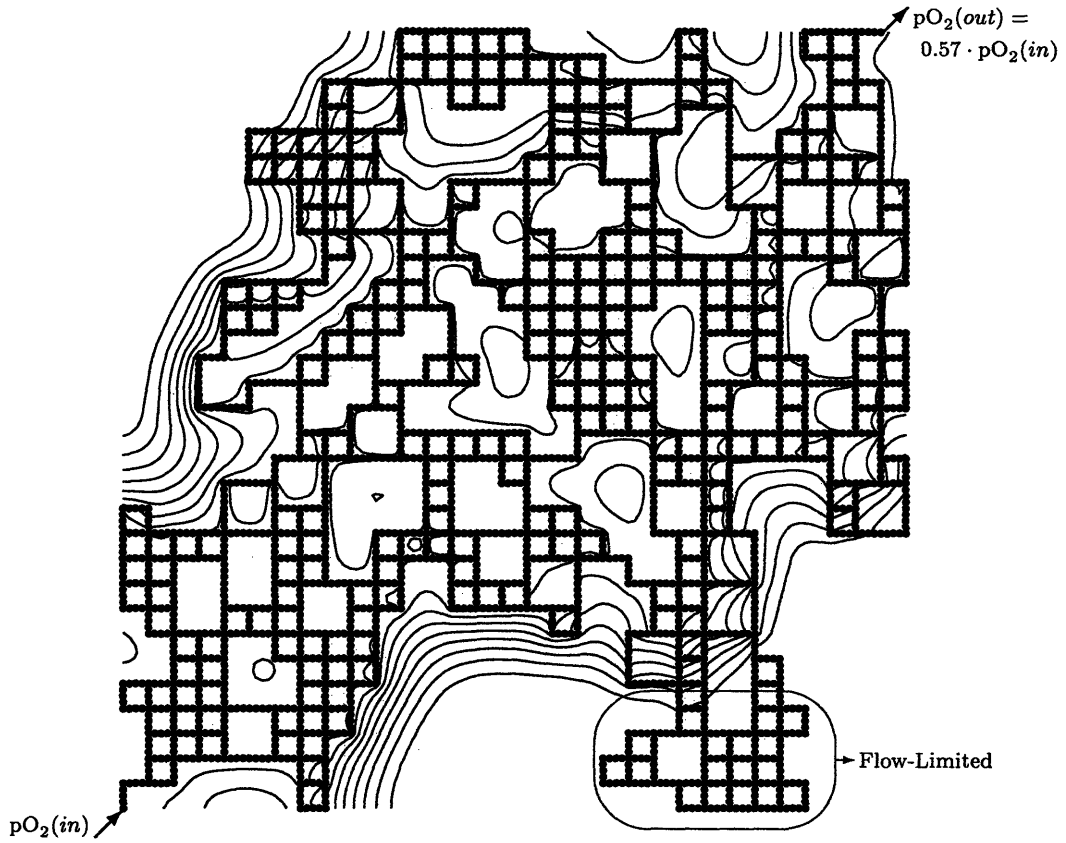

Figure 8-2: Oxygenation contours on a percolation network. Contours reflect 10% increments in oxygen concentration. The above network was generated on a 32 × 32 lattice with an invasion percolation algorithm, which continued until the backbone had a 60% occupancy level. The input was chosen at the lower right corner with 100% oxygen saturation. The output was chosen at the upper left and the flow rate adjusted so the partial oxygen pressure at the exit would be 57% of the inlet oxygen pressure. A locally flow-limited hypoxic region is evident in the lower right.

of a few flow paths which carry a disproportionately large share of the flow in a percolation network (the "singly connected bonds") makes oxygen delivery by such networks much more sensitive to intermittent flow cessation (e.g., by vessel collapse) in random vessels, whereas normal capillary networks are relatively insensitive to flow cessation in a few random capillaries [56]. Furthermore, the flow heterogeneity in a percolation network can result in hypoxic regions due to local flow limitations, even when those regions are fully vascularized.

The effects of percolation-like scaling on oxygen delivery were quantitatively examined in a recent study [7] based on the work described in this thesis. In that study an invasion percolation model was used to generate two-dimensional networks on a a square lattice. Using finite difference techniques, the oxygen convection and diffusion equations were solved for the oxygen concentrations throughout the lattice. The results show that even when oxygen concentration at the network outlet is maintained at relatively high levels, there will still be hypoxic regions due to the effects described above. This can be seen in Figure 8-2 adapted from Reference [7]. The outlet partial oxygen pressure

$pO_2(out)$ was set at $0.57 \cdot pO_2(in)$ by maintaining a high enough inflow rate. Hypoxic regions are evident in the large avascular areas, yet there is also a hypoxic region that is well-vascularized but flow-limited. In the aforementioned study [7] many additional networks were generated and oxygenation levels measured in the model networks were used to generate cumulative oxygenation histograms. These histograms bore a striking similarity to experimentally obtained oxygenation histograms. The small qualitative differences that existed between the in vivo measurements and the model's results could be attributed to the neglect of three-dimensional effects [7].

8.4 Clinical Implications

The inherent limitations to oxygen transport in tumor tissues having percolation-like vascular networks can be generalized to include most other substances that are delivered to target cells by similar convective and diffusive processes. Such substances include both nutrients (e.g., glucose) and therapeutic agents (e.g., drugs or antibodies). As demonstrated in Figure 8-2, even when the tumor as a whole does not deplete the blood of a particular diffusible substance, there will still be regions in the tumor, both vascular and avascular, that will have very low levels of the particular substance. The clinical implication of this observation is that interventions designed to enhance blood flow in tumors may be inadequate to assure delivery of drugs to all tumor cells. In order to assure better delivery of drugs to tumor tissue an intervention that promotes the formation of a more compact and uniform vascular network is necessary. However, the formation of compact vascular networks in tumors may have the detrimental effect of speeding up tumor growth by providing a more uniform supply of nutrients to tumor cells.

Another important implication lies in the potential efficacy of anti-angiogenic interventions. Such interventions are targeted against tumor vessels and are intended to retard tumor growth by cutting off the blood supply to tumor cells. Anti-angiogenic interventions are often combined with standard chemotherapeutic regimens. The effects of such treatments can depend greatly on the manner in which blood vessels are affected. If vessels are resorbed in a random manner, then it is conceivable that anti-angiogenic treatment may push the vascular network closer to the percolation threshold or even below it, thus depriving more tumor tissue of nutrient supply but also reducing the efficacy of any drugs. If vessels, however, are resorbed in manner which creates a sparser yet more compact network, both nutrient supply and drug delivery to tumor cells may become more uniform.

While the conclusions regarding oxygen transport can be readily applied to the transport of most diffusible substances, they may not be valid for heat transfer. Both experimental [74] and theoretical [4] studies have shown that vessels with diameters less than 100 μm are insignificant for heat transfer. In the tissues studied in my experiments virtually no vessels of diameters larger than 100 μm were present. Such vessels can be seen in tissue samples whose dimensions are much

larger than those studied. This implies that in the case of tumor specimens of millimetric size, the heat transfer properties are determined largely by the vasculature of the surrounding tissue and not by the vasculature of the tumor itself. Furthermore, the tissue length scales at which thermally significant vessels are found are most likely above the upper boundary of the fractal range observed in this study (length scales larger than 1 mm). At these length scales the tumor vasculature may no longer display percolation-like scaling (see section 4.3).

Chapter 9

Future Directions

The results described in the preceding chapters can be further explored and developed in the future. The main directions for future research involve three areas:

- Extension of experiments to three dimensions.
- Experiments directed towards refinement of shear-stress based artery-vein growth model.
- Incorporation of the scale-invariant properties of vascular networks into transport models.

9.1 Extension to Three Dimensions

The measurements of fractal dimensions reported in this thesis were performed in a unique quasi-two-dimensional transparent preparation of transplanted tumor cells. Most of the studies of tumor vasculature to date have been performed in transplanted tumors in various specialized preparations. However, it has been suggested that the vasculature of transplanted tumors may show some differences when compared to the vasculature of the primary tumor [31], although the differences reported are mostly physiological and not morphological. Clearly, it would be ideal if the three-dimensional vasculature of a primary lesion could be imaged in situ.

Virtually all possible three-dimensional imaging methods were explored as candidates for imaging vasculature. The requirements for images to be suitable for fractal dimension measurements are those of resolution. The resolution must be high enough that individual vessels could be identified, but low enough to assure that enough vessels will be included in each image so that fractal dimension measurements will be reliable and that a large enough number[1] of vascular networks could be processed in a reasonable time.

[1] In this study fractal dimensions were measured in a total of **106** vascular networks.

One tried method for the imaging of three-dimensional vasculature is that of quantitative reconstruction using serial sections [17]. However, this method has two shortcomings. First, preparing and imaging the serial sections is an extremely laborious process making the imaging of large number of vascular networks practically impossible. Second, this method is destructive, making repeated measurements of the same network impossible.

Another commonly used method for visualization of vessels in their three-dimensional arrangement is that of corrosion casting [116]. Using this technique, a cast is made by injecting a compound into the vessels, the compound subsequently hardens, and the tissue is digested chemically. Traditionally, these casts have been observed using electron microscopy. However, the electron microscopic approach does not offer a way to image the whole three-dimensional structure, but merely to observe limited portions of it. In order to use the corrosion casts for obtaining a three-dimensional vascular image, a truly three-dimensional technique should be used to image them. However, no such technique exists for direct three-dimensional image acquisition (the resolution of NMR microscopy is not sufficient).

Laser scanning confocal microscopy (LSCM) has been used to image vessels at varying focal depths [92]. However, limits on the total thickness and on the minimum focal slice size under the relatively low magnifications necessary for imaging a large enough number of vessels make this method, too, impractical for imaging three-dimensional vascular networks for the purpose of measuring fractal dimensions.

In summary, at this stage no currently known method can produce three-dimensional vascular images of the desired quality and quantity necessary for fractal dimension measurements. These measurements should be performed when an acceptable imaging technique is available.

Extending Computer Algorithms to 3-D

The computer algorithms used to measure fractal dimensions can be trivially extended to three dimensions. The only practical limitation is that of computing time. In order to show that the 2-dimensional tumor vascular growth model (see Chapter 6) can be extended to three dimensions, the source code was modified to incorporate a third spatial dimension (see section A.7). The model was applied to a $32 \times 32 \times 32$ square lattice, with a local amplification factor of $F = 3$, and with a series of accessibility parameters T ranging from 30% to 100% (the percolation threshold in three dimension is $p = 0.3116$ [127]). For each T value, $n = 10$ runs were performed. The relatively small lattice size and sample size were chosen because of computing time limitations. The resulting fractal dimensions are graphed in Figure 9-1.

Figure 9-1 shows that, as in the 2-dimensional case, there is a transition from a non-space-filling fractal vasculature to a space-filling compact vasculature. Furthermore, structures with fractal dimension similar to a three-dimensional critical percolation cluster ($d_f \approx 2.5$ [127]) occur for $T \sim 0.5$,

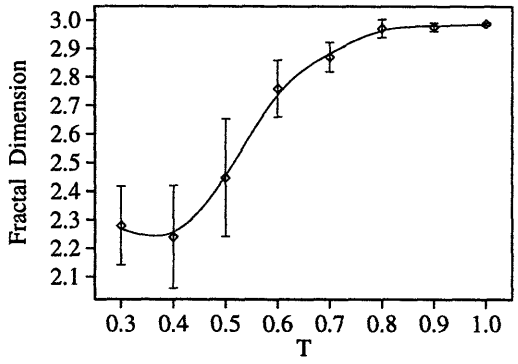

Figure 9-1: Fractal dimensions in 3-D growth model. The tumor vasculature growth model from Chapter 6 was modified to include a third spatial dimension. The model was applied to a $32 \times 32 \times 32$ square lattice, with a local amplification factor of $F = 3$, and accessibility parameter T ranging from 0.3 to 1.0. The transition from a non-space-filling structure to a space-filling structure is evident. The relatively large error bars are due to the small sample size ($n = 10$ in the 3-dimensional runs vs. $n = 100$ in the 2-dimensional runs).

higher than the percolation threshold ($p = 0.3116$) for a three-dimensional square lattice. This behavior is qualitatively similar to the 2-dimensional case. It is due to the fact that not all perimeter sites are explored at each growth cycle because of the finite local amplification factor F (see section 6.5).

In order to further verify the 3-dimensional model, the model was run with parameters corresponding to diffusion-limited aggregation ($F = 0$ and $T = 1.0$) on the same 3-dimensional square lattice. The mean fractal dimension ($n = 10$) calculated was $d_f = 2.49 \pm 0.09$, in agreement with the reported value of $d_f \approx 2.5$ for three-dimensional DLA [90].

The availability of a robust 3-dimensional vascular growth model should further encourage the development of an experimental design to facilitate the measurement of fractal dimensions of three-dimensional vascular networks in tumors in situ.

9.2 Artery and Vein Formation Experiments

The shear-stress based model for vessel growth and resorption presented in Chapter 7 offers a good argument that shear stress variations may be important in the process of artery and vein formation. A large body of existing evidence shows that shear stress variations induce a variety of changes at the molecular and cellular levels in endothelial cells [55, 98, 102, 122]. Furthermore, studies at the network level have suggested that states of altered shear stress are associated with vessel rarefaction [109, 120]. However, to date there have been virtually no studies following the arterialization (or vein formation) process at the single vessel and network levels.

Before the artery-vein growth model can be expanded and improved, several key experimental observations must be made. First, the process of artery and vein formation should be observed in vivo, to establish the time course and the spatial progression of capillary enlargement and resorption. Second, the process of arterialization should be correlated with the hemodynamic behavior preceding it or occurring concomitantly with it. Then experiments should be performed in which shear stress is modified in the observed vessels and the effects of these changes on the process of arterialization are explored. In addition, the process of vein formation should be observed and compared to the arterialization process.

Even if the experiments verify the model's basic assumptions of shear stress based remodeling, there will still be important questions regarding the nature of the homeostatic shear stress levels. Flow changes (and hence shear stress changes) occur constantly in quiescent vascular networks. Therefore, the homeostatic shear stress level may be some sort of average level sensed by the endothelium. Is the homeostatic shear stress level averaged over time, over space, or both? Are there factors which influence the stability of the homeostatic shear stress level? These questions can only be answered by careful experimentation.

9.3 Scale-Invariant Networks in Transport Models

Modeling vascular networks as scale-invariant structures grown according to a specific statistical growth process can contribute to the understanding and modeling of transport processes in tissue by providing a general paradigm to study the effects of various parameters on transport properties. This is important because in tissues with dramatically heterogeneous hemodynamic and geometric vascular parameters (such as tumors), the study of a particular experimental preparation is of limited use. Thus, an approach, such as the growth of percolation networks, that can statistically reproduce these heterogeneities is of interest.

In Chapter 8 it was shown that the scale-invariant nature of tumor networks helps explain the elevated geometric resistance of these networks, and also predicts the scaling of avascular spaces in tumor tissue. Furthermore, a simple oxygen transport model using a percolation network with a single inlet, a single outlet, and uniform vessel diameters produced results which were in agreement with experimentally observed oxygen transport properties in tumors.

The model presented in Chapter 8 can be further extended in several directions.

- The transport properties of a network can be studied at several stages during its growth. This would help determine if there are particular stages in a network's development during which transport properties are particularly different than other stages.

- Including more than one outlet or inlet. In real networks there is rarely a single outlet or inlet. The effects of multiple inlets or outlets should be determined (e.g., is flow reversal more likely

in one setting or another?).

- Incorporating heterogeneous diameters. In real tumor networks the diameters vary significantly, even along a particular segment. The effect of diameter variability should be explicitly studied.

- Introducing physiologic parameters into the model such as active flow control mechanisms (e.g., vasoconstriction).

By allowing computer simulations of many networks which are different in geometric parameters yet are similarly scale-invariant, the most important determinants of tumor network transport properties may be identified. Such a finding could be important for enhancing drug and nutrient delivery.

Aside from molecular transport, the implications of the scale-invariant nature of vascular networks for heat transfer should also be examined. It is important to examine whether the existence of large avascular spaces in tumors may alter the vessel size which is to be considered as thermally significant. In normal tissue, vessels smaller than 100μm in diameter are considered thermally insignificant. Whether this conclusion holds in the case of tumor tissue, where vessel diameter and spacing vary much more than in normal tissue, needs to be explored.

Appendix A

Source Code

Following are listings of the key computer programs written and used in this research. Fractal dimension calculation programs were written using the THINK Pascal programming language (Symantec; Cupertino, CA) and incorporated into the NIH-Image software[1]. The modeling program was written using FORTRAN 77. The stress-based remodeling program was written using MATLAB matrix manipulation software.

A.1 External Pascal Routines

The following external Pascal routines are referred to in the Pascal programs:

DisposPtr frees the memory used by a given pointer variable.

GetDateTime retrieves the system time in seconds.

IUTimeString formats the system time for display.

PutMessage displays a message using a note alert [86].

PutChar writes a character to NIH-Image's text buffer.

PutReal writes a real variable to NIH-Image's text buffer.

PutString writes a string to NIH-Image's text buffer.

SaveAsText saves NIH-Image's text buffer to a file, based on the information obtained through the SFPutFile routine.

SelectAll defines the whole image as the region of interest.

[1] NIH-Image is a public domain program. The software and all supporting documentation are available by anonymous ftp from zippy.nimh.nih.gov.

SFPutFile brings up Macintosh's file saving dialog box and obtains the file saving information from the user [86].

ShowMessage displays a string in NIH-Image's Info window.

A.2 Box Counting Algorithm

```
procedure DoBoxDimension;
 const
  NUM_BOX_SIZES = 25;
  BOXES_WITH_NO_ITERATIONS = 3;
  INCLUDE_SECONDS = FALSE;
  WIDTH_TOTAL = 10;
  WIDTH_FLOAT = 8;
  MAX_LENGTH = 1024;
 type
  ImageType = array[0..MAX_LENGTH, 0..MAX_LENGTH] of Boolean;                    10
  ImageTypePtr = ^ImageType;
 var
  imageArray: ImageTypePtr;
  vstart, vend, hstart, hend, width, width1, height, height1, hloc,
                vloc, k, j, i, i9, dbmFirst, dbmLast: integer;
  sizeIndex, e, e1, end1, xs, ys, xMargin, yMargin, nBoxX, nBoxY,
                noIterationsCount, dbcFirst, dbcLast: integer;
  x0, y0, cx, cy, xStart, yStart, xEnd, yEnd, maxL, LARGEST_BOX_INDEX,
                    FIRST_BOX_INDEX, LAST_BOX_INDEX: integer;
  longtmp1, longtmp2, e2, occBox, minOccBox, configIdx, timeInSeconds,          20
      dbint, dbdec1, dbdec2, sddbint, sddbdec1, sddbdec2: longint;
  temp1, temp2, xr, yr, dbcmn, sddbcmn: extended;
  sx, sy, st2, db, ss, sxoss, t, intcp, sintcp, sddb,
              chi2, chi2Min, sigdat, dbmax, sddbmax: extended;
  theLine: LineType;
  boxSizes: array[1..NUM_BOX_SIZES] of integer;
  occupiedBoxes: array[1..NUM_BOX_SIZES] of longint;
  lnBoxSizes, lnOccupiedBoxes, localSlope:
              array[1..NUM_BOX_SIZES] of extended;
  smessage, smessage1, timeString, strtemp: Str255;                             30
  where: point;
  reply: SFReply;
  AutoSelectAll: Boolean;
begin
 with info^ do begin
  if BinaryPic then begin
   TextBufSize := 0;
   boxSizes[1] := 512;
   temp1 := 512.0;
   for k := 2 to NUM_BOX_SIZES do begin                                         40
    repeat
     temp2 := 0.793700526 * temp1;
     j := round(temp2);
     temp1 := temp2
    until j <> boxSizes[k - 1];
    boxSizes[k] := j;
   end;

   AutoSelectAll := not RoiShowing;
   if AutoSelectAll then                                                        50
    SelectAll(false);

   if roiType <> RectRoi then begin
    PutMessage('I can only deal with a rectangular ROI!');
```

```
 exit(DoBoxDimension);
end;

with RoiRect do begin
 hend := right;
 vend := bottom;                                                    60
 vstart := top;
 hstart := left;
 width := right - left;
 height := bottom - top;
end;
width1 := width - 1;
height1 := height - 1;
if width > height then
 maxL := width
else                                                                70
 maxL := height;

smessage := concat('Idx', tab, 'Size');
LARGEST_BOX_INDEX := 0;
repeat
 LARGEST_BOX_INDEX := LARGEST_BOX_INDEX + 1;
until boxSizes[LARGEST_BOX_INDEX] < maxL;

imageArray := ImageTypePtr(NewPtr(SizeOf(ImageType)));
if imageArray = nil then begin                                      80
 DisposPtr(ptr(imageArray));
 PutMessage('Insufficient memory');
 exit(DoBoxDimension);
end;
if (height > MAX_LENGTH) or (width > MAX_LENGTH) then begin
 longtmp1 := MAX_LENGTH;
 DisposPtr(ptr(imageArray));
 PutMessage(concat('Image width/length must be less than ',
  long2str(longtmp1), ' for dimension calculation!', cr, 'Change
          MAX_LENGTH constant in source code for larger image.'));   90
 exit(DoBoxDimension);
end;
for vloc := 0 to height1 do begin
 GetLine(hStart, vloc + vstart, width, theLine);
 for hloc := 0 to width1 do begin
  if theLine[hloc] = 0 then
   imageArray^[hloc, vloc] := FALSE
  else
   imageArray^[hloc, vloc] := TRUE;
 end;                                                               100
end;

for k := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
 longtmp1 := k;
 longtmp2 := boxSizes[k];
 if k < 10 then
  smessage := concat(smessage, cr, '0', long2str(longtmp1),
                             tab, long2str(longtmp2))
 else
  smessage := concat(smessage, cr, long2str(longtmp1),              110
                             tab, long2str(longtmp2));
end;

smessage := '';
smessage1 := '';

PutString(concat('Number', tab, 'Box Size', tab, 'Occupied Boxes',
       tab, 'ln [Box Size]', tab, 'ln [Occupied Boxes]', tab,
       'Local Slope', cr));
                                                                    120
GetDateTime(timeZero);
noIterationsCount := 0;
```

```
for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
 noIterationsCount := noIterationsCount + 1;
 e := boxSizes[sizeIndex];
 e1 := e - 1;
 if noIterationsCount > BOXES_WITH_NO_ITERATIONS then
  end1 := e1
 else
  end1 := 0;                                                          130
 longtmp1 := end1 + 1;
 e2 := longtmp1 * longtmp1;
 configIdx := 0;
 longtmp1 := width;
 longtmp2 := height;
 minOccBox := longtmp1 * longtmp2;
 for xs := 0 to end1 do begin
  for ys := 0 to end1 do begin
   configIdx := configIdx + 1;
                                                                      140
   if CommandPeriod then begin
    beep;
    DisposPtr(ptr(imageArray));
    exit(DoBoxDimension);
   end;

   xMargin := width + xs;
   yMargin := height + ys;
   occBox := 0;
                                                                      150
   xr := xMargin / e;
   longtmp1 := trunc(xr);
   temp1 := longtmp1;
   if temp1 = xr then
    nBoxX := longtmp1 - 1
   else
    nBoxX := longtmp1;

   yr := yMargin / e;
   longtmp2 := trunc(yr);                                             160
   temp2 := longtmp2;
   if temp2 = yr then
    nBoxY := longtmp2 - 1
   else
    nBoxY := longtmp2;

   for i := 0 to nBoxX do begin
    for j := 0 to nBoxY do begin
     x0 := i * e;
     y0 := j * e;                                                     170

     cx := x0 - xs;
     if cx < 0 then
      xStart := 0
     else
      xStart := cx;

     cx := x0 - xs + e1;
     if cx > width1 then
      xEnd := width1                                                  180
     else
      xEnd := cx;

     cy := y0 - ys;
     if cy < 0 then
      yStart := 0
     else
      yStart := cy;

     cy := y0 - ys + e1;                                              190
```

```
        if cy > height1 then
         yEnd := height1
        else
         yEnd := cy;

        for vloc := yStart to yEnd do begin
          for hloc := xStart to xEnd do begin
            if imageArray^[hloc, vloc] then begin
              occBox := occBox + 1;
              vloc := yEnd;                                              200
              hloc := xEnd;
             end;
           end;
         end;
       end;
      end;
      if occBox < minOccBox then
       minOccBox := occBox;
     end;
    ShowMessage(concat(smessage, cr, cr, 'Configuration ',               210
            long2str(configIdx), ' out of ', long2str(e2),
            cr, 'Minimum occupied boxes:  ',
            long2str(minOccBox)));
  end;
  occupiedBoxes[sizeIndex] := minOccBox;
  GetDateTime(timeInSeconds);
  timeInSeconds := timeInSeconds - timeZero;
  IUTimeString(timeInSeconds, INCLUDE_SECONDS, timeString);
  longtmp1 := boxSizes[sizeIndex];
  longtmp2 := occupiedBoxes[sizeIndex];                                  220
  smessage := concat(smessage1, cr, long2str(longtmp1), tab,
                long2str(longtmp2));
  ShowMessage(concat('Sz', tab, 'Occupied', smessage, cr,
                'Time:  ', timeString));
  smessage1 := smessage;
  smessage := concat(smessage, cr, 'Time:  ', timeString);
  lnBoxSizes[sizeIndex] := Ln(boxSizes[sizeIndex]);
  lnOccupiedBoxes[sizeIndex] := Ln(occupiedBoxes[sizeIndex]);
 end;
                                                                         230
 for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
  if (sizeIndex > LARGEST_BOX_INDEX + 1) and
    (sizeIndex < NUM_BOX_SIZES - 1) then begin
   sx := 0.0;
   sy := 0.0;
   st2 := 0.0;
   db := 0.0;
   for i9 := sizeIndex - 2 to sizeIndex + 2 do begin
    sx := sx + lnBoxSizes[i9];
    sy := sy + lnOccupiedBoxes[i9];                                      240
   end;
   ss := 5.0; {number of points to calculate local slope }
   sxoss := sx / ss;
   for i9 := sizeIndex - 2 to sizeIndex + 2 do begin
    t := lnBoxSizes[i9] - sxoss;
    st2 := st2 + t * t;
    db := db + t * lnOccupiedBoxes[i9];
   end;
   db := 0.0 - db / st2;
   localSlope[sizeIndex] := db;                                          250
  end
  else
   localSlope[sizeIndex] := 0.0;

  longtmp1 := sizeIndex;
  strtemp := long2str(longtmp1);
  PutString(strtemp);
  PutChar(tab);
```

```
  longtmp1 := boxSizes[sizeIndex];
  strtemp := long2str(longtmp1);                                          260
  PutString(strtemp);
  PutChar(tab);
  longtmp2 := occupiedBoxes[sizeIndex];
  strtemp := long2str(longtmp2);
  PutString(strtemp);
  PutChar(tab);
  PutReal(lnBoxSizes[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
  PutChar(tab);
  PutReal(lnOccupiedBoxes[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
  PutChar(tab);                                                           270
  PutReal(localSlope[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
  PutChar(cr);
 end;

where.v := 100;
where.h := 100;
SFPutFile(where, 'Save the results as?', concat(title, '.box'),
        nil, reply);
if reply.good then
 with reply do                                                            280
  SaveAsText(fname, vRefnum);

TextBufSize := 0;
PutString(concat('No.  of pts.', tab, '1st box no.', tab,
             'Last box no.', tab, '1st box size', tab,
             'Last box size', tab, 'Box Dimension', tab,
             'Std Error', cr));
smessage1 := concat('Pts', tab, 'Dbox');

for j := 7 to NUM_BOX_SIZES - LARGEST_BOX_INDEX + 1 do begin             290
  dbmax := 0.0;
  chi2Min := 9999999999.9;
  for LAST_BOX_INDEX := NUM_BOX_SIZES downto
                  LARGEST_BOX_INDEX + j - 1 do begin
    FIRST_BOX_INDEX := LAST_BOX_INDEX - j + 1;
    sx := 0.0;
    sy := 0.0;
    st2 := 0.0;
    db := 0.0;
    for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin                   300
      sx := sx + lnBoxSizes[i];
      sy := sy + lnOccupiedBoxes[i];
     end;
    ss := LAST_BOX_INDEX - FIRST_BOX_INDEX + 1.0;
    sxoss := sx / ss;

    for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
      t := lnBoxSizes[i] - sxoss;
      st2 := st2 + t * t;
      db := db + t * lnOccupiedBoxes[i];                                  310
     end;
    db := db / st2;
    intcp := (sy - sx * db) / ss;
    sintcp := Sqrt((1.0 + sx * sx / (ss * st2)) / ss);
    sddb := Sqrt(1.0 / st2);
    chi2 := 0.0;

    for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
      temp1 := lnOccupiedBoxes[i] - intcp - db * lnBoxSizes[i];
      chi2 := chi2 + temp1 * temp1;                                       320
     end;
    sigdat := Sqrt(chi2 / (ss - 2.0));
    sintcp := sintcp * sigdat;
    sddb := sddb * sigdat;
    db := 0.0 - db;
    if db > dbmax then begin
```

```
    dbmax := db;
    sddbmax := sddb;
    dbmFirst := FIRST_BOX_INDEX;
    dbmLast := LAST_BOX_INDEX;                                          330
   end;
  if chi2Min > chi2 then begin
    chi2Min := chi2;
    dbcmn := db;
    sddbcmn := sddb;
    dbcFirst := FIRST_BOX_INDEX;
    dbcLast := LAST_BOX_INDEX;
   end;
 end;
                                                                        340
longtmp1 := j;
strtemp := concat('MX  ', long2str(longtmp1));
PutString(strtemp);
PutChar(tab);
longtmp1 := dbmFirst;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp1 := dbmLast;
strtemp := long2str(longtmp1);                                          350
PutString(strtemp);
PutChar(tab);
longtmp1 := boxSizes[dbmFirst];
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp2 := boxSizes[dbmLast];
strtemp := long2str(longtmp2);
PutString(strtemp);
PutChar(tab);                                                           360
PutReal(dbmax, WIDTH_TOTAL, WIDTH_FLOAT);
PutChar(tab);
PutReal(sddbmax, WIDTH_TOTAL, WIDTH_FLOAT);
PutChar(cr);

longtmp1 := j;
strtemp := concat('BS  ', long2str(longtmp1));
PutString(strtemp);
PutChar(tab);
if j < 10 then                                                          370
 strtemp := concat('0', long2str(longtmp1))
else
 strtemp := long2str(longtmp1);
smessage1 := concat(smessage1, cr, strtemp);
longtmp1 := dbcFirst;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp1 := dbcLast;
strtemp := long2str(longtmp1);                                          380
PutString(strtemp);
PutChar(tab);
longtmp1 := boxSizes[dbcFirst];
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp2 := boxSizes[dbcLast];
strtemp := long2str(longtmp2);
PutString(strtemp);
PutChar(tab);                                                           390
PutReal(dbcmn, WIDTH_TOTAL, WIDTH_FLOAT);
PutChar(tab);
PutReal(sddbcmn, WIDTH_TOTAL, WIDTH_FLOAT);
PutChar(cr);
```

```
      dbint := trunc(dbcmn);
      dbdec1 := trunc(dbcmn * 10.0) - dbint * 10;
      dbdec2 := trunc(dbcmn * 100.0) - dbint * 100 - dbdec1 * 10;
      sddbint := trunc(sddbcmn);
      sddbdec1 := trunc(sddbcmn * 10.0) - sddbint * 10;                    400
      sddbdec2 := trunc(sddbcmn * 100.0) - sddbint * 100 -
                sddbdec1 * 10;
      smessage := concat(smessage1, tab, long2str(dbint), '.',
                      long2str(dbdec1), long2str(dbdec2), '±',
                      long2str(sddbint), '.', long2str(sddbdec1),
                      long2str(sddbdec2));
      ShowMessage(smessage);
      smessage1 := smessage;
     end;
                                                                           410
    where.v := 100;
    where.h := 100;
    SFPutFile(where, 'Save the results as?', concat(title, '.slopes'),
            nil, reply);
    if reply.good then
     with reply do
      SaveAsText(fname, vRefnum);

    DisposPtr(ptr(imageArray));
                                                                           420
   end
   else begin
    PutMessage('This is not a binary image!');
    exit(DoBoxDimension);
   end;
  end;
end;
```

A.3 Sandbox Algorithm

```
procedure DoSandDimension;
 label
  1120;
 const
  NUM_BOX_SIZES = 25;
  BOXES_WITH_NO_ITERATIONS = 3;
  INCLUDE_SECONDS = FALSE;
  WIDTH_TOTAL = 10;
  WIDTH_FLOAT = 8;
  MAX_LENGTH = 1024;                                                       10
 type
  ImageType = array[0..MAX_LENGTH, 0..MAX_LENGTH] of Boolean;
  ImageTypePtr = ^ImageType;
 var
  imageArray: ImageTypePtr;
  vstart, vend, hstart, hend, width, width1, height, height1, hloc,
                  vloc, k, j, i, i9, dbmFirst, dbmLast: integer;
  sizeIndex, e, xs, ys, minLengthY, minLengthX, xr, yr, minLength,
                                    dbcFirst, dbcLast: integer;
  x0, y0, xStart, yStart, xEnd, yEnd, LARGEST_BOX_INDEX,                   20
                    FIRST_BOX_INDEX, LAST_BOX_INDEX: integer;
  longtmp1, longtmp2, mass, configIdx, timeInSeconds, timeZero,
                   numParticles, xc, yc, two, numIter: longint;
  dbint, dbdec1, dbdec2, sddbint, sddbdec1, sddbdec2: longint;
  temp1, temp2, dbcmn, sddbcmn: extended;
```

```
 sx, sy, st2, db, ss, sxoss, t, intcp, sintcp, sddb, chi2,
                    chi2Min, sigdat, dbmax, sddbmax: extended;
 theLine: LineType;
 boxSizes: array[1..NUM_BOX_SIZES] of integer;
 totalMass: array[1..NUM_BOX_SIZES] of longint;                              30
 lnBoxSizes, lnTotalMass, localSlope:
          array[1..NUM_BOX_SIZES] of extended;
 smessage, smessage1, timeString, strtemp: Str255;
 where: point;
 reply: SFReply;
 AutoSelectAll: Boolean;
begin
 with info^ do begin
  if BinaryPic then begin
    TextBufSize := 0;                                                        40
    boxSizes[1] := 512;
    temp1 := 512.0;
    for k := 2 to NUM_BOX_SIZES do begin
     repeat
      temp2 := 0.793700526 * temp1;
      j := round(temp2);
      temp1 := temp2
     until j <> boxSizes[k - 1];
     boxSizes[k] := j;
    end;                                                                     50

    AutoSelectAll := not RoiShowing;
    if AutoSelectAll then
     SelectAll(false);

    if roiType <> RectRoi then begin
     PutMessage('I can only deal with a rectangular ROI!');
     exit(DoSandDimension);
    end;
                                                                             60
    with RoiRect do begin
     hend := right;
     vend := bottom;
     vstart := top;
     hstart := left;
     width := right - left;
     height := bottom - top;
    end;
    width1 := width - 1;
    height1 := height - 1;                                                   70

    numParticles := 0;
    smessage := concat('Idx', tab, 'Size');

    imageArray := ImageTypePtr(NewPtr(SizeOf(ImageType)));
    if imageArray = nil then begin
     DisposPtr(ptr(imageArray));
     PutMessage('Insufficient memory');
     exit(DoSandDimension);
    end;                                                                     80
    if (height > MAX_LENGTH) or (width > MAX_LENGTH) then begin
     longtmp1 := MAX_LENGTH;
     DisposPtr(ptr(imageArray));
     PutMessage(concat('Image width/length must be less than ',
             long2str(longtmp1), ' for dimension calculation!',
             cr, 'Change MAX_LENGTH constant in source code for
                    larger image.'));
     exit(DoSandDimension);
    end;
    for vloc := 0 to height1 do begin                                        90
     GetLine(hStart, vloc + vstart, width, theLine);
     for hloc := 0 to width1 do begin
      if theLine[hloc] = 0 then
```

```
        imageArray^[hloc, vloc] := FALSE
       else begin
        numParticles := numParticles + 1;
        imageArray^[hloc, vloc] := TRUE;
       end;
      end;
     end;                                                          100

     GetDateTime(timeZero);

{    Find center of mass      }
     xc := 0;
     yc := 0;
     for vloc := 0 to height1 do begin
      for hloc := 0 to width1 do begin
       if imageArray^[hloc, vloc] then begin
        xc := xc + hloc;                                           110
        yc := yc + vloc;
       end;
      end;
     end;
     xc := round(xc / numParticles);
     yc := round(yc / numParticles);

     if xc < yc then begin
      if xc < width - xc - 1 then begin
       if xc < height - yc - 1 then begin                          120
        minLength := xc;
       end
       else begin
        minLength := height - yc - 1;
       end;
      end
      else begin
       if width - xc - 1 < height - yc - 1 then begin
        minLength := width - xc - 1;
       end                                                         130
       else begin
        minLength := height - yc - 1;
       end;
      end;
     end
     else begin
      if yc < width - xc - 1 then begin
       if yc < height - yc - 1 then begin
        minLength := yc;
       end                                                         140
       else begin
        minLength := height - yc - 1;
       end;
      end
      else begin
       if width - xc - 1 < height - yc - 1 then begin
        minLength := width - xc - 1;
       end
       else begin
        minLength := height - yc - 1;                              150
       end;
      end;
     end;

     minLength := minLength div 2;

     for i := 0 to minLength do begin
      for vloc := -i to i do begin
       for hloc := -i to i do begin
        if imageArray^[(xc + hloc), (yc + vloc)] then begin        160
         xc := xc + hloc;
```

```
          yc := yc + vloc;
          goto 1120;
         end;
        end;
      end;
     end;
    PutMessage('Could not find occupied pixel around center of mass');
    exit(DoSandDimension);
                                                                          170
1120:
    if xc < width - xc - 1 then
     minLengthX := xc
    else
     minLengthX := width - xc - 1;

    if yc < height - yc - 1 then
     minLengthY := yc
    else
     minLengthY := height - yc - 1;                                       180

    if minLengthY < minLengthX then
     minLength := minLengthY
    else
     minLength := minLengthX;

    minLength := minLength div 2;
    minLengthX := minLengthX div 2;
    minLengthY := minLengthY div 2;
                                                                          190
    LARGEST_BOX_INDEX := 1;
    while (boxSizes[LARGEST_BOX_INDEX] >= minLength) do begin
     LARGEST_BOX_INDEX := LARGEST_BOX_INDEX + 1;
    end;

    ShowMessage(concat(long2str(minLength), cr, 'CM: ', long2str(xc),
                ',', long2str(yc)));

    for k := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
     longtmp1 := k;                                                       200
     longtmp2 := boxSizes[k];
     if k < 10 then
      smessage := concat(smessage, cr, '0', long2str(longtmp1),
                  tab, long2str(longtmp2))
     else
      smessage := concat(smessage, cr, long2str(longtmp1), tab,
                  long2str(longtmp2));
    end;

    smessage := '';                                                       210
    smessage1 := '';

    PutString(concat('Number', tab, 'Box Size', tab, 'Total Mass',
              tab, 'ln [Box Size]', tab, 'ln [Total Mass]',
              tab, 'Local Slope', cr));

{   Find mass per radius   }

    for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
     totalMass[sizeIndex] := 0;                                           220
     lnBoxSizes[sizeIndex] := Ln(2 * boxSizes[sizeIndex] + 1);
    end;

    two := 2;
    numIter := (two * minLengthX + 1) * (two * minLengthY + 1);
    configIdx := 0;
    for yr := (yc - minLengthY) to (yc + minLengthY) do begin
     for xr := (xc - minLengthX) to (xc + minLengthX) do begin
      if imageArray^[xr, yr] then begin
```

```
    configIdx := configIdx + 1;
    for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
     mass := 0;
     yStart := yr - boxSizes[sizeIndex];
     yEnd := yr + boxSizes[sizeIndex];
     xStart := xr - boxSizes[sizeIndex];
     xEnd := xr + boxSizes[sizeIndex];
     for vloc := yStart to yEnd do begin
      for hloc := xStart to xEnd do begin
       if imageArray^[hloc, vloc] then
        mass := mass + 1;
      end;
     end;
     totalMass[sizeIndex] := totalMass[sizeIndex] + mass;
    end;
   end;
  end;

  GetDateTime(timeInSeconds);
  timeInSeconds := timeInSeconds - timeZero;
  IUTimeString(timeInSeconds, INCLUDE_SECONDS, timeString);
  ShowMessage(concat('Time:   ', timeString, cr, 'Iteration:   ',
     long2str(configIdx), cr, 'Area:   ', long2str(numIter)));

 end;

for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
 lnTotalMass[sizeIndex] := Ln(totalMass[sizeIndex] / configIdx);
end;


for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
 if (sizeIndex > LARGEST_BOX_INDEX + 1) and
    (sizeIndex < NUM_BOX_SIZES - 1) then begin
  sx := 0.0;
  sy := 0.0;
  st2 := 0.0;
  db := 0.0;
  for i9 := sizeIndex - 2 to sizeIndex + 2 do begin
   sx := sx + lnBoxSizes[i9];
   sy := sy + lnTotalMass[i9];
  end;
  ss := 5.0; {number of points to calculate local slope }
  sxoss := sx / ss;
  for i9 := sizeIndex - 2 to sizeIndex + 2 do begin
   t := lnBoxSizes[i9] - sxoss;
   st2 := st2 + t * t;
   db := db + t * lnTotalMass[i9];
  end;
  db := db / st2;
  localSlope[sizeIndex] := db;
 end
 else
  localSlope[sizeIndex] := 0.0;

 longtmp1 := sizeIndex;
 strtemp := long2str(longtmp1);
 PutString(strtemp);
 PutChar(tab);
 longtmp1 := 2 * boxSizes[sizeIndex] + 1;
 strtemp := long2str(longtmp1);
 PutString(strtemp);
 PutChar(tab);
 longtmp2 := totalMass[sizeIndex];
 strtemp := long2str(longtmp2);
 PutString(strtemp);
 PutChar(tab);
 PutReal(lnBoxSizes[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
```

```
 PutChar(tab);
 PutReal(lnTotalMass[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
 PutChar(tab);                                                          300
 PutReal(localSlope[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
 PutChar(cr);
end;

where.v := 100;
where.h := 100;
SFPutFile(where, 'Save the results as?', concat(title, '.sand'),
       nil, reply);
if reply.good then
 with reply do                                                          310
 SaveAsText(fname, vRefnum);

TextBufSize := 0;
PutString(concat('No.  of pts.', tab, '1st box no.', tab,
            'Last box no.', tab, '1st box size', tab,
            'Last box size', tab, 'Sandbox Dimension',
            tab, 'Std Error', cr));
smessage1 := concat('Pts', tab, 'Dbox');

for j := 7 to NUM_BOX_SIZES - LARGEST_BOX_INDEX + 1 do begin            320
 dbmax := 0.0;
 chi2Min := 9999999999.9;
 for LAST_BOX_INDEX := NUM_BOX_SIZES downto
                LARGEST_BOX_INDEX + j - 1 do begin
  FIRST_BOX_INDEX := LAST_BOX_INDEX - j + 1;
  sx := 0.0;
  sy := 0.0;
  st2 := 0.0;
  db := 0.0;
  for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin                   330
   sx := sx + lnBoxSizes[i];
   sy := sy + lnTotalMass[i];
  end;
  ss := LAST_BOX_INDEX - FIRST_BOX_INDEX + 1.0;
  sxoss := sx / ss;

  for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
   t := lnBoxSizes[i] - sxoss;
   st2 := st2 + t * t;
   db := db + t * lnTotalMass[i];                                       340
  end;
  db := db / st2;
  intcp := (sy - sx * db) / ss;
  sintcp := Sqrt((1.0 + sx * sx / (ss * st2)) / ss);
  sddb := Sqrt(1.0 / st2);
  chi2 := 0.0;

  for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
   temp1 := lnTotalMass[i] - intcp - db * lnBoxSizes[i];
   chi2 := chi2 + temp1 * temp1;                                        350
  end;
  sigdat := Sqrt(chi2 / (ss - 2.0));
  sintcp := sintcp * sigdat;
  sddb := sddb * sigdat;
  if db > dbmax then begin
   dbmax := db;
   sddbmax := sddb;
   dbmFirst := FIRST_BOX_INDEX;
   dbmLast := LAST_BOX_INDEX;
  end;                                                                  360
  if chi2Min > chi2 then begin
   chi2Min := chi2;
   dbcmn := db;
   sddbcmn := sddb;
   dbcFirst := FIRST_BOX_INDEX;
```

```
   dbcLast := LAST_BOX_INDEX;
  end;
 end;

longtmp1 := j;                                                          370
strtemp := concat('MX ', long2str(longtmp1));
PutString(strtemp);
PutChar(tab);
longtmp1 := dbmFirst;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp1 := dbmLast;
strtemp := long2str(longtmp1);
PutString(strtemp);                                                     380
PutChar(tab);
longtmp1 := 2 * boxSizes[dbmFirst] + 1;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp2 := 2 * boxSizes[dbmLast] + 1;
strtemp := long2str(longtmp2);
PutString(strtemp);
PutChar(tab);
PutReal(dbmax, WIDTH_TOTAL, WIDTH_FLOAT);                               390
PutChar(tab);
PutReal(sddbmax, WIDTH_TOTAL, WIDTH_FLOAT);
PutChar(cr);

longtmp1 := j;
strtemp := concat('BS ', long2str(longtmp1));
PutString(strtemp);
PutChar(tab);
if j < 10 then
 strtemp := concat('0', long2str(longtmp1))                             400
else
 strtemp := long2str(longtmp1);
smessage1 := concat(smessage1, cr, strtemp);
longtmp1 := dbcFirst;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp1 := dbcLast;
strtemp := long2str(longtmp1);
PutString(strtemp);                                                     410
PutChar(tab);
longtmp1 := 2 * boxSizes[dbcFirst] + 1;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp2 := 2 * boxSizes[dbcLast] + 1;
strtemp := long2str(longtmp2);
PutString(strtemp);
PutChar(tab);
PutReal(dbcmn, WIDTH_TOTAL, WIDTH_FLOAT);                               420
PutChar(tab);
PutReal(sddbcmn, WIDTH_TOTAL, WIDTH_FLOAT);
PutChar(cr);


dbint := trunc(dbcmn);
dbdec1 := trunc(dbcmn * 10.0) - dbint * 10;
dbdec2 := trunc(dbcmn * 100.0) - dbint * 100 - dbdec1 * 10;
sddbint := trunc(sddbcmn);
sddbdec1 := trunc(sddbcmn * 10.0) - sddbint * 10;                       430
sddbdec2 := trunc(sddbcmn * 100.0) - sddbint * 100 -
          sddbdec1 * 10;
smessage := concat(smessage1, tab, long2str(dbint), '.',
```

```
                long2str(dbdec1), long2str(dbdec2), '±',
                long2str(sddbint), '.', long2str(sddbdec1),
                long2str(sddbdec2));
    ShowMessage(smessage);
    smessage1 := smessage;
   end;
                                                                          440
   where.v := 100;
   where.h := 100;
   SFPutFile(where, 'Save the results as?', concat(title, '.slopes'),
         nil, reply);
   if reply.good then
    with reply do
     SaveAsText(fname, vRefnum);

   DisposPtr(ptr(imageArray));
                                                                          450
  end
  else begin
   PutMessage('This is not a binary image!');
   exit(DoSandDimension);
  end;
 end;
end;
```

A.4 Correlation Algorithm

```
procedure DoCorrDimension;
 label
  1120;
 const
  NUM_BOX_SIZES = 25;
  EMBEDDING_DIM = 2;
  INCLUDE_SECONDS = FALSE;
  WIDTH_TOTAL = 10;
  WIDTH_FLOAT = 8;
  MAX_LENGTH = 1024;                                                      10
 type
  ImageType = array[0..MAX_LENGTH, 0..MAX_LENGTH] of Boolean;
  ImageTypePtr = ^ImageType;
 var
  imageArray: ImageTypePtr;
  vstart, vend, hstart, hend, width, width1, height, height1, hloc,
            vloc, k, j, i, i9, dbmFirst, dbmLast: integer;
  sizeIndex, e, xs, ys, minLengthY, minLengthX, xr, yr, minLength,
                              dbcFirst, dbcLast: integer;
  x0, y0, xStart, yStart, xEnd, yEnd, LARGEST_BOX_INDEX,                  20
                FIRST_BOX_INDEX, LAST_BOX_INDEX: integer;
  longtmp1, longtmp2, mass, configIdx, timeInSeconds, timeZero,
              numParticles, xc, yc, two, numIter: longint;
  dbint, dbdec1, dbdec2, sddbint, sddbdec1, sddbdec2: longint;
  temp1, temp2, dbcmn, sddbcmn, distance: extended;
  sx, sy, st2, db, ss, sxoss, t, intcp, sintcp, sddb, chi2, chi2Min,
                          sigdat, dbmax, sddbmax: extended;
  theLine: LineType;
  boxSizes: array[1..NUM_BOX_SIZES] of integer;
  totalMass: array[1..NUM_BOX_SIZES] of longint;                          30
  lnBoxSizes, lnTotalMass, localSlope, normalizationFactor:
                    array[1..NUM_BOX_SIZES] of extended;
  smessage, smessage1, timeString, strtemp: Str255;
```

```
 where: point;
 reply: SFReply;
 AutoSelectAll: Boolean;
begin
 with info^ do begin
  if BinaryPic then begin
   TextBufSize := 0;                                                    40
   boxSizes[1] := 512;
   temp1 := 512.0;
   for k := 2 to NUM_BOX_SIZES do begin
    repeat
     temp2 := 0.793700526 * temp1;
     j := round(temp2);
     temp1 := temp2
    until j <> boxSizes[k - 1];
    boxSizes[k] := j;
   end;                                                                 50

   AutoSelectAll := not RoiShowing;
   if AutoSelectAll then
    SelectAll(false);

   if roiType <> RectRoi then begin
    PutMessage('I can only deal with a rectangular ROI!');
    exit(DoCorrDimension);
   end;
                                                                        60
   with RoiRect do begin
    hend := right;
    vend := bottom;
    vstart := top;
    hstart := left;
    width := right - left;
    height := bottom - top;
   end;
   width1 := width - 1;
   height1 := height - 1;                                               70

   numParticles := 0;
   smessage := concat('Idx', tab, 'Size');

   imageArray := ImageTypePtr(NewPtr(SizeOf(ImageType)));
   if imageArray = nil then begin
    DisposPtr(ptr(imageArray));
    PutMessage('Insufficient memory');
    exit(DoCorrDimension);
   end;                                                                 80
   if (height > MAX_LENGTH) or (width > MAX_LENGTH) then begin
    longtmp1 := MAX_LENGTH;
    DisposPtr(ptr(imageArray));
    PutMessage(concat('Image width/length must be less than ',
            long2str(longtmp1), ' for dimension calculation!',
            cr, 'Change MAX_LENGTH constant in source code for
                  larger image.'));
    exit(DoCorrDimension);
   end;
   for vloc := 0 to height1 do begin                                    90
    GetLine(hStart, vloc + vstart, width, theLine);
    for hloc := 0 to width1 do begin
     if theLine[hloc] = 0 then
      imageArray^[hloc, vloc] := FALSE
     else begin
      numParticles := numParticles + 1;
      imageArray^[hloc, vloc] := TRUE;
     end;
    end;
   end;                                                                 100
```

```
    GetDateTime(timeZero);

{   Find center of mass      }
    xc := 0;
    yc := 0;
    for vloc := 0 to height1 do begin
      for hloc := 0 to width1 do begin
        if imageArray^[hloc, vloc] then begin
          xc := xc + hloc;                                                    110
          yc := yc + vloc;
        end;
      end;
    end;
    xc := round(xc / numParticles);
    yc := round(yc / numParticles);

    if xc < yc then begin
      if xc < width - xc - 1 then begin
        if xc < height - yc - 1 then begin                                    120
          minLength := xc;
        end
        else begin
          minLength := height - yc - 1;
        end;
      end
      else begin
        if width - xc - 1 < height - yc - 1 then begin
          minLength := width - xc - 1;
        end                                                                   130
        else begin
          minLength := height - yc - 1;
        end;
      end;
    end
    else begin
      if yc < width - xc - 1 then begin
        if yc < height - yc - 1 then begin
          minLength := yc;
        end                                                                   140
        else begin
          minLength := height - yc - 1;
        end;
      end
      else begin
        if width - xc - 1 < height - yc - 1 then begin
          minLength := width - xc - 1;
        end
        else begin
          minLength := height - yc - 1;                                       150
        end;
      end;
    end;

    minLength := minLength div 2;

    for i := 0 to minLength do begin
      for vloc := -i to i do begin
        for hloc := -i to i do begin
          if imageArray^[(xc + hloc), (yc + vloc)] then begin                 160
            xc := xc + hloc;
            yc := yc + vloc;
            goto 1120;
          end;
        end;
      end;
    end;
    PutMessage('Could not find occupied pixel around center of mass');
    exit(DoCorrDimension);
```

```
                                                                        170
1120:
    if xc < width - xc - 1 then
     minLengthX := xc
    else
     minLengthX := width - xc - 1;

    if yc < height - yc - 1 then
     minLengthY := yc
    else
     minLengthY := height - yc - 1;                                     180

    if minLengthY < minLengthX then
     minLength := minLengthY
    else
     minLength := minLengthX;

    minLength := minLength div 2;
    minLengthX := minLengthX div 2;
    minLengthY := minLengthY div 2;
                                                                        190
    LARGEST_BOX_INDEX := 1;
    while (boxSizes[LARGEST_BOX_INDEX] >= trunc(minLength / 1.1))
     do begin
      LARGEST_BOX_INDEX := LARGEST_BOX_INDEX + 1;
     end;

    ShowMessage(concat(long2str(minLength), cr, 'CM: ',
                long2str(xc), ',', long2str(yc)));

    for k := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin                200
      longtmp1 := k;
      longtmp2 := boxSizes[k];
      if k < 10 then
       smessage := concat(smessage, cr, '0', long2str(longtmp1),
                   tab, long2str(longtmp2))
      else
       smessage := concat(smessage, cr, long2str(longtmp1), tab,
                   long2str(longtmp2));
     end;
                                                                        210
    smessage := '';
    smessage1 := '';

    PutString(concat('Number', tab, 'Radius', tab, 'Total Mass',
           tab, 'ln [Radius]', tab, 'ln [Norm Mass]', tab,
           'Local Slope', cr));

{    Find mass per ring    }

    for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin        220
      totalMass[sizeIndex] := 0;
      lnBoxSizes[sizeIndex] := Ln(2 * boxSizes[sizeIndex] + 1);
     end;

    two := 2;
    numIter := (two * minLengthX + 1) * (two * minLengthY + 1);
    configIdx := 0;
    for yr := (yc - minLengthY) to (yc + minLengthY) do begin
      for xr := (xc - minLengthX) to (xc + minLengthX) do begin
        if imageArray^[xr, yr] then begin                               230
          configIdx := configIdx + 1;
          for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
            mass := 0;
            yStart := yr - round(1.1 * boxSizes[sizeIndex]);
            yEnd := yr + round(1.1 * boxSizes[sizeIndex]);
            xStart := xr - round(1.1 * boxSizes[sizeIndex]);
            xEnd := xr + round(1.1 * boxSizes[sizeIndex]);
```

```
      for vloc := yStart to yEnd do begin
        for hloc := xStart to xEnd do begin
          if imageArray^[hloc, vloc] then begin                                    240
            distance := sqrt(1.0 * (hloc - xr) * (hloc - xr) +
                             1.0 * (vloc - yr) * (vloc - yr));
            if ((distance <= 1.1 * boxSizes[sizeIndex]) and
               (distance >= 1.0 * boxSizes[sizeIndex])) then
              mass := mass + 1;
            end;
          end;
        end;
        totalMass[sizeIndex] := totalMass[sizeIndex] + mass;
      end;                                                                         250
    end;
  end;

  GetDateTime(timeInSeconds);
  timeInSeconds := timeInSeconds - timeZero;
  IUTimeString(timeInSeconds, INCLUDE_SECONDS, timeString);
  ShowMessage(concat('Time:   ', timeString, cr, 'Iteration:   ',
      long2str(configIdx), cr, 'Area:   ', long2str(numIter)));

 end;                                                                              260

for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
  mass := 0;
  yStart := -round(1.1 * boxSizes[sizeIndex]);
  yEnd := round(1.1 * boxSizes[sizeIndex]);
  xStart := -round(1.1 * boxSizes[sizeIndex]);
  xEnd := round(1.1 * boxSizes[sizeIndex]);
  for vloc := yStart to yEnd do begin
    for hloc := xStart to xEnd do begin
      distance := sqrt(1.0 * hloc * hloc + 1.0 * vloc * vloc);                     270
      if ((distance <= 1.1 * boxSizes[sizeIndex]) and (distance >= 1.0 * boxSizes[sizeIndex])) then
        mass := mass + 1;
    end;
  end;
  normalizationFactor[sizeIndex] := 1.0 * mass;
 end;


for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
  lnTotalMass[sizeIndex] := Ln(totalMass[sizeIndex] /                              280
          (configIdx * normalizationFactor[sizeIndex]));
 end;

for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
  if (sizeIndex > LARGEST_BOX_INDEX + 1) and
     (sizeIndex < NUM_BOX_SIZES - 1) then begin
    sx := 0.0;
    sy := 0.0;
    st2 := 0.0;
    db := 0.0;                                                                     290
    for i9 := sizeIndex - 2 to sizeIndex + 2 do begin
      sx := sx + lnBoxSizes[i9];
      sy := sy + lnTotalMass[i9];
    end;
    ss := 5.0; {number of points to calculate local slope }
    sxoss := sx / ss;
    for i9 := sizeIndex - 2 to sizeIndex + 2 do begin
      t := lnBoxSizes[i9] - sxoss;
      st2 := st2 + t * t;
      db := db + t * lnTotalMass[i9];                                              300
    end;
    db := db / st2;
    localSlope[sizeIndex] := db + EMBEDDING_DIM;
  end
  else
```

```
  localSlope[sizeIndex] := 0.0;

  longtmp1 := sizeIndex;
  strtemp := long2str(longtmp1);
  PutString(strtemp);                                                    310
  PutChar(tab);
  longtmp1 := boxSizes[sizeIndex];
  strtemp := long2str(longtmp1);
  PutString(strtemp);
  PutChar(tab);
  longtmp2 := totalMass[sizeIndex];
  strtemp := long2str(longtmp2);
  PutString(strtemp);
  PutChar(tab);
  PutReal(lnBoxSizes[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);              320
  PutChar(tab);
  PutReal(lnTotalMass[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
  PutChar(tab);
  PutReal(localSlope[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
  PutChar(cr);
 end;

where.v := 100;
where.h := 100;
SFPutFile(where, 'Save the results as?', concat(title, '.corr'),       330
        nil, reply);
if reply.good then
 with reply do
  SaveAsText(fname, vRefnum);

TextBufSize := 0;
PutString(concat('No.  of pts.', tab, '1st rad no.', tab,
             'Last rad no.', tab, '1st rad size', tab,
             'Last rad size', tab, 'Corr Dimension', tab,
             'Std Error', cr));                                        340
smessage1 := concat('Pts', tab, 'Dbox');

for j := 7 to NUM_BOX_SIZES - LARGEST_BOX_INDEX + 1 do begin
  dbmax := 0.0;
  chi2Min := 9999999999.9;
  for LAST_BOX_INDEX := NUM_BOX_SIZES downto
     LARGEST_BOX_INDEX + j - 1 do begin
    FIRST_BOX_INDEX := LAST_BOX_INDEX - j + 1;
    sx := 0.0;
    sy := 0.0;                                                           350
    st2 := 0.0;
    db := 0.0;
    for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
      sx := sx + lnBoxSizes[i];
      sy := sy + lnTotalMass[i];
     end;
    ss := LAST_BOX_INDEX - FIRST_BOX_INDEX + 1.0;
    sxoss := sx / ss;

    for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin                  360
      t := lnBoxSizes[i] - sxoss;
      st2 := st2 + t * t;
      db := db + t * lnTotalMass[i];
     end;
    db := db / st2;
    intcp := (sy - sx * db) / ss;
    sintcp := Sqrt((1.0 + sx * sx / (ss * st2)) / ss);
    sddb := Sqrt(1.0 / st2);
    chi2 := 0.0;
                                                                         370
    for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
      temp1 := lnTotalMass[i] - intcp - db * lnBoxSizes[i];
      chi2 := chi2 + temp1 * temp1;
```

```
  end;
  sigdat := Sqrt(chi2 / (ss - 2.0));
  sintcp := sintcp * sigdat;
  sddb := sddb * sigdat;

  db := db + EMBEDDING_DIM;
  if db > dbmax then begin                                          380
    dbmax := db;
    sddbmax := sddb;
    dbmFirst := FIRST_BOX_INDEX;
    dbmLast := LAST_BOX_INDEX;
   end;
  if chi2Min > chi2 then begin
    chi2Min := chi2;
    dbcmn := db;
    sddbcmn := sddb;
    dbcFirst := FIRST_BOX_INDEX;                                    390
    dbcLast := LAST_BOX_INDEX;
   end;
 end;

longtmp1 := j;
strtemp := concat('MX ', long2str(longtmp1));
PutString(strtemp);
PutChar(tab);
longtmp1 := dbmFirst;
strtemp := long2str(longtmp1);                                      400
PutString(strtemp);
PutChar(tab);
longtmp1 := dbmLast;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp1 := boxSizes[dbmFirst];
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);                                                       410
longtmp2 := boxSizes[dbmLast];
strtemp := long2str(longtmp2);
PutString(strtemp);
PutChar(tab);
PutReal(dbmax, WIDTH_TOTAL, WIDTH_FLOAT);
PutChar(tab);
PutReal(sddbmax, WIDTH_TOTAL, WIDTH_FLOAT);
PutChar(cr);

longtmp1 := j;                                                      420
strtemp := concat('BS ', long2str(longtmp1));
PutString(strtemp);
PutChar(tab);
if j < 10 then
 strtemp := concat('0', long2str(longtmp1))
else
 strtemp := long2str(longtmp1);
smessage1 := concat(smessage1, cr, strtemp);
longtmp1 := dbcFirst;
strtemp := long2str(longtmp1);                                      430
PutString(strtemp);
PutChar(tab);
longtmp1 := dbcLast;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp1 := boxSizes[dbcFirst];
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);                                                       440
longtmp2 := boxSizes[dbcLast];
```

```
    strtemp := long2str(longtmp2);
    PutString(strtemp);
    PutChar(tab);
    PutReal(dbcmn, WIDTH_TOTAL, WIDTH_FLOAT);
    PutChar(tab);
    PutReal(sddbcmn, WIDTH_TOTAL, WIDTH_FLOAT);
    PutChar(cr);

                                                                          450
    dbint := trunc(dbcmn);
    dbdec1 := trunc(dbcmn * 10.0) - dbint * 10;
    dbdec2 := trunc(dbcmn * 100.0) - dbint * 100 - dbdec1 * 10;
    sddbint := trunc(sddbcmn);
    sddbdec1 := trunc(sddbcmn * 10.0) - sddbint * 10;
    sddbdec2 := trunc(sddbcmn * 100.0) - sddbint * 100 -
            sddbdec1 * 10;
    smessage := concat(smessage1, tab, long2str(dbint), '.',
                    long2str(dbdec1), long2str(dbdec2), '±',
                    long2str(sddbint), '.', long2str(sddbdec1),          460
                    long2str(sddbdec2));
    ShowMessage(smessage);
    smessage1 := smessage;
   end;

   where.v := 100;
   where.h := 100;
   SFPutFile(where, 'Save the results as?', concat(title, '.slopes'),
          nil, reply);
   if reply.good then                                                     470
    with reply do
     SaveAsText(fname, vRefnum);

   DisposPtr(ptr(imageArray));

  end
  else begin
   PutMessage('This is not a binary image!');
   exit(DoCorrDimension);
  end;                                                                    480
 end;
end;
```

A.5 Minimum-Path Algorithms

A.5.1 Minimum-Path Identification

```
procedure DoFindMinPath;
 const
  MAX_LENGTH = 640;
 type
  ImageType = array[0..MAX_LENGTH, 0..MAX_LENGTH] of integer;
  ImageTypePtr = ^ImageType;
 var
  vstart, vend, hstart, hend, width, width1, width2, height, height1,
                                  height2, hloc, vloc: integer;
  stepIndex, breakPoint, x, y, xe, ye, xs, ys: integer;                   10
  longtmp1: longint;
  imageArray: ImageTypePtr;
  theLine: LineType;
  AutoSelectAll, BreakThrough, noProgress: Boolean;
begin
```

```
with info^ do begin
 if BinaryPic then begin

   AutoSelectAll := not RoiShowing;
   if AutoSelectAll then                                                    20
    SelectAll(false);

   if roiType <> RectRoi then begin
    PutMessage('I can only deal with a rectangular ROI!');
    exit(DoFindMinPath);
   end;

   with RoiRect do begin
    hend := right;
    vend := bottom;                                                         30
    vstart := top;
    hstart := left;
    width := right - left;
    width1 := width - 1;
    width2 := width - 2;
    height := bottom - top;
    height1 := height - 1;
    height2 := height - 2;
   end;
                                                                            40
   imageArray := ImageTypePtr(NewPtr(SizeOf(ImageType)));
   if imageArray = nil then begin
    DisposPtr(ptr(imageArray));
    PutMessage('Insufficient memory');
    exit(DoFindMinPath);
   end;

   for vloc := 0 to height1 do begin
    for hloc := 0 to width1 do begin
     imageArray^[hloc, vloc] := 0;                                          50
    end;
   end;

   noProgress := TRUE;
   BreakThrough := FALSE;
   stepIndex := 1;
   GetLine(hstart, vstart, width, theLine);
   for hloc := 0 to width1 do begin
    if theLine[hloc] = 255 then begin
     imageArray^[hloc, 0] := stepIndex;                                     60
    end;
   end;

   repeat
    vloc := 0;
    hloc := 0;
    if imageArray^[hloc, vloc] = stepIndex then begin
     GetLine(hstart, vloc + vstart, width, theLine);
     if (theLine[hloc + 1] = 255) and
        (imageArray^[hloc + 1, vloc] = 0) then                              70
      imageArray^[hloc + 1, vloc] := stepIndex + 1;
     GetLine(hstart, vloc + vstart + 1, width, theLine);
     if (theLine[hloc] = 255) and (imageArray^[hloc, vloc + 1] = 0) then
      imageArray^[hloc, vloc + 1] := stepIndex + 1;
     if (theLine[hloc + 1] = 255) and
        (imageArray^[hloc + 1, vloc + 1] = 0) then
      imageArray^[hloc + 1, vloc + 1] := stepIndex + 1;
     noProgress := FALSE;
    end;
                                                                            80
    for hloc := 1 to width2 do begin
     if imageArray^[hloc, vloc] = stepIndex then begin
      GetLine(hstart, vloc + vstart, width, theLine);
```

```
    if (theLine[hloc − 1] = 255) and
      (imageArray^[hloc − 1, vloc] = 0) then
     imageArray^[hloc − 1, vloc] := stepIndex + 1;
    if (theLine[hloc + 1] = 255) and
      (imageArray^[hloc + 1, vloc] = 0) then
     imageArray^[hloc + 1, vloc] := stepIndex + 1;

    GetLine(hstart, vloc + vstart + 1, width, theLine);
    if (theLine[hloc − 1] = 255) and
      (imageArray^[hloc − 1, vloc + 1] = 0) then
     imageArray^[hloc − 1, vloc + 1] := stepIndex + 1;
    if (theLine[hloc] = 255) and (imageArray^[hloc, vloc + 1] = 0) then
     imageArray^[hloc, vloc + 1] := stepIndex + 1;
    if (theLine[hloc + 1] = 255) and
      (imageArray^[hloc + 1, vloc + 1] = 0) then
     imageArray^[hloc + 1, vloc + 1] := stepIndex + 1;
    noProgress := FALSE;
   end;
  end;

 hloc := width1;
 if imageArray^[hloc, vloc] = stepIndex then begin
   GetLine(hstart, vloc + vstart, width, theLine);
   if (theLine[hloc − 1] = 255) and
     (imageArray^[hloc − 1, vloc] = 0) then
    imageArray^[hloc − 1, vloc] := stepIndex + 1;
   GetLine(hstart, vloc + vstart + 1, width, theLine);
   if (theLine[hloc − 1] = 255) and
     (imageArray^[hloc − 1, vloc + 1] = 0) then
    imageArray^[hloc − 1, vloc + 1] := stepIndex + 1;
   if (theLine[hloc] = 255) and (imageArray^[hloc, vloc + 1] = 0) then
    imageArray^[hloc, vloc + 1] := stepIndex + 1;
   noProgress := FALSE;
  end;

 for vloc := 1 to height2 do begin
   hloc := 0;
   if imageArray^[hloc, vloc] = stepIndex then begin
     GetLine(hstart, vloc + vstart, width, theLine);
     if (theLine[hloc + 1] = 255) and
       (imageArray^[hloc + 1, vloc] = 0) then
      imageArray^[hloc + 1, vloc] := stepIndex + 1;
     GetLine(hstart, vloc + vstart − 1, width, theLine);
     if (theLine[hloc] = 255) and (imageArray^[hloc, vloc − 1] = 0) then
      imageArray^[hloc, vloc − 1] := stepIndex + 1;
     if (theLine[hloc + 1] = 255) and
       (imageArray^[hloc + 1, vloc − 1] = 0) then
      imageArray^[hloc + 1, vloc − 1] := stepIndex + 1;
     GetLine(hstart, vloc + vstart + 1, width, theLine);
     if (theLine[hloc] = 255) and (imageArray^[hloc, vloc + 1] = 0) then
      imageArray^[hloc, vloc + 1] := stepIndex + 1;
     if (theLine[hloc + 1] = 255) and
       (imageArray^[hloc + 1, vloc + 1] = 0) then
      imageArray^[hloc + 1, vloc + 1] := stepIndex + 1;
     noProgress := FALSE;
    end;
   for hloc := 1 to width2 do begin
     if imageArray^[hloc, vloc] = stepIndex then begin
       GetLine(hstart, vloc + vstart, width, theLine);
       if (theLine[hloc − 1] = 255) and
         (imageArray^[hloc − 1, vloc] = 0) then
        imageArray^[hloc − 1, vloc] := stepIndex + 1;
       if (theLine[hloc + 1] = 255) and
         (imageArray^[hloc + 1, vloc] = 0) then
        imageArray^[hloc + 1, vloc] := stepIndex + 1;

       GetLine(hstart, vloc + vstart − 1, width, theLine);
       if (theLine[hloc − 1] = 255) and
```

```
        (imageArray^[hloc - 1, vloc - 1] = 0) then
       imageArray^[hloc - 1, vloc - 1] := stepIndex + 1;
      if (theLine[hloc] = 255) and
        (imageArray^[hloc, vloc - 1] = 0) then
       imageArray^[hloc, vloc - 1] := stepIndex + 1;
      if (theLine[hloc + 1] = 255) and
        (imageArray^[hloc + 1, vloc - 1] = 0) then
       imageArray^[hloc + 1, vloc - 1] := stepIndex + 1;
                                                                          160
      GetLine(hstart, vloc + vstart + 1, width, theLine);
      if (theLine[hloc - 1] = 255) and
        (imageArray^[hloc - 1, vloc + 1] = 0) then
       imageArray^[hloc - 1, vloc + 1] := stepIndex + 1;
      if (theLine[hloc] = 255) and
        (imageArray^[hloc, vloc + 1] = 0) then
       imageArray^[hloc, vloc + 1] := stepIndex + 1;
      if (theLine[hloc + 1] = 255) and
        (imageArray^[hloc + 1, vloc + 1] = 0) then
       imageArray^[hloc + 1, vloc + 1] := stepIndex + 1;                  170
      noProgress := FALSE;
     end;
    end;
   hloc := width1;
   if imageArray^[hloc, vloc] = stepIndex then begin
     GetLine(hstart, vloc + vstart, width, theLine);
     if (theLine[hloc - 1] = 255) and
       (imageArray^[hloc - 1, vloc] = 0) then
      imageArray^[hloc - 1, vloc] := stepIndex + 1;
     GetLine(hstart, vloc + vstart - 1, width, theLine);                  180
     if (theLine[hloc - 1] = 255) and
       (imageArray^[hloc - 1, vloc - 1] = 0) then
      imageArray^[hloc - 1, vloc - 1] := stepIndex + 1;
     if (theLine[hloc] = 255) and (imageArray^[hloc, vloc - 1] = 0) then
      imageArray^[hloc, vloc - 1] := stepIndex + 1;
     GetLine(hstart, vloc + vstart + 1, width, theLine);
     if (theLine[hloc - 1] = 255) and
       (imageArray^[hloc - 1, vloc + 1] = 0) then
      imageArray^[hloc - 1, vloc + 1] := stepIndex + 1;
     if (theLine[hloc] = 255) and (imageArray^[hloc, vloc + 1] = 0) then  190
      imageArray^[hloc, vloc + 1] := stepIndex + 1;
     noProgress := FALSE;
    end;
  end;

  longtmp1 := stepIndex;
  ShowMessage(concat('stepIndex=', long2str(longtmp1)));
  if noProgress then begin
    DisposPtr(ptr(imageArray));
    PutMessage('Was not able to break through.');                       200
    exit(DoFindMinPath);
  end;
  stepIndex := stepIndex + 1;
  vloc := height1;
  for hloc := 0 to width1 do begin
    if imageArray^[hloc, vloc] <> 0 then begin
      BreakThrough := TRUE;
      ShowMessage('Breakthrough');
      breakPoint := hloc;
      hloc := width1;                                                   210
    end;
  end;
until BreakThrough;

for vloc := 0 to height1 do begin
  for hloc := 0 to width1 do begin
    theLine[hloc] := 0;
  end;
  PutLine(hstart, vloc + vstart, width, theLine);
```

```
    end;

    vloc := height1;
    hloc := breakPoint;
    stepIndex := imageArray^[hloc, vloc] - 1;
    imageArray^[hloc, vloc] := -1;

    while vloc > 0 do begin
      xs := hloc - 1;
      if xs < 0 then
        xs := 0;
      ys := vloc - 1;
      if ys < 0 then
        ys := 0;
      xe := hloc + 1;
      if xe > width1 then
        xe := width1;
      ye := vloc + 1;
      if ye > height1 then
        ye := height1;
      for y := ys to ye do begin
        for x := xs to xe do begin
          if imageArray^[x, y] = stepIndex then begin
            imageArray^[x, y] := -1;
            stepIndex := stepIndex - 1;
            hloc := x;
            vloc := y;
            y := ye;
            x := xe;
          end;
        end;
      end;

    KillRoi;
    for vloc := 0 to height1 do begin
      for hloc := 0 to width1 do begin
        if imageArray^[hloc, vloc] = -1 then
          theLine[hloc] := 255
        else
          theLine[hloc] := 0;
      end;
      PutLine(hstart, vloc + vstart, width, theLine);
    end;

    UpdatePicWindow;

    DisposPtr(ptr(imageArray));
    ShowMessage('OK');
   end
  else begin
    PutMessage('This is not a binary image!');
    exit(DoFindMinPath);
   end;
 end;
end;
```

A.5.2 Minimum-Path Dimension

```
procedure DoMinPathDim;
 const
  NUM_BOX_SIZES = 25;
  INCLUDE_SECONDS = FALSE;
  WIDTH_TOTAL = 10;
  WIDTH_FLOAT = 8;
```

```
 MAX_LENGTH = 1024;
type
 ImageType = array[0..MAX_LENGTH, 0..MAX_LENGTH] of Boolean;
 ImageTypePtr = ^ImageType;                                                    10
var
 imageArray: ImageTypePtr;
 vstart, vend, hstart, hend, width, width1, height, height1, hloc, vloc,
                          k, j, i, i9, dbmFirst, dbmLast: integer;
 sizeIndex, x, y, neighbors, initPoints, pointIndex, pointLoc, dbcFirst,
                                                  dbcLast: integer;
 x0, y0, xStart, yStart, xEnd, yEnd, maxL, LARGEST_BOX_INDEX,
                        FIRST_BOX_INDEX, LAST_BOX_INDEX: integer;
 longtmp1, longtmp2, e2, occBox, minOccBox, configIdx, timeInSeconds,
                                      timeZero, dx, dy: longint;               20
 dbint, dbdec1, dbdec2, sddbint, sddbdec1, sddbdec2: longint;
 temp1, temp2, r1, r2, eSize, dbcmn, sddbcmn: extended;
 sx, sy, st2, db, ss, sxoss, t, intcp, sintcp, sddb, chi2, chi2Min,
                              sigdat, dbmax, sddbmax: extended;
 theLine: LineType;
 minPath: array[0..MAX_LENGTH, 0..1] of integer;
 boxSizes: array[1..NUM_BOX_SIZES] of integer;
 occupiedBoxes: array[1..NUM_BOX_SIZES] of longint;
 lnBoxSizes, lnOccupiedBoxes, localSlope:
          array[1..NUM_BOX_SIZES] of extended;                                 30
 smessage, smessage1, timeString, strtemp: Str255;
 where: point;
 reply: SFReply;
 AutoSelectAll, doneAll: Boolean;
begin
 with info^ do begin
  if BinaryPic then begin
    TextBufSize := 0;
    boxSizes[1] := 512;
    temp1 := 512.0;                                                            40
    for k := 2 to NUM_BOX_SIZES do begin
      repeat
       temp2 := 0.793700526 * temp1;
       j := round(temp2);
       temp1 := temp2
      until j <> boxSizes[k - 1];
      boxSizes[k] := j;
     end;

    AutoSelectAll := not RoiShowing;                                           50
    if AutoSelectAll then
     SelectAll(false);

    if roiType <> RectRoi then begin
      PutMessage('I can only deal with a rectangular ROI!');
      exit(DoMinPathDim);
     end;

    with RoiRect do begin
      hend := right;                                                           60
      vend := bottom;
      vstart := top;
      hstart := left;
      width := right - left;
      height := bottom - top;
     end;
    width1 := width - 1;
    height1 := height - 1;
    if width > height then
     maxL := width                                                             70
    else
     maxL := height;

    smessage := concat('Idx', tab, 'Size');
```

```
LARGEST_BOX_INDEX := 0;
repeat
 LARGEST_BOX_INDEX := LARGEST_BOX_INDEX + 1;
until boxSizes[LARGEST_BOX_INDEX] < maxL;

imageArray := ImageTypePtr(NewPtr(SizeOf(ImageType)));                    80
if imageArray = nil then begin
  DisposPtr(ptr(imageArray));
  PutMessage('Insufficient memory');
  exit(DoMinPathDim);
 end;
if (height > MAX_LENGTH) or (width > MAX_LENGTH) then begin
  longtmp1 := MAX_LENGTH;
  DisposPtr(ptr(imageArray));
  PutMessage(concat('Image width/length must be less than ',
         long2str(longtmp1), ' for dimension calculation!', cr,           90
         'Change MAX_LENGTH constant in source code for larger
                image.'));
  exit(DoMinPathDim);
 end;

for vloc := 0 to height1 do begin
  GetLine(hStart, vloc + vstart, width, theLine);
  for hloc := 0 to width1 do begin
    if theLine[hloc] = 0 then
     imageArray^[hloc, vloc] := FALSE                                     100
    else
     imageArray^[hloc, vloc] := TRUE;
   end;
 end;

initPoints := 0;
for vloc := 0 to height1 do begin
  for hloc := 0 to width1 do begin
    if imageArray^[hloc, vloc] then begin
      xStart := hloc - 1;                                                 110
      if xStart < 0 then
       xStart := 0;
      yStart := vloc - 1;
      if yStart < 0 then
       yStart := 0;
      xEnd := hloc + 1;
      if xEnd > width1 then
       xEnd := width1;
      yEnd := vloc + 1;
      if yEnd > height1 then                                              120
       yEnd := height1;
      neighbors := -1;
      for y := yStart to yEnd do begin
        for x := xStart to xEnd do begin
          if imageArray^[x, y] then
           neighbors := neighbors + 1;
        end;
       end;
      if (neighbors < 1) or (neighbors > 2) then begin
        DisposPtr(ptr(imageArray));                                       130
        PutMessage('Minimum path must be a continuous
                          skeletonized line!');
       exit(DoMinPathDim);
       end
      else if neighbors = 1 then
       if initPoints = 0 then begin
         minPath[0, 0] := hloc;
         minPath[0, 1] := vloc;
         initPoints := 1;
        end                                                               140
       else begin
         initPoints := initPoints + 1;
```

```
        if initPoints > 2 then begin
          DisposPtr(ptr(imageArray));
          PutMessage('Minimum path must be a continuous line!');
          exit(DoMinPathDim);
         end
       end;
     end;
   end;                                                                    150
end;

pointIndex := 0;
repeat
 doneAll := TRUE;
 hloc := minPath[pointIndex, 0];
 vloc := minPath[pointIndex, 1];
 imageArray^[hloc, vloc] := FALSE;
 xStart := hloc - 1;
 if xStart < 0 then                                                      160
  xStart := 0;
 yStart := vloc - 1;
 if yStart < 0 then
  yStart := 0;
 xEnd := hloc + 1;
 if xEnd > width1 then
  xEnd := width1;
 yEnd := vloc + 1;
 if yEnd > height1 then
  yEnd := height1;                                                       170
 for y := yStart to yEnd do begin
   for x := xStart to xEnd do begin
     if imageArray^[x, y] then begin
       doneAll := FALSE;
       pointIndex := pointIndex + 1;
       minPath[pointIndex, 0] := x;
       minPath[pointIndex, 1] := y;
       y := yEnd;
       x := xEnd;
      end;                                                                180
    end;
  end;
until doneAll;

longtmp1 := pointIndex;
ShowMessage(concat('Points:   ', long2str(longtmp1)));

for k := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
  longtmp1 := k;
  longtmp2 := boxSizes[k];                                               190
  if k < 10 then
   smessage := concat(smessage, cr, '0', long2str(longtmp1),
                tab, long2str(longtmp2))
  else
   smessage := concat(smessage, cr, long2str(longtmp1), tab,
                long2str(longtmp2));
end;

smessage := '';
smessage1 := '';                                                         200

PutString(concat('Number', tab, 'Box Size', tab, 'Occupied Boxes',
            tab, 'ln [Box Size]', tab, 'ln [Occupied Boxes]',
            tab, 'Local Slope', cr));

GetDateTime(timeZero);
for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
  eSize := 1.0 * boxSizes[sizeIndex];
  if CommandPeriod then begin
    beep;                                                                210
```

```
  DisposPtr(ptr(imageArray));
  exit(DoMinPathDim);
 end;

minOccBox := MAX_LENGTH + 1;

for configIdx := 0 to pointIndex do begin
  occBox := 0;
  pointLoc := configIdx;
                                                                    220
  for i := configIdx downto 0 do begin
    dx := minPath[i, 0] - minPath[pointLoc, 0];
    dy := minPath[i, 1] - minPath[pointLoc, 1];
    r1 := sqrt(dx * dx + dy * dy);
    if r1 >= eSize then begin
      dx := minPath[i + 1, 0] - minPath[pointLoc, 0];
      dy := minPath[i + 1, 1] - minPath[pointLoc, 1];
      r2 := sqrt(dx * dx + dy * dy);
      if (r1 - eSize) < (eSize - r2) then begin
        pointLoc := i;                                              230
        occBox := occBox + 1;
       end
      else begin
        pointLoc := i + 1;
        i := i + 1;
        occBox := occBox + 1;
       end
     end;
   end;
  if pointLoc > 0 then                                              240
   occBox := occBox + 1;

  pointLoc := configIdx;
  for i := configIdx to pointIndex do begin
    dx := minPath[i, 0] - minPath[pointLoc, 0];
    dy := minPath[i, 1] - minPath[pointLoc, 1];
    r1 := sqrt(dx * dx + dy * dy);
    if r1 >= eSize then begin
      dx := minPath[i - 1, 0] - minPath[pointLoc, 0];
      dy := minPath[i - 1, 1] - minPath[pointLoc, 1];               250
      r2 := sqrt(dx * dx + dy * dy);
      if (r1 - eSize) < (eSize - r2) then begin
        pointLoc := i;
        occBox := occBox + 1;
       end
      else begin
        pointLoc := i - 1;
        i := i - 1;
        occBox := occBox + 1;
       end                                                          260
     end;
   end;
  if pointLoc < pointIndex then
   occBox := occBox + 1;

  if occBox < minOccBox then
   minOccBox := occBox;
 end;

occupiedBoxes[sizeIndex] := minOccBox;                              270
GetDateTime(timeInSeconds);
timeInSeconds := timeInSeconds - timeZero;
IUTimeString(timeInSeconds, INCLUDE_SECONDS, timeString);
longtmp1 := boxSizes[sizeIndex];
longtmp2 := occupiedBoxes[sizeIndex];
smessage := concat(smessage1, cr, long2str(longtmp1),
                   tab, long2str(longtmp2));
ShowMessage(concat('Sz', tab, 'Occupied', smessage, cr,
```

```
                'Time:   ', timeString));
  smessage1 := smessage;                                                    280
  smessage := concat(smessage, cr, 'Time:   ', timeString);
  lnBoxSizes[sizeIndex] := Ln(boxSizes[sizeIndex]);
  lnOccupiedBoxes[sizeIndex] := Ln(occupiedBoxes[sizeIndex]);
 end;

for sizeIndex := LARGEST_BOX_INDEX to NUM_BOX_SIZES do begin
  if (sizeIndex > LARGEST_BOX_INDEX + 1) and
     (sizeIndex < NUM_BOX_SIZES - 1) then begin
    sx := 0.0;
    sy := 0.0;                                                              290
    st2 := 0.0;
    db := 0.0;
    for i9 := sizeIndex - 2 to sizeIndex + 2 do begin
      sx := sx + lnBoxSizes[i9];
      sy := sy + lnOccupiedBoxes[i9];
     end;
    ss := 5.0; {number of points to calculate local slope }
    sxoss := sx / ss;
    for i9 := sizeIndex - 2 to sizeIndex + 2 do begin
      t := lnBoxSizes[i9] - sxoss;                                          300
      st2 := st2 + t * t;
      db := db + t * lnOccupiedBoxes[i9];
     end;
    db := 0.0 - db / st2;
    localSlope[sizeIndex] := db;
   end
  else
   localSlope[sizeIndex] := 0.0;

  longtmp1 := sizeIndex;                                                    310
  strtemp := long2str(longtmp1);
  PutString(strtemp);
  PutChar(tab);
  longtmp1 := boxSizes[sizeIndex];
  strtemp := long2str(longtmp1);
  PutString(strtemp);
  PutChar(tab);
  longtmp2 := occupiedBoxes[sizeIndex];
  strtemp := long2str(longtmp2);
  PutString(strtemp);                                                       320
  PutChar(tab);
  PutReal(lnBoxSizes[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
  PutChar(tab);
  PutReal(lnOccupiedBoxes[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
  PutChar(tab);
  PutReal(localSlope[sizeIndex], WIDTH_TOTAL, WIDTH_FLOAT);
  PutChar(cr);
 end;

where.v := 100;                                                             330
where.h := 100;
SFPutFile(where, 'Save the results as?', concat(title, '.box'),
        nil, reply);
if reply.good then
 with reply do
  SaveAsText(fname, vRefnum);

TextBufSize := 0;
PutString(concat('No.  of pts.', tab, '1st box no.', tab, 'Last box no.',
             tab, '1st box size', tab, 'Last box size', tab, 'Box                340
                    Dimension', tab, 'Std Error', cr));
smessage1 := concat('Pts', tab, 'Dbox');

for j := 7 to NUM_BOX_SIZES - LARGEST_BOX_INDEX + 1 do begin
  chi2Min := 9999999999.9;
  dbmax := 0.0;
```

```
for LAST_BOX_INDEX := NUM_BOX_SIZES downto
                LARGEST_BOX_INDEX + j - 1 do begin
 FIRST_BOX_INDEX := LAST_BOX_INDEX - j + 1;
 sx := 0.0;                                                          350
 sy := 0.0;
 st2 := 0.0;
 db := 0.0;
 for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
  sx := sx + lnBoxSizes[i];
  sy := sy + lnOccupiedBoxes[i];
 end;
 ss := LAST_BOX_INDEX - FIRST_BOX_INDEX + 1.0;
 sxoss := sx / ss;
                                                                     360
 for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
  t := lnBoxSizes[i] - sxoss;
  st2 := st2 + t * t;
  db := db + t * lnOccupiedBoxes[i];
 end;
 db := db / st2;
 intcp := (sy - sx * db) / ss;
 sintcp := Sqrt((1.0 + sx * sx / (ss * st2)) / ss);
 sddb := Sqrt(1.0 / st2);
 chi2 := 0.0;                                                        370

 for i := FIRST_BOX_INDEX to LAST_BOX_INDEX do begin
  temp1 := lnOccupiedBoxes[i] - intcp - db * lnBoxSizes[i];
  chi2 := chi2 + temp1 * temp1;
 end;
 sigdat := Sqrt(chi2 / (ss - 2.0));
 sintcp := sintcp * sigdat;
 sddb := sddb * sigdat;
 db := 0.0 - db;
 if db > dbmax then begin                                            380
  dbmax := db;
  sddbmax := sddb;
  dbmFirst := FIRST_BOX_INDEX;
  dbmLast := LAST_BOX_INDEX;
 end;
 if chi2Min > chi2 then begin
  chi2Min := chi2;
  dbcmn := db;
  sddbcmn := sddb;
  dbcFirst := FIRST_BOX_INDEX;                                       390
  dbcLast := LAST_BOX_INDEX;
 end;
end;

longtmp1 := j;
strtemp := concat('MX ', long2str(longtmp1));
PutString(strtemp);
PutChar(tab);
longtmp1 := dbmFirst;
strtemp := long2str(longtmp1);                                       400
PutString(strtemp);
PutChar(tab);
longtmp1 := dbmLast;
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);
longtmp1 := boxSizes[dbmFirst];
strtemp := long2str(longtmp1);
PutString(strtemp);
PutChar(tab);                                                        410
longtmp2 := boxSizes[dbmLast];
strtemp := long2str(longtmp2);
PutString(strtemp);
PutChar(tab);
```

```
    PutReal(dbmax, WIDTH_TOTAL, WIDTH_FLOAT);
    PutChar(tab);
    PutReal(sddbmax, WIDTH_TOTAL, WIDTH_FLOAT);
    PutChar(cr);

    longtmp1 := j;                                                          420
    strtemp := concat('BS ', long2str(longtmp1));
    PutString(strtemp);
    PutChar(tab);
    if j < 10 then
     strtemp := concat('0', long2str(longtmp1))
    else
     strtemp := long2str(longtmp1);
    smessage1 := concat(smessage1, cr, strtemp);
    longtmp1 := dbcFirst;
    strtemp := long2str(longtmp1);                                          430
    PutString(strtemp);
    PutChar(tab);
    longtmp1 := dbcLast;
    strtemp := long2str(longtmp1);
    PutString(strtemp);
    PutChar(tab);
    longtmp1 := boxSizes[dbcFirst];
    strtemp := long2str(longtmp1);
    PutString(strtemp);
    PutChar(tab);                                                           440
    longtmp2 := boxSizes[dbcLast];
    strtemp := long2str(longtmp2);
    PutString(strtemp);
    PutChar(tab);
    PutReal(dbcmn, WIDTH_TOTAL, WIDTH_FLOAT);
    PutChar(tab);
    PutReal(sddbcmn, WIDTH_TOTAL, WIDTH_FLOAT);
    PutChar(cr);

    dbint := trunc(dbcmn);                                                  450
    dbdec1 := trunc(dbcmn * 10.0) - dbint * 10;
    dbdec2 := trunc(dbcmn * 100.0) - dbint * 100 - dbdec1 * 10;
    sddbint := trunc(sddbcmn);
    sddbdec1 := trunc(sddbcmn * 10.0) - sddbint * 10;
    sddbdec2 := trunc(sddbcmn * 100.0) - sddbint * 100 - sddbdec1 * 10;
    smessage := concat(smessage1, tab, long2str(dbint), '.',
              long2str(dbdec1), long2str(dbdec2), '±',
              long2str(sddbint), '.', long2str(sddbdec1),
              long2str(sddbdec2));
    ShowMessage(smessage);                                                  460
    smessage1 := smessage;
   end;

   where.v := 100;
   where.h := 100;
   SFPutFile(where, 'Save the results as?', concat(title, '.slopes'),
          nil, reply);
   if reply.good then
    with reply do
     SaveAsText(fname, vRefnum);                                            470

   DisposPtr(ptr(imageArray));

  end
 else begin
  PutMessage('This is not a binary image!');
  exit(DoMinPathDim);
  end;
 end;
end;                                                                        480
```

A.6 2-D Growth Modeling Program

```
      PROGRAM DoModelTime

      REAL*4 RNUNF

*     Assign parameters:
      INTEGER*4 AUTO_ITERATIONS,MAX_LENGTH,MAX_LENGTH1
      PARAMETER (AUTO_ITERATIONS=100,MAX_LENGTH=128,MAX_LENGTH1=127)
      INTEGER*4 MAX_TIME,MAX_NUM_PARTICLES
      PARAMETER (MAX_TIME=1000000,MAX_NUM_PARTICLES=100000)
      INTEGER*4 NO_PROD_RADIUS                                          10
      PARAMETER (NO_PROD_RADIUS=10)
      INTEGER*4 NUM_BOX_SIZES
      PARAMETER (NUM_BOX_SIZES=25)
      INTEGER*4 MAX_PATH_LENGTH
      PARAMETER (MAX_PATH_LENGTH=1000)

*     Declare image-size variables:
      INTEGER*4 width,width1,width2,height,height1,height2
      REAL*4 imageArea
                                                                        20
*     Declare multi-purpose variables:
      INTEGER*4 vloc,hloc,xTemp,yTemp,xNo,yNo,yT,xT
      INTEGER*4 k,i,j
      INTEGER*4 longtmp(0:3)
      LOGICAL*4 bTemp1,bTemp2
      REAL*4 numerator,denominator
      INTEGER*4 compTime,tArray(1:3)

*     Declare model parameters:
      INTEGER*4 halfLife,growthWaitUnits,intL                           30
      INTEGER*4 numGroups,growthThreshold,releaseThreshold
      LOGICAL*4 homogeneousSub,homogeneousDif,centerSeed
      REAL*4 productionRate,stickingProb,extraRelease,growthRange
      REAL*4 releaseAtGrowthSite,tau,fintL
      LOGICAL*4 waitUnit,uniformParticleDist

*     Declare model variables:
      INTEGER*4 xCntr,yCntr,release,numRelease,autoCounter
      INTEGER*4 timeUnits,hitUnits,numDecayed,numParticles,occNeighbors
      INTEGER*4 totalTime,highestParticle,numNew,emptyParticle          40
      INTEGER*4 totalParts,tissueParts,extraParticles,growthSites
      REAL*4 decayProb,weakest,frac
      LOGICAL*4 BreakThrough,keepWaiting,foundEmpty
      INTEGER*4 furthest,bestDistance,bestMass

*     Declare final image variables:
      INTEGER*4 pGroupSize,pAge

*     Declare arrays:
      INTEGER*4 imageArray(0:MAX_LENGTH1,0:MAX_LENGTH1)                 50
      REAL*4 strengthArrayDif(0:MAX_LENGTH1,0:MAX_LENGTH1)
      REAL*4 strengthArraySub(0:MAX_LENGTH1,0:MAX_LENGTH1)
      INTEGER*4 hitArray(0:MAX_LENGTH1,0:MAX_LENGTH1)
      INTEGER*4 xParticle(1:MAX_NUM_PARTICLES)
      INTEGER*4 yParticle(1:MAX_NUM_PARTICLES)
      INTEGER*4 ageParticle(1:MAX_NUM_PARTICLES)
      LOGICAL*4 activeParticle(1:MAX_NUM_PARTICLES)

*     Declare result variables:
      INTEGER*4 bestImage(0:MAX_LENGTH1,0:MAX_LENGTH1)                  60
      INTEGER*4 heavyImage(0:MAX_LENGTH1,0:MAX_LENGTH1)
      REAL*4 dimList(0:AUTO_ITERATIONS)
      REAL*4 sdDimList(0:AUTO_ITERATIONS)
      REAL*4 mpdimList(0:AUTO_ITERATIONS)
```

```
      REAL*4 sdMpdimList(0:AUTO_ITERATIONS)
      REAL*4 timeMassList(0:AUTO_ITERATIONS)
      REAL*4 tissueEffList(0:AUTO_ITERATIONS)
      REAL*4 totalEffList(0:AUTO_ITERATIONS)
      INTEGER*4 timeList(0:AUTO_ITERATIONS),massList(0:AUTO_ITERATIONS)
      INTEGER*4 tissuePartList(0:AUTO_ITERATIONS)                              70
      INTEGER*4 totalPartList(0:AUTO_ITERATIONS)
      INTEGER*4 decayPartList(0:AUTO_ITERATIONS)
      REAL*4 timeMassAvg,totalEffAvg,tissueEffAvg,timeAvg,massAvg
      REAL*4 tissuePartAvg,totalPartAvg,decayPartAvg
      REAL*4 timeMassSD,totalEffSD,tissueEffSD,timeSD,massSD
      REAL*4 tissuePartSD,totalPartSD,decayPartSD
      REAL*4 mpdim,sdmpdim

*     Declare mass-radius variables:
      INTEGER*4 sizeIndex,xStart,yStart,xEnd,yEnd,mass,xc,yc                   80
      INTEGER*4 xr,yr,minLength
      INTEGER*4 LARGEST_BOX_INDEX,FIRST_BOX_INDEX,LAST_BOX_INDEX
      REAL*4 temp1,temp2
      REAL*4 sx,sy,st2,db,ss,sxoss,t,sddb,chi2,sigdat,intcp
      INTEGER*4 boxSizes(1:NUM_BOX_SIZES),totalMass(1:NUM_BOX_SIZES)
      REAL*4 lnBoxSizes(1:NUM_BOX_SIZES),lnTotalMass(1:NUM_BOX_SIZES)

*     Declare minimum path variables:
      LOGICAL*4 noProgress
      INTEGER*4 minPathArray(0:MAX_LENGTH1,0:MAX_LENGTH1)                      90
      INTEGER*4 stepIndex,vstart,vend,hstart,hend
      INTEGER*4 breakPoint,x,y,xe,ye,xs,ys
      INTEGER*4 minPath(0:MAX_PATH_LENGTH,0:1)
      INTEGER*4 pointIndex,pointLoc,occBox,configIdx
      INTEGER*4 dx,dy
      REAL*4 eSize,r1

      CALL ITIME(tArray)
      compTime=tArray(3)+tArray(2)*60+tArray(1)*3600
                                                                               100
*     Initialize parameters:
      intL=32768
      fintL=32768.0

*     Initialize size variables:
      width = MAX_LENGTH
      width1 = width - 1
      width2 = width - 2
      height = MAX_LENGTH
      height1 = height - 1                                                     110
      height2 = height - 2

*     Initialize dimension measurement parameters:
      boxSizes(1) = 512
      temp1 = 512.0
      DO 5 k=2,NUM_BOX_SIZES
    4 temp2 = 0.793700526*temp1
      j = NINT(temp2)
      temp1 = temp2
      IF (j.EQ.boxSizes(k-1)) GO TO 4                                          120
      boxSizes(k) = j
    5 CONTINUE


*     Assign Parameters:
      homogeneousSub = .TRUE.
      homogeneousDif = .TRUE.
      centerSeed = .TRUE.
      waitUnit = .TRUE.
      growthWaitUnits = 10                                                     130
      halfLife = 1000000
      tau = FLOAT(halfLife)/ALOG(2.0)
```

```
      numGroups = 250
      uniformParticleDist = .FALSE.
      imageArea = FLOAT(height)*FLOAT(width)
      productionRate = 0.000056234133
      stickingProb = 1.0
      extraRelease = 0.0
      releaseAtGrowthSite = 0.0
      growthThreshold = 1                                                    140
      releaseThreshold = 1
      growthRange = 1.0

      OPEN (UNIT=15,FILE='Model.Specs',STATUS='NEW')
      WRITE (15,2200) 'Model Specifications:'
      IF (homogeneousSub) THEN
       WRITE (15,2200) 'Homogeneous Substrate'
      ELSE
       WRITE (15,2200) 'Inhomogeneous Substrate'
      ENDIF                                                                  150
      IF (homogeneousDif) THEN
       WRITE (15,2200) 'Homogeneous Diffusion'
      ELSE
       WRITE (15,2200) 'Inhomogeneous Diffusion'
      ENDIF
      IF (centerSeed) THEN
       WRITE (15,2200) 'Center Seed'
      ELSE
       WRITE (15,2200) 'Multiple Seeds'
      ENDIF                                                                  160
      IF (waitUnit) THEN
       WRITE (15,2200) 'Wait Unit:  TIME'
      ELSE
       WRITE (15,2200) 'Wait Unit:  HIT'
      ENDIF
      WRITE (15,2210) 'Growth Wait Units :',growthWaitUnits
      WRITE (15,2210) 'Particle Half-life:',halfLife
      IF (uniformParticleDist) THEN
       WRITE (15,2200) 'Uniform particle origin'
      ELSE                                                                   170
       WRITE (15,2200) 'Particle origin beyond network'
      ENDIF
      WRITE (15,2220) 'Production Rate:  ',productionRate,
     C                      'particles/pixel*time'
      WRITE (15,2230) 'Interaction Probability:',stickingProb
      WRITE (15,2230) 'Extra release per hit:',extraRelease
      WRITE (15,2230) 'Extra release at growth site:',
     C                       releaseAtGrowthSite
      WRITE (15,2210) 'Growth threshold:',growthThreshold
      WRITE (15,2210) 'Extra release threshold:',releaseThreshold            180
      WRITE (15,2230) 'Growth Range:    0.0  -',growthRange
      CLOSE (UNIT=15)

      OPEN (UNIT=16,FILE='Model.Data',STATUS='NEW')
      WRITE (16,2270)
     C    ' No      Dm-r          Dmp         T/M    tis-E',
     C    '   tot-E   Time  Mass    Tis    Tot   Dec'

      bestDistance = MAX_LENGTH/2
      bestMass = 0                                                           190

*     Seeding code:
      CALL RNOPT(6)
      CALL RNSET(0)
*     End of seeding code

*     Initialize iteration counter:
      autoCounter = 0
   10 autoCounter = autoCounter + 1
                                                                             200
```

```
*    Assign growth substrate strengths:
     IF (.NOT.homogeneousSub) THEN
       PRINT *,'Generating random substrate strengths...'
       DO 110 vloc=0,height1
        DO 100 hloc=0,width1
         strengthArraySub(hloc,vloc)= RNUNF()
 100    CONTINUE
 110    CONTINUE                                                    210
       PRINT *,'Random growth substrate generated.'
     ELSE
       DO 130 vloc=0,height1
        DO 120 hloc=0,width1
         strengthArraySub(hloc,vloc)=0.5
 120    CONTINUE
 130    CONTINUE
     ENDIF

                                                                     220
*    Assign diffusion substrate strengths:
     IF (.NOT.homogeneousDif) THEN
       PRINT *,'Generating random diffusion strengths...'
       DO 160 vloc=0,height1
        DO 150 hloc=0,width1
         strengthArrayDif(hloc,vloc)= RNUNF()
 150    CONTINUE
 160    CONTINUE
       PRINT *,'Random diffusion substrate generated.'
     ENDIF                                                           230

*    Initialize model variables:

     DO 200 i=1,MAX_NUM_PARTICLES
      activeParticle(i)=.FALSE.
 200 CONTINUE

     BreakThrough = .FALSE.
     numDecayed = 0                                                  240
     totalTime = 0
     totalParts = 0
     tissueParts = 0
     extraParticles = 0

     DO 220 vloc=0,height1
      DO 210 hloc=0,width1
       imageArray(hloc,vloc)=0
 210  CONTINUE
 220 CONTINUE                                                        250

     IF (centerSeed) THEN
      yCntr = height/2
      xCntr = width/2
      numParticles = 1
      imageArray(xCntr,yCntr)=numParticles

      IF (.NOT.uniformParticleDist) THEN
       DO 250 yNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
        yT = yNo + yCntr                                             260
        IF (yT.GT.height1) yT=yT-height
        IF (yT.LT.0) yT=height-yT
        DO 240 xNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
         xT = xNo + xCntr
         IF (xT.GT.width1) xT=xT-width
         IF (xT.LT.0) xT=width-xT
         IF (SQRT(FLOAT(xNo*xNo+yNo*yNo)).LT.NO_PROD_RADIUS) THEN
          IF (imageArray(xT,yT).EQ.0) imageArray(xT,yT)=-1
```

```
        ENDIF
240     CONTINUE                                                        270
250    CONTINUE
      ENDIF
      ELSE
       PRINT *,'Only center-seed option is available!'
       STOP
      ENDIF


*     ###### Main loop until BreakThrough ######
      DO 1000 WHILE (.NOT.BreakThrough)                                 280
       keepWaiting=.TRUE.
       timeUnits=0
       hitUnits=0

      DO 310 vloc=0,height1
       DO 300 hloc=0,width1
        hitArray(hloc,vloc)=0
300    CONTINUE
310   CONTINUE
                                                                        290
       highestParticle=0
       DO 320 i=1,MAX_NUM_PARTICLES
       IF (activeParticle(i)) highestParticle=i
320   CONTINUE

*      Loop waiting for time for growth:
       DO 700 WHILE (keepWaiting)

       timeUnits=timeUnits+1
       totalTime=totalTime+1                                            300

*      Advance existing particles:
       DO 400 i=1,highestParticle
        IF (activeParticle(i)) THEN

         ageParticle(i)=ageParticle(i)+1

*        Takes into account only the case where homogeneousDif=TRUE:
        k=AINT(4.0*RNUNF())+1
        GO TO (350,360,370,380), k                                      310
350      xTemp = xParticle(i) + 1
        yTemp = yParticle(i)
        IF (xTemp.EQ.width) xTemp=0
        GO TO 390
360      xTemp = xParticle(i) - 1
        yTemp = yParticle(i)
        IF (xTemp.EQ.-1) xTemp=width1
        GO TO 390
370      yTemp = yParticle(i) - 1
        xTemp = xParticle(i)                                            320
        IF (yTemp.EQ.-1) yTemp=height1
        GO TO 390
380      yTemp = yParticle(i) + 1
        xTemp = xParticle(i)
        IF (yTemp.EQ.height) yTemp=0
390     IF (imageArray(xTemp,yTemp).LE.0) THEN
        xParticle(i)=xTemp
        yParticle(i)=yTemp
       ENDIF
      ENDIF                                                             330
400   CONTINUE

*      Produce new particles:

       numNew=AINT(productionRate*imageArea)
       frac=productionRate*imageArea-1.0*FLOAT(numNew)
```

```
      IF (RNUNF().LT.frac) numNew=numNew+1
      emptyParticle = 1
      DO 460 i=1,numNew
       xTemp = NINT(RNUNF()*width1)                                    340
       yTemp = NINT(RNUNF()*height1)
       IF (imageArray(xTemp,yTemp).EQ.0) THEN
        foundEmpty = .FALSE.
        k = emptyParticle - 1
        DO 450 WHILE (k.LT.MAX_NUM_PARTICLES)
         k = k + 1
         IF (.NOT.activeParticle(k)) THEN
          emptyParticle = k
          foundEmpty = .TRUE.
          k = MAX_NUM_PARTICLES                                        350
         ENDIF
  450   END DO

        IF (.NOT.foundEmpty) THEN
         PRINT *,'Too Many Particles!'
         STOP
        ENDIF

        tissueParts = tissueParts + 1
        totalParts = totalParts + 1                                    360
        activeParticle(emptyParticle)=.TRUE.
        xParticle(emptyParticle)=xTemp
        yParticle(emptyParticle)=yTemp
        ageParticle(emptyParticle)=1
        IF (emptyParticle.GT.highestParticle)
     C             highestParticle=emptyParticle
        ENDIF
  460   CONTINUE

                                                                       370
*       Degrade old particles:

        DO 500 i=1,highestParticle
         IF (activeParticle(i)) THEN
          decayProb = EXP(-FLOAT(ageParticle(i))/tau)/tau
          IF (RNUNF().LT.decayProb) THEN
           numDecayed = numDecayed + 1
           activeParticle(i)=.FALSE.
          ENDIF
         ENDIF                                                         380
  500   CONTINUE

*       Mark hits:
        DO 600 i=1,highestParticle
         IF (activeParticle(i)) THEN
          xTemp = xParticle(i)
          yTemp = yParticle(i)
          bTemp1 = ((xTemp.LT.width1).AND.(yTemp.LT.height1))
          bTemp2 = ((xTemp.GT.0).AND.(yTemp.GT.0))
          IF (bTemp1.AND.bTemp2) THEN                                  390
           longtmp(0)=AINT(FLOAT(intL-1+imageArray((xTemp+1),yTemp))/
     C                                              fintL)
           longtmp(1)=AINT(FLOAT(intL-1+imageArray((xTemp-1),yTemp))/
     C                                              fintL)
           longtmp(2)=AINT(FLOAT(intL-1+imageArray(xTemp,(yTemp+1)))/
     C                                              fintL)
           longtmp(3)=AINT(FLOAT(intL-1+imageArray(xTemp,(yTemp-1)))/
     C                                              fintL)
           occNeighbors=longtmp(0)+longtmp(1)+longtmp(2)+longtmp(3)
           IF (occNeighbors.GT.0) THEN                                 400
            IF (RNUNF().LT.(stickingProb*FLOAT(occNeighbors))) THEN
             activeParticle(i)=.FALSE.
             hitArray(xTemp,yTemp)=hitArray(xTemp,yTemp)+1
             hitUnits = hitUnits + 1
```

```
          IF (MOD(hitArray(xTemp,yTemp),releaseThreshold).EQ.0) THEN
           emptyParticle = 1
           numRelease = AINT(extraRelease)
           frac = extraRelease - FLOAT(numRelease)
           IF (RNUNF().LT.frac) numRelease=numRelease+1
           DO 580 release=1,numRelease                                    410
            foundEmpty=.FALSE.
            k=emptyParticle - 1
            DO 540 WHILE (k.LT.i)
             k = k + 1
             IF (.NOT.activeParticle(k)) THEN
              emptyParticle = k
              foundEmpty = .TRUE.
              k = i
             ENDIF
 540         END DO                                                       420
           IF (.NOT.foundEmpty) THEN
            k=highestParticle
            DO 560 WHILE (k.LT.MAX_NUM_PARTICLES)
             k = k + 1
             IF (.NOT.activeParticle(k)) THEN
              emptyParticle = k
              foundEmpty = .TRUE.
              k = MAX_NUM_PARTICLES
             ENDIF
 560         END DO                                                       430
           ENDIF
           IF (.NOT.foundEmpty) THEN
            PRINT *,'Too Many Particles!'
            STOP
           ENDIF
           extraParticles = extraParticles + 1
           totalParts = totalParts + 1
           activeParticle(emptyParticle) = .TRUE.
           xParticle(emptyParticle) = xTemp
           yParticle(emptyParticle) = yTemp                               440
           ageParticle(emptyParticle) = 1
 580       CONTINUE
          ENDIF
         ENDIF
        ENDIF
       ENDIF
      ENDIF
 600  CONTINUE

*     Check if time for growth:                                           450
      IF (waitUnit) THEN
       IF (timeUnits.EQ.growthWaitUnits) keepWaiting=.FALSE.
      ELSE
       IF (hitUnits.GE.growthWaitUnits) keepWaiting=.FALSE.
      ENDIF

      IF (totalTime.EQ.MAX_TIME) THEN
       PRINT *,'Time exceeded maximum'
       STOP
      ENDIF                                                               460

 700  END DO
*     End of loop waiting for growth event time


*     Begin growth:
      IF (hitUnits.GT.0) THEN
       growthSites = 0
       weakest = 1.0
       DO 720 vloc=0,height1                                              470
        DO 710 hloc=0,width1
         IF (hitArray(hloc,vloc).GE.growthThreshold) THEN
```

```
      IF (strengthArraySub(hloc,vloc).LT.weakest) THEN
       weakest=strengthArraySub(hloc,vloc)
      ENDIF
     ENDIF
710   CONTINUE
720  CONTINUE

     DO 900 vloc=0,height1                                                    480
      DO 890 hloc=0,width1
      IF (hitArray(hloc,vloc).GE.growthThreshold) THEN
       bTemp1=(strengthArraySub(hloc,vloc).LE.growthRange)
       bTemp2=(strengthArraySub(hloc,vloc).EQ.weakest)
       IF (bTemp1.OR.bTemp2) THEN

       xTemp = hloc
       yTemp = vloc
       growthSites = growthSites + 1
       numParticles = numParticles + 1                                        490
       imageArray(xTemp,yTemp) = numParticles
       bTemp1=((xTemp.EQ.width2).OR.(xTemp.EQ.1))
       bTemp2=((yTemp.EQ.height2).OR.(yTemp.EQ.1))
       IF (bTemp1.OR.bTemp2) BreakThrough=.TRUE.

*      Mark no production area:
       IF (.NOT.uniformParticleDist) THEN
        DO 760 yNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
        yT = yNo + yTemp
        IF (yT.GT.height1) yT=yT-height                                       500
        IF (yT.LT.0) yT=height-yT
        DO 750 xNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
         xT = xNo + xTemp
         IF (xT.GT.width1) xT=xT-width
         IF (xT.LT.0) xT=width-xT
         IF (SQRT(FLOAT(xNo*xNo+yNo*yNo)).LE.NO_PROD_RADIUS) THEN
          IF (imageArray(xT,yT).EQ.0) imageArray(xT,yT)=-1
         ENDIF
750      CONTINUE
760     CONTINUE                                                              510
       ENDIF
*      End marking no production area

*      Remove additional particles currently residing on growth site:
       DO 770 i=1,highestParticle
        IF (activeParticle(i)) THEN
         IF ((xParticle(i).EQ.xTemp).AND.(yParticle(i).EQ.yTemp))
     C      activeParticle(i)=.FALSE.
        ENDIF
770     CONTINUE                                                              520
*      End removal

*      Release additional particles to nearest neighbors of growth site:
       DO 780 i=0,3
        longtmp(i)=0
780     CONTINUE
       IF (imageArray((xTemp+1),yTemp).LE.0) longtmp(0)=1
       IF (imageArray((xTemp-1),yTemp).LE.0) longtmp(1)=1
       IF (imageArray(xTemp,(yTemp+1)).LE.0) longtmp(2)=1
       IF (imageArray(xTemp,(yTemp-1)).LE.0) longtmp(3)=1                     530
       numRelease = AINT(releaseAtGrowthSite)
       frac = releaseAtGrowthSite - FLOAT(numRelease)
       IF (RNUNF().LT.frac) numRelease=numRelease+1
       bTemp1=((longtmp(0)+longtmp(1)+longtmp(2)+longtmp(3)).EQ.0)
       IF (bTemp1) numRelease=0
       release = 0
       DO 870 WHILE (release.LT.numRelease)
*       Takes into account only the case where homogeneousDif=TRUE
        release = release + 1
        k = AINT(4.0*RNUNF())                                                 540
```

```
          IF (k.EQ.4) k=3
          IF (longtmp(k).NE.1) THEN
           release=release-1
          ELSE
           foundEmpty = .FALSE.
           i = 0
           DO 810 WHILE (i.LT.MAX_NUM_PARTICLES)
            i = i + 1
            IF (.NOT.activeParticle(i)) THEN
             emptyParticle = i                                          550
             foundEmpty = .TRUE.
             i = MAX_NUM_PARTICLES
            ENDIF
 810         END DO
          IF (foundEmpty) THEN
           extraParticles = extraParticles + 1
           totalParts = totalParts + 1
           activeParticle(emptyParticle) = .TRUE.
           ageParticle(emptyParticle) = 1
           k=k+1                                                      560
           GO TO (820,830,840,850), k
 820        xParticle(emptyParticle) = xTemp + 1
          yParticle(emptyParticle) = yTemp
           GO TO 860
 830        xParticle(emptyParticle) = xTemp - 1
          yParticle(emptyParticle) = yTemp
           GO TO 860
 840        xParticle(emptyParticle) = xTemp
          yParticle(emptyParticle) = yTemp + 1
           GO TO 860                                                  570
 850        xParticle(emptyParticle) = xTemp
          yParticle(emptyParticle) = yTemp - 1
           GO TO 860
 860        ENDIF
         ENDIF
 870      END DO
*        End release

        ENDIF
       ENDIF                                                          580
 890   CONTINUE
 900   CONTINUE
*      Print '(1X,A,I5,4X,A,I8,4X,A,I6,4X,A,I5)',
*    C       'Counter:',autoCounter,'Time:',totalTime,'Hits:',
*    C        hitUnits,'Growth Sites:',growthSites
      ENDIF
*     End Growth

1000 END DO
                                                                      590
     PRINT *,'Breakthrough'


*    Find center of mass:
     xc = 0
     yc = 0
     DO 1050 vloc=0,height1
      DO 1040 hloc=0,width1
       IF (imageArray(hloc,vloc).GT.0) THEN
        xc = xc + hloc                                                600
        yc = yc + vloc
       ENDIF
1040  CONTINUE
1050 CONTINUE
     xc = NINT(FLOAT(xc)/FLOAT(numParticles))
     yc = NINT(FLOAT(yc)/FLOAT(numParticles))

     minLength=MIN(xc,yc,(MAX_LENGTH-xc-1),(MAX_LENGTH-yc-1))/2
```

```
    DO 1110 i=0,minLength                                                    610
     DO 1100 vloc=0-i,i
      DO 1090 hloc=0-i,i
       IF (imageArray((xc+hloc),(yc+vloc)).GT.0) THEN
        xc = xc + hloc
        yc = yc + vloc
        GO TO 1120
       ENDIF
1090   CONTINUE
1100   CONTINUE
1110 CONTINUE                                                                620
    PRINT *,'Could not find occupied pixel around center of mass'
    STOP

1120 minLength=MIN(xc,yc,(MAX_LENGTH-xc-1),(MAX_LENGTH-yc-1))/2
    LARGEST_BOX_INDEX = 1
    DO WHILE (boxSizes(LARGEST_BOX_INDEX).GE.minLength)
     LARGEST_BOX_INDEX = LARGEST_BOX_INDEX + 1
    END DO

                                                                             630
    pGroupSize = AINT(FLOAT(numParticles)/FLOAT(numGroups))+1
    DO 1140 vloc=0,height1
     DO 1130 hloc=0,width1
      pAge=imageArray(hloc,vloc)
      IF (pAge.GT.0) THEN
       imageArray(hloc,vloc)=254-AINT(FLOAT(pAge)/FLOAT(pGroupSize))
      ELSE
       imageArray(hloc,vloc)=0
      ENDIF
1130   CONTINUE                                                              640
1140 CONTINUE

    PRINT *,'Find best images'

    furthest = 0
    DO 1160 vloc=0,height1
     DO 1150 hloc=0,width1
      IF (imageArray(hloc,vloc).GT.0) THEN
       yTemp = vloc
       GO TO 1170                                                            650
      ENDIF
1150   CONTINUE
1160 CONTINUE
1170 IF (yTemp.GT.furthest) furthest=yTemp

    vstart=yTemp

    DO 1190 vloc=height1,0,-1
     DO 1180 hloc=0,width1
      IF (imageArray(hloc,vloc).GT.0) THEN                                   660
       yTemp = vloc
       GO TO 1200
      ENDIF
1180   CONTINUE
1190 CONTINUE
1200 IF ((height1-yTemp).GT.furthest) furthest=height1-yTemp

    vend=yTemp

    DO 1220 hloc=0,width1                                                    670
     DO 1210 vloc=0,height1
      IF (imageArray(hloc,vloc).GT.0) THEN
       xTemp = hloc
       GO TO 1230
      ENDIF
1210   CONTINUE
```

```
1220 CONTINUE
1230 IF (xTemp.GT.furthest) furthest=xTemp

    hstart=xTemp                                                          680

    DO 1250 hloc=width1,0,-1
     DO 1240 vloc=0,height1
      IF (imageArray(hloc,vloc).GT.0) THEN
       xTemp = hloc
       GO TO 1260
      ENDIF
1240  CONTINUE
1250 CONTINUE
1260 IF ((width1-xTemp).GT.furthest) furthest=width1-xTemp                690

    hend=xTemp

    IF (furthest.LT.bestDistance) THEN
     bestDistance=furthest
     DO 1290 vloc=0,height1
      DO 1280 hloc=0,width1
       bestImage(hloc,vloc)=imageArray(hloc,vloc)
1280  CONTINUE
1290  CONTINUE                                                            700
    ENDIF

    IF (numParticles.GT.bestMass) THEN
     bestMass = numParticles
     DO 1330 vloc=0,height1
      DO 1320 hloc=0,width1
       heavyImage(hloc,vloc)=imageArray(hloc,vloc)
1320  CONTINUE
1330  CONTINUE
    ENDIF                                                                 710


    PRINT *,'Find mass per radius'

    DO 1340 sizeIndex=LARGEST_BOX_INDEX,NUM_BOX_SIZES
     totalMass(sizeIndex) = 0
     lnBoxSizes(sizeIndex) = ALOG(FLOAT(boxSizes(sizeIndex)))
1340 CONTINUE

    configIdx = 0                                                         720
    DO 1390 yr=(yc-minLength/2),(yc+minLength/2)
     DO 1380 xr=(xc-minLength/2),(xc+minLength/2)
      IF (imageArray(xr,yr).GT.0) THEN
       configIdx = configIdx +1
       DO 1370 sizeIndex=LARGEST_BOX_INDEX,NUM_BOX_SIZES
        mass = 0
        yStart = yr - boxSizes(sizeIndex)
        yEnd = yr + boxSizes(sizeIndex)
        xStart = xr - boxSizes(sizeIndex)
        xEnd = xr + boxSizes(sizeIndex)                                   730
        DO 1360 vloc=yStart,yEnd
         DO 1350 hloc=xStart,xEnd
          temp1=SQRT(FLOAT(vloc-yr)**2+FLOAT(hloc-xr)**2)
          IF (temp1.LE.FLOAT(boxSizes(sizeIndex))) THEN
           IF (imageArray(hloc,vloc).GT.0) mass=mass+1
          ENDIF
1350      CONTINUE
1360      CONTINUE
        totalMass(sizeIndex) = totalMass(sizeIndex) + mass
1370   CONTINUE                                                           740
      ENDIF

1380 CONTINUE
1390 CONTINUE
```

```
      DO 1400 sizeIndex=LARGEST_BOX_INDEX,NUM_BOX_SIZES
       lnTotalMass(sizeIndex) =
     C        ALOG(FLOAT(totalMass(sizeIndex))/FLOAT(configIdx))
1400  CONTINUE
                                                                        750
      PRINT *,'Calculating dimension'

      LAST_BOX_INDEX = 24
      FIRST_BOX_INDEX = LARGEST_BOX_INDEX
      j = LAST_BOX_INDEX - LARGEST_BOX_INDEX + 1
      sx = 0.0
      sy = 0.0
      st2 = 0.0
      db = 0.0
      DO 1420 i=FIRST_BOX_INDEX,LAST_BOX_INDEX                          760
       sx = sx + lnBoxSizes(i)
       sy = sy + lnTotalMass(i)
1420  CONTINUE
      ss = FLOAT(LAST_BOX_INDEX - FIRST_BOX_INDEX) + 1.0
      sxoss = sx/ss

      DO 1440 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
       t = lnBoxSizes(i) - sxoss
       st2 = st2 + t*t
       db = db + t*lnTotalMass(i)                                       770
1440  CONTINUE
      db = db/st2
      sddb = SQRT(1.0/st2)
      intcp = (sy - sx*db)/ss
      chi2 = 0.0

      DO 1460 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
       temp1 = lnTotalMass(i) - intcp - db*lnBoxSizes(i)
       chi2 = chi2 + temp1*temp1
1460  CONTINUE                                                          780
      sigdat = SQRT(chi2/(ss - 2.0))
      sddb = sddb*sigdat

      dimList(autoCounter)=db
      sdDimList(autoCounter)=sddb
      timeMassList(autoCounter)=FLOAT(totalTime)/FLOAT(numParticles)
      tissueEffList(autoCounter)=FLOAT(numParticles)/FLOAT(tissueParts)
      totalEffList(autoCounter)=FLOAT(numParticles)/FLOAT(totalParts)
      timeList(autoCounter)=totalTime
      massList(autoCounter)=numParticles                                790
      tissuePartList(autoCounter)=tissueParts
      totalPartList(autoCounter)=totalParts
      decayPartList(autoCounter)=numDecayed


      PRINT *,'Find minimum path'

      DO 1620 vloc=0,height1
       DO 1610 hloc=0,width1
         minPathArray(hloc,vloc)=0                                      800
1610   CONTINUE
1620  CONTINUE

      noProgress = .TRUE.
      BreakThrough = .FALSE.

      stepIndex = 1
      DO 1630 hloc=hstart,hend
       IF (imageArray(hloc,vstart).GT.0) THEN
        minPathArray(hloc,vstart)=stepIndex                             810
       ENDIF
1630  CONTINUE
```

```
      DO 1680 WHILE (.NOT.BreakThrough)
       DO 1660 vloc=vstart,vend
        DO 1650 hloc=hstart,hend
         IF (minPathArray(hloc,vloc).EQ.stepIndex) THEN

          bTemp1=(imageArray((hloc+1),vloc).GT.0)
          bTemp2=(minPathArray((hloc+1),vloc).EQ.0)                    820
          IF (bTemp1.AND.bTemp2) minPathArray((hloc+1),vloc)=stepIndex+1

          bTemp1=(imageArray((hloc-1),vloc).GT.0)
          bTemp2=(minPathArray((hloc-1),vloc).EQ.0)
          IF (bTemp1.AND.bTemp2) minPathArray((hloc-1),vloc)=stepIndex+1

          bTemp1=(imageArray(hloc,(vloc+1)).GT.0)
          bTemp2=(minPathArray(hloc,(vloc+1)).EQ.0)
          IF (bTemp1.AND.bTemp2) minPathArray(hloc,(vloc+1))=stepIndex+1
                                                                       830
          bTemp1=(imageArray(hloc,(vloc-1)).GT.0)
          bTemp2=(minPathArray(hloc,(vloc-1)).EQ.0)
          IF (bTemp1.AND.bTemp2) minPathArray(hloc,(vloc-1))=stepIndex+1

          noProgress = .FALSE.
         ENDIF
1650    CONTINUE
1660   CONTINUE

      IF (noProgress) THEN                                             840
       PRINT *,'No Progress Finding Minimum Path'
       STOP
      ENDIF

      stepIndex=stepIndex+1

      vloc = vend
      hloc = hstart - 1
      DO 1670 WHILE (hloc.LT.hend)
       hloc = hloc + 1                                                 850
       IF (minPathArray(hloc,vloc).NE.0) THEN
        BreakThrough = .TRUE.
        PRINT *,'Breakthrough minimum path'
        breakPoint = hloc
        hloc = hend
       ENDIF
1670   END DO

1680 END DO
                                                                       860
      LARGEST_BOX_INDEX = 1
      DO WHILE (boxSizes(LARGEST_BOX_INDEX).GE.((vend-vstart)/4))
       LARGEST_BOX_INDEX = LARGEST_BOX_INDEX + 1
      END DO

      vloc = vend
      hloc = breakPoint
      pointIndex = 0
      stepIndex = minPathArray(hloc,vloc) - 1
      minPathArray(hloc,vloc) = -1                                     870
      minPath(pointIndex,0) = hloc
      minPath(pointIndex,1) = vloc

      DO 1720 WHILE (vloc.GT.vstart)
       xs = hloc - 1
       ys = vloc - 1
       xe = hloc + 1
       ye = vloc + 1
       DO 1710 y=ys,ye
        DO 1700 x=xs,xe                                                880
```

```
      IF (minPathArray(x,y).EQ.stepIndex) THEN
       minPathArray(x,y) = -1
       stepIndex = stepIndex - 1
       pointIndex = pointIndex + 1
       hloc = x
       vloc = y
       minPath(pointIndex,0) = hloc
       minPath(pointIndex,1) = vloc
       GO TO 1720
      ENDIF                                                          890
1700  CONTINUE
1710 CONTINUE
1720 END DO

     DO 1780 sizeIndex=LARGEST_BOX_INDEX,NUM_BOX_SIZES
      eSize = FLOAT(boxSizes(sizeIndex))
      occBox = 0
      pointLoc = 0
      DO 1760 i=0,pointIndex
       dx = minPath(i,0) - minPath(pointLoc,0)                       900
       dy = minPath(i,1) - minPath(pointLoc,1)
       r1 = SQRT(FLOAT(dx*dx) + FLOAT(dy*dy))
       IF (r1.GE.eSize) THEN
        pointLoc = i
        occBox = occBox + 1
       ENDIF
1760  CONTINUE

     IF (pointLoc.LT.pointIndex) occBox=occBox+1
                                                                     910
     totalMass(sizeIndex) = occBox
     lnBoxSizes(sizeIndex) = ALOG(FLOAT(boxSizes(sizeIndex)))
     lnTotalMass(sizeIndex) = ALOG(FLOAT(totalMass(sizeIndex)))
1780 CONTINUE

     PRINT *,'Calculating minimum-path dimension'

     LAST_BOX_INDEX = 24
     FIRST_BOX_INDEX = LARGEST_BOX_INDEX
     j = LAST_BOX_INDEX - LARGEST_BOX_INDEX + 1                      920
     sx = 0.0
     sy = 0.0
     st2 = 0.0
     db = 0.0
     DO 1820 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
      sx = sx + lnBoxSizes(i)
      sy = sy + lnTotalMass(i)
1820 CONTINUE
     ss = FLOAT(LAST_BOX_INDEX - FIRST_BOX_INDEX) + 1.0
     sxoss = sx/ss                                                   930

     DO 1840 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
      t = lnBoxSizes(i) - sxoss
      st2 = st2 + t*t
      db = db + t*lnTotalMass(i)
1840 CONTINUE
     db = db/st2
     sddb = SQRT(1.0/st2)
     intcp = (sy - sx*db)/ss
     chi2 = 0.0                                                      940

     DO 1860 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
      temp1 = lnTotalMass(i) - intcp - db*lnBoxSizes(i)
      chi2 = chi2 + temp1*temp1
1860 CONTINUE
     sigdat = SQRT(chi2/(ss - 2.0))
     sddb = sddb*sigdat
```

```
      mpdimList(autoCounter)=0.0-db
      sdMpdimList(autoCounter)=sddb                                          950

      i=autoCounter
      WRITE (16,2250) i,dimList(i),sdDimList(i),
     C            mpdimList(i),sdMpdimList(i),timeMassList(i),
     C            tissueEffList(i),totalEffList(i),timeList(i),
     C            massList(i),tissuePartList(i),
     C            totalPartList(i),decayPartList(i)

      PRINT '(1X,A,I5,A,I5)','Finished Iteration ',autoCounter,
     C            ' out of ',AUTO_ITERATIONS                                 960

      IF (autoCounter.LT.AUTO_ITERATIONS) GO TO 10

      CLOSE (UNIT=16)

      denominator = 0.0
      numerator = 0.0
      DO 1880 i=1,AUTO_ITERATIONS
       denominator=denominator+1.0/sdMpdimList(i)**2
       numerator=numerator+mpdimList(i)*(1.0/sdMpdimList(i)**2)              970
1880  CONTINUE
      mpdim=numerator/denominator

      numerator = 0.0
      DO 1890 i=1,AUTO_ITERATIONS
       numerator=numerator+(mpdimList(i)-mpdim)**2
1890  CONTINUE
      sdmpdim=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

      denominator = 0.0                                                      980
      numerator = 0.0
      DO 1900 i=1,AUTO_ITERATIONS
       denominator=denominator+1.0/sdDimList(i)**2
       numerator=numerator+dimList(i)*(1.0/sdDimList(i)**2)
1900  CONTINUE
      db=numerator/denominator

      numerator = 0.0
      DO 1910 i=1,AUTO_ITERATIONS
       numerator=numerator+(dimList(i)-db)**2                                990
1910  CONTINUE
      sddb=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

      numerator = 0.0
      DO 1920 i=1,AUTO_ITERATIONS
       numerator=numerator+timeMassList(i)
1920  CONTINUE
      timeMassAvg=numerator/FLOAT(AUTO_ITERATIONS)

      numerator = 0.0                                                        1000
      DO 1930 i=1,AUTO_ITERATIONS
       numerator=numerator+(timeMassList(i)-timeMassAvg)**2
1930  CONTINUE
      timeMassSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

      numerator = 0.0
      DO 1940 i=1,AUTO_ITERATIONS
       numerator=numerator+tissueEffList(i)
1940  CONTINUE
      tissueEffAvg=numerator/FLOAT(AUTO_ITERATIONS)                          1010

      numerator = 0.0
      DO 1950 i=1,AUTO_ITERATIONS
       numerator=numerator+(tissueEffList(i)-tissueEffAvg)**2
1950  CONTINUE
      tissueEffSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))
```

```
      numerator = 0.0
      DO 1960 i=1,AUTO_ITERATIONS
       numerator=numerator+totalEffList(i)                                    1020
1960  CONTINUE
      totalEffAvg=numerator/FLOAT(AUTO_ITERATIONS)

      numerator = 0.0
      DO 1970 i=1,AUTO_ITERATIONS
       numerator=numerator+(totalEffList(i)-totalEffAvg)**2
1970  CONTINUE
      totalEffSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

      numerator = 0.0                                                         1030
      DO 1980 i=1,AUTO_ITERATIONS
       numerator=numerator+FLOAT(timeList(i))
1980  CONTINUE
      timeAvg=numerator/FLOAT(AUTO_ITERATIONS)

      numerator = 0.0
      DO 1990 i=1,AUTO_ITERATIONS
       numerator=numerator+(FLOAT(timeList(i))-timeAvg)**2
1990  CONTINUE
      timeSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))                         1040

      numerator = 0.0
      DO 2000 i=1,AUTO_ITERATIONS
       numerator=numerator+FLOAT(massList(i))
2000  CONTINUE
      massAvg=numerator/FLOAT(AUTO_ITERATIONS)

      numerator = 0.0
      DO 2010 i=1,AUTO_ITERATIONS
       numerator=numerator+(FLOAT(massList(i))-massAvg)**2                    1050
2010  CONTINUE
      massSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

      numerator = 0.0
      DO 2020 i=1,AUTO_ITERATIONS
       numerator=numerator+FLOAT(tissuePartList(i))
2020  CONTINUE
      tissuePartAvg=numerator/FLOAT(AUTO_ITERATIONS)

      numerator = 0.0                                                         1060
      DO 2030 i=1,AUTO_ITERATIONS
       numerator=numerator+(FLOAT(tissuePartList(i))-tissuePartAvg)**2
2030  CONTINUE
      tissuePartSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

      numerator = 0.0
      DO 2040 i=1,AUTO_ITERATIONS
       numerator=numerator+FLOAT(totalPartList(i))
2040  CONTINUE
      totalPartAvg=numerator/FLOAT(AUTO_ITERATIONS)                           1070

      numerator = 0.0
      DO 2050 i=1,AUTO_ITERATIONS
       numerator=numerator+(FLOAT(totalPartList(i))-totalPartAvg)**2
2050  CONTINUE
      totalPartSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

      numerator = 0.0
      DO 2060 i=1,AUTO_ITERATIONS
       numerator=numerator+FLOAT(decayPartList(i))                            1080
2060  CONTINUE
      decayPartAvg=numerator/FLOAT(AUTO_ITERATIONS)

      numerator = 0.0
```

```
      DO 2070 i=1,AUTO_ITERATIONS
       numerator=numerator+(FLOAT(decayPartList(i))-decayPartAvg)**2
2070  CONTINUE
      decayPartSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

      CALL ITIME(tArray)                                                    1090
      compTime=tArray(3)+tArray(2)*60+tArray(1)*3600-compTime
      PRINT '(1X,A,I10)','Execution Time (secs):  ',compTime

      OPEN (UNIT=19,FILE='Model.Averages',STATUS='NEW')
      WRITE (19,2200) ' '
      WRITE (19,2200) 'Averages:'
      WRITE (19,2240) 'Fractal Dimension:',db,' +/-',sddb
      WRITE (19,2240) 'Minimum-Path Dimension:',mpdim,' +/-',sdmpdim
      WRITE (19,2240) 'Time/Mass:',timeMassAvg,' +/-',timeMassSD
      WRITE (19,2240) 'Tis.  Efficn.:',tissueEffAvg,' +/-',tissueEffSD      1100
      WRITE (19,2240) 'Tot.  Efficn.:',totalEffAvg,' +/-',totalEffSD
      WRITE (19,2240) 'Time:',timeAvg,' +/-',timeSD
      WRITE (19,2240) 'Mass:',massAvg,' +/-',massSD
      WRITE (19,2240) 'Tis.  Prtcls:',tissuePartAvg,' +/-',tissuePartSD
      WRITE (19,2240) 'Tot.  Prtcls:',totalPartAvg,' +/-',totalPartSD
      WRITE (19,2240) 'Dec.  Prtcls:',decayPartAvg,' +/-',decayPartSD
      CLOSE (UNIT=19)

      OPEN (UNIT=17,FILE='Model.Best',STATUS='NEW')
       DO 2120 vloc=0,height1                                               1110
        DO 2110 hloc=0,width2
         WRITE (17,2260) bestImage(hloc,vloc),'T'
2110    CONTINUE
        WRITE (17,2260) bestImage(width1,vloc),'C'
2120   CONTINUE
      CLOSE (UNIT=17)

      OPEN (UNIT=18,FILE='Model.Heavy',STATUS='NEW')
       DO 2140 vloc=0,height1
        DO 2130 hloc=0,width2                                               1120
         WRITE (18,2260) heavyImage(hloc,vloc),'T'
2130    CONTINUE
        WRITE (18,2260) heavyImage(width1,vloc),'C'
2140   CONTINUE
      CLOSE (UNIT=18)

2200  FORMAT (1X,A)
2210  FORMAT (1X,A,I9)
2220  FORMAT (1X,A,F9.7,1X,A)
2230  FORMAT (1X,A,F9.5)                                                    1130
2240  FORMAT (1X,A,F16.7,A,F16.7)
2250  FORMAT (1X,I3,1X,F6.4,1X,F6.4,1X,F6.4,1X,F6.4,
     C        F8.5,F8.5,F8.5,I7,I6,I7,I7,I6)
2260  FORMAT (1X,I4,A)
2270  FORMAT (1X,A,A)

      END
```

A.7 3-D Growth Modeling Program

```
      PROGRAM DoModelTime

      REAL*4 RNUNF

*     Assign parameters:
      INTEGER*4 AUTO_ITERATIONS,MAX_LENGTH,MAX_LENGTH1
      PARAMETER (AUTO_ITERATIONS=10,MAX_LENGTH=32,MAX_LENGTH1=31)
      INTEGER*4 MAX_TIME,MAX_NUM_PARTICLES
      PARAMETER (MAX_TIME=1000000,MAX_NUM_PARTICLES=50000)
      INTEGER*4 NO_PROD_RADIUS                                          10
      PARAMETER (NO_PROD_RADIUS=5)
      INTEGER*4 NUM_BOX_SIZES
      PARAMETER (NUM_BOX_SIZES=25)
      INTEGER*4 MAX_PATH_LENGTH
      PARAMETER (MAX_PATH_LENGTH=1000)

*     Declare image-size variables:
      INTEGER*4 width,width1,width2,height,height1,height2
      INTEGER*4 depth,depth1,depth2
      REAL*4 imageVolume                                                20

*     Declare multi-purpose variables:
      INTEGER*4 vloc,hloc,xTemp,yTemp,xNo,yNo,yT,xT
      INTEGER*4 zloc,zTemp,zNo,zT
      INTEGER*4 k,i,j
      INTEGER*4 longtmp(0:5)
      LOGICAL*4 bTemp1,bTemp2
      REAL*4 numerator,denominator
      INTEGER*4 compTime,tArray(1:3)
                                                                        30
*     Declare model parameters:
      INTEGER*4 halfLife,growthWaitUnits,intL
      INTEGER*4 numGroups,growthThreshold,releaseThreshold
      LOGICAL*4 homogeneousSub,homogeneousDif,centerSeed
      REAL*4 productionRate,stickingProb,extraRelease,growthRange
      REAL*4 releaseAtGrowthSite,tau,fintL
      LOGICAL*4 waitUnit,uniformParticleDist

*     Declare model variables:
      INTEGER*4 xCntr,yCntr,release,numRelease,autoCounter              40
      INTEGER*4 zCntr
      INTEGER*4 timeUnits,hitUnits,numDecayed,numParticles,occNeighbors
      INTEGER*4 totalTime,highestParticle,numNew,emptyParticle
      INTEGER*4 totalParts,tissueParts,extraParticles,growthSites
      REAL*4 decayProb,weakest,frac
      LOGICAL*4 BreakThrough,keepWaiting,foundEmpty
      INTEGER*4 furthest,bestDistance,bestMass

*     Declare final image variables:
      INTEGER*4 pGroupSize,pAge                                         50

*     Declare arrays:
      INTEGER*4 imageArray(0:MAX_LENGTH1,0:MAX_LENGTH1,0:MAX_LENGTH1)
      REAL*4 strengthArrayDif(0:MAX_LENGTH1,0:MAX_LENGTH1,0:MAX_LENGTH1)
      REAL*4 strengthArraySub(0:MAX_LENGTH1,0:MAX_LENGTH1,0:MAX_LENGTH1)
      INTEGER*4 hitArray(0:MAX_LENGTH1,0:MAX_LENGTH1,0:MAX_LENGTH1)
      INTEGER*4 xParticle(1:MAX_NUM_PARTICLES)
      INTEGER*4 yParticle(1:MAX_NUM_PARTICLES)
      INTEGER*4 zParticle(1:MAX_NUM_PARTICLES)
      INTEGER*4 ageParticle(1:MAX_NUM_PARTICLES)                        60
      LOGICAL*4 activeParticle(1:MAX_NUM_PARTICLES)

*     Declare result variables:
      INTEGER*4 bestImage(0:MAX_LENGTH1,0:MAX_LENGTH1,0:MAX_LENGTH1)
```

```
      INTEGER*4 heavyImage(0:MAX_LENGTH1,0:MAX_LENGTH1,0:MAX_LENGTH1)
      REAL*4 dimList(0:AUTO_ITERATIONS)
      REAL*4 sdDimList(0:AUTO_ITERATIONS)
      REAL*4 mpdimList(0:AUTO_ITERATIONS)
      REAL*4 sdMpdimList(0:AUTO_ITERATIONS)
      REAL*4 timeMassList(0:AUTO_ITERATIONS)                                    70
      REAL*4 tissueEffList(0:AUTO_ITERATIONS)
      REAL*4 totalEffList(0:AUTO_ITERATIONS)
      INTEGER*4 timeList(0:AUTO_ITERATIONS),massList(0:AUTO_ITERATIONS)
      INTEGER*4 tissuePartList(0:AUTO_ITERATIONS)
      INTEGER*4 totalPartList(0:AUTO_ITERATIONS)
      INTEGER*4 decayPartList(0:AUTO_ITERATIONS)
      REAL*4 timeMassAvg,totalEffAvg,tissueEffAvg,timeAvg,massAvg
      REAL*4 tissuePartAvg,totalPartAvg,decayPartAvg
      REAL*4 timeMassSD,totalEffSD,tissueEffSD,timeSD,massSD
      REAL*4 tissuePartSD,totalPartSD,decayPartSD                               80
      REAL*4 mpdim,sdmpdim

*     Declare mass-radius variables:
      INTEGER*4 sizeIndex,xStart,yStart,xEnd,yEnd,mass,xc,yc
      INTEGER*4 zStart,zEnd,zc
      INTEGER*4 xr,yr,minLength
      INTEGER*4 zr
      INTEGER*4 LARGEST_BOX_INDEX,FIRST_BOX_INDEX,LAST_BOX_INDEX
      REAL*4 temp1,temp2
      REAL*4 sx,sy,st2,db,ss,sxoss,t,sddb,chi2,sigdat,intcp                     90
      INTEGER*4 boxSizes(1:NUM_BOX_SIZES),totalMass(1:NUM_BOX_SIZES)
      REAL*4 lnBoxSizes(1:NUM_BOX_SIZES),lnTotalMass(1:NUM_BOX_SIZES)

*     Declare minimum path variables:
      LOGICAL*4 noProgress
      INTEGER*4 minPathArray(0:MAX_LENGTH1,0:MAX_LENGTH1,0:MAX_LENGTH1)
      INTEGER*4 stepIndex,vstart,vend,hstart,hend
      INTEGER*4 dstart,dend
      INTEGER*4 breakPoint,x,y,xe,ye,xs,ys
      INTEGER*4 zbreakPoint,z,ze,zs                                             100
      INTEGER*4 minPath(0:MAX_PATH_LENGTH,0:2)
      INTEGER*4 pointIndex,pointLoc,occBox,configIdx
      INTEGER*4 dx,dy
      INTEGER*4 dz
      REAL*4 eSize,r1

      CALL ITIME(tArray)
      compTime=tArray(3)+tArray(2)*60+tArray(1)*3600

*     Initialize parameters:                                                    110
      intL=32768
      fintL=32768.0

*     Initialize size variables:
      width = MAX_LENGTH
      width1 = width - 1
      width2 = width - 2
      height = MAX_LENGTH
      height1 = height - 1
      height2 = height - 2                                                      120
      depth = MAX_LENGTH
      depth1 = depth - 1
      depth2 = depth - 2

*     Initialize dimension measurement parameters:
      boxSizes(1) = 512
      temp1 = 512.0
      DO 5 k=2,NUM_BOX_SIZES
    4 temp2 = 0.793700526*temp1
      j = NINT(temp2)                                                           130
      temp1 = temp2
      IF (j.EQ.boxSizes(k-1)) GO TO 4
```

```
      boxSizes(k) = j
    5 CONTINUE


*     Assign Parameters:
      homogeneousSub = .FALSE.
      homogeneousDif = .TRUE.
      centerSeed = .TRUE.                                                    140
      waitUnit = .TRUE.
      growthWaitUnits = 10
      halfLife = 1000000
      tau = FLOAT(halfLife)/ALOG(2.0)
      numGroups = 250
      uniformParticleDist = .FALSE.
      imageVolume = FLOAT(height)*FLOAT(width)*FLOAT(depth)
      productionRate = 0.000056234133
      stickingProb = 1.0
      extraRelease = 0.0                                                     150
      releaseAtGrowthSite = 3.0
      growthThreshold = 1
      releaseThreshold = 1
      growthRange = 0.5

      OPEN (UNIT=15,FILE='Model.Specs',STATUS='NEW')
      WRITE (15,2200) 'Model Specifications:'
      IF (homogeneousSub) THEN
       WRITE (15,2200) 'Homogeneous Substrate'
      ELSE                                                                   160
       WRITE (15,2200) 'Inhomogeneous Substrate'
      ENDIF
      IF (homogeneousDif) THEN
       WRITE (15,2200) 'Homogeneous Diffusion'
      ELSE
       WRITE (15,2200) 'Inhomogeneous Diffusion'
      ENDIF
      IF (centerSeed) THEN
       WRITE (15,2200) 'Center Seed'
      ELSE                                                                   170
       WRITE (15,2200) 'Multiple Seeds'
      ENDIF
      IF (waitUnit) THEN
       WRITE (15,2200) 'Wait Unit:  TIME'
      ELSE
       WRITE (15,2200) 'Wait Unit:  HIT'
      ENDIF
      WRITE (15,2210) 'Growth Wait Units :',growthWaitUnits
      WRITE (15,2210) 'Particle Half-life:',halfLife
      IF (uniformParticleDist) THEN                                          180
       WRITE (15,2200) 'Uniform particle origin'
      ELSE
       WRITE (15,2200) 'Particle origin beyond network'
      ENDIF
      WRITE (15,2220) 'Production Rate:  ',productionRate,
     C                     'particles/pixel*time'
      WRITE (15,2230) 'Interaction Probability:',stickingProb
      WRITE (15,2230) 'Extra release per hit:',extraRelease
      WRITE (15,2230) 'Extra release at growth site:',
     C                     releaseAtGrowthSite                             190
      WRITE (15,2210) 'Growth threshold:',growthThreshold
      WRITE (15,2210) 'Extra release threshold:',releaseThreshold
      WRITE (15,2230) 'Growth Range:    0.0  -',growthRange
      CLOSE (UNIT=15)

      OPEN (UNIT=16,FILE='Model.Data',STATUS='NEW')
      WRITE (16,2270)
     C    ' No      Dm-r          Dmp         T/M    tis-E',
     C    '   tot-E   Time  Mass    Tis    Tot   Dec'
                                                                            200
```

```
      bestDistance = MAX_LENGTH/2
      bestMass = 0

*     Seeding code:
      CALL RNOPT(6)
      CALL RNSET(0)
*     End of seeding code

*     Initialize iteration counter:
      autoCounter = 0                                                    210
   10 autoCounter = autoCounter + 1



*     Assign growth substrate strengths:
      IF (.NOT.homogeneousSub) THEN
        PRINT *,'Generating random substrate strengths...'
        DO 110 vloc=0,height1
         DO 105 hloc=0,width1
          DO 100 zloc=0,depth1                                           220
           strengthArraySub(hloc,vloc,zloc)= RNUNF()
  100     CONTINUE
  105    CONTINUE
  110    CONTINUE
        PRINT *,'Random growth substrate generated.'
      ELSE
        DO 130 vloc=0,height1
         DO 125 hloc=0,width1
          DO 120 zloc=0,depth1
           strengthArraySub(hloc,vloc,zloc)=0.5                          230
  120     CONTINUE
  125    CONTINUE
  130    CONTINUE
      ENDIF


*     Assign diffusion substrate strengths:
      IF (.NOT.homogeneousDif) THEN
        PRINT *,'Generating random diffusion strengths...'
        DO 160 vloc=0,height1                                            240
         DO 155 hloc=0,width1
          DO 150 zloc=0,depth1
           strengthArrayDif(hloc,vloc,zloc)= RNUNF()
  150     CONTINUE
  155    CONTINUE
  160    CONTINUE
        PRINT *,'Random diffusion substrate generated.'
      ENDIF

                                                                         250
*     Initialize model variables:

      DO 200 i=1,MAX_NUM_PARTICLES
       activeParticle(i)=.FALSE.
  200 CONTINUE

      BreakThrough = .FALSE.
      numDecayed = 0
      totalTime = 0
      totalParts = 0                                                     260
      tissueParts = 0
      extraParticles = 0

      DO 220 vloc=0,height1
       DO 215 hloc=0,width1
        DO 210 zloc=0,depth1
         imageArray(hloc,vloc,zloc)=0
  210    CONTINUE
```

```
215 CONTINUE
220 CONTINUE                                                          270

    IF (centerSeed) THEN
     yCntr = height/2
     xCntr = width/2
     zCntr = depth/2
     numParticles = 1
     imageArray(xCntr,yCntr,zCntr)=numParticles

     IF (.NOT.uniformParticleDist) THEN
      DO 260 zNo=-NO_PROD_RADIUS,NO_PROD_RADIUS                       280
       zT = zNo + zCntr
       IF (zT.GT.depth1) zT=zT-depth
       IF (zT.LT.0) zT=depth-zT
       DO 250 yNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
        yT = yNo + yCntr
        IF (yT.GT.height1) yT=yT-height
        IF (yT.LT.0) yT=height-yT
        DO 240 xNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
         xT = xNo + xCntr
         IF (xT.GT.width1) xT=xT-width                                290
         IF (xT.LT.0) xT=width-xT
         k=xNo*xNo + yNo*yNo* + zNo*zNo
         IF (SQRT(FLOAT(k)).LT.NO_PROD_RADIUS) THEN
          IF (imageArray(xT,yT,zT).EQ.0) imageArray(xT,yT,zT)=-1
         ENDIF
240     CONTINUE
250    CONTINUE
260   CONTINUE
     ENDIF
     ELSE                                                             300
      PRINT *,'Only center-seed option is available!'
      STOP
     ENDIF


*    ####### Main loop until BreakThrough #######
     DO 1000 WHILE (.NOT.BreakThrough)
     keepWaiting=.TRUE.
     timeUnits=0
     hitUnits=0                                                       310

     DO 310 vloc=0,height1
      DO 305 hloc=0,width1
       DO 300 zloc=0,depth1
        hitArray(hloc,vloc,zloc)=0
300    CONTINUE
305   CONTINUE
310  CONTINUE

     highestParticle=0                                                320
     DO 320 i=1,MAX_NUM_PARTICLES
      IF (activeParticle(i)) highestParticle=i
320  CONTINUE

*    Loop waiting for time for growth:
     DO 700 WHILE (keepWaiting)

      timeUnits=timeUnits+1
      totalTime=totalTime+1
                                                                      330
*     Advance existing particles:
      DO 400 i=1,highestParticle
       IF (activeParticle(i)) THEN

        ageParticle(i)=ageParticle(i)+1
```

```
*       Takes into account only the case where homogeneousDif=TRUE:
        k=AINT(6.0*RNUNF())+1
        GO TO (330,340,350,360,370,380), k
330      zTemp = zParticle(i) + 1                                          340
        yTemp = yParticle(i)
        xTemp = xParticle(i)
        IF (zTemp.EQ.depth) zTemp=0
        GO TO 390
340      zTemp = zParticle(i) - 1
        yTemp = yParticle(i)
        xTemp = xParticle(i)
        IF (zTemp.EQ.-1) zTemp=depth1
        GO TO 390
350      xTemp = xParticle(i) + 1                                          350
        yTemp = yParticle(i)
        zTemp = zParticle(i)
        IF (xTemp.EQ.width) xTemp=0
        GO TO 390
360      xTemp = xParticle(i) - 1
        yTemp = yParticle(i)
        zTemp = zParticle(i)
        IF (xTemp.EQ.-1) xTemp=width1
        GO TO 390
370      yTemp = yParticle(i) - 1                                          360
        xTemp = xParticle(i)
        zTemp = zParticle(i)
        IF (yTemp.EQ.-1) yTemp=height1
        GO TO 390
380      yTemp = yParticle(i) + 1
        xTemp = xParticle(i)
        zTemp = zParticle(i)
        IF (yTemp.EQ.height) yTemp=0
390      IF (imageArray(xTemp,yTemp,zTemp).LE.0) THEN
        xParticle(i)=xTemp                                                370
        yParticle(i)=yTemp
        zParticle(i)=zTemp
       ENDIF
      ENDIF
400   CONTINUE

*       Produce new particles:

      numNew=AINT(productionRate*imageVolume)
      frac=productionRate*imageVolume-1.0*FLOAT(numNew)                   380
      IF (RNUNF().LT.frac) numNew=numNew+1
      emptyParticle = 1
      DO 460 i=1,numNew
       xTemp = NINT(RNUNF()*width1)
       yTemp = NINT(RNUNF()*height1)
       zTemp = NINT(RNUNF()*depth1)
       IF (imageArray(xTemp,yTemp,zTemp).EQ.0) THEN
        foundEmpty = .FALSE.
        k = emptyParticle - 1
        DO 450 WHILE (k.LT.MAX_NUM_PARTICLES)                             390
         k = k + 1
         IF (.NOT.activeParticle(k)) THEN
          emptyParticle = k
          foundEmpty = .TRUE.
          k = MAX_NUM_PARTICLES
         ENDIF
450      END DO

        IF (.NOT.foundEmpty) THEN
         PRINT *,'Too Many Particles!'                                    400
         STOP
        ENDIF

        tissueParts = tissueParts + 1
```

```
      totalParts = totalParts + 1
      activeParticle(emptyParticle)=.TRUE.
      xParticle(emptyParticle)=xTemp
      yParticle(emptyParticle)=yTemp
      zParticle(emptyParticle)=zTemp
      ageParticle(emptyParticle)=1                                          410
      IF (emptyParticle.GT.highestParticle)
     C            highestParticle=emptyParticle
      ENDIF
  460 CONTINUE


*     Degrade old particles:

      DO 500 i=1,highestParticle
      IF (activeParticle(i)) THEN                                           420
       decayProb = EXP(-FLOAT(ageParticle(i))/tau)/tau
       IF (RNUNF().LT.decayProb) THEN
        numDecayed = numDecayed + 1
        activeParticle(i)=.FALSE.
       ENDIF
      ENDIF
  500 CONTINUE

*     Mark hits:
      DO 600 i=1,highestParticle                                            430
      IF (activeParticle(i)) THEN
       xTemp = xParticle(i)
       yTemp = yParticle(i)
       zTemp = zParticle(i)
       bTemp1 = ((xTemp.LT.width1).AND.(yTemp.LT.height1))
       bTemp1 = (bTemp1.AND.(zTemp.LT.depth1))
       bTemp2 = ((xTemp.GT.0).AND.(yTemp.GT.0))
       bTemp2 = (bTemp2.AND.(zTemp.GT.0))
       IF (bTemp1.AND.bTemp2) THEN
        longtmp(0)=                                                         440
     C       AINT(FLOAT(intL-1+imageArray((xTemp+1),yTemp,zTemp))/
     C                                          fintL)
        longtmp(1)=
     C       AINT(FLOAT(intL-1+imageArray((xTemp-1),yTemp,zTemp))/
     C                                          fintL)
        longtmp(2)=
     C       AINT(FLOAT(intL-1+imageArray(xTemp,(yTemp+1),zTemp))/
     C                                          fintL)
        longtmp(3)=
     C       AINT(FLOAT(intL-1+imageArray(xTemp,(yTemp-1),zTemp))/          450
     C                                          fintL)
        longtmp(4)=
     C       AINT(FLOAT(intL-1+imageArray(xTemp,yTemp,(zTemp+1)))/
     C                                          fintL)
        longtmp(5)=
     C       AINT(FLOAT(intL-1+imageArray(xTemp,yTemp,(zTemp-1)))/
     C                                          fintL)
       occNeighbors=longtmp(0)+longtmp(1)+longtmp(2)+longtmp(3)+
     C              longtmp(4)+longtmp(5)
       IF (occNeighbors.GT.0) THEN                                          460
        IF (RNUNF().LT.(stickingProb*FLOAT(occNeighbors))) THEN
         activeParticle(i)=.FALSE.
         hitArray(xTemp,yTemp,zTemp)=hitArray(xTemp,yTemp,zTemp)+1
         hitUnits = hitUnits + 1
         IF (MOD(hitArray(xTemp,yTemp,zTemp),releaseThreshold).EQ.0)
     C     THEN
          emptyParticle = 1
          numRelease = AINT(extraRelease)
          frac = extraRelease - FLOAT(numRelease)
          IF (RNUNF().LT.frac) numRelease=numRelease+1                      470
          DO 580 release=1,numRelease
           foundEmpty=.FALSE.
```

```
          k=emptyParticle - 1
          DO 540 WHILE (k.LT.i)
           k = k + 1
           IF (.NOT.activeParticle(k)) THEN
            emptyParticle = k
            foundEmpty = .TRUE.
            k = i
           ENDIF                                                    480
 540        END DO
          IF (.NOT.foundEmpty) THEN
           k=highestParticle
           DO 560 WHILE (k.LT.MAX_NUM_PARTICLES)
            k = k + 1
            IF (.NOT.activeParticle(k)) THEN
             emptyParticle = k
             foundEmpty = .TRUE.
             k = MAX_NUM_PARTICLES
            ENDIF                                                   490
 560        END DO
          ENDIF
          IF (.NOT.foundEmpty) THEN
           PRINT *,'Too Many Particles!'
           STOP
          ENDIF
          extraParticles = extraParticles + 1
          totalParts = totalParts + 1
          activeParticle(emptyParticle) = .TRUE.
          xParticle(emptyParticle) = xTemp                          500
          yParticle(emptyParticle) = yTemp
          zParticle(emptyParticle) = zTemp
          ageParticle(emptyParticle) = 1
 580      CONTINUE
         ENDIF
        ENDIF
       ENDIF
      ENDIF
     ENDIF
 600 CONTINUE                                                       510

*    Check if time for growth:
     IF (waitUnit) THEN
      IF (timeUnits.EQ.growthWaitUnits) keepWaiting=.FALSE.
     ELSE
      IF (hitUnits.GE.growthWaitUnits) keepWaiting=.FALSE.
     ENDIF

     IF (totalTime.EQ.MAX_TIME) THEN
      PRINT *,'Time exceeded maximum'                               520
      STOP
     ENDIF

 700 END DO
*    End of loop waiting for growth event time


*    Begin growth:
     IF (hitUnits.GT.0) THEN
      growthSites = 0                                               530
      weakest = 1.0
      DO 720 vloc=0,height1
       DO 715 hloc=0,width1
        DO 710 zloc=0,depth1
         IF (hitArray(hloc,vloc,zloc).GE.growthThreshold) THEN
          IF (strengthArraySub(hloc,vloc,zloc).LT.weakest) THEN
           weakest=strengthArraySub(hloc,vloc,zloc)
          ENDIF
         ENDIF
 710     CONTINUE                                                   540
```

```
715   CONTINUE
720   CONTINUE

      DO 900 vloc=0,height1
       DO 890 hloc=0,width1
       DO 880 zloc=0,depth1
        IF (hitArray(hloc,vloc,zloc).GE.growthThreshold) THEN
         bTemp1=(strengthArraySub(hloc,vloc,zloc).LE.growthRange)
         bTemp2=(strengthArraySub(hloc,vloc,zloc).EQ.weakest)
         IF (bTemp1.OR.bTemp2) THEN                                          550

         xTemp = hloc
         yTemp = vloc
         zTemp = zloc
         growthSites = growthSites + 1
         numParticles = numParticles + 1
         imageArray(xTemp,yTemp,zTemp) = numParticles
         bTemp1=((xTemp.EQ.width2).OR.(xTemp.EQ.1))
         bTemp1=(bTemp1.OR.((zTemp.EQ.depth2).OR.(zTemp.EQ.1)))
         bTemp2=((yTemp.EQ.height2).OR.(yTemp.EQ.1))                         560
         IF (bTemp1.OR.bTemp2) BreakThrough=.TRUE.

*        Mark no production area:
         IF (.NOT.uniformParticleDist) THEN
          DO 760 zNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
           zT = zNo + zTemp
           IF (zT.GT.depth1) zT=zT-depth
           IF (zT.LT.0) zT=depth-zT
           DO 755 yNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
            yT = yNo + yTemp                                               570
            IF (yT.GT.height1) yT=yT-height
            IF (yT.LT.0) yT=height-yT
            DO 750 xNo=-NO_PROD_RADIUS,NO_PROD_RADIUS
             xT = xNo + xTemp
             IF (xT.GT.width1) xT=xT-width
             IF (xT.LT.0) xT=width-xT
             k=xNo*xNo + yNo*yNo + zNo*zNo
             IF (SQRT(FLOAT(k)).LE.NO_PROD_RADIUS) THEN
              IF (imageArray(xT,yT,zT).EQ.0) imageArray(xT,yT,zT)=-1
             ENDIF                                                         580
750         CONTINUE
755        CONTINUE
760       CONTINUE
         ENDIF
*        End marking no production area

*        Remove additional particles currently residing on growth site:
         DO 770 i=1,highestParticle
          IF (activeParticle(i)) THEN
           bTemp1 = (xParticle(i).EQ.xTemp)                                590
           bTemp2 = (yParticle(i).EQ.yTemp)
           bTemp2 = (bTemp2.AND.(zParticle(i).EQ.zTemp))
           IF (bTemp1.AND.bTemp2)
     C        activeParticle(i)=.FALSE.
          ENDIF
770       CONTINUE
*        End removal

*        Release additional particles to nearest neighbors of growth site:
         DO 780 i=0,5                                                      600
          longtmp(i)=0
780       CONTINUE
         IF (imageArray((xTemp+1),yTemp,zTemp).LE.0) longtmp(0)=1
         IF (imageArray((xTemp-1),yTemp,zTemp).LE.0) longtmp(1)=1
         IF (imageArray(xTemp,(yTemp+1),zTemp).LE.0) longtmp(2)=1
         IF (imageArray(xTemp,(yTemp-1),zTemp).LE.0) longtmp(3)=1
         IF (imageArray(xTemp,yTemp,(zTemp+1)).LE.0) longtmp(4)=1
         IF (imageArray(xTemp,yTemp,(zTemp-1)).LE.0) longtmp(5)=1
```

```
      numRelease = AINT(releaseAtGrowthSite)
      frac = releaseAtGrowthSite - FLOAT(numRelease)                    610
      IF (RNUNF().LT.frac) numRelease=numRelease+1
      bTemp1=((longtmp(0)+longtmp(1)+longtmp(2)+longtmp(3)+
     C        longtmp(4)+longtmp(5)).EQ.0)
      IF (bTemp1) numRelease=0
      release = 0
      DO 870 WHILE (release.LT.numRelease)
*      Takes into account only the case where homogeneousDif=TRUE
       release = release + 1
       k = AINT(6.0*RNUNF())
       IF (k.EQ.6) k=5                                                  620
       IF (longtmp(k).NE.1) THEN
        release=release-1
       ELSE
        foundEmpty = .FALSE.
        i = 0
        DO 810 WHILE (i.LT.MAX_NUM_PARTICLES)
         i = i + 1
         IF (.NOT.activeParticle(i)) THEN
          emptyParticle = i
          foundEmpty = .TRUE.                                           630
          i = MAX_NUM_PARTICLES
         ENDIF
  810    END DO
        IF (foundEmpty) THEN
         extraParticles = extraParticles + 1
         totalParts = totalParts + 1
         activeParticle(emptyParticle) = .TRUE.
         ageParticle(emptyParticle) = 1
         k=k+1
         GO TO (820,830,840,850,852,854), k                             640
  820     xParticle(emptyParticle) = xTemp + 1
         yParticle(emptyParticle) = yTemp
         zParticle(emptyParticle) = zTemp
         GO TO 860
  830     xParticle(emptyParticle) = xTemp - 1
         yParticle(emptyParticle) = yTemp
         zParticle(emptyParticle) = zTemp
         GO TO 860
  840     xParticle(emptyParticle) = xTemp
         yParticle(emptyParticle) = yTemp + 1                           650
         zParticle(emptyParticle) = zTemp
         GO TO 860
  850     xParticle(emptyParticle) = xTemp
         yParticle(emptyParticle) = yTemp - 1
         zParticle(emptyParticle) = zTemp
         GO TO 860
  852     xParticle(emptyParticle) = xTemp
         yParticle(emptyParticle) = yTemp
         zParticle(emptyParticle) = zTemp + 1
         GO TO 860                                                      660
  854     xParticle(emptyParticle) = xTemp
         yParticle(emptyParticle) = yTemp
         zParticle(emptyParticle) = zTemp - 1
         GO TO 860
  860    ENDIF
        ENDIF
  870   END DO
*       End release

       ENDIF                                                            670
      ENDIF
  880 CONTINUE
  890 CONTINUE
  900 CONTINUE
*     Print '(1X,A,I5,4X,A,I8,4X,A,I6,4X,A,I5)',
*    C      'Counter:',autoCounter,'Time:',totalTime,'Hits:',
```

```
*   C       hitUnits,'Growth Sites:',growthSites
     ENDIF
*    End Growth
                                                                        680
1000 END DO

     PRINT *,'Breakthrough'


*    Find center of mass:
     xc = 0
     yc = 0
     zc = 0
     DO 1050 vloc=0,height1                                             690
      DO 1045 hloc=0,width1
       DO 1040 zloc=0,depth1
        IF (imageArray(hloc,vloc,zloc).GT.0) THEN
         xc = xc + hloc
         yc = yc + vloc
         zc = zc + zloc
        ENDIF
1040   CONTINUE
1045  CONTINUE
1050 CONTINUE                                                           700
     xc = NINT(FLOAT(xc)/FLOAT(numParticles))
     yc = NINT(FLOAT(yc)/FLOAT(numParticles))
     zc = NINT(FLOAT(zc)/FLOAT(numParticles))

     minLength=MIN(xc,yc,zc,(MAX_LENGTH-xc-1),(MAX_LENGTH-yc-1),
    C(MAX_LENGTH-zc-1))/2

     DO 1110 i=0,minLength
      DO 1100 vloc=0-i,i
       DO 1095 hloc=0-i,i                                               710
        DO 1090 zloc=0-i,i
         IF (imageArray((xc+hloc),(yc+vloc),(zc+zloc)).GT.0) THEN
          xc = xc + hloc
          yc = yc + vloc
          zc = zc + zloc
          GO TO 1120
         ENDIF
1090    CONTINUE
1095   CONTINUE
1100  CONTINUE                                                          720
1110 CONTINUE
     PRINT *,'Could not find occupied pixel around center of mass'
     STOP

1120 minLength=MIN(xc,yc,zc,(MAX_LENGTH-xc-1),(MAX_LENGTH-yc-1),
    C(MAX_LENGTH-zc-1))/2
     LARGEST_BOX_INDEX = 1
     DO WHILE (boxSizes(LARGEST_BOX_INDEX).GE.minLength)
      LARGEST_BOX_INDEX = LARGEST_BOX_INDEX + 1
     END DO                                                             730


     pGroupSize = AINT(FLOAT(numParticles)/FLOAT(numGroups))+1
     DO 1140 vloc=0,height1
      DO 1135 hloc=0,width1
       DO 1130 zloc=0,depth1
        pAge=imageArray(hloc,vloc,zloc)
        IF (pAge.GT.0) THEN
         imageArray(hloc,vloc,zloc)=
    C              254-AINT(FLOAT(pAge)/FLOAT(pGroupSize))              740
        ELSE
         imageArray(hloc,vloc,zloc)=0
        ENDIF
1130   CONTINUE
```

```
1135 CONTINUE
1140 CONTINUE

     PRINT *,'Find best images'

     furthest = 0                                                    750
     DO 1160 vloc=0,height1
      DO 1155 hloc=0,width1
       DO 1150 zloc=0,depth1
        IF (imageArray(hloc,vloc,zloc).GT.0) THEN
         yTemp = vloc
         GO TO 1170
        ENDIF
1150   CONTINUE
1155  CONTINUE
1160 CONTINUE                                                        760
1170 IF (yTemp.GT.furthest) furthest=yTemp

     vstart=yTemp

     DO 1190 vloc=height1,0,-1
      DO 1185 hloc=0,width1
       DO 1180 zloc=0,depth1
        IF (imageArray(hloc,vloc,zloc).GT.0) THEN
         yTemp = vloc
         GO TO 1200                                                  770
        ENDIF
1180   CONTINUE
1185  CONTINUE
1190 CONTINUE
1200 IF ((height1-yTemp).GT.furthest) furthest=height1-yTemp

     vend=yTemp

     DO 1220 hloc=0,width1
      DO 1215 vloc=0,height1                                         780
       DO 1210 zloc=0,depth1
        IF (imageArray(hloc,vloc,zloc).GT.0) THEN
         xTemp = hloc
         GO TO 1230
        ENDIF
1210   CONTINUE
1215  CONTINUE
1220 CONTINUE
1230 IF (xTemp.GT.furthest) furthest=xTemp
                                                                     790
     hstart=xTemp

     DO 1250 hloc=width1,0,-1
      DO 1245 vloc=0,height1
       DO 1240 zloc=0,depth1
        IF (imageArray(hloc,vloc,zloc).GT.0) THEN
         xTemp = hloc
         GO TO 1260
        ENDIF
1240   CONTINUE                                                      800
1245  CONTINUE
1250 CONTINUE
1260 IF ((width1-xTemp).GT.furthest) furthest=width1-xTemp

     hend=xTemp

     DO 1272 zloc=0,depth1
      DO 1271 vloc=0,height1
       DO 1270 hloc=0,width1
        IF (imageArray(hloc,vloc,zloc).GT.0) THEN                    810
         zTemp = zloc
         GO TO 1273
```

```
      ENDIF
1270   CONTINUE
1271  CONTINUE
1272 CONTINUE
1273 IF (zTemp.GT.furthest) furthest=zTemp

    dstart=zTemp
                                                                          820
    DO 1277 zloc=depth1,0,-1
     DO 1276 vloc=0,height1
      DO 1275 hloc=0,width1
       IF (imageArray(hloc,vloc,zloc).GT.0) THEN
        zTemp = zloc
        GO TO 1278
       ENDIF
1275   CONTINUE
1276  CONTINUE
1277 CONTINUE                                                             830
1278 IF ((depth1-zTemp).GT.furthest) furthest=depth1-zTemp

    dend=zTemp

    IF (furthest.LT.bestDistance) THEN
     bestDistance=furthest
     DO 1290 vloc=0,height1
      DO 1285 hloc=0,width1
       DO 1280 zloc=0,depth1
        bestImage(hloc,vloc,zloc)=imageArray(hloc,vloc,zloc)              840
1280    CONTINUE
1285   CONTINUE
1290  CONTINUE
    ENDIF

    IF (numParticles.GT.bestMass) THEN
     bestMass = numParticles
     DO 1330 vloc=0,height1
      DO 1325 hloc=0,width1
       DO 1320 zloc=0,depth1                                              850
        heavyImage(hloc,vloc,zloc)=imageArray(hloc,vloc,zloc)
1320    CONTINUE
1325   CONTINUE
1330  CONTINUE
    ENDIF


    PRINT *,'Find mass per radius'

    DO 1340 sizeIndex=LARGEST_BOX_INDEX,NUM_BOX_SIZES                     860
     totalMass(sizeIndex) = 0
     lnBoxSizes(sizeIndex) = ALOG(FLOAT(boxSizes(sizeIndex)))
1340 CONTINUE

    configIdx = 0
    DO 1390 yr=(yc-minLength/2),(yc+minLength/2)
     DO 1385 xr=(xc-minLength/2),(xc+minLength/2)
      DO 1380 zr=(zc-minLength/2),(zc+minLength/2)
      IF (imageArray(xr,yr,zr).GT.0) THEN
       configIdx = configIdx +1                                           870
       DO 1370 sizeIndex=LARGEST_BOX_INDEX,NUM_BOX_SIZES
        mass = 0
        yStart = yr - boxSizes(sizeIndex)
        yEnd = yr + boxSizes(sizeIndex)
        xStart = xr - boxSizes(sizeIndex)
        xEnd = xr + boxSizes(sizeIndex)
        zStart = zr - boxSizes(sizeIndex)
        zEnd = zr + boxSizes(sizeIndex)
        DO 1360 vloc=yStart,yEnd
         DO 1355 hloc=xStart,xEnd                                         880
```

```
        DO 1350 zloc=zStart,zEnd
         temp1=SQRT(FLOAT(vloc-yr)**2+FLOAT(hloc-xr)**2+
     C              FLOAT(zloc-zr)**2)
         IF (temp1.LE.FLOAT(boxSizes(sizeIndex))) THEN
          IF (imageArray(hloc,vloc,zloc).GT.0) mass=mass+1
         ENDIF
1350      CONTINUE
1355      CONTINUE
1360     CONTINUE
       totalMass(sizeIndex) = totalMass(sizeIndex) + mass                    890
1370    CONTINUE
      ENDIF

1380   CONTINUE
1385  CONTINUE
1390 CONTINUE

    DO 1400 sizeIndex=LARGEST_BOX_INDEX,NUM_BOX_SIZES
     lnTotalMass(sizeIndex) =
   C       ALOG(FLOAT(totalMass(sizeIndex))/FLOAT(configIdx))              900
1400 CONTINUE

    PRINT *,'Calculating dimension'

    LAST_BOX_INDEX = 24
    FIRST_BOX_INDEX = LARGEST_BOX_INDEX
    j = LAST_BOX_INDEX - LARGEST_BOX_INDEX + 1
    sx = 0.0
    sy = 0.0
    st2 = 0.0                                                              910
    db = 0.0
    DO 1420 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
     sx = sx + lnBoxSizes(i)
     sy = sy + lnTotalMass(i)
1420 CONTINUE
    ss = FLOAT(LAST_BOX_INDEX - FIRST_BOX_INDEX) + 1.0
    sxoss = sx/ss

    DO 1440 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
     t = lnBoxSizes(i) - sxoss                                             920
     st2 = st2 + t*t
     db = db + t*lnTotalMass(i)
1440 CONTINUE
    db = db/st2
    sddb = SQRT(1.0/st2)
    intcp = (sy - sx*db)/ss
    chi2 = 0.0

    DO 1460 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
     temp1 = lnTotalMass(i) - intcp - db*lnBoxSizes(i)                     930
     chi2 = chi2 + temp1*temp1
1460 CONTINUE
    sigdat = SQRT(chi2/(ss - 2.0))
    sddb = sddb*sigdat

    dimList(autoCounter)=db
    sdDimList(autoCounter)=sddb
    timeMassList(autoCounter)=FLOAT(totalTime)/FLOAT(numParticles)
    tissueEffList(autoCounter)=FLOAT(numParticles)/FLOAT(tissueParts)
    totalEffList(autoCounter)=FLOAT(numParticles)/FLOAT(totalParts)       940
    timeList(autoCounter)=totalTime
    massList(autoCounter)=numParticles
    tissuePartList(autoCounter)=tissueParts
    totalPartList(autoCounter)=totalParts
    decayPartList(autoCounter)=numDecayed


    PRINT *,'Find minimum path'
```

```
      DO 1620 vloc=0,height1                                              950
       DO 1615 hloc=0,width1
        DO 1610 zloc=0,depth1
         minPathArray(hloc,vloc,zloc)=0
1610    CONTINUE
1615   CONTINUE
1620  CONTINUE

      noProgress = .TRUE.
      BreakThrough = .FALSE.
                                                                          960
      stepIndex = 1
      DO 1630 hloc=hstart,hend
       DO 1629 zloc=dstart,dend
        IF (imageArray(hloc,vstart,zloc).GT.0) THEN
         minPathArray(hloc,vstart,zloc)=stepIndex
        ENDIF
1629   CONTINUE
1630  CONTINUE

      DO 1680 WHILE (.NOT.BreakThrough)                                   970
       DO 1660 vloc=vstart,vend
        DO 1655 hloc=hstart,hend
         DO 1650 zloc=dstart,dend
          IF (minPathArray(hloc,vloc,zloc).EQ.stepIndex) THEN

           bTemp1=(imageArray((hloc+1),vloc,zloc).GT.0)
           bTemp2=(minPathArray((hloc+1),vloc,zloc).EQ.0)
           IF (bTemp1.AND.bTemp2) minPathArray((hloc+1),vloc,zloc)=
     C                            stepIndex+1
                                                                          980
           bTemp1=(imageArray((hloc-1),vloc,zloc).GT.0)
           bTemp2=(minPathArray((hloc-1),vloc,zloc).EQ.0)
           IF (bTemp1.AND.bTemp2) minPathArray((hloc-1),vloc,zloc)=
     C                            stepIndex+1

           bTemp1=(imageArray(hloc,(vloc+1),zloc).GT.0)
           bTemp2=(minPathArray(hloc,(vloc+1),zloc).EQ.0)
           IF (bTemp1.AND.bTemp2) minPathArray(hloc,(vloc+1),zloc)=
     C                            stepIndex+1
                                                                          990
           bTemp1=(imageArray(hloc,(vloc-1),zloc).GT.0)
           bTemp2=(minPathArray(hloc,(vloc-1),zloc).EQ.0)
           IF (bTemp1.AND.bTemp2) minPathArray(hloc,(vloc-1),zloc)=
     C                            stepIndex+1

           bTemp1=(imageArray(hloc,vloc,(zloc+1)).GT.0)
           bTemp2=(minPathArray(hloc,vloc,(zloc+1)).EQ.0)
           IF (bTemp1.AND.bTemp2) minPathArray(hloc,vloc,(zloc+1))=
     C                            stepIndex+1
                                                                         1000
           bTemp1=(imageArray(hloc,vloc,(zloc-1)).GT.0)
           bTemp2=(minPathArray(hloc,vloc,(zloc-1)).EQ.0)
           IF (bTemp1.AND.bTemp2) minPathArray(hloc,vloc,(zloc-1))=
     C                            stepIndex+1

           noProgress = .FALSE.
          ENDIF
1650     CONTINUE
1655    CONTINUE
1660   CONTINUE                                                          1010

      IF (noProgress) THEN
       PRINT *,'No Progress Finding Minimum Path'
       STOP
      ENDIF
```

```
      stepIndex=stepIndex+1

      vloc = vend
      hloc = hstart - 1                                                    1020
      DO 1670 WHILE (hloc.LT.hend)
       hloc = hloc + 1
       zloc=dstart-1
       DO 1668 WHILE (zloc.LT.dend)
        zloc=zloc+1
        IF (minPathArray(hloc,vloc,zloc).NE.0) THEN
         BreakThrough = .TRUE.
         PRINT *,'Breakthrough minimum path'
         breakPoint = hloc
         zbreakPoint = zloc                                                1030
         zloc = dend
         hloc = hend
        ENDIF
1668   END DO
1670  END DO

1680 END DO

      vloc = vend
      hloc = breakPoint                                                    1040
      zloc = zbreakPoint
      pointIndex = 0
      stepIndex = minPathArray(hloc,vloc,zloc) - 1
      minPathArray(hloc,vloc,zloc) = -1
      minPath(pointIndex,0) = hloc
      minPath(pointIndex,1) = vloc
      minPath(pointIndex,2) = zloc

      DO 1720 WHILE (vloc.GT.vstart)
       xs = hloc - 1                                                       1050
       ys = vloc - 1
       zs = zloc - 1
       xe = hloc + 1
       ye = vloc + 1
       ze = zloc + 1
       DO 1710 y=ys,ye
        DO 1705 x=xs,xe
         DO 1700 z=zs,ze
          IF (minPathArray(x,y,z).EQ.stepIndex) THEN
           minPathArray(x,y,z) = -1                                        1060
           stepIndex = stepIndex - 1
           pointIndex = pointIndex + 1
           hloc = x
           vloc = y
           zloc = z
           minPath(pointIndex,0) = hloc
           minPath(pointIndex,1) = vloc
           minPath(pointIndex,2) = zloc
           GO TO 1720
          ENDIF                                                            1070
1700     CONTINUE
1705    CONTINUE
1710   CONTINUE
1720 END DO

      DO 1780 sizeIndex=LARGEST_BOX_INDEX,NUM_BOX_SIZES
       eSize = FLOAT(boxSizes(sizeIndex))
       occBox = 0
       pointLoc = 0
       DO 1760 i=0,pointIndex                                              1080
        dx = minPath(i,0) - minPath(pointLoc,0)
        dy = minPath(i,1) - minPath(pointLoc,1)
        dz = minPath(i,2) - minPath(pointLoc,2)
        r1 = SQRT(FLOAT(dx*dx) + FLOAT(dy*dy) + FLOAT(dz*dz))
```

```
      IF (r1.GE.eSize) THEN
       pointLoc = i
       occBox = occBox + 1
      ENDIF
1760  CONTINUE
                                                                      1090
      IF (pointLoc.LT.pointIndex) occBox=occBox+1

      totalMass(sizeIndex) = occBox
      lnBoxSizes(sizeIndex) = ALOG(FLOAT(boxSizes(sizeIndex)))
      lnTotalMass(sizeIndex) = ALOG(FLOAT(totalMass(sizeIndex)))
1780  CONTINUE

      PRINT *,'Calculating minimum-path dimension'

      LAST_BOX_INDEX = 24                                             1100
      FIRST_BOX_INDEX = LARGEST_BOX_INDEX
      j = LAST_BOX_INDEX - LARGEST_BOX_INDEX + 1
      sx = 0.0
      sy = 0.0
      st2 = 0.0
      db = 0.0
      DO 1820 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
       sx = sx + lnBoxSizes(i)
       sy = sy + lnTotalMass(i)
1820  CONTINUE                                                        1110
      ss = FLOAT(LAST_BOX_INDEX - FIRST_BOX_INDEX) + 1.0
      sxoss = sx/ss

      DO 1840 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
       t = lnBoxSizes(i) - sxoss
       st2 = st2 + t*t
       db = db + t*lnTotalMass(i)
1840  CONTINUE
      db = db/st2
      sddb = SQRT(1.0/st2)                                            1120
      intcp = (sy - sx*db)/ss
      chi2 = 0.0

      DO 1860 i=FIRST_BOX_INDEX,LAST_BOX_INDEX
       temp1 = lnTotalMass(i) - intcp - db*lnBoxSizes(i)
       chi2 = chi2 + temp1*temp1
1860  CONTINUE
      sigdat = SQRT(chi2/(ss - 2.0))
      sddb = sddb*sigdat
                                                                      1130
      mpdimList(autoCounter)=0.0-db
      sdMpdimList(autoCounter)=sddb

      i=autoCounter
      WRITE (16,2250) i,dimList(i),sdDimList(i),
     C            mpdimList(i),sdMpdimList(i),timeMassList(i),
     C            tissueEffList(i),totalEffList(i),timeList(i),
     C            massList(i),tissuePartList(i),
     C            totalPartList(i),decayPartList(i)
                                                                      1140
      PRINT '(1X,A,I5,A,I5)','Finished Iteration ',autoCounter,
     C              ' out of ',AUTO_ITERATIONS

      IF (autoCounter.LT.AUTO_ITERATIONS) GO TO 10

      CLOSE (UNIT=16)

      denominator = 0.0
      numerator = 0.0
      DO 1880 i=1,AUTO_ITERATIONS                                     1150
       denominator=denominator+1.0/sdMpdimList(i)**2
       numerator=numerator+mpdimList(i)*(1.0/sdMpdimList(i)**2)
```

```
1880 CONTINUE
     mpdim=numerator/denominator

     numerator = 0.0
     DO 1890 i=1,AUTO_ITERATIONS
      numerator=numerator+(mpdimList(i)-mpdim)**2
1890 CONTINUE
     sdmpdim=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))                    1160

     denominator = 0.0
     numerator = 0.0
     DO 1900 i=1,AUTO_ITERATIONS
      denominator=denominator+1.0/sdDimList(i)**2
      numerator=numerator+dimList(i)*(1.0/sdDimList(i)**2)
1900 CONTINUE
     db=numerator/denominator

     numerator = 0.0                                                     1170
     DO 1910 i=1,AUTO_ITERATIONS
      numerator=numerator+(dimList(i)-db)**2
1910 CONTINUE
     sddb=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

     numerator = 0.0
     DO 1920 i=1,AUTO_ITERATIONS
      numerator=numerator+timeMassList(i)
1920 CONTINUE
     timeMassAvg=numerator/FLOAT(AUTO_ITERATIONS)                        1180

     numerator = 0.0
     DO 1930 i=1,AUTO_ITERATIONS
      numerator=numerator+(timeMassList(i)-timeMassAvg)**2
1930 CONTINUE
     timeMassSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

     numerator = 0.0
     DO 1940 i=1,AUTO_ITERATIONS
      numerator=numerator+tissueEffList(i)                               1190
1940 CONTINUE
     tissueEffAvg=numerator/FLOAT(AUTO_ITERATIONS)

     numerator = 0.0
     DO 1950 i=1,AUTO_ITERATIONS
      numerator=numerator+(tissueEffList(i)-tissueEffAvg)**2
1950 CONTINUE
     tissueEffSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

     numerator = 0.0                                                     1200
     DO 1960 i=1,AUTO_ITERATIONS
      numerator=numerator+totalEffList(i)
1960 CONTINUE
     totalEffAvg=numerator/FLOAT(AUTO_ITERATIONS)

     numerator = 0.0
     DO 1970 i=1,AUTO_ITERATIONS
      numerator=numerator+(totalEffList(i)-totalEffAvg)**2
1970 CONTINUE
     totalEffSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))                 1210

     numerator = 0.0
     DO 1980 i=1,AUTO_ITERATIONS
      numerator=numerator+FLOAT(timeList(i))
1980 CONTINUE
     timeAvg=numerator/FLOAT(AUTO_ITERATIONS)

     numerator = 0.0
     DO 1990 i=1,AUTO_ITERATIONS
      numerator=numerator+(FLOAT(timeList(i))-timeAvg)**2                1220
```

```
1990 CONTINUE
    timeSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

    numerator = 0.0
    DO 2000 i=1,AUTO_ITERATIONS
     numerator=numerator+FLOAT(massList(i))
2000 CONTINUE
    massAvg=numerator/FLOAT(AUTO_ITERATIONS)

    numerator = 0.0                                                    1230
    DO 2010 i=1,AUTO_ITERATIONS
     numerator=numerator+(FLOAT(massList(i))-massAvg)**2
2010 CONTINUE
    massSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

    numerator = 0.0
    DO 2020 i=1,AUTO_ITERATIONS
     numerator=numerator+FLOAT(tissuePartList(i))
2020 CONTINUE
    tissuePartAvg=numerator/FLOAT(AUTO_ITERATIONS)                     1240

    numerator = 0.0
    DO 2030 i=1,AUTO_ITERATIONS
     numerator=numerator+(FLOAT(tissuePartList(i))-tissuePartAvg)**2
2030 CONTINUE
    tissuePartSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

    numerator = 0.0
    DO 2040 i=1,AUTO_ITERATIONS
     numerator=numerator+FLOAT(totalPartList(i))                       1250
2040 CONTINUE
    totalPartAvg=numerator/FLOAT(AUTO_ITERATIONS)

    numerator = 0.0
    DO 2050 i=1,AUTO_ITERATIONS
     numerator=numerator+(FLOAT(totalPartList(i))-totalPartAvg)**2
2050 CONTINUE
    totalPartSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))

    numerator = 0.0                                                    1260
    DO 2060 i=1,AUTO_ITERATIONS
     numerator=numerator+FLOAT(decayPartList(i))
2060 CONTINUE
    decayPartAvg=numerator/FLOAT(AUTO_ITERATIONS)

    numerator = 0.0
    DO 2070 i=1,AUTO_ITERATIONS
     numerator=numerator+(FLOAT(decayPartList(i))-decayPartAvg)**2
2070 CONTINUE
    decayPartSD=SQRT(numerator/FLOAT(AUTO_ITERATIONS-1))               1270

    CALL ITIME(tArray)
    compTime=tArray(3)+tArray(2)*60+tArray(1)*3600-compTime
    PRINT '(1X,A,I10)','Execution Time (secs):  ',compTime

    OPEN (UNIT=19,FILE='Model.Averages',STATUS='NEW')
    WRITE (19,2200) ' '
    WRITE (19,2200) 'Averages:'
    WRITE (19,2240) 'Fractal Dimension:',db,' +/-',sddb
    WRITE (19,2240) 'Minimum-Path Dimension:',mpdim,' +/-',sdmpdim     1280
    WRITE (19,2240) 'Time/Mass:',timeMassAvg,' +/-',timeMassSD
    WRITE (19,2240) 'Tis.  Efficn.:',tissueEffAvg,' +/-',tissueEffSD
    WRITE (19,2240) 'Tot.  Efficn.:',totalEffAvg,' +/-',totalEffSD
    WRITE (19,2240) 'Time:',timeAvg,' +/-',timeSD
    WRITE (19,2240) 'Mass:',massAvg,' +/-',massSD
    WRITE (19,2240) 'Tis.  Prtcls:',tissuePartAvg,' +/-',tissuePartSD
    WRITE (19,2240) 'Tot.  Prtcls:',totalPartAvg,' +/-',totalPartSD
    WRITE (19,2240) 'Dec.  Prtcls:',decayPartAvg,' +/-',decayPartSD
```

```
    CLOSE (UNIT=19)
                                                                          1290
    OPEN (UNIT=17,FILE='Model.Best',STATUS='NEW')
     zloc = depth/2
     DO 2120 vloc=0,height1
      DO 2110 hloc=0,width2
       WRITE (17,2260) bestImage(hloc,vloc,zloc),'T'
2110   CONTINUE
      WRITE (17,2260) bestImage(width1,vloc,zloc),'C'
2120  CONTINUE
    CLOSE (UNIT=17)
                                                                          1300
    OPEN (UNIT=18,FILE='Model.Heavy',STATUS='NEW')
     DO 2140 vloc=0,height1
      DO 2130 hloc=0,width2
       WRITE (18,2260) heavyImage(hloc,vloc,zloc),'T'
2130   CONTINUE
      WRITE (18,2260) heavyImage(width1,vloc,zloc),'C'
2140  CONTINUE
    CLOSE (UNIT=18)

    OPEN (UNIT=18,FILE='Model3.Heavy',STATUS='NEW')                       1310
     DO 2160 vloc=0,height1
      DO 2155 hloc=0,width2
       zloc=depth
       DO 2150 WHILE(zloc.GT.0)
        zloc=zloc-1
        IF (heavyImage(hloc,vloc,zloc).GT.0) GO TO 2152
2150    END DO
2152    WRITE (18,2260) zloc,'T'
2155   CONTINUE
      WRITE (18,2260) heavyImage(width1,vloc,zloc),'C'                    1320
2160  CONTINUE
    CLOSE (UNIT=18)

2200 FORMAT (1X,A)
2210 FORMAT (1X,A,I9)
2220 FORMAT (1X,A,F9.7,1X,A)
2230 FORMAT (1X,A,F9.5)
2240 FORMAT (1X,A,F16.7,A,F16.7)
2250 FORMAT (1X,I3,1X,F6.4,1X,F6.4,1X,F6.4,1X,F6.4,
   C       F8.5,F8.5,F8.5,I7,I6,I7,I7,I6)                                 1330
2260 FORMAT (1X,I4,A)
2270 FORMAT (1X,A,A)

    END
```

A.8 Stress-Based Remodeling Program

The stress-based remodeling program and supporting functions used in the modeling of arteriovenous network formation are listed below. The program and its functions were written using MATLAB.

A.8.1 Main Routine

```
% This routine remodels a square capillary lattice according
% to a shear-stress dependent rule

% latsize = number of nodes per side
latsize=6;

% s = sensitivity parameter
s=0.05;

% dt = time step                                                    10
dt=1
tfinal=1000

% d = segment diamter matrix
% Initially, d is the connectivity matrix
d=conmsq(latsize);

% L = segment length matrix
L=d;
                                                                    20
size=latsize^2;
G=zeros(size);

% G = segment conductance matrix
G(L~=0)=(d(L~=0).^4)./L(L~=0);

known=[1 size];

pknown=[1 0];
                                                                    30
% p = node pressure vector
p=pressure(G,known,pknown);

% Q = segment flow matrix
Q=flow(G,p);

% tau = segment shear stress matrix
% tau0 = segment homeostatic shear stress matrix
tau=zeros(size);
tau0=zeros(size);                                                   40
tau0(d~=0)=Q(d~=0)./((d(d~=0)).^3);

% Set initial perturbation:
d(1,2)=1.21;
d(2,1)=1.21;
d(1,(latsize+1))=1.19;
d((latsize+1),1)=1.19;

% artern = vector of inflow nodes into previously modified segments
artern=[1];                                                         50
% perimn = vector of inflow nodes into segments eligible for remodeling
perimn=[2 (latsize+1)];

d45=[1];
d410=[1];
d910=[1];
d915=[1];
```

```
d1415=[1];
d1420=[1];
d1920=[1];
d1925=[1];

% Main loop:
% iterns = number of generations for remodeling
for iterns=1:3

% dper = diameter matrix of segments eligible for remodeling
 dper=zeros(size);
 dper(perimn,:)=d(perimn,:);
 dper(:,perimn)=d(:,perimn);
 dper(perimn,artern)=zeros(length(perimn),length(artern));
 dper(artern,perimn)=zeros(length(artern),length(perimn));

% Time stepping loop:
 for t=dt:dt:(tfinal*iterns)

% Calculate pressures, flows and shear stresses in all segments:
  G(L~=0)=(d(L~=0).^4)./L(L~=0);
  p=pressure(G,known,pknown);
  Q=flow(G,p);
  tau(d~=0)=Q(d~=0)./((d(d~=0)).^3);

% Change diameter of eligible segments:
  d=d-dper;
  dper(dper~=0)=dper(dper~=0).*(1+(s*dt).*(tau(dper~=0)-tau0(dper~=0))./tau0(dper~=0));
  d=d+dper;

  d45=[d45 d(4,5)];
  d410=[d410 d(4,10)];
  d910=[d910 d(9,10)];
  d915=[d915 d(9,15)];
  d1415=[d1415 d(14,15)];
  d1420=[d1420 d(14,20)];
  d1920=[d1920 d(19,20)];
  d1925=[d1925 d(19,25)];

 end

% Set segment diamter to zero if less than 0.1:
 d(d<0.1)=0.*d(d<0.1);

% Define new set of eligible segments:
 artern=[artern,perimn];
 contem=sum(d(perimn,:));
 itemp=ones(1,length(contem));
 itemp(artern)=zeros(1,length(artern));
 contem=itemp & contem;
 itemp=1:size;
 perimn=itemp(contem);

end
```

A.8.2 Supporting Functions

Connectivity Matrix Generation

```
% function to create a connectivity matrix for a square lattice
% arguments: side = number of nodes per side
% returns: connectivity matrix

function C=conmsq(side)

C=zeros(side*side);

for i=0:(side-2)
  for j=1:(side-1)                                                    10
    k=i*side + j;
    C(k,k+1)=1;
    C(k,k+side)=1;
  end
  k=(i+1)*side;
  C(k,k+side)=1;
  k=side*(side-1) + (i+1);
  C(k,k+1)=1;
end
                                                                      20
C=C+C';
```

Pressure Calculation

```
% function to find the pressures in a network
% courtesy of Dr. James Baish, Bucknell University
% arguments: G = conductance matrix,
%     known = vector of nodes at which pressure is known,
%     pknown = vector of known pressures
% returns: pressure vector

function p=pressure(G, known,pknown)

% n=number of nodes                                                   10
n=length(G);
% i=indices of nodes
i=1:n;
p=0*i;
% all pressures set to zero except known pressures
p(known)=pknown;
% sum conductances over columns (sumG is a vector)
sumG=sum(G);
% subtract sum from main diagonal
A=G-diag(sumG,0);                                                     20
[x,y]=meshgrid(i,known);
% unknown=indicesof unknown pressures
unknown=i(~any(x==y));
% right hand side of matrix equation
B=-G*p';
% right hand side of matrix equation for unknowns only
B=B(unknown);
% left hand side of matrix equation for unknowns only
A=A(unknown,unknown);
% solve matrix equation                                               30
p(unknown)=(sparse(A)\sparse(B))';
```

Flow Calculation

```
% function to find the flow through a network
% courtesy of Dr. James Baish, Bucknell University
% arguments: G = conductance matrix, p = pressure vector
% returns: volumetric flow

function f=flow(G,p)

% n=number of nodes
n=length(G);
% i,j are indices of all non-zero conductances                    10
[i,j,G]=find(G);
% calculate all pressure differences
dp=p(j)-p(i);
% find flows into each node
f=G'.*dp;
% put result in a sparse matrix
f=sparse(i,j,f,n,n);
```

Appendix B

Correlation Dimension Measurements

This appendix contains the fractal dimension measurements of the observed vascular networks in the murine dorsal chamber obtained using the correlation algorithm (see subsection 3.4.3). These measurements are listed separately from the box-counting and sandbox measurements of Chapter 4 because the correlation algorithm was shown to be less accurate than either the box-counting or sandbox algorithms in benchmark tests (see subsection 3.4.4).

Measurements of normal subcutaneous arteriovenous networks in nude mice ($n = 12$) yielded a value of $d_{corr} = 1.71 \pm 0.04$. Measurements of bone-induced arteriovenous networks in nude mice ($n = 10$) yielded a value of $d_{corr} = 1.72 \pm 0.05$. Measurements of normal striated skin muscle capillary networks in nude mice ($n = 12$) yielded a value of $d_{corr} = 2.01 \pm 0.02$. Measurements of human adenocarcinoma LS174T vascular networks in nude mice ($n = 12$) yielded a value of $d_{corr} = 1.94 \pm 0.04$. The results of all these measurements are summarized in Figure B-1. Correlation dimensions measured in C3H mice implanted with three different murine tumor cell lines ($n = 3$ for

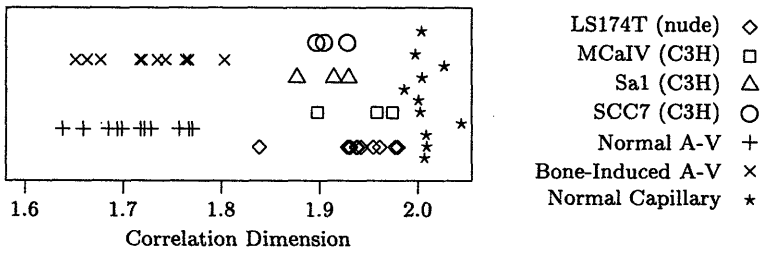

Figure B-1: Correlation dimensions of the observed vascular networks.

Table B.1: Vascular Network Class Separation Based on Correlation Dimension

Network Classes	**Correlation Algorithm**	
	M-W	t-test
Tumor vs. Capillary	$p < 0.0001$	$p < 0.0001$
Tumor vs. A-V	$p < 0.0001$	$p < 0.0001$
A-V vs. Capillary	$p < 0.0001$	$p < 0.0001$
Bone A-V vs. S.C. A-V	$p = 0.6924$	$p = 0.6006$
Tumor (nude) vs. Tumor (C3H)	$p = 0.0646$	$p = 0.2412$

each) fell within the approximate range of the LS174T measurements (see Figure B-1).

When compared to the values obtained using the box-counting or sandbox algorithms (see Chapter 4), the values obtained by the correlation algorithm are slightly but consistently higher. In order to verify whether the correlation dimension results can also be used to separate the observed vascular networks into three distinct classes of scale-invariant behavior, statistical significance tests were carried out similar to those detailed in subsection 4.4.1. The data from Figure B-1 were divided into five pairs of groups. The first three pairs were used to measure between-class separation and consisted of all possible pairings of the groups:

- Arteriovenous networks — normal subcutaneous and bone-induced networks in nude mice ($n = 22$).
- Tumor networks — tumor networks in nude and C3H mice ($n = 21$).
- Capillary networks — normal capillary networks in nude mice ($n = 12$).

The last two pairs were used to measure within-class separation within the arteriovenous network class and within the tumor network class:

- Bone-induced arteriovenous networks in nude mice ($n = 10$) vs. subcutaneous arteriovenous networks in nude mice ($n = 12$).
- LS174T tumor networks in nude mice ($n = 12$) vs. various tumor networks in C3H mice ($n = 9$).

The results are summarized in Table B.1.

The p-values in Table B.1 show that the correlation dimension clearly separates the three observed classes of two-dimensional vascular networks: normal arteriovenous networks, normal capillary networks, and tumor networks. The p-values in Table B.1 also show that the correlation dimensions of bone-induced and subcutaneous arteriovenous networks are statistically similar. Correlation dimensions for tumor networks in nude mice (LS174T tumors) and tumor networks in C3H mice (MCaIV, SCC7 and Sa1 tumors) are also statistically similar.

The results listed above are in accord with the results obtained using the box-counting and sandbox algorithms in Chapter 4. Therefore, the conclusions reached in Chapter 4 are valid for correlation dimension measurements too.

Appendix C

Fractal Dimensions in Tumor Regression

In subsection 4.2.2 the time dependence of vascular network fractal dimensions during tumor growth was described. As this thesis was nearing completion, a new in vivo hormone-dependent regressing tumor model was being developed in the lab. In this appendix, the fractal dimension measurements from two individual regressing tumors are described. While many of the conclusions are not final due to the small sample size ($n = 2$), some general trends can be discerned.

The tumor line used was Shionogi (SC115) murine mammary carcinoma, whose growth is known to be androgen-dependent [24]. The tumors were grown using the dorsal window preparation in SCID mice. The experimental methods used to grow the tumors, acquire the images, and measure the fractal dimensions were similar to those described in Chapter 3, with one exception. In the experiments described in Chapter 3, a dense tumor cell suspension from cell culture was inoculated onto the dorsal skinfold. In the Shionogi tumor experiments, a piece of tumor tissue (from another animal) having a linear dimension of 2 mm was placed in the dorsal chamber of male mice. This difference was due to the inability to grow the tumor rapidly in the dorsal chamber from a cell suspension.

Preliminary microscopic observations showed that as the tumor grew, blood vessels became larger and more tortuous. After hormone depletion by orchiectomy, tumor microvascular morphology changed markedly. As the tumor shrank, vessel diameters decreased and vessels assumed a more linear appearance. Vessel rarefaction appeared to precede tumor regression.

Fractal dimensions were measured when there was a large enough number of vessel segments in the tumor. Since recordings were not made daily, measurements were performed on images acquired on days 11,15,16,18 after tumor implantation. Orchiectomy was performed on day 14, and the tumor regressed during the days that followed. The measurements were performed for the two

Table C.1: Fractal Dimensions of Regressing Shionogi Tumor Networks

Dimension	day 11	day 15	day 16	day 18
d_{box} (tumor 1)	1.87 ± 0.02	1.85 ± 0.03	1.78 ± 0.01	1.79 ± 0.01
d_{box} (tumor 2)	1.87 ± 0.02	1.91 ± 0.02	1.82 ± 0.03	1.81 ± 0.03
d_{min} (tumor 1)	1.15 ± 0.01	1.12 ± 0.01	1.09 ± 0.01	1.09 ± 0.01
d_{min} (tumor 2)	1.07 ± 0.01	1.08 ± 0.01	1.05 ± 0.01	1.07 ± 0.01

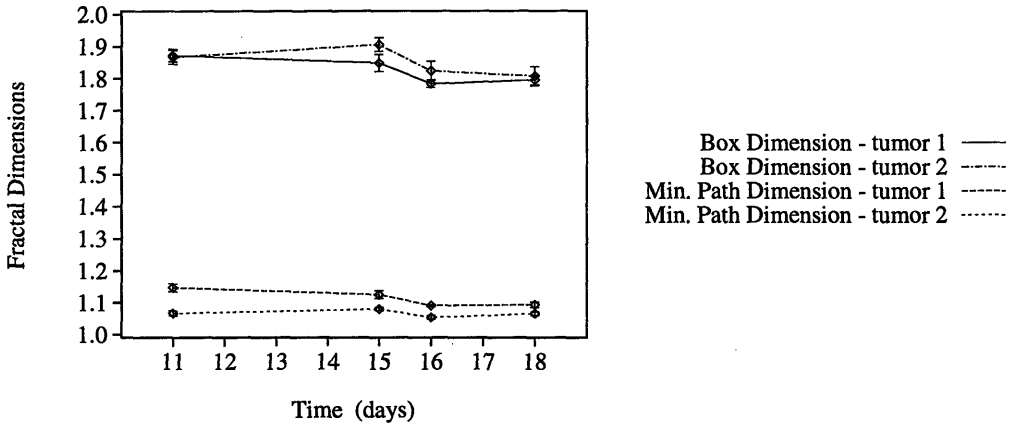

Figure C-1: Fractal dimensions of Shionogi tumor networks during regression. Orchiectomy was performed on day 14, after which the tumors began regressing.

tumors whose optical quality was adequate during all four recording days. The measured values are reported in Table C.1. The values are reported with the standard error of the curve fit. A graphic representation of these results is shown in Figure C-1.

The fractal dimension values measured on day 11, when the tumor was in the growth phase, and on day 15, immediately after orchiectomy, are consistent with the values reported for all other growing tumors (see section 4.2). However, from day 15 to day 16, there is a significant reduction in all fractal dimensions. The decreasing fractal dimension implies that the size of avascular areas increased from day 15 to day 16 (relative to tumor size). This finding is consistent with the observation that vessel rarefaction appeared to precede tumor regression. The decreasing minimum-path dimension is consistent with the observation that vessels appeared to assume a more linear morphology. From day 16 to day 18 there appear to be no statistically significant changes in fractal dimensions. This finding can be interpreted as there having been achieved some sort of balanced rate for vessel resorption and tumor regression. From a scale-invariant vantage, such a balance would maintain the same fractal dimension as the tumor shrank.

The above conclusions are speculative in nature because of the small sample size ($n = 2$) involved. However, they point to one potentially important hypothesis — that the Shionogi tumor networks regress because of vessel resorption (and not vice versa). Measurements in a larger number of tumors are necessary to confirm the aforementioned findings.

Appendix D

In Vitro Verification of Autocrine Mechanism

The growth model for normal capillary networks advanced in Chapter 5 suggested that an autocrine mechanism for growth factor release by endothelial cells is key to normal angiogenesis. In order to corroborate that suggestion, a series of in vitro experiments was designed to examine whether directional dependence in growth factor administration would manifest itself in the morphology of an endothelial cell layer in culture.

While this experiment was being conducted, other investigators published the results of a different experiment which demonstrated more conclusively and elegantly the existence of an autocrine mechanism for angiogenesis [48] (described briefly at the end of section 5.2). It is for this reason that the results of our in vitro experiments are relegated to an appendix, and will be described here in brief only.

In Chapter 5, the autocrine mechanism was proposed to explain the observed absence of gradient sensitive growth during normal angiogenesis. Therefore, it was hypothesized that if growth factor was administered in vitro to an endothelial cell monolayer either in a single dose, which would create a uniform concentration, or by controlled release, which would create a gradient, the morphologic attributes of the layer would be essentially the same in both systems, because the autocrine release of growth factor would "mask" the gradient.

We set out to test this hypothesis by looking at the orientation of endothelial cells in vitro following administration of bFGF. Bovine capillary endothelial cells were prepared and plated on a collagen gel covered with a liquid medium using standard techniques described in the literature [95]. In one set of dishes bFGF in concentrations of 25, 50, or 100 ng/ml was applied topically in one dose 2 days after plating. This procedure assured uniform bFGF concentration throughout the dish. In another set of dishes, a disk approximately 2 mm in diameter composed of heparin sepharose

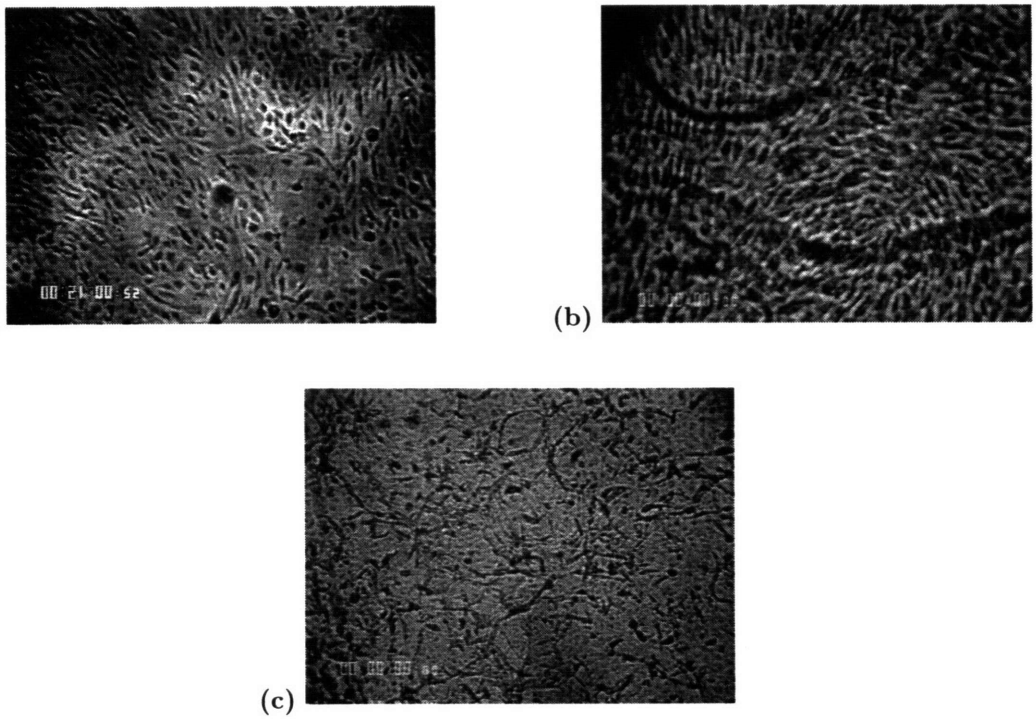

Figure D-1: Morphological archetypes in endothelial cell cultures: **(a)** Newly plated endothelial cells showing random orientation and typified by cell-less voids; **(b)** Endothelial cells in vicinity of disk 5 days after plating, showing locally oriented groups of cells yet no global orientation (i.e., it is impossible to discern the location of the disk); **(c)** Randomly oriented endothelial cell sprouts in a culture where bFGF was topically applied.

coated beads bound with bFGF, was implanted in the periphery of the dish. Since heparin strongly binds bFGF [33], the bFGF was slowly released, thus creating a growth factor gradient across the culture dish. The heparin concentration was adjusted to the level just necessary to elicit visible morphological effects in the endothelial cells.

After approximately a week of transient morphological changes, the endothelial cells achieved a nearly constant morphology, regardless of initial growth factor concentration. This morphology was characterized by dividing the layer into three areas: disk vicinity ("Area 1"), center of dish ("Area 2"), and diametrically opposite the disk ("Area 3"). In dishes without disks corresponding areas were chosen. In each of these areas the morphology of the monolayer and of the sprouts, if any, was described. The monolayer morphology was divided into one of three classes: randomly oriented cells with no directionality, cells in groups of uniform (yet unpredictable) directionality, or cells with a global orientation (as would be observed in gradient-sensitive growth). The sprouts were also visually examined for any preferred orientation, either linear or radial. Results are summarized in Table D.1. The morphological archetypes described in Table D.1 are demonstrated in Figure D-1.

The most important finding was that under no circumstances was global directionality observed in the monolayer of endothelial cells. While groups of cells could be found oriented in a uniform

Table D.1: Morphological Characteristics of Endothelial Cell Cultures

	Controlled Release			**Single Dose**		
	Area 1	Area 2	Area 3	Area 1	Area 2	Area 3
Monolayer	LO	LO	RO	LO	LO	LO
Sprouts	RO	No sprouts	No sprouts	RO	RO	RO
RO = Randomly oriented, no particular orientation LO = Locally oriented groups, but no global orientation GO = Globally oriented, as would be expected in diffusion-limited growth						

direction (see Figure D-1b), this direction would change randomly from one group to another, thus arguing for the existence of a local mechanism for determining orientation, but little global influence. Furthermore, the orientation of the sprouts (tube-like structures resembling capillaries) was random in all cases that sprouts were observed. A weak directional influence was observed in the dishes with disks since sprouts were found only in the proximity of the disks. However, the lack of global endothelial cell orientation even near the disk seems to argue against gradient-sensitive growth. This observation could be explained by postulating that bFGF levels away from the disk had fallen so much that even the autocrine mechanism could not be triggered beyond a certain distance from the disk.

In summary, the lack of any morphological attributes typical of diffusion-limited growth [131] supports the idea of an autocrine mechanism that "masks" the gradients generated using the heparin sepharose disk system. While these experiments were limited in scope, they represented an important preliminary step in corroborating the predictions of Chapter 5. The recent publication of a more definitive experiment [48] provided unequivocal evidence for an autocrine mechanism underlying endothelial cell proliferation.

Bibliography

[1] H. Ahlstrom, R. Christofferson, and L. E. Lorelius. Vascularization of the continuous human colonic cancer cell line LS 174 T deposited subcutaneously in nude rats. *APMIS*, 96:701–710, 1988.

[2] W. E. Allen and D. J. Wilson. Early embryonic angiogenesis in the chick area vasculosa. *J Anat*, 183:579–585, 1993.

[3] O. Arend, S. Wolf, F. Jung, B. Bertram, H. Pöstgens, H. Toonen, and M. Reim. Retinal microcirculation in patients with diabetes mellitus: Dynamic and morphological analysis of perifoveal capillary network. *Brit J Ophthalmol*, 75:514–518, 1991.

[4] J. W. Baish. Formulation of a statistical model of heat transfer in perfused tissue. *J Biomech Eng*, 116:521–527, 1994.

[5] J. W. Baish, Y. Gazit, D. A. Berk, L. T. Baxter, and R. K. Jain. Fractal dimension as a measure of microvascular heterogeneity in tumors: Implications for transport. In *Proc. 42nd Annual Conference of the Microcirculatory Society*, April 1995.

[6] J. W. Baish, Y. Gazit, D. A. Berk, M. Nozue, L. T. Baxter, and R. K. Jain. An invasion percolation model of architectural obstacles to transport in tumors. *Ann Biomed Eng*, 23:S–28, 1995.

[7] J. W. Baish, Y. Gazit, D. A. Berk, M. Nozue, L. T. Baxter, and R. K. Jain. Role of tumor vascular architecture in nutrient and drug delivery: An invasion percolation based network model. *Microvasc Res*, In press, 1996.

[8] D. Balding and D. L. S. McElwain. A mathematical model of tumour-induced capillary growth. *J Theor Biol*, 114:53–73, 1985.

[9] J. B. Bassingthwaighte, L. S. Liebovitch, and B. J. West. *Fractal Physiology*. Oxford University Press, New York, 1994.

[10] E. Ben-Jacob, H. Shmueli, O. Shochet, and A. Tenenbaum. Adaptive self-organization during growth of bacterial colonies. *Physica A*, 187:378–424, 1992.

[11] R. A. O. Bennett, R. N. Pittman, and S. M. Sullivan. Capillary spatial pattern and muscle fiber geometry in three hamster striated muscles. *Am J Physiol*, 260:H579–H585, 1991.

[12] D. A. Berk, F. Yuan, M. Leunig, and R. K. Jain. Fluorescence photobleaching with spatial Fourier analysis: Measurement of diffusion in light-scattering media. *Biophys J*, 65:2428–2436, 1993.

[13] P. C. Brooks, A. M. P. Montgomery, M. Rosenfeld, R. A. Reisfeld, T. Hu, G. Klier, and D. A. Cheresh. Integrin $\alpha_v\beta_3$ antagonists promote tumor regression by inducing apoptosis of angiogenic blood vessels. *Cell*, 79:1157–1164, 1994.

[14] A. Bunde and S. Havlin. A brief introduction to fractal geometry. In A. Bunde and S. Havlin, editors, *Fractals in Science*, pages 1–25. Springer, Berlin, 1994.

[15] P. H. Burri and M. R. Tarek. A novel mechanism of capillary growth in the rat pulmonary microcirculation. *Anat Rec*, 228:35–45, 1990.

[16] G. J. Burton and M. E. Palmer. The chorioallantoic capillary plexus of the chicken egg: A microvascular corrosion casting study. *Scanning Microscop*, 3:549–558, 1989.

[17] N. A. Busch and I. A. Silver. Three dimensional reconstruction of branched tree structures from serial sections. *Adv Exp Med Biol*, 222:77–86, 1988.

[18] F. Caserta, H. E. Stanley, W. D. Eldred, G. Daccord, R. E. Hausman, and J. Nittmann. Physical mechanisms underlying neurite growth: A quantitative analysis of neuronal shape. *Phys Rev Lett*, 64:95–98, 1990.

[19] I. I. H. Chen and R. L. Prewitt. A mathematical representation for vessel network. *J Theor Biol*, 98:211–219, 1982.

[20] S. E. Connolly, T. A. Hores, L. E. H. Smith, and P. A. D'Amore. Characterization of vascular development in the mouse retina. *Microvasc Res*, 36:275–290, 1988.

[21] P. A. D'Amore. Modes of FGF release in vivo and in vitro. *Cancer Metast Rev*, 9:227–238, 1990.

[22] G. E. Davis and C. W. Camarillo. Regulation of endothelial cell morphogenesis by integrins, mechanical forces, and matrix guidance pathways. *Exp Cell Res*, 216:113–123, 1995.

[23] A. Daxer. Fractals and retinal vessels. *Lancet*, 339:618, 1992.

[24] W. J. Desmond, Jr., S. J. Wolbers., and G. Sato. Cloned mouse mammary cell lines requiring androgens for growth in culture. *Cell*, 8:79–86, 1976.

[25] M. W. Dewhirst, C. Y. Tso, R. Oliver, C. S. Gustafson, T. W. Secomb, and J. F. Gross. Morphologic and hemodynamic comparison of tumor and healing normal tissue microvasculature. *Int J Rad Oncol Biol Phys*, 17:91–99, 1989.

[26] E. T. Engelson, T. C. Skalak, and G. W. Schmid-Schonbein. The microvasculature in skeletal muscle. I. Arteriolar networks in rat spinotrapezius muscle. *Microvasc Res*, 30:29–44, 1985.

[27] K. J. Falconer. *Fractal Geometry: Mathematical Foundations and Applications.* Wiley, Chichester, 1990.

[28] F. Family, B. R. Masters, and D. E. Platt. Fractal pattern formation in human retinal vessels. *Physica D*, 38:98–103, 1989.

[29] B. M. Fenton and B. W. Zweifach. Microcirculatory model relating geometrical variations to changes in pressure and flow rate. *Ann Biomed Eng*, 9:303–321, 1981.

[30] I. J. Fidler and L. M. Ellis. The implications of angiogenesis for the biology and therapy of cancer metastasis. *Cell*, 79:185–188, 1994.

[31] S. B. Field, I. A. Burney, S. Needham, R. J. Maxwell, J. Coggle, and J. R. Griffiths. Are transplanted tumours suitable as models for studies on vasculature? *Int J Radiat Biol*, 60:255–260, 1991.

[32] G. F. Fishman and L. R. Moore III. A statistical evaluation of multiplicative random number generators with modulus $2^{31}-1$. *J Am Stat Assoc*, 77:129–136, 1982.

[33] J. Folkman. Tumor angiogenesis. In J. F. Holland, E. Frei, R. C. Bast, D. W. Kufe, D. L. Morton, and R. R. Weichselbaum, editors, *Cancer Medicine*, pages 153–170. Lea & Febiger, Philadelphia, 1993.

[34] J. Folkman. Seminars in medicine of the Beth Israel Hospital, Boston: Clinical applications of research on angiogenesis. *N Engl J Med*, 333:1757–1763, 1995.

[35] J. Folkman. Tumor angiogenesis. In J. Mendelsohn, P. M. Howley, M. A. Israel, and L. A. Liotta, editors, *The Molecular Basis of Cancer*, pages 206–232. W. B. Saunders, Philadelphia, 1995.

[36] J. Folkman and Y. Shing. Angiogenesis. *J Biol Chem*, 267:10931–10934, 1992.

[37] A. W. Fryczkowski and M. D. Sherman. Scanning electron microscopy of human ocular vascular casts: The submacular choriocapillaries. *Acta Anat*, 132:265–269, 1988.

[38] Y. C. Fung. *Biomechanics: Motion, Flow, Stress, and Growth.* Springer, New York, 1990.

[39] J. R. Gamble, L. J. Matthias, G. Meyer, P. Kaur, G. Russ, R. Faull, M. C. Berndt, and M. A. Vadas. Regulation of in vitro capillary tube formation by anti-integrin antibodies. *J Cell Biol*, 121:931–943, 1993.

[40] R. Z. Gan, Y. Tian, R. T. Yen, and G. S. Kassab. Morphometry of the dog pulmonary venous tree. *J Appl Physiol*, 75:432–440, 1993.

[41] Y. Gazit, J. W. Baish, L. T. Baxter, and R. K. Jain. A model for the formation of arterial networks. *Bull Amer Phys Soc*, 41:429–430, 1996.

[42] Y. Gazit, D. Berk, M. Leunig, L. T. Baxter, and R. K. Jain. Fractal analysis of two-dimensional normal and tumor vascular networks. *Bull Amer Phys Soc*, 39:596–597, 1994.

[43] Y. Gazit, D. Berk, M. Leunig, L. T. Baxter, and R. K. Jain. Fractal analysis of two-dimensional normal and tumor vascular networks and its implications. In *Proc. 41st Annual Conference of the Microcirculatory Society*, page 42, April 1994.

[44] Y. Gazit, D. A. Berk, L. T. Baxter, and R. K. Jain. Vascular network growth modeling based on fractal analysis. *Bull Amer Phys Soc*, 40:411, 1995.

[45] Y. Gazit, D. A. Berk, M. Leunig, L. T. Baxter, and R. K. Jain. Scale-invariant behavior and vascular network formation in normal and tumor tissue. *Phys Rev Lett*, 75:2428–2431, 1995.

[46] Y. Gazit, S. Patan, D. Berk, M. Leunig, L. Munn, L. Baxter, and R. K. Jain. New insights into the structure of tumor vasculature: From fractal dimensions to intussusception. *Biorheology*, 32:112, 1995.

[47] M. E. Gottlieb. Angiogenesis and vascular networks: Complex anatomies from deterministic non-linear physiologies. In J. M. Garcia-Ruiz, E. Louis, P. Meakin, and L. M. Sander, editors, *Growth Patterns in Physical Sciences and Biology*, pages 267–276. Plenum, New York, 1993.

[48] A. Gualandris, M. Rusnati, M. Belleri, E. E. Nelli, M. Bastaki, M. P. Molinari-Tosatti, F. Bonardi, S. Parolini, A. Albini, L. Morbidelli, M. Ziche, A. Corallini, L. Possati, A. Vacca, D. Ribatti, and M. Presta. Basic fibroblast growth factor overexpression in endothelial cells: An autocrine mechanism for angiogenesis and angioproliferative diseases. *Cell Growth Diff*, 7:147–160, 1996.

[49] P. M. Gullino. Extracellular compartments of solid tumors. In F. F. Becker, editor, *Cancer: A Comprehensive Treatise*, volume 3, pages 327–354. Plenum, New York, 1975.

[50] H. J. Herrmann and H. E. Stanley. The fractal dimension of the minimum path in two- and three-dimensional percolation. *J Phys A*, 21:L829–L833, 1988.

[51] L. Hesse, J. Chofflet, and Y. Le-Mer. Simulation of the growth pattern of the central retinal artery using a fractal modeling technique. *German J Ophthalmol*, 2:116–118, 1993.

[52] D. E. Hilmas and E. L. Gillette. Morphometric analysis of the microvasculature of tumors during growth and after x-irradiation. *Cancer*, 33:103–110, 1974.

[53] K. Hori, M. Suzuki, S. Tanda, and S. Saito. In vivo analysis of tumor vascularization in the rat. *Jpn J Cancer Res*, 81:279–288, 1990.

[54] K. Horsfield and A. Thurlbeck. Relation between diameter and flow in branches of the bronchial tree. *Bull Math Biol*, 43:681–691, 1981.

[55] H.-J. Hsieh, N.-Q. Li, and J. A. Frangos. Shear stress increases endothelial platelet-derived growth factor mRNA levels. *Am J Physiol*, 260:H642–H646, 1991.

[56] A. G. Hudetz. Percolation phenomenon: The effect of capillary network rarefaction. *Microvasc Res*, 45:1–10, 1993.

[57] D. E. Ingber. Fibronectin controls capillary endothelial cell growth by modulating cell shape. *Proc Nat Acad Sci USA*, 87:3579–3583, 1990.

[58] D. E. Ingber. Extracellular matrix as a solid-state regulator in angiogenesis: Identification of new targets for anti-cancer therapy. *Semin Cancer Biol*, 3:57–63, 1992.

[59] R. K. Jain. Transport of molecules in the tumor interstitium: A review. *Cancer Res*, 47:3039–3051, 1987.

[60] R. K. Jain. Determinants of tumor blood flow: A review. *Cancer Res*, 48:2641–2658, 1988.

[61] R. K. Jain. Delivery of novel therapeutic agents in tumors: Physiological barriers and strategies. *J Natl Cancer Inst*, 81:570–576, 1989.

[62] R. K. Jain. Barriers to drug delivery in solid tumors. *Sci Am*, 271:58–65, July 1994.

[63] L. C. Junqueira, J. Carneiro, and J. A. Long. *Basic Histology*. Lange, Los Altos, CA, fifth edition, 1986.

[64] F. Kallinowski, R. Zander, M. Hoeckel, and P. Vaupel. Tumor tissue oxygenation as evaluated by computerized pO_2 histography. *Int J Rad Oncol Biol Phys*, 19:953–961, 1990.

[65] G. S. Kassab, C. A. Rider, N. Tang, and Y.-C. B. Fung. Morphometry of pig coronary arterial trees. *Am J Physiol*, 265:H350–H365, 1993.

[66] M. F. Kiani and A. G. Hudetz. Computer simulation of growth of anastamosing microvascular networks. *J Theor Biol*, 150:547–560, 1991.

[67] A. Koller, B. Dawant, A. Liu, A. S. Popel, and P. C. Johnson. Quantitative analysis of arteriolar network architecture in cat sartorius muscle. *Am J Physiol*, 253:H154–H164, 1987.

[68] M. A. Konerding, F. Steinberg, and V. Budach. The vascular system of xenotransplanted tumors — scanning electron and light microscopic studies. *Scanning Microsc*, 3:327–336, 1989.

[69] N. Koyama, S. Watanabe, M. Tezuka, N. Morisaki, Y. Saito, and S. Yoshida. Migratory and proliferative effect of platelet-derived growth factor in rabbit retinal endothelial cells: Evidence of an autocrine pathway of platelet-derived growth factor. *J Cell Physiol*, 158:1–6, 1994.

[70] R. Krauss. New technique to demonstrate the network of blood capillaries of the human retina in their three-dimensional arrangement. *Graefe's Arch Clin Exp Ophthalmol*, 228:187–190, 1990.

[71] G. Landini and G. Mission. Simulation of corneal neovascularization by inverted diffusion limited aggregation. *Invest Ophthalmol Vis Sci*, 34:1872–1875, 1993.

[72] G. Landini, G. P. Mission, and P. I. Murray. Fractal analysis of the normal human retinal fluorescein angiogram. *Curr Eye Res*, 12:23–27, 1993.

[73] H. A. Lehr, M. Leunig, M. D. Menger, D. Nolte, and K. Messmer. Dorsal skinfold chamber technique for intravital microscopy in nude mice. *Am J Pathol*, 143:1055–1062, 1993.

[74] D. E. Lemons, S. Chien, L. I. Crawshaw, S. Weinbaum, and L. M. Jiji. Significance of vessel size and type in vascular heat transfer. *Am J Physiol*, 253:R128–R135, 1987.

[75] R. Lenormand. Flow through porous media: Limits of fractal patterns. In M. Fleischmann, D. J. Tildesley, and R. C. Ball, editors, *Fractals in the Natural Sciences*, pages 159–168. Princeton University Press, Princeton, 1990.

[76] J. R. Less, T. C. Skalak, E. M. Sevick, and R. K. Jain. Microvascular architecture in mammary carcinoma: Branching patterns and vessel dimensions. *Cancer Res*, 51:265–273, 1991.

[77] M. Leunig, F. Yuan, D. A. Berk, L. E. Gerweck, and R. K. Jain. Angiogenesis and growth of isografted bone: Quantitative in vivo assay in nude mice. *Lab Invest*, 71:300–307, 1994.

[78] M. Leunig, F. Yuan, M. D. Menger, Y. Boucher, A. E. Goetz, K. Messmer, and R. K. Jain. Angiogenesis, microvascular architecture, microhemodynamics, and interstitial fluid pressure during early growth of human adenocarcinoma LS174T in SCID mice. *Cancer Res*, 52:6553–6560, 1992.

[79] K. Ley, A. R. Pries, and P. Gaehtgens. Topological structure of rat mesenteric microvessel networks. *Microvasc Res*, 32:315–332, 1986.

[80] H. H. Lipowsky and B. W. Zweifach. Network analysis of microcirculation of cat mesentery. *Microvasc Res*, 7:73–83, 1974.

[81] F. Mahler, G. Nagel, H. Saner, and F. Kneubuhl. In vivo comparison of the nailfold capillary diameter as determined by using the erythrocyte column and FITC-labelled albumin. *Int J Microcirc Clin Exp*, 2:147–155, 1983.

[82] M. A. Mainster. The fractal properties of retinal vessels: Embryological and clinical implications. *Eye*, 4:235–241, 1990.

[83] J. E. Maloney and B. L. Castle. Pressure-diameter relations of capillaries and small blood vessels in frog lung. *Resp Physiol*, 7:150–162, 1969.

[84] B. Mandelbrot. How long is the coast of Britain? Statistical self-similarity and fractional dimension. *Science*, 156:636–638, 1967.

[85] B. B. Mandelbrot. *The Fractal Geometry of Nature.* W. H. Freeman, San Francisco, 1982.

[86] D. Mark and C. Reed. *Inside the Toolbox using THINK Pascal*, volume 1 of *Macintosh Pascal Programming Primer.* Addison-Wesley, Reading, 1991.

[87] M. Matsushita and H. Fujikawa. Diffusion-limited growth in bacterial colony formation. *Physica A*, 168:498–506, 1990.

[88] D. S. McLeod, G. A. Lutty, S. D. Wajer, and R. W. Flower. Visualization of a developing vasculature. *Microvasc Res*, 33:257–269, 1987.

[89] P. G. McMenamin and W. J. Krause. Morphological observations on the unique paired capillaries of the opossum retina. *Cell Tissue Res*, 271:461–468, 1993.

[90] P. Meakin. Diffusion-controlled cluster formation in 2–6-dimensional space. *Phys Rev A*, 27:1495–1507, 1983.

[91] P. Meakin, I. Majid, S. Havlin, and H. E. Stanley. Topological properties of diffusion limited aggregation and cluster-cluster aggregation. *J Phys A*, 17:L975–L981, 1984.

[92] F. A. Merchant, S. J. Aggarwal, K. R. Diller, and A. C. Bovik. In-vivo analysis of angiogenesis and revascularization of transplanted pancreatic islets using confocal microscopy. *J Microscop*, 176:262–275, 1994.

[93] J. W. Miller, A. P. Adamis, D. T. Shima, P. A. D'Amore, R. S. Moulton, M. S. O'Reilly, J. Folkman, H. F. Dvorak, L. F. Brown, B. Berse, T.-K. Yeo, and K.-T. Yeo. Vacular endothelial growth factor/vascular permeability factor is temporally and spatially correlated with ocular angiogenesis in a primate model. *Am J Pathol*, 145:574–584, 1994.

[94] R. Montesano, L. Orci, and P. Vassalli. In vitro rapid organization of endothelial cells into capillary-like networks is promoted by collagen matrices. *J Cell Biol*, 97:1648–1652, 1983.

[95] R. Montesano, M. S. Pepper, and L. Orci. Paracrine induction of angiogenesis in vitro by Swiss 3T3 fibroblasts. *J Cell Sci*, 105:1013–1024, 1993.

[96] D. Moscatelli. High and low affinity binding sites for basic fibroblast growth factor on cultured cells: Absence of a role for low affinity binding in the stimulation of plasminogen activator production by bovine capillary endothelial cells. *J Cell Physiol*, 131:123–130, 1987.

[97] C. D. Murray. The physiological principle of minimum work. I. The vascular system and the cost of blood volume. *Proc Nat Acad Sci USA*, 12:207–214, 1926.

[98] T. Nagel, N. Resnick, W. J. Atkinson, C. F. Dewey, Jr., and M. A. Gimbrone, Jr. Shear stress selectively upregulates intercellular adhesion molecule-1 expression in cultured human vascular endothelial cells. *J Clin Invest*, 94:885–891, 1994.

[99] J. A. Nagy, L. F. Brown, D. R. Senger, N. Lanir, L. van de Water, A. M. Dvorak, and H. F. Dvorak. Pathogenesis of tumor stroma generation: A critical role for leaky blood vessels and fibrin deposition. *Biochim Biophys Acta*, 948:305–326, 1988.

[100] R. F. Nicosia, R. Tchao, and J. Leighton. Angiogenesis-dependent tumor spread in reinforced fibrin clot culture. *Cancer Res*, 43:2159–2166, 1983.

[101] J. Nittmann, G. Daccord, and H. E. Stanley. Fractal growth of viscous fingers: Quantitative chracterization of a fluid instability phenomenon. *Nature*, 314:141–144, 1985.

[102] M. U. Nollert, N. J. Panaro, and L. V. McIntire. Regulation of genetic expression in shear stress-stimulated endothelial cells. *Ann NY Acad Sci*, 665:94–104, 1992.

[103] S. Patan, L. L. Munn, and R. K. Jain. Intussusceptive microvascular growth in a human colon adenocarcinoma xenograft: A novel mechanism of tumor angiogenesis. *Microvasc Res*, 51:260–272, 1996.

[104] R. N. Pittman. Influence of microvascular architecture on oxygen exchange in skeletal muscle. *Microcirc*, 2:1–18, 1995.

[105] M. J. Plyley, G. K. Sutherland, and A. C. Groom. Geometry of the capillary network in skeletal muscle. *Microvasc Res*, 11:161–173, 1976.

[106] A. S. Popel. Network models of peripheral circulation. In R. Skalak and S. Chien, editors, *Handbook of Bioengineering*, pages 20.1–20.24. McGraw-Hill, New York, 1986.

[107] R. F. Potter and A. C. Groom. Capillary diameter and geometry in cardiac and skeletal muscle studied by means of corrosion casts. *Microvasc Res*, 25:68–84, 1983.

[108] R. J. Price, G. K. Owens, and T. C. Skalak. Immunohistochemical identification of arteriolar development using markers of smooth muscle differentiation. *Circ Res*, 75:520–527, 1994.

[109] R. J. Price and T. C. Skalak. Circumferential wall stress as a mechanism for arteriolar rarefaction and proliferation in a network model. *Microvasc Res*, 47:188–202, 1994.

[110] R. Rosso, B. Bacchi, and P. La Barbera. Fractal relation of mainstream length to catchment area in river networks. *Water Resour Res*, 27:381–387, 1991.

[111] J. C. Russ. *The Image Proccesing Handbook.* CRC Press, Boca Raton, second edition, 1994.

[112] S. Schröder, M. Brab, G. W. Schmid-Schonbein, M. Reim, and H. Schmid-Schonbein. Microvascular network topology of the human retinal vessels. *Fortschr Ophthalmol*, 87:52–58, 1990.

[113] L. Schweigerer, G. Neufeld, J. Friedman, J. A. Abraham, J. C. Fiddes, and D. Gospodarowicz. Capillary endothelial cells express basic fibroblast growth factor, a mitogen that promotes their own growth. *Nature*, 325:257–259, 1987.

[114] T. W. Secomb, R. Hsu, M. W. Dewhirst, B. Klitzman, and J. F. Gross. Analysis of oxygen transport to tumor tissue by microvascular networks. *Int J Rad Oncol Biol Phys*, 25:481–489, 1993.

[115] E. M. Sevick and R. K. Jain. Geometric resistance to blood flow in solid tumors perfused ex vivo: Effects of tumor size and perfusion pressure. *Cancer Res*, 49:3506–3512, 1989.

[116] A. A. Shah-Yukich and A. C. Nelson. Characterization of solid tumor microvasculature: A three-dimensional analysis using the polymer casting technique. *Lab Invest*, 58:236–244, 1988.

[117] T. F. Sherman. On connecting large vessels to small. The meaning of Murray's law. *J Gen Physiol*, 78:431–453, 1981.

[118] T. F. Sherman, A. S. Popel, A. Koller, and P. C. Johnson. The cost of departure from optimal radii in microvascular networks. *J Theor Biol*, 136:245–265, 1989.

[119] Y. Shing, J. Folkman, C. Haudenschild, D. Lund, R. Crum, and M. Klagsburn. Angiogenesis is stimulated by a tumor-derived endothelial cell growth factor. *J Cell Biochem*, 29:275–287, 1985.

[120] A. C. Shore and J. E. Tooke. Microvascular function in human essential hypertension. *J Hypertens*, 12:717–728, 1994.

[121] D. Shweiki, A. Itin, D. Soffer, and E. Keshet. Vascular endothelial growth factor induced by hypoxia may mediate hypoxia-initiated angiogenesis. *Nature*, 359:843–845, 1992.

[122] Y.-J. Shyy, H.-J. Hsieh, S. Usami, and S. Chien. Fluid shear stress induces a biphasic response of human monocyte chemotactic protein 1 gene expression in vascular endothelium. *Proc Natl Acad Sci USA*, 91:4678–4682, 1994.

[123] T. C. Skalak and G. W. Schmid-Schonbein. The microvasculature in skeletal muscle. IV. A model of the capillary network. *Microvasc Res*, 32:333–347, 1986.

[124] S. A. Skinner, P. J. M. Tutton, and P. E. O'Brien. Microvascular architecture of experimental colon tumors in the rat. *Cancer Res*, 50:2411–2417, 1990.

[125] H. Song and K. Tyml. Evidence for sensing and integration of biological signals by the capillary network. *Am J Physiol*, 265:H1235–H1242, 1993.

[126] H. E. Stanley. Fractals and multifractals: The interplay of physics and geometry. In A. Bunde and S. Havlin, editors, *Fractals and Disordered Systems*, pages 1–49. Springer, Berlin, 1991.

[127] D. Stauffer and A. Aharony. *Introduction to Percolation Theory.* Taylor & Francis, London, 1992.

[128] T. Sun, P. Meakin, and T. Jøssang. Minimum energy dissipation and fractal structures of vascular systems. *Fractals*, 3:123–153, 1995.

[129] A. Timmer, J. W. Oosterhuis, H. S. Koops, D. T. Sleijfer, B. G. Szabo, and W. Timens. The tumor microenvironment: Possible role of integrins and the extracellular matrix in tumor biological behavior of intratubular germ cell neoplasia and testicular seminomas. *Am J Pathol*, 144:1035–1044, 1994.

[130] P. Vaupel, F. Kallinowski, and P. Okunieff. Blood flow, oxygen and nutrient supply, and metabolic microenvironment of human tumors: A review. *Cancer Res*, 49:6449–6465, 1989.

[131] T. Vicsek. *Fractal Growth Phenomena.* World Scientific, Singapore, second edition, 1992.

[132] E. M. Wahl, F. H. Daniels, E. F. Leonard, C. Levinthal, and S. Cortell. A graph theory model of the glomerular capillary network and its development. *Microvasc Res*, 27:96–109, 1984.

[133] T. A. Witten and L. M. Sander. Diffusion-limited aggregation, a kinetic critical phenomenon. *Phys Rev Lett*, 47:1400–1403, 1981.

[134] D. F. Zawicki, R. K. Jain, G. W. Schmid-Schoenbein, and S. Chien. Dynamics of neovascularization in normal tissue. *Microvasc Res*, 21:27–47, 1981.

[135] M. Ziche, L. Morbidelli, G. Alessandri, and P. M. Gullino. Angiogenesis can be stimulated in vivo by a change in GM3:GD3 ganglioside ratio. *Lab Invest*, 67:711–715, 1992.

[136] B. W. Zweifach and H. H. Lipowsky. Quantitative studies of microcirculatory structure and function. III. Microvascular hemodynamics of cat mesentery and rabbit omentum. *Circ Res*, 41:380–390, 1977.