ARC CENTRE FOR
COMPLEX SYSTEMS

Technical Report

ACCS-TR-04-03

Modelling Globular Cell Colony Growth

David Woolford

February 3, 2004

ARC Centre for Complex Systems
School of ITEE, The University of Queensland
St Lucia Qld 4072 Australia
T +61 7 3365 1003
F +61 7 3365 1533
E admin@accs.edu.au
W www.accs.edu.au

Modelling globular cell colony growth

David Woolford

February 3, 2004

Contents

Chapter 1

Introduction

This document outlines the methods used to model globular cell colony growth in a 3D environment using OpenGL. The end result is a program, executable on Linux based systems, that facilitates user interaction and animates cell colony growth starting with a solitary cell. This work was originally motivated by the desire to model the growth pattern of stem cell nerual spheres, which start as a single cell and grow to a sphere containing around 16000 (2^{14}) cells.

Cell replication occurs via a fairly straightforward process known as *mitotic division* [3]. This process can be broken into two basic stages, growth and division. The parent cell first grows to approximately twice its original volume. This is followed by division whereby two identical cells are created, each occupying half the volume occupied by the swollen parent cell. Cell replication involves duplicating the genetic material within the cell and distributing one copy to each of the daughter cells.

The cell cycle then repeats iteslf causing the cell count to increase exponentially. The simulation is able to successfully animate the growth of a cell colony containing 2^{14} cells on a sufficiently powerful platform.

This technical paper aims to give the reader an overview of what the simulation achieved and how this was done. The first section gives a description of the simulation including basic concepts and features of the simulation. The second section outlines the theory, mathematics and philosophy behind different components of the simulation. Finally the last secion presents a discussion and critique of the model and outlines future directions.

1.1 Basic Concepts of the Model

The process of cell replication and division, referred to as the cell cycle, can be broken into two main stages, (1) cell growth and (2) cell division. The model starts with a solitary cell which grows to twice its original volume. Once expanded to the correct volume, the cell splits producing two cells.

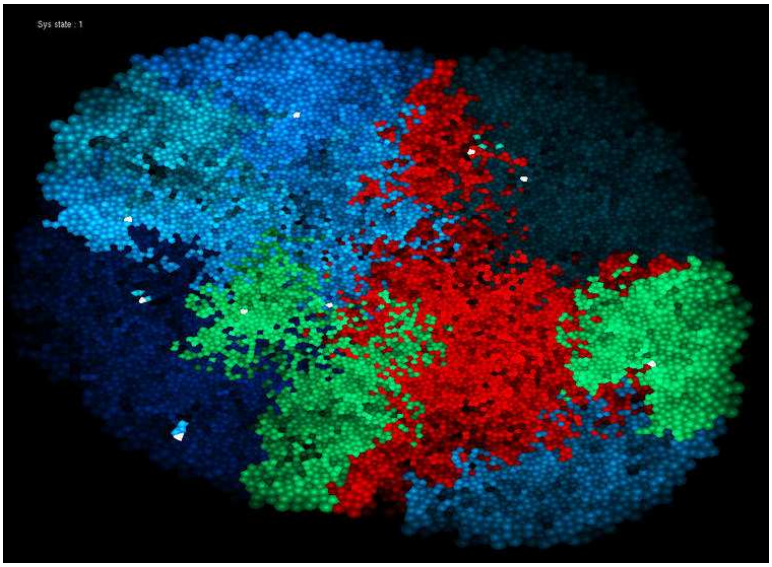

Figure 1.1:
The simulation produces circular like 2D colonies with different strains (colours) that are shaded according to their generation with respect to the first cell

The two daughter cells then undergo the same growth and division process and the colony continues to grow in a cyclical fashion. Examples of the final product of the simulation are shown in **Figures 1.1 and 1.2**

It is important to stress that every occupant in the cell colony is undergoing the same process. In the most simple terms this means that each sphere in the simulation has the same radius at all times, and that there is one global process occuring which switches from growth to division (with intermittent alignment stages), and this is discussed more in **Section 2.5**

The growth animation needs a method that keeps the cells together whilst discouraging overlap. This is achieved using forces. Overlapping cells repulse whilst cells within a certain distance and not overlapping attract each other. The radius of maximum influence is constrained to simplify calculations. These force based interactions can be formulated using Newton's second law and are described in more detail in (**Section 2.2**).

Once cells have grown to twice their original volume cell division can begin. The cell division stage can be described as three main tasks:

- Choose a splitting direction in 3D space.
- Simulate two cells (spheres) moving apart along this trajectory, starting in the same position.
- While the cells move apart their radius is returned to the initial radius.

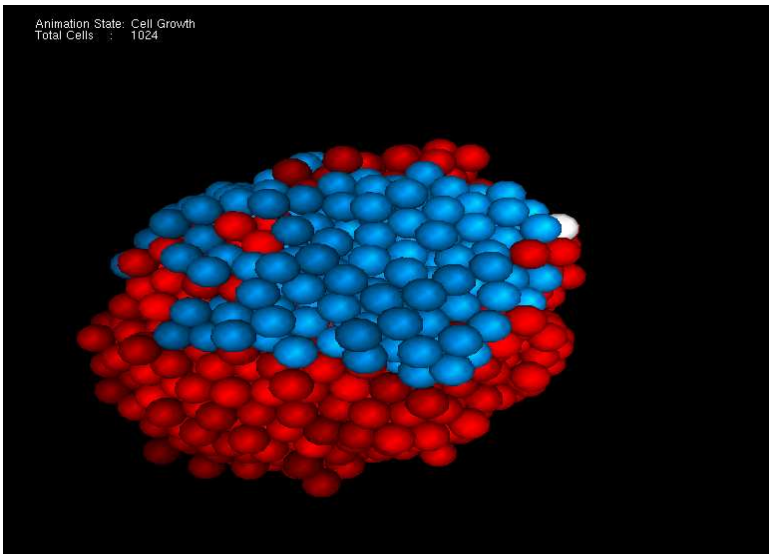

Figure 1.2:
The 3D simulations produces sphere like objects. This particular simulation has had a blue strain added.

The first task is to determine a splitting direction. Ideally we want the dividing cells to end up in relatively close and dense configurations. In order to do this we use the idea of the optimal sphere packing problem whereby a packed colony of spheres all have a 60 degree orientation to their nearest neighbours. Thus in the cell division animation the chosen splitting direction should point 60 degrees to either side of a nominated neighbour cell. Once the splitting direction is determined the two daughter cells move apart so that their edges finally touch at what was the centre of the cell that spawned them. During the splitting stage the force based interactions are abandoned. The splitting stage is discussed in more detail in **Section 2.3**

In the implementation the splitting stage is followed by an overlap alignment stage whereby the total amount of cell overlap is reduced. This ensures that the ensuing growth stage will proceed smoothly, as cells with relatively large overlap can experience a repulsive force that literally expels them from contact with the colony. Infact the the simulation utilises several intermittent stages that promote smooth animation, and these are discussed in **Section 2.5**

1.2 Characteristics of the Model

One aspect of the simulation is that it is possible to do either 2D or 3D simulations. Doing a 2D simulation is similar to looking down on the cross-section of a growing colony of cells, as in analysis via microscopic techniques. The 3D counterpart option generates a roughly spherical ball of cells.

An ascpet of note in the 3D simulation is that there is a random element associated with choosing the splitting angle: a neareast neighbour is chosen and the splitting vector is determined at a 60 degree inclination to the separation vector. This splitting vector can be rotated arbitrarily about the separation vector whilst still preserving the 60 degree offset (recall the optimal sphere packing problem). The simulation employs this random rotation about the splitting vector if 3D animation is enabled.

The 2D simulation quickly assumes a circular shape, where as the 3D simulation requires a few extra cell cycles before the colony starts to clump into a spherical shape. Currently the simulation will produce a fairly good rendition of a cell colony that has witnessed 11-13 cell cycles (2048, 4096, 8192 cells), and an especially powerful system willhandle 16384 cells (**Figure 1.3**). Beyond 2^{14} cell cycles, at the moment, is largely impractical especially in the 3D case. The extra dimension introduces a greater work load to the CPU as an extra system of equations is introduced in the z-direction. The

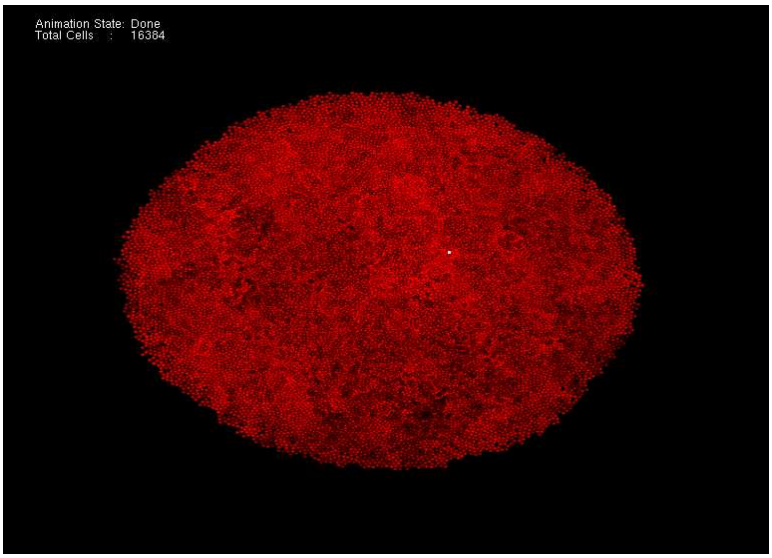

Figure 1.3:
The simulation will currently handle up to 16384 cells in the colony, but requires a powerful workstation to do so effectively.

simulation has a start-up screen that displays the four settings which are user

modifiable, shown in **Figure 1.4**. The start-up screen also includes a simple animation so that the user knows where the current field of view is centred in the virtual universe. The user interacts with the simulation primarily through the mouse. A right click brings up the menu which displays various options. At any stage during the simulation the user can return to the start-up screen and restart the simulation.

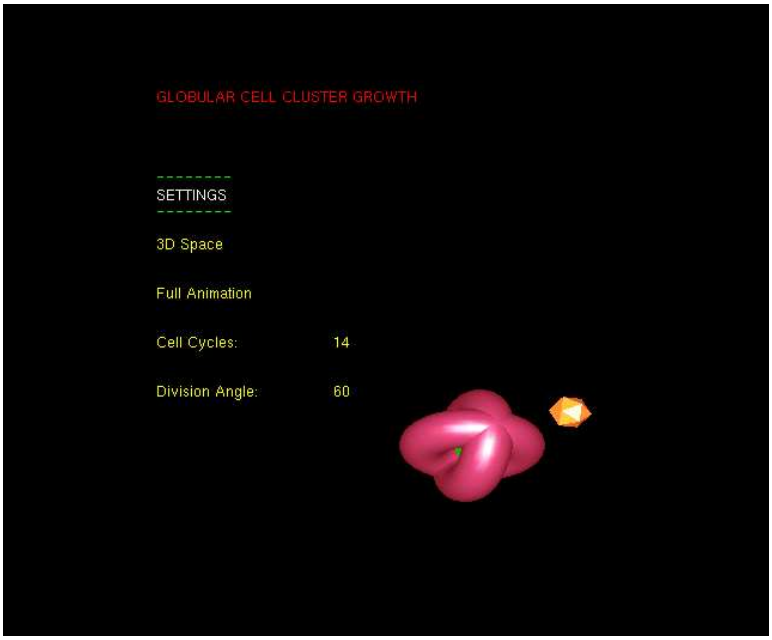

Figure 1.4:
The initial configuration of the colony is done in a startup screen

The user is able to toggle the animation and so reduce the drawing load from the processor. This equates to a small saving in rendering time, but the actual time to get to the final product can still be quite large (about 10 minutes for a 3D cell colony doing 14 cell cycles). The user can also vary the number of cell cycles to be performed and the splitting angle to be chosen (with respect to a near neighbour). The total number of cell cycles and the splitting angle must be entered from a terminal and this is in part due to glut's limited capabilities. The remaining two options are boolean in nature and are simply toggled with the mouse. Two and three dimensional animation are also toggled in this screen (mentioned previously).

The cells have a default colour scheme called the *generational colouring* scheme. Firstly cell division is conceptualised as the process by which a parent reproduces itself and produces a daughter cell. In turn the daughter cell becomes a parent and produces a daughter cell of its own. In other words, during the division stage a cell *retains* its identity and produces an

offspring one generation older. Generational colour works by colour younger generation cells a light shade and older generation cells a dark shade. The root cell of the colony is and always stays white. Daughters of the root cell are always the brightest red and the colouring becomes darker as you descend the generational tree. This is shown in Figure **??**. This feature can be used to evaluate the distribution of different cells in the colony. The simulation also allows the user to insert a random new strain which starts with its own root cell (chosen randomly) and its own strain colour (usually blue or green). This different straing also employs generational colouring.

This concludes the introduction. The next section is on mathematics, theory and philoshophy of the model.

Chapter 2

Modelling and Implementation Notes

2.1 Introduction

This section is meant to cover main concepts that are employed in the simulation. It is intended that this section provide sufficient information for the reader to begin development on their own simulation using the theoretical and conceptual tools presented. The first subsection describes the maths behind the force interactions and details how to create the linear system of equations in any direction. The second subsection discusses the choice of splitting angle and what adaptations were made in order to ensure a smooth animation. Following this is a presentation of a technique for finding nearest neighbours efficiently that involves creating a dynamic spatial coordinate data structure. Finally the fourth section outlines the five animation stages employed in the simulation and gives reasons for this particular approach.

2.2 Force-Based Interactions

The force-based interaction employed in the simulation is based on **??**, but is extended from two to three-dimensions for our model. We define the force F exerted on cell i by its n neighbours based on Newton's second law:

$$\frac{d^2u}{dt^2} + c\frac{du}{dt} = \sum_{j=1}^{n} F_{ij} \tag{2.1}$$

where u is the cell's displacement, m the is the cells' mass and c the damping constant. In practice we vary m and c so that the animation is smooth. The force F_{ij} is proportional to the degree of edge overlap or separation of cells i and j:

$$F_{ij} = k_{ij}O_{ij}$$

If cells i and j have radii r_i and r_j and are located at (x_i, y_i, z_i) and (x_j, y_j, z_j) respectively then:

$$k_{ij} = \frac{1}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}}$$

$$O_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} - (r_j + r_i)$$

For an qualitative analysis of this force relationship see **Appendix A**. The force existing between two given cells can be decomposed into each of the three directions. Using basic algebra it can be shown that the force in the x-direction is as follows:

$$F_{ij(x)} = k_{ij}((x_j - x_i) - k_{ij}(r_j + r_i)(x_j - x_i))$$

And that force in each direction is essentially identical to this form. The forces exerted on each cell can be collected and arranged into a linear system of equations and solved for the displacement.

2.2.1 Constructing the linear system of equations

The coordinates of cell i at time t are its original coordinates $(x_i^{t-1}, y_i^{t-1}, z_i^{t-1})$ plus the displacement induced at the current time step via force interaction (u_{xi}, u_{yi}). Thus we can re-write the force component in the x-direction as follows:

$$F_{ij(x)} = k_1((x_j^{t-1} - x_i^{t-1}) + (u_{xj} - u_{xi})) - k_2(r_j + r_i)$$

Where

$$k_1 = \frac{1}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}}$$

$$k_2 = \frac{(x_j - x_i)}{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

If we assume that the displacement in the current direction is neglible then we can assume k1 and k2 are vary only negligibly with the incurred displacement at the current time step. Thus for the sake of simplicity we say that $x_i^{t-1} \approx x_i$, $y_i^{t-1} \approx y_i$ etc. and then k_1 and k_2 can be approximated using the previous positions of the cells. These constants need to be calculated for each individual cell interaction at each time step. The last thing required to construct the linear system of equations is numerical approximations of the first and second derivates of displacement. This can be done using difference equations in both directions. The following equations are the derivative approximations in the x-direction:

$$\frac{d^2 u_{i_x,t}}{dt^2} \approx \frac{u_{i_x,t} - 2u_{i_x,t-\Delta t} + u_{i_x,t-2\Delta t}}{\Delta t^2}$$

$$\frac{du_{i_x,t}}{dt} \approx \frac{u_{i_x,t} - u_{i_x,t-\Delta t}}{\Delta t}$$

Gathering this information in terms of equation (2.1):

$$m\left(\frac{u_{i_x,t} - 2u_{i_x,t-\Delta t} + u_{i_x,t-2\Delta t}}{\Delta t^2}\right) + c\left(\frac{u_{i_x,t} - u_{i_x,t-\Delta t}}{\Delta t}\right) = \ldots$$

$$\sum_{j=1}^{n}\Big(k_1((x_j^{t-1} - x_i^{t-1}) + (u_{j_x,t} - u_{i_x,t})) - k_2(r_j + r_i)\Big)$$

And this can be arranged into a linear system of the form $Ax = b$ as follows:

$$\left(\frac{[M]}{\Delta t^2} + \frac{[C]}{\Delta t} + [K1]\right)\{u_{x,t}\} = \ldots \tag{2.2}$$

$$[K1]\{x^{t-1}\} - [K2]\{r\} + \frac{[2M]\{u_{i_x,t-\Delta t}\}}{\Delta t^2} + \frac{[C]\{u_{i_x,t-\Delta t}\}}{\Delta t} - \frac{[M]\{u_{i_x,t-2\Delta t}\}}{\Delta t^2}$$

Which can be solved for the unkown displacement vector $\{u_{x,t}\}$ via familiar $Ax = b$ solution methods. In this case the matrix A is diagonally dominant and symmetric making a Conjugate Gradients based approach applicable.

2.2.2 Restricting the field of influence

The field of influence can be reduced so that only cells within a certain distance interact with one another. This unfortunately introduces a parameter into the model but also causes the matrix A to be sparse. In the implementation the field of influence was restricted to 2.5 times the radius of the cell, or in other words the separation distance between two cells needs to be less than half a radius. This helps to keep the colony intact yet prevents spring-like behaviour which can result from more complex interactions.

Restricting the field of influence enables the implementation of sparse matrix handling and this has two main advantages: (1) The amount of memory required is less and (2) the number of flops required in a matrix-vector multiplication can be dramatically reduced. These memory and flop savings can be significant in the context an iterative Conjugate Gradients loop that solves a linear system of equations.

Note

The implementation uses the C++ standard template class *vector* and writes only those entries which are nonzero to memory. In addition only references are passed.

2.3 Directing Division

Once the division stage of the animation is initialised force interactions are abandoned. This is party due to the fact that the simulation models the two daughter cells to be created at exactly the same position with exactly the same radius. The two cells (spheres) are moved apart in tandem whilst simultaneously having their radius attenuated. At the beginning of this process, the calculated repulsive force, using the aforementioned techniques, has an infinite magnitude and this presents various problems: An infinite number has no meaning to a computer, the algorithm would fail; even if this was accomodated for the colony would rapidly disperse (explode). In light of this it was decided that the cell division animation would utilise simple linear motion. This decision brings with it questions that need to be answered intelligently.

- How do you choose the splitting angle so that the colony remains in a cluster?
- How do you avoid cell overlap at the end of the splitting stage?

Each of these questions will now be addressed as briefly as possible.

2.3.1 Choosing the Splitting Angle

The splitting angle can be determined by finding a neighbouring cell, determing the vector between the two cells, and splitting at +/- 60 degrees to the direction of this vector. This indeed is the solution to the optimimum sphere packing problem. See below in **Figure 2.1**, which depicts cells in the splitting stage of the animation. The blue lines indicate their chosen splitting angle. A more careful analysis will reveal that this direction is at 60 degrees to some neighbouring cell.

It is, however, possible for the user to specify their own splitting angle in the start-up screen. For instance a splitting angle of 90 degrees can be entered and the resulting shape can be studied. In practice the splitting angle makes little difference. The force interactions usually compel the colony to cluster into circular/spherical shapes.

It is important to note that the neighbour with which a cell chooses to split does not need to be touching. In the implementation a cell will split with any cell within 2.5 times the current radius of the cell (from center to center). This is the same condition placed on the force interactions and the significance of this is made more apparent in **Section 2.4**.

2.3.2 Reducing the Amount of Overlap

There are stochastic elements associated with determining the split direction including:

- Choosing which neighbour with which to split.

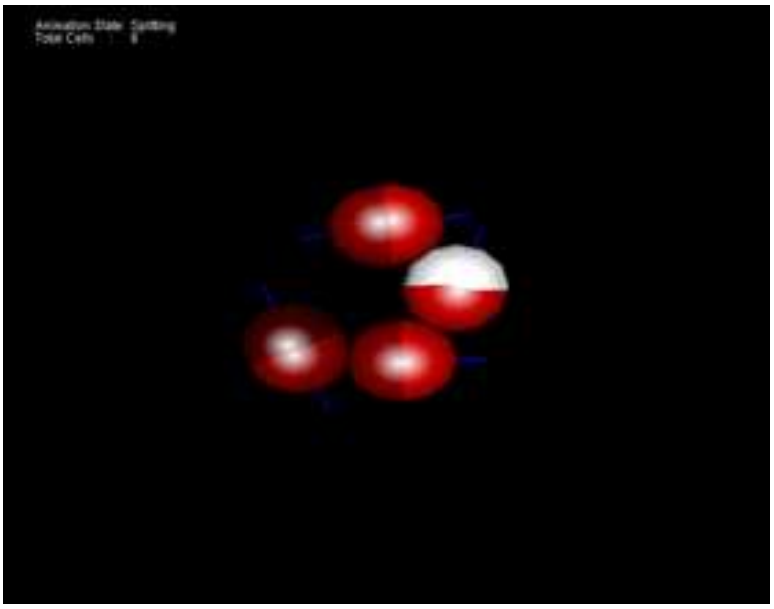

Figure 2.1:
The directin of the division is chosen inentionally, and can be varied by the user

- Choosing which side of the neighbour to aim for (2D).
- Choosing which orientation to take with the chosen neighbour (3D).

Each of these decisions is done in a purely (pseudo) random fashion. If the cells are modelled to move linearly in the split direction than cell overlap will most probably occur. To counteract this, each cell's neighbours (stored in a neighbour list see (**2.4**)) are queried for overlap at each time step during the splitting stage. If overlap is detected the neighbouring cell and its sister (fellow splitting) cell are displaced an amount proportional to the degree of overlap and in the direction of the vector separating the candidate cell from its neighbour.

At the end of the splitting stage it is often the case that there is still overlap. The simulation thus includes and overlap realignment stage at the end of the splitting stage whereby overlap is corrected but the radius of the cells in the colony remains fixed. More discussion on this intermittent realignment stage is presented in **Section 2.4**.

2.4 The Cell Storage Grid (CSG)

At each time step in the simulation, in each animation stage, it is necessary that each cell have a list of the neighbouring cells within 2.5 times the current cell radius. The naive and brute force method is $O(n^2)$. To avoid

this potential bottleneck the simulation employs an alternative scheme based on discretisation of space and coordinate storage. A lattice of cubic volumes is constructed that completely encases the cell colony. The width, height and depth of the cubic volumes is always 2.5 times the current cell radius. This ensures that cells within 2.5 times the radius are in adjacent cells inthe CSG. Instead of querying every cell in the colony, a cell only need check the adjacent cubes in the CSG and this is the main advantage delivered by implementing the CSG.

At each times step a CSG is created (**Figure 2.2**), its dimensions are determined and each cell is stored in its correct cube. Each of this tasks require one pass ($O(n)$) over the colony structure storing the cell data. Once this is done, the CSG is traversed in such a manner as to update the neighbour lists of the cells in each cubic volume. This process is streamlined so to reduce the load on processor. The CSG is a fundamental component of the simulation and it can be drawn to the screen (dynamically) in the mouse-menu. It must be stressed that the CSG is created and updated at every time step.

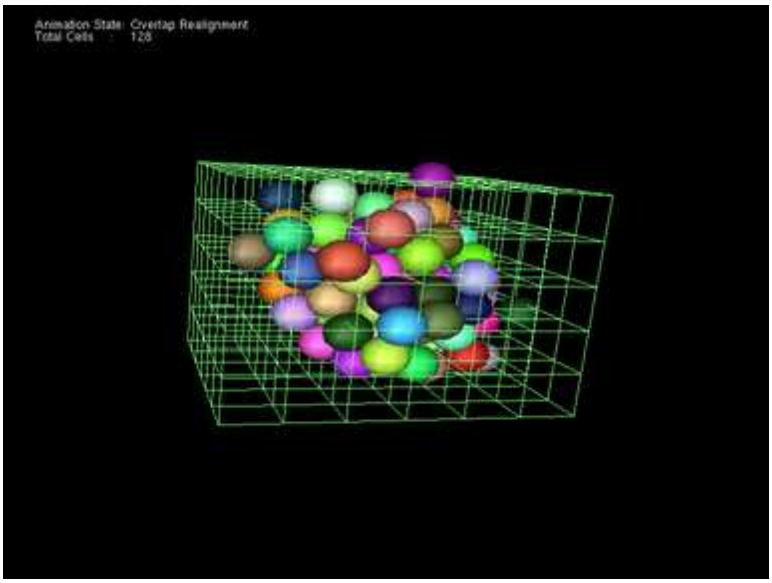

Figure 2.2:
By discretising space into 3D cubic volumes we can more quickly establish which cells are neighbours

2.5 A Five Step Animation

The simulation is separated into five simulation stages that are traversed in a cyclical manner, described in the table below. Each simulation stage is given a set time that is inflated as the number of cells in the colony increases. This is because, as the colony begins to hit the thousands, the time taken to perform one update of the colony (in any stage) increases dramatically and some compensation needs to be made for this.

Step	Process
1 Growth	Cells grow. Force interactions are Taking place, the colony expands to approximately twice its original volume.
2 Align 1	Cells remain same size while force interactions reduce overlap and compress the colony, in preparation for splitting
3 Split	Cells divide and split after choosing a splitting trajectory. Force interactions are abandoned in favour of a simple linear motion model. Cell overlap results in movement. Radius is attenuated.
4 Align 2	Cells realign using simple linear movement which is proportional to the degree overlap. Radius is constant, the colony appears to shuffle.
5 Align 1	Cell colony is subjected to multiple passes of the force interaction algorithm to realign and compress the colony in preparation for cell growth. Radius is constant.

The simplest animation cycle to implement would be to grow the cells and then split them and then to immediately grow them again. But there are problems that result from this approach. The colony tends to lose its circular/spherical shape if the cells are not given time to regroup at the end of the different stages. The growth stage tends to push cells away from each other and so at the end of this stage the cell occupancy of space is less dense than desired. On the other hand the splitting stage often produces a colony of cells with too much overlap which can cause cells to be rocketed away from the colony. To counteract this the simulation includes a force realignment stage (2 in the table above) after the growth stage (1), to encourage autonomous occupation of space whilst balancing density. After the splitting stage (3) the colony is subjected to a basic alignment stage(4) that detects overlap and causes repulsive movement. This stage (4) usually produces a lot a blank space between the cells and so it is following by force realignment stage (5) to compress the colony and minimise empty space. At this point the colony is ready to grow again (1).

Of note is that the stages (2) and (5) are virtually identical to the growth stage (1), except that the radius is held constant. This entails the need to

solve at least two system of equations (2D) at each time step in stages (2) and (5), and so these stages can be expensive for large system sizes.

2.6 Conclusive Remarks

It is necessary to have well grounded methods for achieving the desired ends. This section has outlined the bulk of the reasoning and philosophy behind the cell colony simulation. Force based interactions are used to keep the colony together and simple linear motion is used to conduct cell division. Various tools are implemented to improve the efficiency of the simulation, including the dynamic Cell Storage Grid which reduces computational costs in determing the neighbour cells of a given cell in the colony. Finally it was necessary to implement intermittent realignment stages so as to ensure that the colony stayed together and that erratic movement was minimised.

Chapter 3

Discussion and Future Direction

3.1 What was achieved

This simulation originally set out to model cell colony growth and to provide a tool for studying distributions of different cell types therein. In particulay the idea of the neurosphere gave impetus to this work, and the first goal was to get a spherical colony of cells animated and growing on screen. In this respect the simulation was successful. It was also envisaged that the code should be versatile and that the splitting angles and number of cell cycles should be user modifiable so that different cell shapes could be observed and the way that these vary could be studied and used to further develop the model.

3.2 Problems and Limitations

The model has associated several parameters which can be varied inside the code (see **Appendix B**). Most parameters are associated with the force based model, which needs the cells to have some nominated mass, a velocity damping constant and a pseudo time step. Varying these parameters can yield result that differ significantly. For instance if the damping constant is too small, the spring like interaction of the cells can explode and the colony can disperse. On the other hand if the parameters of the colony have not been set well the colony will oscillate in what looks like a random fashion, similar to 'wobbly' jelly. For these reasosn the parameters need to be played with and their optimum setting is determined by trial and error.

One of the main problems of the simulation is that, in the early stages, the cell colony can assume a random looking configuration that appears to have some branching objects and also holes can be apparent. This is in part due to the division procedure employed by the animation. As opposed

to splitting in a direction that preserves the circular/spherical shape of the colony, the splitting direction is chosen so that the final position of the daughter cell is close to some neighbour. The circular/spherical shape of the colony is assumed once the cell count escalates sufficiently, by which time space is beginning to fill up and the random elements of the cells positioning begin the cancel each other out. This problem could be counterracted by introducing an energy function that would induce movement and that would be minimised when the colony assumes the desired shape.

The code also runs fairly slowly when there is more than 2000 cells in the colony. This is due to the codes reliance on a linear system of equations solver, a (efficient) nieghbour list generation, collision detection and the fact that almost every cell is moving at every time step. The code started out using a C philosophy and would probably benefit from a complete conversion to a C++ coding paradigm.

3.3 Future Direction

The simulation has demonstrated that it is possible to model spherical cell colony growth, and this hints enticingly at different shapes such as filaments (cylinders) and any other number of more complex shapes. If the modelling process were to be developed in more detail the production of different shapes would be possible, but would probably rely on the introduction of energy functions that are minimised when the cell colony assumes the correct shape. Iif simulation of different shapes could be achieved this could be used as a base for building more complex objects comprised of base units of different size and shape. Perhaps the simulation of the generation of a complex organism could be achieved, starting with a solitary cell.

As well as achieving globular cell colony growth, the code also provides a platform for future development. For instance the nieghbour list provides the ground work for implementing models similar to those that use Cellular Automata (CA), as in complex systems. This would facilitate cell differentiation models that need access to a list of neighbours and their types. A differentiation model could inturn be used to study the distribution of different types of cells in the colony which could be used to further understand the dynamics of cell clusters such as neural spheres.

3.4 Acknowledgments

I would like to thank Kevin Burrage who gave me the oppurtunity to carry out this work, and who along with Nick Hamilton provided fundamental guidance and discussion that helped to shape this body of work. I thank Francis Clark who supplied important code and last but not least I thank Geoff Ericksson for programming tuition.

Appendix A

Force Interactions

As an analysis of this system, let $(r_j + r_i) = \alpha$ and:

$$x = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} - (r_j + r_i)$$

Then: “

$$k_{i,j} = \frac{1}{x + \alpha}$$

$$O_{i,j} = x$$

Thus yielding a simplified interpretation of F_{ij}

$$F_{ij} = \frac{x}{x + \alpha}$$

Which displays the characteristics exhibited in **Figure A.1**, assuming $\alpha = 1$. Note that as α is approached from the right the function asymptotes toward $y = \infty$, and as $x \longrightarrow \infty$ the function asymptotes toward $y = 1$. This function ensures that overlapping cells are repulsed whilst separated cells are attracted. One questionable assumption is that the greatest force of attraction exists when the cells are infinitely separated. In practice however this assumption is relaxed somewhat, as the field of influence is usually restricted to a small local domain.

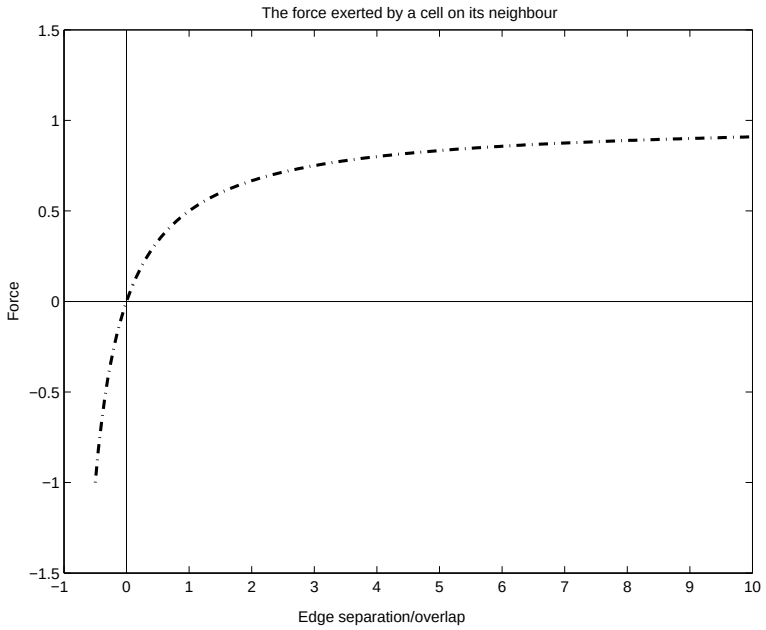

Figure A.1:
This figure demonstrates the nature of the incurred force of two cells as a function of edge overlap/separation.

Appendix B

Parameters of the Simulation

The following table lists the parameters of the model than can be varied to produce different effects in the simulation. These parameters are found in cgccgCellColony.h.

Parameter	Effect
PROC_VEL_GROW	The time allowed for the growth stage (ms)
PROC_VEL_SPLI	The time allowed for the splitting stage (ms)
PROC_VEL_ALIGN	The time allowed for the basic realignment stage stage (ms), after the splitting stage
PROC_VEL_SOFT	The time allowed for the force realignment stage (ms) (both of them are allowed the same time)
TOL	The tolerance of the Conjugate Gradient linear system solver (default 0.0001)

The following table lists the parameters stored in the *Params* struct which, when varied, can significantly alter the behaviour of the simulation.The *Params* struct is defined in gccgCellColony.h but its initial setting are set in gccgCellColony.cpp.

Parameter	Effect
iterations	The number of iterations assigned to each stage when animation is disabled
mCellMass	The modelled mass of each cell
mCellAlpha	The velocity damping constant
mDt	The pseudo time step employed in the linear system solver
$r_i nit$	The initial radius of each cell in the colony (default = 1.0)

Finally the parameter MAX_DIST_INFLUENCE found in the file *cggcCoordGrid.cpp* determines the radius of influence when constructing the CSG and also when creating neighbour lists. It's default value is 2.5

Bibliography

[1] R Smallwood, M Holcombe, D Walkerm D R Hose, S Wood, S MacNeil, J Southgate. *Modelling emergent order: from individual cell to tissue.* Unpublished document - 1/12/03

[2] G. H. Golub, C. F. Van Loan, "Matrix Computations," Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[3] Alberts et al. Molecular Biology of the Cell. 1994. Published by Garland Publishing, New York.