

Nanosimulation of the Cytoskeleton and Related Self-Assembling Structures

Using Computational Techniques to Analyse and Model the Structure and Dynamics of
Microtubules, Actin strands, Viral assembly, Plasmodesmata, and Nanoscale Structures in
General.

This thesis is submitted in partial fulfilment of
the requirements for the Doctor of Philosophy,
Department of Computer Science
Monash University
January 2000

By Christopher Betts

Abstract

This thesis is primarily concerned with studying the phenomenon of self-assembly, particularly among proteins, at the nanoscale level. It analyses the problem of how to simulate nanoscale structures, and describes in detail a suite of computer programs that support generic nanoscale simulation. Using this program, the assembly of actin and tubulin is examined in detail, as well as the generation of abstract geometric structures and a simplified model of viral capsid proteins.

The simulation program is then used to extend a number of other computational analysis methods, and applied to the problem of determining the structure of plant cell plasmodesmata. This is done using a combination of 3-D modelling, image analysis, and using the nanoscale simulator to model the deposition of stain particles. The resultant model is then analysed using a new technique, dubbed 'virtual electron microscopy', which attempts to simulate the action of an electron microscope.

Acknowledgements

I would like to thank Dr Damian Conway, for supervising this thesis and providing timely assistance with the programming and proofreading. I would also like to thank Dr Rosemary White for co-supervising this thesis, and introducing me to cell biology.

I would also like to thank the many friends and colleagues who have helped in one way or another, especially Robert Rendell and Zik Saleeba for their technical assistance throughout the course of this thesis, Tim Bentley and Kylie Weaver for their understanding friendship, Rosie Rush for her sharp eyed proofreading, and most especially Janine Radford for her help with the biology, her contributions to the ideas developed herein, and her unstinting support.

Finally I would like to thank my parents for their many years of education and love; obviously without them this thesis would not have been possible.

Declaration

This thesis contains no material that has been accepted for the award of any other degree or diploma in any university or other institution.

To the best of my knowledge, this thesis contains no material previously published or written by another person, save where due reference is made in the text of the thesis.

Where the work in this thesis is based upon joint research, this thesis discloses the relative contributions of the respective authors.

.....

Christopher Betts

Contents

Chapter 1: Introduction	1
Global Context	1
Local Context	2
The Problem Domain	2
The Role of Computer Modelling and Analysis	4
Scope and Accuracy	5
Dynamic Modelling	5
Computer Analysis	6
The Structure of the Thesis	8
 Chapter 2: Introductory Cellular Biochemistry	 9
Introduction	9
Cellular Molecules	10
Amino Acids	10
Proteins	11
Enzymes	11
Antibodies	12
Carbohydrates	12
Membrane Lipids	12
Energy Sources: ATP and GTP	13
Large Cellular Structures	13
Membranes	13
Actin Filaments	14
Myosin	15
Microtubules	15
Intermediate Filaments	16
Plasmodesmata	16
Plasmodesmal Function	18
Cells	19
Viruses	19
Prokaryotic bacteria	19
Eukaryotic Cells	20

Chapter 3: Introductory Physical Chemistry	21
Introduction	21
Motion in Gases	21
Quantum Mechanics	22
The Boltzmann Distribution	24
The Equipartition Principle	24
The Boltzmann Distribution and Big Molecules	25
Brownian Motion	26
Liquids	27
Modelling Liquids	27
Modelling Molecular Dynamics of Liquids	28
Monte Carlo Methods	28
Modelling the Solute	29
Modelling Rotational Motion	29
Modelling Chemical Interactions	31
 Chapter 4: An Introduction to Computer Analysis	 32
Introduction	32
Image Processing	33
Image Smoothing	33
Image Sharpening	34
Histogram Equalisation	35
Localised Histograms	38
Fourier Transforms and Bandpass Filtering	39
Polar Fourier Transforms and Bandpass Filtering	40
Limitations of Image Processing	41
Modelling and Simulation	41
Analytic Modelling	42
Numerical Modelling	42
Static Models	43
Dynamic Models	43
Limitations of Modelling and Simulation	44
Note on examples	45

Chapter 5: Simulating Protein Self-Assembly	46
Introduction	46
Mathematical Models	47
The Oosawa Model	47
More Complex Mathematical Models	50
Rule Based Models	50
Computer Simulations	51
The Nanoscale Simulator	52
The Need for the Program	52
Applications of the Program	52
The Range and Application of the Nanoscale Simulator	54
Other Work in the Field	55
An Overview of the Nanoscale Simulator	57
Particle Movement: Diffusion	58
The Diffusion Equation in One Dimension	60
A Three-dimensional Solution	62
The Average Displacement of a Molecule	62
A true three-dimensional solution	64
Mean square distance of a spherical concentration	66
Accuracy of the Gaussian approximation	66
Summary of Particle Movement and Diffusion	67
Particle Motion: Determining Diffusion Constants	68
The Stokes-Einstein equation	69
Stokes' relation	69
Friction formulas for different shapes	70
Calculating these constants within the model	71
Diffusion: Some simple worked examples	72
2. A small protein	73
Summary	73
Particle Movement: Simulation	73
Rotation	74
The Rotational Approximation	74
Collisions	77
Binding, and Binding Sites	79

Breakages	80
Future Work	82
Events and State Changes	82
Boundary Conditions	83
The Scaling Problem	84
Summary	86
 Chapter 6: The Nanosimulator	 87
Introduction: The ‘Functional Requirements’ of the Program	87
Technical Requirements	87
User Requirements	88
Architecture of the Nanoscale Simulator	89
Program Objects: The Pieces	90
Physical Objects	90
Monomers	91
Monomer Helper Classes: Sites, Links, States, Spheres and Events	91
Polymers	93
A Note on Nomenclature	93
The Physical Environment	94
The 3-D Data Structure	94
Data Input and Program Initialisation	95
User Output	96
Mathematical Classes	98
The Co-ordinator Class: The Structure into which Everything fits	98
Initialisation: Setting Up the Game	99
Reading Data from the Command Line	99
Reading Data from the .pdf file	100
Initialising the Environment	101
Special Conditions and Work in Progress	102
Program Actions: The Moves	103
The Co-ordinator’s Cycle:	103
Moving Monomers and Dimers	103
‘Linear’ Motion	103
Approximating Rotational Motion	105

Boundary Conditions	107
Populating the Data Grid	108
Summary of Object Movement	109
Object Interaction	109
Collision Probabilities	110
Object Binding	111
Binding Free Objects	111
Binding a Free Monomer and a Bound Monomer	112
Merging Two Polymers	114
Unbinding	115
State Changes and Random Events	116
The Program Cycle: Playing the Game	117
A Special Mode: Working Out the Probabilities	119
Development Notes	120
Development Environment and Portability	120
Related Programs	121
The Collision Simulator	121
3-D Data Creation/Conversion Utility	122
3-D Data Model Viewer	123

Chapter 7: Modelling Simple Structures: Actin	124
Introduction	124
The Different States of Actin	124
The Structure of Actin Filaments	126
Actin Polymerisation and the Oosawa Model	126
Suitability of Actin for Simulation	127
Setting up the Simulation	128
Deriving Figures for the .pddf File	129
Modifying the Multiple Link Breakage Model	130
Handling Nucleation	131
Dephosphorylation	132
The Actin .pddf File	132
Running the Simulation	135
Appearance of the Simulation	136
Filament Growth	137
Polymerisation Curve	139
Filament Size Distribution	140
Conclusion	142

Chapter 8: Modelling Intermediate Structures: Tubulin	143
Introduction	143
The Different States of Tubulin	143
The Structure of Microtubules	145
Tubulin Polymerisation and the Oosawa Model	145
Suitability of Tubulin for Simulation	146
Simulator Limitations regarding Microtubules	146
Physical Structure of Growing Ends	147
Setting up the Simulation	148
Structure	149
Finite State Machine Definition	149
Deriving Figures for the .pdf File	150
The Multiple Link Breakage Model	152
Handling Nucleation	152
Simulating Different Models	153
Initial Conditions	153
Model A: An Initial Estimate	154
Results for Model A	160
Model A: Results	161
Model A: Conclusion	162
Model B: Decreasing the Breakage Probabilities	163
Model B: Results	163
Model B: Conclusion	165
Model C: Increasing Binding Probabilities	166
Model C: Results	166
Model C: Conclusion	168
Model D: Reducing Nucleation Probabilities	169
Model D: Results	169
Model D: Conclusion	171
Model E: Low Nucleation Probability, High Breakage Probability	171
Model E: Results and Conclusion	171
Model F: Low Breakage Probabilities with Side Bonds	172
Model F: Results & Conclusion:	173
Discussion and Conclusion	176

Chapter 9: Modelling Complex Structures and Viruses	178
Introduction: More Complex Protein Assemblies	178
Geometric Shapes: The Platonic Solids	179
The Dodecahedron	179
Dodecahedral Geometry	180
A Platonic Solid: Icosahedron	181
Icosahedral Geometry	181
2-D and 3-D Crystals	182
Simple Two-Dimensional Sheets	182
Triangular Sheet Geometry	183
A Three-Dimensional Lattice	184
3-D Tetrahedral Lattice Geometry	184
Simulating a Virus	186
Recent Work in Virus Assembly	187
An overview of Herpes Simplex A	189
Defining The Four Major Component Protein Types	191
Penton	192
Penton Geometry	192
P-Hexon	193
P-Hexon Geometry	193
E-Hexon	194
E-Hexon Geometry	194
C-Hexon	195
C-Hexon Geometry	195
Overall Structure	196
Some Assembly Required	196
Future Directions	198
Conclusion	199

Chapter 10: Combined Modelling Techniques	200
Introduction: Integrating with other Modelling Methods	200
Electron Micrographs and Image Processing	201
Actin and Tubulin	201
Plasmodesmata	202
Image Processing	203
Using Micrographs and Processed Images	206
3-D Models	206
A Note on Static 3-D Models	207
Models of Actin and Tubulin	208
Proposed Models of Plasmodesmata	209
The Textbook Model	209
The Overall Model	210
The Ding Model	210
The Thomson Model	210
The White Model	211
The Radford Model	211
3-D-Models of Plasmodesmata	211
3-D Computer Models of Plasmodesmata; Real Time Views	212
3-D Models of Plasmodesmata; Raytraced Views	213
Combining Models with the Nanosimulator	214
Introduction	214
Stain Deposition	215
Simulating Stain Deposition	216
Converting the 3-D Model	216
Simulating Stain Movement and Binding	217
Examples of the Deposition Process	218
Modelling Antibody Stain	218
The Final Step: Virtual Microscopy	220
Introduction	220
‘Washing’ the Virtual Sample	221
Use of a Ray-Tracer	221
Preliminary Work	222
lack of perfect focus	222

uneven resin, impurities and other causes of random brightness	
variation	222
magnetic ‘lens’ defects such as skew astigmatism	222
Photographic film graininess	223
Actin Filaments	224
Microtubules	224
Using the Nanosimulator Results in the Virtual TEM	225
Resultant Images	225
Virtually Stained Microtubules	226
Studies of Plasmodesmata	228
Virtual Microscopy without Staining	230
Discussion	234
Conclusion	235
 Chapter 11: Accuracy and Validity of the Nanoscale Simulator	236
Introduction	236
Setting the Scene: Other Simulation Programs	236
Accuracy and Predictive Power	239
Validity of the Model	240
Summation	242
 Chapter 12: Conclusion	243
Introduction	243
Reviewing Theory	243
Implementing Theory	244
Testing the Program	245
Experimenting with Tubulin	245
Extending the Range of the Simulator	245
Combining the Simulator with other Modelling Techniques	246
Discussion of Limitations	246
Summary	247
Future Directions	248
Modelling Further Biological and Non-Biological Systems	248
Modelling Drugs	248

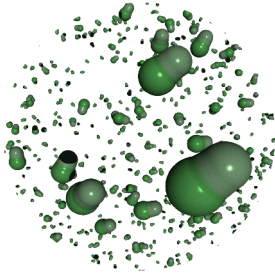
Modelling Conformational Changes	249
Modelling Motion within an Aggregate	249
More Detailed models of Binding	250
Improving Program Features	250
More Detailed Tracking of Individual Aggregates	251
Nanotechnology	251
Pedagogical Uses	252
Modelling of (Speculative) Biological Information Processing	252
The Future	252

Appendix A: Calculating Collision Probabilities	A.1
Introduction	A.2
Defining the Problem	A.3
Analytical solutions	A.4
A numerical solution	A.5
How these figures are used	A.8
Raw Figures	A.9
Conclusion	A.12
 Appendix B: Nanosimulator Operation	 B.1
Introduction	B.1
Program Invocation	B.1
Command Line Parameters	B.2
Parameters Affecting Graphical Display	B.2
Parameters Affecting Logging and Reporting	B.3
Parameters Affecting the Simulation Environment	B.3
General Parameters	B.5
Graphics Display Window	B.6
 Appendix C: Protein Dynamic Description Files	 C.1
Introduction	C.1
.pddf File Syntax	C.2
Basic Data Types: floats, strings, points and colours	C.2
Floats and Strings	C.2
Points	C.2
Colours	C.3
File Structure	C.3
Model Definition	C.4
Geometrical Structure Definition	C.5
Linkage Site Definition	C.6
State Definition	C.7
Event Definition	C.8
Linkage Definitions	C.9
An Example .pddf File	C.10

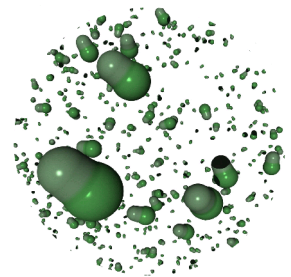
The Future?	C.13
Appendix D: Platonic Solids: .pddf files	D.1
Cube, Dodecahedron and Icosahedron	D.1
Cube	D.1
Dodecahedron	D.4
Icosahedron	D.9
Appendix E: Tetrahedral Lattice .pddf file	E.1
Appendix F: Herpes Simplex A .pddf file	F.1
Appendix G: Povray Microtubule model file	G.1
References	Ref. 1

Chapter 1

Introduction



*To see a world in a grain of sand
And a heaven in a wild flower
Hold infinity in the palm of your hand
And eternity in an hour.*



Auguries of Innocence, William Blake

Global Context

In the last fifty years electronic computers have come from being a military curiosity to a ubiquitous tool. In the same way as the lever and the wheel once extended the physical power of human beings, computers have been able to extend our mental powers, allowing us to process huge quantities of information in hitherto undreamt of ways.

For the researcher, it is becoming possible to examine the world using entirely new tools, making sense of large quantities of data where before we would have been overwhelmed by the task. It is also becoming possible for us to ask complex "what if?" questions about the world, and answer them with something more than pure supposition and imagination. Perhaps the most amazing thing about computers is not that they perform calculations at blinding speeds, or that they can communicate with each other at an incredible rate, but that they allow us to visualize, model and, in a way, interact with environments that would never otherwise be possible.

Through computer modelling we can create virtual environments that allow us to examine the entirety of creation, from galaxies and nebulae to atoms and molecules. We can even attempt to model the processes that occur within a living cell.

Local Context

In the sciences, two of the most valuable things that computers can be used for are improving the analysis of data, and simulating experiments and conditions that are either impossible, impractical, or too expensive to perform physically. Often computers make procedures practical that would otherwise have required large teams of people with slide rules or complex equipment, or might have been completely impossible.

However, one of the current problems with computers is that it is difficult for non-experts to use computers as effectively as possible, especially in specialized fields where pre-written software may not be available. This has led to a number of collaborations, where a specialist in one field works with a computer-literate non-specialist, in an attempt to apply computers to a particular field.

This thesis represents such a collaboration, between plant cell physiologists and computer scientists, initially to examine the transport of material through the tunnels that join plant cells. However, this led into the wider area of transport in general, and from there into a much broader field altogether.

The Problem Domain

The cytoskeleton (literally, ‘skeleton of the cell’) is a complex, dynamic set of structural components within a cell that give the cell shape, and act as an internal ‘railroad’ for the internal movement of cell components. Unfortunately the cytoskeleton is difficult to observe in action. The filaments (Fig. 1.1¹) of the cytoskeleton are too small to observe in detail by light microscopy (although much interesting work has been done on the aggregated filaments using fluorescence

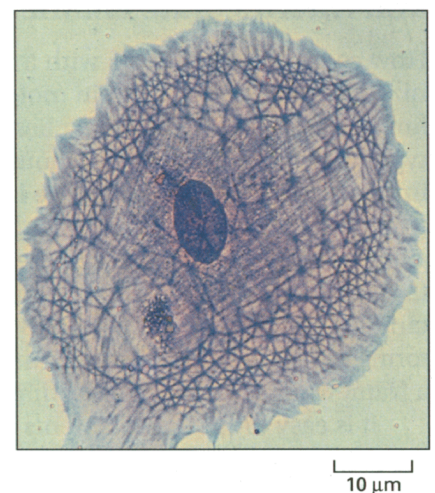


Figure 1.1: The Cytoskeleton in a single cell.

microscopy). Nor can they be observed in detail with an electron microscope without killing the cell. Another complication is that this system of filaments is dynamic, and is continuously being destroyed and recreated within the living cell.

While the static structure of these cytoskeletal components is quite well known, many elements involved in their dynamic operation are poorly understood. Actin filaments and microtubules (cf Fig. 1.2), which are two of the main components of the cytoskeleton, are usually found in a cyclic state of growth and decay, changing their lengths rapidly depending on various environmental conditions that are not fully understood. They also act in concert with movement proteins such as myosin and dynein, which in turn carry or move other proteins around the cell. These processes are not well understood either.

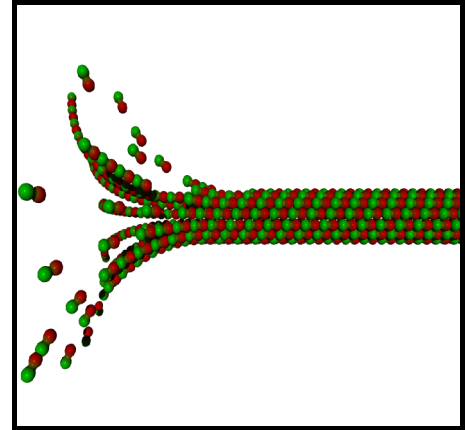


Figure 1.2: A model of a disassembling microtubule end

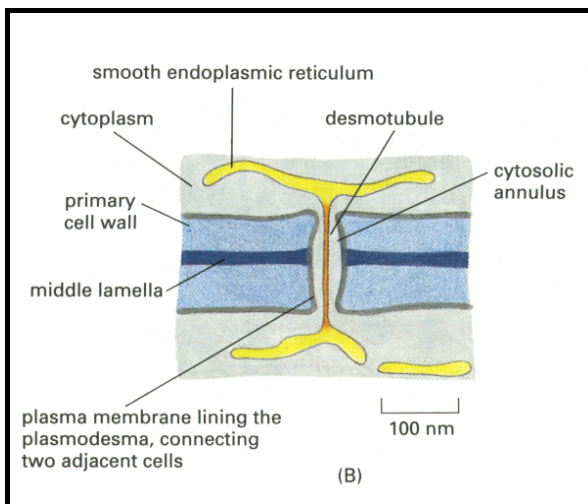


Figure 1.3: A simple plasmodesmata model

The cytoskeleton is responsible for movement within both animal and plant cells. Movement between cells is mediated by a number of different structures, the largest of which are *gap junctions* in animal cells, and *plasmodesmata* in plant cells. Animal cells (in general) are surrounded by a thin, flexible membrane, bridged by the gap junction. Plant cells however have a thick cellulose cell wall, and the plasmodesmata that pass through them take the form of long tunnels (see Fig. 1.3²).

However, like cytoskeletal filaments however, plasmodesmata are too small to observe using light microscopy. Their internal structure is also very difficult to ascertain using current electron micrographic techniques. Plasmodesmata are so small, and their internal structures so tightly integrated into their surrounding membranes, that they are difficult to separate from the

cell wall in which they are embedded, making it hard to prepare specimens for high power electron microscopy. Being embedded in the cell wall makes it difficult to get a clean cross-sectional cut, and also impairs isolation of their compositional proteins. In addition, as cellular gateways they are very sensitive to injury, and will react to close themselves if they sense any injury, as frequently happens during specimen preparation³.

Direct protein analysis is problematic, since it is hard to prepare a large enough quantity of the plasmodesmal proteins independent of the surrounding cell wall proteins, although some attempts have been made^{4,5}. One approach that has been successful is using antibodies to known proteins and examining whether they bind to active sites on exposed proteins in plasmodesmata⁶. Using antibodies, it has been shown that two cytoskeletal movement proteins, actin⁷ and myosin⁸, exist in plasmodesmata.

The study of these structures can be aided by appropriate use of computer simulation techniques. This thesis studies in detail one particular set of techniques, that of modelling the dynamic behaviour of the constituent proteins. In this way a deeper understanding of the way in which dynamic structures such as the cytoskeleton are built up and decay, and the way in which material passes through and interacts with (comparatively) large structures such as the plasmodesmata, can be obtained. To a lesser extent, combining such dynamic modelling with other computational techniques such as 3-D modelling and image processing is also explored.

The Role of Computer Modelling and Analysis

Many of the difficulties faced in examining the cytoskeleton and plasmodesmata can be reduced by using computer analysis in conjunction with experimental techniques. Using computers it is possible to extract more information out of existing electron micrographs. Computers can also be used to model our theories of cytoskeletal behaviour, and of plasmodesmal structure. Even more exciting is the possibility of using computer modelling to examine the dynamics of cytoskeletal and plasmodesmal function.

Scope and Accuracy

To build an accurate model all relevant data must be taken into account. This may be a daunting task, since the amount of possibly applicable data may be huge, so all irrelevant data must simultaneously be excluded. As Albert Einstein said, "Theories should be made as simple as possible, but no simpler".

Deciding what is relevant and what is not is only the first of many difficulties confronting the modeller of complex systems. Often it will be found that it is simply not possible to fully model all factors, and that simplifications and abstractions are necessary to render the problem tractable. This leads to inaccuracies in the model, but may be necessary for the model to be useful. If the inaccuracies become too great, it may be necessary to admit that the system is too complex to model using current techniques and a solution may have to wait on new theoretical and experimental insights, or better computer hardware.

The difficulty with any model is that it is an hypothesis, a theory that the important features of a system can be summarised in a certain way. Thus Newtonian physics, and its derived results, provide an excellent model of, for example, planetary motion. Such model enable the simplification of a vast mass of experimental data into a few simple rules.

Following Karl Popper's observations on theories in general, we can claim that the power of a particular computer model is dependent on the breadth of material it claims to simplify, the strength of its predictions, and the ease with which it makes testable assertions⁹. While the computer models developed in this thesis may be interesting, stimulate discussion, and provide useful tools for thinking about nanoscale objects, their ultimate test will be whether they accurately reflect real-world experiment, and can predict the results of new experiments.

Dynamic Modelling

The majority of this thesis is concerned with modelling dynamic systems, in order to gain insights into complexities of behaviour not easily modelled with analytic theories. This

modelling allows the investigation of extremely complex systems, with large numbers of independent proteins interacting.

In order to establish a baseline, the self assembly of the protein actin (Fig. 1.4), which is well characterised both experimentally and theoretically, is considered in chapter 7, where it is shown how the simulator allows very detailed observation of the assembly process. Continuing on from this work, the self-assembly of the other main cytoskeletal protein, tubulin (which is not so well understood) is covered in Chapter 8.

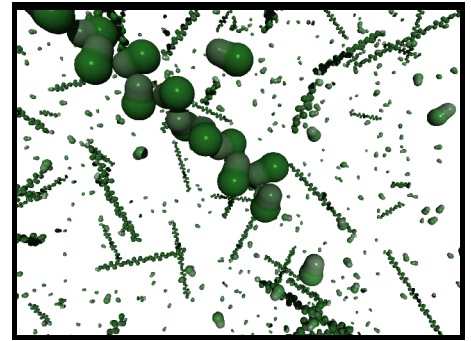


Figure 1.4: A snapshot taken during a simulation of actin self-assembly

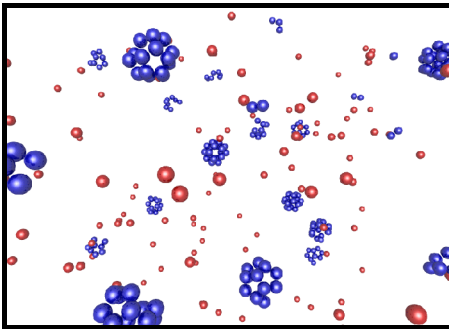


Figure 1.5: Self-assembly of a simple, abstract virus model.

The simulation engine is then turned to a number of other systems, such as the self-assembly of protein crystals and viruses (Fig. 1.5). Some results for these interesting systems are presented in Chapter 9.

Computer Analysis

There are many other contributions that computers can make to biological analysis. For example, there are a large number of other computational techniques for simplifying data, and for extracting pure data from background noise. One method of particular interest is that of image processing, where the various physical defects or shortcomings of an image can be removed or reduced by computer, revealing details that were not originally fully evident by enhancing otherwise hidden structure. A prime example considered in this thesis is that of electron micrographs, which tend to be indistinct at the level of the cytoskeletal and plasmodesmal structures considered.

The construction of static models can also be an important part of determining structure. Using the information available, it may be possible to clarify the arrangement of components and see what is physically possible, by trying to fit the various pieces together in different ways. Traditionally such models have been either purely mental, sketched (usually on napkins in bars) or made from coloured plasticine and toothpicks.

While these are all fine modelling methods, there are some advantages to using computer modelling as a final stage, and it has become quite popular in biological research. When it is possible to accurately specify the sizes of component proteins, it is possible to work out details such as the size of the largest possible transfer molecule that can pass through a plasmodesma (the theoretical “size exclusion limit”). It is possible to readily examine the model (e.g. Fig. 1.6) from many angles, to see whether it corresponds to observed real-world objects, and it becomes easier to see whether hypothesised proteins have sufficient room to exist, and if so, what configurations are possible.

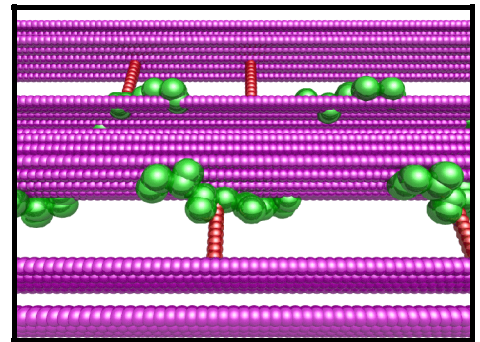


Figure 1.6: A 3-D Model of a plasmodesmata cross-section showing membranes (purple), actin strands (green) and hypothesised structural proteins (red)

As an extension to traditional modelling, this thesis also investigates the possibility of verifying a model by constructing the electron micrograph that such a structure might produce. This “virtual micrograph” allows us to compare the model with actual images of real cellular structures, and thus acts as a validity test.

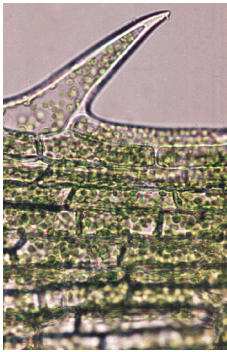
Results combining the power of the dynamic simulation engine with the above methods of 3-D modelling and ‘virtual microscopy’ are presented in Chapter 10, along with details of an attempt to improve the ‘virtual microscopy’ technique by modelling details of the staining and preparation process.

The Structure of the Thesis

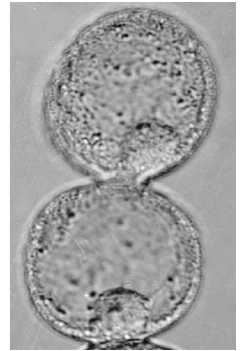
Since this thesis deals with an interdisciplinary topic, an attempt has been made to make it accessible to a wide range of readers. In addition to the main body of the thesis, there are three introductory chapters that attempt to provide a basic understanding of the cell biology, physical chemistry, and computational techniques involved in this work. While the expert may find them superficial or redundant, it is hoped they will provide a minimal background for readers unfamiliar with any or all of these fields, enabling them to follow the work described in the remaining chapters. Readers who are familiar with all three fields may wish to skip these chapters and proceed directly to chapter 5, which is the first to describe work specific to this thesis.

Chapter 2

Introductory Cellular Biochemistry



Felix qui potuit rerum cognoscere causas.
(Happy the man who was able to discover the causes of things.)
- Vergil



Introduction¹⁰

This chapter presents a very brief guide to the basics of cell function. Readers should be aware that many of the generalised statements presented here are by necessity simplifications and abstractions. Readers interested in a more rigorous treatment are directed to some of the excellent introductory texts available*.

A summary of biological molecules is given first. These are the building blocks from which the larger cell structures are built, the most important of which are proteins, the ubiquitous ‘workhorse’ molecule of the cell. Next a small number of larger cell structures are described. These usually consist of different proteins, and the self-assembly of these is described in later chapters. Finally a brief overview of living cells is given, in order to set the context in which the self-assembling structures grow, and to describe the variety of organisms in which they are found.

* Especially recommended are:

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson J.D., *Molecular Biology of the cell* (3rd ed.), Garland Publishing, New York,

Lodish, H., Berk, A., Zipursky, S.L., and Matsudaira, P., *Molecular Cell Biology* (4th ed.)

W.H. Freeman & Co., New York, and (slightly more technical)

Voet, D. and Voet, J., *Biochemistry*, John Wiley & Sons, New York

Cellular Molecules

There are two main types of large molecules, or polymers, within the cell. The first is **proteins**; which are the workhorses of the cell, the machinery that provides most of the cell's functionality, and makes up most of the structures of the cell. The other type is **nucleic acids** (such as DNA), which are the data-banks of the cell, containing all the information necessary to build proteins, and to store and pass this information around. This thesis will concern itself mainly with proteins.

There are a variety of types of smaller molecules, or monomers, within the cell. Proteins are almost entirely built from a set of twenty monomers called **amino acids**. Nucleic acids are made up from a different set of five monomers called **nucleotides**. The two other main types of small molecules are **saccharides** (sugars) and **fatty acids**. Saccharides sometimes polymerise (link together) to make another type of large molecule, the **carbohydrates** which among other things make up the bulk of the cell walls of plants. (Plant cells, unlike animal cells, have large, thick shells called the "cell wall".)

Amino Acids

If proteins are thought of as small machines, amino acids are the nuts and bolts. Amino acids are very small, around ten to thirty atoms. They all have a very similar "head", a carboxyl group, and all save one also have an amino group. The amino groups on each amino acid can easily bind (polymerise) with others, making them easy to string together to form proteins.

A short chain of amino acids is called a **peptide**, and the chemical covalent bond that joins two amino acids is called a **peptide bond**. A long string of peptides such as a whole protein or large protein fragment is called a **polypeptide**.

Proteins

A protein is more than just a long strip of amino acids however. Most of its interesting properties arise from the overall shape of the entire protein structure. A strip of unrestrained amino acids will coil and fold, since the links between different amino acids are extremely flexible. As it folds, chemical cross links are formed between different parts of the amino acid chain, and in this way it forms its final, stable shape. The process of folding is actually quite complex, and not yet fully understood. In some cases (especially for larger Proteins) other proteins 'help' the Protein to fold to the correct shape.

The unwrapped state, where the protein is an unstructured strip, is known as the **denatured** form of the protein, which then **renatures** to the active form, or **native state**.

Enzymes

Enzymes are special proteins that encourage, or **catalyse**, specific chemical reactions. The reactions would take place anyway, although in biological terms these reactions might occur so slowly that the organism could have died first. Enzymes make the reactions happen much more quickly. They do this by having **binding sites** that grab the constituent chemicals to be reacted (also known as **substrates**). They then use **active sites** to bring these substrates together to form the final product. Sometimes the enzyme actually changes its shape, or **conformation**, in order to do this, and sometimes an enzyme needs a helper enzyme called a **co-enzyme** in order to work.

Most enzymes are named after the name of their product, with the suffix "-ase" added. Hence ribonuclease is an enzyme that acts on ribonucleic acid (RNA), while a protease is something which catalyses the breakdown of proteins in general, and so forth.

Antibodies

Antibodies are small proteins that have a **binding site** (similar to the binding site on an enzyme) that is designed to link with one particular molecule, such as a specific protein on the outside of a virus. They are very precise, and act as marker flags for the body's immune system; they designate something as being undesirable, for recognition and elimination by white blood cells.

Antibodies are very precise targeting agents, and may entirely ignore a protein with an amino-acid sequence that differs from the target protein by only a single amino acid.

Carbohydrates

Carbohydrates are sugars, and are used by the cell to store energy. Large, or **complex** carbohydrates are also used as structural components. **Cellulose** is one such carbohydrate, which is especially common in the walls of plant cells.

Membrane Lipids

A lipid is a small molecule that is **hydrophobic** ("water hating") at one end, and occasionally **hydrophilic** ("water loving") at the other. Pure **hydrocarbons** (e.g. petrol) are completely hydrophobic, and are insoluble in water, tending to form small isolated drops of oil. Most lipids are like this, but there is a subspecies that lives entirely in membranes, called **membrane lipids**, and these have both a hydrophilic end and a hydrophobic end, making them **amphipathic**. As a result, groups of amphipathic lipids tend to orient themselves with their hydrophilic ends placed in the surrounding water, and their hydrophobic ends either embedded in oil, or facing the hydrophobic ends of another group of lipids.

Amphipathic lipids can form sheets. For instance, a planar sheet of lipids might divide a watery volume from an oily volume, having both a hydrophobic surface and a hydrophilic surface. An arrangement that is very common in biology is two such sheets, back to back, called a **lipid bilayer**. In a lipid bilayer the outside is hydrophilic and the interior is hydrophobic. This is a very stable structure, and lipids of this sort form many of the membranes within cells.

Energy Sources: ATP and GTP

Many processes in the cell require energy to operate. The action of enzymes, the movement of proteins, or the synthesis of DNA and RNA, all require an energy input. The most common method for energy to be delivered is through small molecules such as ATP (adenosine 5'-triphosphate) and GTP (guanosine 5'-triphosphate). The molecules chemically react with the protein they are providing energy to, losing a phosphate ion to become the diphosphate form, ADP or GDP respectively. Many of the experiments and simulations described later in this thesis use large amounts of ATP or GTP to keep reactions energised.

Large Cellular Structures

Many of the molecules mentioned above combine to form the large structures found in cells, some of which are visible using a light microscope. Modelling the construction, or **self assembly**, of some of these structures forms a large part of this thesis.

Membranes

These membrane-forming lipid bilayers form the cellular sacs that enclose most large cellular components. Cellular components, or **organelles**, are held together in bags made out of lipid bilayer membranes, and most cells also have a lipid membrane as at least the first layer of the cell boundary. Plant cells, for example, have a lipid bilayer enclosing the cellular liquid, inside the much larger, and leaky, cellulose wall that bounds the cell.

Many proteins are found in or on membranes. Proteins in membranes can usually move around within the membrane fairly freely. Some proteins form small gated pores, which regulate the movement of things from one side of the membrane to the other. Others act to control the mobility of proteins in the membrane, or perform any of the other functions of proteins such as chemical synthesis and transport.

Actin Filaments

Actin strands are long polymers of the protein actin, often found bundled together in thick strands, webs or gels. They are found in cross-linked networks lying just under the **plasma membrane** (the outer membrane of the cell), and form the **cell cortex**.

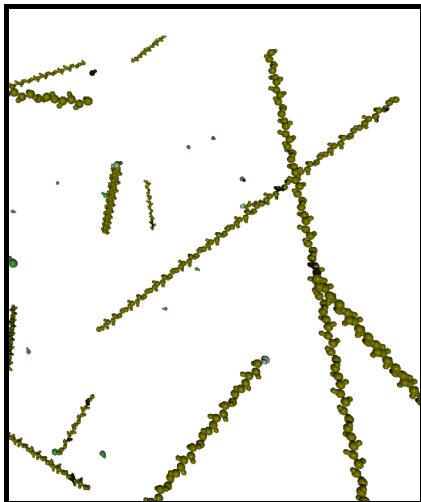


Figure 2.1: Model of Filamentous Actin.

Actin is notable as a self-assembling protein. A solution of actin monomers in a test tube will, under the right conditions, spontaneously form long actin filaments, which may later break down into single monomers (Fig. 2.1). If energy is provided in the form of free ATP molecules, the cycle may repeat indefinitely.

Actin filaments have a definite directionality, a 'plus' end and a 'minus' end. This directionality can be used by the cell to regulate transport, as the **motor proteins** that travel along the actin filaments always travel in a specified direction.

Actin is very important in cell movement, and in cell division. A polymerising actin strand can push things along as it extends, such as the pseudo-pod of an amoeba. More usually though, actin serves as a pathway for motor proteins to move along.

Myosin

Myosin is an important motor protein which, in conjunction with actin, is responsible for much internal cell movement. In animal cells, the actin/myosin complex is responsible for muscular movement, and in plant cells actin and myosin are responsible for **cytoplasmic streaming** - the stirring of the cell's liquid interior, the **cytoplasm**, which ensures the mixing of free-floating cell chemicals.

Microtubules

The protein tubulin forms long, cylindrical structures known as microtubules¹¹ (Fig. 2.2, 2.3). A tubulin monomer is a small, roughly spherical (or globular) protein that comes in three main types, alpha-tubulin, beta-tubulin and gamma-tubulin. Alpha and beta tubulin link together to make very stable tubulin **dimers**, which in turn combine to form the helically wound cylinder. Gamma tubulin is much rarer, and is hypothesised to be a stable form of tubulin used by the cell to create starting sites for microtubules to grow from¹².

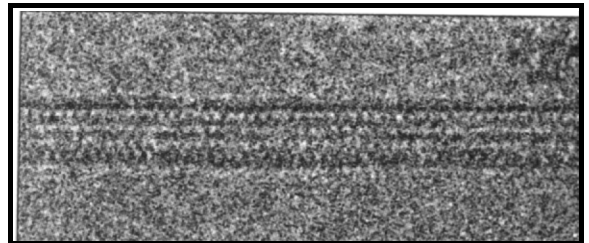


Figure 2.2: Electron micrograph of a microtubule.

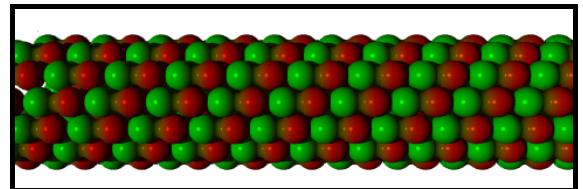


Figure 2.3: Model of a microtubule.

Microtubules grow faster at one end than at the other, and are not usually static. Normally they are either in the process of growing, or the process of shrinking, or growing from one end and shrinking at the other (a feature shared by actin filaments). Associated proteins, known simply as microtubule associated proteins, or MAPs, stabilise the microtubules, and link them to other proteins, to bits of the cell, or to each other.

Microtubules, like actin filaments, are used to transport large proteins and even organelles around. Like actin they have a strong directionality, with a 'plus' end and a 'minus' end. In a

cell, many microtubule 'minus' ends meet at a sort of cellular central railway station called a **microtubule organising centre**, or **centrosome**. This is usually near the nucleus, with the microtubules streaming away from it toward the cell wall.

Two important microtubule-associated proteins are the motor proteins **dynein** and **kinesin**. Dynein links microtubules with other components and moves them towards the minus end, which is often located at a centrosome. The motor protein **kinesin** moves them away from the minus end (and thus possibly away from a centrosome, towards the periphery of the cell).

Like actin, tubulin self-assembles from free-floating molecules, and this can be reproduced in a test tube. The structure of a microtubule is also more complex than that of an actin filament, being a number of helices, somewhat like a multi-strand rope. Probably the most common structure is the "3-start, 13-protofilament spiral", which can be thought of as thirteen rows of tubulin monomers joined to make a tube, or equivalently three strands of monomers spiralling to make the same tube (cf Fig. 2.3).

Intermediate Filaments

Intermediate filaments are the third main component of the cytoskeleton. They are important structural components and are less dynamic than actin filaments and microtubules. They seem to mainly form structural supports, midway in size between actin and tubulin.

Self-Assembly and Nucleation

A number of the structures mentioned (especially actin filaments and microtubules) spontaneously *self-assemble* from their component proteins. An important rate-limiting feature of self-assembly is the need for the structure to reach a minimum size, sometimes called the *critical-nucleus*, below which it is unstable and quite likely to fall apart, rather than to continue to grow. This process in which a small number of component proteins aggregate to form a critical nucleus is called *nucleation*, although the term is sometimes applied generically to the rate-limiting step in an assembly process.

Plasmodesmata

Multicellular organisms generally exchange some material between cells, especially in differentiated organisms where distinct parts of the organism, (such as roots and leaves in plants) perform different functions, producing and consuming different material. This material may be food or it may be more complex proteins and hormones. Animal cells have thin cell membranes, and exchange material via small structures called **gap junctions** which, combined with an elaborate vascular and nervous system, allow

material to pass between cells in a controlled manner. Smaller structures called **ion channels** regulate the flow of ions (very small electrically charged molecules or atoms).

Plant cells have a comparatively thick cellulose cell wall. Although this wall is porous to small molecules, larger molecules require a tunnel of some sort to pass through. Plant cells have such tunnels, called "plasmodesmata" (Fig. 2.4¹³), but their structure is not well understood, and the examination of this structure is considered in later chapters as an opportunity to apply some of the techniques developed in this thesis.

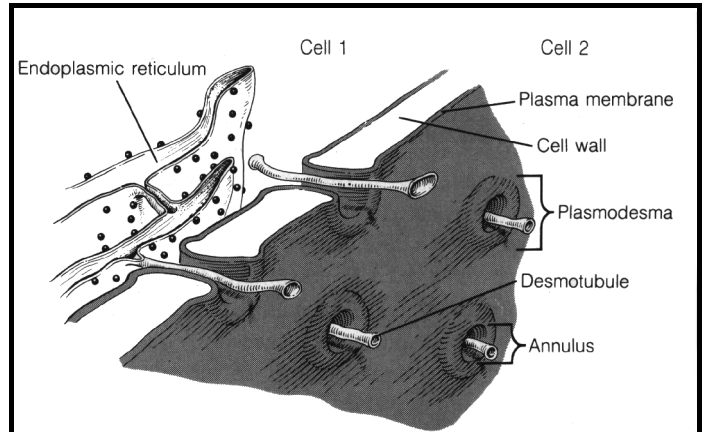


Figure 2.4: Simplified model of plant plasmodesma with the endoplasmic reticulum.

Although there is some variation in plasmodesmal structure (especially among the "simple" or "lower" plants), generally plasmodesmata have a number of common features. The plasma membrane of a cell extends into the plasmodesmata, forming a protective wall as the outside of the tunnel, separating the interior of the plasmodesmata from the interior of the cell wall. In the very centre of most plasmodesmata is a thin tube of shrunk endoplasmic reticulum (ER), usually called the **desmotubule**. Between this central pillar and outside wall are a variety of largely unknown proteins¹⁴.

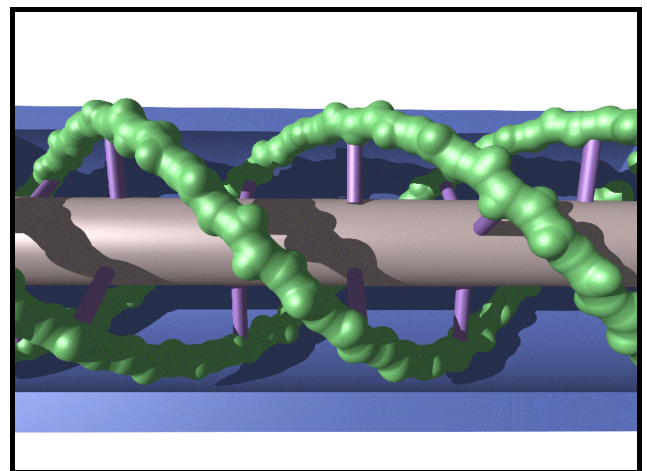


Figure 2.5: A cross-section of a model of a hypothetical plasmodesmal structure, showing outer spirals of actin attached to proteinaceous struts.

In older plasmodesmata there are often enlargements, where the plasma membrane bulges out into the cell wall space. Sometimes the E.R. of the desmotubule also expands within these cell wall 'bubbles', and sometimes plasmodesmata are observed that meet within the cell wall (although the exact method of formation for these joined or branched plasmodesmata is not fully understood)¹⁵.

There is still considerable debate about the exact nature of these internal components, and a number of different structural models are considered in Chapter 10. One such model is shown in figure 2.5, which shows the desmotubule as the central gray cylinder, spiral actin strands as green, the external plasma membrane as blue, and hypothesised 'strut' proteins as purple (this particular model is also the basis for the cross-section shown in Fig. 1.6).

While the proteins between the desmotubule and the plasma membrane external wall are not completely known, some recent observations suggest the presence of the cytoskeletal proteins actin¹⁶ and myosin¹⁷ within plasmodesmata, which may imply some continuity between the transport role of the cytoskeleton within the cell, and that of plasmodesmata between cells. Alternatively, this may have more to do with the creation of plasmodesmata, which are formed during cell division. During cell division there are large amounts of cytoskeletal filaments present, which might be caught within forming plasmodesmata. This may even be the way plasmodesmata are formed, by filaments being caught within the growing cell plate that eventually forms a cell wall separating the two sides of the dividing cell¹⁸.

Plasmodesmal Function

The plasmodesmata transport material between cells. They also have a role in controlling this flow. In extreme cases, such as cell injury, they can close off completely, while there is also evidence to suggest that they partially open and close as part of normal cell operation¹⁹.

In addition to normal cell material being transported, plasmodesmata can also act as a conduit for plant viruses. Some such viruses 'bore' their way through the plasmodesmata, destroying the desmotubule as they go^{20,21} but others seem to be able to pass their genetic material (the RNA or DNA) through without damaging the plasmodesmata²². Since these molecules are

usually much too large to pass through the plasmodesmata, which generally admits only medium-sized molecules, it is thought that the usually highly folded genetic strands must straighten out, and somehow pass through the plasmodesmata like a piece of string through a narrow tube²³.

Cells

Living cells are made up of complex combinations of the molecules and structures described here. In increasing order of complexity are viruses (which arguably are not cells at all, since they cannot survive on their own), **prokaryotic** cells such as bacteria (which have no separate nucleus), and **eukaryotic** cells (which do have a defined nucleus).

Viruses

The simplest type of organism is the virus. It is a matter of definitions as to whether it is even alive, or simply a complex protein. At its simplest, it is just a short strand of genetic material (DNA or RNA) enclosed in a shell of protein **capsids**. If this package finds its way into an appropriate host (a bacterium or a cell) the enclosing shell is broken down, releasing the genetic material, which uses the infected cell to create more capsids and more genetic material, which then in turn self-assembles into more viruses.

Prokaryotic bacteria

More complex are **prokaryotic** bacteria, simple cells that do not have a separate nucleus. These bacteria developed before nucleated cells, and their genetic material floats inside the same membrane enclosed space as the other cellular organelles. They have been found in rocks over 3 billion years old, which makes them the oldest form of true life. Prokaryotic bacteria are usually simply a loop of DNA in a minimal packet of cell cytoplasm.

Some prokaryotic cells are capable of forming very simple multicellular creatures, such as blue-green algae colonies, but more complex multicelled creatures are all made up from the **eukaryotes**, the nucleated cells.

Eukaryotic Cells

These are what we normally think of as a cell. Animal and plant cells are always eukaryotic, meaning that they have a separate membrane-enclosed nucleus containing their DNA. This allows them to regulate the building of proteins more accurately, which in turn allows for far more complex behaviour.

Cells, especially eukaryotic cells, are complex and variant systems. All eukaryotic cells tend to share some common features:

- *Nucleus*: This is a membrane-enclosed region containing genetic material.
- *Nuclear membrane*: a double membrane separating the nucleus from the main body of the cell, which regulates transport via **nuclear pores**.
- *Cytoplasm*: cell material outside the nucleus; liquid, proteins, and other chemicals. In plant cells this is in continual motion due to **cytoplasmic streaming**, a whirlpool-like effect caused by the cytoskeleton.
- *Cytoskeleton*: the network of **actin** and **tubulin** filaments that forms the internal structure of the cell, and provides intra-cellular transport.
- *Endoplasmic reticulum*: a large, intricate membrane within the cytoplasm. Its convolutions pack a large surface area into a small volume, within which certain special operations can be carried out, mainly to do with the manufacture of proteins.
- *Golgi bodies*: small nodes often close to the endoplasmic reticulum, used in protein modification and carbohydrate synthesis.
- *Organelles*: specialised subunits of a cell such as **chloroplasts** (used by plant cells in photosynthesis) and **mitochondria** which produce the chemical ATP, the main energy storage molecule of the cell.

$$D = \frac{\lambda^2}{2\tau}$$

Chapter 3 Introductory Physical Chemistry

$$c_x = \frac{k e^{-x^2/4Dt}}{\sqrt{(Dt)}}$$

Since they travel about through void, atoms must move on either by their own weight or randomly by the stroke of another. For when during motion they have, as often happens, met and clashed, the result is a sudden rebounding in an opposite direction...

- Lucretius, "On the Nature of Things" circa 50 BC

Introduction^{24,25,26}

In order to simulate the self-assembly of organic molecules, it will be necessary to model their motion within liquid. Modelling the motion of molecules is, however, an involved process. This chapter conducts a broad overview of physical chemistry, and how it applies to the modelling of liquids, in preparation for Chapter 5 which explicitly covers the model used in this computer simulation.

As the first step in understanding liquids it is necessary to examine the behaviour of gasses, and derive various laws which can then be applied to the liquid case.

Motion in Gases

The motion of molecules in gases has been studied in great detail, and the theory of particle motion, which can be used to explain the temperature, pressure and density characteristics of a gas, was one of the great triumphs of nineteenth century physical chemistry.

The "Ideal Gas" can be considered to be made up of molecules moving in straight lines, that only interact when they collide. From this simple model, a great many general features of gases can be derived, such as pressure and temperature, and even more complex attributes such as the rate of diffusion.

Unfortunately, for many real gases, this simple model breaks down. It fails in extreme conditions (i.e. high densities and pressures) when the gas starts to display liquid properties, and the assumption the theory makes - that molecules interact only at the instant of collision - fails. More importantly though, it is inaccurate for all but the simplest molecules, even under standard conditions, due the energy of the molecule not being confined to the kinetic energy of linear motion. Instead, the collision energy is absorbed by the molecule in a number of different ways, being spread between kinetic energy, rotational energy and vibrational energy. In extreme conditions it is also absorbed in electronic transitions within the component atoms of the molecule.

Quantum Mechanics

A further complication is the quantum nature of the inter-molecular interactions. Quantum theory and experimental observations show that certain energy-related molecular attributes can adopt only discrete, rather than continuous values. The four attributes of interest to us are linear momentum (kinetic or "translational" energy), angular momentum (rotational energy), vibration (vibrational energy), and the energy of electronic transitions. Energy transfers between molecules occur only in discrete amounts that depend on the values of these four attributes.

Although molecules in any but the smallest containers have an effectively continuous range of allowable kinetic energy values, this is not the case for the other energetically excitable modes of the molecule. Rotation, vibration and electronic transitions are all quantised in a way that affects their behaviour under normal conditions. If we want to model the behaviour of colliding molecules, we need to understand the quantum nature of molecular interactions.

As a broad but useful generalisation one can think of these four different "modes" of energy storage becoming "active" under different conditions. At absolute zero (0 K), each mode of the molecule is at its lowest possible energy, known as its "ground state". This implies that all the molecules in a sample are stationary, not rotating, not vibrating, and all of the electrons in each molecule are at their lowest possible electronic level. As the molecules heat up, a significant number of molecules will leave the ground state for these four modes. Once a significant proportion of the molecular population has left the ground state for a particular mode, the mode is said to have become 'active'.

The first mode to become active is the translational energy mode. This activation is caused by the molecule's straight-line motion. Technically this mode is quantised, but in fact in all but the smallest containers and at the lowest temperatures the possible energy levels are so close as to effectively form a continuum, so that the molecule can adopt almost any velocity.

As the temperature increases, the rotational modes of a molecule are the next to become active. Here the energy levels are not so closely spaced and it takes greater energies for the higher levels to become populated.

As the molecule continues to gain energy, the vibrational modes become active. The activity of this mode can be worked out by approximating the force field that the molecule's component atoms experience due to each other. Since they have a greatly restricted space, they are relatively highly quantised, and the allowable energies are further apart than for the rotational modes.

Finally, with increased heating, the electronic mode becomes active. The quantum equations governing this activation mode can be complex and difficult to solve for any but the very simplest molecules. However, it can be predicted that because the electrons have relatively little room to move, and are also very light, they have greatly separated energy levels, and this is indeed what is found experimentally.

At the point where one particular mode is just starting to become active, there will be only a small number of (quantised) energy levels for that mode, and it is necessary to work them out precisely. If, for example, the rotational states of a small molecule at cryogenic temperatures

were being considered, it would be necessary to explicitly examine each possible rotational state. But once a sufficient number are active, there are useful approximations that can be made which greatly simplify the analysis of the gas, especially if an aggregate is considered in place of an individual molecule.

The Boltzmann Distribution

Once the energy levels that are available to a molecule are known (either theoretically or from experimentation) it is possible to obtain the "Boltzmann distribution", which describes statistically the most probable configuration of energies of all the particles in a gas. This shows, for example, the distribution of translational energy in a gas at a given temperature, which is equivalent to describing the average speed, and speed distribution, of a gas. (This special case of the Boltzmann distribution is better known as "the Maxwell distribution".)

The Boltzmann distribution describes more than just the distribution of speeds in a population; it can predict the distribution of all energy modes, be they translational, rotational, vibrational or electronic.

The Equipartition Principle

A useful result that can be derived from the Boltzmann Distribution is that, when there are a large number of available energy levels the 'equipartition principle' may be used to work out how much energy is bound up in each mode.

The equipartition principle simply states that, provided there are a large number of populated energy levels available in a given mode, each active mode in a molecule will have a mean energy of $\frac{kT}{2}$, where k is the Boltzmann constant and T is the temperature. Thus all energy is 'equally partitioned', spread evenly across all possible energy modes. For a typical gas molecule, this might be three velocity modes (representing motion in the x, y and z directions) and two active rotational modes (for a diatomic molecule say, where the third rotational mode

is not yet active). The mean energy of each molecule would then be $\frac{5kT}{2}$ (since there are five available modes to absorb energy).

From this equation it is possible to compute the mean velocity of molecules in a gas, and also the mean rotational energy of these molecules. At very high temperatures, the mean vibrational and even electronic mean energies may be determined in this manner.

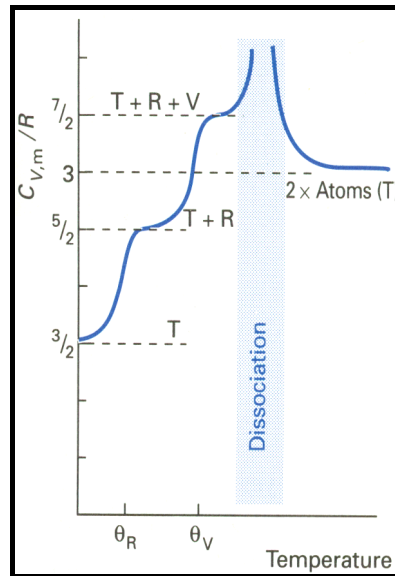


Figure 3.1: Heat capacity for a small diatomic molecule

For example, figure 3.1 shows a heat capacity graph²⁷ for a diatomic molecule with three translational energy modes, two active rotational energy modes, and two active vibrational modes. At first only the translational modes are active, then at θ_R the rotational energy levels become active. At θ_V the vibrational levels become active, and above this the molecule disassociates into its component atoms.

The Boltzmann Distribution and Big Molecules

It is a feature of quantum mechanics that the larger the molecule is, the more closely spaced are its quantised energy levels. Therefore, rather than working out the energy levels explicitly for large molecules, at normal laboratory conditions ('Standard Temperature and Pressure', being

10^5 Pa and 298.15 K), the Boltzmann distribution and the equipartition principle can be used to closely estimate the energies involved in movement and rotation.

This technique is used later in this thesis to justify the treatment of the angular motion for large molecules.

Brownian Motion

In an ideal gas, particles can be thought of as solid spheres in constant motion, continually ricocheting off each other in an endless succession of elastic collisions. In such a gas, the motion of each particle follows a ‘random walk’, also known as Brownian motion or a ‘drunkard's walk’. The particle moves a short random distance (usually considered to be on the order of 10s of nanometres²⁸) before colliding with another particle, and rebounding in a random direction.

Over time, the probable position of the particle is found to approximate a Gaussian distribution (covered in detail in chapter 5). The continual collisions, and the particle's velocity, cause the particle to be bounced away from its starting position. It is still most likely to be found in this general region, but it may have been bounced further afield. The interesting feature of this motion is that while the particles may have a large velocity, the actual distance travelled from their starting position may not be very great, due to the large amount of backtracking the particle may have done. Hence, even in a high temperature gas where the gas molecules are moving very quickly, the actual change in position of a particle over time may be only a small fraction of the distance travelled. There is a graphical representation of this (shown in a different context) in figure 5.2 of Chapter 5.

This random walk applies also to particles in a liquid, and heavy use of it is made to model motion in the simulator developed in later chapters. As well as emerging from a consideration of the motion of individual properties, the random walk can also be derived from consideration of the diffusion properties of large-scale aggregates of molecules. (cf. Appendix A for a detailed treatment of molecular motion.)

Liquids

Liquids are obviously very different from gases. In the ideal gases considered so far all interactions are rigid, inelastic collisions, with unconstrained motion between collisions.

In a liquid though, the molecules are in almost continuous interaction with their neighbours, via electrostatic forces such as dipole or van der Waals forces²⁹. In order to move, a molecule in a liquid must break away from at least some of its neighbours. But in doing so it will immediately fall under the influence of another molecule, and so on.

Another effect of this semi-continuous interaction is that the particles in a liquid are far more closely packed than in a gas. As a result, the motion of a liquid particle before it is involved in a collision with another particle (the "mean free path") is much shorter than it would be in a gas, even ignoring the longer range inter-molecular electrostatic forces.

The large scale effect of this semi-continuous interaction on a particle is, however, the same as in a gas. The particle moves in a random walk, although the "steps" of the random walk are much smaller, being on the order of 100pm for small molecules instead of 100nm for a small gas molecule³⁰. At the smallest scale, in a liquid paths are not even entirely straight, due to interactions with neighbouring particles. But the end result is identical: particles move in a random walk through the liquid, diffusing slowly to move to new positions that are far less distant from their original position than their raw velocities would suggest.

Modelling Liquids

Completely modelling the particles of a liquid is a very complex and computationally intensive task. Since the component molecules all interact with each other almost continuously, modelling a liquid in detail requires far more computation for a given number of molecules over a fixed period of time than for a gas (where the interactions are more infrequent).

Modelling Molecular Dynamics of Liquids

Some computer simulations of liquids attempt to carefully model the trajectories of all the interacting particles, by plotting their motions as the particles move through the potentials of their neighbouring particles; a method called "molecular dynamics". In order to be accurate, the time steps for such simulations must be very short, with complete recomputations of the entire system occurring on the order of femtoseconds. This is less than the average time between collisions, but allows the non-linear trajectories of the particles to be more accurately modelled. The result is still, unfortunately, an approximation, since it is an example of the Newtonian many-body problem which (generally) has no exact solution, but if the time step is made sufficiently small, it can be a very good approximation. Even more detailed simulations modelling the quantum mechanics of small systems can be created, but require a supercomputer to simulate a small number of atoms.

This method would be completely impractical for this thesis, where the intention is to simulate reactions occurring on the order of seconds. Even on the fastest supercomputer available in 2000, such simulations would be impossible using this method.

Monte Carlo Methods

This approach is more like the one used in this thesis, but at a much smaller scale. Each particle in the simulation is moved by a small, random amount, and the potential energy of the new configuration of molecules is calculated. Whether the new configuration is accepted, or whether another random position is generated for the particle, depends on the likelihood of the new configuration. In effect, if the new configuration is unlikely, dice are rolled to see whether or not it will be accepted. This procedure is repeated for all the molecules, and for numerous time steps, to build up a model of the liquid. The advantage of this more abstract method is lower computational requirements.

Modelling the Solute

The complexity of the problem can be greatly reduced if, rather than considering all the solvent molecules, only the smaller number of solute particles are considered. In fact, if the size of the solute molecules is sufficiently larger than that of the solvent molecules, the solute particles can simply be considered as moving in a constant density fluid, rather than in a sea of discrete particles. This approximation greatly simplifies the problem.

In order to model the solute molecules at a molecular level, it is necessary to know how their motion varies, depending on such interrelated factors as temperature, the mass and shape of the particle, the viscosity of the medium, and the frictional coefficient of the molecule. All this data is summarised by the diffusion constant, which describes how quickly molecules of a certain type will diffuse through a solution. From this, the measurable diffusive macro-behaviour of a chemical species, the mobility of an individual molecule can be found. (This mobility can also be theoretically approximated, see Chapter 5.)

The motion of a large molecule within a solvent is again found to be a random walk, and the diffusion constant allows the characterisation of the molecule's motion as a Gaussian distribution, representing the probability that at some given time in the future the particle will be at a particular position. This approximation holds so long as the chance of interaction with another, similar particle is not great. Since the concentration of solute particles, especially when they are macromolecules, is usually quite small, it is possible to model correspondingly larger time steps during computer simulation, often on the order of microseconds. This is nine orders of magnitude longer than the time steps required for molecular dynamics.

Modelling Rotational Motion

Modelling rotational motion, although a little more difficult from a practical point of view, is in theory very similar to modelling linear motion. Whereas linear motion deals with position, velocity, force, mass, and acceleration, rotational motion deals with angular position, angular

velocity, torque, moment of inertia and angular acceleration. In many respects the angular properties are direct analogues of the linear ones, and equations such as "force equals mass times acceleration" become "torque equals moment of inertia times angular acceleration".

The equipartition principle can be used to estimate the statistical distribution of the rotational energy in a group of molecules. Combining this with a molecule's moment of inertia provides the rotational velocity distribution, which can then be used to describe the movement of the molecule. The moment of inertia is easily calculated, as is the centroid, (or centre of rotation) of the molecule.

Unfortunately, although this approach works well for a gaseous molecule, it is less accurate for one suspended in a liquid. A molecule in a liquid is not free to rotate without interference from neighbours, and in fact liquids generally display 'short range order', where the local neighbourhood of a liquid molecule displays a crystalline, solid-like structure, that rapidly degrades. For example, water displays short-range order at the level of individual water molecules similar to that found in ice; it is simply that the structure is imperfect, and the molecules constantly moving, that keeps water from being a solid. If the degree of molecular movement is slowed by cooling, this short range order becomes greater and greater, until the liquid freezes.

Even more complex is the case of a macromolecule. In the most common solvent, water, macromolecules are surrounded by a layer of ordered water molecules known as a 'hydration sphere' (or more generally, a 'solvation sphere') - which increases its effective size to the 'hydrodynamic radius', which must also be taken into account in linear motion. The hydration sphere hinders the macromolecule's rotational motion, as do large non-spherical deviations in the macromolecule's shape. These features make it difficult to predict the macromolecule's rotational behaviour, which is also difficult to measure directly. The effect can be partially compensated for by using a larger volume when calculating theoretical figures, and by using experimental data where possible.

Modelling Chemical Interactions

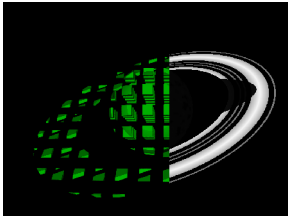
When two molecules collide it is possible they will react together. Unfortunately, determining exactly the conditions and probabilities of a reaction, especially for larger molecules, is a very difficult task, and usually resort is made to direct experiment. But, once experiment has determined values for one set of conditions, they can often be extrapolated to other circumstances.

The theory, in its simplest form³¹, simply treats the molecules of a gas as having both a collision cross-section (the size of the particle used for calculating if it is hit by another particle), and a 'reaction cross-section' related to the collision cross-section by a 'steric factor'. The steric factor is usually much less than 1, and might be interpreted as expressing the fact that reaction is probabilistic: particles that collide have a chance of reacting, but do not necessarily do so.

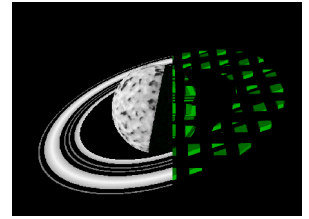
The theory can be extended to liquid reactants, but again heavy reliance must be made on experimental work. Hence, in the models developed in this thesis (where possible) the values for interaction are obtained by reference to experimentally obtained data.

Chapter 4

An Introduction to Computer Analysis



Ne quis ineat qui rationem inire nescit
(Let no-one enter who knows not how to compute)



- anon

Introduction

Computer modelling techniques and, to a lesser extent, image processing techniques, are used widely throughout this thesis. In order to make the material introduced in later chapters more familiar, a brief qualitative introduction to the basics of image processing and computer modelling is given in this chapter.

Image processing is primarily a set of techniques to make images more meaningful to humans, as opposed to computers (many computer techniques work best with the original, unaltered images). Image processing works by removing features of an image that confuse the human eye, and enhancing other features that make the underlying structure of the image recognisable. This is often performed by attempting to understand the physical process that caused the imperfections in the image and then, by mathematically reversing their effect, returning to a closer approximation of the 'ideal', defect-free image. Image processing is now a standard procedure in high-resolution optical microscopy.

Computer modelling attempts to imitate on a computer a real world object or process, by using mathematical equations and rule-of-thumb heuristics. This is usually done in order to investigate a real world system in a cost effective way: it is cheaper to make a model of a bridge, and test assumptions about weight and loads on its component girders, than to build the bridge and experiment on the real object.

Image Processing

Image processing has become a common technique for making images more comprehensible to the human eye. In recent years image processing techniques have become so commonly used that they are now routinely built into common everyday appliances. Scanners and fax machines regularly do contrast and brightness enhancement and balancing, and many digital cameras and video cameras use techniques such as Fourier transformation³² in order to automatically obtain focal lengths.

It is important to note, however, that image processing cannot add anything to a picture that was not there already. Popular cinema aside, it is not possible to take, say, a satellite photograph with a resolution of 10 centimetres, and image process it to reveal the details on a car licence plate. If the information does not exist in some form in the image, no amount of image processing can produce it. There is also a danger with some image processing techniques that artefacts will be introduced that did not exist in the original, by overemphasising random variations in the original image.

The following sections outline a number of the more common techniques that are used later in this thesis.

Image Smoothing

This is the computational equivalent of physically spreading Vaseline on a camera lens before taking a photograph (a technique actually used by cameramen in the past). It blurs the image slightly, allowing colour from one location to seep into the surrounding locations.

This has a number of uses. First, if an image is degraded, smoothing may result in an image more meaningful to human eyes. An example might be a particularly grainy photograph that, when blurred, may appear more natural. Another example is a picture with scattered point-like defects in it (see Fig. 4.1) - smoothing the image may make such defects less apparent.

Second, if an image is obtained via some digital technique such as a fax or scanner, it is possible that the image has been artificially sharpened - that is, that the edges of objects in the image have in some way been intensified. Smoothing, which leaves large regions of the same colour and intensity unchanged, but smears edges, may restore an image's original appearance.

The danger with image smoothing is that it may blur lines and borders, and that it will destroy the fine detail of a picture. The same process that prevents us from seeing the wrinkles in an actor's skin may prevent us from seeing the fine structure of an organelle under an electron microscope, if we use image smoothing to try to improve our image.

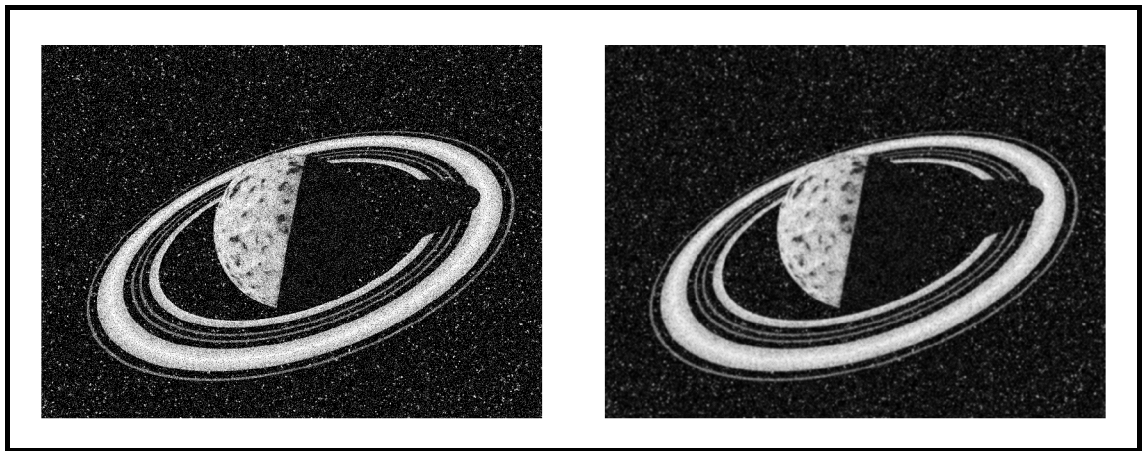


Figure 4.1: An image with 'spot' noise (left) is enhanced by blurring (right).

Image Sharpening

The converse of image smoothing is image sharpening, which deliberately intensifies the edges of objects (Fig. 4.2). While there are not many physical analogues of this process, it is actually similar to what the human optical system does. The eye, in combination with the various visual portions of the nervous system, acts to emphasise lines and borders, at the expense of broad regions of colour - an adaptation that allows swift recognition of the outline of an object.

The difficulty with image sharpening lies in its sensitivity to changes in shade. Whereas it can easily emphasise a boundary between two slightly different shades of grey, it will also emphasise boundaries between an isolated dark or bright dot - thus if there is any noise in an image, image sharpening will increase it, possibly making the image useless.

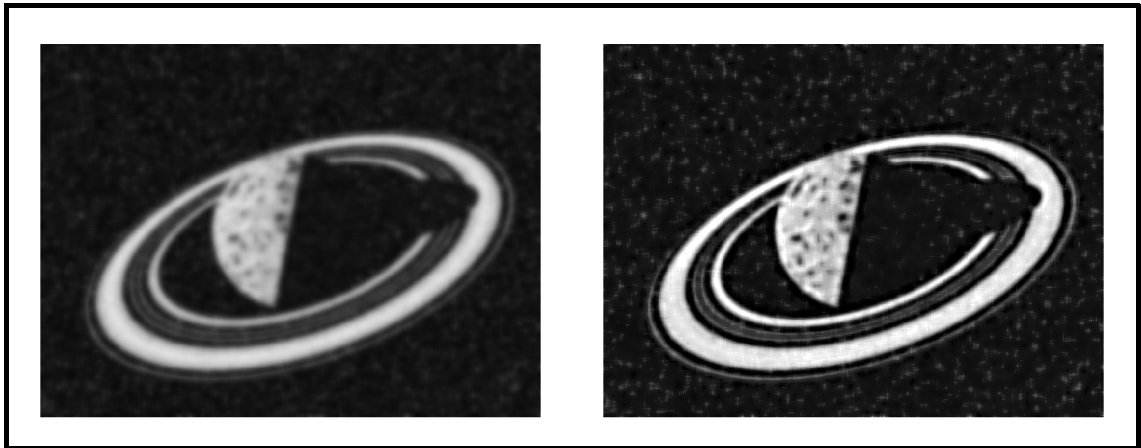


Figure 4.2: A blurred image (left) is enhanced by image sharpening (right).

Histogram Equalisation

Histogram equalisation is used to process images that are lacking in contrast. The lack of contrast may be in all shades (when an image is under- or over-exposed), or may be due to certain shades 'bunching up'. The latter occurs when the image is correctly exposed, but the objects photographed have insufficient contrast between each other. An example of this might be a photograph of a herd of similarly coloured grey elephants - the photograph may be perfectly exposed, but it is difficult to distinguish individual elephants due to their similar shades of grey.

In some ways histogram equalisation is similar to what happens when a skilled developer processes a negative. With the proper use of photographic paper, an under- or over-exposed image may have its contrast improved so that the image appears normal. For example, the intensity range of a heavily under-exposed image, (that would normally appear to be black and shades of dark grey), may be 'stretched' so that while the black stays black, the lightest of the

dark greys becomes white. All the intermediate shades of dark grey are interpolated between white and black, but all become lighter than before.

A computer can also perform this task, but with slightly more control over the final shading. Since a computer image has only a finite number of elements, or pixels, and a finite number of shades of grey, we can tabulate the number of pixels represented with each shade of grey into a histogram (i.e. a frequency count). Using this histogram the computer can assign each pixel for a given shade of grey to a new value, possibly producing a more informative picture. Histogram equalisation is the process whereby the computer attempts to redistribute the shades of the picture to produce a more easily interpreted image. One way of doing this is to ensure that the picture has equal quantities of all the different shades from white to black. There are other results one can obtain using a histogram, such as determining a set of shadings that are distributed normally around the mean shade. This can lead to more 'natural'-looking photographs.

The method used in figure 4.3 is yet another approach, where the image is 'stretched', so that the lightest shade (which in the example is still very dark) becomes white, and all the other shades are mapped linearly to intermediate values.

The example below (Fig. 4.3) demonstrates the use of histograms. The original image is shown in the top left-hand corner. In the top right-hand corner is the same image, darkened considerably (note that this darkening has caused some loss of information - it is not possible to completely recover the original image from the dark image).

There is still considerable room for improvement of this very dark image, however. Simply brightening the image provides some improvement, as can be seen in the image on the bottom left, though it still looks very washed out and gray. On the other hand, histogram equalisation, shown bottom right, that has stretched dark greys to light grey and white, while leaving pure black unchanged. This recovers most of the original contrast. This technique is often used to improve badly exposed photographs, and indeed is often done in hardware by modern digital cameras.

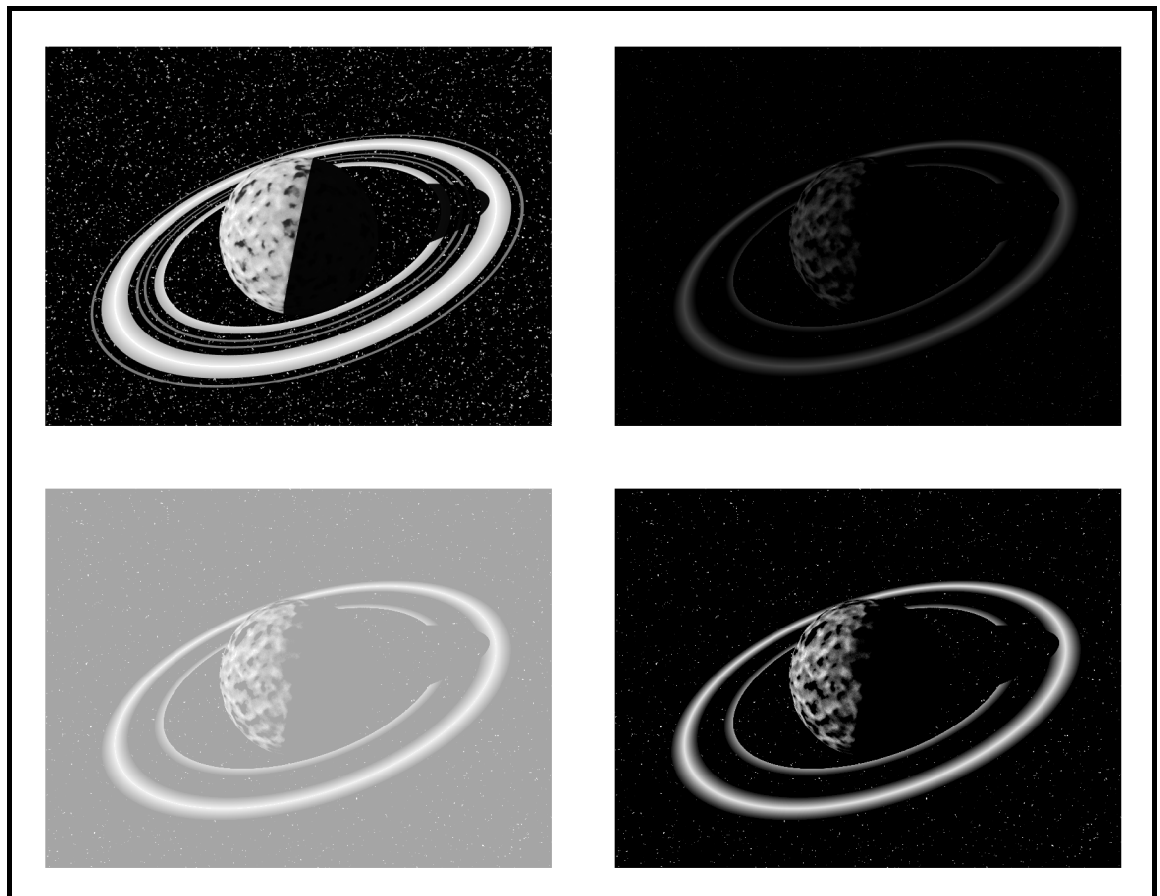


Figure 4.3: An original image (top left) is darkened (top right). The dark image undergoes brightness enhancement (bottom left) and histogram stretching (bottom right).

Localised Histograms

A more elaborate use of histogram techniques is to apply the same principle on a regional basis to an image. This allows detail to be extracted from regions that are unusually lacking in contrast within a larger image that may be perfectly well adjusted otherwise. For example, in a picture of a dark cave entrance taken on a bright sunny day, the cave entrance may appear entirely featureless, but localised histogram equalisation may be used to reveal the black bear lurking within.

There are a number of ways that localised histograms can be used. A naive approach is simply to divide an image into subsections, and individually process each subsection as if it were an entire image. The shortcoming of this approach is revealed when an interesting region straddles the intersection of multiple subregions.

A better approach (implemented by the author as part of the work described in chapter 10) is to construct a histogram of the region surrounding each pixel in the image (Fig. 4.4). (Although this might seem labourious it is possible to keep a running total of the histogram as the focus of the algorithm moves from pixel to pixel.) Using this method, each pixel is guaranteed to be adjusted appropriately to its surroundings. Of course, the exact size of the surrounding area that should be taken into account is a configurable parameter.

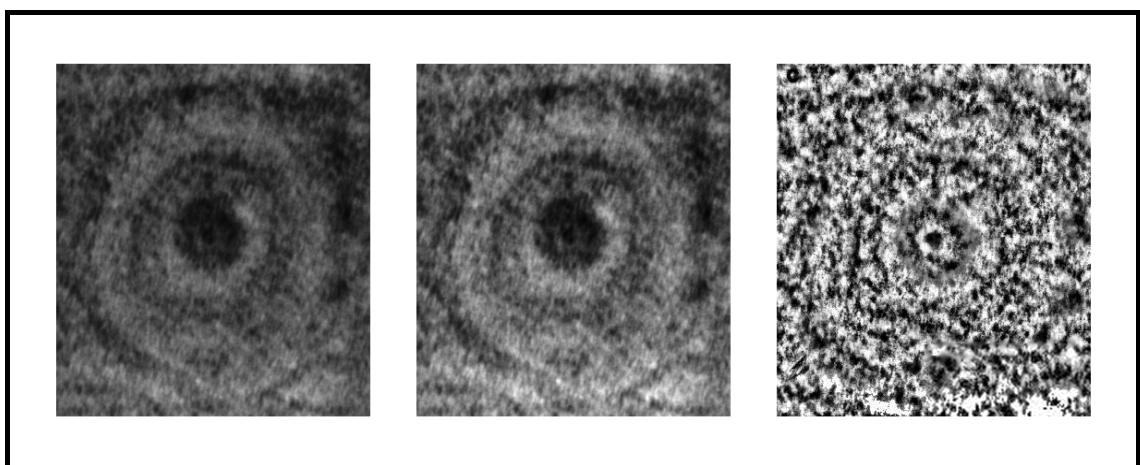


Figure 4.4: An original image of a plasmodesmal cross-section (left) benefits from histogram equalisation (centre), but details of the dark inner core are only revealed by local histogram equalisation (right).

Fourier Transforms and Bandpass Filtering³³

Another set of common image processing techniques use Fourier transforms. A Fourier transform models a particular shape as a series of overlapping sine waves of different frequencies - a process that is a little counterintuitive, but nonetheless very powerful. Fourier transforms can be applied in two dimensions, the image being transformed into a two-dimensional grid of the different frequency waveforms, or components, underlying the image. The Fourier transform takes a normal image of pixels, such as a scanned photograph, and converts it into an image in 'frequency space'. Since no information is lost during the process, the resultant Fourier transform of frequency components can be represented as another image of the same size as the original, usually having the appearance of a black image with a regular pattern of gray dots.

The upshot of this is that features of the image that show regular periodicity throughout the entire image can be manipulated - for example, artefacts such as regular scan lines can be removed from an image by removing the appropriate frequencies in the 'frequency space', or Fourier transform image.

An image can also be 'smoothed', and high-frequency noise (i.e. isolated spots and lines) in the image can be removed by eliminating all the high-frequency components of the Fourier transform image. Conversely, small features such as edges and points can be enhanced by removing all the low-frequency components of the Fourier transform image. This exclusion of a set of frequencies in the Fourier transform is called 'bandpass' filtering (since a range, or band, of frequencies is included/excluded).

In fact Fourier transforms were not widely used in this thesis, but a short investigation of angular periodicity was made using a variant of the Fourier transform, the polar Fourier transform.

Polar Fourier Transforms and Bandpass Filtering

A variant of the Fourier transform is the polar Fourier transform. A normal Fourier transform breaks up an image into a set of Cartesian frequencies, that is the frequency components are arrayed on the x and y axis. But it is also possible to represent an image in polar co-ordinates (i.e. angle and radius) and then use a Fourier transform to find the frequency components in the angular and radial directions.

The advantage of this is that it makes it possible to emphasise features with radial symmetry (i.e. concentric rings) and angular symmetry (such as the spokes on a wheel) by bandpass filtering. If the image is noisy, removing the high-frequency components will reveal any underlying radial or angular regularities, while conversely removing low-frequency components will emphasise the edge features of any such angular or radial features.

This technique is used briefly in chapter 10 to examine the angular symmetry of a biological specimen. The example below (Fig. 4.5) shows the same example of a polar Fourier transform.

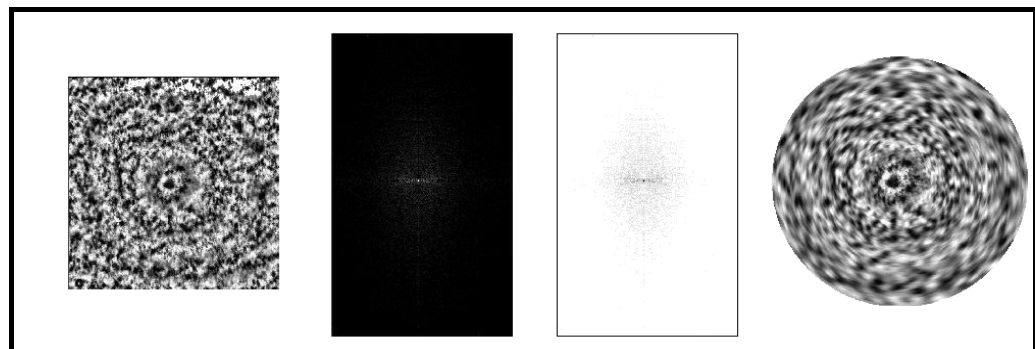


Figure 4.5: A polar Fourier transform - from left to right: the image from figure 4.4 showing some radial symmetry, the polar Fourier transform, the same transform as a negative to show detail, and the resulting image after high-frequency bandpass filtering.

Limitations of Image Processing

The most obvious limitation of image processing, as mentioned in the introduction, is that it cannot reveal details that are not in some way present in the original picture. Further difficulties occur if the techniques are misused - if images are over-processed it is possible to produce artefacts that are not present in the original image, by amplifying random noise until it becomes a feature of the image.

For these reasons it is important to always have the original image available when viewing the results of image processing, so as to be able to compare the image-processed result with its base image.

Modelling and Simulation

Modelling is a broad term covering any abstract representation of a physical object or process. The term 'model' is sometimes reserved for static representations, while 'simulation' is used for representations of dynamic processes, but the difference is largely arbitrary, and the two terms are used interchangeably within this thesis.

Modelling has been used since ancient times as a precursor to construction, and this use continues today. Modelling of the sort needed for modern engineering requires consideration of a great many physical parameters, such as gravity, material properties and wind loading. A mathematical abstraction is constructed of the object, and mathematical terms representing the various physical systems that act on or comprise the object are introduced into the system of equations.

The modelling performed in this thesis is of a similar type - physical objects at the nanometre scale are modelled as a system of equations, and the various physical forces acting on them are introduced as terms in these equations. Due to the complexity of the problem, these various equations are broken up into separate computer algorithms and solved numerically, rather than analytically.

Analytic Modelling³⁴

Under some circumstances, the mathematical equations underlying the model of a particular system may be amenable to a theoretically exact, analytical solution. An example of this is a simple model of the ballistic flight of a ball, where the equations of motion can be expressed mathematically, as $y = \frac{1}{2}gt^2 + v_y t$ and $x = v_x t$ (where 'g' is gravity, 't' is time, and v_y and v_x are the initial velocities in the vertical and horizontal directions respectively). These equations yield exact solutions for all values of 't'.

Numerical Modelling³⁵

Unfortunately many real-world situations cannot be modelled accurately with analytic techniques, because the equations involved are too complex to yield exact solutions. If the above example of a ball in flight was to include consideration of air resistance, turbulent airflow, wind gusts, and gravitational variation, it would probably not be susceptible to an analytic solution.

It is usually possible, however, to obtain approximate answers of arbitrary accuracy using numerical techniques. In these techniques, a 'guess' at a solution is made and inserted into the system of equations. The guess is then modified by small amounts until a solution of acceptable accuracy is derived.

Before the introduction of computers these types of calculations, which were common in many branches of engineering, were labouriously performed either by pen and paper, or using slide

rules^{36,37}. With the introduction of computers, such calculations can be performed far more swiftly, accurately and reliably in an automated fashion.

A related form of modelling is that of graphical modelling, where complex shapes are represented (again mathematically) by the computer, and used to produce visual representations that are presented to the user (who is usually able to view the simulated object in a number of ways). That each such view of the mathematically modelled objects requires a host of numerical solutions to various 3-D viewing equations is of no consequence to the user, since such calculations can be performed rapidly, in real-time, on a modern computer. The visualisation of the state of simulated protein assembly processes in this thesis is done in this way.

Static Models

Another way of classifying models, independently of whether they are analytically or numerically solved, is whether the model describes a process that is changing over time, or is static. Much modelling is done using static representations. Structures such as houses and bridges that are presumed to be immobile are modelled using static models, or indeed the original DNA model by Watson and Crick³⁸. This thesis uses static models to describe theorised biological structures in chapter 10.

Dynamic Models

Other objects, which change over time, are represented by dynamic models. Such models are often more complex than static models, because they may have to model all the features required of a static model, and repeat these calculations frequently as time progresses (e.g. it is easier to model a stable house than the same house sliding down a slope during an earthquake).

A common approach to dynamic modelling is to break the simulation into a number of discrete instants. This is a type of numerical approximation in which a static model is constructed for

a particular instant, and the forces acting on every relevant part of the model are obtained. It is then calculated how far all these objects will move during a short ‘time step’, when the calculations are repeated using the new object positions. This approach, like most numerical methods, can be made arbitrarily precise by choosing smaller and smaller time steps. The nano scale simulator described in this thesis works using time steps the length of which can be set as a program parameter at run time.

Limitations of Modelling and Simulation

The fundamental problem of all modelling is that no matter how elaborate the model, it can never approach the complexity of the real world physics underlying the modelled object, with all its component molecules and atoms, and all the external real-world objects that act upon it. This is sometimes expressed by the colloquial phrase ‘the map is not the territory’, or as the famous picture by Magritte (Fig. 4.6), which is indeed not a pipe, but rather a picture of a pipe. In order to do any meaningful modelling it is necessary to radically reduce the real-world situation to a manageable size.



Figure 4.6: A model of a pipe by the artist Magritte: ‘The Treason of the Pictures (this is not a pipe)’, 1928

Similarly most complex modelling requires even the mathematics of the simplified model to be approximated, and the degree to which this approximation is done must be selected with care. Almost always (and certainly in this thesis) there are trade-offs to be made between accuracy, speed, the usability of the resulting data, and how closely the mathematical model approaches the physical reality being modelled.

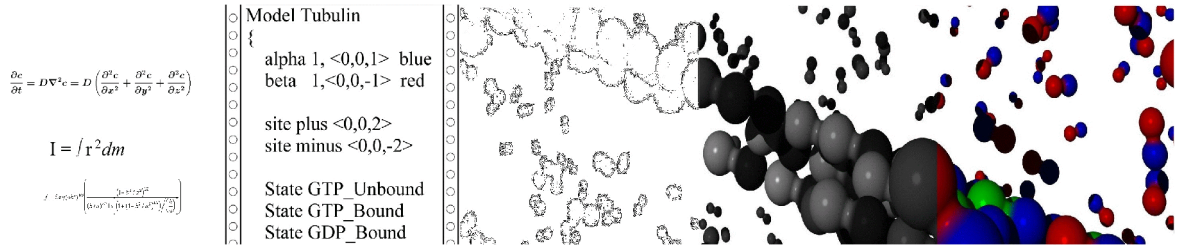
Despite these limitations, modelling has been an extraordinarily successful method of understanding the world, and models with known limitations (e.g. the Newtonian theory of gravitation) are used to great effect by scientists and engineers every day.

Note on examples

With the exception of figure 4.6, all the exemplar images in this chapter were created by the author using “Paint Shop Pro 5.0” by Jasc Software Pty. Ltd., (for the non-Fourier transform imageprocessing examples), or the “Image Explorer” suite of image manipulation tools supplied by SGI for the Fourier Transform example. The raw image of the ringed planet was created by the author using the freeware ‘Povray’ raytracer program, while the cross-sectional TEM of a plasmodesma was provided courtesy of Janine Radford, Dept. Biological Sciences Monash University, Melbourne, Victoria, Australia.

Chapter 5:

Simulating Protein Self-Assembly



Sic parvis componere magna solebam

(In this manner I was accustomed to compare great things to small.)

- Vergil

Introduction

There are many dynamic systems within the cell, and in other small-scale systems, that are difficult to fully understand, and hard to study using conventional experimental analysis. The exact method by which proteins of the cytoskeleton combine, the way that viruses assemble out of their component proteins, and the general interaction of proteins and other macromolecules are all difficult to determine by observation alone.

Direct inspection of living tissue is usually unable to resolve these interactions, except in aggregate form, while higher magnification methods generally work only on specially prepared tissue. The processing needed to prepare specimens for most high magnification techniques, such as electron microscopy, usually kills and immobilises the tissue, halting any dynamic processes. While some preparatory techniques such as rapid freezing may preserve a snapshot of a dynamic process, it is still often difficult to determine the full process, although such frozen specimens can provide useful data about such dynamic processes.³⁹

Indirect methods can, however, be very useful. Techniques such as protein sequencing and X-ray crystallography can reveal the shapes of proteins, and these and other techniques can be used to determine what areas of the protein are active and can bind to other molecules. They

and may even suggest how a protein may change its conformation, and under what conditions. Other methods, such as analysing reaction rates and deducing early states, can also be very useful, as will be shown below.

Despite all these methods, however, it can still be difficult to establish the combinatorial nature of macromolecular interactions at the nano metre scale. Even if the initial conditions and the end result are known, the manner by which the reaction proceeded may still be uncertain. Establishing this method can reveal many interesting things about how structures at this scale function.

Mathematical Models

The most successful method for studying dynamic processes so far has been mathematical analysis of experimental results. Using observations of bulk properties such as gross polymerisation rates, a surprising amount of detail can be inferred.

The Oosawa Model

Fumio Oosawa was a pioneer of this approach. Using a simple mathematical model of polymerisation similar to that used in polymer chemistry, he and his co-workers were able to accurately model the assembly of a number of self-assembling proteins. Much of their work concentrated on actin and bacterial flagella, but they also reproduced a variety of other, less easily modelled, self-assembling protein structures.⁴⁰

The Oosawa model, in essence, treats the growth rate of a polymer as being independent of its length, once the polymer is larger than a certain critical size, known as the 'stable nucleus' size. For a polymer larger than this size, the growth of the polymer can be expressed as a combination of a growth rate, multiplied by the concentration of 'available' free monomers that can be bound, minus the disassociation rate (or loss rate) of bound monomers breaking away from the polymer.

For a given polymer of sufficient size, the growth rate G is

$$G = k_+ c_1 - k_- \quad (5.1)$$

where

k_+ = the binding rate of free monomers to the polymer

k_- = the loss rate of bound monomers from the polymer

c_1 = the concentration of free monomers to be bound

Applied to a single polymer, the above equation must be interpreted in terms of *probabilities*, since the binding and loss rates are only accurate in aggregate. However, this equation can be used to provide a description of the change in the population of all polymers with sub-units of size i , and hence a description of the population of all polymers;⁴¹

$$\frac{dc_i}{dt} = k_+ c_{i-1} c_1 - k_- c_i - k_+ c_i c_1 + k_- c_{i+1} \quad (5.2)$$

where

c_i = the concentration of polymers of size 'i' sub units

t = time

k_+ = the binding rate of free monomers to the polymer

k_- = the loss rate of bound monomers from the polymer

c_1 = the concentration of free monomers to be bound

This equation tells us that the number of polymers of a particular size i subunits

- increases as smaller polymers of size $i-1$ grow to size 'i' (the first term above),
- decreases as polymers of size i lose monomers and sink to size $i-1$,
- decreases as polymers of size i grow to a larger size (third term), and
- increases as larger, $i+1$ size polymers shrink to size i .

Since the rate constants are assumed under this model to be independent of polymer size, they are constant for all size $i > \text{nucleus}$.

A similar equation is used to describe the production of nuclei, that is polymers of critical size i_o , that are assumed to have different ‘on’ and ‘off’ rates:

$$\frac{dc_{i_o}}{dt} = k_+^* c_1 - k_-^* c_{i_o} - k_+ c_{i_o} c_1 + k_- c_{i_o+1} \quad (5.3)$$

where, in addition to the variables described above,

k_+^* = the creation rate of nuclei from free monomers

k_-^* = the destruction rate of nuclei to free monomers

By fitting these two equations with experimentally observed ‘on’ and ‘off’ rates k , (and by deriving further results) it is possible to match the experimentally observed behaviour of some protein species’ polymerisation, reproducing details such as the bulk polymerisation over time and the steady-state size distribution of polymers.

Unfortunately, the techniques used by Oosawa for actin and other polymers do not work so well when applied to more complex polymers, such as tubulin or viral capsids. The Oosawa model makes a number of assumptions that are not universally valid.

- It assumes a single critical nucleus stage.
- It assumes all growth is by single monomer addition.
- It assumes constant k ‘on’ and ‘off’ rates, independent of size or geometry.
- It does not attempt to model the role of any ‘helper’ proteins that may also be involved.
- It does not handle the assembly of heterogenous protein assemblies, such as viral capsids, where multiple protein species are involved.
- It ignores the inhomogeneous nature of the protein environment, such as differences in the free monomer concentration gradient.

These limitations are often not a concern, because for many proteins the Oosawa model produces accurate results, implying that the above assumptions are, for that particular case, valid.

More Complex Mathematical Models

More elaborate methods generalising the approach taken by Oosawa can be used to tackle more complex systems, and produce more accurate results. For example, by extending Oosawa's equations to cover two stages of nucleus assembly,⁴² or multiple intermediate stages of assembly⁴³, the formation of microtubules can be more closely modelled.

While such methods give increasingly accurate results, there is still a degree of approximation involved, because the mathematical models must still make a number of simplifying assumptions. Usually the same assumptions as the Oosawa model are made (single monomer addition, constant ' k ' rates, no helper proteins, a single protein species, and a homogenous environment), with a more detailed handling of the nucleation process being the primary change. Even this single change can result in complex series of simultaneous non-linear equations being generated.⁴⁴

Rule Based Models

For some systems, useful results can be obtained using "rule based" systems, where each protein assembles into a structure dependant on simple positional rules, which adjust the type and angle of bonds. Such a simulation has been used effectively by Berger et al. to study viral capsid assembly⁴⁵. Rule based simulations can give important insights into structural assembly, as well as incorrect assembly, of complex protein structures.

This type of simulation is not, in its simplest form, a dynamic simulation, but is almost entirely mathematical in nature. It can however form a useful basis for more elaborate simulations, as described later in this thesis.

Computer Simulations

In order to better understand the structure and dynamics of a number of these self-assembling complexes, and to overcome some of the limitations of traditional mathematical analysis, it is useful to simulate on a computer the motion and behaviour of self-assembling particles such as proteins.

Of course, this is in fact simply the creation of yet another even more complex mathematical model. But, by expressing our model as a series of computer algorithms, the dynamic system can be modelled in far more detail, in three dimensions, with the aim of capturing all the significant details of the physical processes involved. If the computer model is sufficient, and correctly simulates these processes, it should produce more accurate simulated results (in comparison to experiment) than the simpler mathematical models, and should be able to make predictions about the behaviour of new systems.

In addition, computer simulation allows the relaxation of all of the above mentioned constraints. Multiple-monomer addition and differentiated addition and subtraction k rates can be modelled, as can complex interactions between multiple protein species.

The shortcomings of a computer based approach are that it does not yield analytical solutions in the way that a traditional mathematical model does, and the simulation may require so much input data that its predictive power is compromised.

However, a computer simulation can provide far more information than a traditional mathematical model. A computer model can be used to study details of processes missing in other models. Considering just the process by which tubulin assembles to form microtubules (tubulin being a protein whose behaviour is not fully described by the Oosawa model), researchers have used simulation to study a number of features of this process, such as the addition and subtraction processes occurring at a microtubule growth tip,⁴⁶ and the growth and cyclic behaviour of microtubule populations^{47, 48}.

The Nanoscale Simulator

The Need for the Program

While attempting to model the dynamics of just one such self-assembling protein, tubulin, it became apparent that a generic simulation program dealing with small objects of nanometre scale in solution would be desirable. Indeed, such a simulation, if sufficiently flexible, could be extended to study otherwise static structures involved with dynamic objects (such as the flow of molecules through plant plasmodesmata, or the deposition of stain particles on an electron microscope specimen).

In order to better understand how these and other proteins combine, a generic computer model to simulate their interactions is needed. By comparing the results of this computer model with experimental data, the gross accuracy of the model can be established. While comparable large-scale results do not *prove* that the model is accurately simulating the fine-scale detail of the molecular interactions, it does provide supporting evidence. By attempting to make the details of the model as physically accurate as possible (and checking that, even in the absence of chemical interactions, the simulator correctly models the basic laws of physical chemistry) further credibility is added to the simulator's portrayal of the dynamics of nano-scale self-assembly.

Applications of the Program

The resulting program, dubbed the 'nanoscale simulator', (or 'nanosim' for short) has grown from its original roots as a simulator for a single type of protein, tubulin, and now covers a wide range of nanoscale objects. It is able to model multiple different types of objects (each with their own specific movement and interactive properties), interacting in the same environment. It also models a variety of different states for each individual object - for example whether the object is phosphorylated or not - and defines a set of events for each object that may change that state.

In addition to proving useful in the study of the cytoskeletal and cytoskeleton-related proteinaceous structures originally aimed for, the nanoscale simulator may have a much wider application. There are many other biological self-assembling systems of interest, and it is even conceivable that the simulator would be of value for inorganic systems, such as the micro-machines envisaged by the proponents of nanotechnology⁴⁹.

By allowing us to simulate the behaviour of new systems, the nanosimulator may also have immediate practical benefits. For example, the behaviour of a new, sub-stoichiometric drug that binds to a particular protein in the growing structure to disrupt the assembly process, might be readily modelled^{**}. This may be a handy adjunct or precursor to normal biochemical analysis.

Many anti-cancer drugs, for example, disrupt the cytoskeleton (and hence cell division), to affect rapidly dividing cells such as cancer cells. They do this in very small quantities, presumably by adding defects to the growing cytoskeletal structures, which either renders them incapable of further growth, or encourages their disassembly. Similarly, some anti-viral drugs operate by interfering with the self assembly (or even disassembly⁵⁰) of the viral coat by its constituent protein capsids.

The program will be described in detail in later chapters, along with several significant results. In this chapter details of the physical chemistry that the simulator will be required to model are given, setting the stage for a later, more detailed discussion of programming in Chapter 6.

^{**} A sub-stoichiometric drug is one where a very small amount of the drug has a great effect, because less than one molecule of the drug is needed per molecule of substance that the drug interacts with. In an anti-tubulin drug, for instance, it may be enough for a single molecule of the drug to bind with the growth tip of a microtubule, containing thousands of tubulin molecules, to disrupt the entire microtubule

The Range and Application of the Nanoscale Simulator

The program simulates the movement, interaction and behaviour of objects of roughly 1 nanometre to 100 nanometres in size, moving in solution, over time periods of microseconds to seconds.

The scale of this simulator falls between those of the very, very small-scale simulators that deal with individual atoms (which operate on the scale of picometres and femtoseconds) and of traditional mechanics programs (which model 'real world', mechanical objects on scales of millimetres and metres, and work on the order of seconds or minutes).

The main thrust of the simulator is examining self-assembling objects, and simple forms of 'emergent behaviour' that occur when simple rules applying to low level structural components have significant resultant effects on the high level structures they compose. By modelling individual large molecules it is hoped to derive useful higher-order results, such as the behaviour of the large-scale aggregates they may form.

In order to model the behaviour of these large molecules, the behaviour of the smaller solvent molecules must also be implicitly modelled. General rules for the physics of simulated objects must be defined such that the same rules can be used both for free objects and for objects that have collected together into aggregates.

The simulator does this by modelling the basic physics of the environment. Since to do this completely accurately is computationally infeasible, a number of assumptions and simplifications are made about the way molecules interact with their environment. In this way an abstracted, but manageable, model of the system to be simulated is used.

The simulator models mixed solutions of different types of molecules. Normally the simulator starts with an unstructured environment, with all the components randomly distributed throughout the simulation space. However it is also capable of starting with a more ordered

environment (read in from a file), such as one containing a pre-existing aggregate or indeed a large-scale proteinaceous structure (such as a modelled plasmodesmata).

Using the physical model and the description of the objects it is given, the simulator examines how molecules at this nano-scale level interact, and particularly how large-scale aggregates form and change. The main work that the simulator has been used for so far is looking at actin and tubulin, and the filaments and microtubules they build up. But it has also been used for a number of other purposes, such as modelling the deposition of stain particles on a sample for electron microscopy, and preliminary work on viral self-assembly.

Other Work in the Field

Recent work published by Schwartz et al. follows a similar general path⁵¹. R. Schwartz et. al. combine a rules based system with a kinetic simulator to perform a general 3-D simulation of viral capsid assembly. The rule-based kinetic model uses a sophisticated representation of bonds with lengths, directions, rotation vectors, association and dissociation activation energies, spring constants for bound molecule interactions, allowable neighbours and tolerances. This model also uses a rule-based system for protein conformational change.

A number of these attributes, specifically the variation in bond angles and the activation energy based binding/unbinding rules, are not covered in the model presented in this thesis. The computational load involved in modelling these features is, according to the authors, very great, and limits the number of particles that can be simulated within a reasonable time to a few hundred (on an 8-processor Sun Ultra Enterprise 5000 symmetric multiprocessor), but provides an insight into the dynamics *within* the assembled structure, which is lacking in this thesis.

The model presented is very powerful, and there is no reason to suppose that it cannot be extended to handle its current (apparent) limitations:

- It does not yet handle multiple protein species.
- Currently it only models abstracted viral capsid proteins, and has not been directly compared to experimental assembly results.
- The time scale is abstracted.
- The motion model appears to be a Newtonian model with viscous forces and a randomising Brownian force limited to “avoid drastically altering subunit paths over short distances”.
- These possible shortcomings with the modelling of linear motion mean that rotational behaviour of aggregates is not strongly modelled.
- The simulation field has a hard boundary that bounces subunits back into the field - depending on other factors this might lead to unrealistic concentration effects, especially in the corners of the simulation field.

This “rules based simulation” does have a number of advantages over the work presented in this thesis, especially in its detailed model of the intra-aggregate behaviour of proteins in the assembled aggregate. Inter protein bonds are represented as having a characteristic angle and length, with translational, rotational and bending forces pushing the bond towards its “ideal” position. This makes the bound proteins dynamic, and allows for investigation of “incorrect” bindings, where a complex structure (such as a virus shell) assembles incorrectly because of a protein binding in an incorrect position due to cumulative error caused by the bound units not being in their “ideal” positions⁵².

An Overview of the Nanoscale Simulator

The simulator models the physical environment of objects (usually proteins) moving in a solvent. Since the scale of the simulation is well above the femtosecond level of individual atoms, it does not attempt to model the direct velocities and forces involved at the lowest level. Similarly, interactions between objects are not made at a low, atomic level. Rather, potential "binding sites" for the modelled proteins are identified, and a probabilistic method determines whether binding occurs.

The simulator works in a series of repeated time steps, or "program cycles". A single program cycle might model a period of ten microseconds, during which particles will move, collide, and perhaps bind with others. After each cycle some form of data output may occur, and then the next cycle is run.

The program is thus required to accurately model:

- particle movement;
- particle collisions;
- particle binding (of the same, and different, protein species);
- particles breaking off from polymers;
- environmental effects (e.g. changes in temperature and viscosity); and
- the changes in the state of particles (e.g. protein conformational and energy changes) where they affect the above.

In the following sections the details of the physical chemistry required for each of these steps will be examined.

Particle Movement: Diffusion

The objects/proteins are modelled as moving in a continuous sea of solvent. The objects modelled are usually much larger (perhaps 10,000 times) than the solvent particles around them, and over the course of even one program cycle, will usually interact with many millions of solvent particles.

A fundamental question for our modelling is at what level of accuracy should these molecules be simulated. A complete solution would try to model everything, but of course this is always going to be impractical. The problem is to know where to stop - should the model work at the molecular, the atomic, or even the subatomic level? What level of accuracy is sufficient? The answer is likely to be determined by what level is practicable (i.e. how big the computer is) and what level is absolutely necessary (i.e. at what stage of approximation do the results become meaningless). If these two requirements do not overlap, it may be found that the problem cannot be modelled at all.

One approach, as described in Chapter 3, is to model all the molecules in a liquid. The velocities can be calculated for every molecule, and the collisions of every molecule modelled in the same way as is sometimes done when simulating a gas. In a liquid, however, the closer spacing of the molecules causes their continuous interaction through Van der Waals' forces.⁵³ This continuous interaction causes the molecules to follow more complex, non-linear paths, so this is not an accurate method. However, such a simulation can be done in time steps that are smaller than the average time between collisions, and these more complex paths approximated.

Since the same closer spacing of molecules leads to far more molecular collisions occurring over a given time within a given space, this requirement for small time steps places a great computational load on such a model, which can nonetheless be used to simulate small quantities of liquid, and thus derive various liquid properties.⁵⁴

A similar approach is to model every molecule and perturb them randomly, working out their final positions as probabilistic functions of the potential energy in the local region of each molecule, also described in Chapter 3.⁵⁵

At the scale of the simulations in this thesis, where thousands of self-assembling proteins are simulated, each of which is usually 10kD+ in size, to do this is completely infeasible. For example, a simulation of 10,000 40kD proteins, even at a reasonably high concentration, would still involve on the order of 10,000,000 solvent molecules. As a further complication, the simulation would have to be carried out in steps on the order of picoseconds, whereas many of the results of interest involve polymerisation over entire minutes. Obviously, if a way can be found to deal only with large protein molecules, and on a time scale of microseconds, the complexity of the problem can be reduced by up to twelve orders of magnitude.

Fortunately, it is not too difficult to do this. As outlined in Chapter 3, when a molecule in solution is much larger than the solvent molecules, its motion can be treated as a random walk within a continuous fluid. A random walk is simply the path taken by a particle when it has a great many random collisions. If these collisions are frequent enough they can, paradoxically, be ignored on an individual basis, and simply treated as an aggregate effect, since a particle undergoing a random walk tends to move in a Gaussian manner. That is, the probability of finding the particle at any given point is given by a normal curve: it is most likely to be found at its origin, and is approximately 65% likely to be found within one standard deviation of that origin.

In order to find the rules governing the motion of a single molecule, the large-scale (and measurable) properties of molecules in aggregate must be considered, in order to discover the exact form of the probabilistic normal curve governing the movement of the molecule*.

* The discussion here owes much to P.W. Atkins' excellent book, 'Physical Chemistry, 5th ed.' (1994), OUP, Oxford.

The Diffusion Equation in One Dimension

The diffusion equation describes the macroscopic properties of a solution, specifically that the concentration of a solution changes over time in proportion to the second derivative of the concentration gradient. This is to some extent intuitive: particles are expected to move from areas of high concentration to areas of low concentration, a process that acts, over time, to ‘smooth over’ initial irregularities in concentration .

$$\frac{dc_x}{dt} = D \frac{\partial^2 c_x}{\partial x^2} \quad \text{One-Dimensional Diffusion Equation} \quad (5.4)$$

D = Diffusion constant of molecule

c_x = concentration (in x-direction)

To illustrate, consider a solvent in a volume where, at $t=0$, all of the solute is on the plane $x=0$. (e.g. a bucket with solute molecules on the bottom, into which the solvent is poured at $t=0$, shown in Fig. 5.1.)

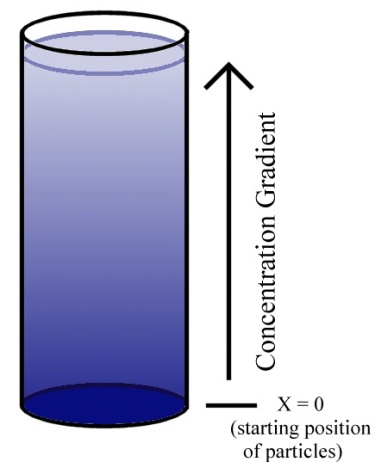


Figure 5.1: One dimensional concentration gradient.

The general solution to a second order partial differential equation of the type above

$$c_x = \frac{k}{\sqrt{Dt}} e^{-x^2/4Dt} \quad \text{Solution to One-Dimensional Diffusion Equation} \quad (5.5)$$

c_x = concentration (in x-direction)

D = Diffusion constant of molecule

k = a constant

Specifying the boundary conditions enables us to determine k . If the total number of solute particles is N , then k can be found by integrating the above expression for concentration (in the x-direction) over all "x" for a suitable t (this is allowable, since the number of solute particles is constant over all t). By then taking $t = 1/D$, the following convenient expression is found:

$$c_x = k e^{-x^2/4} \quad \text{Concentration Gradient at time } t = 1/D \quad (5.6)$$

Integrating this concentration function from $x=0$ to $x=\text{infinity}$ gives:

$$k = \frac{N}{\sqrt{\pi}} \quad \text{Solution for Constant in Diffusion Equation} \quad (5.7)$$

Thus, given the boundary conditions that all N particles at $t=0$ start at $x=0$, the one-dimensional diffusion equation gives the resulting concentration gradient for any positive time t as:

$$c_x = \frac{N}{\sqrt{\pi Dt}} e^{-x^2/4Dt} \quad \text{1-D Concentration Gradient Solution} \quad (5.8)$$

A Three-dimensional Solution

The general form of this equation can be kept, but expanded to three dimensions, by postulating a cross-sectional area A . This gives the 'bucket' a specific radius, and allows the volumetric concentration to be found by simply dividing the above by area. The concentration is assumed to be constant for any given x , varying only in the x direction. This gives the final form of the concentration function as:

$$c_x = \frac{N}{A\sqrt{\pi Dt}} e^{-x^2/4Dt} \quad \text{Linear 3-D Concentration Gradient} \quad (5.9)$$

c_x = "true" 3-D concentration in the x direction

N = the total number of particles

A = the cross-sectional area of the container

D = the diffusion constant

The Average Displacement of a Molecule

Care is needed when considering the motion of a diffusing particle, since it is a random walk. What is of primary interest is the average distance travelled over a period of time t , the 'mean displacement'.

This is different from the actual distance covered by the particle. Furthermore, the instantaneous velocity of the particle may be much greater than would appear from this ‘mean displacement’, as the particle bounces continuously from solvent molecule to solvent molecule. This is due to the fractal nature of Brownian motion, where the particle moves over a much greater distance than would be imagined merely by observing its start and end point over a period of time, as it continually backtracks and sidesteps, bouncing around at random (Fig. 5.2).

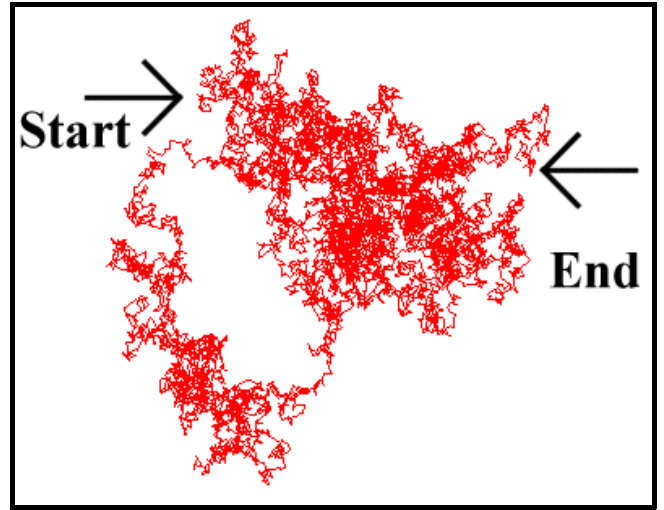


Figure 5.2: Brownian Motion (output is from the author's collision simulation program)

The mean distance travelled by a solute particle can be found by expanding the previous result. At $t=0$, all the particles are at $x=0$. At any later time, they are found spread over the entire range of $0 \leq x \leq \infty$. Hence, the mean distance travelled by all the particles from $t=0$ to some new time $t=T$ is simply the weighted sum of the concentration gradient. Considering the x -direction only, and using the one-dimensional concentration gradient:

$$c_x = \frac{N}{\sqrt{(\pi Dt)}} e^{-x^2/4Dt} \quad (\text{Eqn 5.8 repeated})$$

if $c_x \times x/N$ is integrated to find the weighted sum, this results in the mean distance travelled by all particles, and hence the average distance travelled by any given particle (from time $t=0$ to a new time T). This gives:

$$\begin{aligned} \langle x \rangle &= \int_0^\infty x c_x dx \\ &= \frac{1}{\sqrt{\pi Dt}} \int_0^\infty x e^{-x^2/4Dt} dx \\ &= 2 \left(\frac{Dt}{\pi} \right)^{\frac{1}{2}} \end{aligned} \quad \begin{array}{l} \text{Mean 1D distance travelled} \\ (5.10) \end{array}$$

The mean square distance

In a similar fashion, the one-dimensional mean square distance travelled can be found by integrating $x^2 c_x / N$. This is useful, as it can be generalised to the case where particles are diffusing in both directions from the origin.

$$\begin{aligned}
 \langle x^2 \rangle &= \int_0^\infty x^2 c_x dx \\
 &= \frac{1}{\sqrt{\pi Dt}} \int_0^\infty x^2 e^{-x^2/4Dt} dx \\
 &= \frac{1}{\sqrt{\pi Dt}} \times \left(\Gamma\left(\frac{3}{2}\right) \times \frac{(4Dt)^{3/2}}{2} \right) \\
 &= 2Dt
 \end{aligned}$$

1D Mean Square Distance (5 . 1 1)

This simple result is quite unsurprising: in actual fact it could have been read directly from the concentration function above, by making the substitution $t^2 = x^2/2Dt$ and comparing the result with the standard form of the normal curve: $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.

A true three-dimensional solution

Eqn. 5.11, but only for motion in one co-ordinate. To find the general formula for motion in all three dimensions, the concentration gradient given by an initial point source (say a pellet of solute dropped into a bucket) can be considered (Fig. 5.3).

At $t=0$, all N particles are at the origin ($x=0, y=0, z=0$). After this time, the particles spread throughout the entire space. As before, the solution to the diffusion equation given these boundary conditions is required.

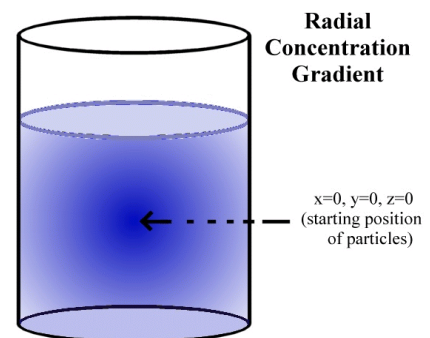


Figure 5.3: Three-dimensional concentration gradient

The diffusion equation in three dimensions is:

$$\frac{\partial c}{\partial t} = D \nabla^2 c = D \left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} + \frac{\partial^2 c}{\partial z^2} \right) \quad \text{3-D Diffusion Equation} \quad (5.12)$$

However, given the geometry of the situation, it will be better to give this in spherical polar co-ordinates, which gives the rather more complex-appearing equation:

$$\frac{\partial c}{\partial t} = \frac{D}{r^2} \left(\frac{\partial}{\partial r} \left(r^2 \frac{\partial c}{\partial r} \right) + \frac{1}{\sin^2 \theta} \frac{\partial^2 c}{\partial \phi^2} + \frac{1}{\sin \theta} \left(\sin \theta \frac{\partial c}{\partial \theta} \right) \right) \quad \text{3-D Polar Diffusion Eq. (5.13)}$$

However, because our concentration is expected to have spherical symmetry only the distance from the origin will matter, not the angle, so this can be simplified to:

$$\frac{\partial c}{\partial t} = \frac{D}{r^2} \frac{\partial}{\partial r} \left(r^2 \frac{\partial c}{\partial r} \right) \quad \text{3-D Polar Diffusion Eq. with Radial Symmetry} \quad (5.14)$$

The general solution to this spherical diffusion equation is:

$$c = \frac{k}{(Dt)^{3/2}} e^{-r^2/4Dt} \quad \text{3-D Radial Polar Diffusion Equation General Solution} \quad (5.15)$$

k = a constant

D = diffusion constant

r = radial distance from origin

t = time

To find the exact form of the equation, for a concentration of N particles, the procedure outlined above can be repeated. By choosing $t = 1/D$, the concentration gradient is obtained in the simplified form $c = k e^{-r^2/4}$. Integrating this over the entire volume, k is found as:

$$k = \frac{N}{8\pi} \quad \text{Solution for 'k' Constant for Eq.(5.15)}$$

This gives the final form for the three-dimensional concentration function for the boundary conditions that at $t=0$, $x=0$ for all N particles:

$$C = \frac{N}{(4\pi Dt)^{3/2}} e^{-r^2/4Dt} \quad \text{3-D Radially Symmetric Concentration function} \quad (5.16)$$

Mean square distance of a spherical concentration

Now that the concentration function for a spherical concentration is defined, the mean square distance can be found in the same manner as above, by integrating $x^2 c/N$ over the volume.

$$\begin{aligned} \langle r^2 \rangle &= \int_0^\infty r^2 C(r^2 \sin\phi) dr d\phi d\theta && \text{Mean Square Distance for Radial} \\ &= \left(\frac{1}{(4\pi Dt)^{3/2}} \right) \left(\Gamma\left(\frac{5}{2}\right) \times \frac{(4Dt)^{5/2}}{2} \right) (4\pi) && \text{Concentration} \quad (5.17) \\ &= 6Dt \end{aligned}$$

This is expected, since it is three times the mean square distance obtained earlier for the linear case. Hence the root mean square value, or standard deviation, of the motion in the 3-D case is the square root of three times the standard deviation in the linear case, which is what would be found by simple addition of vectors.

Accuracy of the Gaussian approximation

All these formulas rely on the Gaussian formula being an adequate approximation to the real random walk nature of the molecular movement. For the approximation to be valid, the times and distances between collisions of the solute particle with the solvent particles must be substantially less than the times and distances being used in our simulation.

The Einstein-Smoluchowski equation⁵⁶ provides a handy check that this is the case:

$$D = \frac{\lambda^2}{2\tau} \quad \text{Einstein-Smoluchowski Equation}$$

λ = the length of the average "jump" between collisions

τ = the average time difference between collisions

It is reasonable to assume that the average distance between collisions will be on the order of the size of a water molecule (say, around 250pm). In this case, for a very small protein with a diffusion constant of $D = 1 \times 10^{-10}$, the value of τ will be 312ps.

Hence, so long as the simulation operates on the order of milli- or micro- seconds, it is reasonable to expect that the particle will undergo thousands or millions of solvent collisions for each simulation step, and the Gaussian approximation will be valid.

Summary of Particle Movement and Diffusion

If the diffusion constant is known for a solute in a given solvent, the initial concentration profile of that solute, and the boundary conditions of the vessel enclosing the solute, the concentration gradient function for all time can be found using the diffusion equation.

Similarly, while considering a single particle, a probability distribution for possible positions in the particle's future can be determined from the same information. While the particle moves in a Brownian (or 'random walk') manner, at a time t in the future the probability of the particle being found at a given point is given by a Gaussian probability function, with a variance of $2Dt$ in each of the x,y,z co-ordinates, unless this is modified by boundary conditions.

This gives the standard deviation for a molecule's motion (the root mean square of the expected change in position over a time period 't') as being, in polar co-ordinates:

$$\sigma_r = \sqrt{6Dt} \quad \text{RMS radial change in position} \quad (5.18)$$

- The actual angle from the origin is entirely random.

Expressing these results in a different way, the standard deviation in linear co-ordinates can be written, where it is simply:

$$\sigma_x = \sigma_y = \sigma_z = \sqrt{2Dt} \quad \text{RMS Displacement from the Diffusion Eqn.} \quad (5.19)$$

Modelling all this activity is infeasible on current (2000) equipment, and is in any case unnecessary. The outcome of such a large number of interactions of a large particle with many solvent particles is well known, and is the famous ‘random walk’ of Brownian motion.⁵⁷

Particle Motion: Determining Diffusion Constants

In the previous section, it was shown how a particle’s diffusion constant can be used to determine its mobility. The diffusion constant of small, agile molecules has a greater numerical value than that of a large, slow-moving macromolecule. The diffusion constant provides an easy way of calculating the RMS distance travelled by molecules over a given period of time.

These diffusion constants can be most accurately obtained by laboratory measurement. There are a number of methods used to do this, such as the capillary and the diaphragm techniques⁵⁸. In the capillary technique a solute-filled capillary tube is dipped into a larger container of solvent, and the outflow of solute molecules is measured by monitoring the concentration in the capillary. The diaphragm technique works in a fundamentally similar fashion.

For the nanoscale simulator however, the situation is more complicated. While it is possible that the diffusion constant will be known for the free monomer, the diffusion constant varies with the size of the diffusing object, and it is unlikely that it will be known for the various sizes of polymer that the monomers form. But when the diffusion constant cannot be directly measured, a reasonably accurate theoretical approximation can be resorted to.

The Stokes-Einstein equation⁵⁹

The Stokes-Einstein equation describes the way that diffusion increases in proportion to temperature, and is inversely proportional to the frictional force experienced by a molecule:

$$D = \frac{kT}{f} \quad \text{Stokes-Einstein Equation} \quad (5.20)$$

D = diffusion constant

k = Boltzmann constant

T = temperature

f = frictional constant

As both the Boltzmann constant and the temperature are known, all that is required to determine the diffusion constant is to find the frictional constant f . Determining this exactly is again a matter for experimentation, but it can also be determined reasonably accurately on theoretical grounds for various polymer shapes and weights.

Stokes' relation⁶⁰

Stokes' relation gives us a theoretical value for the frictional co-efficient of a spherical particle in a solvent:

$$f = 6\pi r \eta \quad \text{Stokes' Relation} \quad (5.21)$$

r = particle radius

η = solvent viscosity

The difficulty here (and where inaccuracy arises) is that the particle radius is not simply the physical radius of the molecule, but is rather the *effective* radius of the molecule, taking into account the effects of solvation (solvation is the layer of ordered water that the electric field of a protein causes to form around the protein). Also, the formula is strictly only accurate only

for a single spherical particle. It can be modified easily to take account of prolate and oblate ellipsoids, but even these, like the sphere, are usually only approximations to the true shape of the particles.

Friction formulas for different shapes ⁶¹

The following formulas give the theoretical frictional constants for different shaped proteins, and are used by the nanosimulator to generate theoretical values when experimental values are unknown.

$$f = 6\pi\eta r$$

Sphere(5.22)

$$f = 6\pi\eta(a^2b)^{1/3} \left(\frac{(a^2/b^2 - 1)^{1/2}}{(a/b)^{2/3} \arctan[(a^2/b^2 - 1)^{1/2}]} \right)$$

Oblate Ellipsoid(5.23)

$$f = 6\pi\eta(ab^2)^{1/3} \left(\frac{(1 - b^2/a^2)^{1/2}}{(b/a)^{2/3} \ln \left[\left(1 + (1 - b^2/a^2)^{1/2} \right) / \left(\frac{b}{a} \right) \right]} \right)$$

Prolate Ellipsoid(5.24)

where

η = solvent viscosity

r = particle radius

a = half major axis

b = half minor axis

Calculating these constants within the model

While the nanoscale simulator allows users to directly input the diffusion constant of their molecule, if known, it will also try to calculate a value using formulas 5.22, 5.23 and 5.24 if the diffusion constant is unknown.

This is straightforward if there is only a single spherical particle being modelled, for which the Stokes' relation, as given above, can be used. However, frequently it is necessary to model molecules that are not even approximately spherical, in which case the simulator allows the entry of multiple spheres to represent the desired structure.

Unfortunately, a set of multiple spheres is neither a prolate nor an oblate ellipsoid, so an approximation is required. (It would be possible to numerically simulate the frictional coefficient of the "true" molecular shape, but that is beyond the scope of this thesis.)

An approximation developed by the author for the purpose of diffusivity calculations, which is easy to implement and fast to execute, is to represent the monomer or polymer by an ellipsoid that has the same moments of inertia as the set of spheres used by the model*. This works well for symmetrical arrangements, and less well for non-symmetrical ones. The moments of inertia (MOIs) are automatically calculated by the simulator when the molecular model for a protein is read in, using an average protein density to determine mass, or more accurately using a known diffusion constant for the individual protein. When a polymer is created, the MOI is calculated from the MOIs of the component monomers. This is then compared against the general formula for the moment of inertia of an ellipsoid:

$$\text{Moment of Inertia} = \frac{1}{5} M(a^2 + b^2) \quad \text{Ellipsoid Moment of Inertia} \quad (5.25)$$

where a specific MOI is taken along a given semi-axis, M is mass, and a and b are the perpendicular semi-axes. (A semi-axis is half of either the major axis or a minor axis).

* This reverses the commonly used system of physical chemists who infer the moment of inertia from diffusivity (cf Atkins, P.W., *Physical Chemistry*, 5th ed (1994), OUP, Oxford, chapter 23). This approximation simply uses the same equations in reverse, to produce diffusivity from a moment of inertia.

The model calculates the effective values of the major and minor axes from the pre-calculated MOI by reversing this formula, and then proceeds to calculate the frictional constant based on these figures, using eqn 5.22, 5.23 or 5.24 as appropriate. It may also optionally add a small, user-defined, amount to the effective size of the molecule, to represent the effect of solvation.

Diffusion: Some simple worked examples

1. A small molecule

Consider the molecule SO_4^{2-} .

It is approximately spherical and has an approximate hydro-dynamic radius

$210 \text{ pm} = 210 \times 10^{-12} \text{ m}$. The Viscosity of water = $0.891 \times 10^{-3} \text{ kg m}^{-1} \text{ s}^{-1}$ (at 300K)

Stokes' equation gives a frictional coefficient:

$$\begin{aligned} f &= 6 \times 3.14159 \times (210 \times 10^{-12}) \times (0.891 \times 10^{-3}) \\ &= 3.527 \times 10^{-12} \text{ kg s}^{-1} \end{aligned}$$

Using the Boltzmann constant, $k = 1.38 \times 10^{-23}$

and $T = 300 \text{ K}$, the Stokes-Einstein equation gives us:

$$\begin{aligned} D &= kT/f \\ &= (1.38 \times 10^{-23}) \times (300) / 3.527 \times 10^{-12} \\ &= 1.16 \times 10^{-9} \text{ m}^2 \text{ s}^{-1} \end{aligned}$$

This compares well with the experimentally determined value⁶² of $1.10 \times 10^{-9} \text{ m}^2 \text{ s}^{-1}$.

2. A small protein

For illustrative purposes only, consider the 68kD protein haemoglobin as being a sphere of radius 3.5nm.

Stokes' equation would give a frictional coefficient of $f = 5.88 \times 10^{-11} \text{ kg s}^{-1}$.

The Stokes-Einstein equation would then predict a diffusion constant of $D = 7.0 \times 10^{-11} \text{ m}^2 \text{ s}^{-1}$, which compares well with the experimentally determined value⁶³ of $D = 6.9 \times 10^{-11} \text{ m}^2 \text{ s}^{-1}$.

Summary

By using Stokes' relation (5.21) and the Stokes-Einstein equation (5.20), and by approximating the shape of monomers and polymers when experimental diffusion constants are not known, theoretically calculated diffusion constants can be determined, allowing the nanosimulator to approximate the diffusive movement of simulated particles.

Particle Movement: Simulation

The Brownian motion of individual particles causes the diffusive behaviour of mixed liquids. Using the diffusion constant D , the change in position of a particle after a time t is found to be probabilistically distributed over a normal curve, with standard deviation:

$$\sigma_x = \sigma_y = \sigma_z = \sqrt{2Dt} \quad \text{RMS Change in Cartesian Co-ordinates (Eq. 5.19)}$$

This allows the full characterisation of Brownian motion. It also allows the modelling of the very first aspect of a particle's behaviour: where it will move after a single time step.

When the program simulates the movement of an object, either a monomer or a polymer, it uses a Gaussian function for each of the x, y and z directions, with the standard deviations defined above, to determine where the object will appear after the next time step.

Rotation

Another aspect of movement to consider is that of rotation. For small particles moving large distances in comparison to their size, it can be assumed that their final orientation is essentially random. They will have been involved in so many collisions that they may face in any possible direction after a time step.

For larger objects, especially large polymers, the situation is not so simple. An entire microtubule, for instance, does not rotate randomly over the course of a microsecond.

The simulator handles this problem by calculating the moment of inertia for all objects, incrementally adjusting the moment of inertia for aggregates as objects join and break off. If the moment of inertia is sufficiently large, it calculates a non-random orientation, based on a heuristic that approximately models the rotational force the object experiences. Rotational behaviour is not strongly characterised in the simulation, but it does not need to be; as long as aggregates and objects rotate sufficiently to avoid artificial effects from unchanging orientations, the rotational behaviour should not strongly affect interactions.

The Rotational Approximation

The approximation used for rotational motion - which is presented without experimental proof, but which makes a useful heuristic for the simulation - is that the rotational behaviour of the object is energetically similar to the Brownian translational motion of the object (see Fig. 5.4). This is not presented as necessarily reflecting physical fact, but as a useful approximation. It is made by analogy with the quantum mechanical behaviour of much smaller molecules, where the equipartition principle⁶⁴ tells us that for a temperature sufficiently above a critical 'rotational temperature' the linear and rotational modes of a molecule will share an equal amount of the

molecule's energy. Since at the scale of the simulation the quantization of these modes is so fine as to form a virtual continuum, the assumption is made that the rotational and translational energies of the simulated molecules are identically distributed.

Using this assumption that the distribution of rotational energy of the macromolecule is equal to the translational energy of the macromolecule, a further assumption is made that the Brownian nature of the translational motion can be extended to the rotational motion of the macromolecule as illustrated in figure 5.4. This implies that the rotational movement of the macromolecule, rather than being a continuous rotation with a slowly changing angular momentum, is instead a series of random changes in angular direction, caused by the irregular impact of large numbers of solvent molecules.

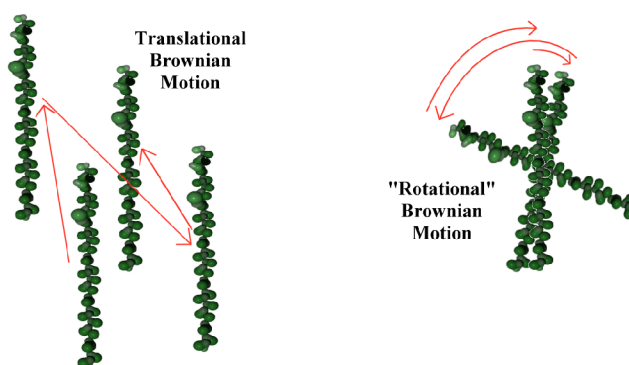


Figure 5.4: Translational and Rotational Brownian Motion

In order to model this, the square of the linear standard deviation for a given direction (remember that equation 5.19 states that this is equal in the x, y and z directions) is multiplied by the mass of the macromolecule, and the result equated to the angular standard deviation times the appropriate principal moments of inertia.

$$\begin{aligned}
 M\sigma_c^2 &= I_{xx} \times \theta_{\sigma x}^2 \\
 &= I_{yy} \times \theta_{\sigma y}^2 \\
 &= I_{zz} \times \theta_{\sigma z}^2
 \end{aligned}
 \quad \text{The Rotational Approximation} \quad (5.26)$$

where

M = the mass of the molecule

σ_c = the linear standard deviation in each of the x, y and z directions

I_{cc} = the three principal moments of inertia

θ_{oc} = the angular standard deviation around the three principal moments of inertia

The above approximation is derived from classical considerations; if the object were moving with a constant linear velocity v and angular velocity ω , its translational energy would be:

$$\text{translational energy: } \frac{1}{2} M v^2 = \frac{1}{2} M v_x^2 + \frac{1}{2} M v_y^2 + \frac{1}{2} M v_z^2 \quad \text{eqn} \quad (5.27)$$

$$\text{while its rotational energy is: } \frac{1}{2} I \omega^2 = \frac{1}{2} I_{xx} \omega_x^2 + \frac{1}{2} I_{yy} \omega_y^2 + \frac{1}{2} I_{zz} \omega_z^2 \quad \text{eqn} \quad (5.28)$$

Since in the simulation the linear displacement per unit time (corresponding conceptually to the v term above), is normally distributed with standard deviation σ_c , the angular displacement standard deviation θ_{oc} , is similarly treated as corresponding to ω . Using the ‘Rotational Approximation’ equation above (5.26) the three angular displacement standard deviations are found as:

$$\theta_{ox} = \sigma_x \sqrt{\frac{M}{I_{xx}}}, \theta_{oy} = \sigma_y \sqrt{\frac{M}{I_{yy}}}, \theta_{oz} = \sigma_z \sqrt{\frac{M}{I_{zz}}} \quad \text{Angular s.d.} \quad (5.29)$$

Using this approximation results in the expected behaviour: long, thin macromolecules will ‘roll’ easily around their major axis, but will not rotate so far around their minor axes, while spherical macromolecules will move with equal ease or difficulty in different angular directions. Due to the Brownian rotational approximation heuristic, the molecules will not smoothly rotate with a fixed angular velocity, but rather will adopt new angular positions varying from their old positions to a greater or lesser extent depending on their rotational inertia.

Although this behaviour is sufficient for the purpose of this thesis, it is emphasised again that this is a simple heuristic approximation, and should not be taken as an accurate simulation of the rotational motion of large molecules in a solvent.

Collisions

One of the fundamental data the simulator needs to determine is whether objects have collided or not. This turns out to be more complex than might at first be thought.

Unfortunately, it is not possible to discover whether particles collide simply by moving them independently, and checking whether their positions in the simulator overlap. There are a number of reasons for this. Firstly, if the standard deviation of their movement (i.e. the average distance they will probably move in a time step) is large compared to their size, the collision probability would be greatly underestimated. Since the particles do not move in straight lines (see Fig. 5.1), two particles which may well have interacted at some stage may end up quite widely separated. Simply checking their final positions says very little about the path they took to get there.

This difficulty could be resolved if very small time steps were used, only allowing the particles to move a fraction of their own size before again checking for collisions. However this too is unsatisfactory. To completely model the Brownian motion in this way, the time step should be dropped to the same time scale as the physical collisions which drive the Brownian motion - which would force simulation at the scale of picoseconds, or less. Otherwise there would still be a certain degree of 'fuzziness' as the object moved, even if it moved only a small proportion of the object's total size, because it would be continuously jiggling as it went. However, even assuming that the simulation is modelling a 10 nanometre sphere which is allowed to move (on average) only 1 nanometre at a time, there are still difficulties.

Equation 5.19 indicates that the net displacement of a particle over a given time period varies as the square root of that time period. Thus, to reduce this displacement by a factor of ten, the time step must be decreased by a factor of a hundred. This means that moving particles by a tiny amount (still an imperfect solution) would require vast amounts of computational time. For most of the systems examined in this thesis, the free monomers are moving around ten times their diameters in a time step. To reduce this to, say, one tenth of their diameters would require

running the program 10,000 times longer, which would greatly reduce its value. This is an example of the ‘scaling problem’, which is covered in more detail below.

Fortunately, this degree of effort is not required. If the diffusion constant and the size and position of two particles are known, it is sufficient to simply find the *probability* of them colliding. Thus, rather than simulating every step leading to their collision, the a priori probability of a collision is used, with the simulator determining randomly if the collision actually occurs.

This probability, while difficult to determine analytically, is comparatively simple to determine using a numerical simulation. This was done by the program described in Appendix A, and a table of scale-independent probabilities was calculated. This table of raw probabilities lists the probabilities of two spheres, of given total radius, colliding at a distance expressed as a multiple of their (weighted) combined RMS movement. Using this table, the nanoscale simulation program can determine the raw collision probabilities for different particles by scaling the results to suit the actual particles it needs to simulate at any particular moment.

The scaling works in the following manner: Consider two spheres A and B , with radii R_A and R_B , moving with RMS displacements per cycle of σ_A and σ_B , at a distance d apart. In order to collide, the centres of the two spheres must approach to within $R_A + R_B$. The problem can thus be simplified to that of a single sphere, of radius $R = R_A + R_B$, colliding with a moving point. If the point is viewed from the perspective of the newly ‘enlarged’ sphere A, it can be seen to have an apparent RMS displacement of $\sigma = (\sigma_A^2 + \sigma_B^2)^{1/2}$. The problem now has only three variables; the combined radius R , and the combined RMS displacement σ , and the distance d . The situation is illustrated below in figure 5.5.

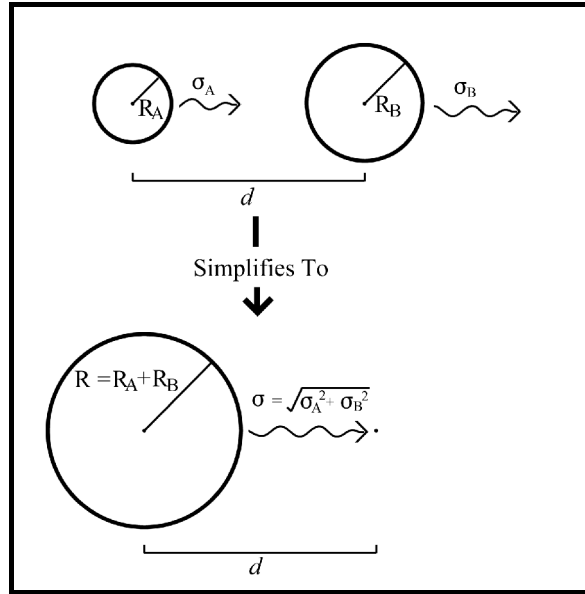


Figure 5.5: Simplifying Collision Geometry

If d and R are divided by σ , R' and d' can be obtained, as scale-independent measures of radius and distance, measured in terms of the standard deviation of displacement. The program from Appendix A tabulates the probabilities of collision for a wide range of such values, for combined radii and distances of 0.1 standard deviations, through to 10 standard deviations, plotted on logarithmic scales. The nanosimulator can quickly look up the probability of any two particles colliding, by calculating the scale-independent values of the combined radius and the distance, and consulting the table to determine the result.

Binding, and Binding Sites

Once the simulator has established that a collision has occurred, it checks to see whether binding occurs. This depends on there being available compatible binding sites on the colliding objects. If there are, and there is no physical impediment, a randomiser is used to generate a real number between 0 and 1, and this value is compared to the predetermined 'strengths' of the binding sites to see if binding occurs.

Determining these binding site 'strengths' is one of the more difficult aspects of this sort of simulation. Where possible, recourse can be made to direct experiment, but often these data have to be induced from overall reaction rates, or by examining the rate at which aggregates

grow. It may be possible in future to derive many of these from the structure of the molecules involved, but this is currently very difficult⁶⁵.

A useful method that has recently been investigated by Schwartz et al.⁶⁶ is to use the theoretical probability, based on the activation energy of association E_r and the activation energy of dissociation $-E_d$, with

$$\text{probability binding} = \frac{e^{E_r / k_b T}}{1 + e^{E_r / k_b T}} \quad (5.30)$$

$$\text{probability breaking} = \frac{1}{1 + e^{E_d / k_b T}} \quad (5.31)$$

k_b = the Boltzmann's constant,

T = temperature

These probabilities are then evaluated if the molecules are within tolerances. This method can be extended to the simulation presented in this thesis, however the probabilities must be adjusted with respect to the time scale of the simulation. (The simulation by Schwartz et al. deliberately uses an abstract time scale.⁶⁷)

Breakages

When all the movement, collision, and binding actions in a single time step have been determined, there is still work for the simulator to do. Many of the types of objects being modelled (especially the cytoskeletal proteins actin and tubulin) do not permanently polymerise - they form temporary bonds that may break. Thus, after each time step, objects that are already bound to aggregates must be checked to see whether they break away. The chance of breakage is usually derived from experimental data, and can usually be determined more easily than the chance of binding.

Many objects will be multiply bound in their aggregates however. So, rather than having a single link to the aggregate, a tubulin dimer in the middle of a microtubule may be bound to

other dimers on every side. It is difficult to determine exactly how the simulator should proceed in this eventuality.

For some polymers, such as actin (and probably also tubulin⁶⁸), it is very rare for a multiply bound monomer to break away. For such polymers, it is reasonable to set the probability for breakage of multiply bound polymers to a very low value.

One possibility is to assume that the breakage probabilities of the links are independent, and to require them to break at the same time (i.e. $\text{probability}(\text{breakage}) = \text{probability}(\text{link A breaks}) \times \text{probability}(\text{link B breaks})$). Since the probability of a breakage is often very low, (often on the order of 0.0001 per cycle), this makes the chance of double breakages very low, and multiple (three or more link) breakages close to impossible. A shortcoming of this method is that the probabilities of these multiple breakages then become dependent on the time scale of the simulation, which is physically unreasonable.

Another option is to simply disallow multiple breakages altogether, and set their probabilities to zero. This may be appropriate for some polymers.

The solution currently implemented is to use an *ad hoc* heuristic, where the chance of breakages is reduced for each additional link, in some reasonable manner. The heuristic currently (optionally) used in the program is to make the probability of breakage an order of magnitude lower than the lowest breakage probability for each extra link. For example, if a monomer is bound with three links, link A having a breakage probability of 0.003, while link B = 0.002, and link C = 0.0005, the probability of breakage would be 0.0005 (the smallest) reduced by two orders of magnitude to produce a final breakage probability of 0.000005. This heuristic has the advantage of being time-independent, and still allowing multiply bound polymers to break off, albeit with a greatly reduced chance (corresponding to what some researchers observe experimentally, although this is still a subject of some debate^{69,70}).

Future Work

Another solution, not currently implemented, would be to allow users to specify the breakage probabilities for all possible linkage combinations. Although this would provide the most flexibility, in all but the simplest cases it would require a considerable amount of work for the user to specify all probabilities for all possible linkages.

Possibly the best solution would be to use a more sophisticated binding model, where the linkages were treated in terms of the energy differences involved, and the association and disassociation constants derived from these numbers. This method has been used to create detailed simulations of microtubule growth tips with some success⁷¹, as well as the recent work with virus assembly already mentioned⁷².

Events and State Changes

One of the complexities of modelling biological molecules, especially proteins, is the number of different states they have. A protein may be in a high-energy or a low-energy state (often depending on whether it has been phosphorylated), it may act differently when it is bound to another protein than when it is floating free, it may even change its conformation.

What state a protein is in largely depends on its history, and its immediate environment. The simulator models this by defining a small number of ‘events’ that may alter the protein's status. The most obvious events are the binding of a particular site, or the breaking of a link from a particular site. A more subtle ‘event’ that the simulator defines is the ‘random event’, which can be used to model a stochastic change from an unstable intermediate state. Thus a bound protein may ‘decay’ from a high-energy to a low-energy state after binding.

An example of such a random event would be the process postulated in one of the theories of microtubule formation⁷³, when a free tri-phosphorylated tubulin dimer is bound to the growing microtubule. Under this theory, the bound dimer decays to a di-phosphorylated state after a

short period of time. Another theory postulates the change almost as soon as the dimer is bound at a particular site⁷⁴.

The simulator keeps track of the state of each object, and checks whenever there is an event that may change that state. When the state changes, various other attributes of the object, such as binding and breaking strengths, may change as well.

Boundary Conditions

A perennial problem with physical simulation is how to handle the boundaries of the volume it being simulated. There are two main approaches. The first is to make the boundaries an impenetrable, elastic wall, which reflects particles that attempt to move past it. The second is to use 'periodic' or 'tessellated' boundary conditions, wherein particles moving off one edge of the simulation field immediately re-enter on the other side. Geometrically, this second approach corresponds to folding the simulation space into a four-dimensional 'hyper-torus'.

The fixed, impenetrable boundary method has many computational advantages, since it avoids complexities inherent in the periodic boundary conditions. However, while it may be legitimate to use this for individual particles, where particles reflected from the wall can be thought of as corresponding to new particles entering the system while old particles leave, it fails for the larger aggregates. The difficulty is that large aggregates, which could theoretically be a substantial fraction of the size of the simulation field, will be in effect forced towards the centre of the simulation field. A further difficulty is the impossibility of an aggregate growing to a larger size than the simulation field.

Periodic boundary conditions allow for the simulation field to be effectively infinite. A particle 'perceives' an infinite, but tessellated, simulation field stretching in all directions. Without any barriers, it is free to move in any direction, as can an aggregate. Further, an aggregate can grow to a length greater than the simulation field by simply 'wrapping around' the simulation field, growing out one side and re-emerging at the other.

Periodic boundary conditions impose a larger burden on the programmer, since complex handling of interactions between particles on different edges of the simulation field must be performed. However it results in a better, less constrained model of the environment.

A difficulty for the nanoscale simulation program in using tessellated boundary conditions is that of *self-interaction*. A sufficiently large object (i.e. a very long actin filament) could grow to the size of the simulation field. Once this happens there is a very good chance that it will interact with itself, possibly binding head to tail to form a closed (effectively four-dimensional!) loop. In any of the simulations involving proteins such as actin or tubulin that undergo growth and decay cycles, such an object, lacking a beginning or an end (the locations where dis-assembly usually occur), would be hyper-stable, which would greatly distort the results of the simulation.

In order to avoid this, the simulation field must either be of such a size that this cannot occur (either small and lacking sufficient monomers to build such a large polymer, or large enough that the probability of a polymer that large being created can be ignored), or the results of the simulation must be monitored for signs that a polymer of this size was created. In practice a polymer of the same size, or larger, than the simulation field is rarely observed.

The Scaling Problem

There is a limit imposed by computer hardware as to how many particles can (or should) be modelled, over what volume, and for how long. Ideally the ‘huge’ quantities used in real laboratory experiments would be used. But since a single mole of a chemical represents 6.02×10^{23} particles, it can be seen that this is infeasible given the cost of computer technology in 2000. In fact, it turns out that the largest number of molecules practically handled on a desktop workstation is around 100,000.

By choosing the number of particles, and choosing the molarity to simulate, the volume of the simulation is determined. For 100,000 particles at a molarity of 20 micromolar, a volume of around about 1000 nanometres cubed, or 1 cubic micrometer would be required. This

represents about the largest reasonable simulation: 100,000 molecules, in a volume of 1 cubic micrometer.

Naturally this excludes solvent particles, which are assumed to be much smaller, and far more numerous, than the large solute objects. In fact, in the simulation above, on the order of 500,000,000 water molecules are included in the volume, but as explained previously they do not need to be explicitly modelled.

The simulation program runs in cycles, each of which lasts a certain amount of time, known as the time step. Therefore, in addition to the above decision, a suitable time step needs to be determined for the program. At this stage another difficulty appears, as a result of the RMS movement equation (5.19) that was derived from the diffusion equation.

The RMS movement equation describes the root mean square displacement (a measure of the average distance travelled) moved by a particle in time t in any given direction. This value is the square root of twice the diffusion constant times t .

To see how this works in practice, consider a time step of 10 milliseconds, which might at first appear reasonable. For a particular protein of around 100kD size, it may turn out that the rough scale of movement would be on the order of a micrometer in 10 milliseconds. However this is also the size of the largest practical simulation volume, and implies that all the protein particles will be moving over the entire region in every time step.

This is unacceptable for two reasons. Firstly, it would be computationally very inefficient to have to check all $(100,000 \times 100,000)/2$ possible interactions of all the objects. But more importantly, it would completely defeat the purpose of simulation, which is to examine the small-scale behaviour of these proteins.

Since the RMS distance travelled decreases as the square root of the time step, the time step must be decreased. To obtain a reasonable-sized average displacement using the above example, a time step of around 1 microsecond might be necessary, which would give us an RMS distance of around 10 nanometres. At 10 nanometres, each object is likely to encounter, on average, less than one other object, which will allow for fine-scale simulation.

But this is also unsatisfactory. Many chemical reactions of interest have features that appear only over the course of entire minutes - and if 1,000,000 cycles per simulated *second* are required, the program will take an unacceptably long time to run.

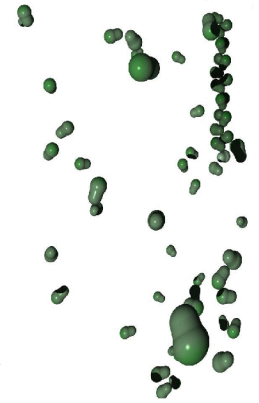
This turns out to be a fundamental limitation of the program. It may be partially traded off by using fewer objects, but even so, the excessive number of time steps that need to be modelled make it difficult to completely simulate *in vitro* experiments.

By using fast algorithms and powerful computers, a reasonable simulation period and space can still be studied, but this limitation does restrict, for the time being, the range and extent of the simulation. Since computer hardware performance appears to increase exponentially over time (the observation called 'Moore's Law') this restriction should cease to be relevant comparatively soon..

Summary

In essence, the simulator treats its objects (usually proteins) as small, partially sticky balls that bounce around within a virtual liquid, occasionally colliding and binding together. The movement of these objects is governed by the laws of physical chemistry, primarily using the diffusion equation. The aggregated groups occasionally fall apart, or collide with other sticky balls, and possibly grow.

Using this simple idea (which turns out to be considerably more complex to implement - see Chapter 6) the way small particles, and in particular proteins, interact may be studied. Chapters 7 through 10 describe specific studies of systems such as the self-assembly of actin and tubulin into large structures, and the deposition of stain particles onto the samples used in electron microscopy.



Chapter 6: The Nanosimulator

'Now let me see' the Golux said. 'If you can touch the clocks and never start them, then you can start the clocks and never touch them. That's logic, as I know and use it. Hold your hand this far away. Now that far. Closer! Now a little farther back. A little farther. There! I think you have it! Do not move!'

The clogged and rigid works of the clock began to whirl. They heard a tick, and then a ticking...

The 13 Clocks, James Thurber

Introduction: The 'Functional Requirements' of the Program

Writing a program to implement the behaviours described in the previous section is a complex undertaking. While modelling the physical processes involved can be difficult, a surprising amount of effort is taken up with the more mundane programming tasks of organising data, establishing the three-dimensional geometry of the simulation volume, and coping with a large number of 'special cases', mainly involving boundary conditions.

Technical Requirements

The simulation program must be able to handle many thousands of objects, which are not necessarily all of the same type. Each individual object (usually a protein monomer) moves according to complex rules, and may interact with other objects in a variety of ways. Further, many of these objects form aggregations, for which various derived properties must be calculated as the program operates. Objects bind and break apart, change their state depending

on events they experience, interact with the edge of the simulation field, and all the while the laws of physics must be accurately simulated.

The program must be able to do this repeatedly, for many billions of calculations, without breaking down or allowing data to become corrupted by cumulative errors. It must be able to run fast enough to simulate long enough periods of time so that results can be compared with experimental data.

User Requirements

In addition to actually running successfully, the program must be able to read in complex description data at the start of its operation that fully describe the starting conditions of the simulation. Those data must be organised in such a way that human users can create and modify the descriptions of objects and their environment in a meaningful and reasonably straight forward way.

The data produced as the simulation runs must be conveyed to the user. This can take the form of numerical values and general statistics on the population of simulated objects, but ideally extends to real-time graphical viewing of the simulation.

Other desirable features of the program include:

- 3-D-modelling capabilities for importing or constructing complex initial conditions containing pre-existing structures;
- the ability to export detailed 3-D models of the state of the simulation at various times for use by other programs (such as raytracers and other modellers)
- the ability to display the environment to the user in different ways, such as viewing only bound objects, or linkage sites, or being able to ‘freeze’ motion and examine a particular stage of assembly in detail; and
- the ability to view the environment in ways imitating scientific instruments, i.e.using a simulated light microscope, or more radically, a simulated electron microscope (‘virtual electron microscopy’).

Since these functions do not need to be tightly integrated into the main simulator, some of them may be implemented as separate, smaller programs that share data between themselves.

Architecture of the Nanoscale Simulator

The problem domain consists of a small number of physical object types with a large set of behaviours and interactions, the management of these objects, and the need for program input/output to communicate with the human user .

An object-oriented programming methodology⁷⁵ was used to divide the problem domain into a number of classes, which can be conceptualized as *physical object classes*, *data containers*, *user input classes*, *user output classes*, and *mathematical utilities*. The various objects of these classes are largely managed by an overall *co-ordinating class*, which runs the main loop through which the program cycles; moving objects, checking for interactions, binding and breaking, saving data, and then repeating.

In the following sections the various components of the simulator and their operation together are described. This chapter covers the important features of the design and implementation of the nanoscale simulator.

Program Objects: The Pieces

The first step to understanding how the program works is to examine the classes of objects used by the program. A number of classes are used to represent simulated physical objects with properties such as mass and inertia, data structures to keep track of these objects, various I/O (input/output) classes, and a small number of utility classes, such as sets of mathematical functions.

The code for these classes, which comprise the nanosimulator program, is online at <http://www.cs.monash.edu.au/~cbetts/phd/code/index.html>.

Physical Objects

The physical objects are, as usual, the easiest to set within the object-oriented framework. The simulation deals with the motions of a large number of particles, generally representing proteins floating in a solvent. Therefore a 'physical object' class is needed, to represent objects floating in solvents.

This generic 'physical object' class, is not useful enough on its own - we need to add details about interactions. So two further classes are derived - a 'monomer' class, and a 'polymer' class.

Physical objects have:

- a central point;
- mass;
- moment of inertia; and
- a diffusion constant.

Monomers

The monomer class describes base objects from which the polymers grow. In addition to the physical parameters defined above, a monomer has a number of other properties. It has more detailed geometric shape information, its shape being described as a number of differently sized spheres. It has linkage sites where other monomers might bind, and detail on whether such linkages exist. Monomers also implement a simple user-defined state machine, which enables their characteristics to change during the polymerisation cycle. For example, some proteins change conformation when they are bound, and this new shape may have different binding characteristics than the previous form. The monomer state can change when certain links bind or break, or by random chance (to simulate stochastic decay processes).

Monomers have:

- all the attributes of ‘Physical Objects’;
- links to other monomers;
- a changeable state; and
- a link to a (possible) parent polymer that they are part of.

Monomer Helper Classes: Sites, Links, States, Spheres and Events

These classes are used by monomers to describe their interactions with each other.

A *site* is the geometric position, in the local co-ordinates of the monomer, at which it may be bound to another monomer. In addition to position it has a name, which is used both by the parsing program described below, and during debugging.

A *link* is the particular type of binding that exists for a particular state, at a particular site. It has a probability for binding to other monomers, and a probability for breaking from the same. If the monomer is actually bound, the link records which other monomer it is bound to.

The *state* of a monomer describes the overall status of the monomer. A change in state affects the links of a monomer - for example, when a protein dephosphorylates and enters a low-energy state, its links may be more inclined to break. As a visual cue to the user, the monomer can be made to change colour when its state changes.

The *spheres* of a monomer describe the rough geometry of the monomer. A long, slender protein can be approximated as a line of such spheres, whereas a flat, plate-like protein might be approximated by a small 2D array of spheres. They are used in collision calculations, and in calculations of physical properties such as moments of inertia and diffusion constants (when these are not supplied by the user). Spheres can be independently coloured, allowing changes in state to be graphically displayed.

(N.b.: sometimes in display these spheres are represented as merging into each other in a fluid manner, such as in the pictures of the ‘monomer’ in figure 6.1 - this is merely for display, mathematically the object is still a pair of spheres).

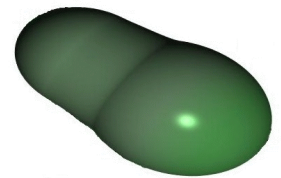


Figure 6.1:
Display Monomer

Events are defined actions that can change the state of the monomer. The three types of event are the *making* of a link with another monomer, the *breaking* of a link with a monomer, and finally a random event that can occur when the monomer enters an unstable state.

As an example of the above, consider the ‘+’ end of a tubulin ‘dimer’ protein (the ‘+’ site⁷⁶) binding with the ‘-’ end of another tubulin protein. They are joined together, each dimer having a different *link* object on their respective linkage *sites*. Each *link* object keeps track of the dimer linked to - so the link for Dimer A knows it is attached to Dimer B, while Dimer B’s link records that it is attached to Dimer A.

When the linkage is made, the *link* object is checked to see what *events* might occur. An *event* is a state transition - so the linkage of the two dimers might trigger a change in the *state* of one of the dimers, between a *state* with the name ‘GTP-tubulin unbound’ to another with the name ‘GTP-tubulin bound’. This particular dimer is then displayed to the user in the graphical output window as two *spheres* that have now changed colour, mirroring the *state* change.

Polymers

When monomers come together, a polymer object is created (Fig. 6.2). The polymer object is represented by a data store that keeps track of its constituent monomers, as well as summing their physical properties to arrive at overall details of mass, moments of inertia and diffusivity.

The polymer is responsible for moving and rotating its constituent monomers in an orderly manner, maintaining each monomer in position relative to its neighbours. Polymers must be able to gain and shed monomers, and be able to merge with other polymers when required.

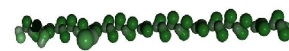


Figure 6.2: Display Polymer

Polymers have:

- all the attributes of ‘physical objects’; and
- a list of component monomers.

A Note on Nomenclature

“Monomer” and “polymer” are used in the program as functional descriptions. The monomer is the basic unit from which the top-level polymer is made. For example, when simulating tubulin, the tubulin dimer is considered by the program to be the monomeric building block, even though the tubulin dimer is itself a polymer of two tubulin sub-units, and these sub-units are themselves complex polymers made up of hundreds of amino-acid molecules.

It is even possible that the program may simulate an inorganic objects such as stain particles, which do not, strictly speaking, polymerise. However in the context of the program fundamental unit is still called the “monomer”, and the aggregate (i.e. a clump of metallic stain) a “polymer”.

The Physical Environment

Other simulated structures that must be represented are the solvent and environmental conditions such as the temperature. While a case could be made for making these classes in their own right, it was decided to implicitly model the solvent within the behavioural classes described below, and to keep any other required information (such as the temperature of the system) as general purpose information within the co-ordinating ‘overall program’ class.

The 3-D Data Structure

An efficient program must balance memory requirements with speed requirements. For the simulator, the memory and speed requirements are that a single monomer requires less than a kilobyte to fully describe its state, and simulating the motion and interactions of a single monomer during a single 10 microsecond timestep takes on the order of 100 microseconds (on midrange computing equipment in 2000). At the beginning of this chapter we mentioned that the simulation size should be of the order of thousands of molecules over many seconds of simulated time. Examining the computing requirements shows that the memory usage will be of megabytes, while the program will require a considerable length of time to run, since 10,000 monomers over a simulated minute might require weeks (on a 1998 workstation).

Since the memory requirements of the program are quite light, while the CPU cost is heavy, data structures that allow fast access are chosen whenever possible, even if this requires increased memory usage. This trade-off (memory for time) is made throughout the program.

An example of this trade-off is the data structure used to organise the monomers in their virtual space. A 3-D ‘data bucket’ system is used to keep track of the monomers, and to access them when required in a time-efficient fashion.

The system divides the simulation space into a three-dimensional Cartesian grid of boxes. After it is moved, each monomer is assigned to a particular box based on its position. In order to decide whether the monomer is able to interact with another monomer, only those monomers in adjoining boxes need to be searched (providing that the ‘probable interaction range’ is not

further than one box width). With a grid of boxes 20 x 20 x 20, this means that only other monomers in the test monomer's starting box and in the near neighbourhood (27 boxes in all) need be considered, rather than testing against all monomers in the simulation (e.g. 8,000 boxes), giving a substantial saving in computation, at the expense of maintaining the large data structure.

Data Input and Program Initialisation

In order to make the program flexible, a great deal of detail about the behaviour of the program, the precise nature of the monomers and of the simulation environment is provided via command line parameters and an initialization file.

At the command line, when the program is invoked, details such as whether to run the program in debugging mode, whether to alter the physical model of the simulation (e.g. disallowing numerically expensive operations such as polymer rotation), and how frequently to output data, are all configurable as command line options, along with many other details to do with the programmatic side of simulation.

The behaviour of the monomers is specified in detail using a 'Protein Dynamic Definition File' such as 'actin.pddf' or 'tubulin.pddf'. These files describe the shape and mass of the monomer, its binding sites, the states through which the monomer passes as it is bound and freed, and the likelihood of the monomer binding with other monomers of the same sort as itself, and with any other chemical species present (see Appendix C, and examples in Appendix D-G).

These initialisation files may include multiple chemical species. For example, a single simulation might include actin, a number of actin-associated proteins, and a number of stain particles. In addition to detail about individual monomer species, the .pddf files include the environmental conditions for which their data are valid, such as the viscosity and temperature at which the measurements were made.

The parsing of command line parameters is a trivial exercise that is done as a minor task by the 'co-ordinating program' class. The parsing of the .pddf data files however is far more complex,

and is handled by a number of classes. The parser reads the raw text, and based on what it reads initialises a data model object for each protein species it comes across. The parser then updates the data model with a number of small utility objects which describe the components of the data model, such as:

- spheres used to describe the geometry of the protein;
- sites which describe the location of binding sites;
- links giving the probability of binding and breaking away from other objects;
- states describing the changes in the protein as links bind and break; and
- events describing the actions that cause changes in state, such as bindings, breakings, or stochastic decay processes.

Once the data model has been set up, it is referred to throughout the life of the program, as a base template for object data, and also for the names used to describe objects, which are used to format user output data.

When the parser has finished and the data model is initialised, the model is used as a template for newly created monomers within the program. Some of the information is copied to the individual monomers, but other less frequently used information (such as a user-assigned monomer name) is simply accessed via a link from the monomer to its data model. In this way data model objects are somewhat similar to SmallTalk metaclasses⁷⁷.

User Output

The entire simulation will be useless if the data it produces cannot be accessed by the user. Hence data output is an important feature of the program, and a number of different strategies are employed.

Firstly, and most importantly, the program saves a data log that gives details about the state of the monomers and polymers in the simulation. This allows the construction of mass histograms showing the size distribution of polymers, as well as polymerisation curves showing the rate at which polymerisation occurs. These data logs are written at regular, user-determined, intervals,

and can be obtained even when there is no graphical output available (for example if the program is being run remotely, or on a non-graphical machine).

Secondly, the program can save geometric information suitable for use by other programs such as ray-tracers. This information also allows users to examine in detail the geometry of the polymeric structures. Again, this can be done without graphical output. Some extra functionality is also available when saving geometric information. By setting switches within the program and recompiling, it is possible to output the linkage sites of objects (displayed as small three dimensional arrows). It is also possible to output the data in such a way that free monomers are represented in a ray tracer as transparent or translucent.

Thirdly, a less quantitative method of data logging is a graphical window displaying the operation of the simulation in real time. This is a useful tool for tracking both gross errors in the program and (more importantly) for errors in the .pddf file. If the user has incorrectly specified the angles and positions of the binding sites, for example, it will quickly become apparent when the resultant warped polymers are graphically viewed. This graphical output is interactive, and allows the user to change their viewing position, freeze the simulation, change perspective, and even to make free monomers invisible in order to more clearly view the created polymers.

This functionality comes at a price. Despite using a number of graphical optimisations, it generally takes about as long to render a scene as it takes to simulate a cycle of the program. To minimize the impact of this overhead usually only every one-hundredth or one-thousandth scene is viewed. However, it is possible to view every cycle, in which case the program runs approximately twice as slowly, at a frame rate of around 10 frames/second for small scenes.

The final data output method is the debugging mode. When this mode is enabled a vast stream of information about the activity of the program becomes available, with numerical details of every attempted linkage, and every binding and breaking event, being written to the console.

In summary, the main data output methods of the program are:

- a numerical data log

- geometric data logs
- a real-time graphical viewer, and
- a verbose stream of internal program data generally used for program maintenance.

Mathematical Classes

Since the simulation is mathematically complex, a number of utility classes are used to simplify the tasks of vector and matrix arithmetic. Using the C++ facility to overload operators, it becomes possible to write many vector and matrix equations in an intuitive form that considerably simplifies the appearance of code.

There are two utility classes used in the program: a point class and a matrix class. They perform vector and matrix arithmetic, and allow expressions such as $X=A+B$, $X=A-B$, $X=A*B$ and $X=A^B$ where the '+' and '-' operators are normal addition and subtraction, while the '*' operator is used for vector cross product and, somewhat counter intuitively, the '^' operator is used for the vector dot product (for programmatic reasons it is impossible in C++ to use the '.' character for this purpose). These functions are defined for various appropriate combinations of scalars, vectors and matrices. Other vector operations are defined to allow various common tasks such as rotation around a fixed axis, the calculation of magnitudes, division by reals, and so on.

Finally, an external mathematical function provided by Professor Chris Wallace of the Department of Computer Science and Software Engineering at Monash University is used to generate fast pseudo-random Gaussian numbers which are widely used throughout the program.

The Co-ordinator Class: The Structure into which Everything fits

An overseer is necessary to co-ordinate the activities of all these separate classes. This co-ordinating object (there is only one) interprets the command line arguments of the user, starts the parsing of the .pddf initialisation parameter file, and initialises the required monomers to start the simulation.

Once this initial housekeeping is finished, the co-ordinator runs the program loop, cycling through the sequence of movement, binding and breaking. During this process it calls the objects responsible for visual output, and directly logs numerical data to a data log file.

Initialisation: Setting Up the Game

Before the program can simulate an environment, it must know what the environmental conditions are, and what proteins (or generic objects) it is simulating. While the general rules of behaviour for objects, such as their Brownian motion behaviour and the way they join together, are immutable and held in program code, exact details such as the size of the simulation field, the number of monomers in the simulation, and the exact properties of those monomers are set by configuration files and command line parameters.

Once the relevant information has been obtained by the program, it must correctly initialise the simulation's data structures, placing monomers in their starting positions, initialising the grid of 3-D 'cells' in which the monomers move, initialising the mathematical look-up tables used for collision calculations, and generally 'setting up the pieces' for the simulation.

Reading Data from the Command Line

Some features of the simulation are independent of the protein data files, and should be easily changeable by the user. These are handled at the command line, and include details such as the molarity of the solution (i.e. the number of objects to be simulated), the size of the simulation field in nanometres, and details of what data to output and how often. Most importantly, it is at the command line that one specifies which .pddf file to use.

It is not, strictly speaking, necessary to explicitly specify all these details. The program uses default values for any unspecified parameter, which is useful for rapid testing and prototyping of .pddf files. These default values can easily be changed within the program but, as an example, typical values are:

- a concentration of 5 μ molar;
- a total field size of 256 nanometres cubed; and
- the .pddf file 'info.pddf'.

Reading Data from the .pddf file

The amount of data necessary to describe the behaviour of a protein monomer are far greater than can be reasonably entered at the command line. As a result, small text files that can be written by human users, are used instead (see 'Appendix C; Protein Dynamic Description Files' for a full language description.)

These 'protein dynamic description files' or '.pddf files' contain all the data necessary to describe (in the simulation) the dynamic behaviour of the monomers. Details such as the general shape of the monomer, the binding sites, states of the monomer, and the events that trigger those states must be listed. Optional details such as the mass and diffusion constants of the protein can also be included, although the program will estimate these if they are unknown or not provided.

A portion of a .pddf file:

```
Model Stain
{
    Sphere stain_centre 1 <0,0,0>

    Site binding_site <0,0,1>

    State unbound blue

    State bound red

    Event bind { binding_site }
                unbound -> bound

    Event break { binding_site }
                bound -> unbound
}

Model Actin
{
    ...
}

Binding Stain binding_site unbound to Actin bottom ATPunbound
        bind=0.01 break=0.001
...

Temperature 310 # kelvin - physiological temperature
Viscosity 0.69 # water viscosity at 37 degrees Celsius
TimeScale 10 # microseconds per program cycle
Mix Actin 90% Stain 10% # the mix of molecules in solution.
```

The program parses these .pddf files, storing the data in internal data structures that are similar, but not identical to, the elements described in the .pddf files.

In order to build these data structures, the parser works through the .pddf file, recording data in a fixed sequence (Fig.6.3): first it reads the data for each individual protein species, then it reads data about the binding linkages, and their chance of breaking, between monomers. Next it reads the environmental variables (which specify details such as the temperature and viscosity of the simulation environment), the timestep for each simulation cycle, and finally the mixture ratio for the different protein species (if multiple types of monomers are present).

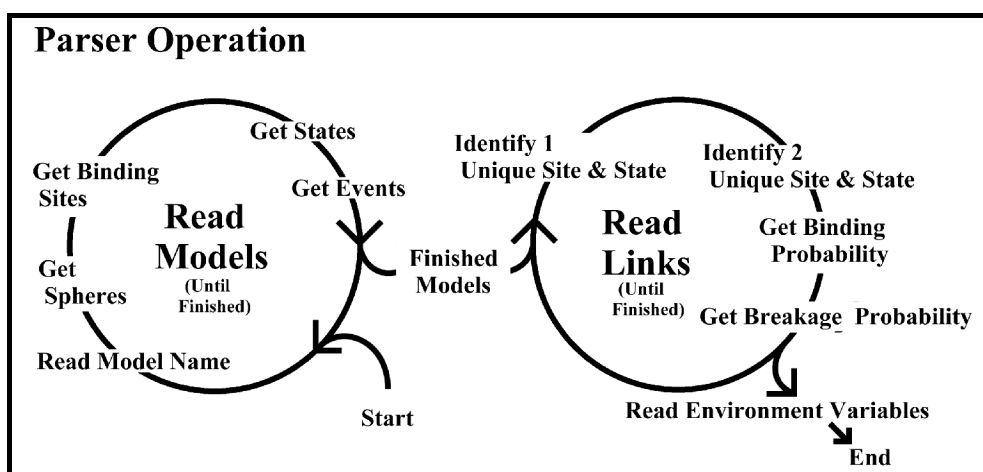


Figure 6.3: The Parser Operation Cycle.

Initialising the Environment

Once this data set has been read into the program, the co-ordinator works out details such as how many monomers should be created and how large the simulation field should be.

Next the 3-D data grid is initialised with the correct field size - this involves initialising each grid cell with the correct 3-D co-ordinates.

Then the monomers are created, and given (uniformly distributed) random positions within the simulation field. They start the simulation in their initial state, with no linkages to other monomers. The temperature and viscosity details have already been used to calculate the diffusivity of the monomers if this was not explicitly stated, and this is converted to a 'root mean square position delta' which describes how far the monomer moves, on average, each time step. (See Chapter 5 for details of this calculation).

Finally, environmental conditions are recorded explicitly and independently of the various monomer species, for use in later calculating the diffusivity of the various polymers.

Special Conditions and Work in Progress

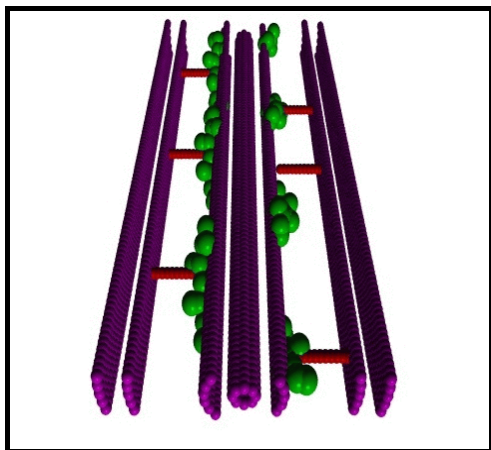


Figure 6.4:
A pre-defined static structure

An addition to the functionality of the program is the ability to define starting positions for the monomers. Among other benefits, this allows simulation of diffusion from high-concentration regions to low-concentration regions. An obvious extension of this procedure would be to allow complete simulation states, including linked polymers, to be read in as starting conditions, but this has not yet been completed.

As an addition to the functionality of the program, it is possible to include static objects in the simulation (e.g. Fig. 6.4), thus allowing the simulation of particles moving through restricted areas, such as channels, holes, or simply past large structures. This is done by allowing the user to define monomer species with a diffusivity of zero, i.e. which do not move.

The program treats these as special cases, and fixes them immutably in space. The actual objects can then be read in using the above method of pre-defined positions, after which they can interact normally with other, freely moving, objects (Fig. 6.5).

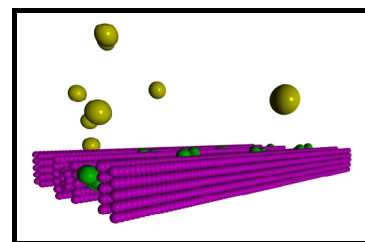


Figure 6.5:
Monomers and a static structure interacting

Program Actions: The Moves

The basic classes and objects give a static picture of the program. This next section describes their dynamic nature - how they behave within the simulation.

The Co-ordinator's Cycle:

At the top level, the co-ordinator works through the initialisation steps outlined above before entering a continuous cycle. But most of these initialisation functions are carried out by other objects; the only functions directly performed by the co-ordinator are those of initialising the virtual environment and logging numeric data.

Once the simulation is running normally, the co-ordinator's role is simply to prompt various objects into performing their actions on cue. Aside from this, the co-ordinator is also responsible for numeric data logging.

Moving Monomers and Dimers

Monomers and polymers are usually required to move during the simulation. This involves both linear displacement and rotational motion.

'Linear' Motion

'Linear' motion is used in the sense of the non-rotational change in position of objects - however, since they are moving in a Brownian manner, the actual path taken by the objects is likely to be far from linear. All objects, both monomers and polymers, have a diffusivity which enables the RMS change in position to be calculated (as described in the previous chapter). Because the particles are travelling in a Brownian manner with a very small 'step-length' their

full path need not be simulated. Instead the particles are moved directly to a randomly determined position using this RMS change in position.

Although the linear motion of large, non-spherical molecules (such as actin polymers) should be slightly different due to their irregular shape, the program currently does not handle this. Very large molecules in fact move very slowly on the scale of the simulation, so this inaccuracy has negligible effect.

The program *does*, however, take into account the approximate shape of the polymer when calculating the raw diffusion constant. It does this by approximating the shape of the polymer with an ellipsoid of the same moment of inertia, and using that ellipsoid in the diffusion constant equations outlined in the previous chapter. Hence, while strict directionality is not modelled, the overall diffusion behaviour is reasonably accurate.

After completing a movement cycle, the program checks that no particles are overlapping, and moves them apart to 'just touching' if this has occurred. This is done by considering the component spheres making up the overlapping objects (cf Fig. 6.1). In order of greatest overlap, each sphere is separated from its neighbour by moving their centres directly apart along the line connecting their centres, and the process is repeated if necessary until the objects are entirely separate. Algorithm 6.1. summarizes this process.

It is theoretically possible that this process could end in an infinite loop (although this has never been observed), so an error check is included as a fail-safe. If the process repeats more than 20 times, the position of one of the objects is set to a random location within the simulation field.

As an alternative to the above, a more elaborate algorithm using a bounding box for each polymer, and moving the centroid of each polymer away until the bounding boxes just touched would avoid the looping problem, although care would have to be taken in the case of polymers with irregular geometry not to leave unnaturally large spaces between the objects.

while (objects overlapping)

- Find a pair of overlapping component spheres A and B, (one from each overlapping object), with radii A_r and B_r , where the distance between centres, minus the sum of their radii ($A_r + B_r$) is smallest.
- Find the separation vector V from the centre of sphere A to the centre of sphere B.
- Find the separation vector S required between the sphere centres for the sphere surfaces to be osculating : $\bar{S} = \hat{V} \times (A_r + B_r)$
- Find the change vector C between the current separation vector V and the required separation vector S ; $C = S - V$
- Move the owning object of sphere A by *minus* $C/2$, and the owning object of sphere B by *plus* $C/2$

repeat until all the object spheres are non-overlapping. (Take emergency action if process has repeated more than 20 times)

Algorithm 6.1 : Interpenetration Avoidance

Approximating Rotational Motion

Rotational motion for monomers is quite straightforward: at the time scales used by the simulation (usually cycles are on the order of microseconds) it is assumed that the monomers will tumble so much as to assume an essentially random orientation at the end of each cycle. Hence, after each cycle they are set to point in a completely random direction. This approximation holds well for timesteps $> 1 \mu s$ and monomer sizes $< 10 \text{ nm}$.

However, as explained in the previous chapter, a more elaborate approximation is used for rotational motion in polymers, which is assumed to be energetically similar to linear motion. In a further approximation, the Brownian nature of the linear motion is extended to rotational motion, and again the distribution is assumed to be the same.

Hence no rotational momentum is calculated, instead the rotating body is assumed to be bumped by innumerable smaller particles, eventually arriving at a final position every time step, in a result based on a Gaussian distribution. This Gaussian distribution is set by the program to be

energetically identical to that of the linear motion equation. In this case however allowance is made for the shape of the polymer, and the moment of inertia is used in calculating the Gaussian distribution along each axis.

A further constraint requires that the user has entered the .pddf files in such a way that the co-ordinates for the primary axes of the MOI (moment of inertia) coincide with the Cartesian axes of the .pddf file. The reason here is simply computational efficiency; in the absence of a more detailed model of rotation there seems to be no reason to correct the slight error caused by approximating the MOI to the co-ordinate axes rather than calculating the full MOI tensor.

Note that the fact the MOI is known and calculated during movement makes it possible for the modelling of the linear motion of the polymer to be improved, if a suitable model for the Brownian motion of irregular objects becomes available.

The final result of these approximations, then, is that large polymers behave in a manner that is mechanically reasonable. Long thin polymers such as actin strands (e.g. Fig. 6.6) or microtubules may spin easily around their major axis, but will not deviate far in rotating around their minor axes, whereas globular polymers (such as viral capsid assemblies) will tumble equally in all directions. The full movement algorithm is given by algorithm 6.2.

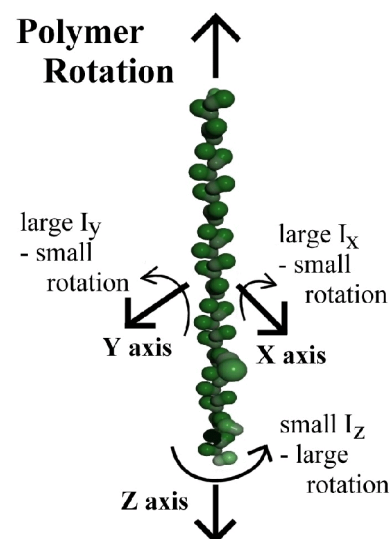


Figure 6.6: Rotating Polymer

- Calculate Mass of Polymer
- Calculate Centre of Mass of Polymer
- Calculate Moment of Inertia
- Calculate Diffusion Constant (eqn 5.20 - 5.24)
- Calculate linear RMS - positional probability distribution. (eqn 5.19)
- Resolve linear position
- Calculate rotational RMS values, giving probability distribution of rotational position (in next time step). (eqn 5.26, 5.29)
- Resolve rotational position

Algorithm 6.2 : Movement Algorithm

Boundary Conditions

One of the most challenging programming tasks was, surprisingly, the correct handling of boundary conditions within the simulation. As explained in the previous chapter, the model uses a tessellated boundary condition to simulate an infinite space for the physical objects to move within (Fig. 6.7). In practice, this simply means that objects leaving one side of the cubical simulation volume are ‘wrapped-around’, appearing immediately through the opposite side of the simulation.

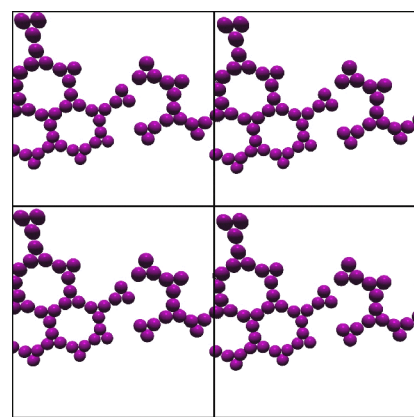


Figure 6.7: Tessellated Boundary Conditions

This is a trivial programming task for single monomers, although some special handling is required during interaction testing and so forth (detailed below), when the centre of the monomer may be on one side of the simulation volume while some of the monomer’s component spheres are on the other side.

However it becomes a far more significant task for polymers, because they will frequently have monomers on two sides of the cubical simulation field. In fact, it is not unknown for a polymer in a corner of the simulation field to have monomers existing in all eight corners of the simulation cube simultaneously. This makes it essential to ‘normalise’ the position of these border polymers when doing linear or rotational movement. This ‘normalisation’ simply involves moving the polymer into the positive octant of the Cartesian co-ordinate system (i.e. all co-ordinates in the +x,+y and +z region of space) before manipulating it, and then ‘denormalising’ the polymer and moving it and its constituent monomers back into the simulation field proper. To put it another way, ‘normalisation’ is enlarging the positive octant so that the entire polymer simulation space falls within it. It should be emphasised that this process is done only for polymers straddling the simulation boundaries, which are usually a very small minority of all the polymers in the simulation.

While this is not too arduous, the programming difficulty is increased when handling unusual situations such as the joining of two polymers, and can even become inconvenient in more

mundane tasks such as the adding or shedding of a monomer from a polymer. For example, it is necessary to normalise the polymer before the moment of inertia and diffusion constant are calculated. These boundary conditions must also be borne in mind when checking for errors or unusual conditions (e.g. for polymers joining across the simulation field boundary), a fact that can make such checks substantially more computationally expensive.

Populating the Data Grid

As mentioned previously, the program uses a 3-D ‘bucket’ system to store the monomers (either free, or bound within polymers) within the system for swift access (Algorithm 6.3 describes the bucket access system). The simulation space is divided up into a grid of cubical cells, each cell of which maintains a list of all the monomers within its borders. The grid is a 3-D array, which is indexed by an x, y and z integer index.

Every time the monomers move, either freely or as part of polymers, this data grid must be reset, and all the monomers registered with the appropriate data cell. Since the simulation field is centred on the origin (i.e. it extends equally into all eight octants), some minor adjustment is needed to allocate a monomer to the correct data cell.

Let **cellsize** be the size in nanometres of a datacell
 Let **fieldsize** be the size in nanometres of the simulation field
 Let **number_of_cells** be the number of datacells in a single row
 Let **pos** be the vector position of a monomer in 3-D space

Take a monomer’s position **pos**
 For each dimension (x, y and z) take the corresponding component value (**c**) of **pos**
 (i.e. **pos.x**, **pos.y** or **pos.z**).
 Using that component value **c**
 Add **fieldsize/2**
 Divide by the **number_of_cells**
 round the result *down* to the next lowest integer (i.e. -0.2 rounds to -1)
 This result is the array index of the monomer within the 3-D data grid for that dimension
 (i.e. index = $\frac{c + \text{fieldsize}/2}{\text{number_of_cells}}$)

Summary of Object Movement

The program's treatment of molecules may be summarized as follows.

Molecules:

- move in a Brownian manner;
- move in discrete jumps;
- always rotate to a random orientation;
- have polymer rotation approximated in a physically reasonable manner;
- have boundary conditions that are allowed for;
- populate a 3-D data grid; and
- have collision checks made after movement.

Object Interaction

After they have moved, the objects must be checked to see if they have come close enough to interact. The actual calculation of whether a protein in solution is going to bind to another would be quite arduous, and would presumably require detailed treatment of the protein's solvation sphere, the surface charge distribution, the geometry and nature of binding sites, and so on. Fortunately all these factors can be reduced to a single probability - the probability that, if and when the two monomers collide, they bind. (In fact, the technique is slightly more subtle than this - see the section on 'calculating binding probabilities' below.)

The first step then in calculating interactions is to see whether the monomers will collide. As explained in the previous chapter, due to Brownian motion this is not as simple as it might first appear.

Collision Probabilities

In classical frictionless Newtonian conditions, particles continue on their way until they are acted upon by a force, such as collision with another particle. In such conditions, determining whether two particles collide is, at least in theory, purely a matter of deterministic mathematics.

In this simulation, however, the number of collisions is so immense that such methods are infeasible. Instead, we must consider the problem of how to calculate the *probability* of two particles, each moving in a Brownian manner, colliding. This is done by numeric simulation, (as explained in the previous chapter, and in Appendix A). In the program the values from such a pre-calculated numerical simulation are used to calculate (given the position, sizes and RMS movement values of two monomers) the chance of collision during the next time step.

Each monomer is tested against its neighbours in its own, and in adjoining, grid cells. This, in turn, provides an upper bound on the length of the time step. As the time step increases, the range of each particle per time step also increases (as the square root of the time step). If the time step is too large, then the simulation becomes inaccurate because a particle may interact with others that are *more* than one grid cell distant - a possibility the program does not check. As a result, for strict accuracy, the time step should be such that the size of a grid cell is larger than three standard deviations of monomer movement (i.e. 3 times the RMS change in position).

In practice this constraint can probably be relaxed without great consequence, because the method used to calculate binding probabilities will adjust these binding probabilities to a slightly higher value to compensate for the lost opportunities for more distant interactions. Nonetheless, time steps leading to one standard deviation monomer movements of more than one grid cell width should almost certainly be avoided, because this will remove the program's ability to accurately simulate local concentration gradients and may affect the accretion rate of large polymers.

Object Binding

When monomers are found to have collided, the chance of binding for the two closest available (i.e. not already in use) binding sites on the two monomers is checked. (This use of the *closest* sites may require binding probabilities to be adjusted if some sites thus become more geometrically likely to link up than others). At this stage, the chance that binding would occur is found using the data originally read in from the .pddf file, and the monomers bind with that probability. A randomiser is used to generate a number from zero to one, and if it is greater than the required probability as specified by the user, then binding commences.

A relatively complex procedure is required to simulate binding. Three situations can occur: either both monomers are unbound and floating free; or either one of the monomers is part of an existing polymer; or both monomers are part of existing polymers.

Binding Free Objects

The first condition that must be satisfied is that the objects are oriented and positioned with respect to each other (Fig. 6.8). This is done by moving the objects (if one is bound, only the free object is moved) so that the two objects' linkage sites are coincident, and their 'direction' vectors are exactly anti-parallel. The objects are then rotated so that their 'twist' vectors coincide.

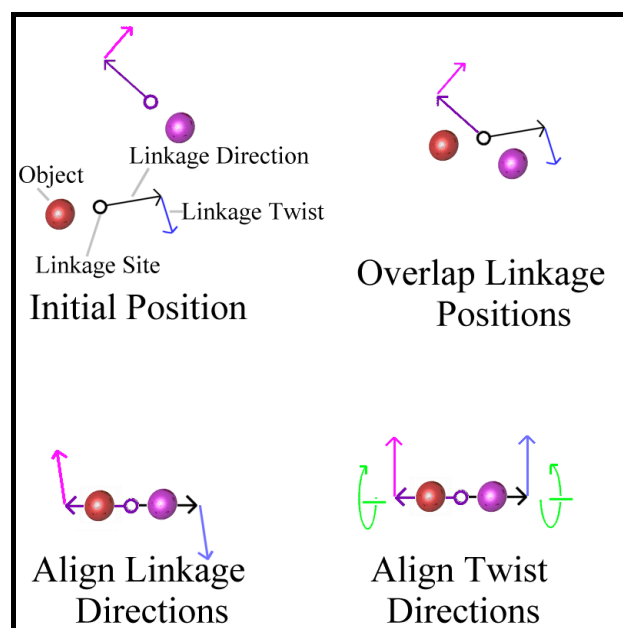


Figure 6.8: Dimer binding and alignment

If both monomers are free, a new polymer object is created to store the data of the union. First a polymer object of mass zero, with no objects, is created. One monomer, and then the other, is added to the new polymer, causing the polymer to update its internal list of ‘owned’ monomers.

At the same time the physical properties of the polymer are updated: the polymer acquires the summed mass of its constituent monomers, the centroid (the ‘centre of gravity’) of the polymer is calculated, and from that the moment of inertia is calculated using the parallel axis theorem to sum the moments of the constituent dimers. Finally, using this moment of inertia and the aggregate mass, the diffusion constant of the new polymer is calculated in the manner outlined in the previous chapter. Before any of these calculations are done the monomers are ‘normalised’, if necessary, to avoid the difficulties of calculating these constants in a polymer that spans the simulation field boundary.

To summarise, when free monomers bind together:

- the monomers are physically aligned according to their binding links;
- a new polymer is created;
- the monomers are registered with this polymer;
- the polymer mass is calculated;
- the centroid is calculated;
- the moment of inertia is calculated; and
- the diffusion constant is calculated.

Binding a Free Monomer and a Bound Monomer

If one of the monomers is bound, the procedure is slightly different. Firstly, a check is performed to make sure that the position into which the new object will fit is not already taken. This is done by iterating through all the monomers already bound to the polymer, and making sure that no physical conflict exists between the proposed position of the newly binding monomer, and the positions of the existing monomers.

Secondly, the new monomer is added, and the polymer properties updated as outlined above. Finally, further checks are made to see whether any additional links are possible - i.e. whether the new monomer has 'slotted into' a position that is adjoined by multiple bound monomers (Fig. 6.9, algorithm 6.5).

While the polymer does not explicitly record the linkages its constituent monomers make, it does initiate these checks for extra 'cross-links' between its 'owned' dimers. The algorithm used is not optimal - it simply interrogates every monomer in turn, asking it whether it is adjacent to the new monomer and in a position to bind it. If this is the case a new link is created. This can lead to quite a complex set of links within a polymer (Fig. 6.9), which must be correctly maintained as monomers are added and, more importantly, as they are deleted.

Example of debugging linkage output and the corresponding linkage web for a small polymer:

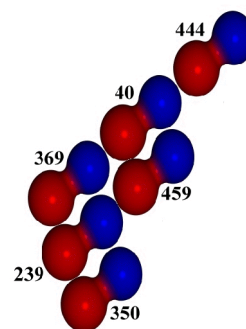
(Output format is the object number, followed by the links for four sites. Output was from a tubulin test run)

program output:

```
Object: 369:  - 239  - 40
Object: 350: 239  -  -  -
Object: 444:  -  - 40  -
Object: 239: 369 350  - 459
Object: 40:  - 459 369 444
Object: 459: 40  - 239  -
```

resulting inferred linkages:

```
369 - 40 - 444
  |   |
239 - 459
  |
350
```



Graphical Representation of output data

Figure 6.9: Polymer Linkage

After the monomer is added, more checking is carried out to ensure that the polymer does not enter an illegal state, e.g. with bound monomers believing they are attached to other monomers that are now free, or (somehow) monomers overlapping each other within the polymer. Fortunately, although such checking is vital during program development, it is usually unnecessary when the program is stable and running normally.

With the new monomer,
 find the proposed position into which it will bind

For each existing monomer,
 check that the proposed position does not overlap with an existing monomer's position.

Add the monomer to the polymer, recalculating the polymer's:

- mass;
- centroid;
- moment of inertia; and
- diffusion constant.

For each existing monomer,
 For each *free* link on the existing monomer
 Make an extra linkage if possible with the new monomer.

Check the new polymer for errors.

Algorithm 6.5: Adding a single Monomer to a Polymer

Merging Two Polymers

The most complex situation is when two colliding monomers are already attached to others, and hence two polymers must be joined. The algorithm used takes a record of every monomer in the smaller polymer, along with its links and state. The monomer which initiated the binding, is bound first. Next that monomer's old neighbours are 'brought across', being removed from the old polymer and attached to the new polymer, and then rebound with the same links that they had previously, and then (recursively) all the neighbours that *they* had are brought across, and so on. As each monomer is added it is tested to see if it physically 'fits' into the new polymer (i.e. that there are no obscuring pre-existing monomers), and if it does not fit it simply

returns to the pool of free monomers in the correct state (algorithm 6.6). While this may seem a little artificial, in fact it seldom occurs that two polymers come together (although this can still be a significant assembly path), and even more seldom does it happen that they have ‘jagged edges’ that break off during assembly. After all the monomers have been brought across, the old polymer (now empty of monomers) is destroyed.

It is theoretically possible that this algorithm could lead to two misshaped polymers overlapping, and the first polymer ‘stealing’ a single monomer, and ‘unzipping’ the remaining polymer completely. While this could occur if the polymers were irregularly shaped, it represents a pathological case that should occur very rarely.

Find the monomer that initiated the contact between the two polymers
(The ‘First Monomer’)

‘**Add**’ the First Monomer

to **Add** a monomer...

Record the monomer’s links and state

Break the monomer off

Attach the monomer to the new polymer (as per free monomer addition)

For each link of the newly added monomer,
 test the link, and **Add** any monomers not yet attached to the new polymer

(Nb: while the algorithm is recursive, it is implemented in code as a multi-pass loop)

Algorithm 6.6: Combining Polymers

Unbinding

In every cycle after all new bindings have been completed, all polymers check to see if any of their constituent monomers break away. (The probability of this happening is set by the user in the .pddf file.) If they do, the monomer must undergo internal checking to see if this event triggers a change in state. Meanwhile the polymer must recalculate its physical characteristics, and check for any orphaned monomers that no longer have links connecting themselves to the bulk of the polymer. Furthermore, the polymer must check that it still exists - if the polymer is reduced to a single monomer, it loses its *raison d’être* and frees the remaining monomer before deleting itself.

The probability of a multiply-bound monomer unbinding must be calculated, depending on the program options used (the methods are explained in more detail in Chapter 5). In the .pddf file the user specifies whether the probability of multiply bound objects is:

- zero : they cannot break off;
- the multiplicative sum of all bound links; or
- the ‘order of magnitude’ heuristic, which divides the smallest link probability by ten for each extra link (see algorithm 6.7).

Find Smallest Breakage Probability of an individual link.
Find number of links.
divide the smallest breakage probability by 10 for each extra link.
set the probability of all links breaking to this newly calculated value.

Algorithm 6.7: Multiple Link Breakage Heuristic

State Changes and Random Events

When monomers bind or break away, their internal state machine checks to see if the particular binding or breaking event was significant enough to change the state of the monomer. It should be emphasised that it is quite possible for an object to have only one ‘state’ - for example the properties of a stain particle might not change depending on whether or not it is bound (although it would probably change valence). Many proteins however undergo relatively complex state cycles, from a free phosphorylated state, through to various bound states, and finally back to a free unphosphorylated state, from which they may recover over time.

The state machine not only checks every binding and breaking event to see if the state changes, but also whether the current state is unstable - i.e. whether there is a random chance of decaying into a different state. Such a random event can be used to simulate the stochastic rephosphorylation of an unphosphorylated protein, or the dephosphorylation of a recently bound protein.

The Program Cycle: Playing the Game

Having examined the individual components of the program, the overall functioning of the simulator should now be fairly clear. The Co-ordinating Object runs the following execution path (Fig. 6.10):

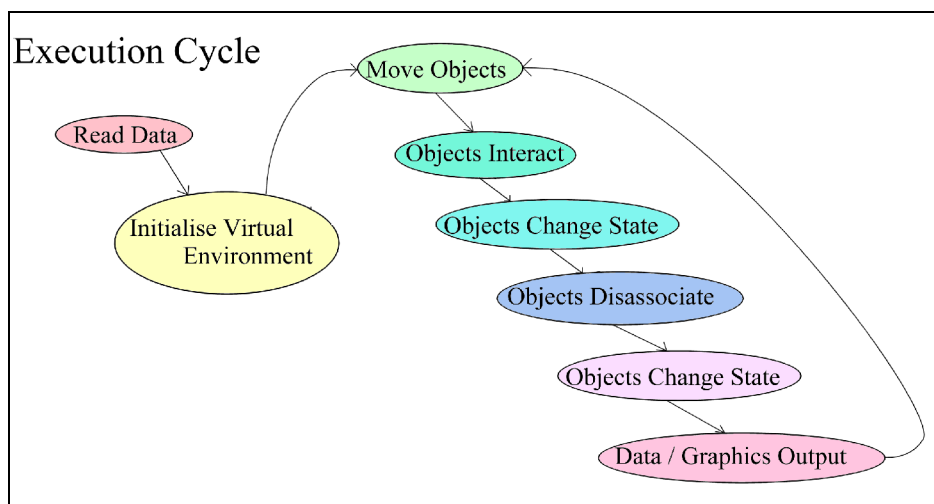


Figure 6.10: Program Execution Cycle

Data are read in from the user in the form of command line arguments and .pddf initialisation files. The virtual environment is then set up: memory is allocated, data structures initialised and objects created. After this is done, the co-ordinator enters a repeating cycle of object movement and interactions that either runs for a fixed length of time, or until interrupted by the user (algorithm 6.8).

The accessing of objects is done in separate ways depending on context. The co-ordinator maintains a complete list of all objects, and this is used during the movement phase to move all unbound monomers. Since monomers are never created or destroyed during the program execution, this is an efficient way to access them. Polymers are continually being created and destroyed, so they are tracked using a linked list. During movement this linked list is iterated through, and each polymer moved. During collision prevention (a sub-phase occurring immediately after movement) and the object interaction phase, the objects are accessed via the 3-D grid cell array, with all grid cells being consecutively searched for objects, and when an object is found, adjacent cells are checked for occupancy and consequent interaction. For disassociation, objects in polymers are again searched by linked list. Objects being checked for

a random state change within polymers are also checked from the polymer linked list, while free objects are accessed via the co-ordinator array of all objects.

State changes occurring due to binding and breaking occur immediately, while state changes due to random events (e.g. rephosphorylation) occur after the breakage phase.

At the end of the cycle comes data output. Usually the program does not output data every cycle. Depending on user parameters the program will output numerical data logs on a regular basis (the default is 200 cycles). These numerical data logs indicate the number of objects, how many objects are free and how many in polymers, a break-down of how many objects are in which states, the size distribution of aggregates, and track the changes over time in selected individual polymers.

In addition the program may save visual and geometric data files. If directed to, it will save a geometric data file suitable for a raytracer (e.g. every 10,000 cycles), and can even save a quick 'snapshot' of the graphics window as a separate option (e.g. every 1,000 cycles). These 'snapshots' can be useful for debugging an unattended run, or quickly creating animations, because they do not require the overhead of running the ray-tracer.

At the end of the program's execution the program outputs all its normal data sets, and some further data. This includes a speed index (number of object cycles per second achieved), and a detailed list of the final size distribution of aggregates.

The Program:

reads in data, initialises virtual environment

Until the simulation is finished

- monomers and polymers move
 - monomers and polymers bind together
 - monomers break away from polymers
 - data are output in a number of formats
- when the simulation finishes, a final data dump occurs

Algorithm 6.8: The Program Cycle

A Special Mode: Working Out the Probabilities

An obvious difficulty with the nano simulator program is how to obtain the probabilities for the binding and breaking away of monomers to and from polymers. Ideally these numbers would be obtained from a detailed understanding of the protein structure and its binding sites. Useful approximates however, can be inferred from studies of bulk properties (i.e. by observing how much raw material polymerises in a solution, combined with a knowledge of how many individual polymers exist). Such experimental studies can give rise to raw ‘on and off’ rates for monomers.⁷⁸

Translating these into program terms is still non-trivial. To convert the ‘off-rate’ into a value that can be used by the program is quite straightforward - if monomers break off the tip of a polymer at the rate of five a second, we can simply multiply by our time step (for illustration, say 1 ms) to find the ‘breakage probability’ for the link (in this case $p(\text{breakage}) = 5 * .001 = 0.005$ per time step).

To translate the ‘on rate’ (which depends on concentration) into a binding probability for a single dimer, it is necessary to know how many collisions occur in one time step.

To discover this, the simulator is run in a special test mode, in which no binding actually takes place. Instead the number of collisions with a stationary test monomer (representing the growth tip of a polymer) is recorded. Using this number, and modifying it if necessary to take into account the number of binding sites, it is possible to work out the binding probability.

For example, if in a 1 ms time step a monomer with two symmetric binding sites experiences, on average, 10 collisions, and the on rate is 12 monomers/second, we would find the binding probability as $P(\text{binding}) = (12 * .001) / 10 = 0.0012$ per time step.

Development Notes

The nanosimulator computer program was written in conjunction with a number of other programs, some written by the author and some available as freeware from other sources.

Development Environment and Portability

The computer language chosen for development was C++, because it is a high-level, object-oriented language that still compiles to fast machine-level instructions. The development took place on a series of Silicon Graphics machines, and was finally finished on an SGI O2 R10000. The final program runs under SGI Irix 6.3 (a Unix variant), and primarily used the ‘gnu’ freeware C++ compiler ‘g++’. It uses the older SGI ‘gl’ graphics library for real-time output.

In an effort at portability, a version was created without the SGI-specific code, and compiled under the other main Unix operating system variant, System V, running on a DEC station. Since the main thrust of the program is numerical simulation, there is good reason to believe it could be compiled with ease on other platforms (such as PCs), but the lower performance of such machines means that this has not yet been attempted.

While real-time graphical output is not currently available on non-SGI systems (an ‘OpenGL’ version is planned for the future, and will run on a wide range of systems including PCs), the program saves ‘Povray’ format ray-tracer files (details of this program are given below).

Related Programs

In addition to the simulator, a number of other programs were developed for this project. Firstly, a 3-D modelling utility program capable of taking a mathematical description of objects and returning it in a number of formats was written. Secondly, a small collision testing program (described below) was put together. Thirdly, testing the program required some test- bench and debugging software to be put together. Fourthly, a 3-D object viewing program was written to allow fast 3-D manipulation of geometric data files, and a subset of this program was later incorporated directly into the nanosim program itself.

A number of externally written programs were used, the most important of which was the Povray ray-tracing program. Povray is a popular ray-tracer developed by a large community of graphics programmers; it is freely available at the web site <http://www.povray.com>, as is documentation, a number of examples, and some other utility programs. The Povray ray-tracer is available on all major platforms (Unix, Macintosh, PC, Amiga, and others). Also used in general image preparation was the 'xv' Unix graphic file viewing utility.

Finally, the image processing described later in this thesis was largely done using the SGI Explorer package, which provided a framework for image processing modules, as well as a number of pre-written basic image processing functions (such as Fourier transforms and filter functions). The extended image processing modules that were written are described separately in the chapter dealing with the processing of synthetic images.

The Collision Simulator

The Collision Simulator was a small program written to numerically calculate the odds of two spheres, both moving with Brownian motion, colliding over a certain time step. This simulation was repeated for different sphere sizes at different distances, providing an array of probability values that are used within the program. The nanosimulator program is able to scale all spheres

to fit these values, and thus establish the chance of collision. (Details of the theory behind this program are found in Appendix A ‘Calculating Collision Probabilities’).

3-D Data Creation/Conversion Utility

In order to create mathematically precise 3-D models, a program was written to take a series of mathematical object descriptions defining a 3-D scene in terms of spheres, blocks, and cones, and also more complex shapes such as helices and trigonometric and polynomial surfaces, and output the result in a variety of formats:

- as monomer objects suitable for reading into the nanoscale simulator;
- as raw polygons in a ‘wavefront’-like format suitable for some third party software packages; and
- as ‘povray’ ray tracer files.

In addition, the raw scene descriptions were able to be directly viewed using the 3-D model viewer described below (Fig. 6.11).

This modeller was used to create static models for the nanoscale simulator (see Fig. 6.4 and 6.5) as well as the basis models used in the ‘Virtual Electron Microscopy’ described in later chapters.

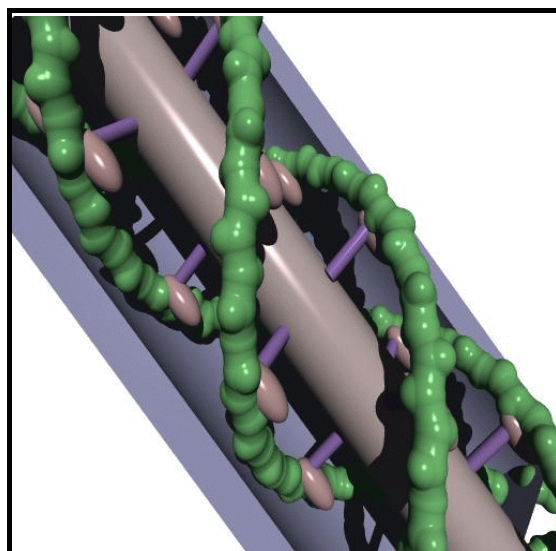


Figure 6.11: A mathematical model of a scene created with the 3-D modeler.

3-D Data Model Viewer

This was a 3-D viewing utility that read in the same mathematical object descriptions as used by the data creation/conversion utility described above, and allowed the user to view them in real time, changing viewpoint and perspective continuously when desired. But images from the 3-D viewer are not of as high a quality as those from the ray-tracer, because there is no anti-aliasing and smooth surfaces are not perfectly represented as the viewer uses polygons (Fig. 6.12 below).

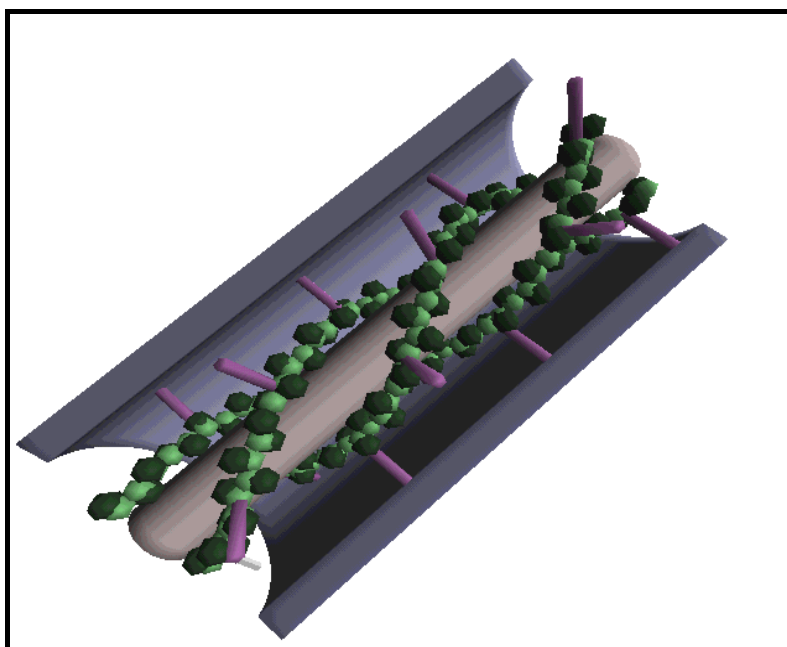
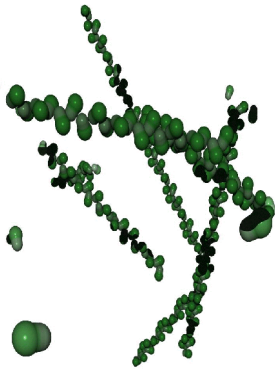


Figure 6.12: A screen shot of real-time 3-D viewer output.

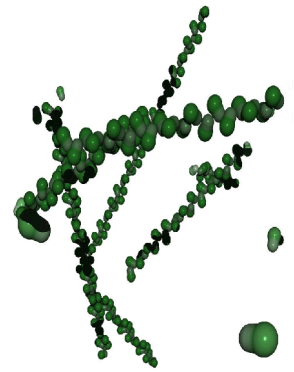
The real-time viewer used the SGI 'GL' library for fast rendering, and benefited from a number of optimisations available on SGI machines, such as fast polygon pipelining. Elements of this program were later incorporated directly into the nanoscale simulator.



Chapter 7

Modelling Simple Structures:

Actin



E lineis fit robur

(Strength comes from cords.)

- anon

Introduction

Actin is a common cytoskeletal protein that has been studied in great detail (see Chapter 2). It forms long, filamentous strands that are extremely common in all eukaryotic cells. It is particularly common in muscle tissue where, together with the protein myosin, it provides the mechanical forces required for muscle action⁷⁹.

This chapter looks at how the nanosim program can be used to simulate the self-assembly of actin. It examines the existing theory and experimental evidence, and shows how this can be processed to produce a behavioural model for a single actin monomer. It then describes the use of the simulator to model actin self-assembly, and reviews the program's results, comparing them with other experimental evidence.

The Different States of Actin

Individual actin proteins in the unbound state are known as *g-actin* (“globular” actin), while actin bound within an actin strand is known as *f-actin* (“filamentous” actin). Actin can self-assemble *in vitro*, and has the property that both assembly and disassembly occur at a faster rate at one end of a filament, called the “plus” end, than at the other end of a filament, the “minus” end (Fig. 7.1)⁸⁰.

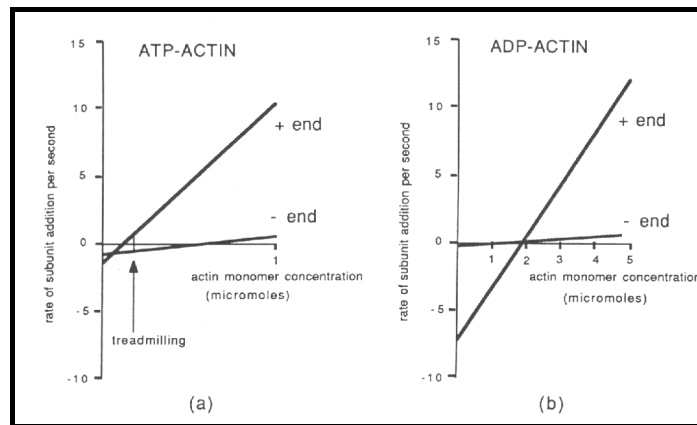


Figure 7.1: Actin assembly/disassembly rates (in one particular environment)

This difference in assembly and disassembly rates at either end can lead, at appropriate concentrations, to a process known as *treadmilling*, where a particular filament may be growing at one end while shrinking at the other.

A further complication is that, in common with many self-assembling proteins, actin *dephosphorylates* or *hydrolyses* when it binds. *Phosphorylated* actin has a bound adenosine 5' tri-phosphate molecule (ATP), a common energy-providing molecule used in a large number of cellular processes. This ATP molecule makes the actin protein far more likely to bind to other actin proteins. Some time after being bound, in a process that is not yet fully understood, the bound ATP molecule hydrolyses, losing a phosphate ion and becoming the less energetic adenosine 5' *di*-phosphate (ADP). This in turn causes the actin protein to become less securely bound.

It has been established that actin filaments usually terminate with phosphorylated ATP-actin, but whether the individual molecule dephosphorylates over time, or upon the addition of another actin molecule, is not yet completely established (a similar process, and a similar debate, occur as regards the tubulin protein examined in Chapter 8).

The Structure of Actin Filaments

The structure of actin has been extensively studied using electron microscopy and image analysis and, despite many experimental difficulties, the nature of actin filaments is now well understood⁸¹. It can be considered as a long two-stranded helix with a repeating length of approximately 36-40 nm (the length 'L' in Fig. 7.2⁸²).

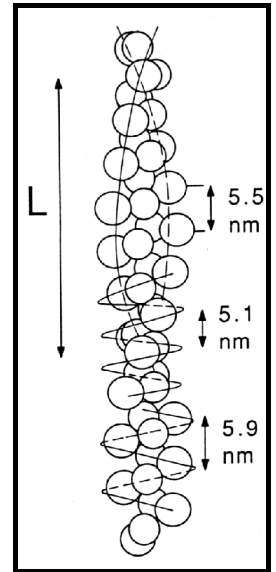


Figure 7.2:
Actin structure

Actin Polymerisation and the Oosawa Model

The assembly of actin is one of the finest examples of Oosawa's model of protein polymerisation in action, closely fitting the mathematical predictions made by the model⁸³ (cf Chapter 5). Oosawa's model predicts that the *form* of the polymerisation curve (a graph of the amount of the actin bound in filaments, as opposed to unbound, over time) will be the same for different environmental conditions, differing only in the rate that the reaction proceeds. Measurement of polymerisation is usually done via the viscosity of the solution, and results for the polymerisation of actin can be shown to have the same shape when plotted against logarithmic time, being simply offset by different amounts depending on whether the environmental conditions favoured rapid or slow polymerisation (Fig. 7.3)⁸⁴:

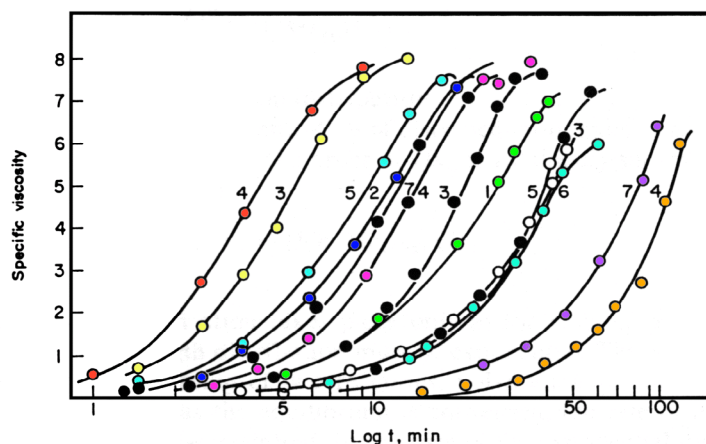


Figure 7.3: Actin polymerisation curves under different environmental conditions.

The Oosawa model predicts that the slope of these curves at intermediate values will be relatively constant, approaching the on-rate times the concentration minus the off rate; $k_{\text{on}}c - k_{\text{off}}$. (This characteristic slope is more evident on a linear scale than on the logarithmic scale above.)

Another prediction of Oosawa's theory is that the length distribution of actin filaments is initially a Poisson distribution at the start of polymerisation, moving to an exponential distribution when a solution has reached steady state. This is confirmed by painstaking experimental studies that have catalogued the length of actin filaments, and have found the expected exponential distribution (Fig.7.4) ^{85,86}:

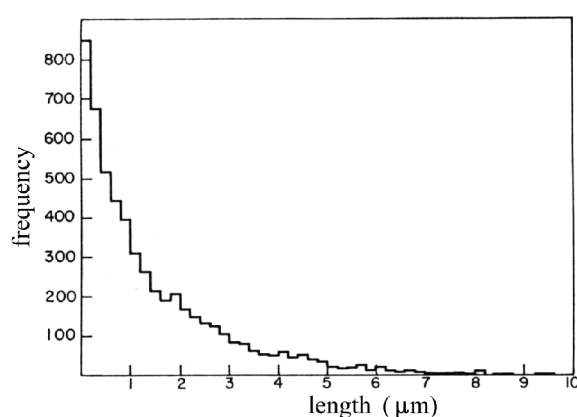


Figure 7.4: Actin polymerisation length histogram

Suitability of Actin for Simulation

These theoretical details, in conjunction with the wealth of experimental data, make actin an excellent candidate for trialing the nanosimulator, to see whether it reproduces correctly the behaviour of this relatively simple self-assembling protein. Actin has been previously simulated mathematically with good results, and a number of sophisticated models (including those modelling actin-associated proteins) have been developed^{87,88,89}. As far as the author is aware however the computational technique of simulating large numbers of individual actin monomers as described below is new.

Setting up the Simulation

Experimental studies of actin have provided accurate details of the assembly and disassembly rates of the growing filament ends. To test the simulation, an arbitrary *in vitro* environment is chosen for which good experimental evidence is available. The current simulation uses data from Pollard⁹⁰, who used a standard experimental environment conducive to filament growth, at 37°C, and obtained the following figures (also used in figure 7.1):

	ATP-Actin		ADP-Actin	
	+ end	- end	+ end	- end
$k_{\text{on}} (\mu\text{M}^{-1}\text{s}^{-1})$	11.6	1.3	3.8	0.16
$k_{\text{off}} (\text{s}^{-1})$	1.4	0.8	7.2	0.27

Table 7.1 Actin association and disassociation rates⁹¹

Some interesting features of these figures deserve note. First, although ATP-actin is the most active and inclined to bind, it is obviously possible at higher concentrations (i.e. around the 2 μM level and above) for ADP-actin to polymerise into filaments. Second, although ATP-actin is generally more likely to bind and less likely to break away than ADP-actin, in a curious inversion ADP-actin is *less* likely to break away from the slow-growth, “-” end of the filament.

Using these figures, a .pddf file (the “protein dynamic description file” used by the nanoscale simulation program) was constructed, that uses a four-state model for actin (Fig. 7.5):

- unbound and phosphorylated;
- bound and phosphorylated;
- bound and dephosphorylated; and
- unbound and dephosphorylated.

This represents the presumed cycle of binding, dephosphorylation, unbinding, and rephosphorylation.

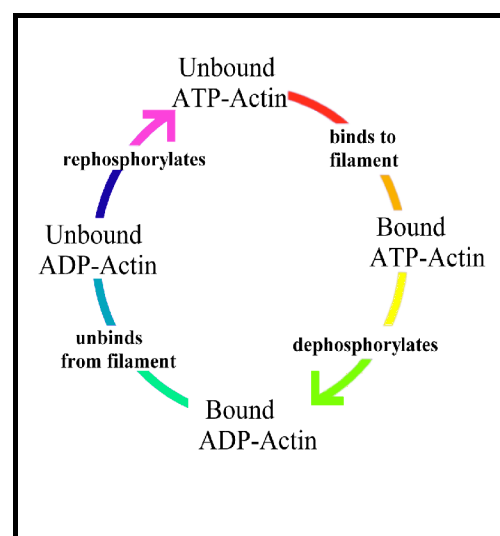


Figure 7.5: 4-state actin cycle

(Nb Figure 7.5 does not show the less common path whereby an unbound ADP-actin monomer binds immediately to a filament without rephosphorylation, although this is included in the model.)

Rephosphorylation requires an energy input into the system, and can occur when free unbound ATP replaces the phosphate-depleted ADP of a free actin protein. This occurs within the cell, and can be simulated *in vitro* simply by providing free molecules of unattached ATP.

Deriving Figures for the .pddf File

Adapting the experimental figures of Table 7.1 for the simulator requires some further work. The .pddf files require binding and breaking values on an individual protein basis, rather than for the entire filament.

Deriving these from the off-rates is quite straight forward; we can use the figures from Table 7.1 directly, modifying them depending on the time step of the simulation. For example, if the simulation uses a 100 μ s time step, then the ATP-actin + end off-rate of 1.4 actin molecules per second becomes .00014 actin molecules per 100 μ s time step.

Deriving on-rates is more complex. In order to derive these parameters, the number of interactions per second of a simulated actin protein must be determined, by running the nanosimulator in test mode. In this case, using a 100 μ s time step, the number of collisions of an actin protein in a 1 μ M solution is found to be 4050/s. The k_{on} figures in Table 7.1, divided by this result, give the interaction probability for a single collision.

These results give us Table 7.2, in a form suitable for inclusion in our .pddf file:

	ATP-Actin		ADP-Actin	
	+ end	- end	+ end	- end
p(bind)/collision	0.00228	0.00033	0.00095	0.0000395
P(break/100 μ s)	0.0001	0.00008	0.00072	0.000027

Table 7.2 Actin association and disassociation probabilities

This is graphically summarised in Fig. 7.6 and Fig. 7.7:

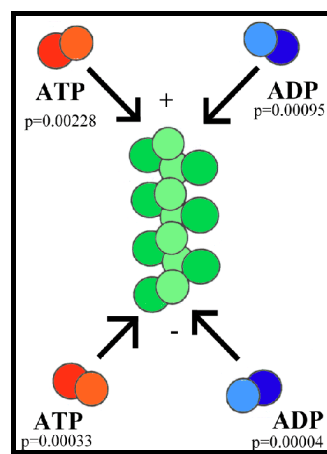


Figure 7.6: Actin Binding Probabilities

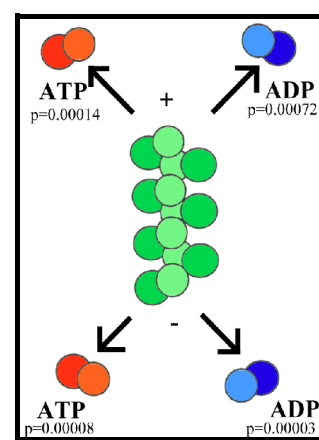


Figure 7.7: Actin Breakage Probabilities

An important point to note is that it is not possible to distinguish, on the basis of the experimental evidence available, differences in binding rates depending on the state of the actin protein already bound at the tip of the filament (hence the filament is shown as uniformly green above). Although it might be suspected that such differences exist (and if they do, they can be modelled using the simulator), this *particular* .pddf model does not explicitly include them.

Modifying the Multiple Link Breakage Model

A complication arises with handling multiple breakages. For larger structures, where a protein is bound by a large number of other links, the *ad hoc* heuristic mentioned in Chapter 5 seems to work reasonably well (setting the overall breakage probability equal to the smallest single link breakage probability, lowered further by an order of magnitude for each extra link). But actin

bound internally within a filament is linked only to two other molecules in this model (although some more recent models hypothesise that in fact links to *four* other actin proteins are involved⁹², see also the nucleation problem below). It is an experimentally observed fact that internally bound actin very rarely, if ever, breaks away. In order to model this, when simulating actin the program is run with a special option specifying that multiple links *never* break.

Handling Nucleation

A well observed feature of *in vitro* actin polymerisation is the lag phase that occurs prior to a rapid polymerisation phase⁹³. This lag phase is due to the instability of filaments when they are first assembling; until they reach a certain size (probably about three molecules, in the case of actin) the filament is more likely to fall apart than to grow. Once above this critical size, the filament tends to grow strongly. As more and more filaments make it over this initial barrier, polymerisation builds up speed. Added evidence that this is the reason for the lag phase lies in the observation that the lag phase can be avoided altogether if the solution is initially *seeded* with fragments of filamentous actin⁹⁴.

There are a number of ways this can be handled in the program. Ideally, this behaviour would grow naturally out of the binding and breaking values of the monomers. This is possible in the case of actin if the model is extended to include monomer links beyond the nearest neighbour. Unfortunately this leads into the difficulty in modelling breakages from multiple links, with the result that the breakage model chosen has a very significant impact on the nucleation lag time; in effect the user can implicitly *set* the lag time simply by choosing an appropriate breakage model.

Given this limitation, the current simulation was set to use a very simple model, with only neighbour-to-neighbour links, and to set the nucleation probability explicitly, by restricting the initial interaction of free actin to occur only between free ATP-actin monomers, and then with only a relatively low probability (less than for combining with a bound filament).

This same approach, of artificially restricting initial nucleation, is used widely later on in the thesis, due to this difficulty of modelling breakage rates for multiply bound proteins. It may be

a promising area for future extension of the program however (see “Future Directions” in Chapter 10).

Dephosphorylation

This particular .pddf file uses an ‘ATP cap’ model for dephosphorylation; i.e. an ATP actin monomer dephosphorylates only when it is bound to the ‘+’ binding site. This implies that actin binding to the ‘+’ end of a filament stays in the ATP state, while the monomer to which it binds dephosphorylates, and that an ATP actin monomer binding to the ‘-’ end of a filament immediately dephosphorylates. A more exact model would probably include a timed decay between these states, but in the absence of experimental evidence the simpler model is used. The simulator can model this type of timed decay; for example a timed decay is often used to describe the rephosphorylation of dimers when an external source of energy is available.

The Actin .pddf File

The following is the .pddf file for actin assembly, based on the figures obtained in the preceding section. As explained in Appendix C, “.pddf files”, lines starting with the ‘#’ character are comments that are ignored by the program (shown below in blue).

```
-----

# Modelling actin assembly
#
# figures derived from T.D. Pollard, J.Cell.Biol 1986 103, 2747-2754
#

Model Actin
{
    # model the geometry as a double sphere - distance in nm, colours as hex RGB

    Sphere centre    1.6 <-.5,-.5,0> 0x66CC66
    Sphere side      1.0 <0,-2.5,-1> 0x66CC66

    #site    name    <position> <direction> [<twist>]
```

```

site    top      <0,0,1.5> <0,0,1> 205.714  # a link site, direction same as position
site    bottom   <0,0,-1.5> <0,0,-1>

#state    name      [optional colour details]

State ATPunbound

State ATPbound      # a new state, with no physical changes
                    # colours as RGB values in hex

    colour centre = 0xCCFFCC
    colour side   = 0xCCFFCC

State ADPbound      # a new state, with a colour change
    colour centre = 0xAAAA00
    colour side   = 0xAAAA00

State ADPunbound    # another new state with no physical changes
    colour centre = 0x999900
    colour side   = 0x999900

# The event list is all of the form:
# Event: link { A B } | break | random (chance / cycle)
#           StateA -> StateB

Event bind { bottom top }                # bottom link causes dimer to become
                                         # part of ATP "cap"

    ATPunbound -> ATPbound

Event bind { top bottom }                # top link causes dimer to become
                                         # dephosphorylated

    ATPunbound -> ADPbound

Event bind { top bottom }                # bottom link causes dimer to become
                                         # susceptible to dephosphorylation

    ATPbound -> ADPbound

Event bind { bottom top }                # ADP actin *can* bind - it's just
                                         # not as likely.

    ADPunbound -> ADPbound

Event bind { top bottom }                # ADP actin *can* bind - it's just
                                         # not as likely.

    ADPunbound -> ADPbound

Event break { }                          # if the top link is broken,
                                         #     object becomes "unbound" with
    ATPbound -> ATPunbound              #     no change in phosphorylation

Event break { }                          # if the top link is broken,

    ADPbound -> ADPunbound

```

```

Event random 0.00001                                # a chance of "unbound" GDP dimer becoming
                                                        # phosphorylated; avg = once every 100 seconds

ADPunbound -> ATPunbound

}

# Specify links and states, and give their binding and breaking chance.
# links are transitive.
#
# nb. All these rely on a time constant of 100 microseconds...
#
# nb II : The 'break' coefficient of unbound objects is, of course, meaningless...
#         since they're not bound, they have nothing to break from... hence the '0's.
#         Also, some of these should never occur, because of the "instant
#         dephosphorylation" model currently used.
#
# nb III : Estimation of free -> free binding links. This is unknown - as a first
#           approximation, use a low initial nucleation probability...
#
#
#ATP - ATP binding
# n.b.: current model never has 2 bound ATPs together; hence break(1) if it does.
# also; bottom state should never happen for the same reason.

Binding Actin top ATPunbound to Actin bottom ATPunbound = bind(0.000001) break(1)
Binding Actin top ATPunbound to Actin bottom ATPbound   = bind(0.00014) break(1)
Binding Actin top ATPbound   to Actin bottom ATPunbound = bind(0.00288) break(1)
Binding Actin top ATPbound   to Actin bottom ATPbound   = bind(0) break(1)

# ATP - ADP bindings
Binding Actin top ATPunbound to Actin bottom ADPunbound = bind(0) break(1)
Binding Actin top ATPunbound to Actin bottom ADPbound   = bind(0.00014) break(1)
Binding Actin top ATPbound   to Actin bottom ADPunbound = bind(0.00095) break(1)
Binding Actin top ATPbound   to Actin bottom ADPbound   = bind(0.00014) break(0.00008)

Binding Actin top ADPunbound to Actin bottom ATPunbound = bind(0) break(1)
Binding Actin top ADPunbound to Actin bottom ATPbound   = bind(0.00004) break(1)
Binding Actin top ADPbound   to Actin bottom ATPunbound = bind(0.00228) break(1)
Binding Actin top ADPbound   to Actin bottom ATPbound   = bind(0.00228) break(0.00014)

# ADP - ADP bindings

Binding Actin top ADPunbound to Actin bottom ADPunbound = bind(0) break(1)
Binding Actin top ADPunbound to Actin bottom ADPbound   = bind(0.00004) break(1)
Binding Actin top ADPbound   to Actin bottom ADPunbound = bind(0.00095) break(1)
Binding Actin top ADPbound   to Actin bottom ADPbound   = bind(0.00095) break(0.00008)

# some environment constants

```

```

Temperature 310      # kelvin - physiological temperature
Viscosity    0.62     # approximate viscosity for water at 37 degrees Celsius
                  # (nb: .69 cP would be more accurate)
TimeScale    100      # microseconds per program cycle - this is at the very top of the
                  # allowable range, so as to get long results; it would be better at 10
                  # microseconds
                  # (nb. This would require recalculating the number of collisions/second)

# Finally, specify the mix of molecules in solution.

Mix Actin 100%       # (We're using only actin in this simulation)

```

Running the Simulation

Using the figures from the previous section, the simulator was run under the following conditions:

simulate time period: 10 seconds

volume: 512nm x 512nm x 512nm; i.e. $\frac{1}{2}$ μm cubed

molarity: 2.5 μM .-

timestep: 100 μs timesteps

temperature: Temperature 310 (physiological temperature)

viscosity: 0.62 cP (approx viscosity of water at 310 K ⁹⁵)

The following sections illustrate results from this (single) run of the program. Note that a relatively high molarity (2.5 μM) was used so that the simulation would produce useful results within 10 simulated seconds. This required approximately a week of computer time on an SGI O2 machine^{**}.

^{**} Note that all times mentioned in seconds during this and subsequent chapters refer to *simulated* seconds - the actual simulation times on the computer were all on the order of hours or days.

Appearance of the Simulation

(These images were rendered in Povray, with perspective.)

At the beginning of the simulation (Fig.7.8), there are only free monomers in solution:

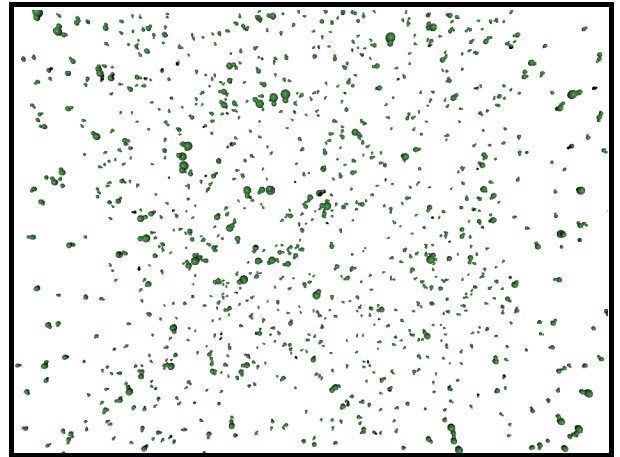


Figure 7.8: Simulation at zero seconds

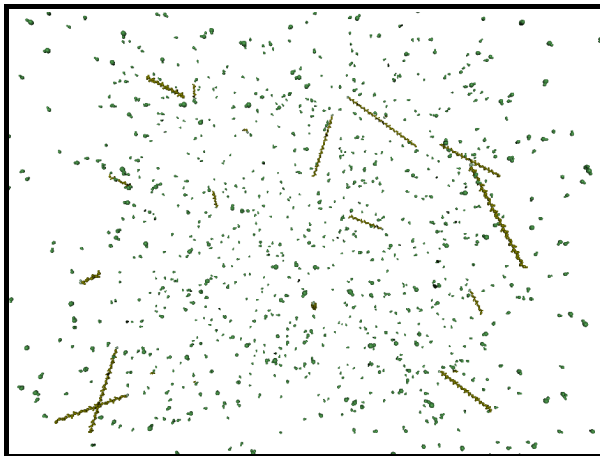


Figure 7.9: Simulation at one second

After a simulated second, a number of short filaments have started to grow (Fig. 7.9). At the end of each filament appears a green monomer, representing the ATP cap of the filament. (This may not be visible at the scale of this picture.)

After ten virtual seconds, the simulation has used up almost all the free monomers, and the filaments are starting to lose monomers, which will slowly rephosphorylate as the solution enters an equilibrium state (Fig. 7.10).



Figure 7.10: Simulation at ten seconds

Filament Growth

One of the features of the program's logging facility is the ability to save data on a single filament, or a small set of filaments, and thus track their growth. This was done in the current simulation for the first filament, which was created between 60 and 80 (virtual) ms into the program run (since the concentration was so high), and remained in existence for the life of the simulation. Figure 7.11 shows that the time at which the bulk of free ATP-actin was used must have been around about the (simulated) 2½ second mark. (Nb the graph starts at '6' since this is the size the filament grew to in the 20 ms pause between creation and the first logging point.)

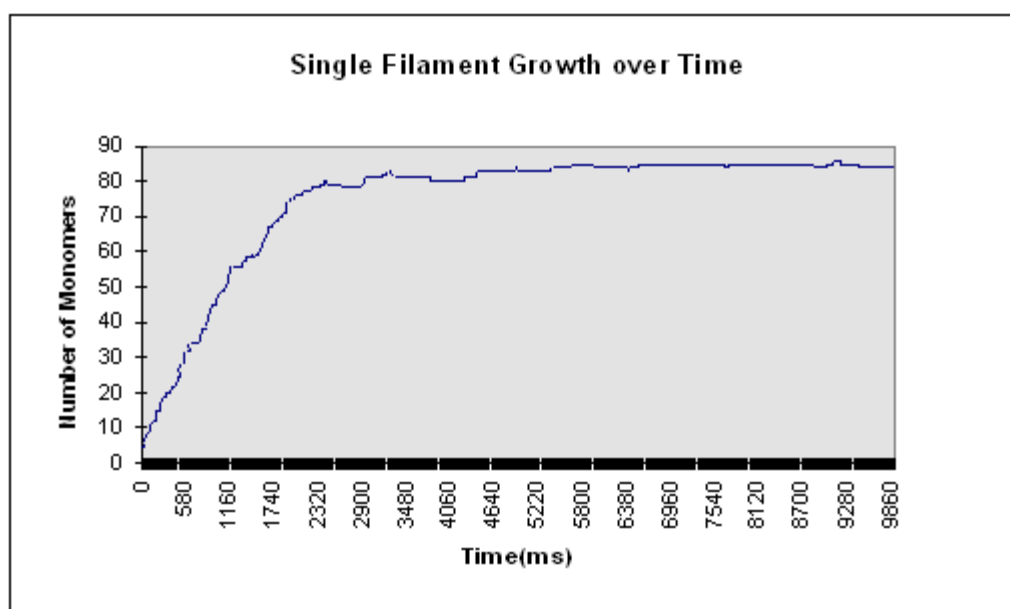


Figure 7.11: Simulated actin filament results over 10 seconds

It is instructive to compare the growth of this single filament with the predictions of table 7.1. In theory, the filament should grow at a rate of 30 monomers/second (remembering that addition occurs at both ends). During the rapid growth phase (the first 1.5 seconds above) the filament grows at greater than the expected rate, adding 53 monomers in the first 1.5 seconds, or 35 monomers/second. Using the binomial approximation that the standard deviation is approximately equal to the square root of the mean ($\sigma = \sqrt{x}$), we find the standard deviation as 5.48, so this faster growth is not particularly remarkable. But the most likely explanation lies in the erroneously low viscosity of 0.62 cP rather than 0.69cP - if the time scale is multiplied by 1.11 (as a first approximation - the real variation is somewhat more complex due to the

program not scaling linearly) to adjust for this, the on rate is found to be 31.5 monomers/second, almost exactly as predicted. A further possible factor is the increased motility of small fragments leads to a slightly larger number of collisions than when the fragment is larger.

Tracking a significantly larger population of filaments however would give a useful check on the system's accuracy, as the average growth rate should still be 30 monomers/second.

After this growth period, with the quantity of free actin greatly reduced, we expect that the filament will slowly shrink. But unless it loses its ATP cap this shrinkage will be quite slow, on the order of one monomer every four seconds (from the '-' end) and 0.8 monomers a second from the '+' end (cf table 7.1).

Polymerisation Curve

Figure 7.12 shows the amount of filamentous actin over time. Since the concentration is high and the volume small, the sample reaches saturation more swiftly than a traditional experimental run in which the polymerisation might curve might be observed over minutes or even hours (as opposed to ten seconds in this run) but the form of the polymerisation curve, as predicted by Oosawa's model, is the same as in those longer runs (cf Fig. 7.3 for experimental results).

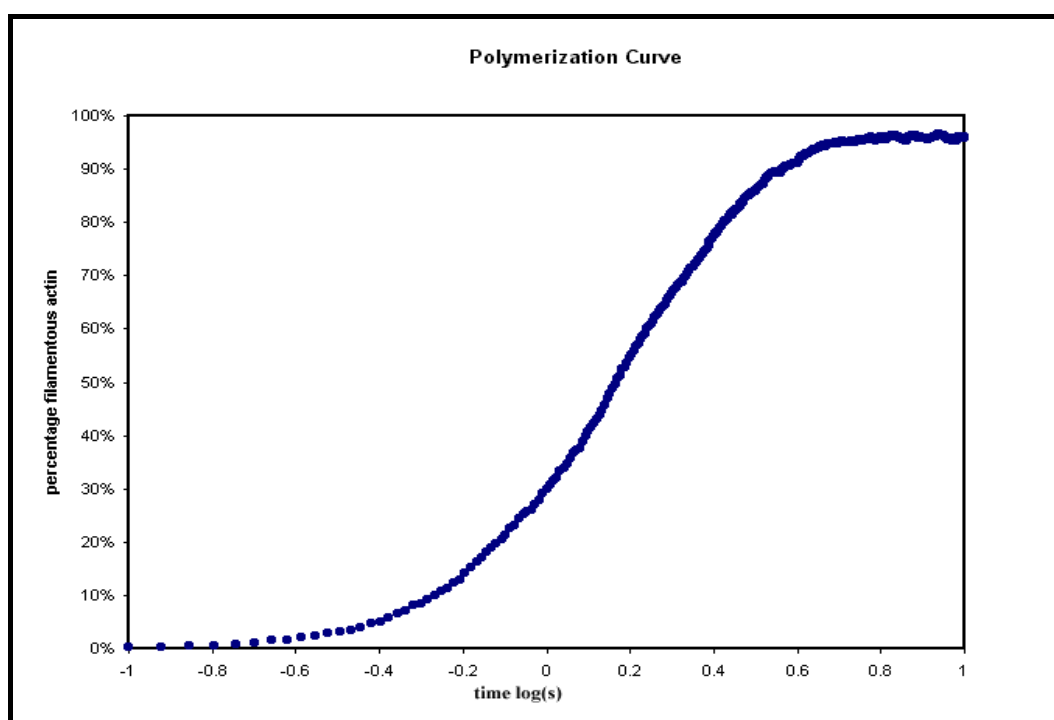


Figure 7.12: Actin Polymerization Curve from nanosimulator

Figure 7.12 shows us that almost all the monomeric actin has been polymerised by five seconds. However, the sample has not yet reached steady state, since ADP-actin has been set (in the .pddf file) to rephosphorylate to ATP-actin, on average, once every 100 seconds. Tracking the state of the monomers, shows that the amount of free ATP-actin passed the theoretical critical concentration (the concentration at which filaments are stable, but do not grow) of $0.18 \mu\text{M}$ (115 monomers) at around four seconds. At the end of the simulation, at ten seconds, there are 65 free-ATP actin monomers (equivalent to $0.1 \mu\text{M}$) still remaining, and the simulation appears to be close to steady state. The smaller than expected number of free monomers may indicate an over estimation of bonding strength in the model.

Filament Size Distribution

Another interesting result is the size distribution of filaments. Unfortunately, due to the small number of filaments (less than thirty) there is significant noise in the data. Figure 7.13 shows the change in the size distribution over the first five seconds (significant change does not occur over the final five seconds of the simulation). The chart shows the number of filaments in a given size range at a particular time (i.e. in the chart below there are five filaments of size 16-24 monomers, at time 0.5 seconds - and this is represented by the dark purple peak below).

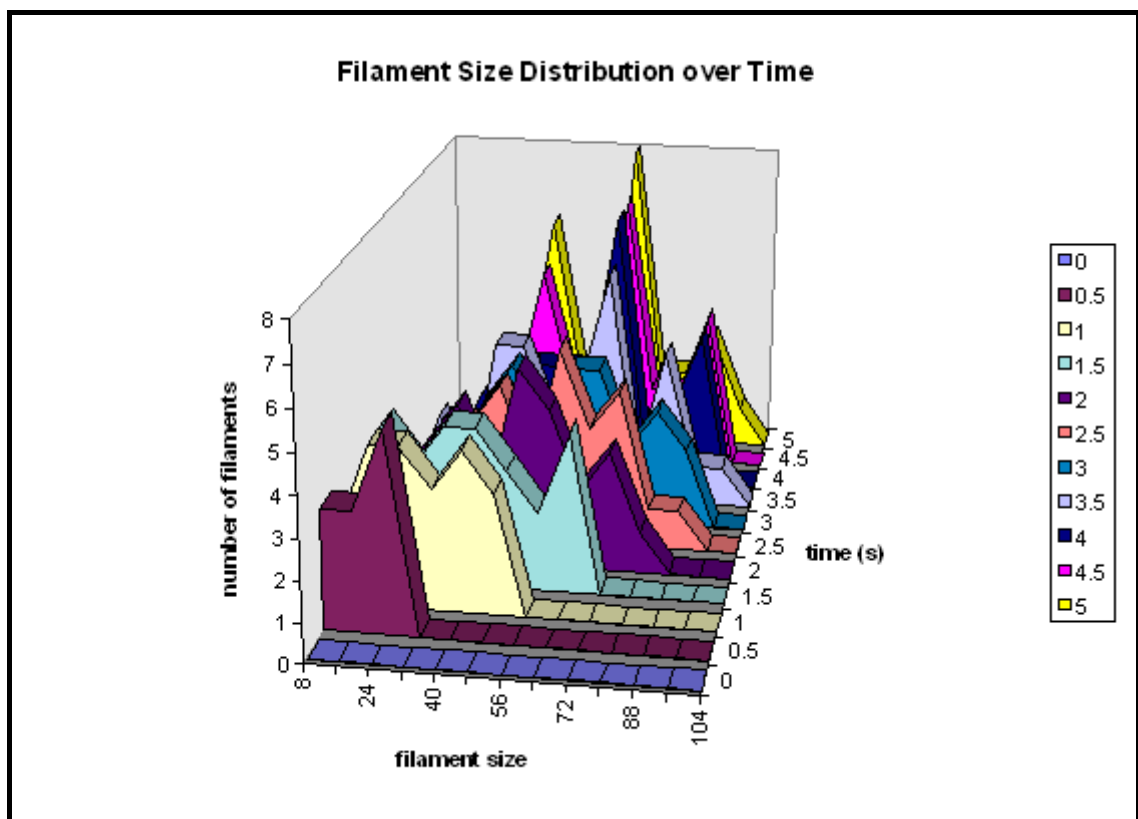


Figure 7.13: Filament Size Distribution

To further clarify the above chart, Fig. 7.14 and 7.15 show the filament size distributions at one second and at five seconds.

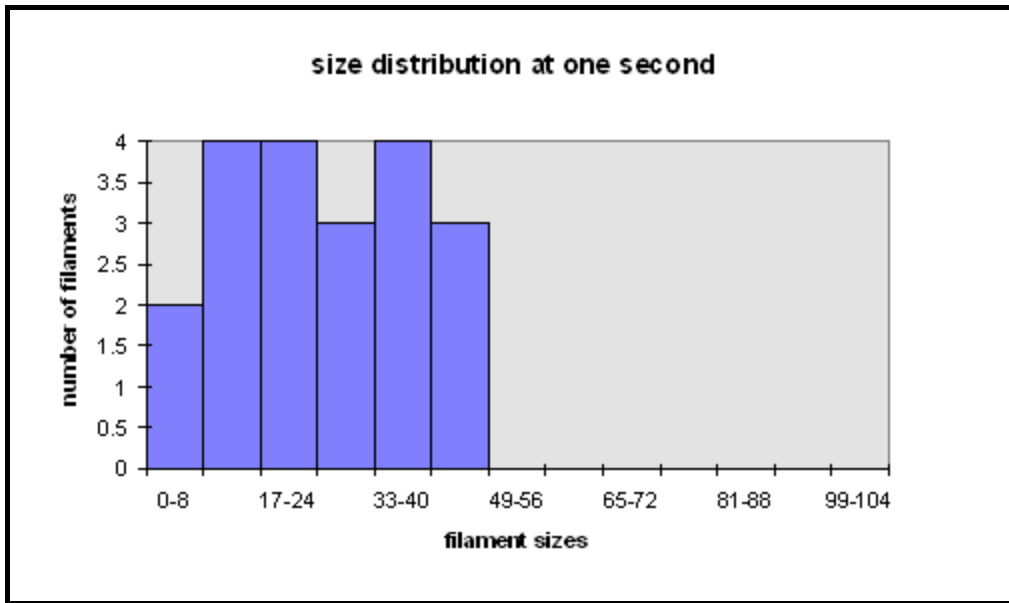


Figure 7.14: Size distribution at one second

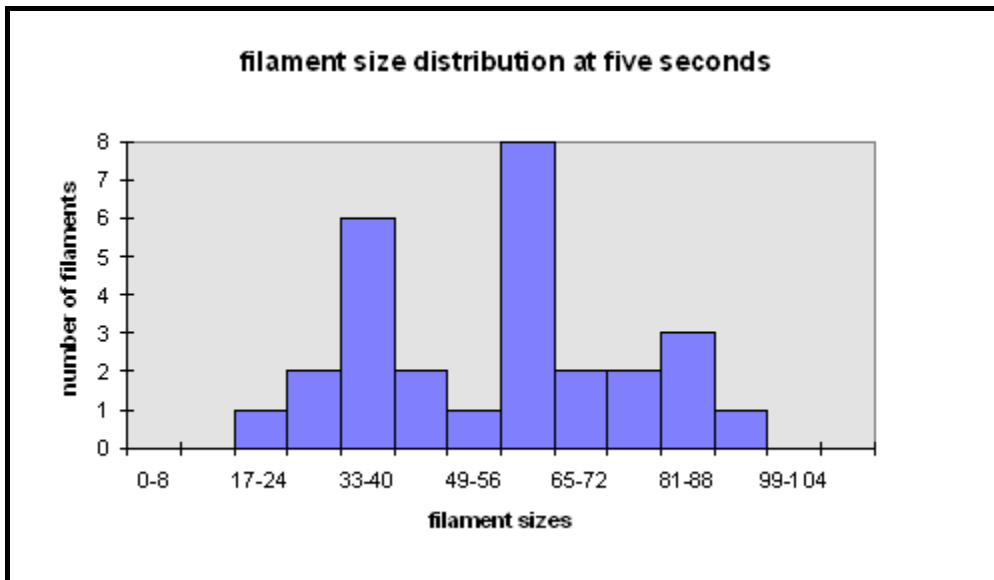


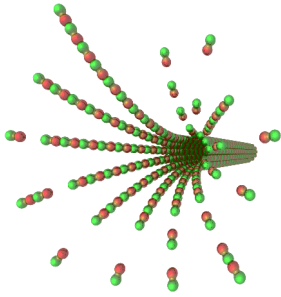
Figure 7.15: Size distribution at five seconds

With a certain amount of imagination these *might* be viewed as the expected Poisson distributions⁹⁶ with a great deal of noise. However strictly all that can be said is that at this scale of simulation, with only dozens of filaments, it is not possible to accurately specify the size distribution of filaments, and that a larger simulation would need to be run on a faster computer to properly study this feature. It is also expected that the simulation would have to be run for longer, to reach steady state, until it could reproduce the experimental results of figure 7.4 .

Conclusion

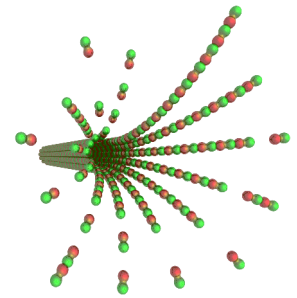
This simulation of actin shows that the nanoscale simulator is capable of modelling many of the characteristics of actin polymerisation during the initial growth phase. The simulator demonstrated the characteristic polymerisation curve, was able to track the initial growth phase of individual filaments, and obtained various statistics and information about the distribution of filaments during growth.

Longer runs in the future should provide even more detailed models of the long-term behaviour of actin. It has not been possible (yet) to run the simulation for long enough to observe features such as the full cycle of actin filament growth, collapse and possible rescue, or the steady state rephosphorylation of ADP-actin into ATP-actin to maintain the growth/collapse cycle. However the ability of the program to run for an extended period of time (in the case of the simulation here, one week) indicates that longer and larger simulation runs should be quite practical in the future, especially given the continuing improvements in computer hardware.



Chapter 8

Modelling Intermediate Structures: Tubulin



Et e fistulis fit robur

Strength also comes from tubes.

- anon

Introduction⁹⁷

Tubulin is another common cytoskeletal protein that has also been studied in great detail⁹⁸ (see Chapter 2). It forms long tubes that are common in almost all eukaryotic cells, and has a role in cellular transport. It plays an essential, if not fully understood, role in cell division.

Part of the work described in this chapter was presented at the 1997 Molecular Biophysics of the Cytoskeleton conference held at the University of Alberta in Banff, Canada⁹⁹.

The Different States of Tubulin

Tubulin comes in two very similar varieties, α -tubulin and β -tubulin, which are usually found bound together to form a tubulin dimer, which is the usual form that polymerises to form microtubules (a third, more stable type ‘ γ -tubulin’ is found in cells, but is less abundant and has not often been used *in-vitro*). The tubulin dimer is the fundamental building block, and how the protein is generally found under normal physiological conditions.

Free tubulin dimers can self assemble, and exhibit the same general behaviour as actin. Microtubules (sometimes abbreviated as mt-s) can self-assemble from free tubulin dimers *in-vitro*, and both dimers and polymers exhibit in broad terms the same general behaviour as actin monomers and polymers. Both assembly and disassembly occurs at a faster rate at one end, called the “plus” end, than at the other, the “minus” end (Fig. 8.1). As with to actin, tubulin dimers come in a high-energy form that favours binding, and a low-energy form that favours

disassociation. Unlike actin however, tubulin uses *guanosine 5'* triphosphate, GTP, rather than *adenosine 5'* triphosphate, ATP, as its energising molecule (though the two are very similar). While the GTP binding site on α -tubulin forms part of its permanent bond with a corresponding β -tubulin monomer, the GTP binding site on β -tubulin can be hydrolysed to GDP (the 'diphosphate' form), causing β -tubulin, and in fact the whole dimer, to become less strongly bound, and more inclined to break away from the microtubule aggregate. Thus, when a dimer is said to be a "GTP-dimer" or a "GDP-dimer", in fact it is the state of the β -tubulin monomer that is being directly specified, although this state affects the entire dimer.

Another difference from actin is that the low energy GDP-tubulin form, when free, does not appear to bind at all to a growing microtubule, while free ADP-actin *will* bind to a growing actin filament, albeit more slowly (see Fig. 8.1¹⁰⁰).

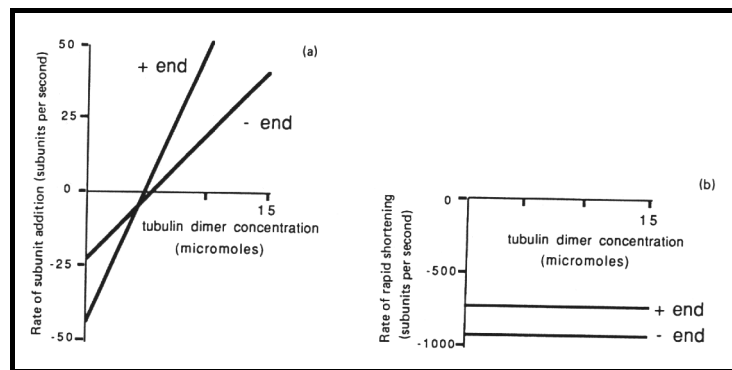


Figure 8.1: Tubulin assembly/disassembly rates (in one particular environment)

As with actin, the difference in assembly and disassembly rates at either end (Fig. 8.1) can induce, at appropriate concentrations, a process known as *treadmilling*, where a particular microtubule may be growing at one end while shrinking from the other.

The structure of microtubules is more complex than that of actin filaments, being a hollow tube, and as a result the processes of assembly and disassembly are not well understood. The standard theory¹⁰¹ is that a *cap* of GTP-dimers forms on the growth end of the microtubule, and that if this cap is lost (due to physical loss of the GTP-dimers, or due to dephosphorylation) the microtubule will switch from growth to very rapid disassembly¹⁰². Under some circumstance whole microtubule populations may switch from growth to disassembly and back again, in a process known as microtubule oscillations, which can be mathematically modelled¹⁰³.

The Structure of Microtubules

The structure of microtubules has been extensively studied using electron microscopy and image analysis, and by observations of their growth under unusual chemical conditions. For example, tubulin can be made to polymerise into sheets or rings by altering the growth environment *in-vitro* using different concentrations of zinc and calcium, and various tubulin affecting drugs such as vinblastine and taxol.

Microtubules come in a variety of structures, the most common of which under normal conditions is a 5-start, period 13 helical tube of dimers¹⁰⁴, which can be visualised as being made up of 13 offset strands, or “*proto-filaments*” longitudinally (Fig. 8.2^{***}).

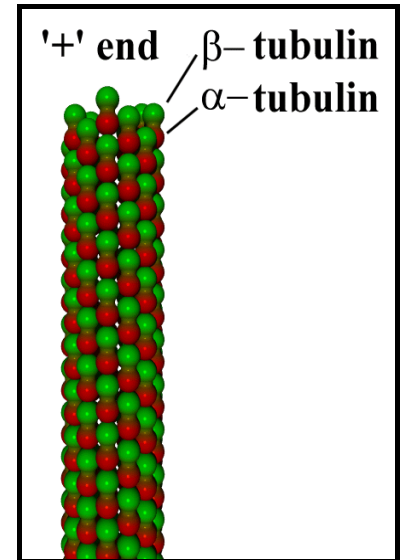


Figure 8.2:
Microtubule Structure:
α-tubulin = red
β-tubulin = green

Tubulin Polymerisation and the Oosawa Model

However, the tubulin polymerisation curves derived from experiments are not well described by this type of model¹⁰⁵, due to the difficulty of determining the nucleation process, and the size of the “*critical nucleus*” (the minimum polymer size at which growth is more probable than disassembly)¹⁰⁶. They are better described analytically using a two-stage nucleation model¹⁰⁷, or with even greater accuracy using a multi-stage nucleation model¹⁰⁸.

*** Note that this model, and the ones below, are static models for illustrative purposes, constructed using povray or other programming tools, and are not output from the simulator unless explicitly stated.

Suitability of Tubulin for Simulation

Microtubule self-assembly is more complex than that of actin. This makes it a more interesting problem to study, as well as presenting more difficulties in determining an accurate model. Since the size, stability and nature of the “GTP-cap” is not known precisely, a given simulation must make assumptions, and choose a particular capping model. Again, this makes for a more interesting area of study, as different cap models can be trialed in the simulator. The trade-off is that the accuracy of each model cannot be easily gauged as the correct model is not known.

A complication arises due to the change in structure of disassembling microtubule ends. These are observed to break up into a hydra-like spray of individual protofilaments (Fig. 8.3): While there has been some dispute as to how much of this is due to experimental conditions, and how much is intrinsic to microtubule disassembly¹⁰⁹, it seems likely that this does play some role in the rapid disassembly observed in microtubules.

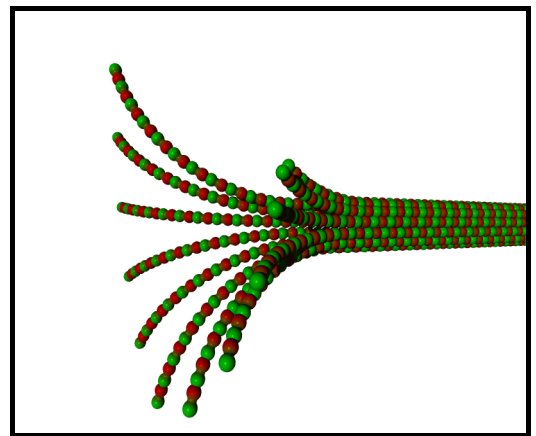


Figure 8.3: Disassembling microtubule end with splayed proto filaments.

Simulator Limitations regarding Microtubules

Since microtubule structure may play a role in both assembly and disassembly, the shape and nature of the ends of the microtubule may effect the rates of assembly and disassembly, and also the possibility of rescue¹¹⁰ (i.e. the splayed shape of the disassembling microtubule shown in figure 8.3 may make it more difficult for it to recover into a stable growth tip.) This presents a challenge to the simulator. While the program architecture and the .pddf modelling language both allow for conformational change modelling of the type required to model the hypothesized changes to end structure, this is not currently supported by the nanosimulator program. Due to this limitation, it is not possible to model those theories that involve structural change with

the current version of the simulator. In addition, it is not currently possible to model the flexibility/elasticity observed in bending microtubules¹¹¹, or in the splayed disassembling tips¹¹².

Physical Structure of Growing Ends

Another issue is the structure of growing ends, with recent research indicating the probable growth structure as an extending curved sheet, that closes up into a tube (Fig. 8.5,8.6), rather than a simple 'flat' ended top (Fig. 8.4):

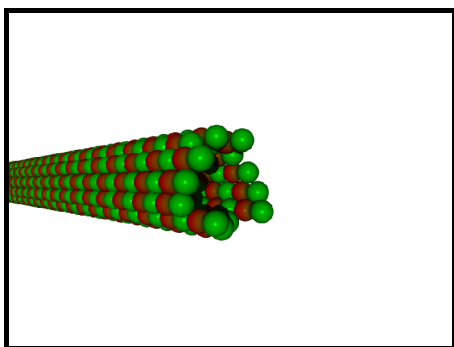


Figure 8.4: 'Flat' Growth Tip

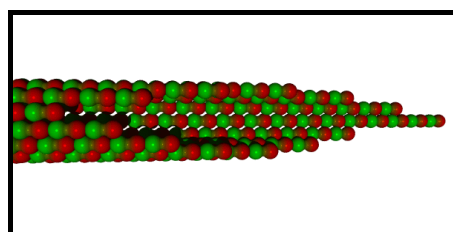


Figure 8.5: 'Sheet' Growth Tip

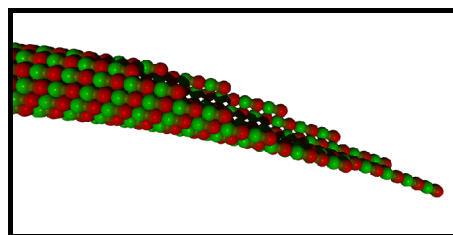


Figure 8.6: 'Sheet' Growth Tip shown from a different angle.

Experimental observation of fast-frozen growth tips also indicates that there is a force resisting the curling of the microtubule tip into a tube, which leads to the growth tip being splayed into a flatter shape than would be indicated by the curvature of the closed microtubule¹¹³. (Nb this flattening is *not* fully represented in Fig. 8.5 & 8.6)

The simulator is able to model the structure of the bonds, but not (currently) the changes in inter-dimer angles that initially create the flattened sheet, and later cause the sheet to close up. However, while these structural changes are doubtless important in microtubule formation, the mathematical model of the simulator is unlikely to be strongly affected, since it deals with dimers on an individual basis.

Setting up the Simulation

Experimental studies of tubulin have provided accurate details of the assembly and disassembly rates of the growing microtubule ends. To test the simulation, an arbitrary *in vitro* environment is chosen for which good experimental evidence is available. The current simulation uses data from Walker et al¹¹⁴, who used a standard experimental environment conducive to microtubule growth, at 37°C, and obtained the following figures (Table 8.1, also used in Fig. 8.1):

	GTP-Tubulin		GDP-Tubulin (hypothesised)	
	+ end	- end	+ end	- end
$k_{\text{on}} (\mu\text{M}^{-1}\text{s}^{-1})$	8.9	4.3	0	0
$k_{\text{off}} (\text{s}^{-1})$	44	23	733	915

Table 8.1 Tubulin association and disassociation rates¹¹⁵

There are some important differences between the tubulin data in Table 8.1 the equivalent actin data (Table 7.1).

- Only free GTP-tubulin is able to bind to a microtubule, whereas free GDP-tubulin cannot bind at all (under normal conditions). In actin both ATP and ADP forms bind.
- The disparity between association and disassociation rates is greater than for actin.
- The relationship between +end and -end disassociation rates is somewhat controversial. Some researchers hypothesise a structural change propagating along the length of the microtubule, causing rapid *synchronous* disassembly to occur at both ends¹¹⁶, other researchers do not observe such an effect¹¹⁷.
- There is considerable discussion over whether the difference between growing and disassembling microtubule tips is due to chemical effects (a GTP cap of debated length) or physical effects (a frayed end of debated shape)¹¹⁸.
- The ultra-rapid disassembly of the microtubule is sometimes halted with the microtubule returning to an assembling state, in a process known as ‘rescue’. While a similar process may occur with actin, it is not as dramatic.

Using these figures, a .pddf file is constructed, that uses a four-state model for tubulin; unbound and phosphorylated, bound and phosphorylated, bound and dephosphorylated, and unbound and dephosphorylated. This represents the presumed cycle of binding, dephosphorylation, unbinding, and rephosphorylation (Fig. 8.7).

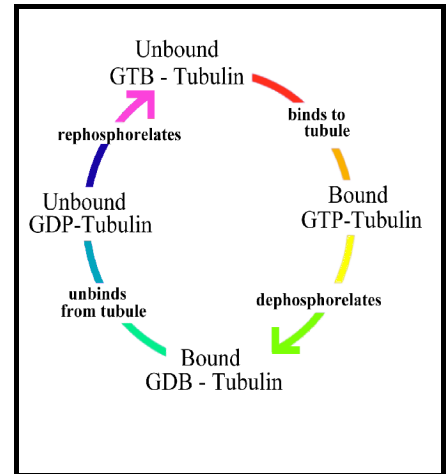


Figure 8.7: Tubulin Cycle

Rephosphorylation requires an energy input into the system, and can occur when free unbound GTP replaces the phosphorus-depleted GDP of a free tubulin protein. This occurs within the cell, and can be simulated *in vitro* simply by providing free molecules of unattached GTP.

Structure

A simple model of the dimer is used, where the dimer is represented as two permanently attached 2nm radius spheres, giving the unit dimensions of 8nm by 4nm by 4nm. This is close to, but not the same as, recent measurements of tubulin dimers as being 46 x 80 x 65 Å¹¹⁹.

Finite State Machine Definition

The model used assumes five possible states for tubulin:

1. Free GTP-tubulin
2. Free GDP-tubulin
3. Bound GTP-tubulin (exposed on the '+' end of the microtubule)
4. Bound stable GDP-tubulin
5. Bound unstable GDP-tubulin (exposed on the '+' end of the microtubule)

The model used is the simple cap model, where GTP-tubulin is hydrolyzed to GDP-tubulin after the ‘top’ link (i.e. the axial β -tubulin link) is bound¹²⁰. In order to simulate the extra instability that occurs in this model when GDP-tubulin is exposed at the ‘+’ end of the microtubule, two GDP-tubulin states are used, a stable and an unstable form.

The events which change the state of the dimer are:

- Binding a free GTP-monomer to the top (‘+’) link turns it to bound stable GDP-tubulin
- Binding a free GTP-monomer elsewhere turns it to bound GTP-tubulin
- Breaking all links of GTP-tubulin turns it to free GTP-tubulin
- Breaking all links of (any) GDP-tubulin turns it to free GDP-tubulin
- Breaking the top link of stable GDP-tubulin turns it to unstable GDP-tubulin

Deriving Figures for the .pddf File

Using the experimental figures of table 8.1 in the simulator requires some further computation. To set up the .pddf files, we need to have binding and breaking values on an individual protein basis. The problem is more complex than with actin, since it is impossible to ignore the multiply bound state of tubulin dimers, and the dimers have four or even six possible binding sites rather than two, since they can bind at the sides as well as forward and back.

For the sake of simplicity, the geometrically simplest (and the most common) species of microtubule is used in this simulation, being the seamless 13 protofilament microtubule¹²¹.

A further difficulty lies in the uncertain structure of the microtubule ends. Since some dimers are likely to be held more tightly than others, and it is unclear exactly how many dimers at the ends will be held by one, two, or three bonds, it is difficult to calculate *a priori* the individual dimer link breakage probabilities. For the same reason, while any of the protofilament ends can theoretically bind free dimers, obviously the more exposed monomers will be more likely to bind than those with only one available link, or those that are partially blocked by their neighbours.

Another important point to note is that it is not possible to distinguish, on the basis of the experimental evidence available, differences in binding rates depending on the state of the

tubulin protein already bound at the tip of the microtubule (whether phosphorylated or not). Although it might be suspected that such differences exist (and if they do, they can be modelled using the simulator), this *particular* .pddf model does not explicitly include them.

Due to these difficulties, a number of different models based on the figures in Table 8.1 were trialed in the simulator. The method used was to estimate the dimer probabilities with the aid of the following equations:

$$\begin{aligned} p(\text{binding}) = & p(\text{top collision}) * p(\text{top bind}) + p(\text{bottom collision}) * p(\text{bottom bind}) \\ & + p(\text{left collision}) * p(\text{left bind}) + p(\text{right collision}) * p(\text{right bind}) \end{aligned} \quad (8.1)$$

This equation describes the probability, per dimer, of binding another dimer in terms of the probability of various types of collision (side, +end, -end), and the probability of binding given such a collision.

To estimate breakage rates more assumptions must be made about the state of the microtubule ends. The total off rate will be determined by the number of dimers available to break and the number of bonds to be broken. Using the “order of magnitude” breakage heuristic described below, the probabilities for a dimer breaking away is that of the strongest bond (side, +end or -end) divided by 10 for every link beyond the first.

For example, if the “sheet growth tip” model is employed, it is assumed that on average one dimer will be singly bound, eleven doubly bound, and one triply bound (e.g. the tip in Fig. 8.5).

With the further assumption that the bond strength is the same for side and +end bonds (this is a large assumption) then, using the multiple link breakage model described below, the per-link breakage rate could be found from:

$$\begin{aligned} p(\text{dimer breakage/microtubule}) &= p(\text{single bond breakage}) * (1 + 11/10 + 1/100) \\ &= p(\text{single bond breakage}) * 2.11 \end{aligned} \quad (8.2)$$

Since the total off rate for the microtubule is 44 dimers/s, or 0.0044/100 μ s, the probability of breakage pre bond for this model can be found from equation 8.2, using $p(\text{dimer breakage/microtubule}) = 0.0044$, which gives $p(\text{single bond breakage}) = 0.0044/2.11$, or 0.0021/100 μ s for a single dimer link.

Since such figures are highly model dependant, a number of different models were trialed. It should be noted that behaviour will be rather different during nucleation (before the microtubule has closed to become a tube) as there will be far more partially-bound dimers available to break off.

The Multiple Link Breakage Model

Multiple link breakage is handled using the standard ad-hoc heuristic mentioned in Chapter 5, where the overall breakage probability is made equal to the smallest single link breakage probability, lowered by an order of magnitude for each extra link. This makes the breakage probability of a dimer held within the walls of an intact microtubule extremely low, but still non-zero. (While tubulin dimers generally do not break away from within intact microtubules, there is some evidence that this does occasionally occur¹²².)

It is debatable as to whether this is an adequate model for the complex interactions in varying geometries of dimers at the growth ends of microtubules; certainly far more sophisticated models have been used, albeit on the smaller scale of a single microtubule tip of constrained geometry¹²³.

Handling Nucleation

A frequently observed feature of *in vitro* tubulin polymerisation is the lag phase that occurs prior to a rapid polymerisation phase (as with actin and many other self-assembling proteins). This lag phase is due to the instability of tubulin aggregates when they are first assembling. Until they reach a certain size the filament is more likely to fall apart than to grow. Once over this critical size, the filament tends to grow strongly. As more and more filaments pass this initial

barrier, and turn into steadily growing microtubules, bulk polymerisation builds up speed. Further evidence that this is the reason for the lag phase lies in the observation that the lag phase can be avoided altogether if the solution is initially *seeded* with microtubule fragments, or helper proteins that act as microtubule growth centres.

Lag behaviour can grow naturally out of the binding and breaking values of the dimers. Unfortunately, this leads into the difficulty in modelling breakages from multiple links, with the result that breakage model chosen, and the values of the specific links, have a very significant impact on the nucleation lag time; in effect the user can implicitly *set* the lag time simply by choosing an appropriate breakage model, without necessarily affecting the normal addition and subtraction probabilities of the microtubule tips as a whole.

It is also possible that larger aggregates have a physically stabilising affect on the newly added dimer. As this effect may not be present when two dimers initially join, it may be necessary to set a lower initial binding probability to simulate this added instability for very small aggregates.

The necessity of estimating these various binding factors, as well as the ad-hoc nature of the binding model, unfortunately act to limit the predictive power of the simulator in this case.

Simulating Different Models

A number of different models were trialed, making different assumptions about the nature of microtubule growth. In order to obtain rapid results, a high concentration of tubulin was used ($15\mu\text{M}$), and a large time step ($100\mu\text{s}$) in all simulation runs.

Initial Conditions

The simulator was run in collision test mode, and determined that at a concentration of $10\mu\text{M}$ (used for ease of calculation), a stationary dimer would collide with other dimers, in a simulation field of 256nm cubed on average 5.3 times per time step (with an estimated error of 5% based on three 10s trials).

Model A: An Initial Estimate

There are a great many different variables that a model must consider. This first model makes the following (somewhat simplistic) assumptions in simulating 'standard' 13 proto filament microtubules.

It is assumed that:

1. Linkage from the '-', or 'bottom' direction, is more likely for free dimers than linkage in the side directions.
2. Linkage opportunities (i.e. collisions) are evenly distributed between available sites.
3. Collisions are evenly distributed over all 13 exposed tip dimers.
4. Most breakage comes from singly held dimers from a ragged tip.
5. On average there is one singly held dimer on the growing microtubule tip, the others being doubly bound (cf Fig. 8.5).
6. The two side bonds always are of equal strength.
7. Only GTP tubulin binds, and free GDP-tubulin does not (there is some evidence that it binds weakly, but this model ignores this effect).
8. Assume that free GTP tubulin binds as well as bound GTP tubulin (for estimating polymer to polymer binding).

The figures below use a concentration of 10 μ M for ease of calculation (remembering that the simulation uses the same probability figures for all concentrations, so that different concentrations can be simulated without needing to recalculate anything). The '+' direction refers to the '+' tip of the microtubule, hence at the '+' end of the microtubule the '+' end of a bound tip dimer binds to the '-' end of the free dimer.

Modifying table 8.1 for 100μs timesteps and 10μM gives:

	Bound GTP-tubulin dimer		GDP-tubulin (hypothesised)	
	+ end	- end	+ end	- end
$k_{on} (10\mu M)^{-1}(100\mu s)^{-1}$	0.0089	0.0043	0	0
$k_{off} (100\mu s)^{-1}$	0.0044	0.0023	0.0733	0.0915

Table 8.2 Scaled tubulin association and disassociation rates

Assumption 3 combined with the simulated collision rate of 5.3 collisions/second, means that as a first approximation the on-rates per dimer can be found by dividing the above k_{on} rates by (13 tip dimers * 5.3 collisions/sec) = 68.9 (evenly distributed collisions) .

Using assumptions 4&5 (most breakages come from a singly bonded dimer, and on average there is one such dimer) in equation 8.3, results in a conversion factor of 2.11. Dividing the above k_{off} rates by 2.11 give the breakage probability for an individual singly-bound dimer.

Modifying table 8.2 to give binding probabilities per collision per dimer, and breakage probabilities per (singly bound) dimer.:

	Bound GTP-tubulin dimer		GDP-tubulin (hyp.)	
	+ end	- end	+ end	- end
$p(on) (dimer)^{-1}(collision)^{-1}$	0.00013	0	0	0
$k_{off} (100\mu s)^{-1}$	0.002085	0.00011	0.0347	0.0434

Table 8.3 Tubulin association and disassociation per dimer

Based on assumptions 1&2&6, and equation 8.1, the following calculation is made for a top ('+' end) dimer. Since there is no chance of binding from the bottom for an established microtubule end, the $p(bottom\ bind) = 0$, so:

$$\begin{aligned}
 p(bind\ for\ +\ end\ dimer) &= 0.25 * (top) + 0.5 * (side) + 0.25 * (bottom = 0) \\
 &= 0.25 * (top) + 0.5 * (side)
 \end{aligned}
 \tag{8.3}$$

Since Assumption 1 is that side links are relatively weaker, the following figures are chosen, satisfying Equation 8.3.

$$\begin{aligned} p(\text{top} +) &= .00038 \\ p(\text{side}) &= .00007 \end{aligned}$$

Using the same logic for the bottom link (of the dimer at the '-' end of the microtubule) provides the following equation:

$$\begin{aligned} p(\text{bind for - end}) &= 0.25 * (\text{top} = 0) + 0.5 * (\text{side}) + 0.25 * (\text{bottom}) \\ &= 0.5 * (\text{side}) + 0.25 * (\text{bottom}) \end{aligned} \quad \begin{matrix} (8 \\ .4 \\) \end{matrix}$$

Since Assumption 6 is that side links are always the same, the value for $p(\text{bottom})$ is constrained to be:

$$p(\text{bottom}) = 0.00011$$

Setting realistic breakage rates are difficult, due to the multiple-link-breakage problem. As a first order approximation it is assumed that the probability of breakage is equal for all links, and the values from table 8.3 are used directly. To summarise, the breakage probabilities for a bound dimer held by a single link are:

$p(\text{bottom link breakage})$ unstable GDP-tubulin	= 0.035
$p(\text{side link breakage})$ unstable GDP-tubulin	= 0.035
$p(\text{bottom link breakage})$ GTP-tubulin	= 0.0021
$p(\text{side link breakage})$ GTP-tubulin	= 0.0021
$p(\text{top link breakage})$ stable GDP-tubulin	= 0.0011
$p(\text{bottom link breakage})$ stable GDP-tubulin	= 0.0011
$p(\text{side link breakage})$ stable GDP-tubulin	= 0.0011

(Note that GTP-tubulin and unstable GDP-tubulin are *never* bound at the top)

This gives the following .pddf file:

Tubulin Dimer .pddf File

```
Model Dimer
{
    #
    #           Define the tubulin dimer as two 2nm radius spheres
    #

    Sphere top      2 <0,0,2>  red
    Sphere bottom 2 <0,0,-2> lightred

    #
    #           Define the four linkage sites
    #

    site  bottom  <0,0,-4>
    site  left    <0,2,1.53846> <0,1,0>
    site  right   <0.929446,-1.7709,-1.53846> <0.929446,-1.7709,0>
    site  top     <0,0,4>

    #
    #           Define the five states
    #

    State GTPunbound

    State GTPbound

        colour top      = green
        colour bottom = lightgreen

    State GDPbound

        colour top      = blue
        colour bottom = lightblue

    State GDPunbound

        colour top      = magenta
        colour bottom = magenta

    State GDPunbound    # another new state with no physical changes

        colour top      = gray
        colour bottom = black

    #
    #           Define the events that change the dimer state
```

```

#

# *** bindings ***

Event bind { bottom top }
    GTPunbound -> GTPbound

Event bind { left right }
    GTPunbound -> GTPbound

Event bind { right left }
    GTPunbound -> GTPbound

Event bind { top bottom }
    GTPunbound -> GDPSbound

Event bind { top bottom }
    GTPbound -> GDPSbound

Event bind { top bottom }
    GDPUnbound -> GDPSbound

# *** breakages ***

Event break { }
    GTPbound -> GTPunbound

Event break { }
    GDPUnbound -> GDPunbound

Event break { }
    GDPSbound -> GDPunbound

Event break { top }
    GDPSbound -> GDPUnbound
# model unstable + end GDP dimers...

# *** random events ***

Event random 0.9
    GDPunbound -> GTPunbound
# a large chance of rephosphorelating
# free GDP-tubulin dimers. see note below.

}

#
# Binding/Breaking probabilities
# (nb. figures rounded)

Binding Dimer top    GTPunbound    to Dimer bottom GTPunbound = 0.0004    1
Binding Dimer left   GTPunbound    to Dimer right  GTPunbound = 0.00007  1

Binding Dimer top    GTPunbound    to Dimer bottom GTPbound  = 0.0004    1
Binding Dimer bottom GTPunbound    to Dimer top    GTPbound  = 0.0004    1
Binding Dimer left   GTPunbound    to Dimer right  GTPbound  = 0.00007  1

```

```

Binding Dimer right  GTPunbound  to Dimer left  GTPbound = 0.00007  1

Binding Dimer top    GTPunbound  to Dimer bottom GDPSbound = 0.0004  1
Binding Dimer bottom GTPunbound  to Dimer top    GDPSbound = 0.0004  1
Binding Dimer left   GTPunbound  to Dimer right  GDPSbound = 0.00007  1
Binding Dimer right  GTPunbound  to Dimer left  GDPSbound = 0.00007  1

Binding Dimer top    GTPunbound  to Dimer bottom GDPbound = 0.0004  1
Binding Dimer bottom GTPunbound  to Dimer top    GDPbound = 0.0004  1
Binding Dimer left   GTPunbound  to Dimer right  GDPbound = 0.00007  1
Binding Dimer right  GTPunbound  to Dimer left  GDPbound = 0.00007  1

# n.b. following figures have a rounding error in the breakage prob; should be .0021
# rather .0022 (from Table 8.3)

Binding Dimer top    GTPbound  to Dimer bottom GTPbound = 0.0004  0.0022
Binding Dimer left   GTPbound  to Dimer right  GTPbound = 0.00007  0.0022

Binding Dimer top    GTPbound  to Dimer bottom GDPSbound = 0.0004  1
Binding Dimer bottom GTPbound  to Dimer top    GDPSbound = 0.0004  0.0022
Binding Dimer left   GTPbound  to Dimer right  GDPSbound = 0.00007  0.0022
Binding Dimer right  GTPbound  to Dimer left  GDPSbound = 0.00007  0.0022

Binding Dimer top    GTPbound  to Dimer bottom GDPbound = 0.0004  1
Binding Dimer bottom GTPbound  to Dimer top    GDPbound = 0.0004  1
Binding Dimer left   GTPbound  to Dimer right  GDPbound = 0.00007  0.0022
Binding Dimer right  GTPbound  to Dimer left  GDPbound = 0.00007  0.0022

Binding Dimer top    GDPSbound  to Dimer bottom GDPSbound = 0.00007  0.0011
Binding Dimer left   GDPSbound  to Dimer right  GDPSbound = 0.00007  0.0011

Binding Dimer top    GDPSbound  to Dimer bottom GDPbound = 0.0004  0.036
Binding Dimer bottom GDPSbound  to Dimer top    GDPbound = 0.0004  1
Binding Dimer left   GDPSbound  to Dimer right  GDPbound = 0.00007  0.036
Binding Dimer right  GDPSbound  to Dimer left  GDPbound = 0.00007  0.036

Binding Dimer top    GDPbound  to Dimer bottom GDPbound = 0  0.036
Binding Dimer left   GDPbound  to Dimer right  GDPbound = 0  0.036

#
# some environment constants
#

Temperature 310      # kelvin - physiological temperature
Viscosity 0.69       # approximate viscosity for water at 37 degrees Celsius
TimeScale 100        # 100 microseconds per program cycle
Mix Dimer 100%

```

Results for Model A

Initially, this model had a (relatively) low rephosphorylation probability - a free GDP-dimer would become a GTP-dimer, on average, once a (simulated) second. The immediate result of this was that the entire simulation 'burnt out' in a fraction of a (simulated) second, with *only* GDP-dimers present. Almost every GTP-dimer entered into a short term binding that dephosphorylated it, after which the fragments fell apart, returning free GDP-dimers into the pool of available dimers. With a very high turnover, and a rapidly diminishing pool of free GTP-dimers to choose from, no nucleating fragments of over size 6 were found.

To overcome this, the rephosphorylation rate was increased dramatically, until (as in the file above) it stands at a 0.9 chance *per cycle*. (I.e. the mean time for rephosphorylation is around 110 μ s). This is unrealistic, but was required for the present model. A more realistic model, which includes a time lag between binding and dephosphorylation would fix this problem, In the mean time the effect on assembly should be much the same.

Once this change was made, the simulation was run again. This time the model created a very large number of very small fragments. Only once did a large microtubule begin to form - it reached a size of 84 dimers - however the average size was under 6, and most of the time all fragments were under 36 dimers in size. The simulation ran for 10 virtual seconds, but the microtubule size distribution did not change much after the first second.

Model A: Results

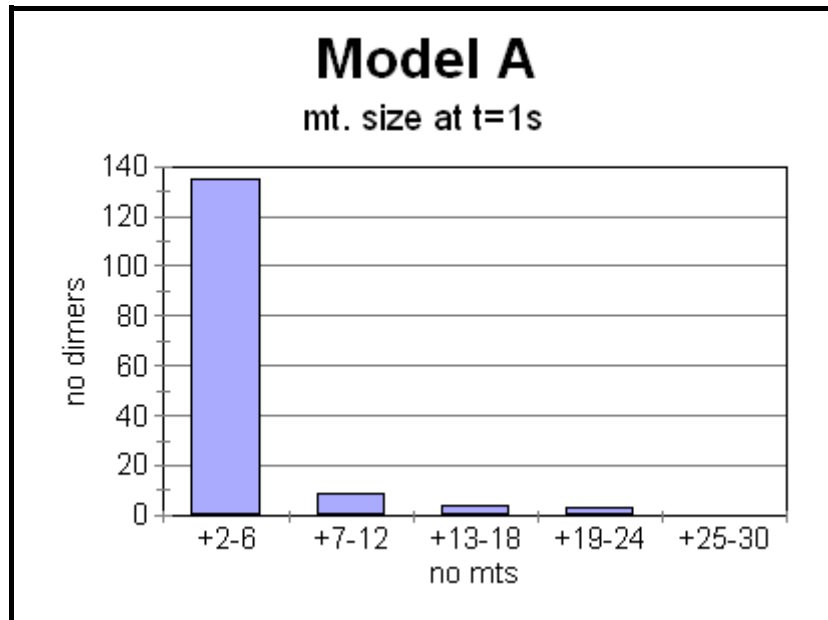


Figure 8.8: Model A size histogram

The simulated microtubule (mt) size distribution at t=1s:

The polymerisation curve was likewise largely unchanged after 1s.

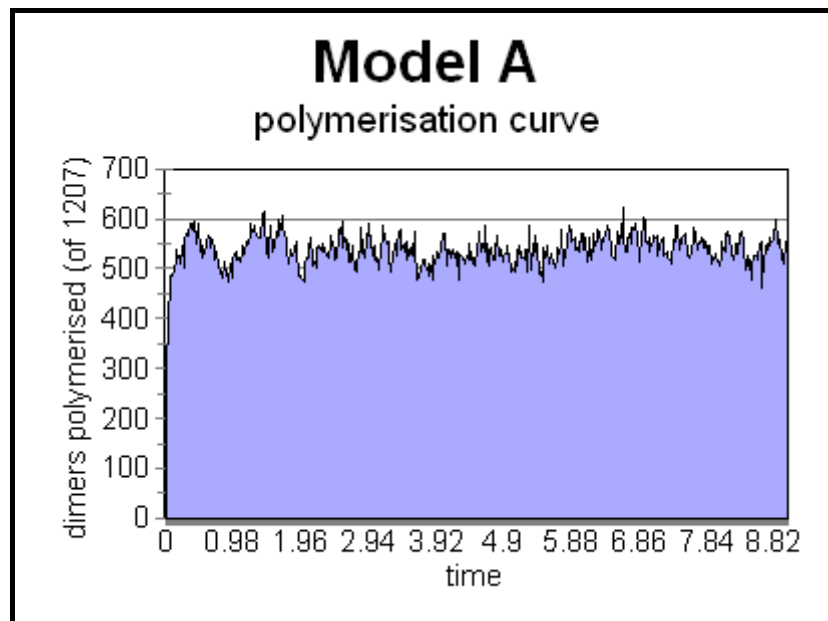


Figure 8.9: Model A: polymerisation curve.

The state of the simulation at one second is shown in figure 8.10. It consists of a large number of small to medium sized fragments. The colour code (with alpha tubulin usually represented as slightly lighter than beta tubulin), used throughout this section, is :

- red - free GTP tubulin
- black - free GDP tubulin
- green - bound GTP tubulin
- blue - bound stable GDP tubulin
- purple - bound unstable GDP tubulin

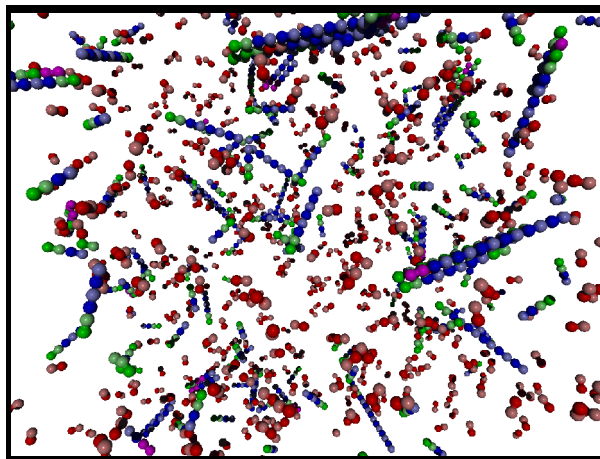


Figure 8.10: Model A at one second.

Model A: Conclusion

This model represents a system where nucleation is too easy, and extended polymerisation too difficult. While many dimers can form the initial two or three dimer aggregates necessary to start building a microtubule, it is not possible for the microtubule to grow particularly large, since dimers break away too easily. Examination of the program trace logs indicates that the few larger aggregates that did form were largely the result of smaller aggregates combining, rather than accretion of dimers. At 15 μ M concentration, a microtubule, once nucleated, should grow at around 100 dimers/second (Fig. 8.1) - obviously no aggregate did this, despite there being a number of reasonably large aggregates that might have displayed such behaviour had conditions been favourable.

Model B: Decreasing the Breakage Probabilities

The previous simulation failed to generate any large microtubules. One possible explanation for this is that the fragments are too friable, since the breakage chance is too large. In order to investigate this, the breakage probabilities were divided by two. This corresponds to a model of ragged microtubule tips where *two* dimers are held by a single bond.

The .pddf file is identical to Model A, except that (non-unity) breakage probabilities are divided by two,

p(bottom breakage) unstable GDP-tubulin	= 0.018
p(side breakage) unstable GDP-tubulin	= 0.018
p(bottom breakage) GTP-tubulin	= 0.0011
p(side breakage) GTP-tubulin	= 0.0011
p(top breakage) stable GDP-tubulin	= 0.00055
p(bottom breakage) stable GDP-tubulin	= 0.00055
p(side breakage) stable GDP-tubulin	= 0.00055

Model B: Results

The simulation was run for 6 simulated seconds. Again, no very large polymers were grown, although more material was polymerised and slightly larger nucleating fragments appeared. A measure of turnover can be obtained by noting that in six seconds over 18,000 aggregates were created and destroyed, implying that each dimer, on average, joined and broke apart from an aggregate 15 times.

The simulation has attained steady state by the one second mark. At that stage, the size histogram is:

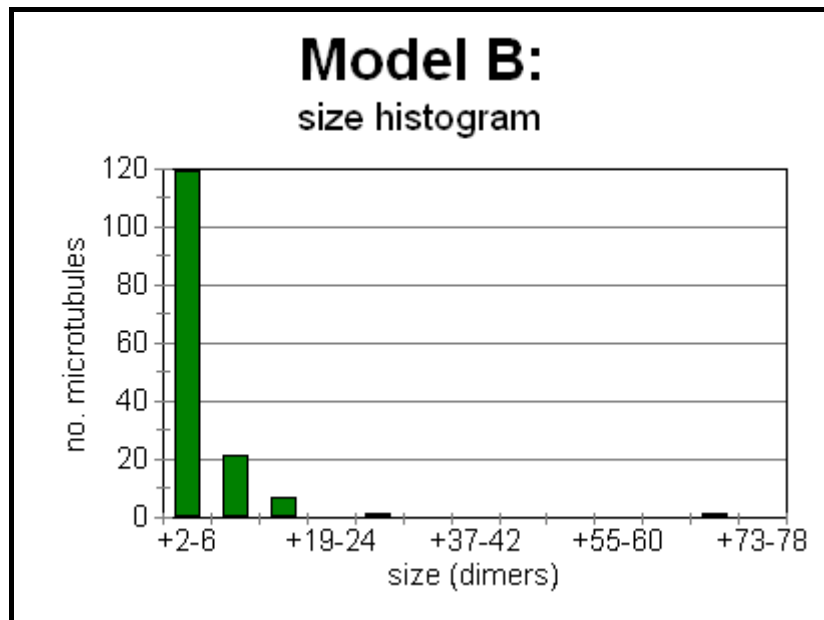


Figure 8.11: Model B: size histogram

The polymerisation curve was also stable within a narrow range after its initial rise.

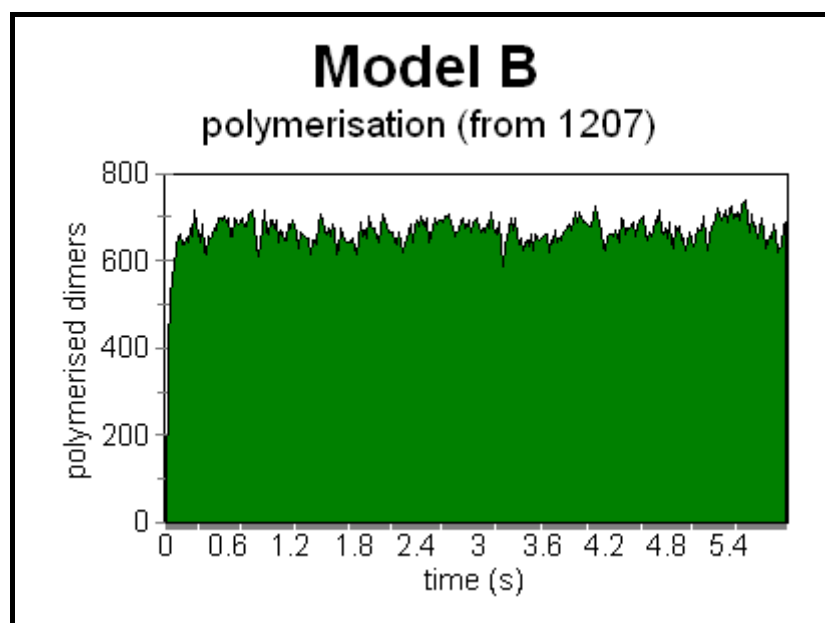


Figure 8.12: Model B: polymerisation curve.

Figure 8.13 shows the simulation field at one second.

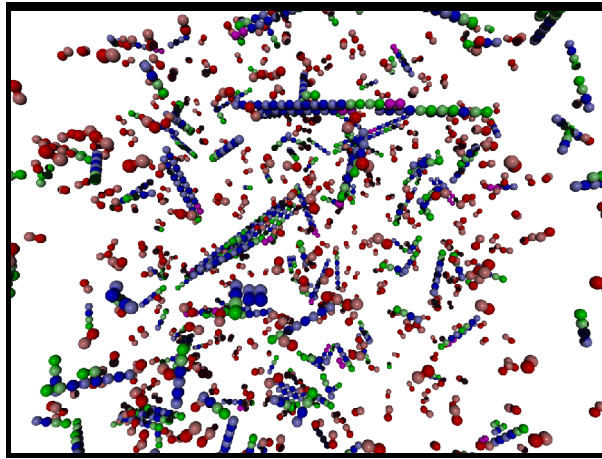


Figure 8.13: Model B at one second

Model B: Conclusion

The second model suffered from the same problems as the first. Nucleation was too rapid, while microtubules were too unstable. Again, most large aggregates probably resulted from the merging of smaller aggregates, rather than gradual accretion. An interesting feature of both this and the previous model, is that the free dimers that were left represent a molarity of around 5-7 μM of free GTP-tubulin (since almost all free dimers are GTP-tubulin). This is refreshingly close to the theoretical value of 5 μM predicted for the steady state critical concentration, at which the amount of tubulin dimers accreting is the same as the amount breaking off (Fig. 8.1). The lower breakage figures did however give rise to slightly more, and slightly larger, nucleating fragments.

Model C: Increasing Binding Probabilities

There is a possibility that underestimation of the binding probabilities is hindering microtubule growth. To test this hypothesis the binding probabilities of the Initial Estimate Model A are increased by a factor of 10:

$$\begin{aligned}p(\text{top } +) &= .004 \\p(\text{side}) &= .0007 \\p(\text{bottom}) &= 0.0011\end{aligned}$$

Note that this change presupposes an error in the original estimation of the number and type of collisions occurring at the tip of the microtubule. No such error is believed to have been made; this model is simply to test the possibility.

Model C: Results

The solution rapidly degenerated into a large number of small to medium size fragments, possibly corresponding to a gel state. Since very few free GTP-tubulin dimers were available, the only growth path available was binding between aggregates, which did occur, and presumably would eventually have led to large microtubules, given sufficient time. However, in order to observe whether inter-aggregate binding was necessary, the simulation was run again *without* inter-aggregate binding being allowed, and a similar result (many small fragments) was obtained. The results reported here are from this second simulation.

The size histogram, unlike the histograms for model A and model B, contains more medium size fragments than small fragments.

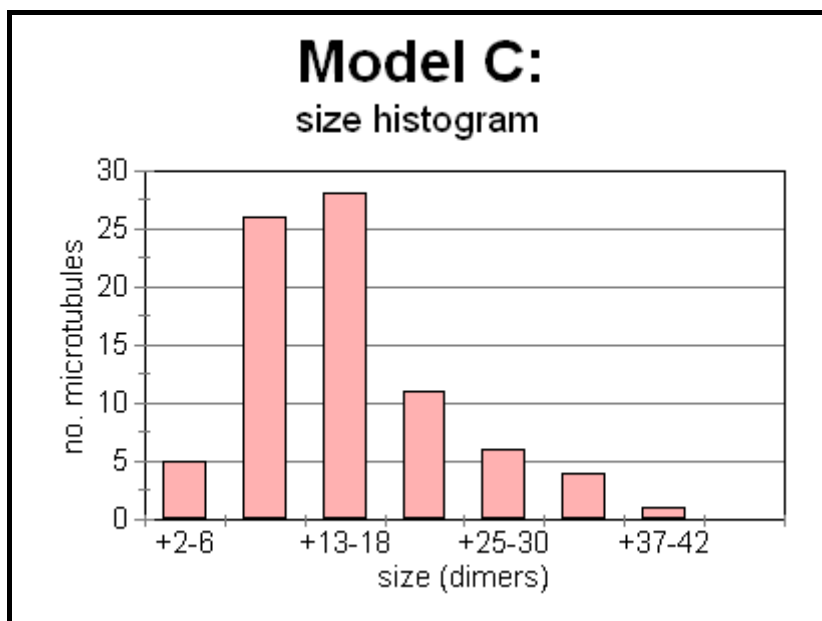


Figure 8.14: Model C: size histogram at one second

The polymerisation curve was very rapid, and peaked at a very high value (99.2% of starting dimers were polymerised at 0.9 seconds).

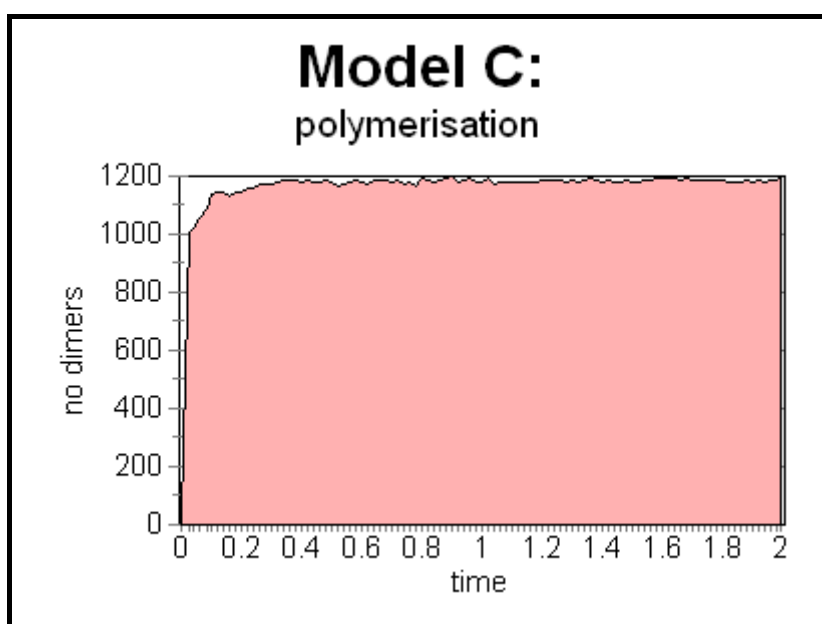


Figure 8.15: Model C: polymerisation curve

While the amount of polymerised material does not change, the distribution of polymer size does become slowly larger. A second later, the size histogram is slightly broader and slightly flatter (compare with Fig. 8.14):

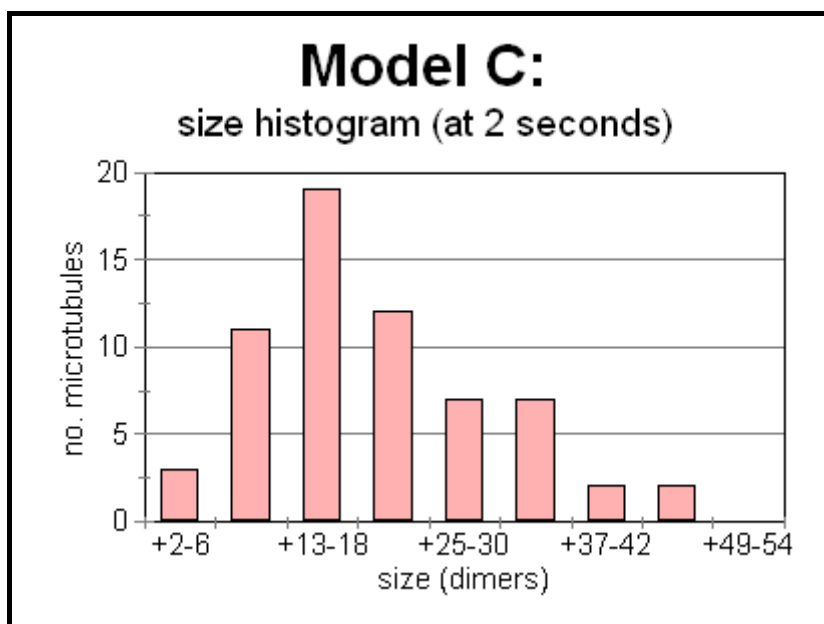


Figure 8.16: Model C: second size histogram

Figure 8.17 shows the simulation field at two seconds:

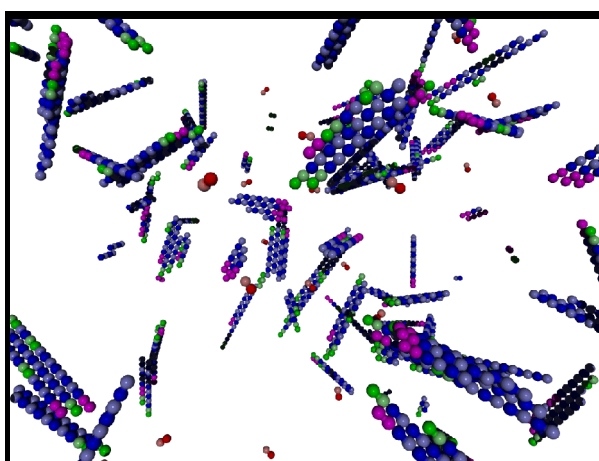


Figure 8.17: Model C: field at two seconds.

Model C: Conclusion

This does not represent the normal, dynamic microtubule growing environment corresponding to our base figures (since polymerisation has been very rapid), however it might well represent an abnormal environment that encouraged strong bonding.

Model D: Reducing Nucleation Probabilities

Another possibility is that nucleation is too easy; that the initial bond between dimers should be lower, perhaps as a result of the greater mobility of the two binding objects, or some other effect such as the link being unnaturally weak without other dimers being present. In order to simulate this, the binding probabilities for un-attached dimers *only* are lowered by a factor of 100.

```
p(top +) = .000004  
p(side)   = .0000007  
p(bottom) = 0.0000011
```

Model D: Results

These probabilities provided a more realistic model of microtubule growth, corresponding more closely to what might be expected from the conditions outlined at the start of the chapter. The simulation was run for 1.25 virtual seconds, during which time less than half the free dimers were polymerised, but three large polymers formed, one of size greater than 140 dimers.

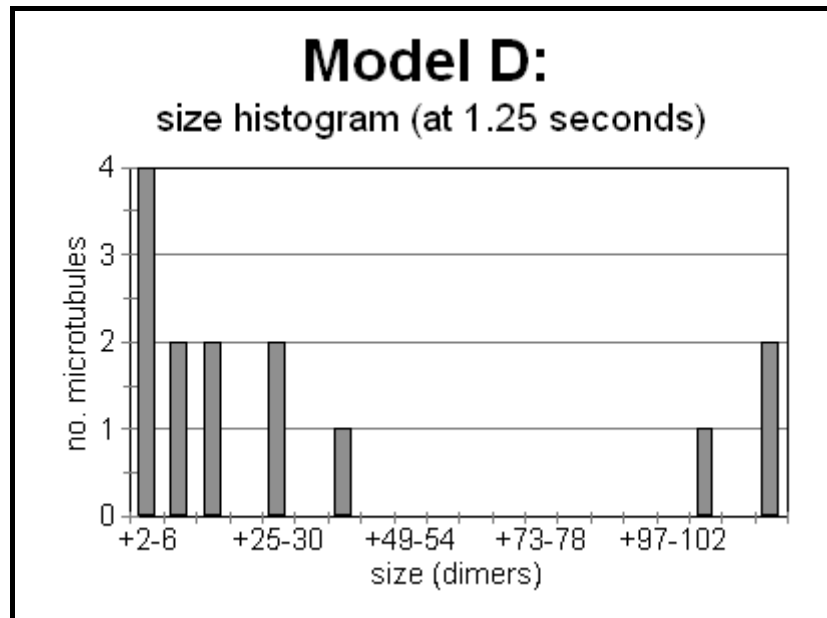


Figure 8.18: Model D: size histogram.

The polymerisation curve was also more consistent with experimental results:

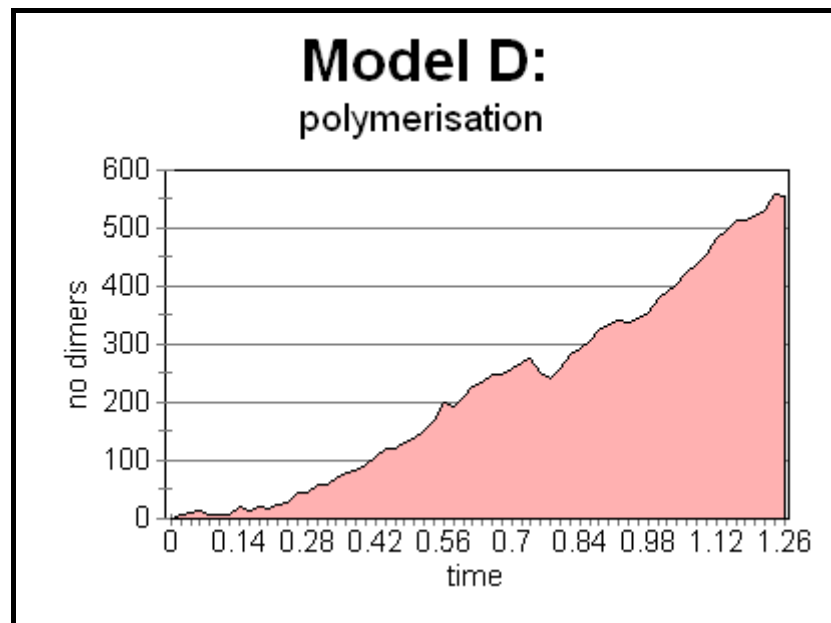


Figure 8.19: Model D: polymerisation curve

Figure 8.20 shows the simulation field at 1.25 seconds:

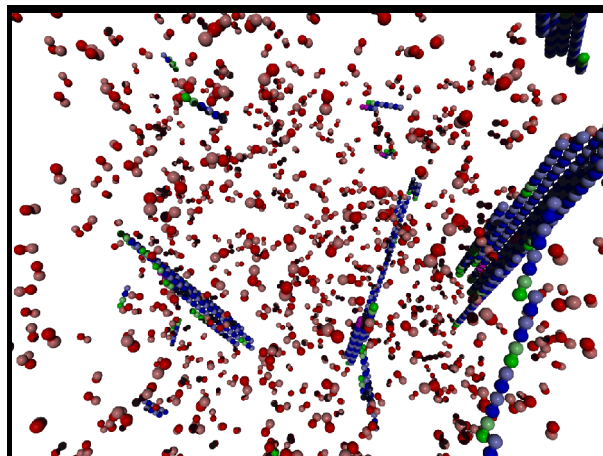


Figure 8.20: Model D at 1.25 s

Model D: Conclusion

This is the most credible result so far. The creation of large, and still growing, polymers, combined with a realistic (albeit very rapid) polymerisation curve, gives credence to the notion that the initial nucleation probability is lower than that expressed by simply using the normal binding figures. This is probably a symptom of the simplicity of the binding model, which might be solved by moving to a more complete binding model based on energy considerations, where two bonds might be considerably stronger than ‘ten times the strength of single bond’, the *ad-hoc* heuristic currently used (cf Chapter 5).

Model E: Low Nucleation Probability, with Higher Breakage Probabilities

To investigate the effect of increased breakage probabilities on model D, the breakage probabilities were increased by a factor of ten, corresponding to a model where dimers are all multiply bound at the tip (such as shown in Fig. 8.4).

Model E: Results and Conclusion

The combination of low nucleation and high breakage resulted in a marked absence of polymers. During the (relatively fast) ten second run of the program, only one aggregate of size greater than 6 was created, and it survived for less than 20 milliseconds. The largest number of fragments present at one time was 5. No microtubules or aggregates of any reasonable size were ever created.

This leads to the conclusion that the breakage figures are at least approximately correct, and lends some support to the view that the growth tip of the microtubule is not the flat tip of figure 8.4, but a spiky tip such as in figure 8.5-8.6, or at least a ragged tip with more singly bound dimers.

Model F: Low Breakage Probabilities with Side Bonds

Another model was investigated, this time using stronger side links. This led to a difficulty, that if the side links are made of equal strength to the top link, then the bottom link is forced to be a very low strength (due to equation 8.4). This led to the following figures being used:

$$\begin{aligned}p(\text{top}+) &= .00013 \\p(\text{side}) &= .00013 \\p(\text{bottom}) &= .00001\end{aligned}$$

In addition, the breakage probabilities were lowered by a factor of two, as in Model B, in order to further encourage assembly.

$$\begin{aligned}p(\text{side}) \text{ gdp} - \text{gdp}(-s) &= 0.018 \\p(\text{bottom}) \text{ gdp} &= 0.018 \\p(\text{bottom}) \text{ gtp} - \text{anything} &= 0.0011 \\p(\text{side}) \text{ gtp} - \text{anything} &= 0.0011 \\p(\text{top/bottom}) \text{ gdp-stable} &= 0.00055 \\p(\text{side}) \text{ gdp-s} - \text{gdp-s} &= 0.00055\end{aligned}$$

Model F: Results

The simulation ran for 5.3 seconds, at the end of which only 5 large aggregates existed, the smallest of which was 30-36 dimers in size, and the largest of which was over 220 dimers in size.

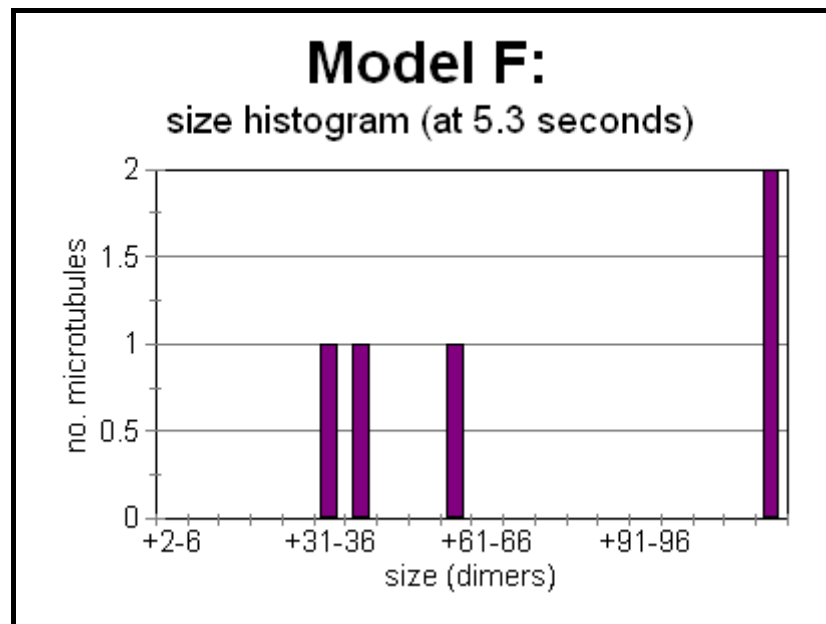


Figure 8.21: Model F: size histogram

The polymerisation curve was rather more volatile than expected (partially due to the small number of growing polymers in the simulation):

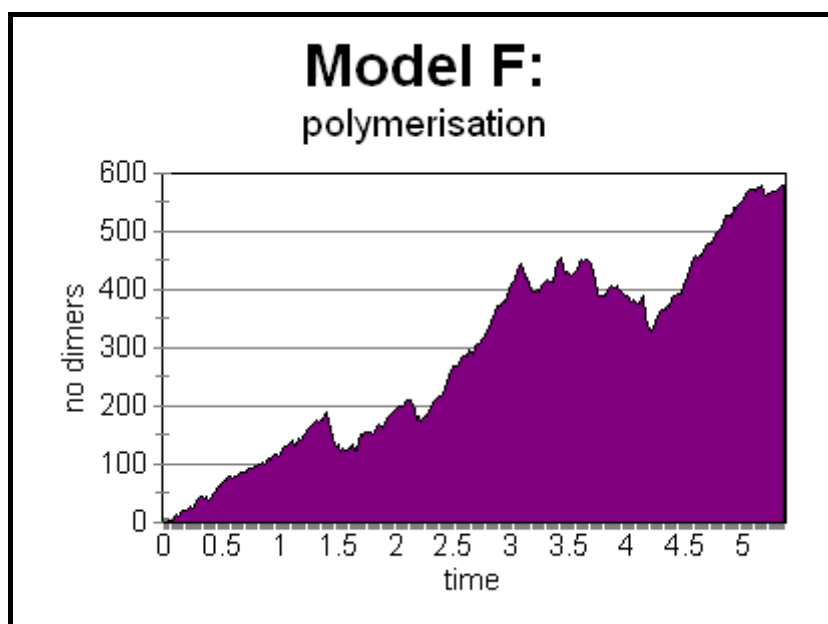


Figure 8.22: Model F: Polymerisation curve

A third of a second earlier, at $t=5s$, there were 8 aggregates (rather than 5), as shown in figure 8.23.

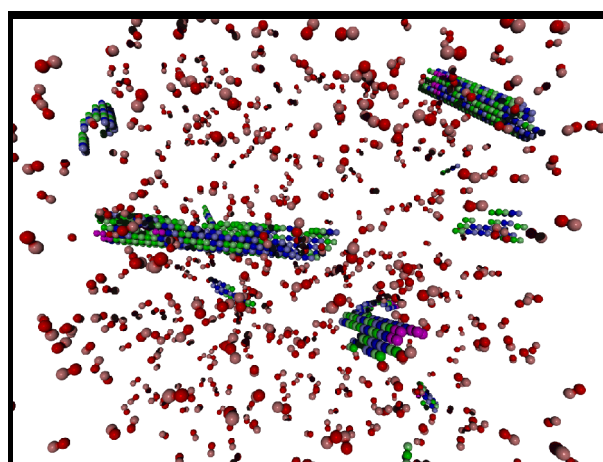


Figure 8.23: Model F: field at 5 seconds

Model F: Conclusion:

This is another result that appears plausible. It indicates that there can be a certain amount of flexibility in the balance between side and back/forward bonds without large scale polymerisation becoming impossible. This implies that a number of growth paths may be possible for microtubules under differing circumstances.

The highly variable polymerisation curve is curious. It may be due to the slightly smaller number of aggregates in the simulation, but it also seems possible that because these aggregates have more side growth (compare Fig. 8.20), they are somewhat more 'ragged', and are more inclined to rapid breakdown (possibly because they have more bound GTP-tubulin exposed). This in turn may be due to some subtle problems within the simulator, where binding events do not always seem to be correctly propagated to the neighbours of a newly bound dimer that 'slots into' a position with multiple bindings.

Discussion and Conclusion

This simulation of various tubulin models shows that the nanoscale simulator is capable of modelling many of the characteristics of microtubule polymerisation during the initial growth phase. The simulator was able to track the initial growth phase of individual aggregates, and obtained statistics and information about the size distribution and polymerisation state of filaments during growth, as well as testing a variety of tubulin models.

Some tentative conclusions might be drawn from the preliminary results above. It appears that a modification to the initial tubulin model is required to prevent the rapid dephosphorylation of the pool of free GTP-tubulin through short-term assembly. The most likely mechanism is probably a time lag in the dephosphorylation of tubulin binding to the 'GTP cap'. The simulator results seems to support the possibility of short term links forming between dimers that do not immediately dephosphorylate them.

Two possible paths for growth are apparent. Either rapid and widespread nucleation, followed by merging of aggregates, or little nucleation, followed by growth from a steady addition of individual dimers. The simulator seems to support the latter as the more likely growth mode of large microtubules, however the initial effect (merging of polymers) may well be very important during nucleation to establish a stable nucleus.

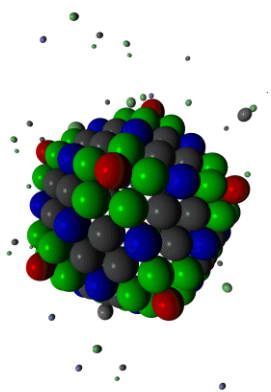
There seems no reason at this stage to include large-scale effects explicitly in the model, as the simulator shows that microtubule construction can proceed adequately on the basis of local rules.

While there is some margin of error, there is a range of workable values for polymerisation. If breaking values are too great, no large aggregates form, while if binding values are too great, many small aggregates form. The simulator shows that values worked out from bulk properties are roughly correct, except that initial nucleation values are still debatable.

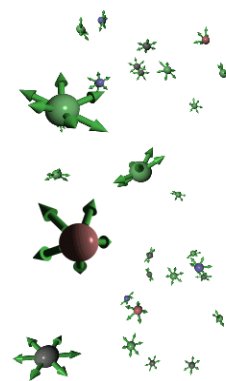
The simulator shows that a range of growth modes (stronger side links, or stronger axial links) appear geometrically possible.

The simulator supports the plausibility of the GTP-cap model, and shows that it can lead to microtubule creation. In addition, the simulator showed rapid disassembly of microtubules fragments that lost the stabilising cap.

The results in this chapter were obtained for relatively small numbers of dimers in a small volume, compared to the earlier actin results, due to limitations on computer time. However the experiments performed here show the utility of even such small scale simulations; larger simulation runs in the future should provide even stronger results, extending to features such as treadmilling and collapse/rescue. A larger simulation should also produce a smoother size histogram.



Chapter 9: Modelling Complex Structures and Viruses



Herba mala cito crescit

(An evil plant grows quickly)

- Roman Proverb

Introduction: More Complex Protein Assemblies

Writing a program to model the behaviour of assembling actin and tubulin is challenging, but once it is created the same principles can be applied to other self-assembling nano-scale structures. The use of program input files (the “.pddf” files - cf. Appendix C) means that different proteins and other organic molecules, and possibly even inorganic objects, can be modelled without having to modify the original nanoscale simulator program.

To perform such simulations accurately requires detailed experimental measurements of the chemical and physical properties of such objects, especially details such as binding energies between each component type, which are beyond the scope of this thesis. Without such measurements, though, it is still possible to investigate the geometry and general behaviour of various hypothetical structures, composed of different types of building blocks with differing geometrical relationships to each other. This chapter outlines a number of geometric structures as a proof of concept, before attempting in broad terms to model the capsid construction of a “typical” virus - herpes simplex A.

Geometric Shapes: The Platonic Solids

Regular solids are some of the easiest objects to model, since their geometric relationships with each other are simple to determine mathematically.

The five platonic solids (the four-sided tetrahedron, six-sided cube, eight-sided octahedron, twelve-sided dodecahedron and twenty-sided icosahedron) are straightforward to model. The icosahedral form is especially interesting since, as will be seen later, the coat protein capsid sheath of many viruses is icosahedral in nature. Pddf files are presented for cubes, dodecahedron and icosahedrons in Appendix D: The Platonic Solids.

The Dodecahedron

The dodecahedron is a twelve sided solid, each side of which is a pentagon. It can be modelled easily using identical monomeric subunits, each of which has five possible linkage sites (Fig. 9.1, Table 9.1). The linkage sites are represented as the base of the green arrows in figure 9.1, while the direction of the links is indicated by the arrows themselves.

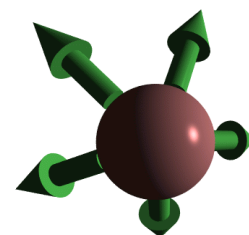


Figure 9.1:
Dodecahedron facial
subunit

Note that there is a geometric similarity between the icosahedron (Table 9.2), with twenty faces and twelve corners, and this dodecahedron, with twelve faces and twenty corners. As a result this same subunit is later used for the “corner” subunit of an icosahedral viral shell.

Dodecahedral Geometry

The geometry of the dodecahedral subunit used is as follows:
(taken from the .pddf file, and truncated to three significant digits):

central sphere: 0.9 nm

5 linkage sites (local co-ordinates, as <x-position, y-position, z-position>):

position A: <0.85,0,-0.526>

position B: <0.263, 0.809, 0.526>

position C: <-0.688,0.5,-0.526>

position D: <-0.688,-0.5,-0.526>

position E: <0.263, -0.809, -0.526>

Notes: Links bind with each other collinearly, with a twist that orients the binding object's centre axis to pass through the centre of the dodecahedron.

Table 9.1: Dodecahedron

A simulation run with monomers of this sort produces twelve unit aggregates, as shown in figure 9.2. Free monomers are in red, aggregated monomers in blue. Note the partially assembled dodecahedrons in the background.

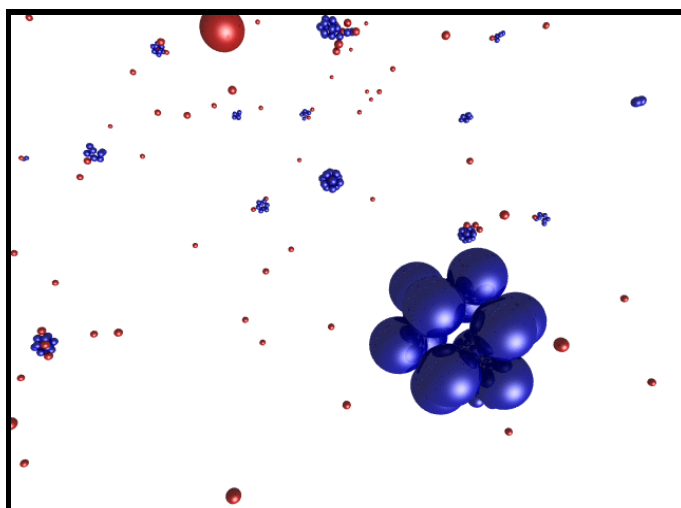


Figure 9.2: The twelve unit dodecahedral aggregates

A Platonic Solid: Icosahedron

The icosahedron is a 20-sided object, each side of which is a triangle. It can be modelled easily using identical monomeric subunits, each having three possible linkage sites (Table 9.2).

Icosahedral Geometry

The geometry of the icosahedral subunit used is as follows:
(taken from the .pddf file, and truncated to three significant digits):

central sphere: 2.7 nm

3 linkage sites (local co-ordinates, as <x-position, y-position, z-position>):

position A: <0,2.491,-0.951>

position B: <-2.16, 1.25, -0.951>

position C: <2.16,1.25,-0.951>

Notes: Links bind with each other collinearly, with a twist that orients the binding object's centre axis to pass through the centre of the icosahedron.

Table 9.2: Icosahedron

A simulated sea of such monomers produces twenty unit aggregates, as shown in figure 9.3. Free monomers are in red, aggregated monomers in green. Note the partially assembled icosahedral aggregates in the background. Also note how having the monomer units in the centre of each face gives rise, as a consequence of the geometry, to an open structure where the faces of each icosahedron are surrounded by five spheres.

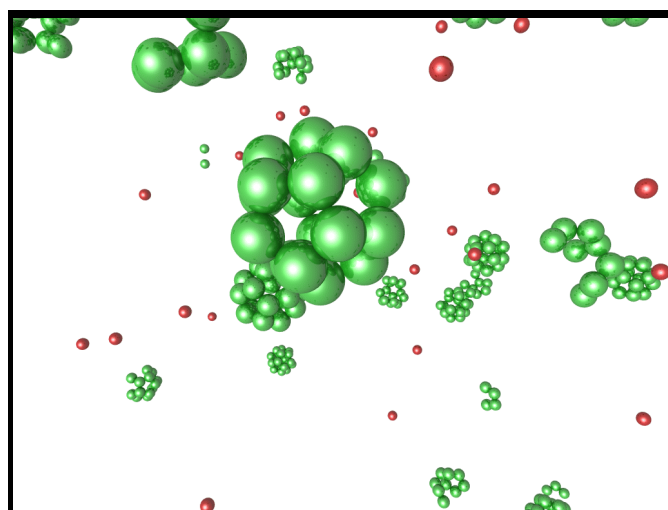


Figure 9.3: 20 unit icosahedral assembly

2-D and 3-D Crystals

Many objects form infinitely repeating regular structures, including some proteins that, in purified form, will condense into regular crystals that can be used for X-ray diffraction and other experimental techniques that take advantage of the periodicity of such crystals. Since the nanoscale simulator operates primarily on the level of the individual object, no special programming is needed to allow the modeller to simulate the growth of such crystals. But because the physical chemistry of the interaction of the objects is being approximated with the concept of “linkage sites”, the interactions between growing crystal regions that result in cleavage planes are not strongly modelled.

Simple Two-Dimensional Sheets

A simple 2-D sheet can be grown using a monomer with three linkage sites placed in a plane with 120 degree separation angles (Table 9.3). Such a sheet grows indefinitely, eventually wrapping around the simulation volume to meet itself (recall that a particle that leaves one ‘side’ of the simulation volume immediately re-enters on the opposite side).

Although it would be theoretically possible for the sheet to meet itself and bind (forming in effect a 3-dimensional torus with a 4-dimensional twist), in fact the program does not handle this situation very reliably because internal checks, that prevent an aggregate colliding with itself, trigger warnings. In general, one part of an aggregate should never collide with another part of the same aggregate during normal movement, so the program treats any such collision as an error condition. But, although the program currently fails in this (rare) limiting case, aggregates up to the size of the simulated field size are easily handled. The .pddf file for this 2-D lattice is given as the example file in Appendix C, Protein Dynamic Description Files.

2-D sheet growth could be used to simulate the assembly of the flat sheets of actin or tubulin that can be grown under certain experimental conditions¹²⁴.

Triangular Sheet Geometry

The geometry of the triangular subunit used is as follows:
(taken from the .pddf file, and truncated to three significant digits):

central sphere: 0.9 nm

3 linkage sites (local co-ordinates, as <x-position, y-position, z-position>):

position A: <1,0,0>

position B: <-0.5, 0.866, 0>

position C: <-0.5, -0.866, 0>

Notes: Links bind with each other collinearly. Since the structure is two dimensional, they have no explicit twist set; the program orients them all in a common direction.

Table 9.3: Triangular Sheet

Using this geometry for monomers, the following structures were produced in a simulation run. The probability of initial nucleation is deliberately set low, so that only one structure is likely to be created. When a nucleation event finally occurs (Fig. 9.4), the structure grows quickly until all available free monomers are bound (Fig. 9.5).

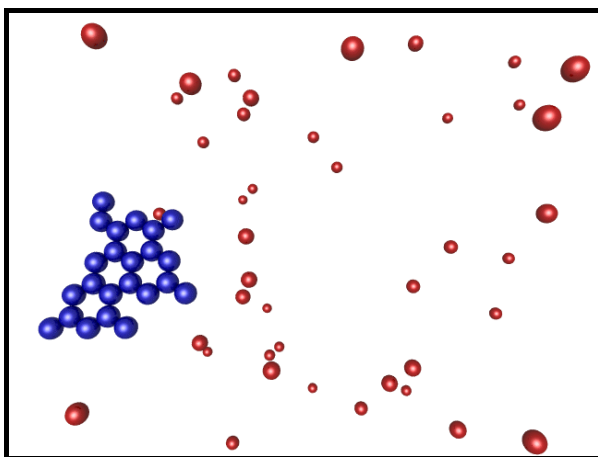


Figure 9.4: The 2-D sheet beginning its growth

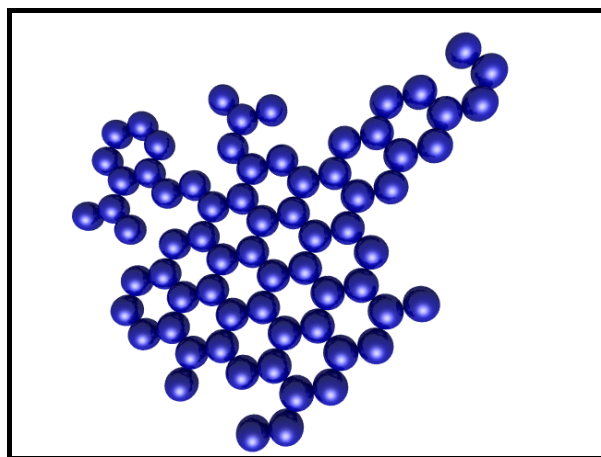


Figure 9.5: The final form of the 2-D crystal.

A Three-Dimensional Lattice

In the same way a three-dimensional crystal can be formed, using a repeating subunit with three-dimensional links. For example, a “diamond-like” tetrahedral lattice can be constructed by using subunits with four links, spaced so as to form the edges of a tetrahedron (Fig. 9.6):

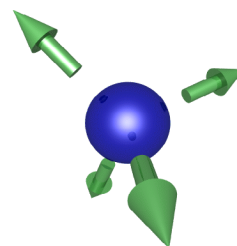


Figure 9.6: Subunit for 3-D “diamond” lattice.

In order to make the structure a little more obvious, the spherical particles have been modelled at a smaller radius than their linkages, hence the gap between the sphere surface and the base of the arrows, representing the link sites. This also leads to the final structure being an “open” lattice, with space between components (Table 9.4).

3-D Tetrahedral Lattice Geometry

The geometry of the tetrahedral subunit used is as follows:
(taken from the .pddf file, and truncated to three significant digits):

central sphere: 0.9 nm

4 linkage sites (local co-ordinates, as <x-position, y-position, z-position>):

- position A: <1,1,1>
- position B: <1,-1,-1>
- position C: <-1,-1,1>
- position D: <-1,1,-1>

Notes: Links bind with each other collinearly. The links are given a twist to invert the linked subunit, so as to create a diamond-like lattice, rather than closed 4-unit tetrahedrons. (This is the exact opposite to what is done for the closed platonic solids described in geometry 9.1 and 9.2).

Table 9.4: Tetrahedral Lattice

The resulting structure formed during simulation (Fig. 9.7) is difficult to interpret visually, in part due to the number of lattice defects in the form of missing subunits. (The set of binding rules used to construct this model favoured rapid growth, leaving a number of holes in the structure as the aggregate grew.) The crystalline structure is still apparent in the ordered rows of objects that can be made out running approximately perpendicular to the page:

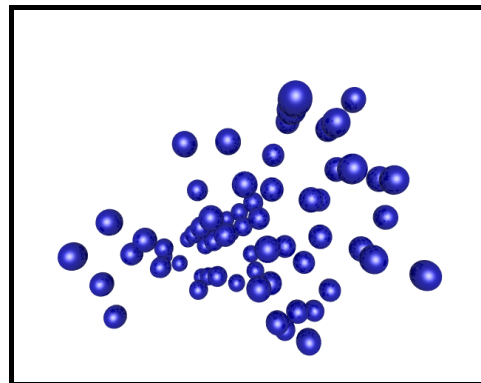


Figure 9.7: “Diamond-form” lattice

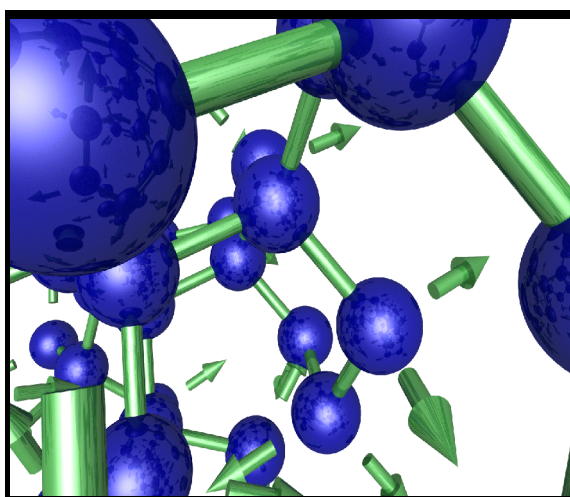


Figure 9.8: “Diamond” lattice with links

To get a better idea of the structure, a section of the above lattice is shown (Fig. 9.8) with links (and unfilled linkage sites)^{****}.

The .pddf file representing this structure is given in Appendix E, “Tetrahedral Lattice .pddf File”.

**** Outputting links is a special simulator feature only available within the output raytracer files.

Simulating a Virus

Some of the most complex and interesting self-assembling structures found in cells are in fact viruses that have invaded from the outside, and are using the machinery of the cell to reproduce themselves. Although a number of types of viruses exist, one common type involves a hard proteinaceous shell in an icosahedral shape, known as the *capsid*, surrounding the fragile viral DNA. In broad terms, the shell protects the viral DNA when the virus is outside the cell, and once the virus enters a new cell this outside sheath is removed, allowing the viral DNA to begin replicating itself, producing more viral DNA and capsid proteins.

It appears that these capsid proteins are capable of self-assembling into completed capsids without help from external scaffolding proteins, although the exact details of the process are not yet fully understood. If this is the case, then viruses are among the most complex self-assembling proteinaceous structures in the cell, with a typical virus shell having hundreds of individual capsid subunits¹²⁵.

Simulating the method by which a virus is constructed is a difficult undertaking, requiring a careful understanding of the molecular nature of the viral coat proteins, and is well beyond the scope of this thesis. But as an example of future directions, the following section outlines how such an exercise might be conducted using a program such as the nanoscale simulator. The example studied is a simplified model of the Herpes Simplex A virus, based heavily on the work of Zhou et al.^{126,127,128,129}. Herpes Simplex A was selected as an example of a well studied virus of intermediate complexity.

Recent Work in Virus Assembly

Shortly after the work described in this thesis was performed, some excellent work (already referred to in Chapter 5) was published by Schwartz et al.¹³⁰, building on earlier work by Berger et al.¹³¹. They showed that mathematical rules, at an individual capsid level, could determine virus structure as well as explaining some experimentally observed defects.

The system of mathematical rules used by Berger et al. operates on the theory that viral capsids are not necessarily of distinct types (neither chemically different, nor chemically similar but with a different conformation), but rather it assumes that capsids can assume different conformations, with different binding properties, depending on their neighbours *at the time of binding*. This rules based system included a model of the energy involved in each link, and iteratively optimized forces and torques within the system to arrive at a final structure. From this model a number of interesting results were obtained, especially in studying models of viral mis-formation, where even a single incorrect addition leads to a malformed capsid¹³².

It should be noted that this premise, that different capsids change conformation depending on the type of binding they undergo, may not be a complete description for all viruses. The work of Zhou et al., which forms the basis of the work later in this chapter, describes the viral capsid assembly of HSV-1 (herpes simplex) in terms of capsids that, although similar, are in fact structurally different, being made up from slightly different arrangements of (shared) basic subunits^{133,134}.

In the later (Dec. 1998) paper Schwartz et al. extended the 'local rule-based theory' to a kinetic simulation similar in some respects to that presented in this thesis. The authors combined the detailed modelling of intra-aggregate forces with a simple model of Brownian motion to simulate a sea of up to 300 particles, using a powerful 8-processor computer. In order to obtain results quickly they used a very high concentration of 247 μM to create a model of a generic T=1 capsid virus assembly in a simulation of 25,000 time steps. Due to the deliberately abstracted nature of the simulation, there was no attempt to obtain a time scale.

The authors of this latest paper derive a number of interesting results related to the degree of tolerance that can be allowed in the links between capsids before viral capsid assembly fails. They also demonstrate a modelling system which provides a considerably more detailed model of the intra-aggregate forces than is attempted in this thesis.

While the details of aggregates are modelled in depth, the larger-scale modelling of the local-rule based system is highly abstracted, which leads to a number of shortcomings.

- The modelling of Brownian motion as a Newtonian system using a “damping force combined with a small randomized force of adaptive magnitude designed to increase kinetic energy” seems an oversimplification, since it ignores the true fractal path of the particle, and is physically unrealistic, giving particles an effective momentum.
- The system also uses a simple Newtonian collision model which, as seen in Chapter 5, under-estimates the number of interactions that particles undergo (although this could be compensated for by increasing the binding probabilities elsewhere in the simulation).
- Schwartz et al. constrain their simulation with a fixed boundary that reflects particles back into the simulation volume. Combined with the basically linear paths of objects, this may lead to edge effects, especially in small simulations with small numbers of particles. For example, it might be possible for a partially assembled virus to get ‘stuck’ in a corner of the simulation field, where the assembling portions of the virus may become ‘starved’ of free monomers, as the bulk of the virus and the proximity of the walls block off the assembling section.
- The system only models a single protein species (albeit with multiple conformations). While there is no theoretical reason that their system cannot be expanded, it may require a major software development effort if the program has not been designed with this in mind.

One of the strengths of the Schwartz model is the detail with which intra-aggregate forces are modelled. However this comes at a great computational price - the simulation runs performed were of the order of 300 particles x 25,000 steps. In comparison, the actin simulation of Chapter 7, on a substantially slower computer, modelled 1,600 particles x 100,000 time steps, in a volume 500 times larger. While the nanosimulator is faster, it is at the expense of this

detailed modelling of forces within the aggregate, and it is likely a combination of the two techniques would be rewarding.

Despite the effort Schwartz et al. have gone to in representing collisions in terms of binding energies, it appears that it is still necessary to employ various (apparently) *ad-hoc* angular and distance tolerances for potential bonds. As these presumably affect the probability of collisions that may lead to binding opportunities, they effectively modify the basic probabilities that have been determined based on association activation energies.

In conclusion, as the only other major work in the area of simulating the self-assembly of large numbers of proteins from a simulated solution of free monomers that this author is aware of, the viral assembly work of Schwartz et al. Offers valuable insights that in many ways complement the work described in this thesis. Whereas this thesis spends most of its effort on the large scale problems of dimer and aggregate movement and collision, and optimising algorithms for speed, the virus assembly work of Schwartz et al. concentrates on the fine detail of how individual aggregates behave, and abstracts the larger scale details. It would be advantageous if the two approaches could be combined, and there is no reason to think that they could not. The software architecture and .pddl language of the nanoscale simulator allows for this type of conformational change on binding, and associated local rule system, although it has not been implemented.

An overview of Herpes Simplex A

Herpes Simplex A is a complex “T16” icosahedral virus (the T16 refers to its geometric structure; it means it is an icosahedral virus, with each “face” consisting of 15 subunits, with units on the edges being shared with other faces). Its structure, as described and illustrated by Zhou et al. on their webpage¹³⁵ is made up of “protein subunits in ... three distinct morphological units: penton, hexon and triplex of this T=16 icosahedral particle”. They present the illustration reproduced in figure 9.9:

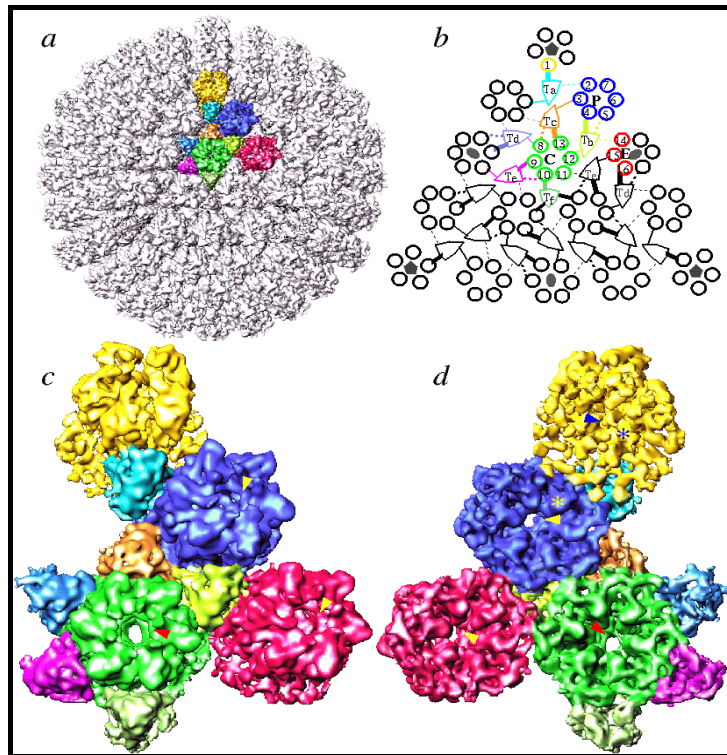


Figure 9.9: Zhou et al; Herpes Simplex A structure

and the following descriptive text:

“(a) The whole capsid is shown as shaded surface with the structural components in one asymmetric unit in color. Included in an asymmetric unit are morphological components from penton (yellow), P-hexon (blue), E-hexon (red), C-hexon (green) and six different types of triplex (other colors). (b) The schematic diagram of one triangular face of the herpes virus capsid illustrates the interactions between the subunits (VP5) of penton and hexons (1 - 16) and triplexes Ta, Tb, Tc, Td, Te and Tf. Subunits within an asymmetric unit are shown using the same color coding as in (a). Four kinds of interactions between triplex and VP5 are depicted, including strong head (thick solid line), weak head (thick dashed line), tail (thin solid line) and arm (thin dashed line). (c) The contiguous group of morphological components colored in (a) is blown up and shown as viewed from outside (c) and inside (d) respectively. Arrowheads in (c) indicate the contacts between triplexes and their adjacent penton and hexons. Only those visible at this view are indicated.”¹³⁶

Defining The Four Major Component Protein Types

To summarise the above description, the major capsid proteins are of four main types:

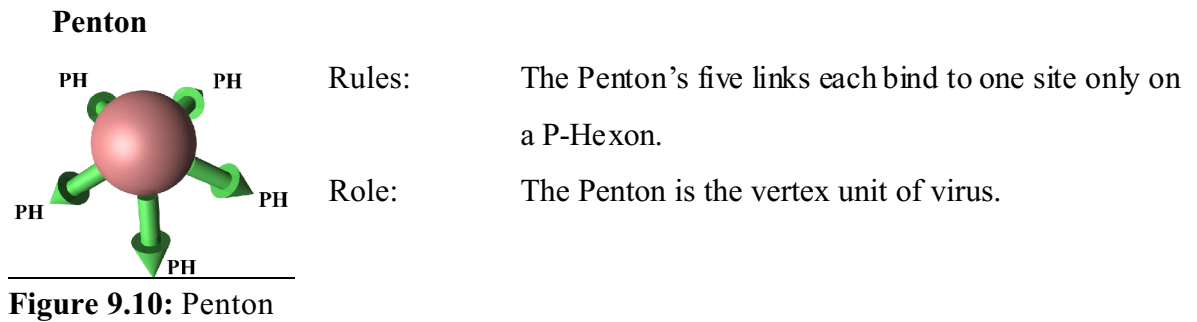
- a corner capsid subunit, the “Penton”, which has five major subunit neighbours;
- a “Hexon” adjacent to it with six neighbours, the penton-adjacent subunit, or “P - Hexon”;
- a further edge subunit, or “E-Hexon”; and finally
- a centre-face subunit, or “C-Hexon”.

In addition, there are a large number of much smaller proteins that join these larger components together, the “triplexes”.

Although a full simulation would include the triplexes, in this discussion they are ignored, and only the larger pentons and hexons are modelled, along with some simplified rules of association that can all be deduced from the relationships described above.

The geometry of the structure is somewhat involved. The relative positions of the capsids were carefully worked out by hand (as with all the other structures presented in this chapter), but a complication arises with the linking behaviour. When an object binds another object, their new orientation is specified by the ‘twist’ vector parameter of the link (cf chapter 6, “Binding Free Objects”). The twist vectors were also carefully worked out, on geometrical grounds, to ensure that when capsids bound to other capsids they were correctly oriented. Sometimes the program default twist was adequate (if no twist vector is specified, the program calculates the twist as being at right angles to the link direction, in the same plane as the z-axis), while at other times it had to be specified directly. All link directions were calculated so that capsids would bind collinearly.

An overview of the structure of the capsid proteins follows, and complete details of the .pddf file are given in Appendix F.



(Note: In figure 9.10 and following images the links are labelled with the destination type, using the following abbreviations:

P = Penton;

PH = P-Hexon, or “Penton Adjacent Hexon”;

EH = E-Hexon or “Edge Hexon”;

CH = C-Hexon or “Centre Hexon”.)

Penton Geometry

central sphere: 0.9 nm

5 linkage sites <position>:

position A (to P-Hexon): <0.851, 0, -0.526>

position B (to P-Hexon); <0.263, 0.809, -0.526>

position C (to P-Hexon): <-0.688,0.5,-0.526>

position D (to P-Hexon): <-0.688,-0.5,-0.526>

position E (to P-Hexon): <0.263, -0.809, -0.526>

Note: the default twist was used for this capsid.

Table 9.5: Penton Geometry

P-Hexon

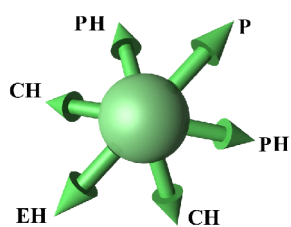


Figure 9.11: P-Hexon

Rules:

The P-Hexon has one link that binds to a Penton. The two links adjacent to that bind to other P-Hexons. The link opposite the Penton link binds to an E-Hexon, and the two adjacent links bind to C-Hexons.

Role:

The P-Hexon is an edge unit, lying adjacent to the Penton. ("Penton associated, or adjacent, Hexon".)

P-Hexon Geometry

central sphere: 0.9 nm

6 linkage sites <position> and <twist vector>:

position A (to Penton):	<1,0,0>	twist:<0,0,1>
position B (to P-Hexon):	<0.5,0.809,-0.309>	twist:<0.162,0.263,0.951>
position C (to C-Hexon):	<-0.5,0.809,-0.309>	twist:<-0.162,0.263,0.951>
position D (to E-Hexon):	<-1,0,0>	twist:<0,0,1>
position E (to C-Hexon):	<-0.5,-0.809,-0.309>	twist:<-0.162,-0.263,0.951>
position F (to P-Hexon):	<0.5,-0.809,-0.309>	twist:<0.162,-0.263,0.951>

Table 9.6: P-Hexon Geometry

E-Hexon

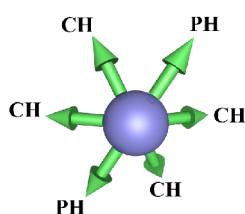


Figure 9.12:E-Hexon

Rules:

Two opposite links bind to P-Hexons, the others all bind to C-Hexons.

Role:

The E-Hexon is the other edge unit, lying in the very middle of an edge between two P-Hexons. (“Edge Hexon”.) Its geometry is identical to that of the P-Hexon although its binding rules differ.

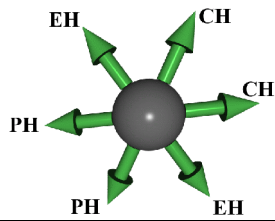
E-Hexon Geometry

central sphere: 1 nm

6 linkage sites <position> and <twist vector>:

position A (to P-Hexon):	<1,0,0>	twist:<0,0,1>
position B (to C-Hexon):	<0.5,0.809,-0.309>	twist:<0.162,0.263,0.951>
position C (to C-Hexon):	<-0.5,0.809,-0.309>	twist:<-0.162,0.263,0.951>
position D (to P-Hexon):	<-1,0,0>	twist:<0,0,1>
position E (to C-Hexon):	<-0.5,-0.809,-0.309>	twist:<-0.162,-0.263,0.951>
position F (to C-Hexon):	<0.5,-0.809,-0.309>	twist:<0.162,-0.263,0.951>

C-Hexon



Rules: Two adjacent links bind to other C-Hexons, the opposite links bind to P-Hexons, while the other two, diametrically opposing, links bind to E-hexons.

These hexons make up the central three units of a virus face.

Figure 9.13:C-Hexon (“Centre Hexon”.)

C-Hexon Geometry

central sphere: 1 nm

6 linkage sites <position> and <twist vector>:

position A (to E-Hexon):	<1,0,0>	twist:<0,0,1>
position B (to C-Hexon):	<0.5,0.866,0>	twist:<0,0,1>
position C (to C-Hexon):	<-0.5,0.866,0>	twist:<0,0,1>
position D (to E-Hexon):	<-1,0,0>	twist:<0,0,1>
position E (to P-Hexon):	<-0.5,-0.866,0>	twist:<0,0,1>
position F (to P-Hexon):	<0.5,-0.866,0>	twist:<0,0,1>

Note: Since the C-Hexon is essentially flat, all the twist vectors point in one direction, +z .

Table 9.8: P-Hexon Geometry

Overall Structure

These units fit together to form a virus side as shown in figure 9.14 (the colour coding is the same as above, but the shades are darker). This simplified structure should be compared with figure 9.9 above.

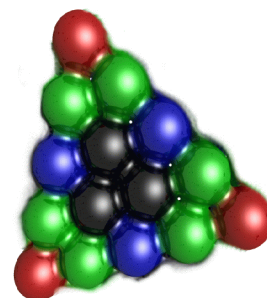


Figure 9.14: A single virus face showing the four subunit types.

Some Assembly Required

By initialising the nanoscale simulator with a mixture of the hexons and pentons described above, in approximate proportion to their frequency in the shell (a complete shell has 12 Pentons, 60 P-Hexons, 30 E-Hexons, and 60 C-Hexons) the initial conditions are created for viral shell formation. In this simulation, the probability of initial nucleation is deliberately kept low, resulting in a small number of initial nucleation sites, which then grow into full-sized shells with little or no interaction between aggregates.

Since this simulation was intentionally abstract, it was run in a favourable environment. The concentration was relatively high (6.2 μM) in a 512nm cubed simulation space (4000 monomers). Binding strengths were made artificially high (0.9 prob of binding), and the simulation was run for less than a second, until reasonably complete structures appeared. The numbers of different proteins was in the following proportions (from the .pddf file:) Penton 8%, HexonP 37%, HexonE 18%, HexonC 37%.

Initially, the four capsid element types are floating free in the simulation (Fig. 9.15):

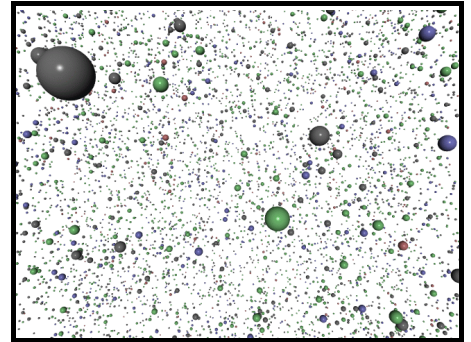


Figure 9.15: Virus starting condition.

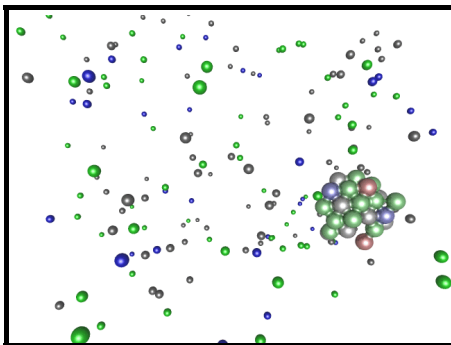


Figure 9.16: Virus fragment during growth.

Figure 9.16 shows a close up of a partially constructed virus capsid, currently about half completed.

Figure 9.17 shows a nearly complete capsid at close range.

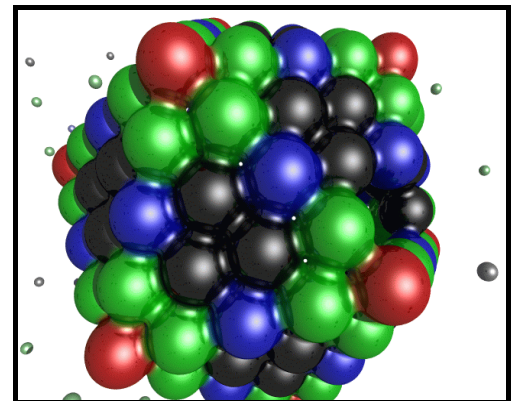


Figure 9.17: Close-up of virus capsid

Future Directions

The above is far from an accurate simulation of viral assembly, however it demonstrates the power of the nanosimulator, in that a large number of particles (4,000) of four different protein species were successfully simulated over several thousand time steps, and a complex structure comprised of hundreds of individual units self assembled based only the geometric rules of the four component capsid proteins. As far as the author is aware, this is the first time the assembly of such a geometrically complex aggregate, comprised of different proteins has been simulated.

To improve the simulation, physically reasonable values for the binding and breaking probabilities should be obtained, or at least approximated, and the role of joining proteins (the triplexes) should be included in the model. It is also possible that the viral DNA or RNA itself should be modelled as a physical component that fits within the capsid shell; it might be speculated that it plays a role in the construction of the shell itself, perhaps as a nucleation point for the capsid? (Although it should be noted that viral capsids can self-assemble *in-vitro*, even without the presence of viral DNA/RNA.)

A further possibility for experiment is adopting the local rule-based model mentioned previously, where instead of four major protein species (and potentially six ‘triplex’ glue proteins), only one protein capable of adopting different conformations on binding is used (and similarly a single ‘triplex’ glue protein capable of adopting six conformations). This may well turn out to be a physically more realistic model for many viruses, and is a promising avenue of research.

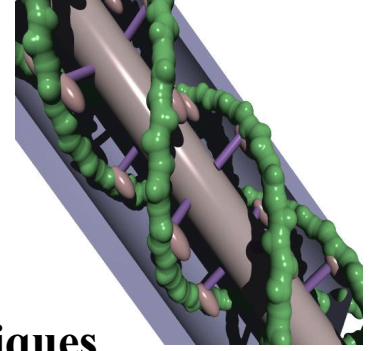
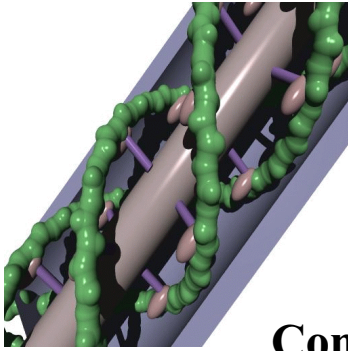
Modelling viral assembly in this way may be helpful for drug design, since some anti-viral drugs act by disrupting the coat assembly/disassembly process¹³⁷. For example, a drug that acted as a nucleation site to produce geometrically inaccurate shells might be useful in disrupting the viral growth cycle, and its effects could be “designed” to an extent using the nanoscale simulator¹³⁸.

Conclusion

Having tested the simulator on tubulin and actin, this chapter has shown that the simulator can be applied as a general purpose tool for self-assembling systems.

The nanoscale simulator has been shown to be sufficiently flexible to model a broad range of objects, ranging from small aggregates made up of a single type of subunit, through to very large, complex aggregates such as virus capsids, made up of a number of different types of subunits with different properties. Since the nanoscale simulator uses the same rules and data structures for manipulating all these objects, modelling a new type of subunit simply requires using a different “.pddf” file, and does not require any changes to the central program.

A wide range of different protein species (or other nanoscale objects), each with its own geometry, linkage definitions, and physical characteristics, can be simulated in large numbers over a large number of time steps. Another useful feature is that, in the absence of experimental data, the program will estimate features such as mass, moments of inertia and diffusion coefficients based on average protein density. This allows the nanosimulator to be used for more speculative simulation, while maintaining physically realistic values.



Chapter 10

Combined Modelling Techniques

Simulando discitur

(By simulation learning is acquired)

- anon

Introduction: Integrating with other Modelling Methods

Simulating nanoscale objects can be a powerful technique for understanding some of the mechanisms involved in cell function. It can be even more useful when combined with other computer-based techniques, such as 3-D modelling and image processing.

This chapter looks at how the nanosimulator can be used to improve modelling of cytoskeletal structures in plant cells, such as actin and tubulin filaments, and more complex structures such as plasmodesmata. A number of modelling techniques are used; image processing is done on raw electron micrographs, 3-D models are constructed based on these micrographs and other experimental data, and the resulting models are inserted into the nanosimulator as static structures, around which other objects (such as stain particles and antibodies) move and interact. The results of these simulations are taken from the nanosimulator and processed to produce ‘Virtual Electron Micrographs’ (VEMs) using physically reasonable models. These can then be compared with the original micrographs, to provide a check on the modelled data.

Part of the material in this chapter has been presented at the Third International Workshop on Basic and Applied Research in Plasmodesmal Biology, Zichron-Yakov, Israel 1996 under the title “Computer Assisted Analysis and Modelling of Plasmodesmata” (Betts, C., Conway, D., Radford, J., White, R.), and as “Virtual Electron Microscopy” (Betts, C., White, R.), at the 15th Australian Conference for Electron Microscopy, Hobart, Tasmania.

Electron Micrographs and Image Processing

This section presents some of the raw material biological modellers work with. These ‘Electron Micrographs’ (or simply ‘micrographs’) are produced by a Transmission Electron Microscope (TEM), and show images of actin filaments and microtubules, as well as those of plasmodesmata. The images are taken from a variety of sources. Some were made using stained samples embedded in a thin sheet of resin, or sprayed onto a coated grid, or set in vitreous ice. All were taken at very high magnification, at close to the limit of a transmission electron microscope (other types of electron microscope offer higher magnification, but are unsuitable for many types of biological specimens).

Actin and Tubulin

These images show some standard micrographs of actin and tubulin:

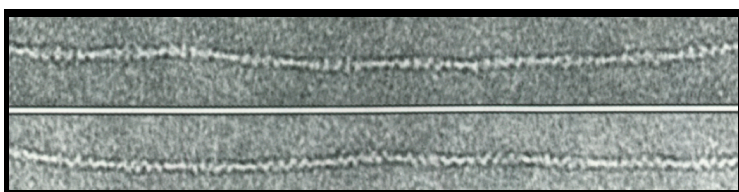


Figure 10.1: Actin TEM micrograph

Figure 10.1 shows a typical micrograph of negatively stained actin from a coated grid. Note the regularly spaced lines, caused by the helical nature of actin filaments.¹³⁹

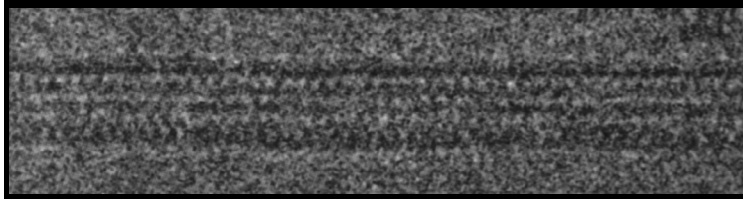


Figure 10.2: Microtubule TEM micrograph

Figure 10.2 shows a high magnification image of a microtubule in ice. Note the regularities, again caused by the structure's helical nature.¹⁴⁰

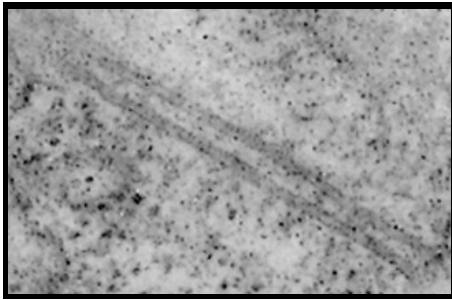


Figure 10.3: Microtubule TEM micrograph in-vivo.

Figure 10.3 shows another image of *two* microtubules, showing how microtubules appear within a cell.¹⁴¹ The details are more difficult to see, and the ends are not visible in this image.

Plasmodesmata

Plasmodesmata, the connectors between plant cells described in Chapter 2, are hard to examine under an electron microscope, due to difficulties in specimen preparation (primarily because it is difficult to cleave a plasmodesmata cleanly; usually the cleavage plane goes over or under the cylindrical structure). The following images show a transverse section and a longitudinal section.

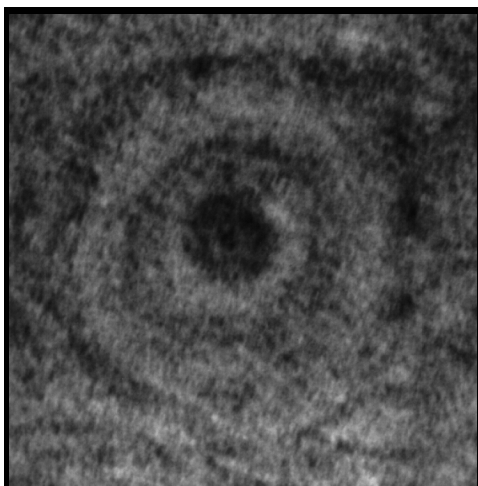


Figure 10.4: Plasmodesma cross-section micrograph

Figure 10.4 shows a plasmodesma cross section. Notice the dark inner 'desmotubule', the (middle ring) plasma membrane, and the outer ring (probably caused by the opening funnel of the plasmodesmata).¹⁴²

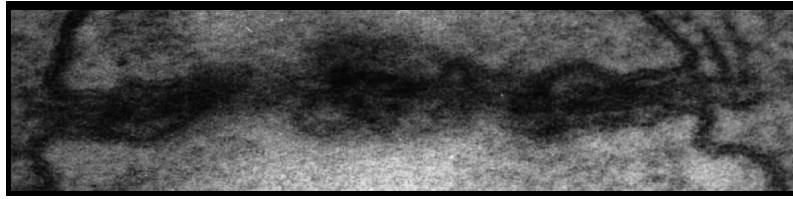


Figure 10.5: Plasmodesma, transverse TEM micrograph

Figure 10.5 shows the longitudinal section of a plasmodesma. The desmotubule can be seen as the thick line running down the centre of the plasmodesma, while the plasma membrane can be seen as the thick outer lines¹⁴³. The lines bracketing the image at both ends are the cell walls of the two cells between which the plasmodesma runs.

Image Processing

These raw micrographs, combined with other experimental results, are used by biologists in determining structure. However, due to practical difficulties with instruments, they are often far from perfect, as can be seen in the above images. At high magnification, a TEM micrograph becomes grainy and blurred, and there may be additional problems with contrast and photographic development.

Using computers it is possible to ‘post-process’ such images (as described in Chapter 3), to reduce the severity of these effects. For example, by stretching the grey-scale histogram of an image, the contrast can be improved, and by using an ‘sharpening’ filter, colour gradients can be intensified. To illustrate, these corrections were applied to figure 10.5 using the standard commercial PC package ‘Paintshop Pro 5.0’), slightly improving its quality:

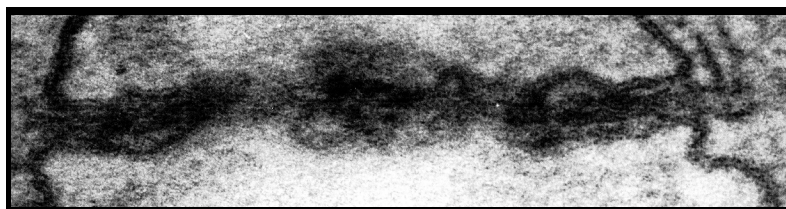


Figure 10.6: Plasmodesma micrograph after histogram equalisation and edge enhancement.

Histogram equalisation (summarised in Chapter 4) is a way of increasing the contrast of an image on a global basis, which can be very useful if an entire image is underexposed or overexposed. However, this technique often fails if the image contains both light and dark regions, but the individual regions are under- or over-exposed. It becomes more useful to do ‘local histogram equalization’¹⁴⁴, where each pixel is adjusted relative to its local area.

In order to enhance images of plasmodesmata such as figure 10.4, which feature very dark areas, especially in the desmotubule, a simple local histogram equalization algorithm was written as a plug-in to the ‘Image Explorer’ suite of image manipulation tools supplied by SGI. Applying this local histogram technique on figure 10.4 produced the following image:

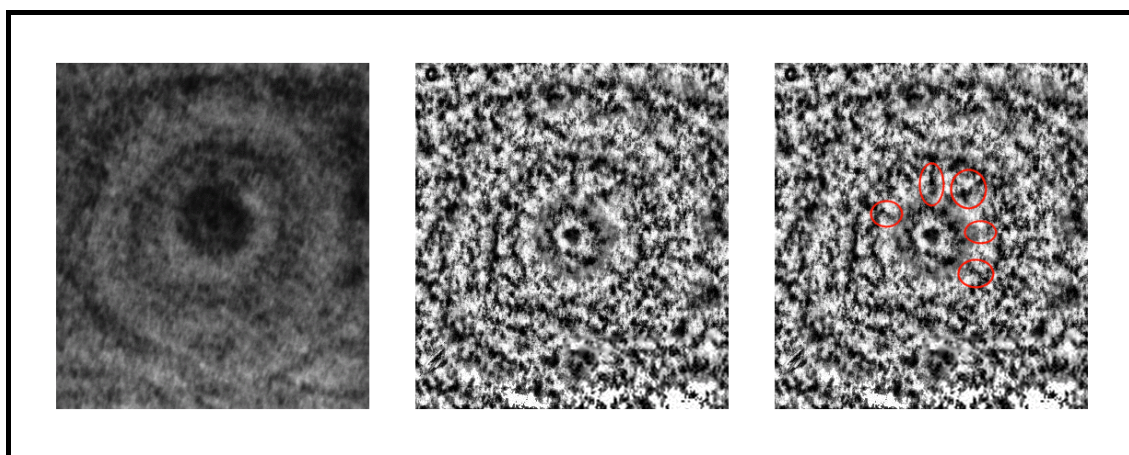


Figure 10.7: Plasmodesmata: original TEM (left); with local histogram equalisation (centre); and the same image again with hypothesized ‘struts’ between the desmotubule and plasma membrane circled (right).

Local histogram equalization may make it harder to recognize global details (such as the concentric rings of the plasma membrane). However it brings into better relief the fine detail of objects embedded within the desmotubule, and within the plasma membrane, as well as improving the definition of the lines leading from the desmotubule to the plasma membrane (Fig. 10.7). In fact, some details which are barely visible in the first image become clearly visible in this image, such as the hypothesized linkage strut on the lower right of the desmotubule.

There are many other possibilities for enhancing images. As a further example, a question that has often arisen regarding the large particles that are either embedded in the desmotubule^{145,146}, or positioned between the desmotubule and the plasma membrane¹⁴⁷. Specifically it is unclear whether or not these particles are symmetrically arranged.

To help resolve this question, the image (Fig. 10.7) was processed using a polar Fourier transform (again, written as a plug-in for Explorer on the SGI), after which the angular high frequency components were dropped, using a low frequency angular bandpass filter (see Chapter 4 for a brief explanation of polar Fourier transform bandpass filtering).

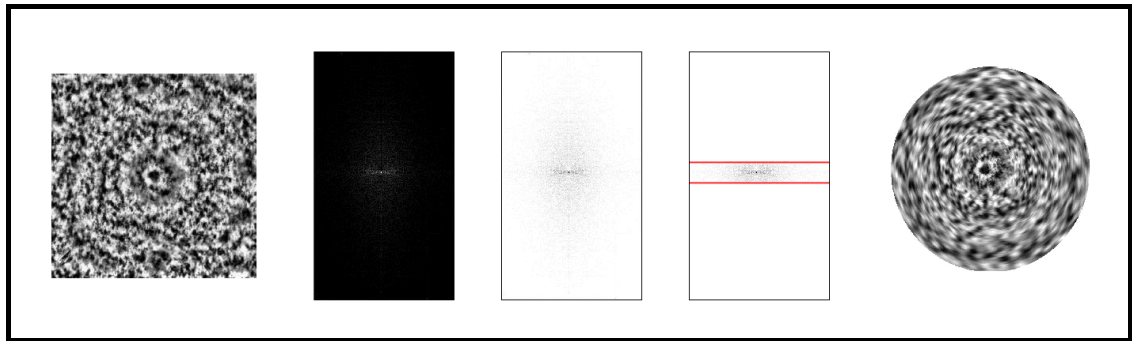


Figure 10.8: Polar Fourier Transform: original image (far left), polar Fourier transform (middle left), negative of same (centre), *example* of bandpass filtering (middle right), final image (far right) (reproduced in 10.9).

Figure 10.8 shows the general progression. The original image is converted to a polar Fourier transform, reproduced as a negative for clarity. The higher frequency angular components (angular frequency in these examples is the vertical axis) are trimmed off, and the inverse polar Fourier transform applied to obtain the final image, using only angular low frequency components.

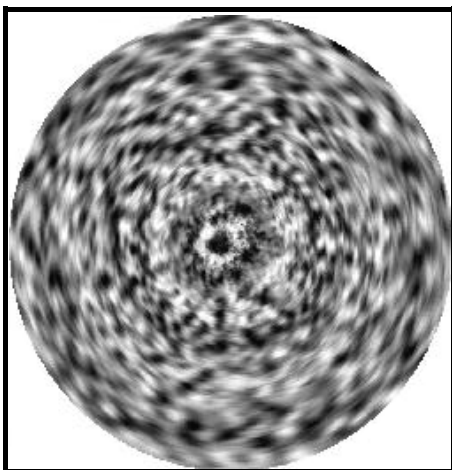


Figure 10.9: Plasmodesmal cross-section after low-frequency angular band-pass filtering of polar Fourier spectrum.

The resulting image shows only the low frequency, i.e. rotationally repeated, components. We can see this in the image on the left, where the circular rings remain visible, but most of the other detail is either lost or reduced in intensity. Features closer to the centre are more accurately reproduced, since they subtend a relatively larger arc, and are thus more accurately described by the truncated Fourier spectrum.

The preceding image does *not* in fact show any strong evidence of rotational symmetry in the electron opaque (i.e. dark) areas in the desmotubule in the preceding image. Such rotational symmetry should express itself as a repeating rotational pattern in figure 10.9, but no such pattern is discernable. However a more intensive study on a large number of images using this and similar techniques would be needed before reaching any conclusion.

Using Micrographs and Processed Images

When the image, and even the processed image, does not give enough detail to fully determine the structure, other methods of investigation can be used to add further information. Usually however the electron micrograph at least places some general boundaries around the problem, by giving the rough dimensions of the structure, and possibly details about the regularity of the structure. These features, combined with available information on proteins present (derived from other experiments and analyses), can be used to create theoretical models that test the practicality of different arrangements of the proteins, membranes and other molecules making up biological structures.

3-D Models

A useful and long-established means of studying proteins and protein structures is the creation of 3-D models. Such 3-D models have ranged from toothpicks and plasticine to quite sophisticated works of machined wood and metal. Increasingly though, the ease with which computers can help researchers produce and manipulate 3-D models has led to the widespread adoption of computer-assisted modelling by researchers attempting to determine what protein conformations and arrangements are geometrically possible.

A Note on Static 3-D Models

Despite the popularity of molecular modelling programs and 3-D CAD packages, it was not possible to find modelling software that could be easily used for this project, and as a result a simple 3-D modelling package was written from scratch, comprising of a number of separate viewing, scene creation, and processing utilities. This small package was able to read and save object definitions, output data in a variety of forms that could be used by other programs (including the nanosimulator), display results in real time using the Silicon Graphics hardware graphics library, and (most importantly) read in a mathematical description of object placement (especially helices). This last feature allowed the “single line” definition of thousands of separate objects making up a large helix, and was essential in creating some of the models described below.

All these features were available, and implemented to a far greater degree of sophistication, in other pre-existing packages, but unfortunately no single package combined all the features required. In order to maintain compatibility with future work, and the possibility of a better package appearing in the future, the modelling package was written to output data in both *povray* and *wavefront* formats, allowing other programs to also use the data.

In this thesis, most of the images are rendered in the ‘Povray’ raytracer, rather than using screen shots from the computer console, since this allows a great deal more detail to be displayed. To this end, minor details such as camera and illuminant positions are sometimes set manually to better frame the objects, as are material properties, such as whether the objects appear glossy or matte, and whether spheres are displayed as entirely discrete, or whether adjacent spheres are allowed to melt into each other. These finishing touches do not affect the geometry of the model, and are simply done to make the pictures clearer by giving the reader more visual cues to aid interpretation of the 3-D structure represented by the object. The reader should bear in mind that while the 3-D modeller utilities can model a wide variety of objects, such as cylinders, hollow cylinders, rectangular blocks etc., the nanosimulator itself is only capable of manipulating spheres.

Models of Actin and Tubulin

Creating models of actin filaments and microtubules was a straightforward task. Models can easily be created by saving data from a nanosimulator session that had resulted in the growth of actin filaments or microtubules. This is normally done when the output of the nanosimulator is viewed and studied after the program is run.

Another way to create models of actin and tubulin assemblies is to model them as simple helices, twisting strands defined with mathematical precision. This can be an easier approach if it is necessary to precisely position an object. For example, if the object needs to be centred mathematically on the origin, this can still be accomplished with data saved from the nanosimulator, but usually requires the object to be manually re-positioned, since the object will have been moving about randomly within the simulation volume.

The models presented here were defined using simple mathematical helix definitions, and then rendered using povray. The model of actin in figure 10.10 is the standard model¹⁴⁸, however the microtubule model requires more explanation. Microtubules assemble into a number of different types of helix, and a given microtubule may even have a different structure within different regions¹⁴⁹.

These different structures are categorised by the number of *protofilaments*, or component longitudinal strands, within the microtubule. The most common variety (and the one modelled in previous examples by the nanosimulator) is the 13-prot filament model, but 14-prot filament microtubules are also quite common, and 12 and 15 proto filament microtubules also occur. As it is equally easy, using a mathematical description, to model any of these, a 12-prot filament microtubule based on the observational work of D.Chreti  n et al.¹⁵⁰ was chosen. The most obvious difference between these different microtubule types is that only the 13-start microtubule has straight protofilaments - in all other types the protofilaments form slowly twisting helices, as can be seen in figure 10.11 below. A final point to note is that in this particular representation no visual distinction is made between α -tubulin and β -tubulin.

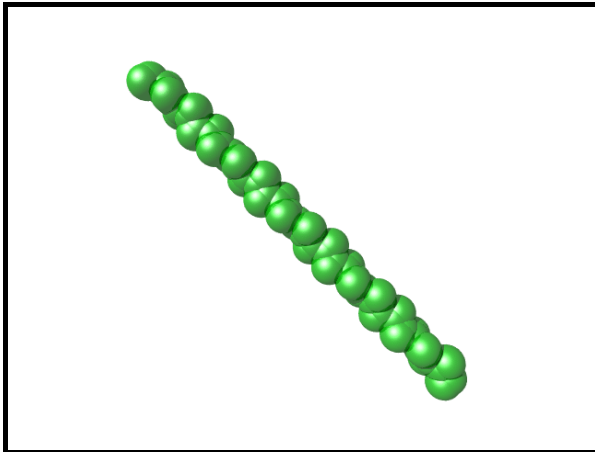


Figure 10.10: The Actin Model

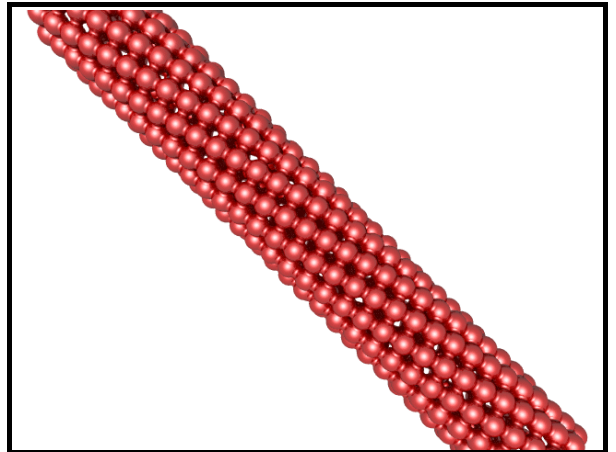


Figure 10.11: A '12-start' Tubulin Model

Proposed Models of Plasmodesmata

Plasmodesmata, the connecting passages between cells in higher plants, were introduced in Chapter 2. The exact structure of plasmodesmata is not known for certain, although a number of similar models exist. A brief summary of some of the proposed models is given below:

The Textbook Model

This model (Fig. 10.12) shows the commonly accepted basic features of plasmodesmata; a tunnel between the cell walls lined with the plasma membrane, with a thin tube of endoplasmic reticulum (E.R.) passing through the centre, connecting, *to some extent*, the E.R. of the adjacent cells.¹⁵¹

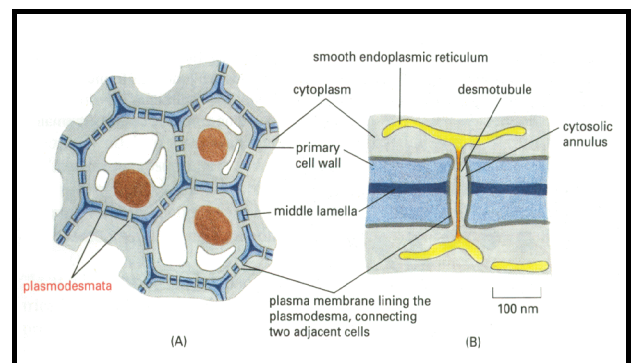


Figure 10.12: Plasmodesmata: Standard Model

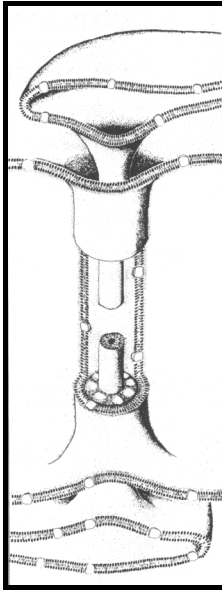


Figure 10.13:
Overall Model

The Overall Model

The Overall Model (Fig. 10.13) extends the standard model by postulating the existence of a number of globular proteins in the lumen between the desmotubule and the external plasma membrane sheath.¹⁵²

The Ding Model

This model (Fig. 10.14) modifies the standard model by including a role for a number of undetermined globular proteins embedded in the central desmotubule (or 'appressed endoplasmic reticulum protein complex', as it is referred to by the authors), and on the inner side of the plasma membrane sheath that lines the outer walls of the plasmodesmata, rather than in the lumen as above.^{153,154}

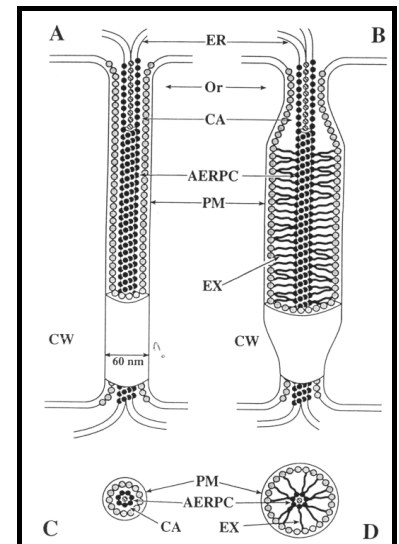


Figure 10.14: Ding Model

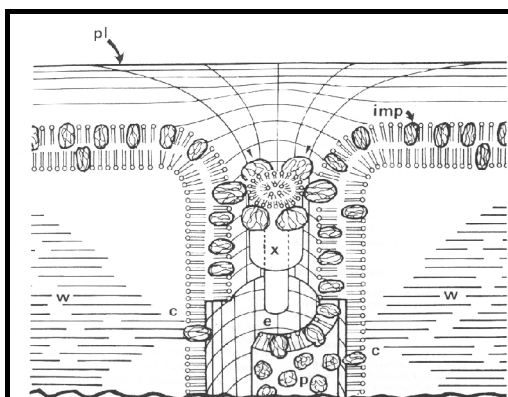


Figure 10.15: Thomson Model

The Thomson Model

The Thomson model (Fig. 10.15), derived from observation of *Tamarix* salt glands, postulates particles in the lumen and embedded in the plasma membrane wall, in order to explain features found in freeze-fracture images.¹⁵⁵

The White Model

The White model (Fig. 10.18,10.20), based on the discovery of actin within plasmodesmata and TEM images showing “blobs” and “spiral blobs”, postulates the existence of filamentous actin strands within the lumen, and raises the possibility of these being twisted around the central desmotubule.¹⁵⁶

The Radford Model

The Radford model (Fig. 10.19, 10.21) extends the White model to take into account the presence of myosin (an actin-associated motor protein) and TEM images showing apparent “spokes” within plasmodesmata.¹⁵⁷

3-D-Models of Plasmodesmata

Four of the above models, namely the Ding, Overall, White, and Radford models, were modelled using the 3-D modelling tool. The modelling tool takes a short mathematical description of the models and renders them as a combination of rods, spheres, cylinders, cylindrical sheaths and helices of spheres. It is also capable of outputting their details in a variety of formats, for use by other programs, such as other real time viewers, ray tracers, and the nanosim program.

The fastest way of observing these models is by immediate rendering using the Silicon Graphics system. This also allows real time examination of the model, as the author’s 3-D viewing program allows basic 3-D manipulation of viewpoint using a combination of mouse movement for rotation, and keyboard operation for panning the view point in and out.

3-D Computer Models of Plasmodesmata; Real Time Views

The following images (Fig. 10.16 to 10.19) are screen shots from the 3-D viewing program, showing the computer models constructed from four of the theories outlined above:

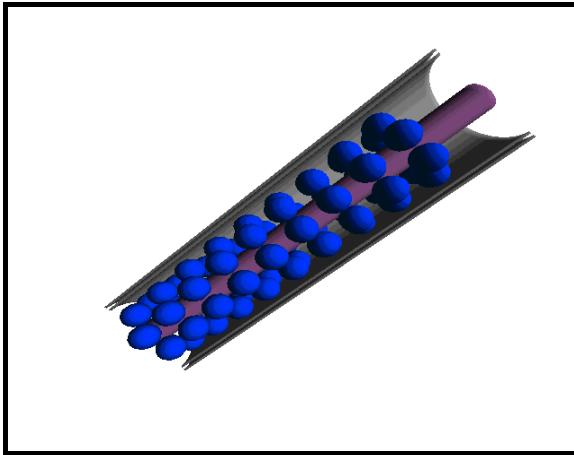


Figure 10.16: Overall Model in 3-D

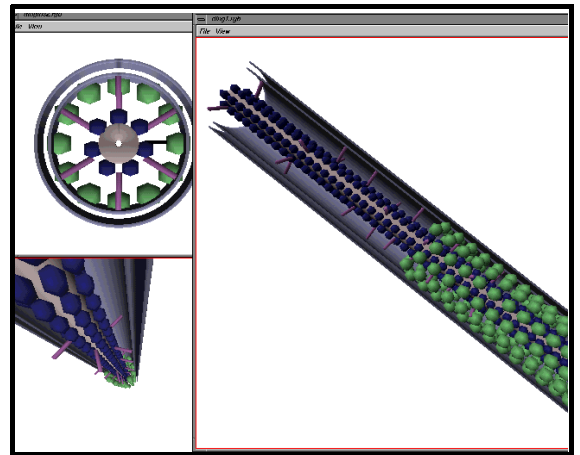


Figure 10.17: Ding Model in 3-D: 3 Views

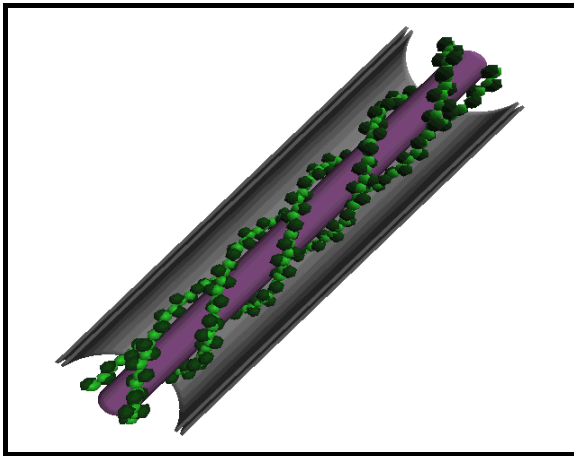


Figure 10.18: White Model in 3-D

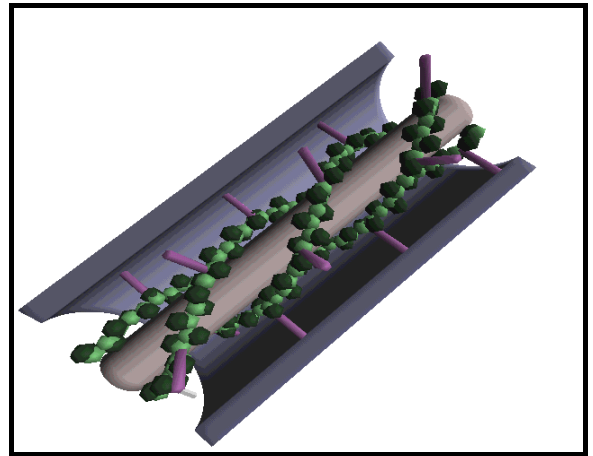


Figure 10.19: Radford Model in 3-D

3-D Models of Plasmodesmata; Raytraced Views

The above models can be very useful for real time viewing and examination, when an impression of the three dimensionality of the model can be obtained by real-time manipulations such as rotating the object, and by viewing from multiple angles. However, for still frame images it can be useful to render the objects in more detail using a ray tracer, to produce an image at a far greater resolution. This is important for high quality print reproduction, and it also provides a number of visual cues such as shadows, reflective effects, and improved shading, which may make the object appear more substantial and thus easier for viewers to interpret.

To this end, as part of the modelling program suite a simple conversion program was written to turn the mathematical description of models (such as those above) into povray raytracer files. This allowed the production of higher quality images such as these renditions of the White and Radford models below (Fig. 10.20, 10.21):

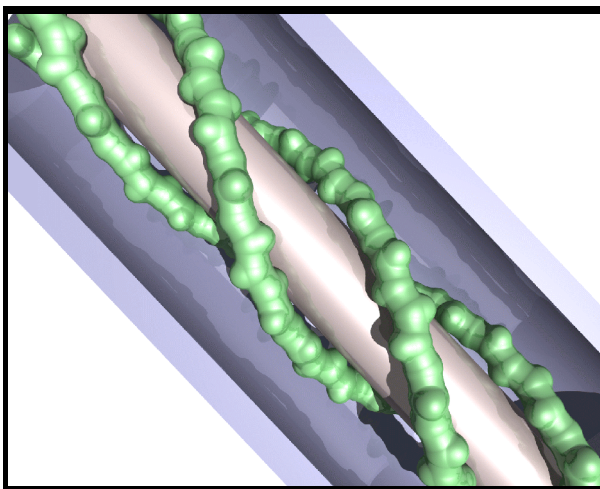


Figure 10.20: White Model, Raytraced

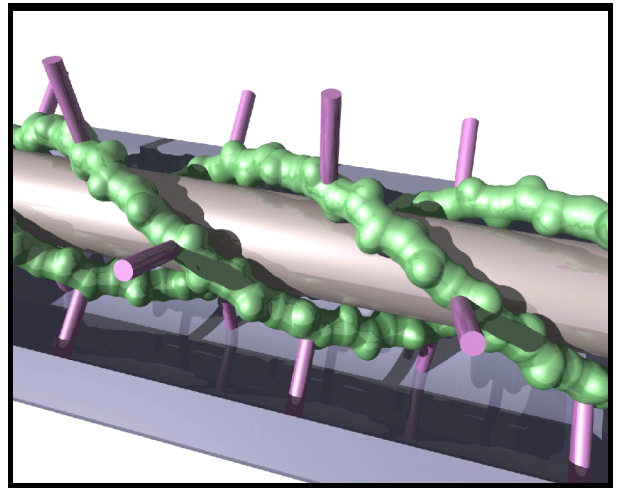


Figure 10.21: Radford Model, Raytraced

Combining Models with the Nanosimulator

Introduction

The nanosimulator was enhanced to allow it to read in pre-initialised structures, with the (current) limitation that such structures had to be static, and would not be able to move like other objects within the simulation. This was easily enforced within the program by giving such objects a diffusion coefficient of zero.

This allowed the input of models, such as those described above, into the nanosim program, where their interaction with other, motile, particles could be simulated. There are a great many possibilities for work combining pre-modelled structures with the nanosim program, even with the current limitation that such structures must be static.

Some such possibilities include:

- modelling the diffusion of particles of various sizes through the different models of plasmodesmata, to compare with experimental diffusion values.
- modelling the growth of microtubules from organised nucleation sites (such as centrosomes and Microtubule Organising Centres¹⁵⁸).
- modelling the interaction of actin strands with the cell cortex.
- modelling the deposition of stain particles on proteins during the staining process commonly applied to specimens prior to electron microscopy.

To examine the usefulness of this technique, this last process of stain deposition will be examined and modelled in some detail.

Stain Deposition

An electron microscope works by passing a focussed beam of electrons through a sample of material, and displaying an image of the resultant attenuated beam. Since some parts of the specimen block the electron beam more than others, an image showing contrasting sections of the specimen can usually be obtained. How “electron opaque” an object is depends roughly on the size of its constituent atoms - light atoms are less likely to interact with the electron beam, while heavy atoms are more likely. So, for example, water is reasonably transparent under an electron microscope while heavy metals such as uranium are generally opaque.

Unfortunately, proteins and other biological molecules are, in general, made of fairly similar light atoms (carbon, hydrogen, nitrogen, oxygen) and so are comparatively transparent to the electron beam and difficult to distinguish from each other. In order to increase contrast between proteins (and also other organic molecules) in a biological specimen, the specimen is often stained with a heavy metal preparation that will preferentially bind to some, but not to all, of the material in the specimen.

After the specimen has been stained, and any excess stain removed, the specimen can be viewed with much greater contrast in the electron microscope. However stain deposition is not a well understood process, and artifacts can occur when stain is distributed unevenly.

A specimen for electron microscopy is usually prepared by embedding it in hard resin, to make the specimen physically stronger. This resin may impede the diffusion of stain particles, or prevent it altogether, depending on the exact resin and the stain used. Large stain particles, such as are used with antibody labelling techniques, usually cannot penetrate the resinous substrate, and will only stain, (or ‘label’) the part of the specimen that is exposed outside the resin. Smaller stain particles on the other hand may be able to penetrate the resin, although they may take some time to do so.

The nanosim program can simulate the movement of the stain particles around a computer model of a 3-D proteinaceous structure (such as those shown in the previous section), with the model being optionally embedded, or only partially protruding from, an impenetrable resinous substrate. A solution of stain particles is modelled as drifting over the surface of the structure, with some of the stain particles binding to the structure. How the substrate is modelled, or whether it is modelled at all, depends on the type and size of stain and the permeability of the resin being simulated.

Simulating Stain Deposition

This method of stain deposition is shown on a variety of cell structures: filamentous actin, microtubules, and one of the plasmodesmata models. By also modelling an impermeable substrate, and applying the stain particles from the ‘top’ of the model, it is possible to also simulate any uneven distribution of stain particles over a sample.

Converting the 3-D Model

As mentioned previously, the nanoscale simulator is capable of reading in a pre-existing structure at start up. However, since the nanosim program is only capable (at the moment) of working with objects composed of spheres, it was necessary to modify the 3-D modelling program to output the existing models in a form that the nanoscale simulator could read. The difficulty here is that the 3-D modelling program deals with complex surfaces, such as rotationally symmetric parabolic or trigonometric surfaces, as well as rectangular blocks, cones, cylinders and such like, in addition to spheres.

This was resolved by saving such non-spherical objects as a large number of component spheres. The method is similar to the well-known computer technique of modelling 3-D objects as a vast number of small, cubical building blocks known as ‘voxels’¹⁵⁹. This increased the computational load on the simulator, since it greatly multiplied the number of objects the simulator had to consider, but the population remained within manageable limits.

Creating a model suitable for the nanoscale simulator was thus a two stage process. Firstly the model was constructed using the 3-D modelling program, and then the model was converted into a vast number of spheres for use by the nanoscale simulator. A small change to the 3-D modelling program enabled the different components of the high level description to be given names, which were set to correspond to defined protein types in the nanoscale simulator's 'protein dynamic description file', or '.pddf' file.

Simulating Stain Movement and Binding

The stain particles were modelled as very small, very mobile, very easily bound particles that started in a layer above the 3-D structure and substrate, and diffused downwards. The particles were assumed to bind to everything except each other. While the actual heavy metal ions that form most common stains are in fact minute on the scale of the simulator, the actual particles were made somewhat larger, to reflect their associated solvation layer water molecules (which slow their movement).

In order to simulate the differential binding of the stain to different parts of the model, the number of binding sites for the different proteinaceous components of the model was varied, or their strength altered. Thus proteins known to stain heavily were given a large number of binding sites, whereas proteins that are harder to stain were given fewer and/or weaker binding sites.

After a period of time representing the staining process, unbound stain particles were removed, corresponding to the washing out of unbound stain particles that occurs during specimen preparation. The resulting simulator state was saved to an output file suitable for the povray raytracer.

Examples of the Deposition Process

Figure 10.23 shows the stain having been deposited on the microtubule model in figure 10.22. The model is not shown actually *in* the sea of stain particles, since the picture would appear as almost entirely black. Only the stain particles that have been bound to the surface of the model are included in the picture.

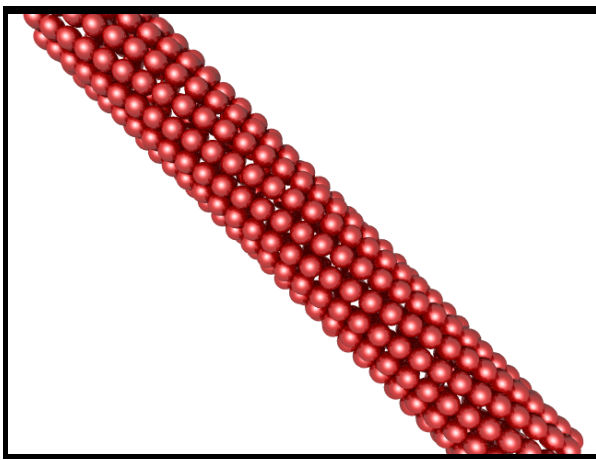


Figure 10.22: The Microtubule model, as component spheres.

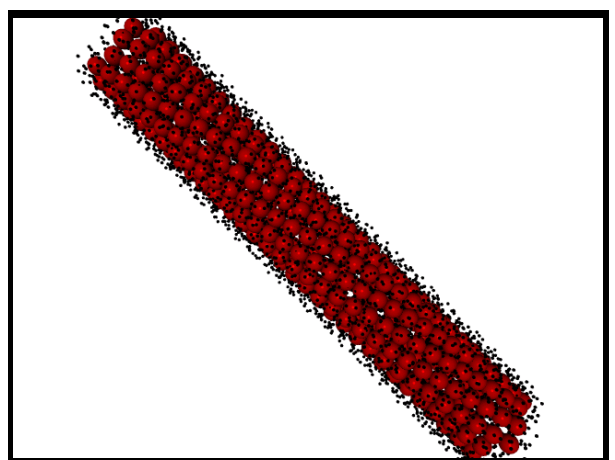


Figure 10.23: Output from the nanosimulator; The Microtubule model, covered with stain.

Modelling Antibody Stain

The type of stain shown above involves a very large number of very small stain particles. A different type of stain is used in antibody staining, where smaller numbers of very large and specialised stain particles are used instead. Antibody staining is usually a two stage process. The primary antibody (a protein that binds very specifically to another protein) is used to bind to the target protein in the specimen. Because the antibody is a protein itself, it is transparent to electrons (or ‘electron lucent’). As the purpose of staining the specimen is to create electron opaque regions, a further stage of staining is done with a secondary antibody to the first antibody. The secondary antibody is conjugated to an electron opaque metal particle. This complex attaches to the bound antibody creating the final stain marker, which indicates the presence of the particular protein species the primary antibody was targeting.

An attempt was made to model this on a cross-section of a plasmodesmata model, showing the action of antibodies staining for actin.

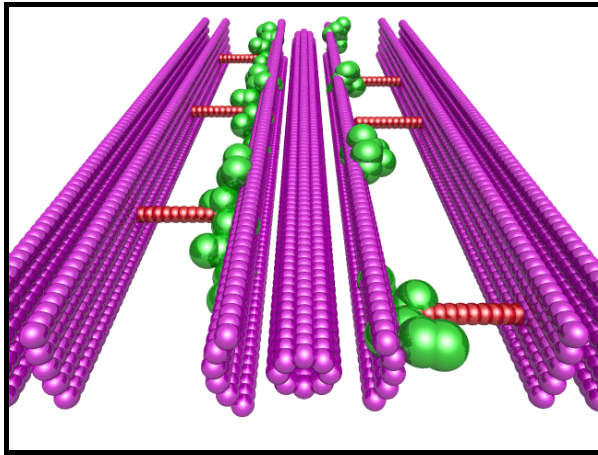


Figure 10.24: Pd: Slice Model

The model used in this example is the Radford Model (Fig. 10.19,10.21). To make our simulation as realistic as possible, the model is imbedded in a ‘virtual resin’, by making the bottom portion of the simulation field impermeable to stain particles, and only the thin exposed surface layer is modelled, giving effectively a thin slice through the model as an active region.

Notice the way the plasmodesma cross-section (especially the various membrane bi-layers) has been remodelled as a large number of spheres (Fig. 10.24). Also observe the many *overlapping* spheres. Although different objects within the nanosimulator cannot overlap, the spheres making up the components of one particular object in the nanosimulator can (this was shown earlier with actin in Chapter 7). In the current simulation this idea is extended. All static objects are allowed to overlap, and no collision checks are performed between two static objects, only between objects where at least one object is motile.

Figure 10.25 shows the primary antibodies, starting above the plasmodesmal section, ready to diffuse (the substrate layer has been made transparent in this image).

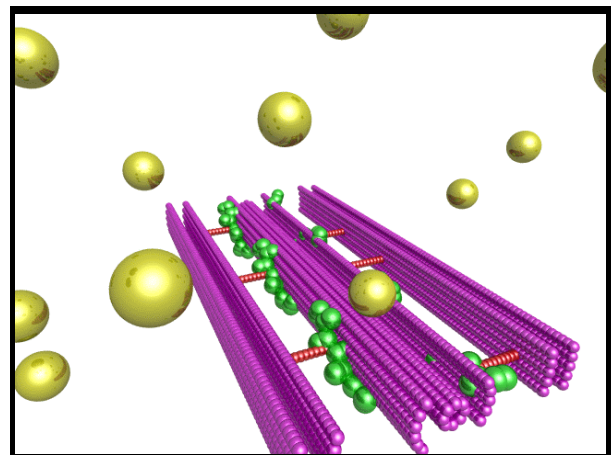


Figure 10.25: Pd antibody staining.

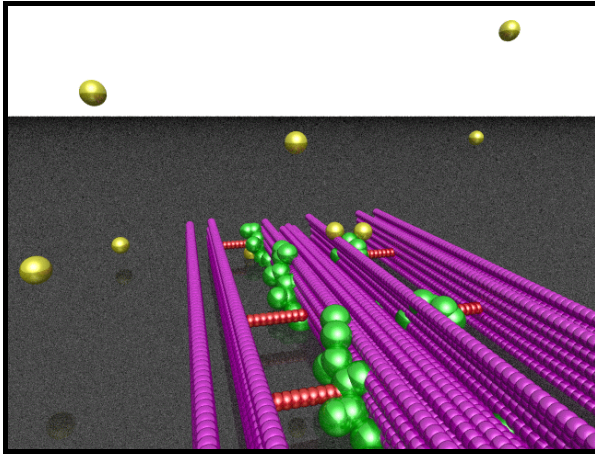


Figure 10.26: Pd with bound antibodies.

In figure 10.26 a period of time has passed: some primary antibodies are free while a few (visible at the rear of the image) have bound to the model below. In this example the model is imbedded in impenetrable resin, beyond which the stain particles cannot move. (The static spheres are not so restricted, and in fact the model is very slightly embedded in the resin).

In figure 10.27 the simulation is finished, and the unbound particles have been removed. Second stage staining by the metallic particle (black) and the secondary antibody (red) has *not* been fully simulated; they have simply been added on to the primary antibody (yellow) in an opposite direction to the link (i.e. perfect antibody recognition has been assumed).

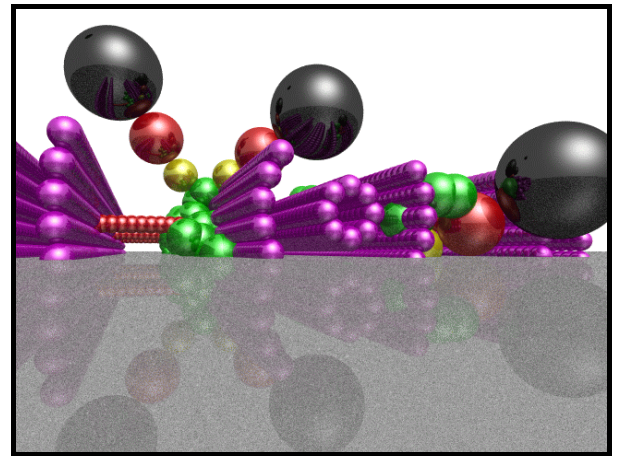


Figure 10.27: Pd with antibodies and stain particles.

The Final Step: Virtual Microscopy

Introduction

A classic problem with modelling cellular structures on a computer is in comparing them with images of the real structures taken from electron microscopy. The computer models are usually too precise and uncluttered, whereas the real structure suffers from the difficulties of staining, contamination, resin embedding, damage during preparation and the limitations of the imaging technique used.

‘Washing’ the Virtual Sample

In a real laboratory staining exercise, after the specimen has been exposed to the stain for a period, the excess stain is washed off, leaving behind only the stain particles that have bound to the specimen. The program emulates this by removing all unbound stain particles after the simulation of deposition has been completed.

Use of a Ray-Tracer

Having obtained our model results from the nanoscale simulator in the previous section of stain deposition or antibody motion, a ‘virtual electron micrograph’ (or VEM) is produced using a ray tracing program that represents stain particles as being very dark, and the rest of the structure and substrate as being translucent. The result is an image of the computer model as it might appear to a ‘virtual electron microscope’, and this enables the researcher to compare their model, to some extent, with experimental data.

This simple approach (i.e. using a raytracing program) to electron microscope modelling is possible due to relative translucency of thin biological specimens. If a denser object was being simulated (such as a mineral sample), it would be necessary to model the multiple interactions of each electron with the sample as the electrons bounced through the material, using more complex techniques such as Monte-Carlo simulation, which uses a randomiser to model a large number of electron paths. However, since the chance of multiple sample interaction with a thin biological sample is very low, this complexity can be ignored, and a single pass model can be used. This can be done easily using a normal ray-tracing program with no perspective adjustments (i.e. parallel rays).

Preliminary Work

An investigation of the basic premise of this idea was attempted before proceeding with the full stain deposition simulation. A simple computer model, which represented stain particles as black dots on an otherwise invisible model was constructed. This simple model did not include any substrate. It produced a simple image, which was then post-processed to reproduce some of the visual flaws of an electron microscope. There are a number of such flaws that could be simulated.

lack of perfect focus

This can be simulated using a small scale blur filter, similar to those in Chapter 2, or by a Fourier transform that removes high frequency components.

uneven resin, impurities and other causes of random brightness variation

This can be simulated using a noise generator, i.e. a simple algorithm that randomly brightens and darkens areas of the image. The simplest example is a point noise generator, that randomly adds black and white dots to an image.

magnetic ‘lens’ defects such as skew astigmatism

Astigmatism can be simulated by a global transformation function over the whole image, using a shear transformation matrix. This is a standard technique¹⁶⁰, where every pixel in an image is repositioned using the formula $p' = Tp$, where p' is the new position of the pixel, p the original position, and T the transformation matrix:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{Formula 10.1: 2-D Vector Transformation.}$$

x', y' - the new position of the pixel

a, b, c, d - elements of the transformation matrix

x, y - the original position of the pixel

When the matrix T is the identity matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ the position of the pixels does not change. If instead the values of a and d are set to some other value than 1, while b and c remain zero (i.e. $\begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}$), a linear shear, or skew, results that emulates the effect of astigmatism.

Photographic film graininess

The hexagonal grain pattern of a real photographic image could be simulated by averaging hexagonal areas, and re-rendering as black circles with a radius giving the same overall darkness to the previously averaged area. The method is very similar to the ray tracing technique of *oversampling*, used to reduce aliasing, when an image is generated (sampled) at a higher resolution than the final image, each pixel in the final image being an average of several samples¹⁶¹. It is possible to do the same with hexagonal pixels, as illustrated below.

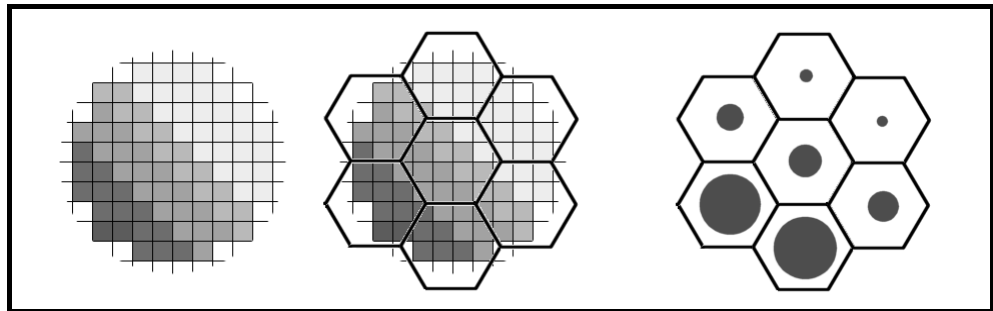


Figure 10.28: Simulating photographic film grains (right) using a hexagonal overlay (centre) of an original pixel grid (left).

In this work, and in the later stain deposition simulation work, only the first two flaws described above are modelled.

Since the model includes no attempt at random stain patterning, it produces very obvious (and unrealistic) regularities in the image. Lines and patterns of points (exacerbated by perspective effects) can be seen which do not appear in real TEM micrographs. Real TEMs do of course have some regularities, and these are very important for researchers determining structure, but the initial images below are far more regular than in real images.

Actin Filaments

Actin was the first structure that was considered. Figure 10.29 shows an actin filament at various stages of image processing - the image second from the right is the final 'virtual electron micrograph'.

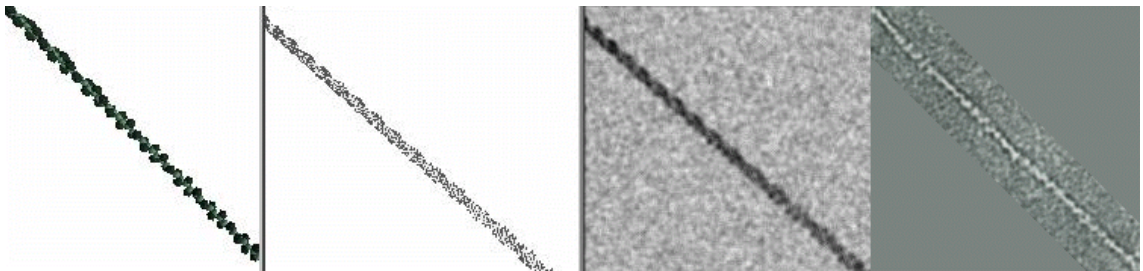


Figure 10.29: VEM modelling of an actin strand: original model(far left); simple VEM (middle left); simple VEM + noise (middle right); and a real TEM (Fig. 10.1) (far right)

Microtubules

In the same manner, a virtual micrograph of a microtubule was constructed. Note the pronounced regularity of the staining pattern.

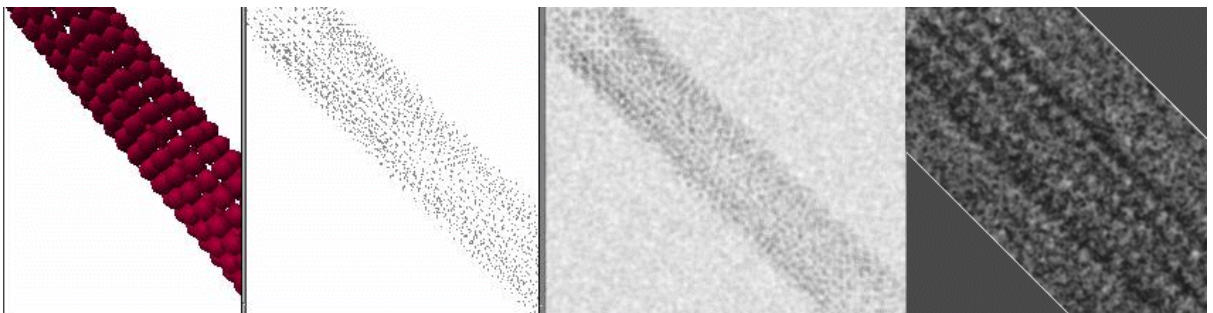


Figure 10.30: Simple VEM modelling of a microtubule: original model (far left); simple VEM (middle left); VEM + noise middle right; and a real TEM (Fig. 10.2) (far right)

Using the Nanosimulator Results in the Virtual TEM

While the above can be a useful method, since it is very quick to produce such an image from a 3-D model, it suffers from not simulating the stain deposition very accurately, which gives rise to the regularities and Moiré patterns shown above. A more complete approach is to use the results from the previous section that modelled in more detail the deposition of stain.

Taking the results from the nanosimulator, the stain particles are rendered as being a dark translucent grey (corresponding to the electron opaque nature of stain), while the model itself is rendered as a far lighter translucent grey (corresponding to the electron lucent nature of most proteins), as was the substrate. To imitate the operation of an electron microscope, all refraction effects were turned off in the ray tracer, so that the 'light rays' the ray tracer simulates would go straight through all the model materials without any prismatic bending or refraction. In addition, the raytracer's rays were made to run parallel (or with less than one minute of arc of divergence), thus removing any perspective distortion, and again modelling the essentially parallel nature of the electron beam in a TEM.

Resultant Images

This method was used on well known structures first, to establish its basic validity, and then on some less well known structures. The first structure simulated in this way was that of a 12-filament microtubule, after which the same technique was used on a hypothetical model of plasmodesmal structure.

Virtually Stained Microtubules

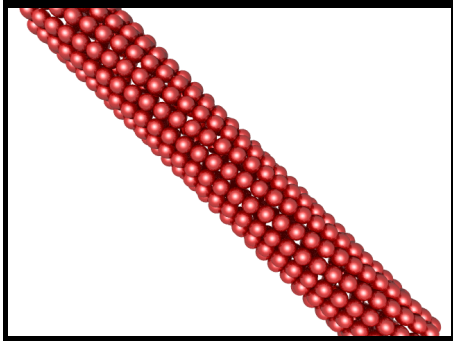


Figure 10.31: MT model

These images show the original microtubule model, and again show the resultant ‘virtually stained’ image from a small, resin-infiltrating stain particle.

Figure 10.31 is the Original Model (From Fig. 10.22).

The Model covered with stain, displayed without perspective, and with stain particles displayed as 150 picometer spheres is shown in figure 10.32 (see also figure 10.23). The stain particles are assumed to be able to attach to any of 25 equivalent binding sites on the tubulin dimers, although geometry will prevent all of these sites being bound to (i.e. other tubulin dimers obscure some of these attachment sites).

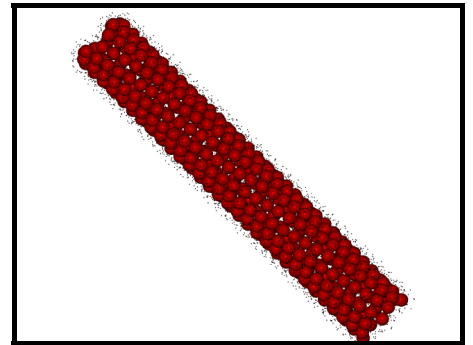


Figure 10.32: MT model and stain

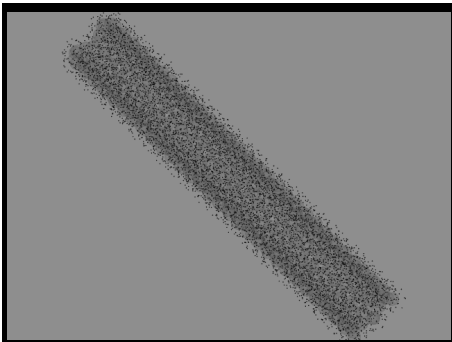


Figure 10.33: MT VEM

Figure 10.33 shows a rendering of the VEM, without any post processing.

Figure 10.34 shows the VEM with random shot noise to simulate TEM imperfections. In this technique, a random 10% of pixels are set (with equal probability) to either completely black or completely white.

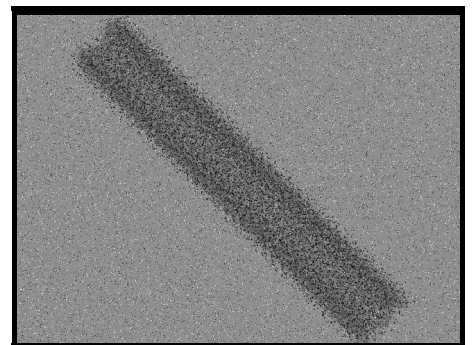


Figure 10.34: VEM with noise

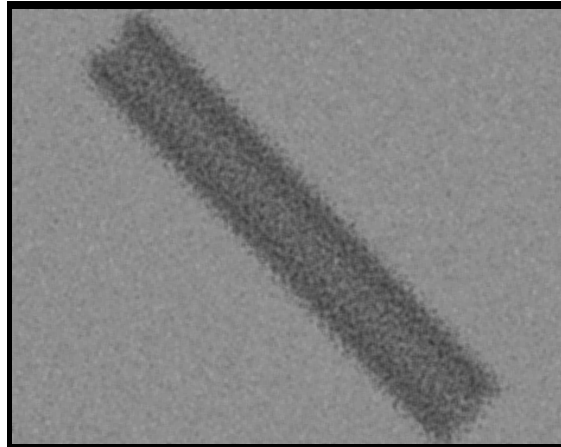


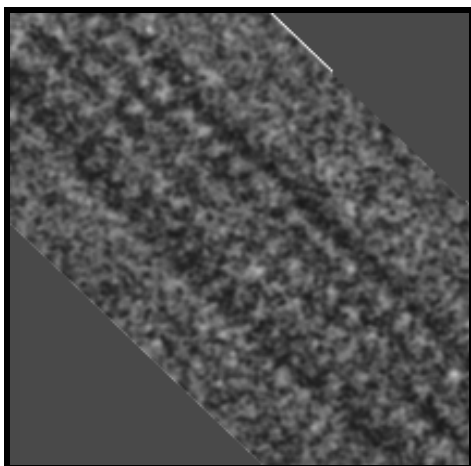
Figure 10.35: Final blurred VEM

Figure 10.35 is the final virtual micrograph, after a blur filter has been applied. The blur was a simple 5×5 averaging mask:

$$\begin{bmatrix} 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.04 \end{bmatrix}$$

In other words, the value of each pixel is determined by placing the above filter over the image, centred on the pixel being blurred, and the resultant pixel is the average of all the neighbouring pixels.

Other more elaborate masks were also used, but gave much the same effect, while the above filter has the advantage of speed, as it can be implemented as a simple summation of all the neighbourhood values, followed by a single division by 25.



For comparison, here is an inset of figure 10.2, rotated to the same angle as above. As can be seen, the above model fails to reproduce the regularities of the real TEM, probably due to inadequacies in the staining model being used, which assumed an even distribution of stain attachment sites on the modelled tubulin dimers (these can be compared with the *unstained* VEMs presented later in this chapter).

An added factor is that the above model is a twisting, 12-filament microtubule, while the above picture is presumably the more common 13-filament type, however the difference is not actually that great; images of 12-filament microtubules show a slight change along their lengths as their filaments spiral, but preserve the basic features of this photograph.

(Further study of microtubules, done without simulated staining, is done after the next section)

Studies of Plasmodesmata

Next, a hypothesised plasmodesmata model (the Radford model, cf Fig. 10.19,10.21) was examined. This particular model includes a number of twisted actin strands (green), as well as some unusually small myosin proteins (red) and a number of membranes representing the desmotubule (the inner cylinder) and the outer plasma membrane of the plasmodesmata (the outer skin of the model). The labelling of this model by a primary antibody, followed by a secondary antibody with an associated gold particle, is shown in figures 10.24 to 10.27.

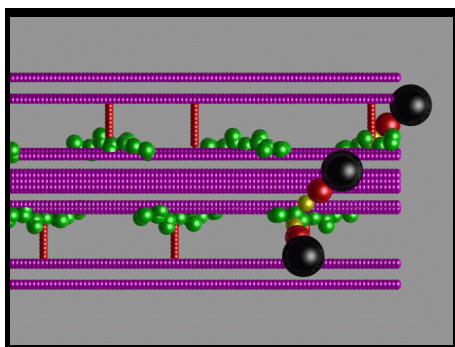


Figure 10.37: Pd VEM model

Figure 10.37 shows the plasmodesma, labelled by a number of small antibody particles. It is identical with the model shown in figure 10.27, after the stain deposition modelling process (Fig. 10.24 to 10.26) has occurred. This image is simply taken from above, with no perspective.

Figure 10.38 is the raw VEM, showing the plasmodesma embedded in resin, being labelled by three large antibody particles. Note that the unstained proteinaceous components are comparatively faint next to the completely black antibody heads, representing the gold spheres used in this type of antibody label.

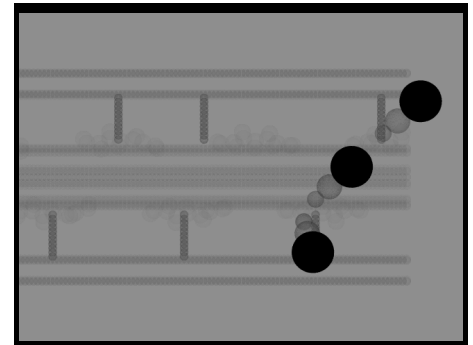


Figure 10.38: Pd VEM after noise and blur applied.

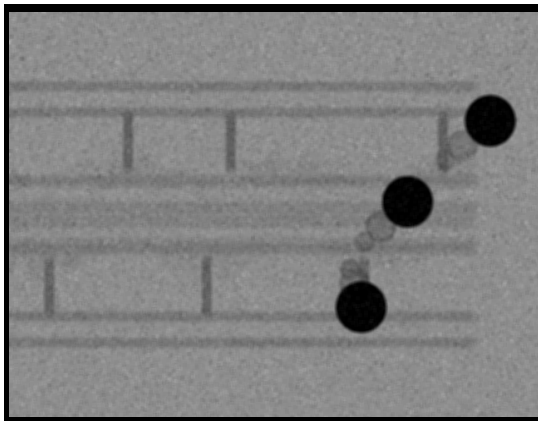


Figure 10.39: Plasmodesma VEM

Finally, figure 10.39 shows the VEM with the same shot noise and blurring used previously (i.e. 10% of pixels set evenly to black or white, followed by a 5×5 averaging filter).

This can be compared to a real image (Fig. 10.40) of antibody labelled plasmodesma (the plasmodesmata is running vertically, at a slight angle, and the position of the stain particles are marked by the black indicating arrows).¹⁶² This image is slightly misleading however, since in addition to antibody labelling, it has also been post-stained with a traditional heavy metal stain, as was modelled in the microtubule example previously.

As can be seen the images are similar only in broad outline. The grainy appearance of this particular real TEM (probably caused by stain clumping, and possibly some degree of photographic development artifacts) is modelled poorly by pixel level shot noise: a more advanced approach would be required, possibly some variation of the hexagonal 'photographic film graininess' approach, combined with a modelling of the secondary staining process.

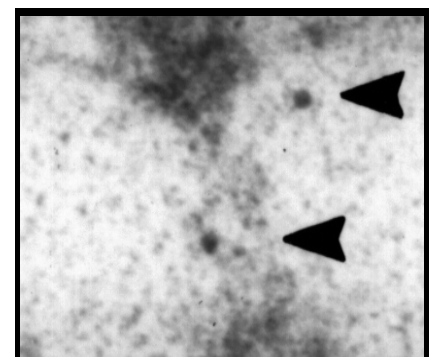


Figure 10.40: Plasmodesmata TEM

Virtual Microscopy without Staining

While modelling stain deposition is a challenging problem, useful results can be obtained (in real microscopy as well as virtual) without stain. While the contrast is fainter, very clean specimens such as *in-vitro* assembled microtubules can still produce excellent results.

Some exceptional work on *in-vitro* assembled microtubules has been done by Denise Chrétien of the Laboratoire de Biologie Structurale in France, using highly purified tubulin polymerising in thin films, which are then rapidly frozen and examined on a cryogenic stage in an electron microscope^{163,164}. In particular the work suggests some interesting features of assembling microtubule ends, indicating that they are curved and splayed out, as is shown by the following image (Fig. 10.41)¹⁶⁵. Chrétien et al. interpret these (and other) images to suggest that microtubule growth occurs when an initially flat sheet grows and curls into a closed microtubule.

To demonstrate further the use of virtual electron microscopy, a static model (not from the simulator) is constructed displaying some of the curved end features postulated by Chrétien et al. (i.e. a double-curvature growth spike, but not the flattening effect, which is shown in the upper diagram within figure 10.41).

The model was constructed in povray using the file shown in Appendix G. It was then blurred, and noise added, in order to simulate the imperfections of electron microscopy.

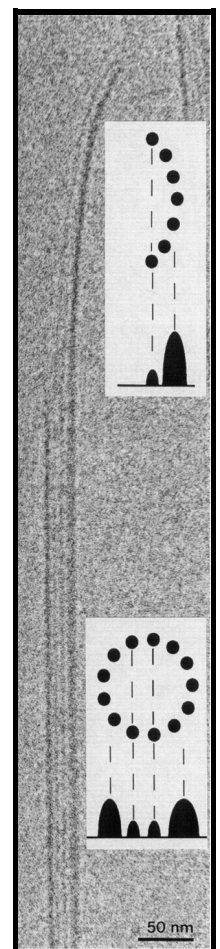


Figure 10.41:
Chrétien
microtubule.

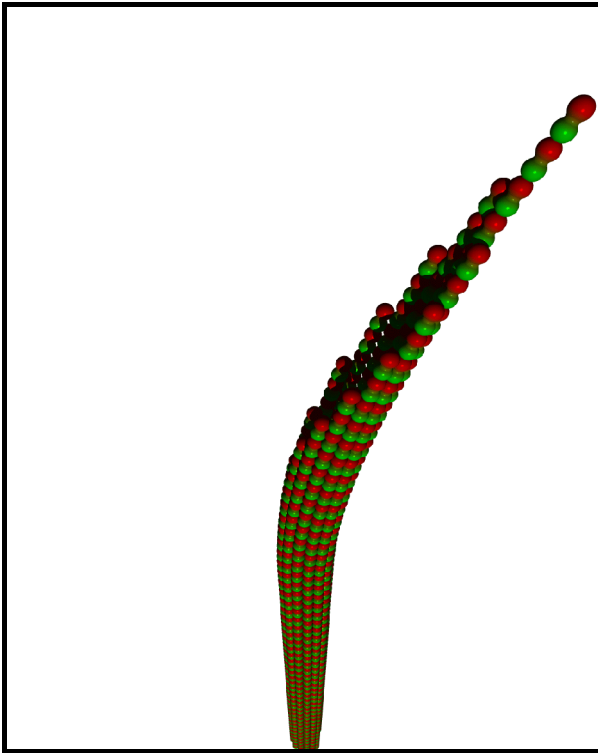


Figure 10.42: Perspective view of mt model.

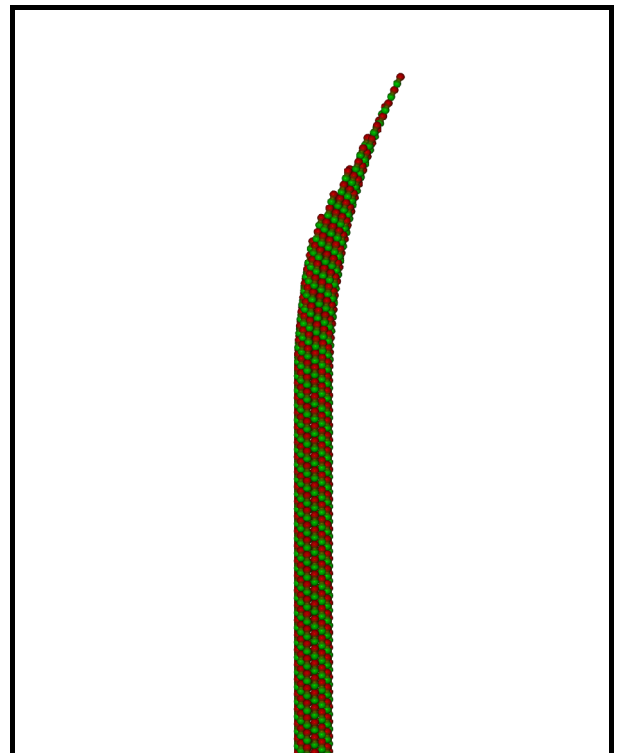


Figure 10.43: Orthographic (no perspective) view of model

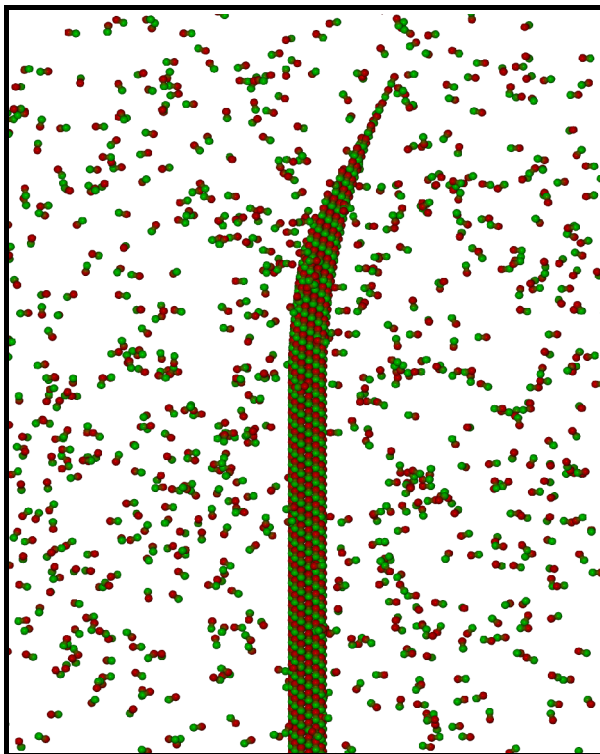


Figure 10.44: Orthographic view with free dimers

Images 10.42-10.44 show the modelled microtubule, with the curved, open end. The first image (Fig. 10.42) is a perspective view of the model, showing the nature of the doubly curved open end. The second image (10.43) is an orthographic view of the model side on. The third image (10.44) shows the model (again with an orthographic view) within a randomly positioned cloud of dimers (statically modelled, rather than simulated).

Simulating an electron micrograph using the model from figure.10.44 produces the image shown in figure 10.45. Note the unrealistically crisp resolution of the image. The image is produced using a light source directly behind the model, and by making the dimers translucent. The dimers are also shrunk slightly, to represent the more electron opaque core of the protein (hence they appear as double spheres rather than dumbbells).

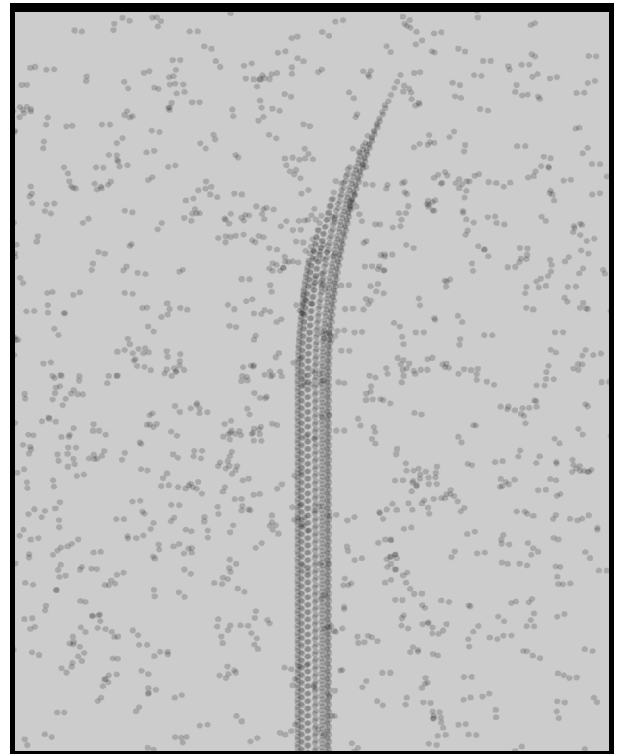


Figure 10.45: The raw VEM, without noise or blurring

In figure 10.46 a Gaussian blur of radius 4 pixels is applied (the image is 1280x1024 pixels in size). A small amount of pixel level noise is added to give a slight graininess to the image.

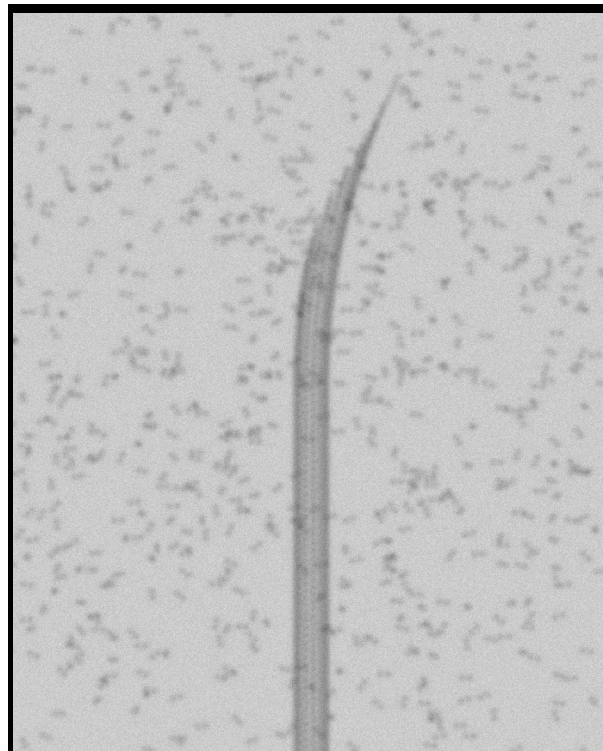


Figure 10.46: The same image after blurring and noise.

As can be seen, the VEM faithfully reproduces a number of the features of the real micrograph (Fig.10.47), such as the banding along the body of the tube, and the relative intensity of those bands.

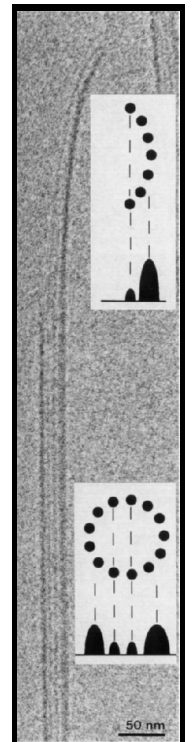


Figure 10.47: microtubule TEM

However, the features of the growth tip are not identical, which is a useful result, as it indicates that the model used (a curved, tapered section of the normal microtubule with the same cross-sectional curvature) is probably not what realistically occurs in nature (which is likely to be a thinner spike with fewer protofilaments¹⁶⁶).

The VEM image retains rather more contrast and resolution than the original, but although it could be degraded further there is probably no need. The close modelling of striations and the relative strengths of the striations, especially in the closed body of the microtubule, is an excellent example of the strength of the technique. VEMs of this sort (i.e. without staining) can be made relatively quickly, probably in real time, and are a useful adjunct for the protein modeller. To demonstrate, the following images (10.48, 10.49, and 10.50) of microtubule fragments are taken from a simulated microtubule construction run from Chapter 8:

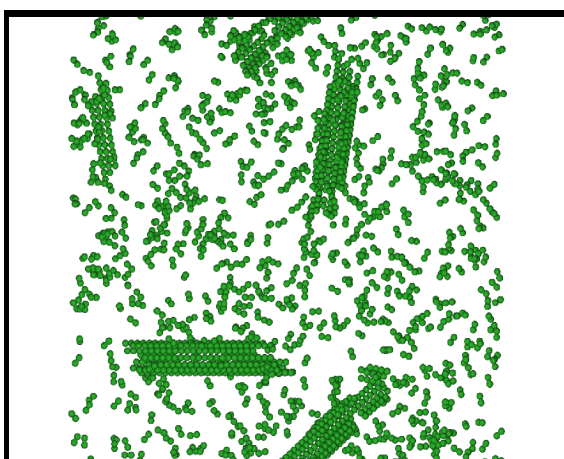


Figure 10.48: Original simulator output

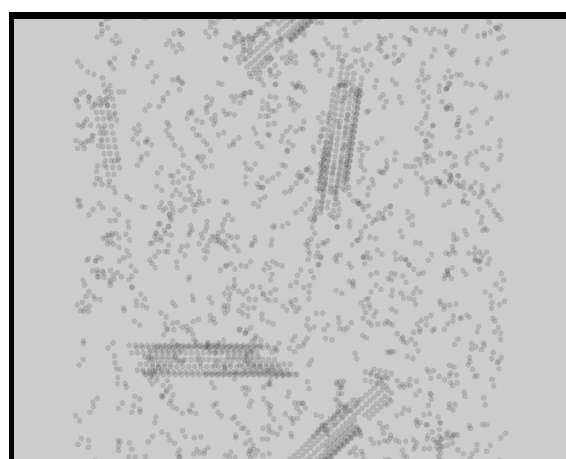


Figure 10.49: Raw VEM image

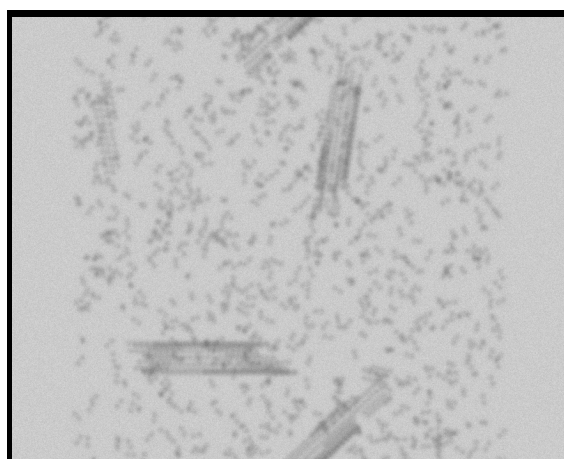


Figure 10.50: VEM Image after blurring and noise added.

Discussion

Creating a virtual micrograph of a computer model is simply another way of looking at a model - it neither proves, nor disproves, the model's validity. It may however *suggest* that a certain model is more accurate than another, by showing that the protein structure corresponding to a given model would be more likely to produce the TEM images observed than the protein structure corresponding to another model.

A difficulty with this method is that the actual process whereby stain particles bond to proteins is not entirely understood. In modelled structures containing unknown or unidentified proteins, it can be particularly difficult to decide how well stain particles will bind to a particular component of the model. For example, when a very regular stain deposition was assumed (e.g. Fig. 10.30) the staining was unnaturally regular, which created visual artifacts. When a more random staining model was assumed (e.g. Fig. 10.35) there was possibly not *enough* regularity to properly model the staining process, since the degree of regularity found in real TEM micrographs was not observed (e.g. Fig. 10.2).

A further complication, shown by figures 10.39 and 10.40, is that the simple staining model used does not handle the interaction between stain particles, which sometimes results in stain clumping into electron opaque aggregates, that can again give an image a grainy appearance.

The usefulness of the technique is shown in figures 10.41 to 10.50, which demonstrate how virtual electron microscopy can support a structure hypothesis, by confirming the observed banding and edge density effects shown in microtubule TEMs.

Conclusion

This chapter shows how the principle of the nanosimulator can be extended to model the interaction of particles with static structures. It illustrates one way that static structures could be incorporated into the nanosimulator, and then showed the nanosimulator in operation, using both static and dynamic objects, and links between these two types of object.

This chapter has demonstrated some of the wider possibilities for the nanosimulator and how it can complement traditional image processing techniques. A number of image processing techniques were investigated, and the results used to improve raw TEM images.

The concept of virtual electron microscopy was explored, where 3-D models of biological structures are algorithmically manipulated in a way that emulates the physical processes involved in obtaining and developing a transmission electron micrograph. A brief investigation of the defects and artifacts introduced in the physical micrograph was made, with some suggestions for future work.

Extending the idea of the virtual micrograph, an investigation was made of how the staining process used for electron micrograph specimen preparation might be modelled using the nanosimulator. Preliminary results were obtained that showed promise, however it was also shown that there was need for a more sophisticated model of staining.

The illustration of staining is only one of many possibilities for modelling nanoscale structures. Modelling diffusion and structured growth will also be feasible using the existing program, and doubtless many other types of simulated experiments will also be possible.

Chapter 11

Accuracy and Validity of the Nanoscale Simulator

Nunc est disputandum

(Now is the time for discussion.)

- Horace (adapted)

Introduction

The two questions that must always be asked for any computer simulation are "How reliable is it?", and "How well does it reflect the reality of the process being modelled?". The first question is usually easier to answer than the second. Previous chapters have compared the results of the nanosimulator with experimental data to show that the simulator broadly models the general form of the experimental data for the proteins actin and tubulin. The question of how well it actually simulates the modelled process is more difficult to answer, since direct experimental evidence of the low-level processes involved is sparse and difficult to interpret.

This chapter examines the nanoscale simulator's theoretical utility, and the degree to which its results should be accepted as an accurate reflection of the underlying physical processes being simulated.

Setting the Scene: Other Simulation Programs

In order to clarify these two basic questions regarding the accuracy of the results, and the faithfulness of the model with regard to the process being modelled, it may be instructive to briefly examine and discuss a number of other areas of computer simulation in different fields.

Physics

Some systems have features that are well described by a small number of analytic equations, for which simulations can produce accurate, repeatable predictions of physical phenomena, often to an arbitrary degree of accuracy limited only by computational resources. Examples of such systems can be found in any physics text book and include planetary motion, wave propagation, relativity, simple mechanics and so on. An example of practical relevance is the increasing use of computerised car crash simulations as a cheaper and more informative alternative to physical controlled crashes¹⁶⁷.

Frequently these systems have limits beyond which their behaviour becomes chaotic (as in the gravitational n-body problem). Theoretically, these systems can still be accurately simulated, but such simulation may be impractical due to errors in measurement of the initial state, or limitations of computing equipment. Within defined limits such simulations make strong predictions, and strongly model the underlying physics of their system.

Meteorology

Complex systems may be well-characterised by physical laws, but may be difficult to simulate due to ‘chaotic’ behaviour (extreme sensitivity to initial conditions causing greatly differing outcomes for otherwise very similar initial states).

An example of such a system is turbulence¹⁶⁸, which on a large scale is largely what hinders weather prediction. Weather prediction attempts to model a highly chaotic system, which limits its usefulness because in addition to imperfections in the model (it is computationally infeasible to model every molecule of air for a year), the initial conditions are not known (it is not physically possible to measure every molecule of air before beginning the simulation). As a result, weather models must simulate their initial conditions before simulating future predictions, leading to inaccuracy that is magnified by chaotic effects over time. This has the all-too-familiar

effect of making weather predictions increasingly unreliable the further they lie in the future¹⁶⁹. Weather simulation makes only moderate predictions, despite strong modelling the underlying physics of the system.

Expert Systems

The reverse of this may occur when a workable *ad-hoc* heuristic takes the place of detailed low-level modelling of a system¹⁷⁰. For example both neural networks^{171,172} and expert systems¹⁷³ have been used successfully to study protein conformation and sequence in ways that may have strong predictive power, based on an analysis of existing data, without necessarily any modelling of underlying process whatsoever.

Economics

Some elaborate simulations have been made of systems whose complexity appears to defy analysis. For example, a number of simulations of economic activity have been made which appear to have extremely weak predictive power, while still (debatably¹⁷⁴) modelling the underlying process closely¹⁷⁵.

Biology and Physical Chemistry

Simulation of biological systems is common, ranging from those related to this thesis such as microtubule assembly^{176,177}, virus assembly¹⁷⁸, and actin filament bundling behaviour¹⁷⁹, to atomic level simulations of diffusion and liquid behaviour¹⁸⁰, detailed protein modelling¹⁸¹, modelling of membrane bilayers¹⁸² and to much larger simulations of organs¹⁸³, organisms and ecosystems.

Due to the great complexity of biological systems, most simulations occupy a middle ground, using an abstracted, but still relevant, model of the system. Predictive power can be weak, where a model is demonstrated to show the plausibility of a process without making strong

experimental predictions. For example, the virus work of Berger et al.¹⁸⁴ which demonstrates strongly the plausibility of their local rules based virus model, without making predictions about any particular virus.

Alternatively a model may make very strong predictions, such as protein structure predictions, which specify atomic position to within a few angstroms, and which can be verified by other researchers.

Accuracy and Predictive Power

It is instructive to evaluate the nanosimulator in this context. Does the simulator produce accurate results, and how well does it model the underlying processes it is attempting to simulate? It is also necessary to consider how much initial data the simulator requires. If the simulator requires so much data that it could not fail to produce accurate results, then despite its accuracy it is still not a useful model.

The simulator does in fact require quite a lot of data, not all of it evident. It requires a reasonably advanced knowledge of the physics involved, for instance. But, since that is common across all the different simulations it attempts, it is more pertinent to ask how much specific information about each type of object or protein it requires.

The answer is that it needs every active binding site specified for every active state, with both the breaking and binding strength of each binding site fully specified, as well as the events that cause it to change state. It further requires a model to handle the breaking and binding of multiple links. It can also use the overall diffusion constant for the object, although it will work this out on theoretical grounds if this is unknown.

From this input the simulator predicts a large amount of information. The simulation produces results for the overall polymerisation behaviour for any concentration, as well as describing the nucleation behaviour, and the behaviour of the growing or shrinking aggregate. It can attempt to simulate more complex situations involving multiple chemicals, and it handles a number of

physical processes (such as changes in local concentrations caused by polymerisation) that are very difficult to include in traditional mathematical treatments. By allowing the investigator to examine the minutiae of the polymerisation (e.g. the histories of individual polymers, the evolving histograms of the sizes of polymers over time, and so on) the simulation makes available a great deal of detail not currently accessible to experimental researchers.

The simulator produces data that cannot currently be obtained by other means, namely details of protein interactions. If the gross results (i.e. polymerisation curves, histogram distributions) are accurate, then these other details are potentially useful. However as they cannot (yet) be experimentally confirmed they must remain hypotheses.

As experimental techniques become more and more sensitive, and knowledge of protein interactions grows, it is likely that it will be possible to test more accurately the predictions of the nanoscale simulator, and at the same time fine-tune it to a greater extent than is currently possible.

Validity of the Model

The second question is more difficult: how well does the simulator reflect the physical chemistry of the polymerisation of objects? It should be clear that in order to deal with the physical and temporal scales of interest, the simulator necessarily takes an abstract view of many of the details involved in the physical and chemical reactions taking place.

Depending on the questions being asked, and the particular insights being looked for, this can often be an advantage. Certainly, in an examination of the exact way an antibody binds to a particular viral coat protein, or the precise way in which the conformation of a protein changes to expose a binding site, this simulator will be of little use. Its purpose is to model the details of larger scale assembly and disassembly, and for that purpose the level of abstraction is quite appropriate. It does not matter *exactly* how a site binding, a bond breaking, or a state change of an individual object or protein occurred, what matters is that it *did* occur. At this level of

individual objects the physical reality of the reaction is indeed being simulated, even though the abstractions required mean that it is not making particularly strong predictions.

At the next level up, the simulator *does* make predictions. The way that objects are initiated (at least *in vitro*), the way that aggregates grow and break down, are all predicted in detail. Unfortunately though, this level of detail is just outside the range of current experimental apparatus. Although it is possible to see in much greater detail than the simulator through an electron microscope, it is not possible to observe dynamic, active systems at the nanometre scale. And although these systems can be observed growing and changing using light microscopy, it is usually possible to see them only as tiny lines or filaments, and no finer detail can be made out.

This does however allow the testing of some of the gross predictions of the model, such as growth and shrinkage rates, overall polymerisation masses, and so on. Whether one then accepts that the underlying model is accurate, given that the simulator produces correct answers for these more measurable values, is a nice exercise in epistemology. There is reasonable suggestive evidence that this intermediate level of simulation is accurate, because the physics of movement and collision have been included in the simulation, and the top level results (e.g. polymerisation curves) are approximately correct. But this is only indirect evidence - without experimental support it is not possible to firmly claim that this is the method by which such objects interact.

Summation

Until the interactions modelled by the nanoscale simulator have been examined in more detail, the results of the program must be treated with caution. Due to the abstracted nature of the model implemented by the program, there is a certain degree of resilience to the underlying physical chemistry involved, but there is still a dependency. The limitations of the simulator in other respects (conformation change, treatment of the energetics of multiple bonds, and hence of their breakage and binding probabilities, elasticity and so forth) also constrain its scope.

The paucity of experimental evidence at the level required to fully model individual monomers also restricts the simulation, but this problem is also an opportunity, because the simulator may be able to shed light on some details by demonstrating that certain configurations are unlikely to give rise to the observed large-scale results.

In the meantime one of the great attractions of this technique is the way that so many large scale properties, such as diffusion, concentration gradients, growth and decay cycles, structural geometry, and complex interaction cycles all emerge from the base level description of the fundamental units involved. Even in the absence of strong experimental predictions, the modelling of process is an important and interesting result - the fact that the modelling of the process gives rise to experimentally testable predictions is an added benefit.

Chapter 12

Conclusion

Finis coronat opus

(The end crowns the work.)

- Roman Proverb

Introduction

This thesis has demonstrated how modelling of self assembling nanometre-scale structures can be achieved. It has outlined the basics of the physical chemistry required, shown a design for a computer simulation program, implemented that program, and used it to simulate a wide variety of biological systems.

Reviewing Theory

The basic theory required to build a nanoscale simulator was discussed in chapter 5. The standard mathematical background to the relevant physical chemistry was covered, along with a discussion of various heuristics that can be used to calculate required details such as the approximate moment of inertia of aggregates of arbitrary shape and size, the rotational behaviour of such aggregates, and the required steps to model Brownian motion and collisions in a computationally non-intensive manner.

In addition, the concepts of protein states and events were introduced, with a discussion of how a computer program might be able to model these features. The various difficulties that must be overcome when translating such theory to a computer program were discussed, such as handling boundary conditions, and the difficulty of simulating large numbers of particles.

Implementing Theory

A general purpose program implementing the basic theory was created, and is described in Chapter 6. The program must handle a myriad of details, ranging from correct modelling of the theory covered in Chapter 5, through to a large number of purely programmatic details.

Modelling the theory correctly required many mathematical classes to be implemented, including classes for manipulating vectors and matrices. Also required were a large number of data structures for storing, and allowing fast access to, the many physical attributes of the simulated structures.

More difficult than the above was implementing the complex data structures required to represent the objects. The program accesses its simulated proteins in a variety of ways. Sometimes it searches for them by spatial position, which required a 3-D grid data structure. Sometimes it accesses them linearly, requiring them to be kept in a list. Sometimes they are accessed through the transient aggregates they form as they are incorporated into, or expelled from, growing clumps of objects. Since the simulator implements not a static, but a dynamic model, every protein was a small finite state machine, which requires the program to track the state of each protein, in addition to physical position and mechanical properties. The program must also change this state appropriately when specified events occur, such as the collisions that are also monitored.

Testing the Program

It was demonstrated that the simulation was consistent with the experimental results obtained for the well understood self-assembling protein actin. The results of Chapter 7 showed the same broad behaviour as is predicted for actin, with polymerisation curves of both the entire simulated volume, and individual actin filaments, behaving as expected. The relatively small number of actin monomers simulated (thousands, rather than thousands of millions), however, meant that bulk comparison was not possible.

Experimenting with Tubulin

Results were also obtained for the more controversial protein tubulin. This was a difficult area of simulation, and two popular models were demonstrated, without a firm decision of which was preferable being reached. The power of the nanosim program was shown, however, in the relative ease with which different models could be trialed, without any necessity to rewrite program code, but simply by modifying the text file (the .pddf file) passed to the program.

Extending the Range of the Simulator

To show the breadth of possibilities for the simulator, a number of other (abstracted) self-assembling structures were also demonstrated. Systems such as two-dimensional crystals, three-dimensional crystals, and various simple platonic solids were shown to be easily reproduced by the program.

In addition to these, an attempt was made to model a large, non-trivial system. A virus model, consisting of four distinct protein capsid types, was demonstrated to be able to self-assemble into intact viral capsids. This is an important result, demonstrating that the program can handle

heterogeneous populations of protein objects with large numbers of links. It also highlighted the difficulty of constructing such complex models by hand, because implementing the virus required a great deal of hand calculation and a very long .pddf file. For future work, a more human-friendly method of creating these data files will be required.

Combining the Simulator with other Modelling Techniques

Finally, it was shown how this technique of simulating nanoscale proteins can be extended even further by combining it with other simulation techniques such as image processing and 3-D modelling. Some results of such techniques were presented in Chapter 10, along with an investigation of ‘virtual electron micrographs’, which attempt to process a 3-D model in an equivalent manner to how a biological structure would be viewed by an electron microscope.

These ‘virtual micrographs’ were used to study a large, pre-assembled biological structure, the plasmodesmata found in plant cell walls, as well as pre-prepared models of large structures such as actin filaments or microtubules. ‘Virtual micrographs’ were shown both of raw 3-D models, and of models where the nanosimulator was used to model the deposition of stain.

Although the technique is still in its infancy, it provided some interesting results, and may reward further investigation as another instrument in the scientific modeller’s toolkit.

Discussion of Limitations

Some limitations of the current approach were discussed, the primary difficulty being obtaining hard data on the behaviour of the inter-protein links, especially the behaviour of multiple links. This is an area that would reward further work, and a more rigorous modelling of the inter-protein interactions would improve the accuracy of the simulation, as would more detailed experimental data.

Summary

Despite some limitations, the predictive power and wide scope of this technique show great promise as a tool for both experimentalists and theoreticians grappling with the difficulties of understanding the low level behaviour of proteins and other nanometre-scale objects. Possible beneficiaries include not only biologists, but potentially workers in such disparate fields as medical technology, geology, chemistry, materials engineering, and of course the emerging field of nanotechnology.

It may also be useful as a low-cost 'virtual laboratory' for researchers to trial new techniques, new drugs, and new theories, without the expense in time and materials of lab work. In the same role of 'virtual laboratory' it may be a powerful tool for teaching and learning, allowing students and researchers to interact with virtual molecules, obtaining a more intuitive understanding of the processes involved.

In general, the applications of a computer simulation of this type have a remarkably wide scope for the study of biological systems. Although a variety of simulation techniques have been used on an atomic and molecular level for some time, no equivalent set of tools exists for the protein biologist studying 'large' structures. It is hoped that this thesis may provide a step along the path to producing such simulations, which may, at some stage in the future, reach the ultimate goal of being able to simulate at the protein level all the biological processes necessary to create a 'virtual cell'.

Future Directions

There are a large number of possibilities for extending this program, and this approach in general. This section briefly lists some of the opportunities that exist to use and extend the nanosimulator.

Modelling Further Biological and Non-Biological Systems

The simulator might be extended to cover a wider range of self-assembling systems. There are a large number of biological systems that might benefit from simulation in this way. The technique could be expanded to cover other self-assembling structures such as membranes, and with the future addition of conformational state changes, might even be used to simulate cellular machinery such as the actin-myosin motor. It is hoped that other researchers will benefit from, and extend, this program, and will be able to use and improve the theoretical techniques it is based upon. The “nanosim” nanoscale simulation program has been deliberately made quite generic, and new proteins and structures can be simulated simply by altering the text-based data files that the program reads in when it first starts operating (Appendix B).

Modelling Drugs

Some benefits might be obtained by modelling the effect of various medicinal chemicals on the growth of biological aggregates. Some anti-virus drugs attempt to inhibit or modify the construction of the viral protein coat. Using the simulator it may be possible to test small modifications to the binding sites of such drugs to determine the most efficient way of disrupting the coat construction. Similarly, several chemicals affect the growth of microtubules, such as *colchicine*, *nocodazole* and *taxol*, most of which are derived from various plants. Since microtubules play an important role in cell division, and hence in the growth of cancerous cells, the less toxic drugs of this type, such as *vinblastine*¹⁸⁵, and *taxol* variants¹⁸⁶ are used in cancer

treatment. Drugs like these that affect the cytoskeleton might also be usefully modelled using the simulator.

Modelling Conformational Changes

The program and .pddf file structure has been designed with the intention of including changes in the shape of objects, and changes in the positioning of linkage sites, as the objects change state. This would allow more accurate simulation of systems such as the fraying ends of microtubules, and other systems that involve motile components, possibly extending to dynamic protein systems such as actin-myosin complexes¹⁸⁷, or possibly structures such as ionophores and membrane transport carrier proteins. In fact, being able to model motile proteins would allow a very wide field of study to be simulated, with a huge range of biological processes being amenable to this approach.

Modelling Motion within an Aggregate

The simulator could be extended by including movement within bound aggregates: at the moment they are completely rigid, which is physically unrealistic. Including internal motion might also allow the simulation of bending and internal stress in a structure. A more elaborate simulation of the internal dynamics of objects might also be able to model periodic oscillations or waves that may occur in some structures.

More Detailed models of Binding

The simulator has taken a very high level, abstracted view of the binding process between objects. But there is no reason why a more sophisticated and detailed interaction model could not be incorporated. Ideally this binding code would be separated into a separate class, which could be rewritten by researchers without reference to the rest of the program.

Some features that might be dealt with in more detail by such a model could be:

- the distributed nature of hydrostatic forces;
- the mathematical relationship between the strength of a 'single' bond between a pair of proteins as compared to the bonds between multiple proteins; and
- whether intermediate binding states require explicit modelling.

Improving Program Features

The nanoscale simulator can be improved programmatically in a number of ways. For example:

- many of the algorithms used could be further optimised for speed.
- It could create VRML output data for web based transfer of 3-D data.
- saving and reading files that include state and linkage information, to allow programs to be stopped and restarted, and to allow pre-prepared structures to be more easily incorporated; and
- a more user friendly method of creating .pddf files, possibly using a 3-D graphical interface, will be needed for complex simulations.

More Detailed Tracking of Individual Aggregates

It is a straightforward task to improve the reporting from the program on the state of individual aggregates. Internally, the nanosimulator tracks each aggregate very closely, and this data could be easily output to files. But the gigabytes of data that would be produced would require some processing to make this a useful exercise. Currently a small sample gives the histories of just a few randomly selected aggregates - a more extensive data set could produce more interesting statistical information on the model, as well as track unusual events which might reveal either errors in the model, or interesting insights into the modelled process.

Nanotechnology

The idea of miniature, nanometre scale machines has been an ideal pursued by researchers for some time. Recent experiments have shown the possibility of manipulating individual atoms, while others have been able to use some of the protein structures mentioned in this thesis, such as actin filaments and microtubules, to move very small objects. In order to design and build such tiny systems, computer simulation similar to that presented in this thesis will be required, that is capable of modelling the laws of physics at the intermediary level between quantum and classical mechanics. This will be especially true with nanomachines operating in liquids, such as the medical devices which are the grail of some researchers in this embryonic field.

Pedagogical Uses

A simulator of this sort can be useful in providing a conceptual framework for students (and researchers!) to understand the motion of proteins in solution, and to provide insight into the types of reactions they undergo. Concepts such as diffusion, Brownian motion, and self-assembly can be demonstrated, and in the future more complex subjects such as the action of motile proteins could be reproduced, e.g. myosin-actin, microtubule-kinesin and microtubule-dynein motors.

Modelling of (Speculative) Biological Information Processing

Some highly speculative, but intriguing, models have been proposed wherein simple “game of life” style computation is done using the regular structure of microtubules¹⁸⁸. Using and extending the state model within the nanosimulator, it should be possible to model such hypothesised activity as state changes within the microtubule lattice. This could be seamlessly combined with the traditional modelling of assembly and disassembly, and interactions between the two processes modelled.

The Future

It is planned to make the program and these results publically available (the raw code is available at <http://www.cs.monash.edu.au/~cbetts/phd/code/index.html>, while the rest of the work will be made available online as soon as possible). It is hoped that the ideas presented in this thesis, combined with the generic object-oriented architecture of the program, will make it possible for others to use and extend it. It would give the author great pleasure if, in ten years time, the program was still in use, although not a single line of the original code remained.

As we attempt to understand the world around us, at scales from the gigantically large to the infinitesimally small, computers are playing a larger and larger role in aiding that understanding.

This thesis is a first step in providing a computerised tool for understanding the intriguing behaviour of biological systems at the level lying between quantum and classical mechanics, the nanoscale world.

Appendix A

Calculating Collision Probabilities

Introduction

A fundamental feature of a simulator modelling moving objects must obviously be how it handles collisions and interactions. And in order to do this, it must first of all determine whether a collision has occurred between a pair of objects.

The normal, and most straightforward way, of handling this problem is to simply simulate the movement of the objects, and calculate mathematically whether they overlap. If they do overlap, a collision has occurred.

Unfortunately, this is not entirely practical at the nanoscale. The difficulty lies in the fact that particles are not moving in straight lines, rather they are following a Brownian motion random-walk that are approximated as a Gaussian probability distribution. The scale of the simulation could be altered to the level where the particles were (effectively) moving in straight lines, but this would involve modelling the objects in picoseconds and picometres, which is unsustainable. Such modelling is already done, and quite widely, but is many orders of magnitude too slow for examining the interactions of large groups of macro-molecules.

The approach taken here is to calculate the odds of the particles colliding, and then to randomly determine whether or not they did collide. Determining the odds of collision turns out to be a challenging task.

Defining the Problem

The problem fundamentally is *"With two spheres of given radius, with given standard deviations of movement, at a certain separation distance, what is the probability that they will collide within a specified time interval?"*.

This can be simplified to some extent. Consider two spheres A and B , with radii R_A and R_B , moving with RMS displacements per cycle of σ_A and σ_B , at a separation distance d apart. Firstly, the standard deviations of movement can be combined¹⁸⁹ to $\sigma = (\sigma_A^2 + \sigma_B^2)^{1/2}$, so that it is possible to treat one sphere as stationary, while the other sphere moves with RMS displacement σ .

Secondly, the radii can be adjusted, so that we can treat the problem as that of a single sphere, with a single standard deviation of movement, "colliding" with a point at the given distance. Since, in order to collide, the centres of the two spheres must approach to within $R_A + R_B$, the problem can be simplified to that of a single sphere, of radius $R = R_A + R_B$, colliding with a moving point (with radius zero).

This reduces the problem to only three variables; the combined radii R , the combined RMS displacements σ , and the separation distance d between the sphere surface and the point. If we divide d and R by σ we can obtain R' and d' , scale-independent measures of radii and distance, measured in terms of the standard deviation of displacement.

The problem can now be stated as *"What is the probability that a sphere of radius R' , will collide with a point at a position d' before time T "?*

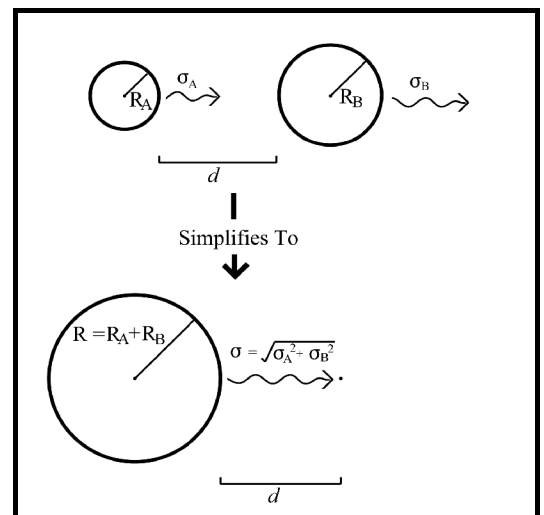


Figure A.1 : The Collision Problem

Analytical solutions

Even with these simplifications though, the problem still remains difficult to solve analytically. To find the collision probability at time T , a number of solutions might be tried:

A naive solution, of integrating the intersection of the Gaussian movement function with the region of (potential) intersection, fails because this would only give the probability of the point being within the bounds of the sphere at the instant $t=T$. The analytical formulation of this is in any case quite difficult to solve.

A more elaborate analytical solution, that of integrating the Gaussian movement function over time $t=0$ to $t=T$, and summing the above formula for the intersection of point and sphere, also fails. This would merely compute what *proportion* of time, on average, the point would spend inside the sphere. Since this formula does not take account of actual collision, but merely the time spent with the target point inside the sphere, this would be a misleading answer.

It is possible that a true analytical solution could be found by integrating the dependent probability tree (by integrating over time t the function describing "given probability(n) that a collision has not occurred prior to time t , what is the probability of a collision in the next instant"). However the approach seemed sufficiently intractable as to merit a numerical approach.

A numerical solution

For the simulator's purposes it is necessary to know a range of sphere sizes and a range of interaction distances. Firstly the standard deviation of movement, and the time scale, were set to unity. This makes it possible to model all situations, since different time periods simply lead to different standard deviations, and all standard deviations can be reduced to unity by simply scaling the sphere size and interaction distance appropriately.)

With Brownian motion, the moving object tends to meander a great deal on its path, frequently backtracking, and jumping from side to side. An example of the Brownian motion of a particle with a step size of standard deviation 0.1 is shown in figure A.2:

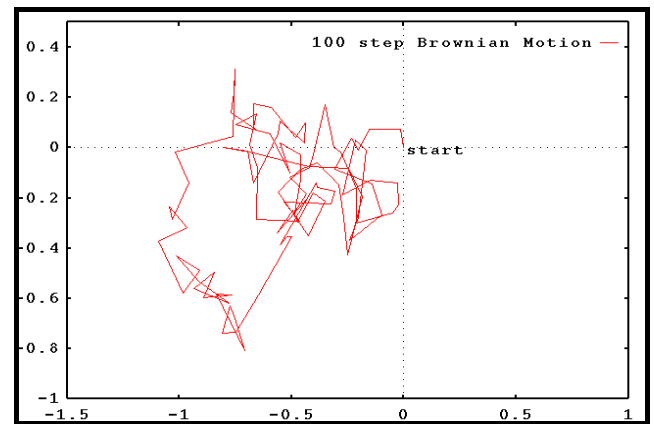


Figure A.2: "Coarse" Brownian motion.

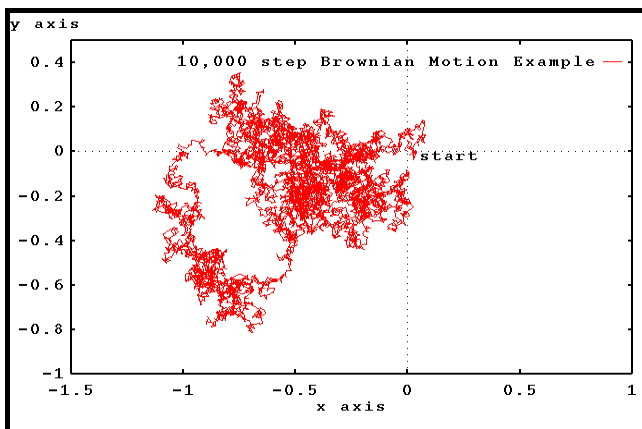


Figure A.3: "Fine" Brownian Motion

One of the difficulties in simulating a system like this is in deciding what level of detail to use. As shown in figure A.2 above, the motion is quite erratic, especially as each of the small, linear paths represented in the image is actually another small random walk. Figure A.3 shows the *same* particle's path, but at a resolution of 10,000 steps, each with a distance of 0.01 standard deviations.

As these graphs show, Brownian motion is fractal in nature - the path becomes more and more convoluted as the scale is reduced. In reality, when considering the trajectories of molecules, the paths really do become almost straight lines at the picometer scale (some curvature may be caused, especially in liquids, due to inter-molecular forces), but this is well below the level of the simulation. In order to make sure this collision simulation program is accurate enough, a step size must be picked that is small compared to the radius of the spheres being tested.

To simulate the problem domain, a series of tests were run. Since the problem was scalable, everything was measured in arbitrary size units. The standard deviation of the test point's movement was set to be 1 unit, which also sets the basic scale of the simulation - everything can now be measured either in "standard units" or equivalently in multiples of "one standard deviation".

The test was run for spheres sized 0.1 to 10 standard units in radius, and these were tested against interaction distances from 0 to 10 standard units in size. (As expected, and as is shown below, there were negligible interactions past about two standard deviations, or two "standard units", in distance.)

In all, 100,000 iterations (i.e. 100,000 different test points and spheres were used to test each radius and distance combination) for 20 different distances and 40 different sphere sizes. To simulate the "random walk" of the particle, rather than moving in one step of standard deviation 1, each sphere was moved in 10,000 steps of standard deviation .01, for the reasons explained above. This step size was chosen so that the resultant standard deviation of .01 units would be substantially smaller than the smallest radius tested, which was 0.1 units in size.

Despite the large number of individual steps (10,000 steps of standard deviation 0.01 units) the eventual standard deviation is the same (1 unit), and the step size is sufficiently smaller than the smallest radius tested, and thus accurately simulates the random walk nature of the particle's movement. The graphical results are presented below, with distance 1 equalling 1 unit (which is also 1 standard deviation), and sphere radius 1 that is likewise 1 unit (or 1 standard deviation) in size:

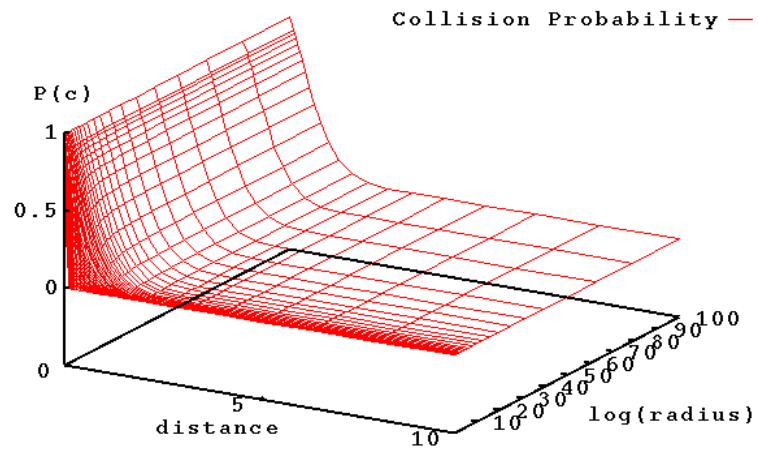


Figure A.4: Collision probability vs linear distance and log sphere radius.

As can be seen in figure A.4, the function is, unsurprisingly, concentrated about large spheres and small distances. Slightly clearer are the same figures, presented in figure A.5 as a graph of probability against the log of distance and sphere size.

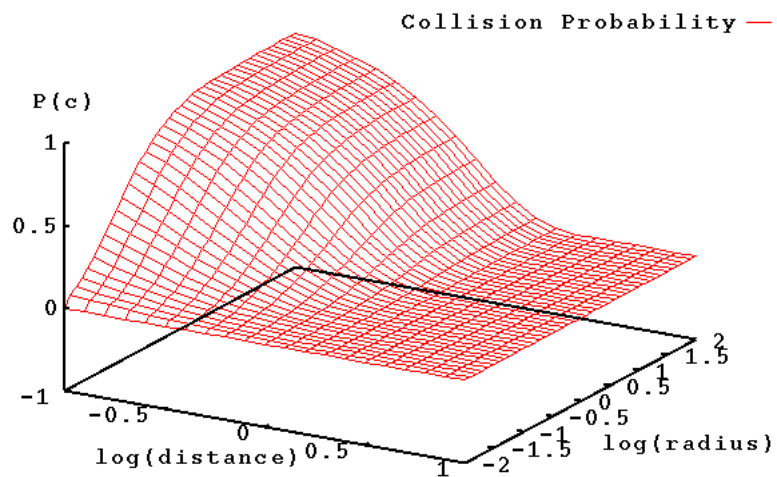


Figure A.5: probability of collision by log of distance and log of sphere size

How these figures are used

These numbers are used in the program whenever two objects come sufficiently close to each other (usually within three movement standard deviations or less) that they might possibly collide within the next time step. The probabilities above, scaled appropriately, are referred to via a lookup table, and a randomiser is used to determine whether a collision did, in fact, occur with the indicated probability.

The use of a lookup table ensures that, although computing the figures initially took some time (on the order of five hours), the speed of the program is unimpaired, since the time for finding the appropriate collision probability in the pre-calculated table is very small, and constant.

Raw Figures

The raw data produced by the collision simulator are presented on the next two pages. The tables give the probability of collision, tabulated against the inter-sphere distances (across the top, measured in combined standard deviations of movement σ), and the combined radii of the spheres, (also measured in standard deviations of movement σ). (See page A.2 for a fuller explanation of the measurements used.)

Because the figures are derived from a probabilistic simulation, there is some inaccuracy. It must be remembered that each figure represents a trial of only 100,000 tests. This can be seen especially in the smaller numbers (i.e. at greater distances, or for smaller radii).

This can also be seen in the limiting case where the point and the sphere start at zero distance. In this case, it is theoretically certain ($P = 1.0$) that the point, moving in a Brownian manner, will collide with the sphere. Due to the discrete nature of the simulation though, this is only approximated in the results (see the “Distance Between Spheres” column for distance = 0 overleaf) which range from 0.447 for the smallest sphere with radius 0.0125σ , to 0.996 for the largest, at 100σ).

This result for the tiny sphere (0.0125σ) is not surprising, since the “step size” of the Brownian motion was only 0.01σ . More accurate figures could be obtained by simply decreasing the step size; however in practice it is unlikely (and undesirable) that simulation objects will be moving, on average, 100 times their size! Conversely, the figures (for this limiting case only) are within 5% of the theoretical limit once the sphere size is within about $1/8$ of the RMS movement σ .

Collision Probabilities 1:

This page covers inter-sphere distances from 0.1 to 1 standard deviations.

Distance Between Spheres

Radii	0	0.1	0.1258	0.1584	0.1995	0.2511	0.3162	0.3981	0.5011	0.6309	0.7943	1
0.01												
0.0125	0.4468	0.0663	0.0547	0.0438	0.0371	0.0287	0.0222	0.0158	0.0112	0.0071	0.0039	0.0022
0.0158	0.5580	0.0950	0.0781	0.0641	0.0532	0.0419	0.0320	0.0229	0.0158	0.0104	0.0058	0.0032
0.0199	0.6496	0.1283	0.1069	0.0881	0.0728	0.0572	0.0445	0.0329	0.0222	0.0146	0.0081	0.0044
0.0251	0.7247	0.1673	0.1396	0.1167	0.0962	0.0768	0.0585	0.0443	0.0296	0.0193	0.0108	0.0059
0.0316	0.7830	0.2118	0.1779	0.1495	0.1238	0.0990	0.0755	0.0578	0.0386	0.0251	0.0143	0.0077
0.0398	0.8280	0.2612	0.222	0.1883	0.1555	0.1252	0.0959	0.0731	0.0497	0.0322	0.0186	0.0098
0.0501	0.8650	0.3153	0.2723	0.2310	0.1922	0.1552	0.1201	0.091	0.0624	0.0407	0.0236	0.0126
0.0630	0.8931	0.3744	0.3249	0.2782	0.2327	0.1892	0.1478	0.1116	0.0776	0.0504	0.0297	0.0155
0.0794	0.9157	0.4341	0.3809	0.3282	0.2766	0.2277	0.1790	0.1353	0.0951	0.0624	0.0369	0.0193
0.1	0.9338	0.4935	0.4385	0.3809	0.3243	0.2682	0.2133	0.1612	0.1144	0.0754	0.0456	0.0239
0.1258	0.9476	0.5513	0.4944	0.4342	0.3729	0.3115	0.2489	0.1899	0.1362	0.0903	0.0550	0.0292
0.1584	0.9587	0.6042	0.5478	0.4871	0.4209	0.3549	0.2867	0.2202	0.1602	0.1072	0.0660	0.0350
0.1995	0.9672	0.6538	0.5980	0.5372	0.4698	0.3994	0.3272	0.2545	0.1856	0.1261	0.0774	0.0414
0.2511	0.9740	0.6969	0.6445	0.5839	0.5178	0.4440	0.3670	0.2893	0.2140	0.1469	0.0907	0.0486
0.3162	0.9794	0.7357	0.6863	0.6274	0.5613	0.4870	0.4081	0.3251	0.2439	0.1695	0.1057	0.0575
0.3981	0.9829	0.7692	0.7231	0.6681	0.6034	0.5297	0.4483	0.3617	0.2755	0.1940	0.1226	0.0669
0.5011	0.9855	0.7977	0.7555	0.7032	0.6404	0.5689	0.4875	0.3979	0.3069	0.2196	0.1402	0.0777
0.6309	0.9878	0.8219	0.7831	0.7345	0.6749	0.6055	0.5250	0.4339	0.3380	0.2458	0.1599	0.0896
0.7943	0.9895	0.8416	0.8054	0.7604	0.7041	0.6374	0.5582	0.4664	0.3700	0.2718	0.1800	0.1023
1	0.9909	0.8583	0.8246	0.7825	0.7310	0.6658	0.5890	0.4976	0.3994	0.2982	0.2003	0.1160
1.2589	0.9919	0.8719	0.8406	0.8008	0.7515	0.6898	0.6154	0.526	0.4268	0.3233	0.2201	0.1297
1.5848	0.9928	0.8827	0.8537	0.8159	0.7695	0.7096	0.6380	0.5506	0.4514	0.3460	0.2395	0.1440
1.9952	0.9933	0.8914	0.8640	0.8287	0.7840	0.7275	0.6575	0.5715	0.4734	0.3666	0.2575	0.1570
2.5118	0.9938	0.8985	0.8724	0.8389	0.7964	0.7421	0.6739	0.5892	0.4913	0.3852	0.2732	0.1694
3.1622	0.9942	0.9040	0.8789	0.8471	0.8059	0.7535	0.6875	0.6039	0.5069	0.4009	0.2882	0.1805
3.9810	0.9945	0.9085	0.8847	0.8537	0.8131	0.7625	0.6983	0.6164	0.5203	0.4141	0.3003	0.1906
5.0118	0.9948	0.9123	0.8892	0.8592	0.8196	0.7702	0.7072	0.6263	0.5314	0.4251	0.3106	0.1992
6.3095	0.9950	0.9150	0.8927	0.8638	0.8248	0.7762	0.7143	0.6349	0.5404	0.4339	0.3193	0.2066
7.9432	0.9951	0.9173	0.8954	0.8670	0.829	0.7811	0.7198	0.6417	0.5477	0.4412	0.3267	0.2128
10	0.9953	0.9192	0.8979	0.8700	0.8322	0.7848	0.7243	0.6473	0.5540	0.4477	0.3328	0.2177
12.589	0.9954	0.9208	0.8996	0.8721	0.8350	0.7880	0.7280	0.6518	0.5590	0.4527	0.3372	0.2222
15.848	0.9955	0.9218	0.9007	0.8738	0.8373	0.7906	0.7310	0.6554	0.5628	0.4567	0.3412	0.2258
19.952	0.9956	0.9227	0.9019	0.8753	0.8389	0.7927	0.7333	0.6580	0.5658	0.4600	0.3443	0.2287
25.118	0.9956	0.9233	0.9028	0.8765	0.8404	0.7943	0.7351	0.6605	0.5681	0.4627	0.3466	0.2308
31.622	0.9957	0.9240	0.9035	0.8774	0.8415	0.7958	0.7368	0.6623	0.5702	0.4647	0.3489	0.2325
39.810	0.9957	0.9244	0.9041	0.8779	0.8424	0.7967	0.7383	0.6637	0.5719	0.4663	0.3505	0.2338
50.118	0.9957	0.9247	0.9046	0.8783	0.8429	0.7977	0.7393	0.6650	0.5731	0.4675	0.3517	0.2350
63.095	0.9957	0.9250	0.9049	0.8787	0.8434	0.7983	0.7400	0.6659	0.5739	0.4687	0.3526	0.2359
79.432	0.9957	0.9252	0.9052	0.8790	0.8438	0.7988	0.7407	0.6665	0.5749	0.4697	0.3535	0.2367
100	0.9957	0.9254	0.9054	0.8793	0.8440	0.7993	0.7412	0.6671	0.5755	0.4704	0.3541	0.2373

Collision Probabilities 2:

This page covers inter-sphere distances from 1 to 10 standard deviations. (The ‘1’ column is repeated for clarity):

Distance Between Spheres

Radii	1	1.2589	1.5848	1.9952	2.51189	3.16228	3.98	5.01	6.30	7.94	10
0.01											
0.0125	0.0022	0.0009	0.0003	0	0	0	0	0	0	0	0
0.0158	0.0032	0.0012	0.0004	0	0	0	0	0	0	0	0
0.0199	0.0044	0.0017	0.0006	0.0001	1.00E-05	0	0	0	0	0	0
0.0251	0.0059	0.0023	0.0008	0.0001	1.00E-05	0	0	0	0	0	0
0.0316	0.0077	0.0031	0.0011	0.0002	2.00E-05	0	0	0	0	0	0
0.0398	0.0098	0.0040	0.0014	0.0002	2.00E-05	0	0	0	0	0	0
0.0501	0.0126	0.0051	0.0018	0.0003	2.00E-05	0	0	0	0	0	0
0.0630	0.0155	0.0064	0.0022	0.0003	2.00E-05	0	0	0	0	0	0
0.0794	0.0193	0.0080	0.0026	0.0005	2.00E-05	0	0	0	0	0	0
0.1	0.0239	0.0099	0.0032	0.0006	4.00E-05	0	0	0	0	0	0
0.1258	0.0292	0.0121	0.0039	0.0007	7.00E-05	0	0	0	0	0	0
0.1584	0.0350	0.0148	0.0047	0.0009	0.0001	0	0	0	0	0	0
0.1995	0.0414	0.018	0.0057	0.0010	0.00011	1.00E-05	0	0	0	0	0
0.2511	0.0486	0.0216	0.0066	0.0012	0.00013	1.00E-05	0	0	0	0	0
0.3162	0.0575	0.0257	0.0078	0.0015	0.00016	1.00E-05	0	0	0	0	0
0.3981	0.0669	0.0299	0.0095	0.0017	0.0002	1.00E-05	0	0	0	0	0
0.5011	0.0777	0.0350	0.0112	0.0021	0.00023	1.00E-05	0	0	0	0	0
0.6309	0.0896	0.0412	0.0135	0.0026	0.00027	1.00E-05	0	0	0	0	0
0.7943	0.1023	0.0477	0.0161	0.0031	0.00034	1.00E-05	0	0	0	0	0
1	0.1160	0.0547	0.0189	0.0038	0.00046	2.00E-05	0	0	0	0	0
1.2589	0.1297	0.0618	0.0220	0.0044	0.00052	2.00E-05	0	0	0	0	0
1.5848	0.1440	0.0695	0.0249	0.0052	0.00062	2.00E-05	0	0	0	0	0
1.9952	0.1570	0.0777	0.0283	0.0060	0.00074	2.00E-05	0	0	0	0	0
2.5118	0.1694	0.0850	0.0317	0.0069	0.00085	2.00E-05	0	0	0	0	0
3.1622	0.1805	0.0922	0.0352	0.0078	0.00095	2.00E-05	0	0	0	0	0
3.9810	0.1906	0.0985	0.0385	0.0087	0.00102	2.00E-05	0	0	0	0	0
5.0118	0.1992	0.1040	0.0412	0.0095	0.00113	2.00E-05	0	0	0	0	0
6.3095	0.2066	0.1088	0.0439	0.0105	0.00121	4.00E-05	0	0	0	0	0
7.9432	0.2128	0.1131	0.0462	0.0113	0.00128	5.00E-05	0	0	0	0	0
10	0.2177	0.1171	0.0482	0.0119	0.00141	5.00E-05	0	0	0	0	0
12.589	0.2222	0.1197	0.0497	0.0125	0.00147	7.00E-05	0	0	0	0	0
15.848	0.2258	0.1222	0.0510	0.0131	0.00159	7.00E-05	0	0	0	0	0
19.952	0.2287	0.1243	0.0520	0.0135	0.00164	8.00E-05	0	0	0	0	0
25.118	0.2308	0.1260	0.0530	0.0139	0.0017	8.00E-05	0	0	0	0	0
31.622	0.2325	0.1273	0.0537	0.0142	0.00182	8.00E-05	0	0	0	0	0
39.810	0.2338	0.1284	0.0543	0.0145	0.00187	8.00E-05	0	0	0	0	0
50.118	0.2350	0.1291	0.055	0.0147	0.00189	8.00E-05	0	0	0	0	0
63.095	0.2359	0.1299	0.0554	0.0149	0.00191	8.00E-05	0	0	0	0	0
79.432	0.2367	0.1304	0.0557	0.0151	0.00194	8.00E-05	0	0	0	0	0
100	0.2373	0.1309	0.0560	0.0152	0.00194	8.00E-05	0	0	0	0	0

Conclusion

While an analytic solution would be desirable, in its absence the probabilities of collision of objects have been obtained by simply simulating a sphere and a randomly moving test point many thousands of times.

The results of this simulation have been incorporated into a data file which the modelling program reads when it starts running, allowing it to quickly determine the probability of collision for any pair of objects being simulated.

Appendix B

Nanosimulator Operation

Introduction

This Appendix describes how to run the nanosimulator program and the command line parameters it supports.

The program requires a .pddf file (see Appendix C) to run. (If none is specified, it will attempt to use the file ‘default.pddf’.) The .pddf file specifies all the information the program requires in order to simulate the virtual environment. A number of other parameters may be passed to the program in order to specify how the program outputs data; does it log data, echo reports to the console, save raytracer files, display realtime graphical views of the simulation, and so on.

Program Invocation

The program is run from the console using “nanosim [options]” - e.g.

```
>nanosim -h
```

(This brings up some simple online help)

```
>nanosim -Dactin.pddf -T100000 -d1000
```

(This starts the program using the ‘actin.pddf’ file, and runs the program for 100,000 cycles, saving log files every 1000 cycles.)

Command Line Parameters

The program can be given a number of optional command line parameters, which primarily affect the display and logging behaviour of the program, but can also set frequently changed program variables, such as the molarity of the solution being simulated, and the size of the simulation field. There should not be a space between a command line parameter and its argument (if there is an argument) - the argument should follow directly..

Parameters Affecting Graphical Display

A number of parameters are concerned only with how the graphical display window of the program appears, and how quickly it is updated:

- c[Integer] - cycles per update: specifies the number of computation cycles between visual updates.
e.g. -c100 (update the graphics window every 100 program cycles)
- G - no Graphics: Suppress graphical output altogether. This can be used when the program is being run remotely, i.e. via telnet.
- r[Integer] - rgb files: regularly save SGI 'rgb' format graphics files every 'integer' cycles. (The saving is done using a system-level screen capture, so the graphics window must be visible for this to work.)
- x[Integer] - this sets the width of the graphics window, which otherwise defaults to 512.
- y[Integer] - this sets the height of the graphics window, which otherwise defaults to 512.
e.g. -x1024 -y768 sets the graphics window to be 1024 by 768.

Parameters Affecting Logging and Reporting

These parameters are concerned with how frequently text data is output by the program, and how extensive that data is. The program logs data to a number of text files ('tubdata1.log' through to 'tubdata5.log', and 'tuberror.log'), as well as echoing information to the console and optionally writing raytracer files (of the form 'tubNNNNN.pov', e.g. 'tub00412.pov').

- d[Integer] - data logs: This sets the frequency at which the data logs are saved (default = 200 cycles).
e.g. -d1000 would save new information to the data logs every 1000 cycles
- p[Integer] - povray raytracer files: This sets the frequency for saving 'povray' (cf <http://povray.org>) raytracer files (default = no files saved). The files are named 'tubNNNNN.pov', where 'NNNNN' is a unique, consecutive 5 digit number starting from '00000'.
e.g. -d500 saves a new povray file every 500 cycles.
- s - silent: this suppresses all *console* text output (except errors)
- V - Verbose: produces detailed reporting used in debugging.

Parameters Affecting the Simulation Environment

Most parameters that affect the simulation environment are set in the .pddf file. However some parameters may be changed frequently, possibly every time the program is run. This includes running the program in a special mode, changing the modelling method, and setting how large a volume the program will simulate, with how many particles.

- B - Breakage: Changes the model to specify no breakage of multiply linked objects.

-D[File Name] - Data file: gives the name of the .pddf file to use (default is 'info.pddf')

e.g. -Dvirus_models/influenza57.pddf

(specifies use of the 'influenza57.pddf' file in the 'virus_models' directory)

-F - Freeze aggregate movement: this cause collections of objects to be stationary, reducing computation time at the expense of realism. (default = no freezing)

-f[Integer] -> field size: this sets the width of the simulation cube in nanometres. For reasons of internal program efficiency, this volume must be a power of 2 greater than 32 (i.e. 32,64,128,256,512,1024 ...). (the default = 256)
e.g. -f1024 creates a simulation volume just over 1 micrometer cubed.

-m[Integer] - monomer used: this sets the total number of discrete monomer objects to use. (If multiple object species are present, they will *add* to this number.) The program reports on the equivalent molarity of the solution. (default = 500)
e.g. -m800 (use 800 objects in the simulation).

-M[Integer] - Molarity: this sets the molarity in micromoles. The program reports on the total number of objects this is equivalent to. This is another way of specifying the same information as the '-m' parameter above. It represents the aggregate molarity of all the object species present in the solution. (default = ~6 μM)
e.g. -M2.5 (sets the molarity of the solution to 2.5 micromole)

-NM - No Merging: this stops the program merging aggregates together. Merging aggregates can be computationally intensive, and is not always reliable.
(default = merging).

-R - no Rotation: This turns aggregate rotation off. Rotation *probably* doesn't play a large role in polymerisation, and can be computationally intensive.
(default = rotation)

General Parameters

Some parameters simply set general program behaviour.

- h/-H - help: displays a short help message.
- Hgraphics - Help for graphics: displays help on the graphics display window, and how to interact with it.
- C[+/-] - Collision test: tests the number of collisions occurring for a single non-binding object. This is used to work out the number of collisions/second that occur at a particular time scale. the argument '+' indicates the test object is motile, while '-' indicates the test object is stationary. Usually the stationary test is used, to work out the number of collisions experienced by a particle on a growth tip of a large aggregate.
 - C+ (test the number of collisions undergone by a moving object)
 - C- (test the number of collisions undergone by a stationary object).
- P - Parser test: parse the .pddf file, but don't run the program. Useful for testing a .pddf file that is being created.
- T[Integer] - Timed run: only run the program for a set number of iterations.
e.g. -T100000 (run the program for 100,000 cycles)
- v - version number: prints the version number of the program.

Graphics Display Window

The graphics display window shows a realtime view of the simulator in action. It also allows a limited amount of interaction with the user via the mouse and a set of hotkeys. Using the mouse the user can change their viewpoint by “rotating” the simulation space (The user’s viewpoint remains a fixed distance from the centre, as if the simulation were within a globe, and the user was rotating that globe).

The user can also use the following keys to alter the way the program displays its output.

- R - Refresh the view: This causes the program to refresh the graphical output immediately, without waiting for the next update.
- B - toggle Bound objects off/on: This toggles the program between only showing aggregates, and showing both aggregates and free objects.
- P - Pauses the simulation: This freezes the simulation, allowing the user to rotate and examine the field without it being updated. (The program does not continue the simulation until the ‘P’ key is pressed again.)
- S - Save the current image: This saves the current window as an Silicon Graphics ‘rgb’ file.
- V - save as povray: This saves the current image as a povray raytracer file.
- L - show Links: This shows the linkage sites on objects as little spheres.

Finally. the ‘Q’ key can be used to end the simulation run.

- Q - Quit the entire program.

Appendix C

Protein Dynamic Description Files

Introduction

This Appendix describes the simple object description language used in the “Protein Dynamic Description Files”, or “.pddf files” to designate the important properties of the monomers used by the nanoscale simulation program. This is the central file used by the program to specify all the user-set details of the simulation (aside from a small number of command line options that mainly specify details of reporting and logging - see Appendix E).

These .pddf files must specify the:

- geometry of the objects
- location of binding sites
- strengths and attractiveness of these binding sites
- states of the objects
- state changes the objects undergo, and their causes
- colours of the objects in their various states
- environmental conditions (viscosity, temperature etc.) of the simulation
- timestep of the simulation
- mix of molecules in the simulation

.pddf File Syntax

The .pddf file is a series of definitions describing the structure and behaviour of the objects the nanosimulator will process and simulate.

Basic Data Types: floats, strings, points and colours

Floats and Strings

The .pddf files use a number of basic data types for atomic information. The most common is the floating point number (e.g., ignoring quotes; '48', '0.4' or '9000.992'). Strings are frequently used to name objects, and simply appear as unadorned text (e.g., ignoring quotes; 'frog', 'pond_47'). In addition, two other special data types are used, *points* and *colours*.

Points

Points represent a position in three dimensional Cartesian space. They are expressed as a triplet comprised of an x, y and a z co-ordinate value (each a floating point number) separated by angle brackets:

point:

<x co-ord, y co-ord, z co-ord>

e.g. *<14, -2.6, 0>*

No spaces should appear between the start and end angle bracket in a point definition.

Colours

Colours are used for display purposes only. They are useful to show the different molecular species in a simulation, and the different states of the objects. Colours are specified by *either* a small number of keyword strings, or a hexadecimal RGB triplet expressing a 24 bit colour.

colour:

red, green, blue, gray, grey, black, white, lightRed, lightGreen, lightBlue, yellow, purple, cyan

or

OxRRGGBB

(Where RRGGBB are hexadecimal digits, the 'RR' digits specifying the red value, the 'GG' triplet specifying the green value, and the 'BB' triplet specifying the blue value)

e.g. 0xFF0000 is the same as red.

File Structure

A .pddf file has two main parts; a model definition, followed by linkage definitions. The model definition must come first.

.pddf file:

model definitions

linkage definitions

Model Definition

The model definition must describe each molecular species in terms of its geometry, ‘colour’ (used for display purposes only), binding site geometry, states, and list the events that cause state changes. The order this is done in *is* important, as it is not possible to use entities before they have been defined (i.e. a component sphere cannot be mentioned in a state definition before it has been defined).

model definition:

```
{  
    geometrical structure  
    linkage sites  
    states  
    events  
}
```

Geometrical Structure Definition

The geometrical structure of the object is defined solely as a set of component spheres. This usually involves approximating the structure of the object to some extent; for example a rod-like protein must be represented as a string of spheres.

structure definition:

<list of component spheres>

component sphere:

sphere name radius centre colour

sphere - a key word specifying the start of a component sphere declaration. In the future, other keywords may be added for other geometries.

name - a human readable unique name that is used within the .pddf file, and for logging info.

radius - a floating point number specifying the radius of the sphere in nanometers

centre - a point specifying the position of the sphere centre

colour - the initial colour of this sphere in the starting state

e.g.

```
sphere Molecule_Centre 0.9 <0,0,0> red
```

Linkage Site Definition

A linkage site is an approximation of molecular binding, that reduces the link between two molecules to a single point of contact. The linkage site definition must specify the position of the site, as well as the direction of the site and optionally a “twist” vector for the site, which must be at right angles to the direction. When another molecule links to this site with its own linkage, the positions of the two linkages overlap, the direction vectors are set to be exactly *opposite* to each other, and the ‘twist’ vectors if present, are made to *overlap* exactly. (The rationale being that the two links should be at the same position, pointing in opposite directions, with the ‘twist’ vector acting to set a common third direction (i.e. ‘up’) to orient both molecules.)

linkage definition:

site name position [direction [twist]]

<i>site</i>	- a keyword specifying the start of a linkage site specification
<i>name</i>	- a human readable unique name for the site, used by the .pddf file and for logging
<i>position</i>	- a point giving the position of the linkage site
<i>direction</i>	- a point optionally giving the direction of the linkage site. If not specified, the direction defaults to the position vector, normalised (i.e. straight out from the centre).
<i>twist</i>	- a point optionally giving a twist vector to the direction vector (which must be specified if a twist is specified). If not specified the binding orientation of the molecules is not strongly defined by the program; it will be regular according to an internal rule of the program that is not necessarily obvious to the user.

e.g.

site antibody_linkage_site <1,0,0> <1,0,0> <0,1,0>

(This sets a site called ‘antibody_linkage_site’ at position <1,0,0>, also in direction <1,0,0>, and specifies that any molecule linking to the site must rotate to align twist vectors to <0,1,0>.)

State Definition

Often polymerising proteins undergo a state change when certain links are made or broken. To reflect this, the program implements a simple state machine. In the linkage definitions below (not to be confused with the linkage site section above) it is possible to define different binding strengths for different states of the object. It is also possible to display different states in different colours for the edification of viewers. Often states are given names appropriate to the event that created them; e.g. 'bound' or 'unbound'.

state definition:

state name [colour]

state - a keyword specifying a state definition

name - a human readable name used in the .pddf file and for logging

colour - either a hex format rgb triplet in the format 0xRRGGBB, or one of the following predefined colours {red, green, blue, yellow, magenta, cyan, pink, lightgreen, lightblue, lightred, white, lightgray, gray, darkgray, black,}

e.g.

```
state bound 0xFF0000
```

or

```
state energized red
```

(Note that the colour is the same in both instances.)

Event Definition

An event is an program occurrence that causes an object to change state. The most usual occurrence is a binding or breaking of a particular link, but it is possible to define random state changes as well (e.g. an object in an unphosphorylated state might have a chance of randomly rephosphorylating in the presence of free ATP).

event definition;

```
event bind {[linkA, linkB]} state-name1 -> state-name2
```

or

```
event break {[link]} state-name1 -> state-name2
```

or

```
event random probability state-name1 -> state-name2
```

event	- key word specifying an event definition
bind	- key work specifying a binding event definition
break	- key word specifying a breakage event definition
random	- key work specifying a random event definition
[linkA,linkB]	- optional parameters specifying particular links that must bind
[link]	- an optional parameter specifying a particular link that must break
probability	- the probability (per cycle) of the event occurring.
state	- the state from which (first argument), or to which (second argument), the
object	changes to when this event occurs
->	- key symbol

e.g.

```
event bind { } unbound -> bound
event break { first_link } bound -> unbound
random 0.1 adp -> atp
```

Linkage Definitions

Once the various object models and their linkage sites have been defined, it is necessary to specify the binding and breaking probabilities of each linkage site with each other linkage site that it can interact with (unspecified linkage combinations are set to zero).

linkage definition:

binding *object site state* to *object site state* = *bindprob breakprob*

binding	- keyword for linkage definition
object	- the object being linked
site	- the particular site being linked (may include keyword 'all' - see note below)
state	- the state that this link applies to (may include keyword 'all' - see note below)
to	- keyword
=	- key symbol
bindprob	- a floating point number giving the probability of the two links binding (if they touch during a collision). Optionally this may be enclosed by the keywords "bind(...)" for clarity.
breakprob	- a floating point number giving the probability (per cycle) of the link breaking once formed Optionally this may be enclosed by the keywords "break(...)" for clarity

e.g.

```
binding virus siteA unbound to antibody activesite unbound = 0.001 0.0045
binding actin top bound to actin bottom unbound = bind(0.02) break(0.0001)
```

An Example .pddf File

Combining all the elements above into a single file is not quite as onerous as might be imagined. The following is a simple, if verbose, example of a .pddf file that defines the structure of an (imaginary) flat triangular structure. Notice how the structure and links effectively define (in this example) a two-dimensional structure.

```
# spheres in a graphite-like flat triangular lattice structure

Model Triangle
{
    Sphere a 0.9 <0,0,0> red

    # site    name    <position> <direction> twist
    #
    site Aside <1,0,0> <1,0,0>
    site Bside <-0.5,0.8660254,0> <-0.5,0.8660254,0>
    site Cside <-0.5,-0.8660254,0> <-0.5,-0.8660254,0>

    State unbound

    State bound      # a new state, with no physical changes

    colour a = blue

    # The event list is all of the form:
    # Event: link { A B } | break | random (chance / cycle)
    #           StateA -> StateB

    Event bind { } unbound -> bound
    Event break { all }                               # if the top link is broken,
                                                        # object becomes "unbound" with
    bound -> unbound                                   # no change in phosphorylation
}

# specify links and states, and give their binding and breaking chance
```

```

# links are transitive.

# unbound links

Binding Triangle Aside unbound to Triangle Aside unbound = 0.001 0
Binding Triangle Aside unbound to Triangle Bside unbound = 0.001 0
Binding Triangle Aside unbound to Triangle Cside unbound = 0.001 0

Binding Triangle Bside unbound to Triangle Aside unbound = 0.001 0
Binding Triangle Bside unbound to Triangle Bside unbound = 0.001 0
Binding Triangle Bside unbound to Triangle Cside unbound = 0.001 0

Binding Triangle Cside unbound to Triangle Aside unbound = 0.001 0
Binding Triangle Cside unbound to Triangle Bside unbound = 0.001 0
Binding Triangle Cside unbound to Triangle Cside unbound = 0.001 0

# half bound links

Binding Triangle Aside bound to Triangle Aside unbound = 0.9 0
Binding Triangle Aside bound to Triangle Bside unbound = 0.9 0
Binding Triangle Aside bound to Triangle Cside unbound = 0.9 0

Binding Triangle Bside bound to Triangle Aside unbound = 0.9 0
Binding Triangle Bside bound to Triangle Bside unbound = 0.9 0
Binding Triangle Bside bound to Triangle Cside unbound = 0.9 0

Binding Triangle Cside bound to Triangle Aside unbound = 0.9 0
Binding Triangle Cside bound to Triangle Bside unbound = 0.9 0
Binding Triangle Cside bound to Triangle Cside unbound = 0.9 0

# unbound links

Binding Triangle Aside unbound to Triangle Aside bound = 0.9 0
Binding Triangle Aside unbound to Triangle Bside bound = 0.9 0
Binding Triangle Aside unbound to Triangle Cside bound = 0.9 0

Binding Triangle Bside unbound to Triangle Aside bound = 0.9 0
Binding Triangle Bside unbound to Triangle Bside bound = 0.9 0
Binding Triangle Bside unbound to Triangle Cside bound = 0.9 0

```

```
Binding Triangle Cside unbound to Triangle Aside bound = 0.9 0
Binding Triangle Cside unbound to Triangle Bside bound = 0.9 0
Binding Triangle Cside unbound to Triangle Cside bound = 0.9 0
```

```
#bound links
```

```
Binding Triangle Aside bound to Triangle Aside bound = 0.9 0
Binding Triangle Aside bound to Triangle Bside bound = 0.9 0
Binding Triangle Aside bound to Triangle Cside bound = 0.9 0
```

```
Binding Triangle Bside bound to Triangle Aside bound = 0.9 0
Binding Triangle Bside bound to Triangle Bside bound = 0.9 0
Binding Triangle Bside bound to Triangle Cside bound = 0.9 0
```

```
Binding Triangle Cside bound to Triangle Aside bound = 0.9 0
Binding Triangle Cside bound to Triangle Bside bound = 0.9 0
Binding Triangle Cside bound to Triangle Cside bound = 0.9 0
```

```
# some environment constants
```

```
Temperature 310 # kelvin - physiological temperature
Viscosity .69 # approximate viscosity for water at 37 degrees Celsius
TimeScale 100 # 100 nanoseconds per program cycle
# Finally, specify the mix of molecules in solution.
```

```
Mix Triangle 100%
```

The Future?

The above syntax is somewhat verbose, as every link in every state must be explicitly specified. The program is moving to support the keyword ‘all’ as a valid linkage state and linkage site. At the time of writing this is only partially implemented (for sites, but not for states). At the moment, the above file should be able to be rewritten as:

```
-----  
----  
Model Triangle  
{  
    Sphere a 0.9 <0,0,0> red  
  
    site Aside <1,0,0> <1,0,0>  
    site Bside <-0.5,0.8660254,0> <-0.5,0.8660254,0>  
    site Cside <-0.5,-0.8660254,0> <-0.5,-0.8660254,0>  
  
    State unbound  
  
    State bound colour a = blue  
  
    Event bind { } unbound -> bound  
    Event break { } bound -> unbound  
}  
  
# unbound links  
  
Binding Triangle all unbound to Triangle all unbound = 0.001 0  
  
# half bound links  
  
Binding Triangle all bound to Triangle all unbound = 0.9 0  
  
# unbound links  
  
Binding Triangle all unbound to Triangle all bound = 0.9 0
```

```
#bound links
```

```
Binding Triangle all bound to Triangle all bound = 0.9 0
```

```
# some environment constants
```

```
Temperature 310 # kelvin - physiological temperature
```

```
Viscosity .69 # approximate viscosity for water at 37 degrees Celsius
```

```
TimeScale 100 # 100 nanoseconds per program cycle
```

```
# Finally, specify the mix of molecules in solution.
```

```
Mix Triangle 100%
```

```
-----
```

```
----
```

... Which will obviously be somewhat more manageable. This can only be used however for molecules with a number of identical links that can thus be treated as a group, and is thus not suitable for all cases.

Appendix D

Platonic Solids: .pddf files

Cube, Dodecahedron and Icosahedron

Cube

```
# Another example pddf ... this time a "protein" that forms cubes
#
# again, this is not meant to be an accurate simulation, but
# is a demonstration file.
```

```
Model Cubid
{
    Sphere a 1.414 <0,0,0> red

    #site    name      <position> <direction> twist

    site Aside <0,1,1> <0,1,1> <0,-1,1>
    site Bside <0,-1,1> <0,-1,1> <0,1,1>
    site Cside <1,0,1> <1,0,1> <-1,0,1>
    site Dside <-1,0,1> <-1,0,1> <1,0,1>

    State unbound

    State bound

    colour a = green

    Event bind { Aside any }
        unbound -> bound
    Event bind { Bside any }
```

```

        unbound -> bound
Event  bind { Cside any }
        unbound -> bound
Event  bind { Dside any }
        unbound -> bound

Event  break { all }

        bound -> unbound
}

Binding Cubid Aside unbound  to Cubid Aside unbound = 0.01 0
Binding Cubid Aside unbound  to Cubid Bside unbound = 0.01 0
Binding Cubid Aside unbound  to Cubid Cside unbound = 0.01 0
Binding Cubid Aside unbound  to Cubid Dside unbound = 0.01 0
Binding Cubid Bside unbound  to Cubid Aside unbound = 0.01 0
Binding Cubid Bside unbound  to Cubid Bside unbound = 0.01 0
Binding Cubid Bside unbound  to Cubid Cside unbound = 0.01 0
Binding Cubid Bside unbound  to Cubid Dside unbound = 0.01 0
Binding Cubid Cside unbound  to Cubid Aside unbound = 0.01 0
Binding Cubid Cside unbound  to Cubid Bside unbound = 0.01 0
Binding Cubid Cside unbound  to Cubid Cside unbound = 0.01 0
Binding Cubid Cside unbound  to Cubid Dside unbound = 0.01 0
Binding Cubid Dside unbound  to Cubid Aside unbound = 0.01 0
Binding Cubid Dside unbound  to Cubid Bside unbound = 0.01 0
Binding Cubid Dside unbound  to Cubid Cside unbound = 0.01 0
Binding Cubid Dside unbound  to Cubid Dside unbound = 0.01 0

Binding Cubid Aside bound    to Cubid Aside unbound = 0.9 0
Binding Cubid Aside bound    to Cubid Bside unbound = 0.9 0
Binding Cubid Aside bound    to Cubid Cside unbound = 0.9 0
Binding Cubid Aside bound    to Cubid Dside unbound = 0.9 0
Binding Cubid Bside bound    to Cubid Aside unbound = 0.9 0
Binding Cubid Bside bound    to Cubid Bside unbound = 0.9 0
Binding Cubid Bside bound    to Cubid Cside unbound = 0.9 0
Binding Cubid Bside bound    to Cubid Dside unbound = 0.9 0
Binding Cubid Cside bound    to Cubid Aside unbound = 0.9 0
Binding Cubid Cside bound    to Cubid Bside unbound = 0.9 0
Binding Cubid Cside bound    to Cubid Cside unbound = 0.9 0
Binding Cubid Cside bound    to Cubid Dside unbound = 0.9 0
Binding Cubid Dside bound    to Cubid Aside unbound = 0.9 0
Binding Cubid Dside bound    to Cubid Bside unbound = 0.9 0

```

Binding Cubid Dside bound to Cubid Cside unbound = 0.9 0
Binding Cubid Dside bound to Cubid Dside unbound = 0.9 0

Binding Cubid Aside unbound to Cubid Aside bound = 0.9 0
Binding Cubid Aside unbound to Cubid Bside bound = 0.9 0
Binding Cubid Aside unbound to Cubid Cside bound = 0.9 0
Binding Cubid Aside unbound to Cubid Dside bound = 0.9 0
Binding Cubid Bside unbound to Cubid Aside bound = 0.9 0
Binding Cubid Bside unbound to Cubid Bside bound = 0.9 0
Binding Cubid Bside unbound to Cubid Cside bound = 0.9 0
Binding Cubid Bside unbound to Cubid Dside bound = 0.9 0
Binding Cubid Cside unbound to Cubid Aside bound = 0.9 0
Binding Cubid Cside unbound to Cubid Bside bound = 0.9 0
Binding Cubid Cside unbound to Cubid Cside bound = 0.9 0
Binding Cubid Cside unbound to Cubid Dside bound = 0.9 0
Binding Cubid Dside unbound to Cubid Aside bound = 0.9 0
Binding Cubid Dside unbound to Cubid Bside bound = 0.9 0
Binding Cubid Dside unbound to Cubid Cside bound = 0.9 0
Binding Cubid Dside unbound to Cubid Dside bound = 0.9 0

Binding Cubid Aside bound to Cubid Aside bound = 0.9 0
Binding Cubid Aside bound to Cubid Bside bound = 0.9 0
Binding Cubid Aside bound to Cubid Cside bound = 0.9 0
Binding Cubid Aside bound to Cubid Dside bound = 0.9 0
Binding Cubid Bside bound to Cubid Aside bound = 0.9 0
Binding Cubid Bside bound to Cubid Bside bound = 0.9 0
Binding Cubid Bside bound to Cubid Cside bound = 0.9 0
Binding Cubid Bside bound to Cubid Dside bound = 0.9 0
Binding Cubid Cside bound to Cubid Aside bound = 0.9 0
Binding Cubid Cside bound to Cubid Bside bound = 0.9 0
Binding Cubid Cside bound to Cubid Cside bound = 0.9 0
Binding Cubid Cside bound to Cubid Dside bound = 0.9 0
Binding Cubid Dside bound to Cubid Aside bound = 0.9 0
Binding Cubid Dside bound to Cubid Bside bound = 0.9 0
Binding Cubid Dside bound to Cubid Cside bound = 0.9 0
Binding Cubid Dside bound to Cubid Dside bound = 0.9 0

Aside generic bind

#Binding Antibody all to Dimer top all = 1 0

some environment constants

Temperature 310

```
Viscosity    0.69
TimeScale    100
```

```
Mix Cubid 100%
```

Dodecahedron

```
# Another example pddf ... spheres making up the faces of an dodecahedron
#
```

```
Model Dodec
```

```
{
  Sphere a 0.9 <0,0,0> red

  # site name <position> <direction> [<twist>]

  site Aside    <0.85065081,0,-0.5257311>
                <0.85065081,0,-0.5257311>
                <0.5257311,0,0.85065081>

  site Bside    <0.26286556,0.809017,-0.5257311>
                <0.26286556,0.809017,-0.5257311>
                <0.16245985,0.5,0.85065081>

  site Cside    <-0.68819096,0.5,-0.5257311>
                <-0.68819096,0.5,-0.5257311>
                <-0.42532539,0.309017,0.85065081>

  site Dside    <-0.68819096,-0.5,-0.5257311>
                <-0.68819096,-0.5,-0.5257311>
                <-0.42532539,-0.309017,0.85065081>

  site Eside    <0.26286556,-0.809017,-0.5257311>
                <0.26286556,-0.809017,-0.5257311>
                <0.16245985,-0.5,0.85065081>
}
```

```
State unbound
```

```
State bound
```

```
colour a = blue
```

```
Event bind { } unbound -> bound
```

```

Event break { all } bound -> unbound

}

# specify links and states, and give their binding and breaking chance
# links are transitive.
# (nb. for large models, these should be machine generated in some way -
# e.g. scripts or wordprocessor macros )

# unbound links

Binding Dodec Aside unbound to Dodec Aside unbound = 0.01 0
Binding Dodec Aside unbound to Dodec Bside unbound = 0.01 0
Binding Dodec Aside unbound to Dodec Cside unbound = 0.01 0
Binding Dodec Aside unbound to Dodec Dside unbound = 0.01 0
Binding Dodec Aside unbound to Dodec Eside unbound = 0.01 0

Binding Dodec Bside unbound to Dodec Aside unbound = 0.01 0
Binding Dodec Bside unbound to Dodec Bside unbound = 0.01 0
Binding Dodec Bside unbound to Dodec Cside unbound = 0.01 0
Binding Dodec Bside unbound to Dodec Dside unbound = 0.01 0
Binding Dodec Bside unbound to Dodec Eside unbound = 0.01 0

Binding Dodec Cside unbound to Dodec Aside unbound = 0.01 0
Binding Dodec Cside unbound to Dodec Bside unbound = 0.01 0
Binding Dodec Cside unbound to Dodec Cside unbound = 0.01 0
Binding Dodec Cside unbound to Dodec Dside unbound = 0.01 0
Binding Dodec Cside unbound to Dodec Eside unbound = 0.01 0

Binding Dodec Dside unbound to Dodec Aside unbound = 0.01 0
Binding Dodec Dside unbound to Dodec Bside unbound = 0.01 0
Binding Dodec Dside unbound to Dodec Cside unbound = 0.01 0
Binding Dodec Dside unbound to Dodec Dside unbound = 0.01 0
Binding Dodec Dside unbound to Dodec Eside unbound = 0.01 0

Binding Dodec Eside unbound to Dodec Aside unbound = 0.01 0
Binding Dodec Eside unbound to Dodec Bside unbound = 0.01 0
Binding Dodec Eside unbound to Dodec Cside unbound = 0.01 0
Binding Dodec Eside unbound to Dodec Dside unbound = 0.01 0
Binding Dodec Eside unbound to Dodec Eside unbound = 0.01 0

```

half bound links

Binding Dodec Aside bound to Dodec Aside unbound = 0.9 0
Binding Dodec Aside bound to Dodec Bside unbound = 0.9 0
Binding Dodec Aside bound to Dodec Cside unbound = 0.9 0
Binding Dodec Aside bound to Dodec Dside unbound = 0.9 0
Binding Dodec Aside bound to Dodec Eside unbound = 0.9 0

Binding Dodec Bside bound to Dodec Aside unbound = 0.9 0
Binding Dodec Bside bound to Dodec Bside unbound = 0.9 0
Binding Dodec Bside bound to Dodec Cside unbound = 0.9 0
Binding Dodec Bside bound to Dodec Dside unbound = 0.9 0
Binding Dodec Bside bound to Dodec Eside unbound = 0.9 0

Binding Dodec Cside bound to Dodec Aside unbound = 0.9 0
Binding Dodec Cside bound to Dodec Bside unbound = 0.9 0
Binding Dodec Cside bound to Dodec Cside unbound = 0.9 0
Binding Dodec Cside bound to Dodec Dside unbound = 0.9 0
Binding Dodec Cside bound to Dodec Eside unbound = 0.9 0

Binding Dodec Dside bound to Dodec Aside unbound = 0.9 0
Binding Dodec Dside bound to Dodec Bside unbound = 0.9 0
Binding Dodec Dside bound to Dodec Cside unbound = 0.9 0
Binding Dodec Dside bound to Dodec Dside unbound = 0.9 0
Binding Dodec Dside bound to Dodec Eside unbound = 0.9 0

Binding Dodec Eside bound to Dodec Aside unbound = 0.9 0
Binding Dodec Eside bound to Dodec Bside unbound = 0.9 0
Binding Dodec Eside bound to Dodec Cside unbound = 0.9 0
Binding Dodec Eside bound to Dodec Dside unbound = 0.9 0
Binding Dodec Eside bound to Dodec Eside unbound = 0.9 0

unbound links

Binding Dodec Aside unbound to Dodec Aside bound = 0.9 0
Binding Dodec Aside unbound to Dodec Bside bound = 0.9 0
Binding Dodec Aside unbound to Dodec Cside bound = 0.9 0
Binding Dodec Aside unbound to Dodec Dside bound = 0.9 0
Binding Dodec Aside unbound to Dodec Eside bound = 0.9 0

Binding Dodec Bside unbound to Dodec Aside bound = 0.9 0
Binding Dodec Bside unbound to Dodec Bside bound = 0.9 0

Binding Dodec Bside unbound to Dodec Cside bound = 0.9 0
Binding Dodec Bside unbound to Dodec Dside bound = 0.9 0
Binding Dodec Bside unbound to Dodec Eside bound = 0.9 0

Binding Dodec Cside unbound to Dodec Aside bound = 0.9 0
Binding Dodec Cside unbound to Dodec Bside bound = 0.9 0
Binding Dodec Cside unbound to Dodec Cside bound = 0.9 0
Binding Dodec Cside unbound to Dodec Dside bound = 0.9 0
Binding Dodec Cside unbound to Dodec Eside bound = 0.9 0

Binding Dodec Dside unbound to Dodec Aside bound = 0.9 0
Binding Dodec Dside unbound to Dodec Bside bound = 0.9 0
Binding Dodec Dside unbound to Dodec Cside bound = 0.9 0
Binding Dodec Dside unbound to Dodec Dside bound = 0.9 0
Binding Dodec Dside unbound to Dodec Eside bound = 0.9 0

Binding Dodec Eside unbound to Dodec Aside bound = 0.9 0
Binding Dodec Eside unbound to Dodec Bside bound = 0.9 0
Binding Dodec Eside unbound to Dodec Cside bound = 0.9 0
Binding Dodec Eside unbound to Dodec Dside bound = 0.9 0
Binding Dodec Eside unbound to Dodec Eside bound = 0.9 0

#bound links

Binding Dodec Aside bound to Dodec Aside bound = 0.9 0
Binding Dodec Aside bound to Dodec Bside bound = 0.9 0
Binding Dodec Aside bound to Dodec Cside bound = 0.9 0
Binding Dodec Aside bound to Dodec Dside bound = 0.9 0
Binding Dodec Aside bound to Dodec Eside bound = 0.9 0

Binding Dodec Bside bound to Dodec Aside bound = 0.9 0
Binding Dodec Bside bound to Dodec Bside bound = 0.9 0
Binding Dodec Bside bound to Dodec Cside bound = 0.9 0
Binding Dodec Bside bound to Dodec Dside bound = 0.9 0
Binding Dodec Bside bound to Dodec Eside bound = 0.9 0

Binding Dodec Cside bound to Dodec Aside bound = 0.9 0
Binding Dodec Cside bound to Dodec Bside bound = 0.9 0
Binding Dodec Cside bound to Dodec Cside bound = 0.9 0
Binding Dodec Cside bound to Dodec Dside bound = 0.9 0
Binding Dodec Cside bound to Dodec Eside bound = 0.9 0

```
Binding Dodec Dside bound to Dodec Aside bound = 0.9 0
Binding Dodec Dside bound to Dodec Bside bound = 0.9 0
Binding Dodec Dside bound to Dodec Cside bound = 0.9 0
Binding Dodec Dside bound to Dodec Dside bound = 0.9 0
Binding Dodec Dside bound to Dodec Eside bound = 0.9 0
```

```
Binding Dodec Eside bound to Dodec Aside bound = 0.9 0
Binding Dodec Eside bound to Dodec Bside bound = 0.9 0
Binding Dodec Eside bound to Dodec Cside bound = 0.9 0
Binding Dodec Eside bound to Dodec Dside bound = 0.9 0
Binding Dodec Eside bound to Dodec Eside bound = 0.9 0
```

```
# environment constants
```

```
Temperature 310
Viscosity    0.69
TimeScale    100
```

```
# Finally, specify the mix of molecules in solution.
```

```
Mix Dodec 100%
```

Icosahedron

```
# Another example pddf ... spheres making up the faces of an icosahedron
#
```

```
Model Icos
```

```
{
```

```
  Sphere a 2.7 <0,0,0> red
```

```
  #site    name      <position> <direction> twist
```

```
    site Aside      <0,-2.490712,-0.95136732>
                        <0,-2.490712,-0.95136732>
                        <0,0.95136732,-2.490712>
```

```
    site Bside      <-2.1570199,1.245356,-0.95136732>
                        <-2.1570199,1.245356,-0.95136732>
                        <-.82390828,-0.47570,-2.490712>
```

```
    site Cside      <2.1570199,1.245356,-0.95136732>
                        <2.1570199,1.245356,-0.95136732>
                        <-.82390828,-0.47570,-2.490712>
```

```
  State unbound
```

```
  State bound
```

```
    colour a = blue
```

```
  Event bind { Aside Aside }
            unbound -> bound
```

```
  Event bind { Aside Bside }
            unbound -> bound
```

```
  Event bind { Aside Cside }
            unbound -> bound
```

```
  Event bind { Bside Aside }
            unbound -> bound
```

```
  Event bind { Bside Bside }
```

```

        unbound -> bound
Event  bind { Bside Cside }
        unbound -> bound
Event  bind { Cside Aside }
        unbound -> bound
Event  bind { Cside Bside }
        unbound -> bound
Event  bind { Cside Cside }
        unbound -> bound

Event  break { all } bound -> unbound
}

# specify links and states, and give their binding and breaking chance
# links are transitive.
# (nb. for large models, these should be machine generated in some way -
# e.g. scripts or wordprocessor macros )

Binding Icos Aside unbound  to Icos Aside unbound = 0.01 0
Binding Icos Aside unbound  to Icos Bside unbound = 0.01 0
Binding Icos Aside unbound  to Icos Cside unbound = 0.01 0
Binding Icos Bside unbound  to Icos Aside unbound = 0.01 0
Binding Icos Bside unbound  to Icos Bside unbound = 0.01 0
Binding Icos Bside unbound  to Icos Cside unbound = 0.01 0
Binding Icos Cside unbound  to Icos Aside unbound = 0.01 0
Binding Icos Cside unbound  to Icos Bside unbound = 0.01 0
Binding Icos Cside unbound  to Icos Cside unbound = 0.01 0

Binding Icos Aside bound    to Icos Aside unbound = 0.9 0
Binding Icos Aside bound    to Icos Bside unbound = 0.9 0
Binding Icos Aside bound    to Icos Cside unbound = 0.9 0
Binding Icos Bside bound    to Icos Aside unbound = 0.9 0
Binding Icos Bside bound    to Icos Bside unbound = 0.9 0
Binding Icos Bside bound    to Icos Cside unbound = 0.9 0
Binding Icos Cside bound    to Icos Aside unbound = 0.9 0
Binding Icos Cside bound    to Icos Bside unbound = 0.9 0
Binding Icos Cside bound    to Icos Cside unbound = 0.9 0

Binding Icos Aside unbound  to Icos Aside bound = 0.9 0
Binding Icos Aside unbound  to Icos Bside bound = 0.9 0
Binding Icos Aside unbound  to Icos Cside bound = 0.9 0
Binding Icos Bside unbound  to Icos Aside bound = 0.9 0

```

```
Binding Icos Bside unbound to Icos Bside bound = 0.9 0
Binding Icos Bside unbound to Icos Cside bound = 0.9 0
Binding Icos Cside unbound to Icos Aside bound = 0.9 0
Binding Icos Cside unbound to Icos Bside bound = 0.9 0
Binding Icos Cside unbound to Icos Cside bound = 0.9 0
```

```
Binding Icos Aside bound to Icos Aside bound = 0.9 0
Binding Icos Aside bound to Icos Bside bound = 0.9 0
Binding Icos Aside bound to Icos Cside bound = 0.9 0
Binding Icos Bside bound to Icos Aside bound = 0.9 0
Binding Icos Bside bound to Icos Bside bound = 0.9 0
Binding Icos Bside bound to Icos Cside bound = 0.9 0
Binding Icos Cside bound to Icos Aside bound = 0.9 0
Binding Icos Cside bound to Icos Bside bound = 0.9 0
Binding Icos Cside bound to Icos Cside bound = 0.9 0
```

```
# environment constants
```

```
Temperature 310
Viscosity 0.69
TimeScale 100
```

```
# Finally, specify the mix of molecules in solution.
```

```
Mix Icos 100%
```

Appendix E

Tetrahedral Lattice .pddf file

```
# Another example pddf ... spheres in a diamond-style tetrahedral
# lattice structure
#

Model Diamond
{
  Sphere a 1.0 <0,0,0> red

  #site    name      <position> <direction> twist

  site Aside <1,1,1> <1,1,1> <0,1,-1>
  site Bside <1,-1,-1> <1,-1,-1> <0,-1,1>
  site Cside <-1,-1,1> <-1,-1,1> <0,1,1>
  site Dside <-1,1,-1> <-1,1,-1> <0,-1,-1>

  State unbound

  State bound

  colour a = blue

  Event bind { }          unbound -> bound
  Event break { all }     bound -> unbound
}
```

unbound links

Binding Diamond Aside unbound to Diamond Aside unbound = 0.001 0
Binding Diamond Aside unbound to Diamond Bside unbound = 0.001 0
Binding Diamond Aside unbound to Diamond Cside unbound = 0.001 0
Binding Diamond Aside unbound to Diamond Dside unbound = 0.001 0

Binding Diamond Bside unbound to Diamond Aside unbound = 0.001 0
Binding Diamond Bside unbound to Diamond Bside unbound = 0.001 0
Binding Diamond Bside unbound to Diamond Cside unbound = 0.001 0
Binding Diamond Bside unbound to Diamond Dside unbound = 0.001 0

Binding Diamond Cside unbound to Diamond Aside unbound = 0.001 0
Binding Diamond Cside unbound to Diamond Bside unbound = 0.001 0
Binding Diamond Cside unbound to Diamond Cside unbound = 0.001 0
Binding Diamond Cside unbound to Diamond Dside unbound = 0.001 0

Binding Diamond Dside unbound to Diamond Aside unbound = 0.001 0
Binding Diamond Dside unbound to Diamond Bside unbound = 0.001 0
Binding Diamond Dside unbound to Diamond Cside unbound = 0.001 0
Binding Diamond Dside unbound to Diamond Dside unbound = 0.001 0

half bound links

Binding Diamond Aside bound to Diamond Aside unbound = 0.9 0
Binding Diamond Aside bound to Diamond Bside unbound = 0.9 0
Binding Diamond Aside bound to Diamond Cside unbound = 0.9 0
Binding Diamond Aside bound to Diamond Dside unbound = 0.9 0

Binding Diamond Bside bound to Diamond Aside unbound = 0.9 0
Binding Diamond Bside bound to Diamond Bside unbound = 0.9 0
Binding Diamond Bside bound to Diamond Cside unbound = 0.9 0
Binding Diamond Bside bound to Diamond Dside unbound = 0.9 0

Binding Diamond Cside bound to Diamond Aside unbound = 0.9 0
Binding Diamond Cside bound to Diamond Bside unbound = 0.9 0
Binding Diamond Cside bound to Diamond Cside unbound = 0.9 0
Binding Diamond Cside bound to Diamond Dside unbound = 0.9 0

Binding Diamond Dside bound to Diamond Aside unbound = 0.9 0
Binding Diamond Dside bound to Diamond Bside unbound = 0.9 0
Binding Diamond Dside bound to Diamond Cside unbound = 0.9 0

Binding Diamond Dside bound to Diamond Dside unbound = 0.9 0

unbound links

Binding Diamond Aside unbound to Diamond Aside bound = 0.9 0

Binding Diamond Aside unbound to Diamond Bside bound = 0.9 0

Binding Diamond Aside unbound to Diamond Cside bound = 0.9 0

Binding Diamond Aside unbound to Diamond Dside bound = 0.9 0

Binding Diamond Bside unbound to Diamond Aside bound = 0.9 0

Binding Diamond Bside unbound to Diamond Bside bound = 0.9 0

Binding Diamond Bside unbound to Diamond Cside bound = 0.9 0

Binding Diamond Bside unbound to Diamond Dside bound = 0.9 0

Binding Diamond Cside unbound to Diamond Aside bound = 0.9 0

Binding Diamond Cside unbound to Diamond Bside bound = 0.9 0

Binding Diamond Cside unbound to Diamond Cside bound = 0.9 0

Binding Diamond Cside unbound to Diamond Dside bound = 0.9 0

Binding Diamond Dside unbound to Diamond Aside bound = 0.9 0

Binding Diamond Dside unbound to Diamond Bside bound = 0.9 0

Binding Diamond Dside unbound to Diamond Cside bound = 0.9 0

Binding Diamond Dside unbound to Diamond Dside bound = 0.9 0

#bound links

Binding Diamond Aside bound to Diamond Aside bound = 0.9 0

Binding Diamond Aside bound to Diamond Bside bound = 0.9 0

Binding Diamond Aside bound to Diamond Cside bound = 0.9 0

Binding Diamond Aside bound to Diamond Dside bound = 0.9 0

Binding Diamond Bside bound to Diamond Aside bound = 0.9 0

Binding Diamond Bside bound to Diamond Bside bound = 0.9 0

Binding Diamond Bside bound to Diamond Cside bound = 0.9 0

Binding Diamond Bside bound to Diamond Dside bound = 0.9 0

Binding Diamond Cside bound to Diamond Aside bound = 0.9 0

Binding Diamond Cside bound to Diamond Bside bound = 0.9 0

Binding Diamond Cside bound to Diamond Cside bound = 0.9 0

Binding Diamond Cside bound to Diamond Dside bound = 0.9 0

Binding Diamond Dside bound to Diamond Aside bound = 0.9 0

```
Binding Diamond Dside bound to Diamond Bside bound = 0.9 0
Binding Diamond Dside bound to Diamond Cside bound = 0.9 0
Binding Diamond Dside bound to Diamond Dside bound = 0.9 0
```

```
# environment constants
```

```
Temperature 310
Viscosity    0.69
TimeScale    100
```

```
# Finally, specify the mix of molecules in solution.
```

```
Mix Diamond 100%
```

Appendix G

Povray Microtubule model file

This is a model file for the povray raytracer program (<http://www.povray.org>). It constructs a model microtubule with 13 protofilaments. It can also be used as a base to model microtubules with different numbers of filaments or different structures, or to model microtubules with splayed or curved tips.

Note that the language used is the internal povray scene description language. This language is similar to C, and uses the '#' symbol for compiler directives, rather than as the UNIX style comments they are used for in the .pddf files previously. Instead, the '//' token is used for comments instead.

```

#include "colors.inc"
#include "shapes.inc"
#include "textures.inc"

global_settings { max_trace_level 20 }

camera {
    location <150, 10, 0>
    look_at <0,0,0>
    orthographic
}
background {colour rgb<0.8,0.8,0.8> }

light_source { <1500, -100, -100> color rgb <1,1,1>}

#declare dimer =
blob
{
    threshold 0.15
    component 1.0,1.0, <0,0,1>
    component 1.0,1.0, <0,0,-1>
    texture {
        pigment { color rgbf <0.95,0.95,0.95,0.95> }
        finish {diffuse 1.0}
    }
    no_shadow
}

#declare helix_no = 5;                // number of dimer helices
#declare pf_count = 13;               // number of protofilaments
#declare dimer_size = 4;
#declare delta_theta = 360/pf_count;  // how much to rotate
    // what the offset between p.f.s is
#declare delta_z      = dimer_size*helix_no/pf_count;

```

```

// function for drawing a single protofilament
#macro drawFilament(rot)
#local length=-10;
#if (rot>6)
    #declare length=length-(rot*2)+12;
#else
    #declare length=length+(rot*2)-12;
#end

#while (length < 30)
    #declare z_dist = (length*dimer_size) + (delta_z*rot);
    #if (z_dist < 0)
        object
        {
            dimer
            #declare rot_angle = (z_dist)*.275;
            #declare rPos =
                vaxis_rotate( <0,-200,0>, <-1,0,0>, rot_angle );
            #declare rPos = rPos + <0,200,0>;
            rotate -x*rot_angle
            #declare circ_offset =
                vaxis_rotate(<0,4,0>, <0,0,1>, delta_theta*rot);
            translate circ_offset
            translate rPos
        }
    #else
        object
        {
            dimer translate <0,4,z_dist> rotate z*delta_theta*rot
        }
    #end
    #declare length=length+1;
#end
#end

```

```

#declare pf=0;
#while (pf < pf_count)                                // draw each protofilament
    drawFilament(pf)
    #declare pf=pf+1;
#end

#declare randomStream = seed(1);                        // draw background dimers
#declare i = 0;

#while (i < 1000)
    object
    {
        dimer translate <rand(randomStream)*200-100,
                        rand(randomStream)*200-100,
                        rand(randomStream)*200-100>
        rotate -x*rand(randomStream)*360
        rotate -y*rand(randomStream)*360
    }
    #declare i=i+1;
#end

```

References

Ref. 1

Appendix G

Povray Microtubule model file

This is a model file for the povray raytracer program (<http://www.povray.org>). It constructs a model microtubule with 13 protofilaments. It can also be used as a base to model microtubules with different numbers of filaments or different structures, or to model microtubules with splayed or curved tips.

Note that the language used is the internal povray scene description language. This language is similar to C, and uses the '#' symbol for compiler directives, rather than as the UNIX style comments they are used for in the .pddf files previously. Instead, the '/' token is used for comments instead.

```

#include "colors.inc"
#include "shapes.inc"
#include "textures.inc"

global_settings { max_trace_level 20 }

camera {
    location <150, 10, 0>
    look_at <0,0,0>
    orthographic
}
background {colour rgb<0.8,0.8,0.8> }

light_source { <1500, -100, -100> color rgb <1,1,1>}

#declare dimer =
blob
{
    threshold 0.15
    component 1.0,1.0, <0,0,1>
    component 1.0,1.0, <0,0,-1>
    texture {
        pigment { color rgbf <0.95,0.95,0.95,0.95> }
        finish {diffuse 1.0}
    }
    no_shadow
}

#declare helix_no = 5;           // number of dimer helices
#declare pf_count = 13;         // number of protofilaments
#declare dimer_size = 4;
#declare delta_theta = 360/pf_count; // how much to rotate
    // what the offset between p.f.s is
#declare delta_z = dimer_size*helix_no/pf_count;

```

```

// function for drawing a single protofilament
#macro drawFilament(rot)
#local length=-10;
#if (rot>6)
    #declare length=length-(rot*2)+12;
#else
    #declare length=length+(rot*2)-12;
#end

#while (length < 30)
    #declare z_dist = (length*dimer_size) + (delta_z*rot);
    #if (z_dist < 0)
        object
        {
            dimer
            #declare rot_angle = (z_dist)*.275;
            #declare rPos =
                vaxis_rotate( <0,-200,0>, <-1,0,0>, rot_angle );
            #declare rPos = rPos + <0,200,0>;
            rotate -x*rot_angle
            #declare circ_offset =
                vaxis_rotate(<0,4,0>, <0,0,1>, delta_theta*rot);
            translate circ_offset
            translate rPos
        }
    #else
        object
        {
            dimer translate <0,4,z_dist> rotate z*delta_theta*rot
        }
    #end
    #declare length=length+1;
#end
#end

```

```

#declare pf=0;
#while (pf < pf_count)                                // draw each protofilament
    drawFilament(pf)
    #declare pf=pf+1;
#end

#declare randomStream = seed(1);    // draw background dimers
#declare i = 0;

#while (i < 1000)
    object
    {
        dimer translate <rand(randomStream)*200-100,
                        rand(randomStream)*200-100,
                        rand(randomStream)*200-100>
        rotate -x*rand(randomStream)*360
        rotate -y*rand(randomStream)*360
    }
    #declare i=i+1;
#end

```

References

1. Image of cell stained with Coomassie blue from Colin Smith, sourced from Alberts B., et al, (1994) *Molecular Biology of the Cell*, Garland Publishing, New York, p787
2. Figure from ibid, p962
3. Radford J.E., Vesik, M., and Overall, R.L. (1998) *Callose deposition at plasmodesmata*, Protoplasma Vol 201 pp 30-37
4. Epel, B.L., van Lent, J.W.M., Cohen, L., Kotlizky, G., Katz, A., and Yahalom, A. (1996) *A 41 kDa protein isolated from maize mesocotyl cell walls immunolocalizes to plasmodesmata*, Protoplasma Vol 191, pp 70-78
5. Yahalom, A., Lando, R., Katz, A. and Epel, B.L. (1998) *A calcium-dependent protein kinase is associated with maize mesocotyl plasmodesmata*, J. Plant Phys., Vol 153(3-4) pp 354-362
6. Blackman, L. M., Overall, R. L. (1998) *Immunolocalisation of the cytoskeleton to plasmodesmata of Chara corallina*, Plant J. Vol 14, pp 733-741.
7. White, R.G., Badelt, K, Overall, R.L. and Vesik, M. (1994) *Actin associated with plasmodesmata*, Protoplasma Vol 180 pp 169-184
8. Radford, J.E. and White, R.G., (1998) *Localization of a myosin-like protein to plasmodesmata*, The Plant Journal, Vol 14(6), pp 743-450
9. Popper, K., as expounded over the course of his life, esp in (1934) *The Logic of Scientific Discovery*.
10. Images on headpiece courtesy Radford, J.E., Dept. Biological Sciences, Monash University, Melbourne, Australia.
11. (Detail from) TEM Image from Alberts, B. et al, *Molecular Biology of the Cell*, (1994), Garland Publishing Inc., New York, Fig. 16-49, p789
12. Li, Qingqin & Joshi, H.C., (1995) *γ -Tubulin Is a Minus End-specific Microtubule Binding Protein*, J. Cell. Biol., Vol 131, No 1., pp 207-214.
13. Taiz, L. and Zeigler, E. (1991) *Plant Physiology*, Benjamin/Cummings Publishing Company, inc., California.
14. Epel, B.L, (1994) *Plasmodesmata: Composition, Structure and Trafficking*, Plant Mol.Biol., Vol 26, pp 1343-1356
15. Lucas, W.J., Ding, B., Van der Schoot, C. (1993) *Plasmodesmata and the supracellular nature of plants* New Phytol. Vol 125, pp 435-476
16. White, R.G. et al (1994) op. cit.

17. Radford, J.E. et al. (1998) op. cit.
18. Hepler, P.K., and Bonsignore, C.L. (1990) *Caffeine inhibition of cytokinesis: ultrastructure of cell plate formation/degradation*, Protoplasma Vol 157 pp182-192
19. Epel, B.L. op. cit.
20. Van Lent, J., Storms, M., Van der Meer, F., Wellink, J., and Goldbach, R. (1991) *Tubular structures involved in movement of cowpea mosaic virus are also formed in infected cowpea protoplasts.*, J. Gen. Virol. Vol 72, pp 2615-2623
21. Kim, K.S., and Lee K. W. (1992) *Geminivirus induced microtubules and their suggested role in cell-to-cell movement.* Phyto.Path. Vol 82 pp 664-669
22. Wolf, S., Deom, C.M., Beachy, R.N. and Lucas, W.J. (1989) *Movement protein of tobacco mosaic virus modifies plasmodesmatal size exclusion limit* Science Vol 246, pp 377-379.
23. Ghoshory, S., Lartey, R., Sheng, J., and Citovsky, V. (1997) *Transport of proteins and nucleic acids through plasmodesmata* Ann. Rev. Plant Phys. Plant Mol. Biol. Vol 48: pp 27-50
24. The material in this chapter is a summary of undergraduate level Physical Chemistry, and a large number of excellent texts exist. The most complete is: Atkins, P.W., *Physical Chemistry*, 5th Ed. (1994), OUP, Oxford.
25. Also good is Chap 1-6 of Pascoe, K.J., *Properties of Materials for Electrical Engineers*, (1973) John Wiley & Sons, Bristol.
26. Also Chap 1-5 and 15-21 of Chang, R. *Physical Chemistry with Applications to Biological Systems* (1981), Macmillan, New York
27. Image from Atkins op. cit., p 707
28. The actual distance, so long as it is small on the time and distance scale being considered, is not actually very important. It is usually assumed (e.g. Atkins op. cit. p 855) for a liquid to be on the order of a small molecular diameter, e.g. 210pm for an SO_4^{2-} molecule, and would be less for a large molecule. Hence for simulations dealing in nanometres, the Brownian motion approximation is valid for all molecules in liquid. For a gas the distance is much greater, e.g. 70nm for N_2 at room temperature (Atkins op.cit. p 40).
29. Pascoe, K.J. op cit., P 76
30. These are obviously only orders of magnitude figures, since actual values will vary greatly depending on molecular size, temperature, pressure etc. See Atkins op. cit. chapter 1.
31. Atkins, op. cit., chap 27

32. James, J.F., (1995) *A student's guide to Fourier transforms with applications in physics and engineering*, Cambridge University Press, Cambridge
33. The material presented here is relatively complex, (i.e. it would be covered in an undergraduate mathematics course), and space does not permit a full exposition of Fourier transforms. The reader is directed to a good mathematics text, such as Kreyszig, E, *Advanced Engineering Mathematics*, (1983), John Wiley & Sons, Inc., New York.
34. All good Physics and Engineering texts that cover mathematical solutions to problems are implicitly analytical modelling texts: e.g. Kreyszig op cit.
35. A large number of textbooks on numerical techniques are available, e.g. Smith, G.D., *Numerical Solution of Partial Differential Equations: Finite Difference Methods (3rd Ed.)*, (1985), Oxford University Press, Oxford.
36. An intriguing description of the process of mass slide rule calculation can be found in the description of airship manufacture, in Nevil Shute's autobiography: Shute, N., *Slide Rule*, (1956), William Heinemann Ltd., London, Chapter 3
37. The author's grandfather, Mr H.C.V.Woollard, also described how competing teams of slide rule operators worked to calculate the mesh modeling the weakest cross-section of double-curvature dams in the Tasmanian Hydro-Electric Commission in the 1950s. The use of multiple teams was an important error checking mechanism.
38. Watson, J. D. (1968), *The Double Helix*, Penguin Books Ltd., Ringwood, Victoria, Australia.
39. E.g. Chrétien, D., Fuller, S.D. & Karsenti, E. (1995) *Structure of Growing Microtubule Ends: Two-Dimensional Sheets Close into Tubes at Variable Rates*, J. Cell. Biol., Vol 129, No 5, pp 1311-1328
40. Oosawa, F. & Asakura, S. (1975) *Thermodynamics of the Polymerisation of Protein* (Academic Press, New York)
41. Ibid, eqn 43 pp 48
42. Voter, W.A. & Erickson, H.P. (1984) *The Kinetics of Microtubule Assembly*, J. Biol. Chem., Vol 259, No 16, pp 10430-10438
43. Flyvbjerg, H., Jobs, E. & Leibler, S. (1996) *Kinetics of self-assembling microtubules: An "inverse problem" in biochemistry*, Proc. Natl. Acad. Sci. USA, Vol 93, pp 5975-5979
44. Ibid, pp 5978
45. Berger et al (1994) *Local rule-based theory of virus shell assembly*, Proc. Natl. Acad. Sci., USA, Vol 91, pp 7732-7736

46. Tran, P.T., Joshi, P. and Salmon, E.D. (1997) *How Tubulin Subunits are lost from the Shortening Ends of Microtubules*, J. Struct. Bio., Vol 118, pp107-118
47. Gliksman, N.R., Skibbens, R.W. & Salmon, E.D. (1993) *How the Transition Frequencies of Microtubule Dynamic Instability (Nucleation, Catastrophe, and Rescue) Regulate Microtubule Dynamics in Interphase and Mitosis: Analysis Using a Monte Carlo Computer Simulation*, Mol Bio of the Cell, Vol 4, 1035-1050
48. Odde, D.J., Buettner, H.M. & Cassimeris, L. (1996) *Spectral Analysis of Microtubule Assembly Dynamics*, AIChE Journal, Vol 42, pp 1434-1442
49. Such devices are still speculative. Possible the best exposition occurs in popular literature such as Stephenson, N., (1996) *The Diamond Age*, Penguin, London
50. Badger, J et al. (1988) Proc. Natl. Acad. Sci. U.S.A. Vol 85, pp 3304-3308
51. Schwartz, R., P.W.Shor, P.E.Prevelige Jr., and B. Berger et al. (1998), *Local Rules Simulation of the Kinetics of Virus Capsid Self-Assembly*, Biophysical Journal, Vol 75, pp2626-2636
52. Prevelige, P.E. Jr., (1998), *Inhibiting virus-capsid assembly by altering the polymerisation pathway*, Tibtech, Vol 16, pp 61-65
53. Background information on molecular motion in liquids is available in most materials science textbooks, e.g. Pascoe K.J., (1972) *Properties of Materials for Electrical Engineers*, (John Wiley & Sons, Chichester)
54. Atkins, P.W. (1995) op.cit. pp 832
55. ibid, pp 831
56. ibid, p855
57. E.g. Iranpour R, Chacon P (1988) *Basic stochastic processes*. Macmillan Publishing Company. London. Chapter 7.
58. Atkins, P.W., (1995) op.cit., pp 853-854
59. Ibid. See Chapter 23, pp 795-6, and Chapter 24, 846-856 for a more In depth treatment.
60. Ibid, p 795
61. Ibid, Appendix C26
62. Ibid, p 855
63. Tanford, C., *Physical chemistry of macromolecules*, Wiley, New York (1961) (cited in Atkins op.cit.)
64. See any introductory physical chemistry text, e.g. Pascoe, K.J., op. cit.

65. Janin, J., (1995), *Principles of protein-protein recognition from structure to thermodynamics*, Biochimie Vol 77 pp 497-505.
66. Schwartz, R., et al. (1998) op. cit.
67. Ibid, pp 2631 "It has not been possible to determine the exact mapping between time steps and physical time in terms of time-dependent phenomena such as the diffusion rate". However, while this was not necessary for Schwartz et al, there is nothing inherent in their model to suggest that it could not be done in future.
68. This can be seen by the following two papers (ref 69, 70) which, while disagreeing on the degree of stability, both agree that the microtubule wall is a fairly stable construct.
69. E.g. modeling of 'holes' in the microtubule wall in Semenov, M.V., (1996) *New Concept of Microtubule Dynamics and Microtubule Motor Movement and New Model of Chromosome Movement in Mitosis*, J. Theor. Biol., Vol 179, pp 91-117
70. E.g. apparent stability of microtubule wall in Waterman-Storer, C.M. and Salmon, E.D. (1998), *How Microtubules Get Fluorescent Speckles*, Biophysical Journal, Vol 75, pp 2059-2069
71. Martin, R.S., Schilstra, M.J. and Bayley, P.M. (1993), *Dynamic Instability of Microtubules: Monte Carlo Simulation and Application to Different Types of Microtubule Lattice*, Biophysical Journal, Vol 65, pp 578-596
72. Schwartz, R et al. (1998) op. cit.
73. Direct structural evidence of this theory appears in Nogales, E., Whittaker, M., Milligan, R.A. and Downing, K.H. (1999) *High-Resolution Model of the Microtubule*, Cell, Vol 96, p86.
74. e.g. Bayley, P.M., Schilstra, M.J. and Martin, S.R. (1990), *Microtubule dynamic instability: numerical simulation of microtubule transition properties using a lateral cap model*, J. Cell Sci., Vol 95, pp 33-48
75. Booch, G., *Object-Oriented Analysis and Design (2nd Ed)*, (1994) Addison Wesley, Menlo Park, California
76. refer any standard microtubule text. Original paper; Allen C & Borisy GG (1974) *Structural polarity and directional growth of microtubules of Chlamydomonas flagella*, J. Mol. Biol, Vol 90 pp 381-402
77. Dr. Damian Conway, Dept Computer Science and Software Engineering, Monash University, Melbourne, Australia - personal communication.
78. Oosawa, F. & Asakura, S., op. cit.
79. For further background see Amos, L.A. and Amos, B.A.(1991), *Molecules of the Cytoskeleton*, Macmillan, London, Chapter 3 pp 42-55
80. Image from ibid, p 51

81. Engleman, E.H. (1985) *The structure of the actin thin filament*. J.Muscle Res. & Cell Motil., Vol 6, pp 129-151
82. Image from Amos, op. cit., p43
83. Oosawa, F. and Asakura, S. (1975), *Thermodynamics of the Polymerisation of Protein*, Academic Press, London.
84. Figure ibid p50
85. Kawamura, M., and Maruyama, K. (1970), *Electron microscopic particle length of F-actin polymerised in vitro*, J. Biochem, Vol 67, pp 437-457
86. Kawamura, M., and Maruyama, K. (1972), *A further study of electron microscopic particle length of F-actin polymerised in vitro*, J. Biochem, Vol 72, 179-188
87. Civelekoglu, G., and Edelstein-Keshet, L., (1994), *Modelling the dynamics of f-actin in the cell*, Bull. Math. Biol., Vol 56, No 4., pp 587-616
88. Sherratt, J.A. and Lewis, J., (1993), *Stress induced alignment of actin filaments and the mechanism of cytogel*, Bull. Math. Biol., Vol 55, pp 637-654
89. Alt. W., (1987) *Mathematical models in actin-myosin interaction*, in Fortshritte der Zoology, Nature and Function of Cytoskeletal Proteins in Motility and Transport, Band 34, K.E. Wohlfarth-Bottermann(Ed)., pp 219-230, Stuttgart: Gustav Fisher Verlag, cited Civelekoglu op. cit.
90. Pollard, T.D. (1986), *Rate constants for the reactions of ATP- and ADP- actin with the ends of actin filaments.*, J. Cell Biol., Vol 103, pp 2747-2754
91. Ibid
92. Amos, L.A. and Amos, B.A. (1991), *Molecules of the Cytoskeleton*, Macmillan, London, Chapter 3, p52
93. Oosawa, F. and Asakura, S. (1975), *Thermodynamics of the Polymerisation of Protein*, Academic Press, London.
94. Ibid.
95. This figure, which was widely used by the author, was read off the graph of water viscosity that appears in Atkins (op. cit.) on p834. A later calculation by the author using the empirical equation (Atkins, footnote to Appendix C table 24.3):

$$\log(\eta_{20} / \eta) = \frac{1.37023(t - 20) + 8.36 \times 10^{-4} (t - 20)^2}{109 + t}$$
 gives the more exact value 0.69 cP. The effect of this adjustment is to make water more viscous; hence the movement of molecules in simulations using the smaller figure is slightly faster than is strictly correct.
96. Oosawa, F. and Asakura. S. op. cit. p46

97. For further background see Amos, L.A. and Amos, B.A.(1991), *Molecules of the Cytoskeleton*, Macmillan, London, Chapter 7 pp 117-141
98. e.g. Gaskin, F., Cantor, C.R., and Shelanski, M.L. (1974), *Turbidimetric Studies of the in-Vitro Assembly and Disassembly of Porcine Neurotubules*, J. Mol. Biol., Vol 89, pp 737-758
99. Betts, C., 1997, *Simulating Microtubule Assembly*, Molecular Biophysics of the Cytoskeleton conference, Banff, Canada (hosted University of Alberta).
100. Image from ibid, p 134
101. Although somewhat controversial, especially in detail, this is the theory presented in textbooks such as Amos op.cit., and papers such as Bayley, P.M., Schilstra, M.J., and Martin, S.R., (1989) *A lateral cap model for microtubule dynamic instability* FEBS Lett. Vol 259, pp 181-184, and is used in existing models in papers such as Flyvbjerg, H., Holy, T.E., and Leibler, S., (1994), *Stochastic Dynamics of Microtubules: A Model for Caps and Catastrophes*, Phys. Rev. Lett. Vol 73, No. 17, pp 2372 - 2375
102. The cap theory is mathematically modeled in Flyvbjerg, H., Holy, T.E., and Leibler, S., (1996), *Microtubule dynamics: Caps, catastrophes, and coupled hydrolysis*, Phys Rev. E., Vol 54, No 5, pp 5538-5560
103. Jobs, E., Wolf, D.E., and Flyvbjerg, H., (1997), *Modeling Microtubule Oscillations*, Phys. Rev. Lett. Vol 29, No.3., pp 519-522
104. Amos & Amos, op.cit. p117-119
105. Oosawa, F. and Asakura, S. op. cit.
106. Fygenson, D.K. et al., (1995), *Spontaneous nucleation of microtubules* Phys. Rev. E., Vol 51, No. 5, pp 5058-5063
107. Voter, W. A. and Erickson, H.P. (1984), *The Kinetics of Microtubule Assembly: Evidence for a Two-Stage Nucleation Mechanism*, J. Bio. Chem., Vol 259, No 16, pp 10430-10438.
108. e.g. Flyvbjerg, H., Jobs, E. and Leibler, S. (1996), *Kinetics of self-assembling microtubules: An "inverse problem" in biochemistry*, Proc. Natl. Acad. Sci, Vol 93, pp5975-5979.
109. Tran, P.T., Joshi, P. and Salmon, E.D. (1997), *How Tubulin Subunits are Lost from the Shortening Ends of Microtubules*, J. Struct. Biol. Vol 118, pp 107-118, esp. p114.
110. Chrétien, D., Fuller, S.D. and Karsenti, E., (1995) *Structure of Growing Microtubule Ends: Two-Dimensional Sheets Close Into Tubes at Variable Rates*, J. Cell. Biol., Vol 129, No.5, pp 1311-1328

111. Chrétien, D., Flyvbjerg, H., Fuller, S.D., (1998) *Limited flexibility of the inter- protofilament bonds in microtubules assembled from pure tubulin*, Eur. Biophys. J., Vol 27, pp 490-500.
112. Jánosi, I.M., Chrétien, D. and Flyvbjerg, H., (1998) *Modeling elastic properties of microtubule tips and walls*, Eur. Biophys. J. Vol 27, pp 501-513.
113. Ibid.
114. Walker, R.A., O'Brien, E.T., Pryer, N.K., Soboeiro, M.R., Voter, W.A., Erickson, H.P. & Salmon, E.D. (1988), *Dynamic instability of individual microtubules analyzed by video light microscopy: rate constants and transition frequencies.*, J. Cell Biol., Vol 103, pp 2747-2754
115. Ibid
116. Amos, L.A. and Amos, B.A.(1991), *Molecules of the Cytoskeleton*, Macmillan, London, Chapter 7 p 136
117. e.g. the various microtubule severing experiments, esp. Walker, R.A., Inoué, S., and Salmon, E.D. (1989), *Asymmetric behaviour of severed microtubule ends after ultraviolet-microbeam irradiation of individual microtubules in vitro*, J. Cell Biol, Vol 108, pp 931-37
118. cf review article Desai A. and Mitchison, J, (1997) *Microtubule Polymerization Dynamics*, Annu. Rev. Cell Dev. Biol., Vol 13 pp 83-117, esp sections “*The GTP CAP Model*” p95-97, “*Structural Basis of Dynamic Instability*” pp97-99 and “*Relationship of Structural and Chemical Transitions*” pp 99-100
119. Nogales, E., Whittaker, M., Milligan, R.A., and Downing, K.H., (1999) *High Resolution Model of the Microtubule*, Cell, Vol 96, pp 79-88
120. The model receives strong support from Nogales, E. *ibid*.
121. Amos, L.A. and Amos, B.A. *op. cit.* Chapter 7
122. Semënov, V. M., (1995) *New Concept of Microtubule Dynamics and Microtubule Motor Movement and New Model of Chromosome Movement in Mitosis*, J. theor. Biol., Vol 179 pp 91-117
123. A series of papers examining this issue has appeared from the lab of Peter M. Bayley, e.g. Martin, S.R., Schilstra, M.J. and Bayley, P.M. (1993) *Dynamic Instability of Microtubules: Monte Carlo Simulation and Application to Different Types of Microtubule Lattice*, Biophys. J. Vol 65, pp 578-596.
124. Amos, L.A. and Amos, B.A. (1991) *op. cit.*
125. Alberts et al, (1994), *Molecular Biology of the Cell*, Garland Publishing, New York, Chapter 6 esp pp274-282.

126. Zhou, Z.H., Prasad, B.V.V., Jakana, J., Rixon, F. and Chiu, W. (1994) *Protein subunit structures in the herpes simplex virus capsid from 400 kV spot-scan electron cryomicroscopy*. J. Mol. Biol. Vol 242, pp456-469.
127. Zhou, Z.H., He, J., Jakana, J., Tatman, J., Rixon, F. and Chiu, W. (1995) *Assembly of VP26 in HSV-1 inferred from structures of wild-type and recombinant capsids*. Nature Struct. Biol. Vol 2, 1026-1030.
128. Zhou, Z. H., Chiu, W., Haskell, K., Spears, H. J., Jakana, J., Rixon, F. J. and Scott, L. R. (1998). *Refinement of herpesvirus B-capsid structure on parallel supercomputers*. Biophys. J. Vol 74, 576-588.
129. Zhou, Z. H., Macnab, S. J., Jakana, J., Scott, L. R., Chiu, W. & Rixon, F. J. (1998). *Identification of the sites of interaction between the scaffold and outer shell in HSV-1 capsids by difference electron imaging*. Proc. Natl. Acad. Sci. USA Vol 75, 2778-2783.
130. Schwartz, R. et al. (1998) op. cit.
131. Berger, B. et al. (1994) op. cit.
132. Prevelige, P.E. Jr., (1998), *Inhibiting virus-capsid assembly by altering the polymerisation pathway, Tibtech, Vol 16 pp 61-65*.
133. Z. Hong Zhou et al. (1995), *Assembly of VP26 in herpes simplex virus-1 inferred from structures of wild-type and recombinant capsids*, Nature Struc. Bio. Vol 2, No. 11, pp 1026-1030
134. Trus, B.L., Newcomb, W.W., Booy, F.P., Brown, J.C. & Steven, A.C. (1992), *Distinct monoclonal antibodies separately label the hexons or the pentons of herpes simplex virus capsid*, Proc. Natl. Acad. Sci. USA Vol 89, pp 11508-11512
135. See the website <http://ncmi.bcm.tmc.edu/Projects/icos/icos.html>
136. Ibid.
137. E.g. Badger, J. et al., (1988), *Structural analysis of a series of antiviral agents complexed with human rhinovirus 14*, Proc. Natl. Acad. Sci. USA Vol 85, pp 3304-3308
138. Prevelige, P.E. Jr., (1998) op. cit.
139. Image from Alberts, B. et al, *Molecular Biology of the Cell*, (1994), Garland Publishing Inc., New York, Fig. 16-2, p789
140. (Detail from) Image from Alberts, B. et al, *Molecular Biology of the Cell*, (1994), Garland Publishing Inc., New York, Fig. 16-49, p789
141. Image courtesy J. Radford, Monash University Dept. Biological Sciences.
142. Image courtesy J. Radford, Monash University Dept. Biological Sciences.

143. Image courtesy J.Radford, Monash University Dept. Biological Sciences.
144. Gonzalez, R.C. and Wintz, P., *Digital Image Processing*, (2nd ed. 1987), Addison-Wesley Publishing Company Inc., Reading Massachusetts, Chap 4.2.4 pp 158-160
145. Ding, B., Turgeon, R. and Parthasarathy, M.V. (1992) *Substructure of freeze-substituted plasmodesmata* Protoplasma, Vol 169, pp 28-41
146. Botha, C.E.J., Hartley, B.J. and Cross, R.H.M., *The Ultrastructure and Computer-enhanced Digital Image Analysis of Plasmodesmata at the Kranz Mesophyll-Bundle Sheath Interface of Themeda triandra var. imberbis (Retz) A. Camus in Conventionally-fixed Leaf Blades*, Ann. Bot. Vol 72: pp255-261
147. Overall, R.L., Wolfe, J. and Gunning, B.E.S., (1982) *Intercellular Communication in Azolla Roots: I. Ultrastructure of Plasmodesmata* Protoplasma, Vol 111, pp 134-150
148. The actin model is now well understood - see any standard reference such as Amos, L.A; Amos, W.B. *Molecules of the Cytoskeleton*, (1991) Guilford Press, New York
149. D.Chretien et al, *Lattice Defects in Microtubules: Protofilament Numbers Vary Within Individual Microtubules*, (1992) J. Cell Bio., Vol 117, pp 1031-1040
150. Ibid.
151. Alberts, B. et al. op.cit. p962
152. Gunning B.E.S. and Overall R.L., *Plasmodesmata and Cell-to-Cell Transport in Plants* (1983), BioScience Vol 33 No 4 pp 260-265
153. Ding B., et al, *Substructure of freeze substituted plasmodesmata*, (1992) Protoplasma Vol 169, pp 28-41.
154. Illustration from Lucas et al, *Tansley Review No. 58, Plasmodesmata and the supracellular nature of plants*, (1993) New Phytol, Vol 125, pp 435-476, based on Ding B., et al.(1992) above.
155. Thomson W.W. and Platt-Aloia, K. (1985), *The Ultrastructure of the Plasmodesmata of the Salt Glands of Tamarix as revealed by Transmission and Freeze-Fracture Electron Microscopy* Protoplasma Vol 125, pp 12-23
156. White R., personal communication.
157. Radford, J., *Ultrastructure of Plasmodesmata*, (1994) Honours Thesis, University of Sydney, Dept Biological Sciences.
158. Alberts, B. et al, *Molecular Biology of the Cell*, (1994), Garland Publishing Inc., New York, pp807-808
159. Voxels are a well known 3D graphics technique, mentioned in most good textbooks - e.g. Hearn, D. and Baker, M.P., *Computer Graphics (2nd ed.)*(1997) Prentice Hall, Sydney, Australia, pp 360-362.

160. Ibid. pp 203-205, 423
161. This is a common ray tracing technique; e.g. see Cook, R.L., *Stochastic Sampling and Distributed Ray Tracing*, Chapter 5 of 'An Introduction to Ray Tracing', edited by Glassner, A.S., 1989, Academic Press Ltd., San Diego, CA.
162. White, R. et al. (1994) op. cit.
163. Chrétien, D., Meteo, F., Verde, F., Karsenti, E., & Wade, R.H., (1992) *Lattice Defects in Microtubules: Protofilament Numbers Vary Within Individual Microtubules*, J. Cell Biol., Vol 117, No. 5., pp 1031-1040
164. Chrétien, D. (1991) *Apports de la cryomicroscopie électronique à l'étude de microtubules assemblés in vitro*, Ph.D. Thesis, Université Joseph Fourier, Grenoble, France (cited ibid)
165. Chrétien, D., Fuller, D., and Karsenti, E. (1995), *Structure of Growing Microtubule Ends: two-Dimensional Sheets Close Into Tubes at Variable Rates*, J. Cell. Biol., Vol 129, No.5, June 1995, pp 1311-1328
166. Jánosi, I.M, Chrétien, D., Flyvbjerg, H. (1998), *Modeling elastic properties of microtubule tips and walls*, Eur Biophys J., Vol 27, pp 501-513
167. Thomke, S., Holzner, M. and Gholami, T. (1999) *The Crash in the Machine*, Sci. Am. Vol 280 No 3. pp 72-77
168. Moin, P. and Kim, J., (1997), *Tackling Turbulence with Supercomputers*, Sci.Am., Vol 276, No. 1., pp 46 - 52
169. Author's and colleague's experiences at the Bureau of Meteorological Research Centre, Melbourne, Australia, 1988
170. An interesting review of such work is Rawlings, C.J. and Fox, J.P., (1994) *Artificial intelligence in molecular biology: a review and assessment*, Phil. Trans. R. Soc. Lond., Vol 344, pp 353-363
171. Qian, N., & Sejnowski, T.J., (1988), *Predicting the secondary structure of globular proteins using neural network models*, J. Molec. Biol. Vol 202, pp 865-884 (cited Rawlings, C.J. op. cit.)
172. Brunak, S., Engelbrecht, J & Knudsen, S., (1991) *Neural network detects errors in the assignment of mRNA splice sites*. Nucl. Acids Res. Vol 18 pp 4797-4801 (cited Rawlings, C.J. op. cit.)
173. E.g. Clark, D.A., Rawlings, C.J., Barton, G.J & Archer, I., (1990) *Knowledge-based orchestration of protein sequence analysis and knowledge acquisition protein structure prediction*. Proceedings: AAAI Spring Symposium 1990, pp 28-32 (cited Rawling C.J. op. cit.)

174. Many fundamental elements of economics are still hotly debated, e.g. Dowe, L.D & Korb, K.B.,(1996) *Conceptual Difficulties with the Efficient Market Hypothesis: Towards a Naturalized Economics*, Proceedings: ISIS (Information, Statistics and Induction in Science) 1996, pp 212-223
175. Mandelbrot, B.B. (1999) *A Multifractal Walk down Wall Street*, Sci. Am.Vol 280, No. 2., pp50 - 53
176. E.g. Martin, R.S. (1993) op. cit.
177. e.g. Flyvbjerg, H., et al. (1996) op. cit.
178. Schwartz, R. et al. (Dec. 1998) op. cit.
179. Civelekoglu, G., and Edelstein-Keshet, L. (1994) *Modelling the Dynamics of F-actin in the cell*, Bul. Math. Biol., Vol 56, No. 4., pp 587-616
180. Allen, M.P. and Tildesley, D., (1986) *Computer simulation of liquids*, Clarendon Press Oxford (cited Atkins op.cit.)
181. Gerstein, M. and Levitt, M., (1998) *Simulating Water and the Molecules of Life*, Sci. Am.Vol 279, No. 5, pp 74 - 79
182. A good review article is Merz, K. M. Jr., (1997) *Molecular dynamics simulations of lipid bilayers*, Curr Opin Struc Bio., Vol 7, No 4., pp 511-517
183. E.g. Hinton, E.G., Plaut, D.C. and Shallice, T (1993) *Simulating Brain Damage*, Sci. Am. Vol 269, No. 4., pp 58-65
184. Berger, B.,(1994) et al.
185. Amos, L.A. and Amos, W.B. (1991) *Molecules of the Cytoskeleton*, Macmillan, London, p 130.
186. Nicolaou, K.C., Rodney, K.G. and Potier, P. (1996) *Taxoids: New Weapons against Cancer*, Sci. Am. Vol 274, No. 6, pp 84-88
187. Amos & Amos op. cit.
188. Hammeroff, S. and Watt, R. (1982) *Information processing in microtubules*, J. Theor. Biol, Vol 98, 549-562
189. This is a well known lemma; e.g. Bhattacharyya, G.K. & Johnson, R.A, (1977) *Statistical Concepts and Methods*, John Wiley & Sons, pp 202