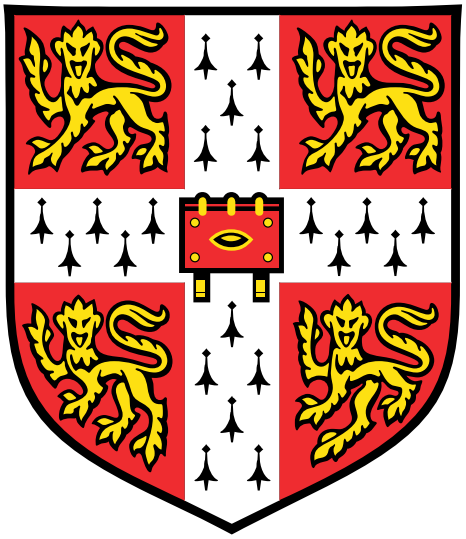
Modelling Physical Mechanisms Driving Tissue Self-Organisation in the Early Mammalian Embryo



Christopher Keith Revell

Department of Physics
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Gonville and Caius College May 2018

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Christopher Keith Revell
May 2018

Acknowledgements

Completion of this thesis would not have been possible without a small group of close supporters. Most notably I am eternally grateful to Evelyn Boettcher for having more faith in me than I could ever have in myself, and for always making me feel better when things were hard. Countless walks, dinners of curry noodles and halloumi, coffees, and hugs made more difference than I can fully express. I am also thankful for the constant digital presence of Katie Davidson who, although often thousands of miles away, was always willing to listen. I must also acknowledge the generosity and patience of my parents and grandfather, without whom this project would never have been possible.

I have greatly appreciated the patience and input of my supervisors Raphael Blumenfeld and Kevin Chalut, who managed to remain dedicated to the completion of this project despite everything else happening in their lives. All experimental results from within the group cited within this thesis were produced by Ayaka Yanagida and Sarra Achouri, and I am particularly indebted to Ayaka for providing feedback on this manuscript.

The Santa Fe Institute made a huge contribution to my progress by reinvigorating my belief in both science and myself during their Complex Systems Summer School, and it was here that I met Marius Somveille, to whom I am extremely grateful for working with me on the project, begun at the SFI, that resulted in my first publication.

Finally, I am grateful to all those at Gonville and Caius College, whether it be my tutor Jonathan Evans, the porters, or many others, for providing the environment in which to survive 9 years and 3 degrees at Cambridge, and to Caius Boat Club, for providing an escape from those degrees, and the platform for my rowing endeavours.

Abstract

In the mammalian embryo, between 3.5 and 4.5 days after fertilisation, the cells of the inner cell mass evolve from a uniform aggregate to an ordered structure with two distinct tissue layers - the primitive endoderm and epiblast. It was originally assumed that cells differentiated to form these layers in situ, but more recent evidence suggests that both cell types arise scattered throughout the inner cell mass, and it is thus proposed that the tissue layers self-organise by physical mechanisms after the specification of the two cell types. We have developed a computational model based on the subcellular element method to combine theoretical and experimental work and elucidate the mechanisms that drive this self-organisation. The subcellular element method models each cell as a cloud of infinitesimal points that interact with their nearest neighbours by local forces. Our method is built around the introduction of a tensile cortex in each cell by identifying boundary elements and using a Delaunay triangulation to define a network of forces that act within this boundary layer. Once the cortex has been established, we allow the tension in the network to vary locally at interfaces, modelling the exclusion of myosin at cell-cell interfaces and consequent reduction in tension. The model is validated by testing the simulated interfaces in cell doublets and comparing to experimental data and previous theoretical work. Furthermore, we introduce dynamic tension to model blebbing in primitive endoderm cells. We investigate the effects of cortical tension, differential interfacial tension, and blebbing on interfaces, rearrangement, and sorting. By establishing quantitative measurements of sorting we produce phase diagrams of sorting magnitude given system parameters and find that robust sorting in a 30 cell aggregate is best achieved by a combination of differential interfacial tension and blebbing.

Table of contents

List of figures

Chapter 1

Introduction to Self-Organisation, Embryogenesis, and Cell Sorting

The application of physics to embryology began over a century ago with Wilhelm Roux, who coined the term "developmental mechanics" [1–3] and attempted to explain development as a physical process. However, despite some interest from the great minds of 20th century physics [4–6], it was not until recent years that physical processes and physical reasoning came to prominence as an important consideration in understanding biological systems, including development. The field of biological physics is now well established and is helping to transform biology into a quantitative, predictive science [7–10].

Embryogenesis is one of the great mysteries in biology and a subject to which this rising field of physical biology has recently begun to turn its attention. The purpose of this project was to apply physical reasoning and computational modelling to shed light on the self-organisation of stem cell tissue layers within the early mammalian embryo. It is proposed that in this process, an initially random distribution of two cell types is produced by stochastic transcriptional noise, and that physical differences in these cells then drive their separation into two distinct tissue layers. We aim to show how this process is governed to a large extent by physical mechanisms.

The background of inner cell mass development will be introduced in a manner that is accessible to physicists. With this background understanding, a variety of theories that have previously been proposed to lead to cell sorting by physical mechanisms are reviewed. Theoretical modelling work on the mammalian embryo is limited, but examining past work on modelling cellular systems demonstrates the limitations of most previous approaches, which will, in turn, lead into a discussion of how modelling can be improved and applied to cell sorting in the inner cell mass.

1.1 A Definition of Self-Organisation

The concept of self-organisation arises in a great many different fields [11], from physics and biology [12], to computing, or the social sciences. Self-organisation is typically the result of non-equilibrium systems; biological cells being a classic example of such a system. Self-organisation is a fundamental principle underlying much of biology. It can be seen in systems ranging in scale from the folding of proteins [13], to the dramatic patterns in a murmuration of starlings [14, 15].

Broadly, we can think of self-organisation as the formation of some macroscopic order from disordered components by the action of simple rules defining the interactions between these components. These interactions provide the local communication that can lead to global order. Although we can form such an intuitive understanding of what the phrase means, a precise definition of self-organisation can be difficult, and was addressed in detail in W. Ross Ashby's seminal work "Principles of the Self-Organizing System" [16]. Ashby points out that most "organisations" that a system may produce are bad ones with respect to some purpose, and that in biology we may think of self-organisation as "changing from a bad organisation to a good one". This again hinges on the definition of "good", which may depend on context, but could, for example, include the orientation of a set of cells that allows normal development of an embryo. Thus, for our purposes, self-organisation is the rearrangement of cells from an orientation in which development is not possible to one in which development of the embryo can proceed correctly.

1.2 Blastocyst Formation

We are interested in an example of self-organisation that occurs in the mouse embryo between 3.5 and 4.5 days after fertilisation (E3.5-4.5). The following section describes the development of the embryo up to 3.5 days after fertilisation, specifically the formation of the blastocyst structure. The early development of the mouse embryo has been well studied and the physical mechanisms driving blastocyst formation are well understood [17].

Embryogenesis begins with the fertilisation of an oocyte (egg cell) to form a zygote, which is a single, large cell containing all genetic material necessary to produce a full organism. Soon after fertilisation, the zygote cleaves into two daughter cells. Subsequent division of daughter cells forms an aggregate of 8 cells [18]. These divisions involve proliferation of nuclei and separation of pre-existing cytoplasm - no growth occurs and the total size of the cell aggregate does not change significantly from the size of the zygote but the nucleus/cytoplasm

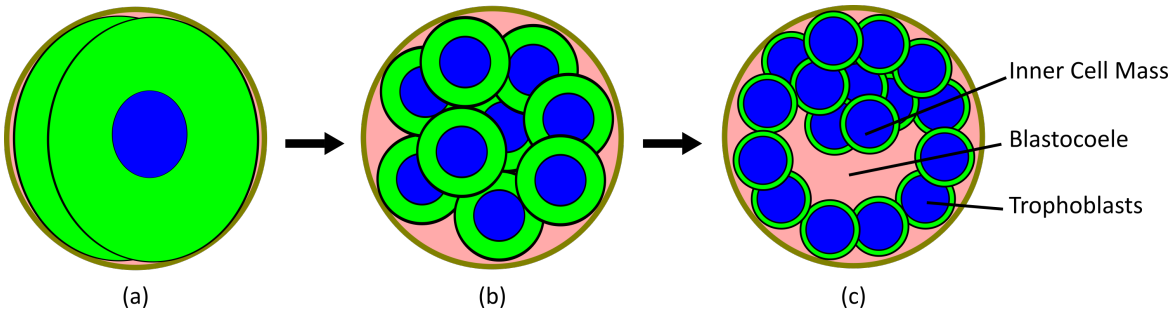


Fig. 1.1 Simplified diagram of embryogenesis from the post cleavage stage to the early blastocyst. a, Post-cleavage 2 cell stage. b, Morula, roughly 8 to 32 cells. c, Early blastocyst, 32-64 cells [20].

ratio increases dramatically. This 8 cell aggregate undergoes a process of compaction to reach a stage known as the morula [Figure 1.1] [19].

The cells in the morula continue to divide until, around 3.5 days after fertilisation, the embryo consists of up to 32 cells and has entered the blastocyst stage [Figure 1.1 and Figure 1.2] [17, 21]. During the process of blastocyst formation, the cells begin to excrete a fluid which forms multiple small fluid-filled cavities [22, 23] within the embryo. Over time, these cavities coalesce to form a larger cavity known as the blastocoele, surrounded by a tissue called the trophectoderm [24], formed from cells called trophoblasts [Figure 1.2]. The membranes of these cells form tight junctions to create a boundary that seals and protects the blastocyst, helping to maintain the blastocoele fluid at a higher pressure than the external environment. If this seal is broken the blastocyst will collapse, indicating the importance of the fluid pressure in maintaining the spherical structure [25]. The trophectoderm is the first "extra-embryonic" tissue to develop. Embryonic tissues at this stage are pluripotent, meaning their cells have the capacity to form any cell type in the foetus, but these extra-embryonic tissues lose that pluripotency in their differentiation to a specific extra-embryonic cell type. The formation of "extra-embryonic" tissues is characteristic of mammalian embryogenesis [26, 17].

Trophectoderm cells are allocated by polarised division along the radial axis of the morula, with the outer daughter cells from such a division forming trophoblasts [27]. Those daughter cells at the inner edge of the division axis do not lose their pluripotency and are allocated to a tissue known as the inner cell mass (ICM) that forms an aggregate at one pole of the blastocyst [Figure 1.2] [28]. It is this inner cell mass that will be of primary interest to us.

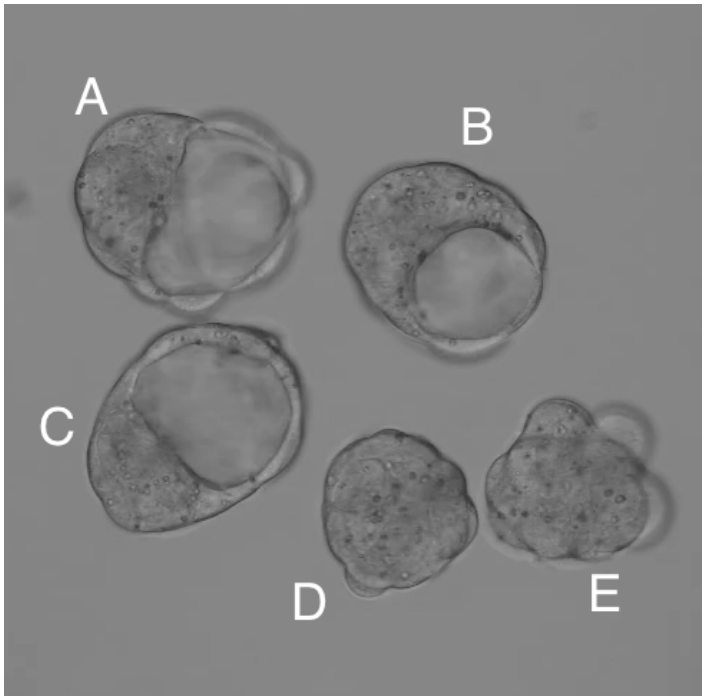


Fig. 1.2 Confocal microscopy image of 5 mouse embryos. Embryos A, B and C are in the blastocyst stage, and the blastocoele, trophectoderm and inner cell mass can clearly be seen. Embryos D and E are in the morula stage. Image courtesy of Berenika Plusa.

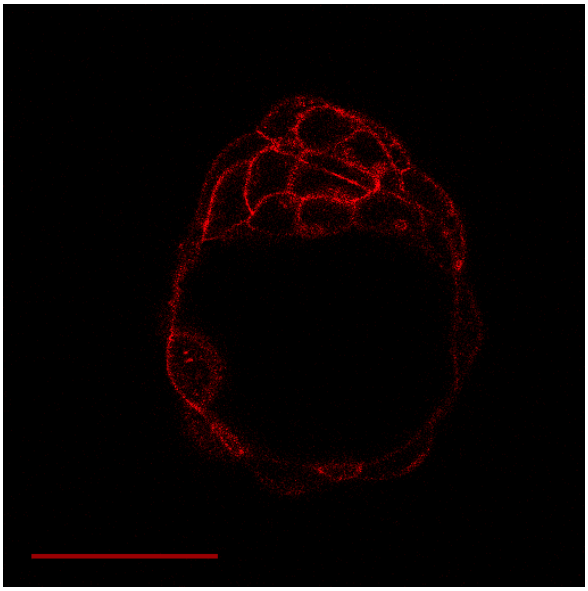
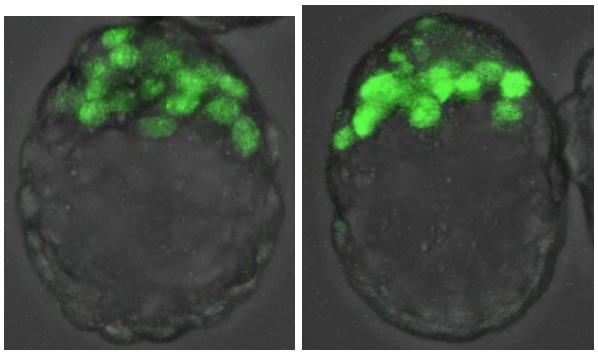


Fig. 1.3 A confocal image of mouse blastocyst with membrane-localised mCherry fluorescent protein. The empty fluid filled blastocoele, and the cells of the inner cell mass are all clearly visible. Scale bar is 50μm. Image produced by Ayaka Yanagida in our group.



(a) (b)

Fig. 1.4 Blastocysts at 3.5 and 4.5 days after fertilisation in which PDGFRα (characteristic of primitive endoderm) is labelled with green fluorescent protein. a, E3.5, with primitive endoderm marker positive cells distributed throughout ICM. b, E4.5, with primitive endoderm marker positive cells localised around the boundary with the blastocoele. Images courtesy of Berenika Plusa.

1.3 Lineage Specification in the Inner Cell Mass

Once the ICM has formed, from about 3.5 days after fertilisation, its constituent cells differentiate into two distinct cell lineages: the epiblast and the primitive endoderm. The epiblast cells are those from which the foetus will develop, and are described as pluripotent because they have the capacity to form any tissue of the final organism. The primitive endoderm is the second extra-embryonic tissue. It forms a protective layer between the epiblast and the blastocoele, and will eventually produce the yolk sac.

Lineage specification within the early embryo is characterised by differential expression of transcription factors [29–31]. The precise function of these transcription factors is not relevant to this project - for us they simply serve as markers for each lineage by virtue of their expression being mutually exclusive. Specifically, epiblast cells are characterised by expression of the pluripotency factor Nanog [32, 33], Sox2, and Oct4 [34], while primitive endoderm cells are characterised by expression of PDGFRα [30], Gata4 [35], Gata6 [36, 37], and Sox17 [38–40]. Most notable among these are Nanog for epiblasts and Gata6 for primitive endoderm. It is also worth noting that the initial fate decisions made by cells in the inner cell mass produce “precursor” cells that have been shown to retain some degree of plasticity, such that if transplanted into another embryo they can switch to the other type [41], lending further credence to the idea of cells moving within some continuous state space between epiblast and primitive endoderm. However, this is also beyond the scope of the current project, and once the blastocyst reaches 4.5 days after fertilisation, these fate decisions are observed to be fully determined.

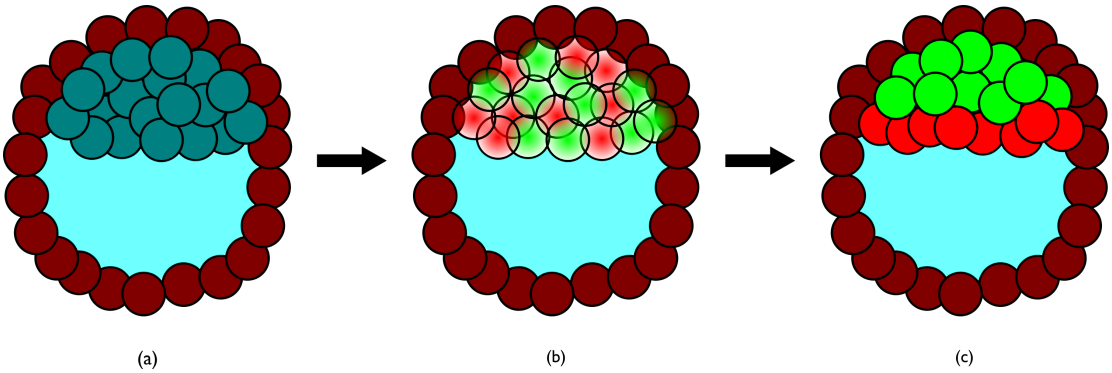


Fig. 1.5 Simplified diagram of fate acquisition and sorting in the inner cell mass.
a, Ground state inner cell mass cells.
b, "Salt and pepper" distribution of epiblast and primitive endoderm precursors.
c, Self-organised epiblast and primitive endoderm layers.

The observed positions of the primitive endoderm and epiblast cells within the late stage ICM [Figure 1.4b], when cell phenotypes are clearly visible, led to an initial assumption that the two lineages differentiated *in situ*, driven by environmental cues such as proximity to the blastocoele or trophectoderm [42]. More recently, it has been found that in the early and mid-stage blastocyst, before observed formation of distinct primitive endoderm and epiblast layers, expression of these transcription factors is distributed throughout the ICM in a mixed but mutually exclusive manner [36] [Figure 1.4a]. This suggests that the process of lineage specification is stochastic [43] and precedes the emergence of physically separated epiblast and primitive endoderm. Thus it is proposed that the initial "salt and pepper" distribution [27] of cells in the inner cell mass eventually evolves into distinct layers by self-organisation [44].

The evolution of the inner cell mass between 3.5 and 4.5 days after fertilisation is shown in Figure 1.5. Figure 1.5a shows a blastocyst containing an aggregate of undifferentiated blastomeres that make up the inner cell mass at E3.5. Figure 1.5b shows how an initial disorganised "salt and pepper" distribution of two cell lineages is formed before the physically segregated tissue layers become evident, as shown in Figure 1.5c. Note that the change from diffuse to solid colours indicates that fate decisions are locked in by E4.5.

It has been proposed that the physical properties of epiblast and primitive endoderm cells differ [45, 46]. We propose that it is these physical differences that lead to self-organisation of the tissue layers by physical processes. This fits with the self-organising framework proposed for earlier stages of embryo development [47]. The questions that remain are: precisely what mechanisms drive this process, and how do we formulate these into a model that will realistically describe the inner cell mass and help further our understanding of embryogenesis? Although earlier processes described in Section 1.2 are well understood, the

precise mechanisms of inner cell mass self-organisation have not been well studied [17], so this project was an initial attempt to formulate a hypothesis for what drives these processes.

It is worth noting that epiblast and primitive endoderm cells are observed to self-organise as an aggregate, without extra-embryonic tissue. In this case, the cells form a spherical aggregate with epiblast cells moving to the centre and primitive endoderm moving to the outside [48]. For the purposes of our simulations we start out modelling this *in vitro* system.

1.4 Physical Mechanisms of Tissue Self-Organisation

The self-organisation of cells into distinct tissue layers can be regarded as analogous to the separation of oil and water [49, 50]. However well oil and water are mixed, they will always eventually separate into two distinct layers. The forces governing this behaviour are the differing strengths of hydrogen bonding for oil and water molecules, and gravity, which breaks the symmetry of the system. The analogy to primitive endoderm and epiblast cells, which are seen to separate into distinct layers from an initial random distribution, is clear, and we will now discuss the analogous forces that could drive the separation of primitive endoderm and epiblast cells.

We identify two factors necessary for cell self-organisation: differing mutual affinities between cells and one or more energetic mechanisms to drive the kinetics.

1.4.1 Sources of Differing Mutual Affinity

When Townes and Holtfreter [51] first demonstrated the spontaneous separation of mixtures of embryonic cells in vitro, they proposed that such behaviour was driven by the cells's differing "affinity" for one another. We now understand that this is a critical component of cell sorting; when cells are able to rearrange, a difference in mutual affinity between two cell types will drive them to self-organise. Two primary mechanisms have been proposed to change the mutual affinity of cells. These are differential adhesion and differential interfacial tension.

Differential Adhesion

Malcolm Steinberg was first to propose that differing affinity could arise from differential adhesion between the surfaces of cells [52–54]. This is now known as the Differential Adhesion Hypothesis (DAH) and is a widely studied mechanism for cell sorting. The idea is simple: that in a random distribution of cells, those with strongest mutual adhesion would

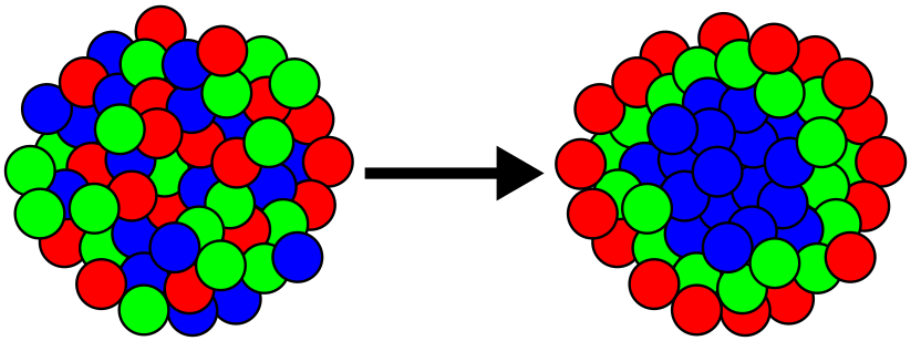


Fig. 1.6 Diagram showing a mixed aggregate of cells of types A (blue), B (green), and C (red), and the resulting sorted aggregate predicted by the differential adhesion hypothesis where the mutual adhesion of type A is greater than that of type B, which is itself greater than that of type C.

draw together and force the other cells with weaker mutual adhesion to the outside of the ball of cells.

Steinberg describes the DAH as the "spontaneous progressions of motile and mutually adhesive cell populations toward configurations of minimal interfacial (adhesive) free energy" [55] in which stronger adhesions will tend to displace weaker ones. He also explains how, with further analogy to oil and water, sorting by differential adhesion should tend to form clusters of more adhesive cells within an aggregate of less adhesive cells, and that these clusters will steadily coalesce and move to the inside of the aggregate [55]. It is possible to speculate that the process should be transitive such that if cell type A is more mutually adhesive than cell type B, and B is more mutually adhesive than C, then tissue comprising cells A, B and C should form with C on the outside, A in the centre and B between the two [Figure 1.6]. Steinberg clarified that proposing this kind of transitivity does not assume an identical physical or chemical means for adhesion between different cells or in different systems, but simply that there is a thermodynamic scale of adhesion intensity whose underlying details are not relevant.

This process can be viewed from another perspective. Cell adhesion leads to an effective surface tension for the tissue as a whole (note that this is not the same as cell surface membrane tension but rather is analogous to the surface tension of a liquid arising from the mutual attraction of its molecules). Lower cell adhesion leads to a lower tissue surface tension and vice versa, so we can rephrase the sorting rules as: the tissue with lowest "surface tension" will be found to envelope a tissue of higher "surface tension" [56, 57]. The DAH model also predicts minimisation of the global surface area of cell aggregates, which is

observed in the tendency of irregularly shaped tissue fragments to steadily take on a more spherical shape [58], as with liquid droplets.

Going beyond this simple explanation, it is possible to be more precise with the mathematics of DAH. If there are two cell types in an aggregate, A and B, the manner of sorting will depend on their relative adhesive strengths. If adhesion between A and B is stronger than between A and A or between B and B ($AB > AA > BB$) then a mixed salt and pepper distribution will be formed. If like cells adhere more strongly than unlike cells ($AA > BB > AB$) then sorting will occur, with the cells with the stronger mutual adhesion (A) collecting on the inside of the aggregate and those with the weaker mutual adhesion (B) moving to the outside. We can also consider cases in which one type of cells may adhere to themselves most strongly, but also to the other type of cell more strongly than that type of cells adhere to themselves ($AA > AB > BB$). In this case if the average of like to like adhesion is greater than the like to unlike adhesion then the cells will sort out, otherwise they will not. In other words, sorting if $(AA + BB)/2 > AB$, no sorting if $(AA + BB)/2 < AB$ [59].

Cell adhesion can be measured by atomic force microscopy. One cell is attached to the probe tip and this cell is then brought into contact with another. The probe is then pulled away and the force applied by each cell onto the other can be measured until the two cells separate [60]. Cell adhesion can also be measured by observing the overall deformation of an aggregate of cells when centrifuged.

Some have questioned the DAH, arguing that the analogy between adhesive cells and liquid molecules, though appealing, is fallacious [61]. Most of the arguments raised are technicalities concerning the precise nature of adhesion between cells compared to attraction between molecules. For example, it is possible that "unsticking" two cells is not precisely the inverse of sticking them together in the way that moving two molecules towards and then away from each other is movement within a conservative potential. Such details may indeed cause cells to behave differently from molecules in a liquid, and these may be factors that need to be considered in the precise formulation of any modelling attempt. However, such details do not necessarily make the general hypothesis that differential adhesion could lead to cell sorting invalid.

Although the differential adhesion hypothesis is a widely established framework for understanding cell sorting [18], it has been shown that differential adhesion is not sufficient to explain *in vitro* sorting of epiblast and primitive endoderm cells [48] since the two cell types adhere to one another equally.

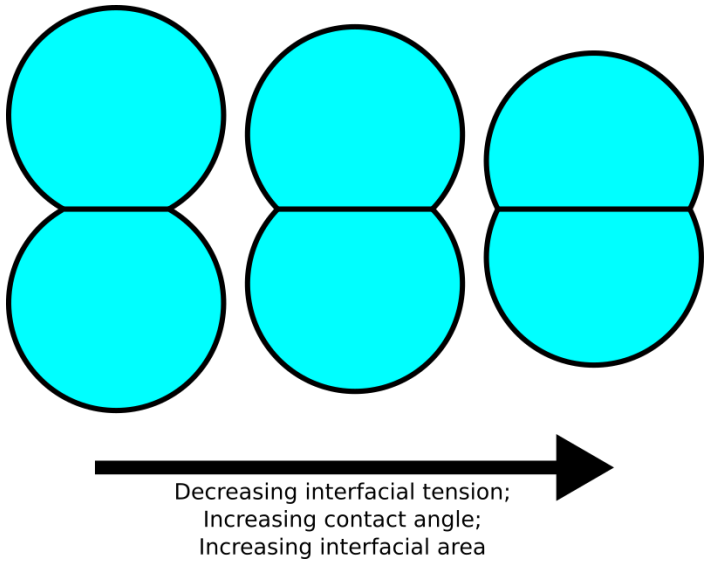


Fig. 1.7 Two dimensional diagram of cell doublets showing the effect of changing interfacial tension on contact angle and interface area.

Differential Interfacial Tension

The Differential Interfacial Tension Hypothesis (DITH) or Differential Surface Contraction (DSC) Hypothesis was initially proposed as an alternative to the DAH [61]. We can summarise the two basic postulates of the DITH as follows: the cell surface membrane is differentially contractile, and can have different tension values in different regions over the same membrane; secondly, the cell membrane is most contracted and has the highest tension when in contact with the surrounding medium (not other cells) and least contracted when in contact with other cells of the same type. Similarly to the DAH, inequalities to describe the relative tensions for which sorting occurs can be formulated fairly straightforwardly [62]. Figure 1.7 shows how a reduction in interfacial tension between two cells in a doublet results in an increased interfacial area, increased contact angle, and hence increased mutual affinity. It is this mutual affinity that allows differential interfacial tension (DIT) to drive self-organisation in a cell aggregate.

Cortical tension is the force generated within cells across their surface and parallel to the surface, produced by the cell cortex. The cell cortex comprises a dense network of actin microfilaments and myosin motors just inside the cell surface membrane [63, 64]. Tension is produced by a number of processes, including myosin II activity, cross linking, network structure, and active movements of the internal cell cytoskeleton, which transfer forces to the cell surface membrane [65, 66]. Actin fibres attach to the cell surface membrane via localised bundles of transmembrane proteins known as cadherins and integrins. The action

of myosin motors on the actin cortex changes the local tension of the fibre network [60], and by pulling the cadherin and integrin transmembrane protein bundles together, changes the effective tension of the cell surface in that local area. This allows cells to vary the local tension in their surface membrane [67].

Cortical tension can be measured by cutting the cell membrane with a laser (laser ablation) and measuring how quickly the cut edges of the membrane contract [68, 69], by probing the cell surface with an atomic force microscope [60, 70], or by applying force-balance analysis to 3D image stacks [71].

Cortical tension has already been shown to be important in critical cellular processes such as shape changes [72], motility [73**?** , 74], tissue mechanics [75], and adhesion [76]. Notably for us, the importance of cortical tension has been widely claimed in studies of morphogenesis [67]. These studies often focus on the importance of cortical tension in 2 dimensional epithelial tissues. Notable such systems are the maintenance of the dorsoventral compartment boundary in the *Drosophila* wing imaginal disc [77, 78], *Drosophila* germ-band extension [79], germ-layer organisation in zebrafish [80], and in *Xenopus* and *C. elegans* gastrulation [81, 82]. Generally this work can be divided into three categories: boundary formation, boundary maintenance, and shape changing. Sorting in the inner cell mass is an example of boundary formation, but lessons from boundary maintenance can be illuminating. For example, boundary maintenance in the *Drosophila* wing imaginal disc, where a boundary between dorsal and ventral tissue layers is initially formed by a biochemical gradient, but then maintained as a sharp boundary by tension forces [77]. This is a good example of the differential interfacial tension hypothesis.

Experimentally, it has been shown that actin and myosin accumulate around the the dorso-ventral boundary of the drosophila wing imaginal disk, and that this compartment boundary fails in mutants with faulty myosin [78]. This suggests a role for actin and myosin, and hence cortical tension, in the boundary maintenance. It has also been shown that treating cells with blebbistatin, an agent that stops myosin motors from working, prevents cell sorting [66]. It is now understood, on the basis of experimental and computational work [77], that the cortical tension of cells in the wing imaginal disc increases at interfaces between unlike dorsal and ventral cells, and it is this local increase in tension at unlike cell-cell interfaces that maintains the boundary between the two tissue layers [78, 83, 84]. Local changes in cortical tension at cell to cell interfaces change the area of those interfaces, and thus the relative mutual affinity of the two cells. Increasing the tension at unlike cell interfaces in the *Drosophila* wing imaginal disc minimises the contact area between the two cell types, resulting in spatial segregation and a sharp boundary.

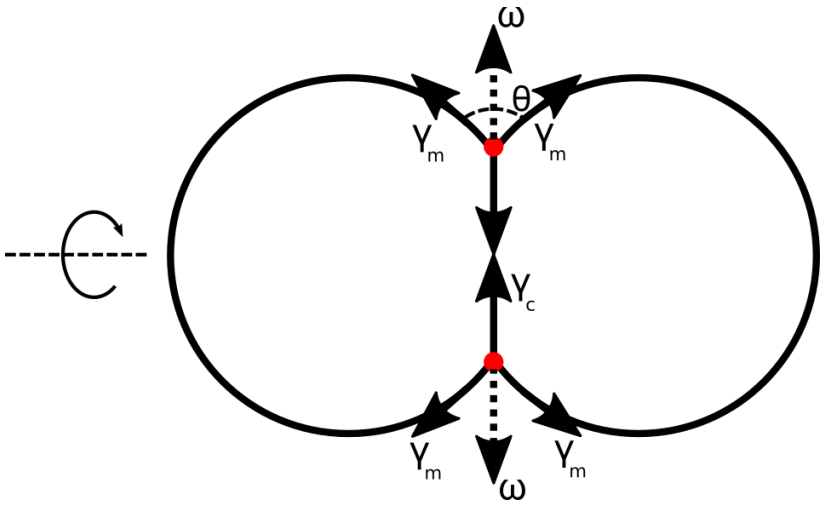
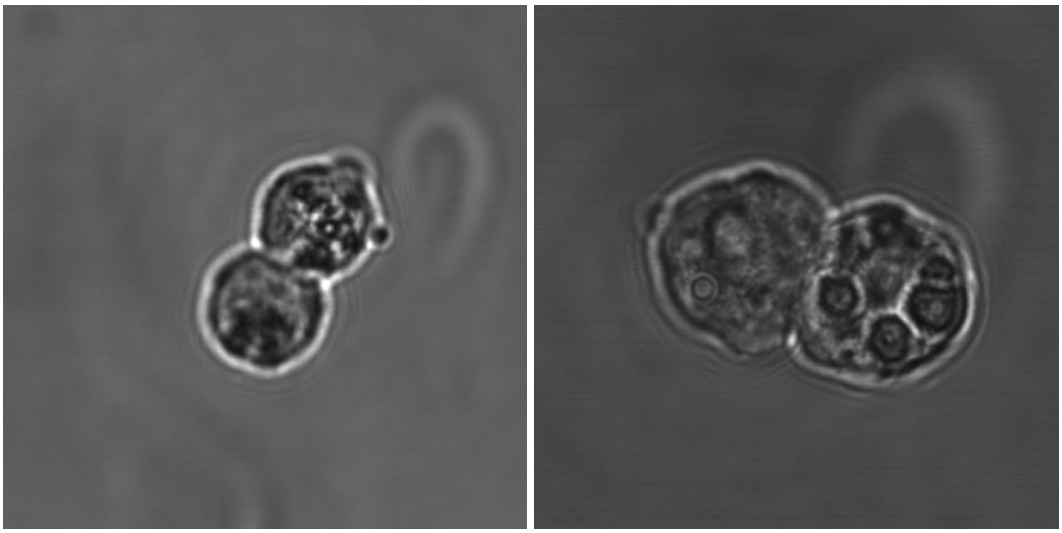


Fig. 1.8 Diagram outlining the linear force balance model of cell doublet contact angles, as presented in Maître 2012 [88]. Adhesion is reframed as a tension force ω acting to pull the interface vertices apart.

Having considered the effects of DIT in a two dimensional system such as the *Drosophila* wing imaginal disc, it is also illuminating to note its relevance and importance to systems closer to the inner cell mass. Notably, it has been shown that local variation in cortical tension is responsible for internalising the first set of internal cells in the morula [85], and thus establishing the first distinction between cells that will produce the inner cell mass, and those that will produce the trophectoderm. This work overturned previous assumptions that oriented cell division established the inner cell layer [45], and showed that local variations in tension act to organise the system after randomly oriented cell divisions. The parallels to the inner cell mass are clear, and it seems reasonable to speculate that similar forces might be at play.

Furthermore, it has also been shown how differential interfacial tension can drive compaction of the morula [86] and allocation of cells to the inner cell mass tissue [87] by reducing tension at cell-cell interfaces. This work validates differential interfacial tension for processes earlier in the embryo, and introduces a framework for testing the relative affinity of cell types. This is done by measuring the the contact angles of cell doublets [Figure 1.7]. Such measurements are contextualised within a theoretical model of cell doublet contact angles presented in Maître et al. 2012 [88], and outlined below.

The two factors contributing to the equilibrium interface between two cells are their mutual adhesion and their cortical tension. In the theoretical model, the complex statistical mechanics problem of how adhesion affects this interface is simplified so that it can be treated as a one dimensional problem of two points - the edges of the interface - acted upon by forces



(a) (b)

Fig. 1.9 *In vitro* cell doublet images. a, Epiblast-epiblast cell doublet. b, Primitive endoderm-primitive endoderm cell doublet. Images produced by Ayaka Yanagida in our group.

parallel to the interface [Figure 1.8]. The adhesion component is reframed as a tension force ω acting to pull the points apart. Also considered are the effects of the cortical tension at the cell-cell interface, γ_c, and the cortical tension away from the interface, γ_m. From Figure 1.8, it is then straightforward to formulate Equation 1.1, which predicts the contact angle θ.

$$2\gamma_c = 2\gamma_m \cos(\theta/2) + \omega \tag{1.1}$$

It is then assumed, based on experimental results [89, 90, 88], that in the high tension limit adhesion is insignificant and can be neglected, and thus Equation 1.1 can be simplified to Equation 1.2.

$$\theta = 2\arccos(\gamma_c/\gamma_m) \tag{1.2}$$

The reduction of tension is proposed to occur by the exclusion of myosin from the cortex at cell interfaces, caused by the adhesion acting between the cells [88]. This suggests a passive mechanism that allows morphogenesis to proceed without active movements by the cells involved, and provides the coordination needed for complex morphogenesis [67]. Doublet experiments done directly with inner cell mass embryonic cells support the existence of conditions required for sorting by differential interfacial tension [Figure 1.9]. Specifically, it is observed in experiments performed within our group by Dr. Ayaka Yanagida that epiblast-epiblast doublets have a typical contact angle of 106°, primitive endoderm to primitive

endoderm doublets have a typical contact angle of 77°, and epiblast-primitive endoderm doublets have a typical contact angle of 67°. Thus we might expect epiblasts to have higher mutual affinity than primitive endoderm, which have still higher mutual affinity than primitive endoderm and epiblast pairs. We set out to investigate whether these conditions are sufficient to explain sorting in the inner cell mass using a computational model. This model will be calibrated using doublet experiment data, and compared to the linear force balance doublet model for validation.

1.4.2 Energetic Mechanisms Driving Kinetics

Although it is always important to consider the timescales of any process, little exists in the literature describing precise timescales for lineage specification, cell sorting, cell deformation and so on. However, we have noted from movies of embryo development [Figure 1.10] that thermal motion is not the major contributor to cell movement, but rather the cells seem to move over much longer timescales. This begs the question: if thermal motion is not a significant factor in the system, what provides the perturbation needed to force the system out of equilibrium, leading to rearrangement and sorting? In order for cells to self-organise, differing mutual affinities are necessary but not sufficient; there must be movement of cells to rearrange the system. This can be achieved without active cell movement by introducing energy into the system, which will drive the system to reorganise to a new steady state. In our cell system, we hypothesise two mechanisms that could drive rearrangement.

Division

When two cells divide, their surface area to volume ratio is increased. This increase in surface area brings with it a corresponding increase in surface energy arising from cortical tension. This introduction of potential energy drives the system to rearrange to a new steady state, with the cells in a new orientation. Thus we might expect that a high cortical tension might be required in order to drive rearrangement. Division has been proposed as a driver of rearrangement in other systems [91] and is something that can be tested with respect to the inner cell mass in our model.

Figure 1.10 shows a C. elegans embryo, with images taken at intervals to show how cell division in this system appears to coincide with cell rearrangement.

Blebbing

It has been shown in recent experiments within our group that primitive endoderm cells, but not epiblast cells, bleb quite dramatically on a timescale of minutes [Figure 1.11, Figure 1.12].

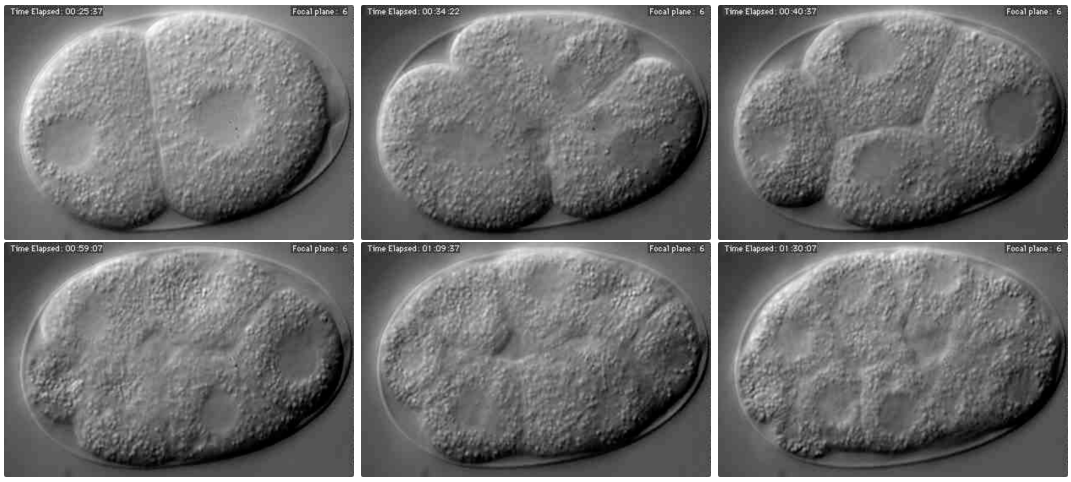
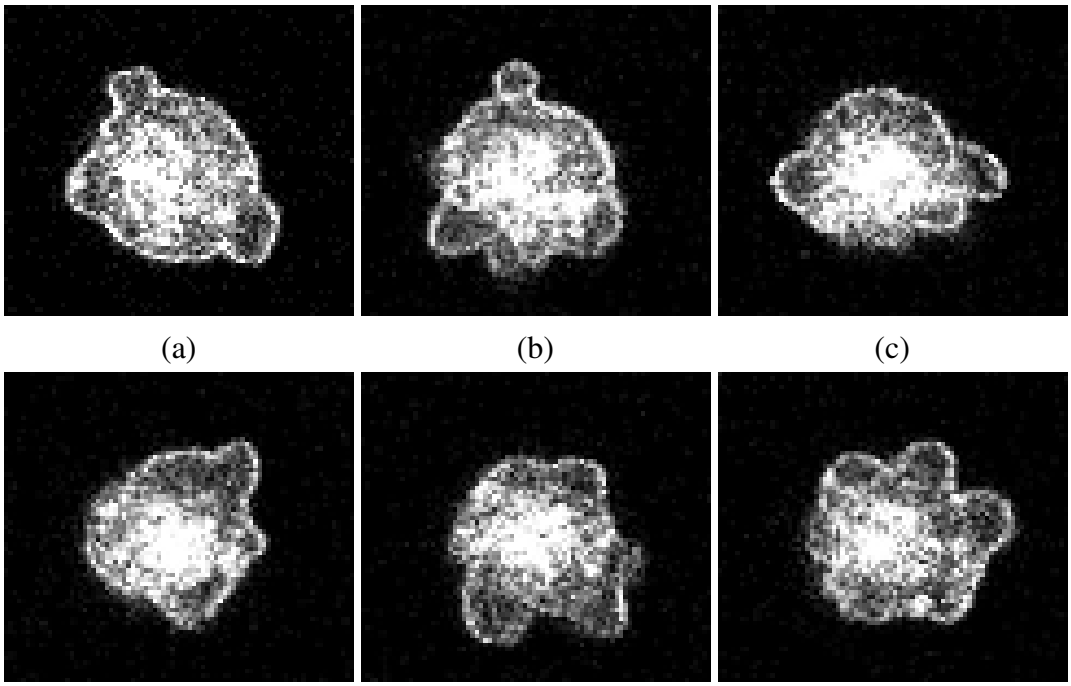


Fig. 1.10 Time sequence of a C. elegans embryo showing rearrangement due to cell division. Images courtesy of Magdalena Zernicke-Goetz.

This blebbing corresponds to local fluctuations in the cortical tension of the primitive endoderm cells [92]. we hypothesise that these fluctuations may also contribute to rearrangement of the system, and thus facilitate or speed up the process of self-organisation.

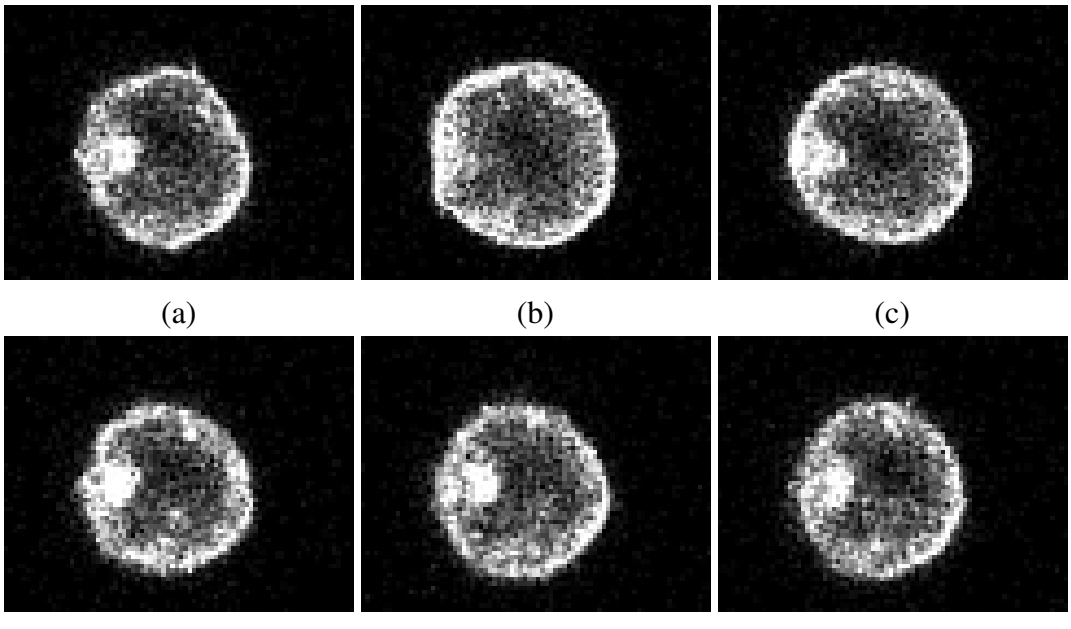
Not only could blebbing increase the extent of rearrangement in systems, but since it occurs only in primitive endoderm and not epiblasts, it also serves as an additional factor that could affect mutual affinity, and thus is another potential driver of sorting. Blebbing is a factor that we will include in our model, and its effects on self-organisation will be tested.



(a) (b) (c)

(d) (e) (f)

Fig. 1.11 Time sequence demonstrating blebbing in a primitive endoderm cell due to local fluctuations in cortical tension. Images produced by Ayaka Yanagida in our group.



(a) (b) (c)

(d) (e) (f)

Fig. 1.12 Time sequence demonstrating lack of blebbing in an epiblast cell. Images produced by Ayaka Yanagida in our group.

Chapter 2

Introduction to Tissue Modelling

Before approaching the problem of modelling self-organisation in the inner cell mass, it is worth exploring previous modelling methods that have been applied to other systems. There exists a broad literature of work using theoretical and computational techniques to model a great variety of biological systems, with varying levels of success [93]. Many of these techniques are summarised elsewhere [94–100]; what follows is an overview of those that were considered for this project. This is by no means a complete list, but merely a discussion of notable techniques and relevant features to enlighten and contextualise our work. Some models discussed below are generalised techniques, such as vertex models or cellular automata, for which we explain the theoretical foundations, whereas others are branded, self-contained software packages made available for the research community, such as CHaSTE and PhysiCell.

2.1 Mathematical Modelling

Mathematical modelling has a long history in biology, including the celebrated work of Alan Turing [101], and the separate field of mathematical biology is now well established, encompassing everything from morphogenesis to ecosystems and epidemiology. Typically, mathematical modelling of morphogenesis involves formulating a set of differential equations to describe a system, often reaction-diffusion equations [102], which must then be solved computationally. However, such modelling is typically only appropriate where a system can be treated in the continuum limit, such as treating a tissue as a continuous viscoelastic material, or modelling the migration of a large number of cells as movement of a fluid.

Notable examples of effective work treating tissue as a continuous material include explanations of folding in brain cortex tissue [103, 104], folding of epithelial tubes such as gut and renal tubes [105], and inversion of volvox embryos [106]. Similarly, we can

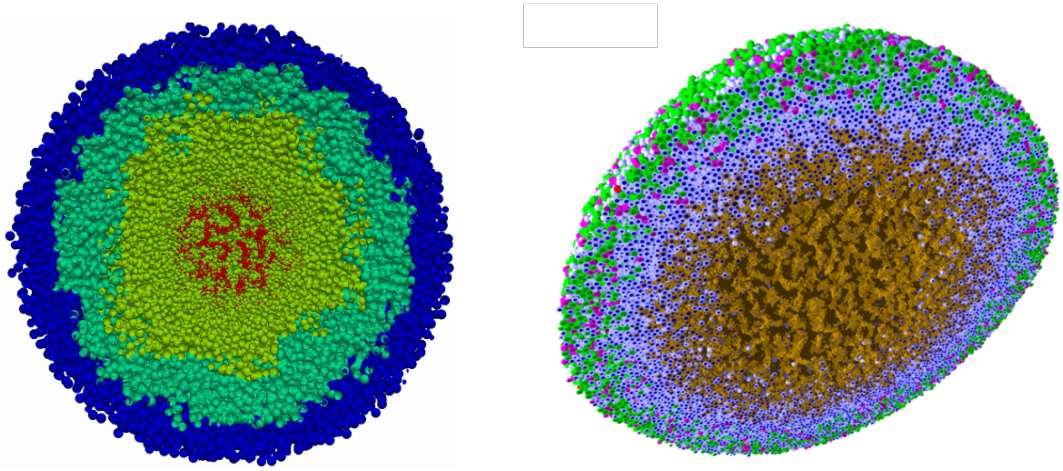
find work attempting to solve partial differential equations to explain self-organisation in developmental biology [107], collective migration of cells [108], long-range hydrodynamic interactions of microswimmers [109], and material properties of cancerous tumours [110]. All of these systems share tissues or aggregates in which cell-level features are orders of magnitude smaller than the system as a whole, which renders a continuum limit appropriate. This is not the case in the inner cell mass, so we are not able to use these techniques.

2.2 Agent-Based or Individual-Based Models

Agent-based modelling is widely used throughout fields ranging from sociology and economics to ecology, physics, and the study of complex systems [111–113]. In agent-based modelling, the most fundamental unit of the simulation is the agent, which in tissue modelling is typically a single cell. These agents interact, evolve, and move over time according to some predetermined set of rules. Some of the other methods discussed in this chapter, such as Biocellion and PhysiCell, are specific examples of agent-based models. Agents may exist on a lattice or move continuously, in 2 or 3 dimensions. On a fundamental level, agent-based models are driven by algorithms, not equations, which creates an important distinction with mathematical modelling [114, 115].

In studying morphogenesis, agent-based modelling has been applied to development of branching patterns in vasculature [116], *C. elegans* embryogenesis [117], collective cell migration [118], epidermal morphogenesis [119], and cancerous tumour development [120, 121], to name but a few. The precise details of all of these systems vary, but in each case the agent is an individual cell, and these cells form complex patterns over time by interacting with each other and with some background environment according to a set of rules laid out to simulate the details of the system in question. These interactions cause the cell to change fate, experience forces, or otherwise evolve. Examples of typical agent-based model results are shown in Figure 2.1.

Although agent-based individual cell models have proven successful in modelling multicellular systems with inter-cellular interactions, our assessment was that they did not incorporate sufficient integration of macroscopic behaviour with intra-cellular processes for our intended modelling of the inner cell mass, and thus we are unable to use these techniques. However, agent-based models of the mammalian blastocyst have been proposed previously, and these will be discussed later in this chapter [Section 2.10].



(a) (b)

Fig. 2.1 Visualisation showing the cross section of necrotic cancerous tumours as simulated with PhysiCell and Biocellion. Images from Macklin et al. 2016 [126] and Biocellion home page [123].

2.2.1 Biocellion

Biocellion is a self-contained software package originally developed for modelling cancerous tumours at the University of Washington [122, 123]. It is a generalised individual, agent-based model, treating each cell as a single object moving continuously within a background lattice that models the external environment, such as diffusion of biochemical factors. Its primary motivation was to provide parallelised simulations, allowing billions of cells to be run quickly over many processor cores on a large cluster. Consequently, it focuses on the wrong lengthscales for our systems and does not integrate subcellular physical processes into macroscopic behaviour. An example of a necrotic cancerous tumour containing a large number of cells and simulated with Biocellion is shown in Figure 2.1a.

2.2.2 PhysiCell

PhysiCell is another self-contained agent-based modelling tool released as open-source software by Paul Macklin's MathCancer group [124]. It was developed as a tool for modelling cancerous tumours [125, 126]. It is strikingly similar to Biocellion and shares the same advantages and disadvantages. Figure 2.1 shows a comparison between necrotic cancerous tumours as simulated with Biocellion and PhysiCell.

2.3 CHaSTE

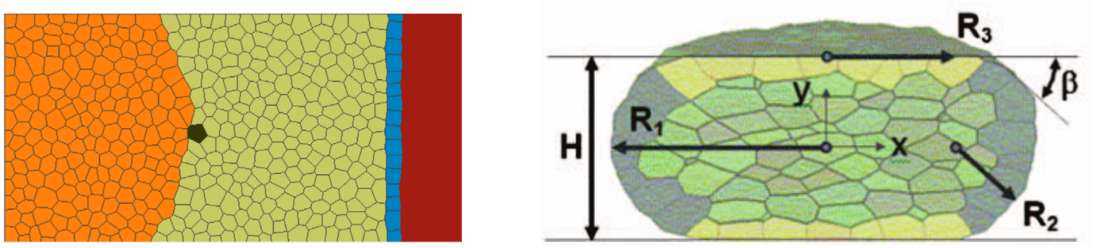
CHaSTE is an acronym that stands for Cancer, Heart, and Soft Tissue Environment [127]. It was developed primarily at the University of Oxford as an attempt to bring industry software development techniques to bear on tissue modelling in order to produce portable, reliable, reusable software that could allow biologists with less background in physical sciences or programming to contribute to computational biology, biological physics, and modelling without needing to start from scratch and "reinvent the wheel" [128]. The code is written primarily in C++ and is available to download on an open source license [129]. CHaSTE has multiple libraries for different capabilities, and is able to run on- and off-lattice individual-based models, in which each cell is represented by a single object or agent in the simulation, partial differential equation solvers, and vertex models. Chaste has been widely used to model a variety of different biological systems [130–132], but does not provide the integration of intra-cellular and inter-cellular processes that we required for our system.

2.4 Virtual Cell

Virtual Cell is a modelling and analysis platform developed at the University of Connecticut to solve stochastic partial differential equations and run agent based models of the inner workings of a cell [133–135]. It is available as open source software [136] and has found application in, for example, modelling calcium signalling in neurons [137]. However, since this method focusses exclusively on intra-cellular processes, it is unsuitable for our work on multi-cellular self-organisation in the inner cell mass.

2.5 Finite Element Models

The finite element method has existed for many years and is a mainstay of engineering and applied physical sciences [138]. The method has been applied to everything from plate tectonics [139] to studying the biomechanics of knee joints [140]. The finite element method is a technique to find the solution for partial differential equations in continuous systems such as elastic materials, soil, and so on. It involves dividing a continuous domain into smaller parts, known as finite elements, which can be solved more easily in relation to each other by eliminating spatial derivatives. This allows the equations describing the system to be solved approximately at a number of points across an entire system. Essentially by using the interactions between the elements of a body it is possible to model the behaviour of the body as a whole whilst capturing local effects on a smaller scale than the whole system.



(a) (b)

Fig. 2.2 Diagrams of a single cell modelled with FEM and an FEM model of a metastatic cell escaping from a cancerous tissue. Images from Brodland et al. 2009 [143] and Brodland et al. 2012 [141].

When applied to biological tissues, there are two broad categories of FEM models. Tissue level models typically treat the tissue as a whole as a continuous body, whilst individual cells form the finite elements into which the system is divided [141, 142] [Figure 2.2a]. These typically model a two dimensional system such as an epithelium, or a 2 dimensional approximation of a more complicated system. Meanwhile, other cell-level models exist to investigate the behaviour of a single cell, in which the cell itself is divided into finite elements [143, 144] [Figure 2.2b]. Although efforts have been made to add cell-cell interactions into these models, the work remains limited [145]. We require inter-cell interactions in our model, and thus will not emply a finite element model.

2.5.1 FEM/DEM

Drawing inspiration from granular matter research, it occurred to us that FEM/DEM might be suitable for multicellular system modelling. FEM/DEM originated as an extension of the more basic modelling tools FEM and DEM (Discrete Element Method). DEM is a very general form of modelling in which a system is composed of discrete elements or particles, each of which does not intrinsically change in time. Rather, the system as a whole evolves in time as the interactions between these unchanging particles causes them to rearrange with respect to each other and change their interactions with each other. This method is suitable for modelling gases, granular materials etc, and is distinguished from agent-based models by the passive nature of the elements. FEM/DEM or Finite-Discrete Element Method was developed in the 1980s by Antonio Munjiza [146, 147]. The basic idea is to model the behaviour of individual particles by DEM type methods while allowing each individual particle to be discretised into smaller elements by FEM methods to allow for deformation and changes in particle properties. The applicability of such a method to cell dynamics has, to the best of our knowledge, not been investigated but could be promising.

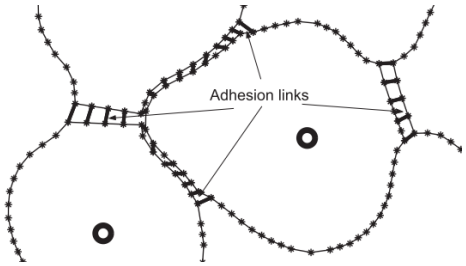


Fig. 2.3 Diagram of the immersed boundary method, showing where elastic boundaries apply elastic forces to an incompressible Newtonian fluid. Image from Rejniak 2007 [148].

2.6 Immersed Boundary Model

The immersed boundary method [149] was originally developed as a numerical tool to study blood flow around the heart [150, 151]. The method has since been adapted to model cells, and has been applied to systems such as cancer development [152, 148], morphogenesis [153], blebbing [154], endocytosis [155], and the formation of bacterial biofilms [156]. The immersed boundary method considers cells to be regions of fluid bounded by an elastic membrane within a continuous fluid, and simulates this system by solving the fluid behaviour using the Navier-Stokes equation on a fine grid where an additional elastic force term is added to the equation in regions of the fluid where a boundary is located. Thus the fluid itself is not treated as discontinuous but it still affected by the presence of these elastic forces arising from an immersed boundary. The movements of these fluid regions in which the elastic force is applied correspond to shape changes and movement of the membranes, and thus cells. This gives the method the great advantage over agent-based models of incorporating intra-cellular fluid mechanics and cell shape changes into the behaviour of the system. The method also allows spaces to exist between cells, resulting in discontinuous tissues, and giving an advantage over vertex models or cellular automata. By also incorporating adhesive forces between regions of fluids where cell boundaries exist, it is possible to model inter-cellular interactions and multi-cellular systems [152] [Figure 2.3]. It is also possible to model growth by introducing a fluid source into the centre of each cell, and division by applying a contractile ring to the plane bisecting a dividing cell. These features make it a strong contender for modelling tissue self-organisation in the inner cell mass. We could, for example, introduce differential interfacial tension by changing the elastic force applied by the immersed boundary when this boundary forms an interface with another. Whilst this seems like a promising way forward, we have explicitly set out to produce a 3 dimensional model of the inner cell mass, whereas most work with the immersed boundary method has been carried out in 2 dimensions. Although there is no theoretical problem with extending the

method to three dimensions, the computational cost of solving the Navier-Stokes equation over a reasonable number of cells in 3 dimensions would escalate rapidly, rendering these simulations difficult to perform and raising questions over whether these computational costs warrant the advantages that the immersed boundary method offers. We therefore chose not to implement the immersed boundary method, and instead to find a more computationally efficient technique.

2.7 Cellular Automata

Since they were first devised by the great polymath John von Neumann at the Los Alamos National Loboratory in the 1940s [157], cellular automata have become perhaps of the most widespread and established modelling technique of all, finding application in almost every field, from Ising spin models [158] and Conway's game of life [159], to the first forays into "artificial life" [160, 161].

At its most fundamental level, a cellular automaton is a regular lattice in which each point can have one of two or more states, such as "alive" or "dead". Lattice points change state according to a specified set of rules, and the system is allowed to evolve over discrete time steps. Most early applications of cellular automata to biological systems treated a single cell as occupying a single point within the lattice. A lattice point could then have two states: occupied by cell or not occupied by cell. This allows coarse grained modelling of morphogenesis in systems whose scale is much larger than that of an individual cell, such as vessel morphogenesis and branching [162], slime mould formation [163], or boundary maintenance [164]. It is also possible to treat cellular flows using lattice gas models, in which each point in the lattice is occupied by a cell that has not only a location but also a velocity through the lattice that can be maintained in time [165, 166]. Such models are comparable to the agent-based models previously described, but rather more limited. Since these early applications, most cellular automaton models applied to morphogenesis have been cellular Potts models.

2.7.1 Cellular Potts Models

Cellular Potts models are an adaptation of the original Potts model, a cellular automaton used to describe Ising spin systems [167, 168]. The number of spin states available at each state is known as the Q number, with the simple ferromagnetic Ising model described by q=2. The cellular Potts model is essentially a large-Q Potts model, meaning that each point in the lattice has a large number of possible states. The method was developed by Graner and

Glazier precisely for the purpose of modelling cell sorting by differential adhesion [169–171]. The innovation of the cellular Potts model was to treat a connected region of lattice sites with the same state as a single biological cell. Thus, changing the state of lattice sites does not indicate a new cell or movement of an existing cell as in original cellular automata, but rather represents a change of shape or growth of an existing cell. This feature allows cell growth, shape, and subcellular structure to be included in a model, with these features arising naturally from the rules established for a simulation. These rules define a Hamiltonian for the system such that any state of the lattice corresponds to a global potential state. The system evolves by a Monte Carlo method in which the state of a random lattice point is randomly altered. This change of state causes a corresponding change in global potential, determined by the Hamiltonian. The method then draws upon statistical mechanics by calculating the Boltzmann factor of the new potential state, which results in a probability that can finally be compared to a random number to determine whether or not the change of state to the system is accepted. Thus, changes that significantly increase the potential are highly unlikely to be accepted, whereas those that cause no change in potential are much more likely to be accepted.

Given this framework, the critical component of a cellular Potts model is the precise definition of the global Hamiltonian. The Hamiltonian will vary from one system to another and one study to another, but can include components for cell-cell adhesion, volume conservation, gravity, biochemical interactions, or cortical tension [172]. Over subsequent years, the cellular Potts model has been used to study many systems, particularly cell sorting by differential adhesion [173–176], but also systems outside of biology such as foams [177]. The technique has also been developed into self-contained software packages such as CompuCell [178] for easy implementation of simulations by biologists with less programming experience.

Despite the widespread use and success of the cellular Potts model, we felt that it was not the right technique for our model of the inner cell mass. Cellular systems are active and constantly producing energy in the form of ATP. The use of a global Hamiltonian is artificial and seems unphysical such systems where energy is explicitly not conserved. Furthermore, the global nature of the Hamiltonian is incompatible with small scale local effects, such as blebbing and differential interfacial tension, which we believe to be critical components of modelling tissue self-organisation in the inner cell mas. In order to avoid “reminiscence modelling”, we intend to avoid energies and examine forces and dynamics in the system directly, and thus will not use cellular automata or the cellular Potts model.

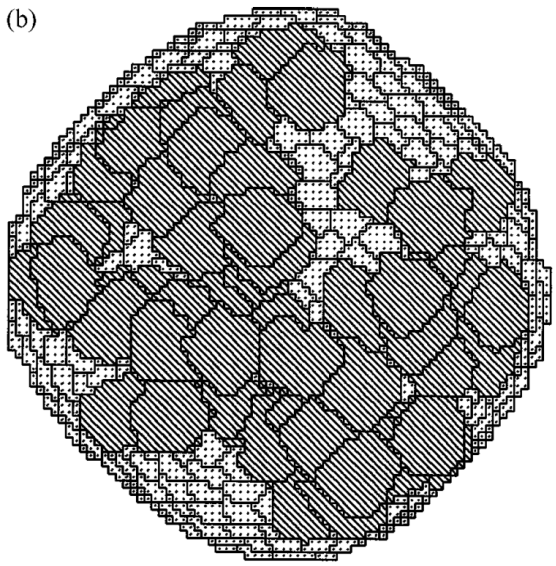


Fig. 2.4 A cellular Potts model of a multicellular aggregate of two differently sized cell types. Image from Mombach et al. 1995 [173].

2.8 Vertex Methods

Vertex models are another widely used technique in tissue modelling, and can be thought of as a special case of finite element methods [**?**]. The earliest application of vertex methods described the use of Dirichlet domains to cover an entire tissue with polygonal cells [179]. This is the fundamental idea of vertex methods: a tissue is represented as a lattice of irregular polygons in which each polygon represents a biological cell; movement of these cells and morphogenesis of the tissue as a whole arises from the relative movement of the vertices of these polygons.

The system evolves in time by randomly choosing a vertex and randomly altering it's position. A global Hamiltonian is defined for the system, typically including components that are functions of edge length, corresponding to cortical tension, and cell volume or area, corresponding to cell elasticity [83]. Some vertex models derive their dynamics from differentiating the Hamiltonian as a function of all vertex positions. Other vertex models employ Monte Carlo methods in which a change in the position of a vertex changes the global potential defined by this Hamiltonian, and, as for the cellular Potts model, this potential corresponds to a Boltzmann factor that defines a probability. This probability is compared to a random number to determine whether or not the change in position is accepted. An increase in the global potential, as defined by the Hamiltonian, corresponds to a small probability that the change will be accepted, whereas a decrease in the global potential corresponds to a higher probability that the change will be accepted.

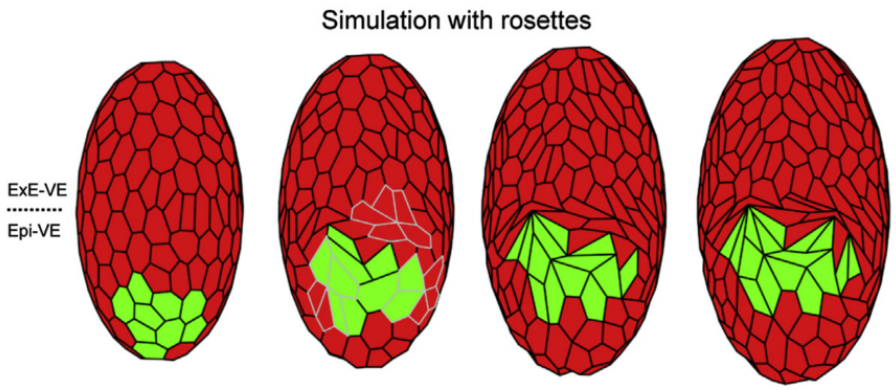


Fig. 2.5 Vertex model of rosette formation in mouse visceral endoderm. Image from Fletcher et al. 2014 [97].

Vertex models have been used extensively in modelling systems such as epithelial morphogenesis [97, 99, 180], *Drosophila* wing boundary maintenance [83, 181, 78, 84], or *Drosophila* germ band extension [182]. These systems all share three common properties that render them suitable for modelling with vertex methods: the cells are tightly packed, forming confluent tissues with no spaces; the systems are two dimensional; the cells in the system have fairly regular shapes that can be well approximated by polygons. Although vertex models have been extended to three dimensions [183], and advanced with the active vertex model [184] and self-propelled voronoi models [185], they retain a number of significant drawbacks that render them unsuitable for modelling tissue self-organisation, notably a reliance on global Hamiltonians as discussed for the cellular Potts model, a requirement for confluent tissue, and treating cells as polygons. Furthermore, vertex models start with interfaces between cells as a fundamental component, whereas we wish to model interfaces emerging from underlying behaviour in our testing of differential interfacial tension. Therefore vertex models were deemed inappropriate for our purposes.

Figure 2.5 shows a typical example of a vertex model.

2.9 Subcellular Element Method

The Subcellular Element Method (SEM) was developed by Timothy Newman in 2005 as a technique to integrate sub-cell-scale processes into multicellular morphogenesis [186]. Although relatively new and not fully explored, SEM has since been applied to a number of multicellular biological systems [187–190].

SEM divides a cell into a number of discrete, point like 'elements', which interact via local forces that can be tuned based on the underlying physics of the system. Element positions are updated based on these local forces using over-damped Langevin dynamics. By also considering local forces acting between elements in different cells, SEM allows a very straightforward extension from internal dynamics to external dynamics. Treating both inter- and intra-cellular dynamics as fundamentally similar makes for a parsimonious approach, unlike in some other techniques that require different mechanisms for inter- and intra-cellular mechanics. Furthermore, it is straightforward to include properties such as cell growth - by introducing new elements into cells and allowing the system to relax - and cell division - by simply defining half of the elements in one cell as now being part of a different cell.

This basic framework can be extended easily, as was done previously to model the effect of active cytoskeleton remodelling in response to strain on the behaviour of a cell stretched between two plates [190]. New elements were added in areas of high strain and removed from areas of low strain to model the formation and removal cytoskeleton filaments in response to external strain. It has been shown in such studies that SEM correctly predicts properties such as the microrheology of an individual biological cell [187].

The subcellular element method has a number of advantages for our attempts to model self-organisation in the inner cell mass. Notably, the integration of intra-cellular and local processes into multicellular aggregates, emphasis on local forces rather than global energies, and ability to model growing aggregates on a scale of tens of cells are exactly what we were looking for in our modelling technique. Thus, we decided to proceed with the subcellular element method and will discuss it further in Chapter 3.

2.10 Models of the Blastocyst and Inner Cell Mass

Before proceeding, it is worth noting how some of the techniques outlined in this chapter have previously been applied to the mammalian blastocyst and inner cell mass.

Honda et al. propose a noteworthy vertex model of blastocyst formation [23]. They show that a simple model with cell surface elasticity, blastocoele growth and external pressure from the zone pellucida are sufficent to recapitulate full blastocyst development from morula stage to blastocyst with blastocoele and inner cell mass [Figure 2.6]. Their three dimensional vertex model begins with a spherical morula into which spaces are introduced between cells to model represent secretion of fluid into the interstitial space. As the volume of these fluid-filled spaces is increased over time they coalesce to form a blastocoele. They show that the formation of an asymmetrical structure with an inner cell mass at one pole is a better

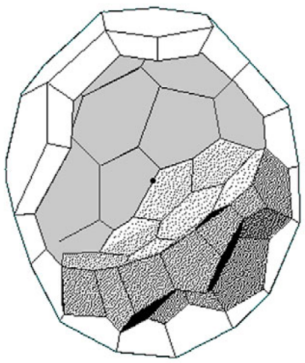


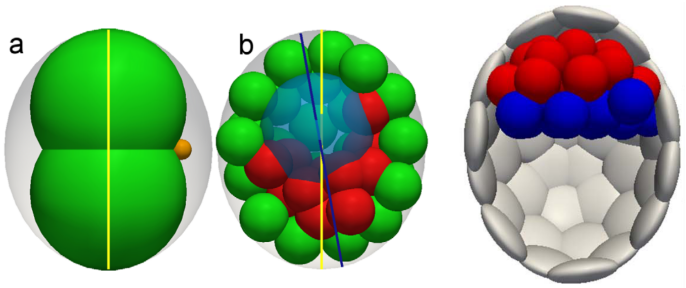
Fig. 2.6 Cross section of a blastocyst resulting from the vertex model of Honda et al. Image from Honda et al. 2008 [23].

minimisation of the free energy resulting from external and internal pressure than a simple spherically symmetric shell of cells surrounding the blastocoele.

The model may not be entirely accurate since it requires external pressure from the zone pellucida, and it has been shown experimentally that blastocyst formation can proceed normally when the zone pellucida is removed [191]. However, it seems highly plausible that the bulging shape of the ICM may indeed be formed in part by free energy minimisation due to the pressure of the blastocoele. Nonetheless, although this model is extremely interesting, it deals only with the system leading up to that with which we are concerned.

Meanwhile, Krupinski et al. present an agent-based model that attempts to explain not only the formation of the blastocyst structure [Figure 2.7], but also, of more interest to us, the self-organisation of the inner cell mass [192, 193]. Their model treats cells as spherical agents interacting with their neighbours with elastic forces. Regarding the inner cell mass, they set out to show how primitive endoderm and epiblast cells self-organise with the primitive endoderm lying along the boundary of the blastocoele using differential adhesion. They find, unsurprisingly, that differential adhesion can cause the two cell types to sort into separate aggregates. However, they find that differential adhesion alone is insufficient to stabilise the primitive endoderm alongside the blastocoele, and find that this is only achieved by pressure from the blastocoele on the inner cell mass combined with adhesion of epiblasts to the trophectoderm [Figure 2.8].

This is interesting work, but as discussed in Chapter 1, it has been shown experimentally that differential adhesion cannot be the explanation of self-organisation in the inner cell mass, and there is no experimental reason to suggest increased adhesion between epiblasts and the trophectoderm. Furthermore, there is significant evidence to show that differential interfacial tension is an active factor in blastocyst development, and that the conditions required for self-organisation by differential adhesion exist in the cells of the inner cell mass.



(a) (b)

Fig. 2.7 Simulated blastocyst resulting from the agent-based model of Krupinski et al. Images from Krupinski et al. 2011 [192].

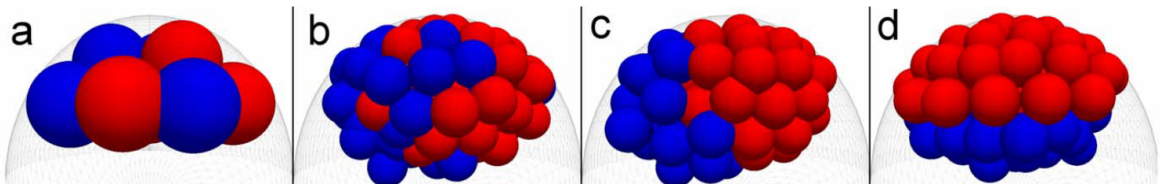


Fig. 2.8 Simulated inner cell mass resulting from differential adhesion in the agent-based model of Krupinski et al., showing in d how adhesion of epiblasts to the trophectoderm stabilises the primitive endoderm against the blastocoele. Image from Krupinski et al. 2011 [192].

Thus, although we agree that the pressure from the blastocoele, perhaps combined with the geometry of the inner cell mass, may contribute to breaking the symmetry of spherical sorting to produce a primitive endoderm layer that lies alongside the blastocoele, we feel that it is incorrect to propose sorting by differential adhesion. The true mechanisms are more subtle and less obvious, and it is our intention in this work to elucidate those mechanisms.

Chapter 3

The Sub-cellular Element Method

3.1 Theoretical Foundations

The subcellular element method was devised as a multi-scale framework for modelling tissue dynamics in three dimensions, incorporating both individual cell structures and sub-cellular features to allow for adaptive shape changes and complex inter- and intra-cell behaviour [186]. The method treats each cell as a cloud of infinitesimal points that interact with their nearest neighbours via local forces. Thus the behaviour of a cell emerges entirely from the behaviour of underlying elements. Nearest neighbour interactions occur between elements in the same cell and between elements in different cells, allowing both inter- and intra-cellular behaviour to be modelled with the same mechanism. Figure 3.1 shows a 2 dimensional diagram of two SEM cells. Elements are shown as green circles, with dotted black lines showing nearest-neighbour interactions. The two dotted red lines show inter-cell nearest neighbour interactions.

The nearest neighbour interactions between elements take the form of a Morse potential [194] [Equation 3.1], which is repulsive at short distances, attractive at longer distances and approximately harmonic around an equilibrium position. For later work, it is worth noting that the potential can be broken into an attractive component and a repulsive component.

$$\begin{aligned} V(r) &= D_e \left(1 - e^{-a(r-r_e)}\right)^2 \\ V(r) &= D_e \left(e^{-2a(r-r_e)} - 2e^{-a(r-r_e)} + 1\right) \end{aligned} \tag{3.1}$$

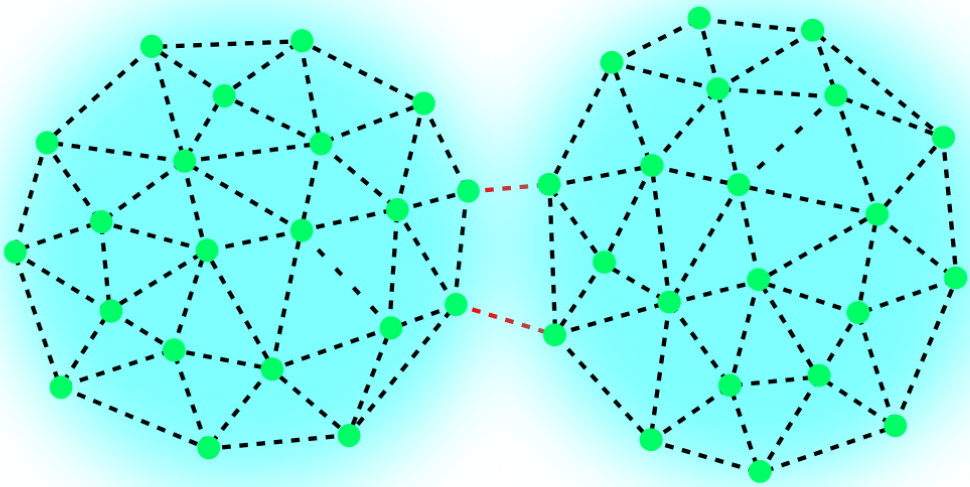


Fig. 3.1 A 2D representation of two SEM cells, with intra-cellular interactions shown in black, and inter-cellular interactions in red.

This potential form has historically been used to model inter-atomic forces in molecules and materials [195]. Figure 3.2 shows the functional form of the Morse potential for radii between 0 and 1.8 times the equilibrium radius. Basic values used in future simulations are shown below.

r_e	12.9
a	30.5
D_e	2.29

3.2 The SEM Program

The basic SEM code was written in FORTRAN and comprises a total of 27 FORTRAN modules and several supplementary scripts [Code block 3.1]. The following is a brief description of the main features of the code.

3.2.1 Data Structures

The basic SEM program is a set of routines to manipulate and update two fundamental data classes: `element` and `cell`. The `element` data structure is the basic unit of the simulation - the elements with which a cell is modelled. These are the objects upon which all position update

```
ScEM_0_arrays.f90
ScEM_0_input.f90
ScEM_0_ran_array.f90
ScEM_0_useful.f90
ScEM_1_inflexion.f90
ScEM_1_potential.f90
ScEM_1_types.f90
ScEM_2_ageing.f90
ScEM_2_com.f90
ScEM_2_deallocate.f90
ScEM_2_diffusion.f90
ScEM_2_division.f90
ScEM_2_flag_relist.f90
ScEM_2_growth.f90
ScEM_2_identity.f90
ScEM_2_initial_create.f90
ScEM_2_initial_exist.f90
ScEM_2_integrate.f90
ScEM_2_output.f90
ScEM_2_output_final.f90
ScEM_2_pairs.f90
ScEM_2_relist.f90
ScEM_2_resize.f90
ScEM_3_initialize.f90
ScEM_3_iterate.f90
ScEM_master.f90
ScEM_script_clear_data
ScEM_script_create_to_exist
ScEM_script_intel_compile
config_data
elements_config
rng.f90
```

Code Block 3.1 List of files in the original SEM program

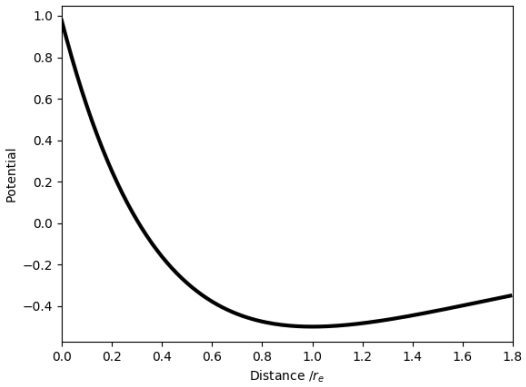


Fig. 3.2 Plot of the basic Morse potential as a function of equilibrium radius.

```
type element
  integer :: label
  integer :: parent
  integer :: stage
  integer :: type
  real*8  :: age
  real*8  :: strength
  real*8, dimension(3) :: position
  real*8, dimension(3) :: velocity
end type element
```

Code Block 3.2 Definition of the `element` data class in the original SEM code.

routines act. The original definition for the `element` class is shown in Code block 3.2. Each element has a unique label that is used to identify that element in calculations. Also stored within the data structure is the label of the element's "parent" cell, which is the unique label of the `cell` structure to which the element is allocated. The `stage` and `strength` components control the interaction strength of the element, and are used to steadily increase the interaction strength of an element from zero to full strength when a new element is inserted into a cell, allowing for a more gradual relaxation to the new arrangement [Subsection 3.2.5]. The `age` component stores the time that has elapsed in the simulation since the element was created, while the `position` and `velocity` components are the crucial 3 dimensional arrays that hold the position and velocity of that element at the current time. The `type` components allows elements to be defined as different types but was not used in the original program. The set of `element` structures is stored a one-dimensional array `elements` of type `element`, defined in module `ScEM_0_arrays.f90`.

```
type cell
  integer :: label
  real*8  :: age
  real*8  :: rad_gyration
  real*8, dimension(3)            :: position
  integer, dimension(0:4*ne_cell) :: c_elements
end type cell
```

Code Block 3.3 Definition of the `cell` data class in the original SEM code.

The `cell` data class is a secondary structure that organises sets of elements. A separate `cell` structure is created for each biological cell simulated, and all are stored in a one dimensional array `cells`, again defined in module `ScEM_0_arrays.f90`. The `label` component is the unique identifier of this cell, and corresponds to the `parent` component of all elements within this cell. The `age` component stores the smaller value of either the simulation time that has passed since either the beginning of the simulation, or since the division event that produced the cell. The `position` component is the centre of mass of all elements within the cell. The `c_elements` component is a one-dimensional array that stores the unique identifying labels of all `element` structures organised into this `cell`. The zeroth component of the array is the number of elements within the cell, and all subsequent components are the labels of these elements. The size of the array exceeds the number of elements it will ever need to contain in order to prevent potential segmentation faults or the need to resize the array during a run. A cell will divide [Subsection 3.2.6] when the number of elements it contains exceeds double the original number of elements, but the buffer is built in to accommodate whatever margin the number of elements exceeds this value by in the timestep immediately before division is triggered.

Similarly, a significant buffer is built into the `elements` and `cells` arrays in order to prevent dynamic updating of the array size during a run, which is a slow process. Updating of the array sizes is triggered when the total number of elements or cells exceeds some fraction of the array size. These array size updates are handled in module `ScEM_2_resize.f90`.

3.2.2 Initialisation

The first step in the program is to call the subroutine `scem_input`, from module `ScEM_0_input.f90`, which defines all parameters and initial conditions for the simulation as global variables. This includes setting the total time for the simulation, typical length scales, the typical number of elements per cell, parameters for the defining the inter-element poten-

tial, and basic values such as a value of pi and a 3-dimensional packing fraction. Also defined are a number of control flags - `flag_create`, `flag_diffusion`, `flag_growth`, `flag_division` - whose values control which subroutines are called in future iterations of the program. Once these parameters are defined, the simulation is initialised by creating the initial system.

The original program incorporated two protocols for initialising the simulations. Both protocols produce an arrangement of elements and cells that consitute the initial state of the system to be simulated, and instantiate all necessary classes and arrays. The module `ScEM_2_initial_create.f90` is used to create a *de novo* system; the module `ScEM_2_initial_exist.f90` is used to initialise simulations with a pre-defined system, for which all necessary information is read from a small set of files. Both modules are set up in the original SEM program to create just one cell. Whether the program calls routines from `ScEM_2_initial_create.f90` or from `ScEM_2_initial_exist.f90` depends on the value of integer `flag_create`, which is set to 0 to use `scem_initial_exist` and 1 to use `scem_initial_create`.

The module `ScEM_2_initial_create.f90` is built around Code Block 3.4. This block is repeated within a while loop until the number of elements created equals the minimum number of elements per cell. It calls `ran_array` to create a random 3D position within a cube of side `2*r_cell` where `r_cell` is the pre-defined typical cell radius. This position is the possible location of a new element. Subsequently the code checks to ensure that the new element position is within a radius of `r_cell` - effectively ensuring that elements are only introduced within a sphere, not a cube, and that the new element position is not within distance `r_close` of any previously created elements. If either of these conditions are not met, the block starts again with a new random position, but if the conditions are met an element is introduced at this location, all of its corresponsing data are instantiated, and the process is repeated for the next element until all required elements have been created. When this *de novo* creation routine is used, the time step for subsequent updates is reduced, to accommodate any relaxation required by the newly created cell, which may not be in equilibrium and will quickly adjust.

The module `ScEM_2_initial_exist.f90` skips the process of creating a new cell *de novo* by reading the initial system state from saved files. The file `elements_config` stores the initial locations of all elements in the system, and is read with a loop within `ScEM_2_initial_exist.f90` as shown in Code block 3.5.

Reading the system state from file also requires the `config_data` file, which stores the initial number of cells, initial number of elements, and initial number of inter-element interaction pairs in the system, although the first two of those are technically redundant. The data from `config_data` is read in `ScEM_3_initialize.f90`, which takes the initial number

```
flag_success=1
call ran_array(ra,1,3,iseed)
pos_1(:)=r_cell*(2*ra(1,:)-1)
if (dim.eq.2) pos_1(3)=0.0
rad_sq=dot_product(pos_1,pos_1)
if (rad_sq.lt.r_cell_sq) then
  icount_p=1
  do while ((flag_success.eq.1).and.(icount_p.le.icount))
     pos_2(:)=elements(icount_p)%position(:)
     dx(:)=pos_1(:)-pos_2(:)
     dist_sq=dot_product(dx,dx)
     if (dist_sq.lt.r_close_sq) then
        flag_success=0
     end if
     icount_p=icount_p+1
  end do
else
  flag_success=0
end if
```

Code Block 3.4 Loop that creates pseudo-randomly positioned elements for an initial cell

```
open(unit=12,file='elements_config',status='old')
do n=1,ne
   read(12,*)elements(n)%position(:)
   elements(n)%label=n
   elements(n)%type=1
   elements(n)%stage=1
   elements(n)%parent=1
   elements(n)%age=establishment_time
   elements(n)%strength=1.0
   elements(n)%velocity(:)=0.0
end do
```

Code Block 3.5 Loop within `ScEM_2_initial_exist.f90` to read element position data from `elements_config`.

of pairs, elements and cells from the file. If module `ScEM_2_initial_create.f90` is used instead, module `ScEM_3_initialize.f90` does not read data from `config_data`, but instead sets the number of cells to 1, the number of elements to `ne_cell`, and estimates the number of interaction pairs as the total number of elements in the system (which for one cell will be the same as `ne_cell`).

Regardless of how the initial system state is initialised, `ScEM_3_initialize.f90` performs a number of housekeeping tasks to set up the simulation. First, it allocates the size of a number of important arrays according to the initial system size. It then call a sequence of subroutines, each contained in a corresponding module, as follows: `scem_relist(0)` to initialize the sector array (see Subsection 3.2.3), `scem_pairs` to find all inter-element interaction pairs (Subsection 3.2.4), `scem_identity` to set the internal `c_elements` array of the `cell` data structure, `scem_com` to calculate the centre of mass and radius of gyration of all cells in the system, and `scem_output` to save the initial state of the system to file.

3.2.3 Sector Array

The sector array divides the space in which simulations run into an abstract three dimensional grid. The grid elements have equal lengths of `sector_size`, which itself is equal to 1.1 times the maximum interaction range for inter-element potentials, `r_interaction_max`. The sector array is mostly handled by module `ScEM_2_relist.f90`, which contains subroutine `scem_relist`. This subroutine is initally called from `ScEM_3_initialize.f90` with argument 0. The array `head` has one component for each grid element of the sector array, and in this instance with argument 0, subroutine `scem_relist` begins by setting all components of `head` to be 0. Subsequently there is a loop over all elements as shown in Code block 3.6. For each element, the loop calculates the 3 dimensional array `ixe`, which corresponds to the specific sector array grid element within which the element is located. `ixe` is then used to identify the component of the `head` array corresponding to this sector array grid element, and its value is updated to the label of the element. Thus when the loop is completed, the each component of the `head` array stores the highest value label of all SEM elements located within the corresponding sector array grid element. Furthermore, the component of the `list` array corresponding to each SEM element is used to store the previous value of the `head` array, which is the next highest value element label within this sector array location. Thus by using the `head` and `list` arrays it is possible to find the full set of elements within each sector array grid element, because `head[a,b,c]=n` where n is the highest label element in sector abc, and `list[n]=m` where m is the next highest label element in sector abc, and so on until `list[m]=0`, at which point m is the lowest lavel element in sector abc. This ability is important for identifying nearest-neighbour interaction pairs [Subsection 3.2.4].

```
do n=1,ne
   ixe(:)=1 + int((elements(n)%position(:) + x_cen(:))*recip_sector_size)
   list(n)=head(ixe(1),ixe(2),ixe(3))
   head(ixe(1),ixe(2),ixe(3))=n
end do
```

Code Block 3.6 Loop over all elements to evaluate each component of the `head` array. From `ScEM_2_relist.f90`.

```
do n=1,ne
  if (list(n).eq.0) then
     ixe(:)=1+int((xe_compare(n,:)+x_cen(:))*recip_sector_size)
     head(ixe(1),ixe(2),ixe(3))=0
  end if
end do
```

Code Block 3.7 Loop triggered to initiate relisting sector array. From `ScEM_2_relist.f90`.

After the initial call of subroutine `scem_relist` with argument 0, every subsequent call uses argument 1, and occurs when the `flag_relist` variable is set to 1 by the module `ScEM_2_flag_relist.f90`. Within this module, the subroutine `scem_flag_relist` checks whether any element has moved from its previous location by more than a sufficient distance to warrant rechecking which sectors each element is located within. If this criterion is met, `flag_relist` is set to 1 and `scem_relist` is called again with argument 1. This triggers the loop shown in Code block 3.7, which for any element that was previously the lowest label element in its sector, sets the `head` value for that sector to be zero. Subsequently the subroutine contines through Code block 3.6.

3.2.4 Interaction Pairs

The module `ScEM_2_pairs.f90` contains the subroutine `scem_pairs`, which is called to find all inter-element nearest neighbour interaction pairs. These pairs are stored within the `pairs(:,:)` array. The first index of this array gives the pair label, and the second index takes values of 1 or 2, such that `pairs(n,1)` gives the label of the first element in the interaction pair, and `pairs(n,2)` gives the label of the second element in the interaction pair. To avoid double-counting, the labels are arranged such that the first label always has a lower value than the second label.

The subroutine is built around a loop over all elements in the system. Within this loop, there are two main sections. The first, shown in Code block 3.8, handles the actual identification of interaction pairs. The second resizes the `pairs` array to double its original capacity if the number of pairs counted so far within the loop exceeds some fraction of the current size of the `pairs` array.

For the identification of interaction pairs (Code block 3.8), the first step is to identify the position of the element under consideration and hence calculate its grid position within the sector array, `ixe(:)`. Using the sector array avoids another loop over all elements within the system, since the size of the sector grid is such that any element within interaction range of the current element must lie within the same sector or the 26 surrounding sectors. Consequently, the next step is to loop over these possible sectors, using the `head` and `list` arrays to identify any elements located within these sectors, and testing each of these elements to determine whether it falls within `sector_size` of the current element (where `sector_size=1.1*r_interaction_max`). If this condition is met, the two elements are stored as an interaction in the `pairs` array.

Following the loop that contains Code block 3.8, the number of pairs in the system is updated to the total value of icount found in the loop: `np=icount`.

3.2.5 Growth

Cell growth is handled by the subroutine `scem_growth` within module `ScEM_2_growth.f90`. This subroutine is built around a loop over all cells in the system, `nc`. For each cell, the subroutine uses the radius of gyration of that cell to define the radius of an internal core region within which new elements can be added: `r_core=frac_growth*cells(k)%rad_gyration`. The use of a growth core ensures that cells grow from the inside rather than potentially introducing new elements randomly to the outside of the cell. Once this region has been defined, the code laid out in Code block 3.9 attempts to place a new element within it. The first step is to call a random number generator and compare the result to the probability of placing a new element in a given timestep, `prob_new_element`. This probability is defined by the number of elements per cell and a predefined cell cycle time, giving the expected time taken for one round of cell division `prob_new_element=dt*ne_cell/cell_cycle_time`. If the random number is smaller than the probability of placing a new element, the program begins attempting to place a new element within the cell. This process is controlled by `flag_success`, which is set to 1 once an element has been placed. In order to place a new element, the program randomly selects an existing element within the cell. If this element is within 90% of the radius of gyration of the cell from the centre of the cell, it is accepted for use as the "nucleation" point of a new element. A random direction is chosen, and the

```
pos_1(:)=elements(n)%position(:)
ixe(:)=1 + int((elements(n)%position(:) + x_cen(:))*recip_sector_size)
do ix=-1,1
  do iy=-1,1
    do iz=-1,1
      nn=head(ixe(1)+ix,ixe(2)+iy,ixe(3)+iz)
      do while (nn.ne.0)
        if (n.lt.nn) then
          pos_2(:)=elements(nn)%position(:)
          dx(:)=pos_1(:)-pos_2(:)
          sep_sq=dot_product(dx,dx)
          if (sep_sq.le.sector_size_sq) then
            icount=icount+1
            pairs(icount,1)=n
            pairs(icount,2)=nn
          end if
        end if
        nn=list(nn)
      end do
    end do
  end do
end do
```

Code Block 3.8 Code to identify nearest neighbour inter-element interaction pairs, from `ScEM_2_pairs.f90`

new element is placed at `0.6*r_equi` in this direction from the nucleation element, where `r_equil` is the equilibrium distance of the inter-element interactions. This is followed by a section of bookkeeping to update relevant components of the `elements` and `cells` arrays. Finally, a block searches for all new inter-element interaction pairs for this new element, before moving on to the next cell.

New elements are created with the `strength` and `stage` components of the `element` data structure set to 0. The `stage` component can take values 0, 1 or 2, to indicate that the element is "fading in", "steady state" or "fading out" (fading out is not used in the basic SEM). For "steady state" elements (`stage=1`) the element will interact with its nearest neighbours normally. Newly created elements are given `stage=0`, indicating that they are "fading in". This causes the interaction potentials experienced by the element to be updated by a factor given by the `strength` component of the element. The `strength` of an element is given by $(age/establishmenttime)^2$, where the establishment time is the time it takes to fully fade in a new element. It is a coefficient used to slowly increase the potential interactions of an element as it fades in. The value increases with time, from 0 to 1, so when an element initially starts to fade in it does not interact with its neighbours and after the establishment time it interacts normally. By using this modulation of inter-element interactions, new elements can be made to steadily interact more and more strongly after they are created, starting with no interaction and eventually behaving normally. This allows for a slower relaxation to equilibrium and smoother growth.

3.2.6 Division

Cell division is handled by subroutine `scem_division` within module `ScEM_2_growth.f90`. This subroutine is built around a loop over the number of cells in the system before division, as shown in Code block 3.10. For each cell currently existing in the system, this loop checks whether the number of elements in the cell is greater than or equal to double the baseline number of elements per cell, `ne_cell`. If this is the case, division is triggered in this cell.

The first step in cell division is identifying the axis along which the cell will divide, which is done by finding the longest possible displacement vector between any two elements in the cell, thus identifying the longest axis of the cell. The cell is then divided by a plane perpendicular to this axis and passing through the centre of mass of the cell. Elements within the cell are identified as on either side of the plane by taking the dot product of their displacement from the centre of mass with the long-axis vector. Elements on one side of the plane are allocated to the new cell, while those on the other side of the plane remain part of the old cell. Finally, some book keeping sets the age of each daughter cell to zero an resets the internal element list, `c_elements`, for each cell.

```
do while (flag_success.eq.0)
  call ran_array(ra1,1,1,iseed)
  rn=ra1(1,1)
  m=1+int(rn*cells(k)%c_elements(0))
  n=cells(k)%c_elements(m)
  pos(:)=elements(n)%position(:)-cells(k)%position(:)
  rad_sq=dot_product(pos,pos)
  if (rad_sq.lt.r_core**2) then
    flag_success=1
    ne=ne+1
    call ran_array(ra2,1,2,iseed)
    phi=2*pi*ra2(1,1)
    theta=acos(2*(ra2(1,2)-0.5))
    d_pos(1)=0.6*r_equil*sin(theta)*cos(phi)
    d_pos(2)=0.6*r_equil*sin(theta)*sin(phi)
    d_pos(3)=0.6*r_equil*cos(theta)
    if (dim.eq.2) d_pos(3)=0.0
    elements(ne)%label=ne
    elements(ne)%parent=k
    elements(ne)%stage=0
    elements(ne)%type=1
    elements(ne)%age=0.0
    elements(ne)%strength=0.0
    elements(ne)%position(:)=cells(k)%position(:)+pos(:)+d_pos(:)
    elements(ne)%velocity(:)=0.0
    cells(k)%c_elements(0)=n_el_cell_k+1
    cells(k)%c_elements(n_el_cell_k+1)=ne
  end if
end do
```

Code Block 3.9 Protocol for placing a new element within cell k, from `ScEM_2_growth.f90`

```
do k=1,nc_old
  if (cells(k)%c_elements(0).ge.2*ne_cell) then
    nc=nc+1
    do i=2,cells(k)%c_elements(0)
      n=cells(k)%c_elements(i)
      do j=1,i-1
        nn=cells(k)%c_elements(j)
        pos(:)=elements(n)%position(:)-elements(nn)%position(:)
        max_sep=max(max_sep,dot_product(pos,pos))
        if (max_sep.gt.max_sep_old) then
          long_axis(:)=pos(:)
        end if
        max_sep_old=max_sep
      end do
    end do
    x_com(:)=cells(k)%position(:)
    c_el_temp1(:)=0
    c_el_temp2(:)=0
    do i=1,cells(k)%c_elements(0)
      n=cells(k)%c_elements(i)
      relative_pos(:)=elements(n)%position(:)-x_com(:)
      epsilon=dot_product(relative_pos,long_axis)
      if (epsilon.ge.0.0) then
        c_el_temp1(0)=c_el_temp1(0)+1
        c_el_temp1(c_el_temp1(0))=n
        elements(n)%parent=k
      else
        c_el_temp2(0)=c_el_temp2(0)+1
        c_el_temp2(c_el_temp2(0))=n
        elements(n)%parent=nc
      end if
    end do
    cells(k)%age=0.0
    cells(nc)%age=0.0
    cells(k)%c_elements(:)=c_el_temp1(:)
    cells(nc)%c_elements(:)=c_el_temp2(:)
  end if
end do
```

Code Block 3.10 Algorithm for cell division, from `ScEM_2_division.f90`.

```
allocate(potential_deriv(0:n_bins-1,2))
allocate(pot_deriv_table(0:n_bins))
allocate(r_sq_table(0:n_bins))
do j=0,n_bins
   sep_sq=j*d_r_sq
   r_sq_table(j)=sep_sq
   factor=exp(rho*(1.0-sep_sq/r_equil_sq))
   pot_deriv_table(j)=force_amplitude*factor*(factor-1.0)
end do
do j=0,n_bins-1
   potential_deriv(j,1) = (pot_deriv_table(j+1) -
    ↪  pot_deriv_table(j))*d_r_sq_recip
   potential_deriv(j,2) = (pot_deriv_table(j)*r_sq_table(j+1) -
    ↪  pot_deriv_table(j+1)*r_sq_table(j))*d_r_sq_recip
end do
potential_deriv=potential_deriv/damping_element
```

Code Block 3.11 Defining Morse potential for inter-element interactions, from `ScEM_1_potential.f90`.

3.2.7 Interaction Potential

In the basic SEM, only one potential is defined. This takes the form of a morse potential [194] for diatomic interactions, with repulsion at small separation and steadily decaying attraction for greater separation [Figure 3.2]. This potential is used for all interactions in the system: both inter-cellular interactions and intra-cellular interactions. However, the magnitude of these interactions is varied by means of a coefficient that differs for inter- and intra- cellular interactions. Values for this coefficient are stored in the `rel_strength` array, defined in `scem_input`. Initially value are set so that inter-cellular interactions have half the magnitude of intra-cellular interactions.

Within the basic SEM program, the potential is defined by the subroutine `scem_potential` in module `ScEM_1_potential.f90` [Code block 3.11]. This subroutine is called once at the beginning of the program and stored the potential data in the `potential_deriv` array, meaning that the potential can be looked up rather than recalculated at later stages of the program.

Once `potential_deriv` has been defined it is used within subroutine `scem_integrate`. This subroutine uses a loop over all pairs in the system to calculate the velocity of each element in the system from its interaction pairs, position, `strength` value, and the `potential_deriv` array [Code block 3.12]. Once this velocity has been calculated, it can be used to update the position of the element, as discussed in Subsection 3.2.8

```
if (sep_sq.le.r_interaction_max_sq) then
  fadein_amp = elements(n)%strength*elements(nn)%strength
  bin = int(sep_sq*d_r_sq_recip)
  r_s = fadein_amp*rel_strength(1, 1, elements(n)%type,elements(nn)%type,
   ↪  index_intra)
  pot_deriv_interp=r_s*(sep_sq*potential_deriv(bin, 1) +
   ↪  potential_deriv(bin, 2))
  elements(n)%velocity(:) = elements(n)%velocity(:) +
   ↪  dx(:)*pot_deriv_interp
  elements(nn)%velocity(:) = elements(nn)%velocity(:) -
   ↪  dx(:)*pot_deriv_interp
end if
```

Code Block 3.12 Code block within loop over all interaction pairs to interpolate force and calculate velocities of elements `n` and `nn`, from `scem_integrate`.

3.2.8 Updating

The SEM elements exhibit over-damped Langevin dynamics [186]. Equation 3.2 shows the Langevin equation [196] for particle dynamics, where $\underline{x}$ is the position of the particle, U is the potential field experienced by the particle, γ is a drag coefficient, and ζ is the stochastic component.

$$m\ddot{\underline{x}} = -\nabla U(\underline{x}) - \gamma\dot{\underline{x}} + \underline{\zeta} \tag{3.2}$$

In the over-damped regime, where $m \to 0$, Equation 3.2 can be simplified to Equation 3.3.

$$\gamma\dot{\underline{x}} = -\nabla U(\underline{x}) + \underline{\zeta} \tag{3.3}$$

Thus, the velocity of each element can be calculated by summing the gradients of each inter-element potential that the element experiences. Note that for our purposes the magnitude of ζ was found to not have a significant effect on system dynamics and was thus neglected in our simulations.

Updating the system is handled by subroutine `scem_iterate` in module `ScEM_3_iterate.f90`. This subroutine contains a `while` loop that performs a set of commands and increments the system time until the max time is reached. The loop calls subroutines `scem_flag_relist`, `scem_relist`, `scem_pairs`, `scem_ageing`, `scem_growth`, `scem_division`, `scem_resize`, `scem_com`, and `scem_output`, which all update stored information according to the movement of the elments. The loop also calls `scem_integrate`

```
call scem_integrate
forall(n=1:ne) elements(n)%position(:) = elements(n)%position(:) +
 ↪  0.5*dt*elements(n)%velocity(:)
forall(n=1:ne) elements(n)%velocity(:) = 0.0
call scem_integrate
forall(n=1:ne) elements(n)%position(:) = xe_prev(n,:) +
 ↪  dt*elements(n)%velocity(:)
forall(n=1:ne) elements(n)%velocity(:) = 0.0
```

Code Block 3.13 Terms to implement a 2nd order Runge-Kutta and update element positions, from the `while` loop in `ScEM_3_iterate.f90`

to calculate element velocities, and applies these velocities to update the element positions, as shown in Code block 3.13.

Finally, the loop within `scem_integrate` calls `scem_diffusion`, which handles the stochastic component of the Langevin equation by applying a random Gaussian change to the position of each element.

3.2.9 Data Output

Outputting data from the simulation is handled by `scem_output` and `scem_output_final` in modules `ScEM_2_output.f90` and `ScEM_2_output_final.f90`.

`scem_output` outputs data as the simulation progresses. Data is written to file at specified output time intervals, `time_out_1` and `time_out_2`. The positions of all elements in the system and the positions of all cells in the system are written to files, and a summary of the current system state is printed to the command line. The `elements` file stores the positions of all elements at all output intervals, while a separate file of the form `fort.xx` where xx is a number between 30 and 40 is created for each output interval to store the element positions at that point.

`scem_output_final` is called only once the simulation has finished, and saves the final positions of all elements to the `elements_final` file and the number of cells, elements, and pairs to the `end_of_run_data` file.

3.2.10 Auxiliary Scripts

The initial SEM program came with 3 auxiliary shell scripts to perform simple housekeeping tasks. `ScEM_script_intel_compile` calls the fortran compiler to precompile all modules separately and then combine into the `ScEM_master` binary. `ScEM_script_create_to_exist`

takes the `elements_final` and `end_of_run_data` datafiles produced by a run of the program and converts them to the `elements_config` and `config_data` files needed to run the final state of this system as the initial state of a new system with the `ScEM_2_initial_exist.f90` module. Finally, `ScEM_script_clear_data` removes all datafiles produced by a run of the program.

Chapter 4

Implementation of a Model of Cell Sorting with the Subcellular Element Method

The basic SEM program is a powerful tool, and we have already seen how the program has been adapted and augmented to model other systems [190]. However, it lacks specific features required for our investigation of tissue self-organisation in the inner cell mass. This chapter outlines updates made to the program as part of this project.

4.1 Cell Lineages

The first thing to consider is that we require two cell types in the system. This will allow us to identify two separate populations of cells within the simulations and give them different properties to study the effect on their organisation. To achieve this, we added another component to the `cell` data structure, which we called `fate` [Code block 4.1]. This component can takes integer values of 1 to indicate an epiblast cell or 2 to indicate a primitive endoderm cell. The use of an integer value rather than boolean leaves open the potential for introducing any number of additional cell types in future, such as trophoblasts. Adding this component to the data structure was straightforward but required a significant number of downstream alterations in the code before it became useful. Because of this, the relevance and use of the `fate` component will be clearer after we have discussed additional augmentation of the code.

In order to allocate fates to new cells after division, Code block 4.2 was added to subroutine `scem_division`. We also introduce `flag_symmetric_division`. If this variable is equal to 1, division is symmetric, meaning cells of type 1 divide to produce two daughter

```
type cell
  integer :: label
  integer :: fate
  integer :: triplet_count
  real*8  :: rad_gyration
  real*8  :: age
  real*8  :: volume
  real*8, dimension(3) :: position
  real*8, dimension(3) :: original_position
  integer, dimension(0:4*ne_cell)      :: c_elements
  integer, dimension(0:2*ne_cell)      :: cortex_elements
  integer, allocatable, dimension(:,:) :: triplets
end type cell
```

Code Block 4.1 Updated `cell` data structure definition. Contains a number of updates to the original definition that will be outlined in subsections of this chapter.

cells of type 1, and cells of type 2 divide to produce two daughter cells, also both of type 2. Otherwise, division is asymmetric, and an equal probability is given to producing two daughter cells of the same type as the parent or of different types. It is never possible for a parent cell to produce two daughter cells both of the other fate. The algorithm uses a random number to choose between these two equally likely outcomes. This means that after division there will always be at least one cell of the same fate as the parent, and an equal probability of the other daughter cell having either fate. In real cellular systems these relative probabilities for cell fates may be different, which in itself might naturally lead to a form of aggregation of each cell type, but for the purposes of proving minimal requirements for sorting by physical mechanisms, this 50:50 probability is an adequate starting point.

As a further level of detail, we could take into account the plasticity of cell fates in the ICM [41] by treating fate as a vector in a 2D or 3D space. If the vector points along one axis then a firm fate decision has been made, but anywhere in between would reflect the plasticity of the cell fate, with the component of the vector in each dimension reflecting the extent to which the cell is tending towards that fate. This would allow us to model factors influencing the fate decisions of the cells in the inner cell mass, and how they might influence sorting, but this was not implemented in the current project and will have to wait for later work.

```
if (flag_symmetric_division.EQ.1.OR.intro) then
  cells(nc)%fate=cells(k)%fate
  cells(nc)%label=nc
else
  CALL RANDOM_NUMBER(fate_decider)
  if (fate_decider.GE.0.5) then
    cells(nc)%fate=cells(k)%fate
  else
    cells(nc)%fate=MOD(cells(k)%fate,2)+1
  endif
endif
```

Code Block 4.2 Algorithm for allocating fates to daughter cells after division, with a 50:50 chance of symmetric or asymmetric division.

4.2 Boundary and External Pressure

The inner cell mass is bounded by trophoblasts and the blastocoele. This means it will be necessary to introduce a boundary into our model so that the cells are contained within a limited space. This will eventually allow us to study the effect of geometry on sorting.

After considering some other possible mechanisms, the method for introducing a boundary to the system that we eventually implemented was to introduce a system-wide potential field that acted on any element that strayed from a specific region, and applied a returning force to these elements to keep them within the system boundary. This can be thought of as a potential well, within which the elements do not directly experience a force from the boundary.

The algorithm is implemented as subroutine `scem_background1` within `ScEM_2_background.f90`. This subroutine is shown in Code block 4.3. The first step is to calculate the total volume of all cells in the system. This total volume is used to define the volume of the spherical potential well. Once the total volume has been found, the radius of the potential well is set to 120% of the radius of a sphere with the volume of the sum of all cell volumes. This ensures that there is only just enough room within the boundary for the system. Once this radius has been calculated, a loop is performed over all elements to check whether any elements lie outside this radius from the centre of the system. Any element that meets this condition experiences a constant force directed radially towards the centre of the system. The magnitude of this force scales with the magnitude of other forces in the system.

```
subroutine scem_background1
  real*8  :: spherical_radius
  real*8  :: volume_sum
  integer :: n
  volume_sum = 0
  do n=1, nc
    volume_sum = volume_sum + cells(n)%volume
  enddo
  spherical_boundary_radius =
   ↪  1.2*(((3.0*volume_sum)/(pi*4.0))**(1.0/3.0))
  do n=1, ne
    spherical_radius =
     ↪  DOT_PRODUCT(elements(n)%position,elements(n)%position)
    if (spherical_radius.gt.spherical_boundary_radius) then
      elements(n)%velocity(:) = elements(n)%velocity(:) - 0.1*
 ↪  stiffness_factor*elements(n)%position(:)/spherical_radius
    endif
  enddo
end subroutine
```

Code Block 4.3 Subroutine for applying external boundary pressure due to a spherical boundary, from `ScEM_2_background.f90`

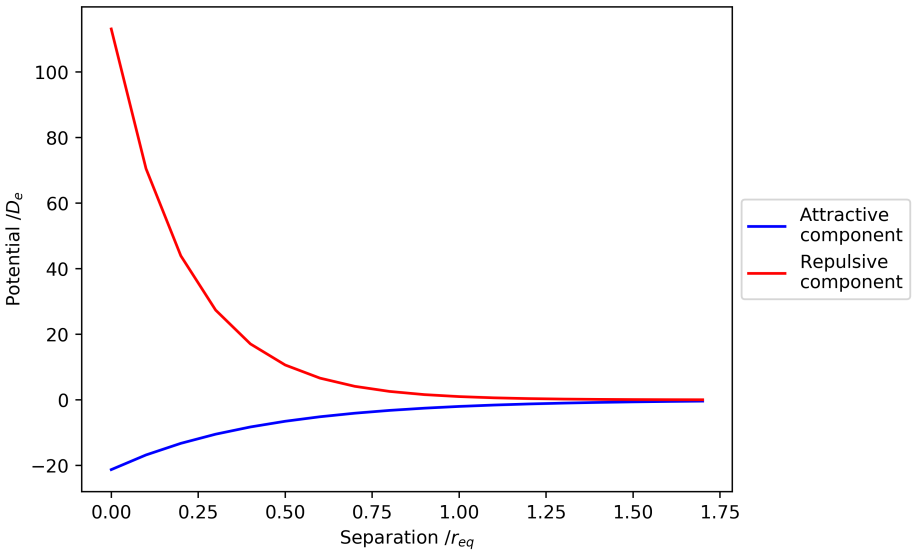


Fig. 4.1 Separate plots of the attractive and repulsive components of the Morse potential.

We also created a second background potential in the form of a spherical cap, to better represent the shape of the inner cell mass, using a similar algorithm to that of the spherical boundary. However, for most implementations of the model so far, we used the spherical boundary, since sorting is also observed in spherical aggregates of inner cell mass cells that have been removed from the blastocyst.

The boundary subroutine is controlled by the value of `flag_background` within module `ScEM_0_input.f90`, which allows the boundary to be turned off as necessary.

4.3 Interaction Potentials

In order to more carefully control the behaviour of our simulations, we decided to expand upon the standard nearest neighbour inter-cell potentials. The standard Morse potential [Chapter 3] can be broken into an attractive component and repulsive component [Figure 4.1].

By defining these components separately, we are able to change the relative magnitudes of the two components in interactions and thus tune the interactions between elements depending on their type. For example, once a cortex has been defined [Section 4.4], we can change the magnitude of the attractive component for inter-cell cytoplasm-cytoplasm interactions to be zero, so that the internal elements of different cells are purely repulsive. This helps to prevent cells from overlapping in space, a problem that was observed with the basic SEM. To implement this splitting of potential components, we changed the definitions

```
!For attractive potential
do j=0,n_bins
   sep_sq = j*d_r_sq
   r_sq_table(j) = sep_sq
   factor = exp(rho*(1.0-sep_sq/r_equil_sq))
   pot_deriv_table(j) = -force_amplitude*factor
end do
do j=0,n_bins-1
   potential_deriv1(j,1) = (pot_deriv_table(j+1) -
    ↪  pot_deriv_table(j))*d_r_sq_recip
   potential_deriv1(j,2) = (pot_deriv_table(j)*r_sq_table(j+1) -
    ↪  pot_deriv_table(j+1)*r_sq_table(j))*d_r_sq_recip
end do
potential_deriv1 = potential_deriv1/damping_element
!For repulsive potential
do j=0,n_bins
   sep_sq = j*d_r_sq
   r_sq_table(j) = sep_sq
   factor = exp(rho*(1.0-sep_sq/r_equil_sq))
   pot_deriv_table(j) = force_amplitude*factor**2
end do
do j=0,n_bins-1
   potential_deriv2(j,1) = (pot_deriv_table(j+1) -
    ↪  pot_deriv_table(j))*d_r_sq_recip
   potential_deriv2(j,2) = (pot_deriv_table(j)*r_sq_table(j+1) -
    ↪  pot_deriv_table(j+1)*r_sq_table(j))*d_r_sq_recip
end do
potential_deriv2 = potential_deriv2/damping_element
```

Code Block 4.4 Definitions of attractive and repulsive potential components in `scem_potential`.

in subroutine `scem_potential`. Rather than defining one array `potential_deriv`, we now define two arrays `potential_deriv1` for the attractive component and `potential_deriv2` for the repulsive component [Code block 4.4]. A sum of these two arrays recreates the original `potential_deriv` array, but we can now also combine them with different magnitudes.

Once these two components have been defined, we can use them to calculate a velocity [Code block 4.15]. Note that we now use an expanded `rel_strength` array with 6 indices, allowing a different value for each possible set of element types and cell types in the interaction (indices 2, 3, 4, and 5), inter- and intra-cell interactions (index 6), and for attractive and repulsive components (index 1). The `rel_strength` array is defined within `scem_input` and its components are user-specified.

The number of parameters needed to specify all components in the array quickly drops once we consider various symmetries of the possible combinations of indices. Some values must by definition be zero, such as intra-cell interactions with different cell type values, which is impossible. Others are defined to be zero, such as the adhesive components of all inter-cell interactions involving internal cytoplasm elements. Both adhesive and repulsive components of all intra-cell interactions (cortex-cytoplasm, cytoplasm-cytoplasm, cortex-cortex) are given the same magnitude, specified in `scem_input` as `stiffness_factor`. This component also vanishes as an input parameter when we consider that all components scale together, and thus we can set it to be equal to one and treat all other components as ratios thereof. Meanwhile, inter-cell repulsive components are simply given magnitudes high enough to reliably prevent cells from overlapping in space. Finally, the interesting components are those three related to inter-cell cortex-cortex adhesion. The 3 components relate to epiblast-epiblast, epiblast-primitive endoderm, and primitive endoderm-primitive endoderm adhesion. By considering that adhesion magnitude is related to the density of cadherin molecules on the surface of a cell, we make the assumption that the epiblast-primitive endoderm adhesion has the same magnitude as the weakest of epiblast-epiblast and primitive endoderm-primitive endoderm adhesion. Thus we need only focus on two parameters, `epi_adhesion` and `pre_adhesion`. Given experimental evidence concerning adhesion of epiblasts and primitive endoderm, we can make the further assumption that both cell types have the same mutual adhesion, and thus reduce the parameters further to only a single adhesion magnitude.

4.4 Defining a Cortex

The actomyosin cortex is one of the crucial structures that allow a cell to interact with its environment; in Chapter 1 we discussed differential interfacial tension as a mechanism that could drive cells to self-organise and its potential importance in the inner cell mass.

```
type element
  integer :: label
  integer :: parent
  integer :: stage
  integer :: type
  integer :: DIT_factor
  real    :: adhesion_factor
  real*8  :: age
  real*8  :: strength
  real*8, dimension(3) :: position
  real*8, dimension(3) :: velocity
  real*8, dimension(3) :: polar
end type element
```

Code Block 4.5 New `element` data structure definition.

Furthermore, we also proposed that creation of new cortex in cell division and variation in tension due to blebbing could be significant factors in perturbing the system to facilitate rearrangement. In order to include these mechanisms in our model, it is vital that there be some force acting over the surface of SEM cells. This force must also allow for local variation in its magnitude according to the local context. We decided to model the cortex as an additional force acting between surface elements in each SEM cell.

The first step towards the objective of a cell cortex is to identify surface elements within each cell, which is not entirely trivial. We require a distinct outer layer of elements with different properties in each cell. For this we begin by using the `type` component of the element data structure to define a `type` that labels the elements that make up the actin cortex. This component can take values of 1, corresponding to cytoplasm or bulk elements, or 2, corresponding to cortex elements. Once these elements have been suitably labelled, we can distinguish them from other elements to alter their behaviour. Thus, the `element` data structure is updated as shown in Code block 4.5. Note that there are a number of other new components here to which we will return later.

After considering a number of possibilities for algorithms to identify cortex elements, we settled on the procedure outlined below. We begin by calculating the spherical polar coordinates of each element in each cell relative to the centre of mass of the parent cell. For this we created a module `ScEM_2_polar.f90` to apply this calculation to all elements in the system whenever subroutine `scem_polar` is called [Code block 4.6]. These polar coordinates are stored as a 3 vector in the newly defined `polar` component of the `element` data structure [Code block 4.5].

```
do i=1, ne
  r_vector(:) = elements(i)%position(:) -
   ↪  cells(elements(i)%parent)%position(:)
  r_squared = r_vector(1)**2 + r_vector(2)**2 + r_vector(3)**2
  elements(i)%polar(1) = sqrt(r_squared)
  cos_theta = r_vector(3)/elements(i)%polar(1)
  elements(i)%polar(2) = ACOS(cos_theta)
  elements(i)%polar(3) = pi+ATAN2(r_vector(2),r_vector(1))
end do
```

Code Block 4.6 Loop over all elements in the system to calculate and store their polar coordinates relative to the cell centre of mass of their parent cell, from `ScEM_2_polar.f90`

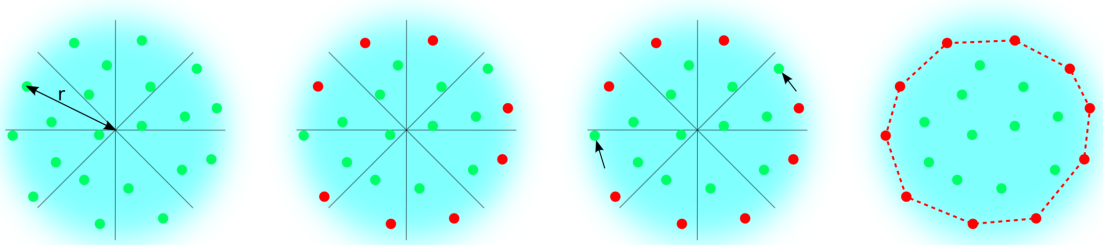


Fig. 4.2 Sequence demonstrating the algorithm for allocating cortex elements.

Once polar coordinates have been found, it is straightforward to divide the cell into 32 bins - 4 in the polar angle and 8 in the azimuthal angle - and allocate each element to a bin. The element radially furthest from the cell centre of mass in each bin is allocated cortex type (`elements(n)%type=2`). Any other elements in the bin with a radius greater than 80% are also given cortex type. This radius corresponds to roughly 50% of the volume of the cell.

In order to store information about the cortex elements in each cell, we updated the `cell` data structure as shown in Code block 4.7 by adding the `cortex_elements` array as a new components of the data structure. `cortex_elements` is similar to the original `c_elements` array in that the 0th components stores the number of cortex elements in the cell, and the remaining elements of the array store the labels of the cortex elements within the SEM cell. Note that there are other new additions to the `cell` data structure, to which we will return later.

Cortex allocation is performed at each timestep by the subroutine `scem_cortex` in module `ScEM_4_cortex.f90`. The main section of this subroutine is shown in Code block 4.8. The subroutine begins by calling subroutine `scem_polar` [Code block 4.6] to calculate the polar

```
type cell
  integer :: label
  integer :: fate
  integer :: triplet_count
  real*8  :: rad_gyration
  real*8  :: age
  real*8  :: volume
  real*8, dimension(3) :: position
  real*8, dimension(3) :: original_position
  integer, dimension(0:4*ne_cell)      :: c_elements
  integer, dimension(0:2*ne_cell)      :: cortex_elements
  integer, allocatable, dimension(:,:) :: triplets
end type cell
```

Code Block 4.7 Updated definition of the `cell` data structure, from `ScEM_1_types.f90`.

coordinates of each element relative to the centre of mass of its parent cell. We then set the `type` component of each element in the cell to 1. This refreshing of the state at each element prevents a net accumulation of cortex elements over time. Subsequently, there is a loop over all cells `nc`, within which the cortex elements for each cell are allocated. Within this loop we introduce three new arrays, defined within module `ScEM_0_arrays.f90`: `bin_counters`, `bin_contents`, and `bin_max_radius`. For each of these arrays, the first dimension has 32 elements, corresponding to the 32 pyramid bins of the cell. Arrays `bin_counters` and `bin_max_radius` store the number of elements and the maximum radius of those elements for each corresponding pyramid bin. Array `bin_contents` has a second dimension of size 100, and stores the full list of element labels within each bin. These three arrays are used in a loop over all elements within a cell to determine the contents and maximum radius of each polar bin. Finally, another loop over all polar bins determines which elements within the corresponding slice of `bin_contents` have a radius greater than 80% of the maximum radius. All such elements are allocated cortex type. This algorithm is summarised in Figure 4.2 and an example of a single cell with cortex elements highlighted is shown in Figure 4.3

4.5 Introducing Tension in Cortex

Once the set of cortex elements has been established, it is necessary to introduce a force that acts only between these elements. This will allow us to model cortical tension as a tangential tension force acting throughout the boundary of each cell. The first attempt to do

```
call scem_polar
FORALL(n=1:ne) elements(n)%type = 1
do i=1, nc
  cells(i)%cortex_elements(:)=0
  bin_counters(:)   = 0
  bin_contents(:,:) = 0
  bin_max_radius(:) = 0
  do l=1, cells(i)%c_elements(0)
    n = cells(i)%c_elements(l)
    j = int(elements(n)%polar(2)/(pi/4)) + 1
    k = int(elements(n)%polar(3)/(pi/4))
    bin_counters(j+k) = bin_counters(j+k) + 1
    bin_contents(j+k,bin_counters(j+k)) = n
    bin_max_radius(j+k) = MAX(bin_max_radius(j+k), elements(n)%polar(1))
  end do
  do k=1,32
    do m=1, bin_counters(k)
      n=bin_contents(k,m)
      if (elements(n)%polar(1).GT.(0.8*bin_max_radius(k))) then
        elements(n)%type = 2
        cells(i)%cortex_elements(0)=cells(i)%cortex_elements(0)+1
        cells(i)%cortex_elements(cells(i)%cortex_elements(0)) = n
      end if
    end do
  end do
end do
```

Code Block 4.8 Algorithm for allocating cortex elements, forming the first section of subroutine `scem_cortex` in module `ScEM_4_cortex.f90`.

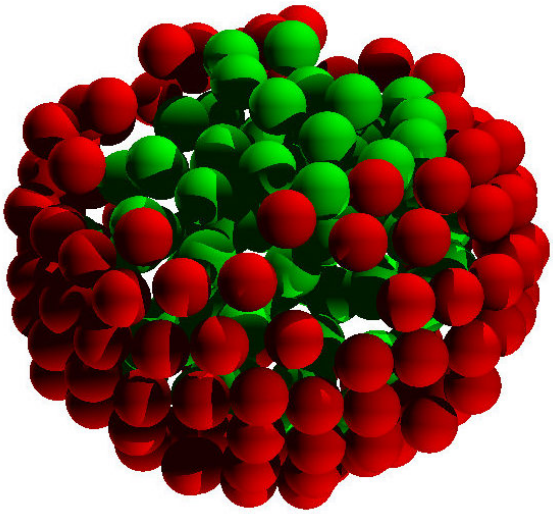


Fig. 4.3 Cutaway PovRay image of one SEM cell showing cortex elements as red spheres and cytoplasm elements as green spheres.

this involved simply identifying existing nearest-neighbour interactions that happen to act between elements that are identified as cortex elements, and then increasing the magnitude of these interactions. This did not work well at all, as it quickly became clear that the spacing between cortex elements was such that very few were within the minimum distance for standard nearest-neighbour interactions. This problem can be seen in Figure 4.4, which shows the inter-element interaction pairs of one typical SEM cell. Those interactions acting between two cortex elements are coloured red; all others are blue. Clearly this network of cortex-cortex interactions is not adequate to span the surface of the cell.

This algorithm was particularly problematic immediately after division. A crucial factor in our model is the introduction of energy by the creation of new cortex during division. However, under this protocol, the separation of cortex elements over the flat cell surface at the interface between the two daughter cells was significantly too large to produce any cortex-cortex interactions across the surface. Thus, no energy could be introduced following division, inhibiting the ability of the system to rearrange. Furthermore, occasionally boundary elements could become detached from a cell. If they happened to move far enough away from the other elements in the cell, they could exceed the maximum distance for inter-element nearest neighbour interactions, and thus have no forces keeping them attached to the cell. An example of this is shown in Figure 4.5, in which cortex elements are shown as red spheres, cytoplasm elements are shown as green spheres, and nearest neighbour interactions are shown

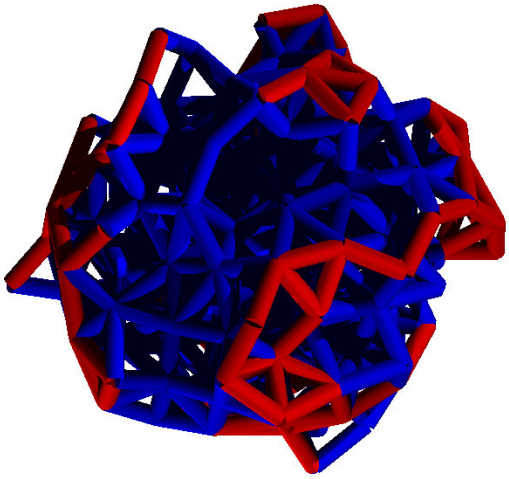


Fig. 4.4 Visualisation of a single SEM cell, showing all nearest-neighbour element interactions as cylinders, with those acting between cortex elements coloured red. It can clearly be seen that the standard interactions between cortex elements are insufficient to span the surface.

as blue cylinders. One highlighted element has no nearest neighbour interactions and is at risk of being separated from the cell.

4.5.1 Delaunay Triangulation Over Cortex Elements

The issues with existing interactions as a means of producing cortical tension can be solved by introducing a new set of interactions that act between cortex elements regardless of their separation from one another. Thus the cortex network is guaranteed to span the surface of the cell and it will be impossible for elements to be lost as there will always be a returning force. To achieve this, we perform a Delaunay triangulation [197] over the set of cortex elements. The triangulation identifies a network of nearest neighbour connections within the set of cortex elements by dividing the elements into a network of triangles whilst maximising the value of the minimum angle within each triangle. Thus the triangulation favours roughly equilateral triangles with no unusually long edges, which produces a smooth surface across the whole of the cell. Any nearest neighbour connection between cortex elements - an edge of any triangle in the Delaunay triangulation - is used to introduce a constant force acting between those two cortex elements, parallel to the triangle edge [Figure 4.6].

To implement this method, we began by introducing a new data structure, defined within the module `ScEM_1_types.f90`: the `cortexpair` data structure [Code block 4.9], which is used to store information about all cortex-cortex interactions defined by the Delaunay

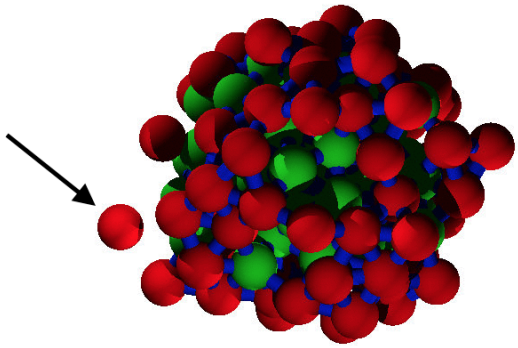


Fig. 4.5 One SEM cell, showing an element that has moved too far from other elements and lost its nearest neighbour interactions.

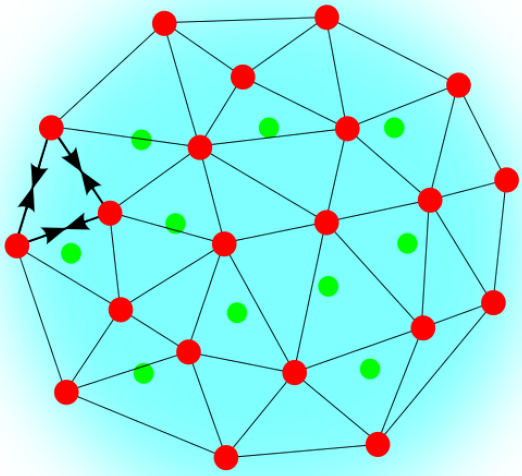


Fig. 4.6 Two dimensional diagram of cortical tension forces as defined by a Delaunay triangulation in one SEM cell. Cytoplasm elements are green and cortex elements are red; the Delaunay triangulation over the cortex elements is shown, with one set of cortex forces indicated by arrows.

```
type cortexpair
  integer :: label1
  integer :: label2
  real*8  :: cortex_factor
end type cortexpair
type(cortexpair), allocatable, dimension(:) :: pairs_cortex
```

Code Block 4.9 Definition of the `cortexpair` data structure, and the `pairs_cortex` array to store the newly defined data structures.

triangulation. This data structure has 3 components: the labels of the two elements involved in the cortex-cortex interaction, and the `cortex_factor` value, to which we will return later. We also added two components to the `cell` data structure: the `triplet_count` component is an integer that stores the number of triangles in the Delaunay triangulation of this cell, and the `triplets` array stores the labels of all three elements within each triangle of the Delaunay triangulation [Code block 4.7]. So for example, `cells(n)%triangles(3,i)` is the 3rd element in the ith cortex triangle of the nth cell.

Having defined this new data type, the Delaunay triangulation is performed by calling subroutine `scem_delaunay` in subroutine `scem_cortex` [Code block 4.10]. Thus the Delaunay triangulation is refreshed at each timestep. Subroutine `scem_delaunay` [Code block 4.11] performs the Delaunay triangulation with a loop over all cells in the system, for each cell setting up a number of data arrays using the `cortex_elements` data from that cell. The `x`, `y`, and `z` arrays store the x, y, and z Cartesian coordinates of all cortex elements relative to the cell centre of mass, with the radius normalised to unity from the centre of mass, thus ensuring that all values fall on a sphere. The fundamental calculations that produce the Delaunay triangulation are performed using subroutines from the Stripack algorithm published by Robert Renka [198]. These routines were tidied into several separate modules, and the `scem_delaunay` subroutine acts as an interface between the SEM program and the Stripack routines, passing data from the SEM in the format required by the Stripack subroutines. For our purposes the significant subroutines are `trmesh` and `trlist2`. When these two subroutines are called back to back, they produce two objects that are of use to us: the first is `nt`, which is the number of triangles used to span the surface of the cell, and is immediately stored in `cells(j)%triplet_count=nt`; the second is `ltri`, which is a two dimensional array listing the elements in all triangles. So for example, `ltri(1,i)` gives the label of the 1st cortex element in the ith surface triangle. However, the labels used in `ltri` are simply the indices of these elements within the `cells(j)%cortex_elements` array, so in order for these to be useful

```
call scem_delaunay
if (allocated(pairs_cortex)) deallocate(pairs_cortex)
allocate(pairs_cortex(np_cortex))
pair_counter=0
do i=1, nc
  do j=1, cells(i)%triplet_count
    pair_counter = pair_counter+1
    pairs_cortex(pair_counter)%label1 = cells(i)%triplets(1,j)
    pairs_cortex(pair_counter)%label2 = cells(i)%triplets(2,j)
    pair_counter = pair_counter+1
    pairs_cortex(pair_counter)%label1 = cells(i)%triplets(2,j)
    pairs_cortex(pair_counter)%label2 = cells(i)%triplets(3,j)
    pair_counter = pair_counter+1
    pairs_cortex(pair_counter)%label1 = cells(i)%triplets(3,j)
    pairs_cortex(pair_counter)%label2 = cells(i)%triplets(1,j)
  enddo
enddo
pairs_cortex(:)%cortex_factor = 1
if (.NOT.intro) call scem_dit
```

Code Block 4.10 Algorithm for creating the `pairs_cortex` arrary, forming the second part of subroutine `scem_cortex`.

we must convert them to the global labels of these elements. So for example, the global label of the element referenced by `ltri(1,i)` would be `cells(j)%cortex_elements(ltri(1,i))`.

Once the Delaunay triangulation has been performed, the final step of the `scem_cortex` subroutine is to fill the `pairs_cortex` array with details from the Delaunay triangulations of all cells. This array is of data type `cortexpair` [Code block 4.9], meaning that each element of the array is a 3 component data structure. For each cell, a loop over all `triplets` array components fills the `label1` and `label2` components of 3 elements of the `pairs_cortex` array with the two element labels for each edge of each surface triangle. In other words, each surface triangle in each cell has 3 edges corresponding to 3 cortex-cortex interactions and thus takes up 3 components of the `pairs_cortex` array. Thus, the `pairs_cortex` array contains all cortex pairs from all cells, and the value of `pair_counter` tells us exactly how many such edges there are across all cells. Note that because each cortex element pair defined by the triangulation forms an edge of precisely two triangles, each pair is counted twice. In practice this does not matter since we can simply compensate with a smaller tension magnitude, which is easier and quicker than preventing double counting. The third component of each element of the `pairs_cortex` array is the `cortex_factor` component, to which we will return in Section 4.6.

Figure 4.7 shows the result of this algorithm: one SEM cell in which the edges of the Delaunay triangulation over the surface elements are shown by red cylinders.

4.5.2 Applying Tension Forces Within Triangulation

Now that there is more than one inter-element interaction to consider, we need to restructure the element velocity calculations previously handled by `scem_integrate`. The machinery for calculating velocities due to nearest-neighbour inter-element forces, as originally defined in the basic SEM, was moved into subroutine `scem_near_neighbour_update` in module `ScEM_3_near_neighbour_update.f90` [Code block 4.15]. A new subroutine, `scem_cortical_tension_update`, was created in module `ScEM_3_cortical_tension_update.f90` to handle calculations related to cortical tension forces. Due to the linearity of the overdamped Langevin equations used in SEM, we can calculate velocity components due to these forces separately and then sum the components to calculate a final velocity. Thus `scem_integrate` now contains nothing but 3 subroutine calls to `scem_near_neighbour_update`, `scem_cortical_tension_update`, and `scem_background`, the three subroutines that each calculate a velocity component for every element in the system due to nearest neighbour interactions, cortical tension, and boundary pressure respectively.

The important details of the newly created subroutine `scem_cortical_tension_update` are shown in Code block 4.12. The subroutine is built around a loop over all cortex-cortex

```
allocate(x(cells(j)%cortex_elements(0)))
allocate(y(cells(j)%cortex_elements(0)))
allocate(z(cells(j)%cortex_elements(0)))
do i=1, cells(j)%cortex_elements(0)
  k = cells(j)%cortex_elements(i)
  r_vector = elements(k)%position-cells(j)%position
  radius_squared  = DOT_PRODUCT(r_vector,r_vector)
  radius          = sqrt(radius_squared)
  x(i)            = r_vector(1)/radius
  y(i)            = r_vector(2)/radius
  z(i)            = r_vector(3)/radius
end do
allocate(list(6*(cells(j)%cortex_elements(0)-2)))
allocate(lptr(6*(cells(j)%cortex_elements(0)-2)))
allocate(lend(6*(cells(j)%cortex_elements(0)-2)))
allocate(near(cells(j)%cortex_elements(0)))
allocate(next(cells(j)%cortex_elements(0)))
allocate(dist(cells(j)%cortex_elements(0)))
allocate(ltri(3,(2*cells(j)%cortex_elements(0)-4)))
call trmesh ( cells(j)%cortex_elements(0), x, y, z, list, lptr, lend,
 ↪  lnew, near, next, dist, ier )
call trlist2 ( cells(j)%cortex_elements(0), list, lptr, lend, nt, ltri,
 ↪  ier )
if (allocated(cells(j)%triplets)) deallocate(cells(j)%triplets)
allocate(cells(j)%triplets(3,(2*cells(j)%cortex_elements(0)-4)))
do i=1, nt
  cells(j)%triplets(1,i)=cells(j)%cortex_elements(ltri(1,i))
  cells(j)%triplets(2,i)=cells(j)%cortex_elements(ltri(2,i))
  cells(j)%triplets(3,i)=cells(j)%cortex_elements(ltri(3,i))
end do
cells(j)%triplet_count = nt
np_cortex = np_cortex + 3*nt
```

Code Block 4.11 Procedure within `scem_delaunay` for finding the Delaunay triangulation over the cortex elements of one cell. This block is contained within a loop over all cells, in module `ScEM_3_delaunay.f90`, and contains subroutines `trmesh` and `trlist2` from modules `trmesh_module.f90` and `trlist2_module`.

```
do m=1,np_cortex
  n=pairs_cortex(m)%label1
  nn=pairs_cortex(m)%label2
  dx(:)=elements(n)%position(:)-elements(nn)%position(:)
  sep_sq=dot_product(dx,dx)
  dx(:)=dx(:)/sqrt(sep_sq)
  if(cells(elements(n)%parent)%fate.EQ.1) then
    elements(n)%velocity(:) = elements(n)%velocity(:) -
 ↪  dx(:)*cortex_constant1*pairs_cortex(m)%cortex_factor
    elements(nn)%velocity(:)= elements(nn)%velocity(:)+
     ↪  dx(:)*cortex_constant1*pairs_cortex(m)%cortex_factor
  else
    bleb_factor_n  = 1.0
    bleb_factor_nn = 1.0
    if (flag_pre_blebbing.EQ.1.AND..NOT.intro) then
      if (elements(n)%DIT_factor.EQ.0)  bleb_factor_n = 1.0 +
 ↪  bleb_amp*SIN(10*2.0*pi*elements(n)%age/cell_cycle_time)
      if (elements(nn)%DIT_factor.EQ.0) bleb_factor_nn= 1.0 +
       ↪  bleb_amp*SIN(10*2.0*pi*elements(nn)%age/cell_cycle_time)
    endif
    elements(n)%velocity(:) = elements(n)%velocity(:) -
     ↪  dx(:)*cortex_constant2*
     ↪  pairs_cortex(m)%cortex_factor*bleb_factor_n
    elements(nn)%velocity(:)= elements(nn)%velocity(:)+
     ↪  dx(:)*cortex_constant2*
     ↪  pairs_cortex(m)%cortex_factor*bleb_factor_nn
  endif
end do
```

Code Block 4.12 Algorithm for calculating element velocity component due to cortical tension, from module `ScEM_3_cortical_tension_update.f90`.

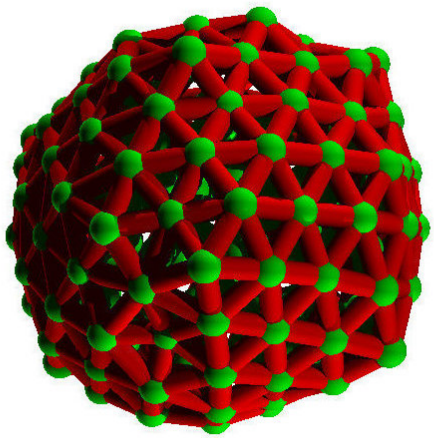


Fig. 4.7 PovRay image of one SEM cell, with elements shown as green spheres and the Delaunay triangulation, and hence cortical tension forces, shown as red cylinders.

interaction pairs stored in the previously defined `pairs_cortex` array. For each such pair, first the `label1` and `label2` components are used to find the global labels of the two elements involved, and then the unit vector parallel to the line between them. For cells of type 1, a component is then added to the velocity of each element in the direction of the normalised displacement vector between the two elements. The magnitude of this velocity component is given by the value of `cortex_constant1`, a user-specified parameter for the cortical tension of cell type 1, multiplied by the `cortex_factor` of the pair, to which we will return in Section 4.6. A similar calculation is performed to find a velocity component for the two elements if they are in cells of type 2, but this is complicated by the introduction of blebbing, and will be discussed further in Section 4.7.

4.6 Differential Interfacial Tension

Once a cortex has been introduced into the system, we have the capacity to model differential interfacial tension, another important hypothesis that we want to test as a mechanism of self-organisation. The objective is to have the cortical tension of a cell vary locally over the cell surface. The variation in local tension should be influenced by the surface with which

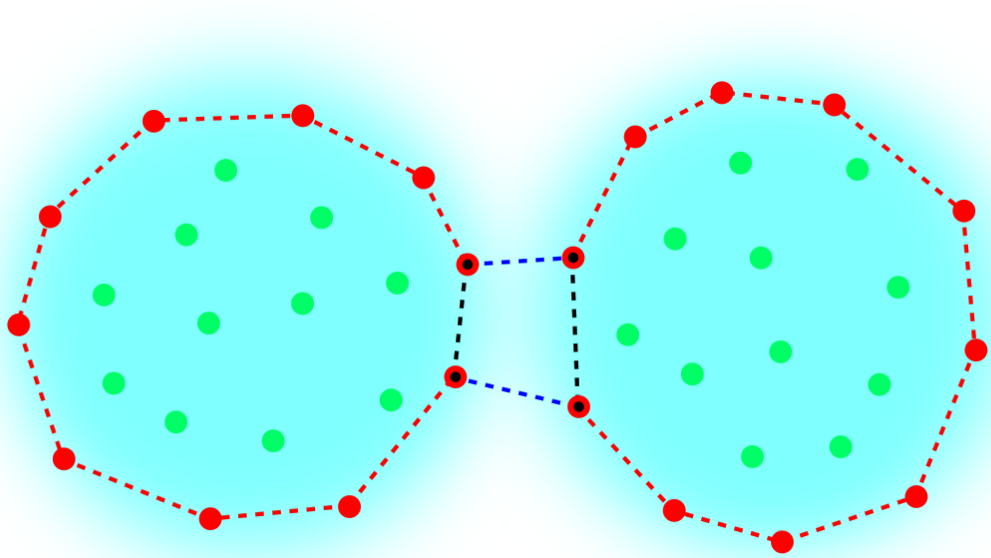


Fig. 4.8 Diagram outlining algorithm for implementing differential interfacial tension between two cells. The cell-cell interface is defined by inter-cell interactions, and those elements at the interface are labelled with a black dot. The tension is altered for any cortex interaction between two elements both labelled in this way.

it forms an interface. Changes in the local tension at interfaces can reduce or increase the surface area of those interfaces, which effectively changes the mutual affinity for the two objects and draws them together or forces them apart [Chapter 1]. Within our SEM system, local changes in the cortical tension can be implemented by changing the value of the force in some cortex interaction pairs within the cortical tension network defined by the Delaunay triangulation. This is the purpose of the `cortex_factor` component of the `cortexpair` data structure - allowing a different factor to be applied to the tension force in each cortex pair interaction.

The first problem in introducing differential interfacial tension is to define an interface for an SEM cell, and thus to identify which interactions within the cortical tension network will have their tension magnitude changed. To do this, we added a new `DIT_factor` component to the `element` data structure [Code block 4.5]. This component is used to label cortex elements according to the type of interface at which they lie. We identify 3 different kinds of interface: `DIT_factor=0` for an external medium (not cell-cell) interface is the default value; for homotypic interfaces, in which both cells at the interface are of the same type, `DIT_factor=1`; for heterotypic interfaces, in which the cells at the interface are of different types, `DIT_factor=2`. An interface is defined where the cortex elements of one cell share

standard nearest-neighbour interactions with the cortex elements of another cell. So cortex elements are labelled according to whether they share nearest neighbour interactions with elements in another cell, and by what the fate of that other cell is. For example, for a given element in a cell of type 1, if this element shares a nearest neighbour interaction with a cortex element in another cell of type 1, the `DIT_factor` for this element is set to 1. The cortical tension magnitude is changed for any interaction pair in which both elements have the same `DIT_factor` value.

This algorithm is summarised in Figure 4.8, in which elements at the cell-cell interface are labelled with black dots. In Figure 4.8 there are two cortex elements in each cell labelled in this way. This is because these elements share nearest neighbour interactions with elements in another cell. An interface interaction is then defined between any two elements that both have the black dot label. Such an interaction is seen in Figure 4.8 as a dotted black line. This interaction is then given a different force from the other cortex interactions. Reducing the force in this interaction will reduce the local tension at the interface between the two cells and increase their mutual affinity; increasing the force will increase the local tension in the interface and decrease their affinity.

The DIT algorithm described above is implemented as subroutine `scem_dit` in module `ScEM_2_dit.f90` [Code block 4.13]. The subroutine begins by refreshing the `DIT_factor` for all elements in the system. There is then a loop over all standard nearest-neighbour interactions. If both elements in the pair are within the same cell, the `DIT_factor` values for the elements are not changed. If the two elements within the pair are in different cells, the elements are considered part of an inter-cell interface. We then change the `DIT_factor` values for both elements to a value of 1 if both parent cells are of the same fate, or a value of 2 if the parent cells have a different fate. Once this loop has finished, all elements are allocated a `DIT_factor` value that indicates what sort of interface they lie at. Note that this loop can also change the `DIT_factor` for internal cortex elements, but this is irrelevant since the `DIT_factor` is only used for cortex elements.

Subsequently a loop is performed over all cortex-cortex interaction pairs in the `pairs_cortex` array as previously defined in subroutine `scem_cortex`. The `pairs_cortex` array is of type `cortexpair` as defined in module `ScEM_1_types.f90` [Code block 4.9]. The `cortexpair` data structure contains not just the labels of both elements in the pair, but also the `cortex_factor` component, which controls the magnitude of the force between the two elements. The value of the `cortex_factor` component is the factor by which the baseline cortical tension is multiplied for this interaction. The default value is 1, but if the `DIT_factor` component is the same for both elements in the pair, the value is updated according to the `DIT_response` array. This array introduces new user-specified parameters, and is defined in

```
FORALL(j=1:ne) elements(j)%DIT_factor = 0
do j=1,np
  cell_1  = elements(pairs(j,1))%parent
  fate_1  = cells(cell_1)%fate
  cell_2  = elements(pairs(j,2))%parent
  fate_2  = cells(cell_2)%fate
  if (cell_1.EQ.cell_2) then
    CYCLE
  elseif (fate_1.EQ.fate_2) then
    elements(pairs(j,1))%DIT_factor = 1
    elements(pairs(j,2))%DIT_factor = 1
  else
    elements(pairs(j,1))%DIT_factor = 2
    elements(pairs(j,2))%DIT_factor = 2
  endif
enddo
do j=1, np_cortex
  if (elements(pairs_cortex(j)%label1)%DIT_factor.EQ.
   ↪  elements(pairs_cortex(j)%label2)%DIT_factor) then
    DIT_index1 = cells(elements(pairs_cortex(j)%label1)%parent)%fate
    DIT_index2 = elements(pairs_cortex(j)%label1)%DIT_factor
    pairs_cortex(j)%cortex_factor = DIT_response(DIT_index1,DIT_index2)
  else
    pairs_cortex(j)%cortex_factor = 1.0
  endif
enddo
```

Code Block 4.13 Differential interfacial tension algorithm as implemented in subroutine `scem_dit`.

subroutine `scem_input`. The array has 2 dimensions: the first dimension divides the components into those for cell type 1 and those for cell type 2; the second dimension allows different values for different interface types. Thus for a given cortex interaction pair, the corresponding component of the `DIT_response` array is obtained by passing the fate of the parent cell as the first index, and the `DIT_factor` of the two elements in the pair (remember that they must have the same value) as the second index. These two indices are calculated in Code block 4.13 as `DIT_index1` and `DIT_index2` before setting the corresponding `cortex_factor` value of this pair to equal `DIT_response(DIT_index1,DIT_index2)`.

Once we see how `cortex_factor` is defined in this way for all cortex-cortex interaction pairs in the system, we can refer back to the use of `pairs_cortex(m)%cortex_factor` in subroutine `scem_cortical_tension_update` [Code block 4.12] to understand how the `cortex_factor` value is used to change the magnitude of the force applied to both elements in the pair according to the type of interface at which those elements are located.

4.6.1 Decoupling Tension From Adhesion

Care is required when implementing the DIT algorithm described in Section 4.6 because changes to the local cortical tension magnitude in cortex-cortex interaction pairs will change the distance between elements in the pairs and thus result in changes to the local density of elements. Since adhesion between cells is mediated by these elements, a change in the element density will affect the local adhesion strength between neighbouring cells. This could act to counteract the expected effects of differential interfacial tension. For example, increasing the local tension at an interface, which should reduce the affinity between two cells, will result in a higher density of elements and thus a stronger local adhesion between the cells, thus increasing their affinity in opposition to the effect of the change in tension. In real cellular systems, the density of adhesion molecules per unit surface area is constant regardless of the tension in, or surface area of, the interface, so this problem must be addressed.

To solve this problem, we devised an algorithm to normalise the adhesion magnitude of an element by the local element density. We begin by calculating the total area of all triangles in the Delaunay triangulation of cortex elements that have the element under consideration as one of their vertices. This is shown in Figure 4.9, where the local area around the element labelled with a black dot is highlighted with grey shading. An interface between cells is labelled with white dots on the corresponding cortex elements. The subfigure on the left shows how the arrangement of these interface elements changes when interfacial tension is reduced, and the subfigure on the right shows a possible orientation for increased interfacial tension. We can see how the shaded area is inversely proportional to the local density of elements. Thus, by making the adhesion magnitude of the element labelled with a black dot

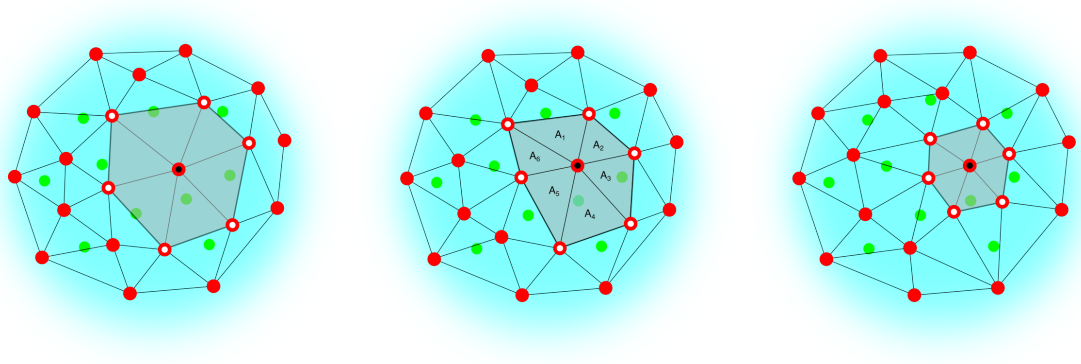


Fig. 4.9 Diagram to explain how local area around an element is calculated to produce a normalisation factor that decouples the local adhesion magnitude from the local element density, and hence from the local cortical tension magnitude.

proportional to the grey area, the adhesion magnitude is normalised to a constant adhesion per unit area.

The algorithm outlined above is handled by subroutine `scem_decouple_adhesion` [Code block 4.14] in module `ScEM_2_decouple_adhesion.f90`, which is called from subroutine `scem_near_neighbour_update` prior to calculating element velocities due to nearest-neighbour interactions. For this routine, we introduce a new `adhesion_factor` component to the `element` data structure [Code block 4.5]. This is used to store the total area of all Delaunay triangles surrounding the element in question, divided by the equilibrium value, `area_normalisation_factor`, which is calculated before the simulation begins [Section 4.8].

Subroutine `scem_decouple_adhesion` begins by refreshing the `adhesion_factor` component of all elements in the system. There is then a loop over all cells in the system, and within that a loop over all cortex elements within each cell. For each cortex element, there is a loop over all triangles in the Delaunay triangulation for that cell. For each triangle, we find the labels of all 3 of its vertex elements, given by `cells(i)%triplets(:,k)`. If any one of these labels is the same as the label of the cortex element under consideration, given by `element_label = cells(i)%cortex_elements(j)`, then we add the area of this triangle to the local area of the cortex element under consideration. This area is found using a cross product of displacement vectors between the triangle vertices: $A = \frac{1}{2}|\underline{a} \times \underline{b}|$. The `adhesion_factor` component of the element is then set to equal this local area sum divided by a constant `area_normalisation_factor`. This `area_normalisation_factor` is the mean baseline value of the local area around cortex elements of a single cell in a steady state. The value can vary according to the parameters of the system, but is found to be constant as a cell grows, justifying the use of a single value. This value is calculated before a run, as discussed

```
FORALL(i=1:ne) elements(i)%adhesion_factor=1
do i=1, nc
  do j=1, cells(i)%cortex_elements(0)
    element_label = cells(i)%cortex_elements(j)
    local_area = 0
    do k=1, cells(i)%triplet_count
      t1 = cells(i)%triplets(1,k)
      t2 = cells(i)%triplets(2,k)
      t3 = cells(i)%triplets(3,k)
      if (t1.EQ.element_label.OR.t2.EQ.element_label.OR.
       ↪  t3.EQ.element_label) then
        a = elements(t1)%position - elements(t2)%position
        b = elements(t1)%position - elements(t3)%position
        c = CROSS_PRODUCT(a,b)
        local_area = local_area + 0.5*SQRT(DOT_PRODUCT(c,c))
      else
        CYCLE
      endif
    enddo
    if (nc.GT.1) then
      elements(element_label)%adhesion_factor =
 ↪  local_area/area_normalisation_factor
    else
      area_normalisation_factor = area_normalisation_factor+local_area
    endif
  enddo
  if (nc.EQ.1) area_normalisation_count =
   ↪  area_normalisation_count+cells(i)%cortex_elements(0)
enddo
```

Code Block 4.14 Algorithm for decoupling adhesion magnitude from local cortical tension, as implemented in `scem_decouple_adhesion`.

in Section 4.8. This normalisation allows adhesion magnitudes to be comparable across parameter sets; otherwise an adhesion magnitude of, say, 1.0, would be relatively stronger in a system with a cortical tension magnitude of 0.05 than a system with cortical tension of 0.20, due to the smaller local areas expected around cortex elements with a higher tension.

Once the `adhesion_factor` value has been found for all cortex elements within each cell, it is used within `scem_near_neighbour_update` to normalise the magnitude of inter-cell forces for changes in cortex element density. Code block 4.15 shows the main part of the `scem_near_neighbour_update` subroutine. This is a loop over all elements of the nearest-neighbour `pairs` array. After first checking that the separation of the two elements in the pair are within the maximum interaction range, the first step is to find the relevant prefactors for the attractive and repulsive components of the interaction potential according to the type of elements and cells, and whether the pair is inter- or intra-cell. These prefactors are found from the `rel_strength` array [Section 4.3]. Subsequently there is a test for whether the pair is intra-cell, and whether either element is of type 1 (cytoplasm). If either of these criteria is met, no update is applied to the adhesive component between the two elements. However, if both elements are cortex elements (type 2) and are in different cells, the adhesive component of the potential is updated in the `potential_deriv_interp1` calculation. Note that the value used in the calculation is `MIN(elements(n)%adhesion_factor,elements(nn)%adhesion_factor)`, meaning the minimum value of the `adhesion_factor` components for each of the two elements. Since the adhesion strength between cells depends upon the density of cadherin molecules that mediate the adhesion, it seems logical that the adhesion strength should depend on whichever cell has the lowest density, corresponding to the lowest `adhesion_factor` component. Finally, the values from these calculations are used to update the velocities of both elements in the pair.

4.7 Dynamic Tension and Blebbing

As discussed in Chapter 1, it has been observed that primitive endoderm cells undergo significant blebbing during the period over which self-organisation occurs in the inner cell mass. This blebbing is not observed in epiblasts so it seems reasonable to ask whether it has some effect on the self-organisation of the two cell types. Models of blebbing have been proposed before [154], but for our purposes the exact mechanisms of the blebbing are less important than the fact that it involves a local change in the cortical tension resulting in a protrusion from the cell surface. To achieve this we devised a simple algorithm demonstrated in Figure 4.10. For each cortex element on the surface of a primitive endoderm cell, the cortical tension forces experienced by this element are varied sinusoidally in time, causing

```
if (sep_sq.le.r_interaction_max_sq) then
  fadein_amp = elements(n)%strength*elements(nn)%strength
  bin        = int(sep_sq*d_r_sq_recip)
  if (intro) then
    r_s1 = fadein_amp*intro_rel_strength(1,cells(k)%fate,cells(kk)
 ↪  %fate,elements(n)%type,elements(nn)%type,index_intra)
    r_s2 = fadein_amp*intro_rel_strength(2,cells(k)%fate,cells(kk)
     ↪  %fate,elements(n)%type,elements(nn)%type,index_intra)
  else
    r_s1 = fadein_amp*rel_strength(1,cells(k)%fate,cells(kk)%fate,
 ↪  elements(n)%type,elements(nn)%type,index_intra)
    r_s2 = fadein_amp*rel_strength(2,cells(k)%fate,cells(kk)%fate,
     ↪  elements(n)%type,elements(nn)%type,index_intra)
  endif
  if (index_intra.EQ.1.OR.elements(n)%type.EQ.1
   ↪  .OR.elements(nn)%type.EQ.1.OR.intro) then
    pot_deriv_interp1 = r_s1*(sep_sq*potential_deriv1(bin,1) +
 ↪  potential_deriv1(bin,2))
    pot_deriv_interp2 = r_s2*(sep_sq*potential_deriv2(bin,1) +
     ↪  potential_deriv2(bin,2))
  else
    pot_deriv_interp1 = r_s1*MIN(elements(n)%adhesion_factor,
 ↪  elements(nn)%adhesion_factor)*(sep_sq*potential_deriv1(bin,1) +
 ↪  potential_deriv1(bin,2))
    pot_deriv_interp2 = r_s2*(sep_sq*potential_deriv2(bin,1) +
     ↪  potential_deriv2(bin,2))
  endif
  elements(n)%velocity(:) = elements(n)%velocity(:) +
   ↪  dx(:)*(pot_deriv_interp1 + pot_deriv_interp2)
  elements(nn)%velocity(:)= elements(nn)%velocity(:) -
   ↪  dx(:)*(pot_deriv_interp1 + pot_deriv_interp2)
endif
```

Code Block 4.15 Algorithm in `scem_near_neighbour_update` for calculating velocities due to nearest-neighbour interactions, updated to include normalisation of inter-cell adhesion for local element density.

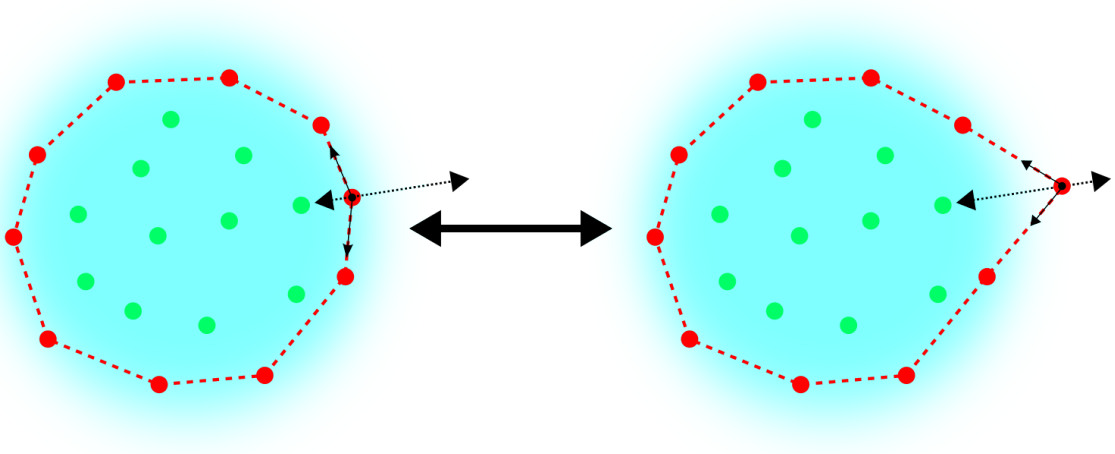


Fig. 4.10 Diagram demonstrating our model of blebbing in primitive endoderm. Focussing on the cortex element labelled with a black dot, the cortical tension forces experienced by this element from its neighbours are varied in magnitude sinusoidally in time, causing the element to protrude from the cell surface before being pulled back in.

the tension it experiences to oscillate and resulting in the element protruding from the cell surface - modelling a bleb - before being pulled back in.

To implement the algorithm outlined in Figure 4.10, a few simple changes were made to the code. We introduced `flag_pre_blebbing` to control the blebbing behaviour. If this variable is equal to 1, blebbing occurs, otherwise it does not. Secondly we created the user-specified parameter `bleb_amp`, which controls the amplitude of the sinusoidal variation in cortical tension experienced by PrE cortex elements. So for example, if `bleb_amp=0.1`, the cortical tension experienced by each PrE cortex element varies by factors between 0.9 and 1.1 of the baseline value. The time variation of this change in tension is controlled by the age of the element. This required an update to `scem_aging` such that any newly created element has a randomly determined phase added to its age [Code block 4.16]. Thus the age of the element becomes the phase of the sinusoidal variation in cortical force that it experiences, and increasing the age of the element corresponds to incrementing the phase. This variation is calculated in the section of `scem_cortical_tension_update` shown in Code block 4.17.

The cortical tension force applied to each element n and nn in the cortex pair is multiplied by `bleb_factor_n` and `bleb_factor_nn` respectively. These factors are calculated by adding a sinusoidal component to 1.0. This sinusoidal component uses the age of the element as its phase, with a period of `cell_cycle_time/10`.

```
do n=1,ne
  elements(n)%age=elements(n)%age+dt
  if (elements(n)%stage.eq.0) then
    elements(n)%strength=(elements(n)%age/establishment_time)**2
    if (elements(n)%age.gt.establishment_time) then
      elements(n)%stage=1
      elements(n)%strength=1.0
      CALL RANDOM_NUMBER(age_ran)
      elements(n)%age=elements(n)%age+age_ran*cell_cycle_time/10.0
    end if
  endif
end do
```

Code Block 4.16 Update to `scem_ageing` to show how when an element is newly established its age is updated with a random phase.

```
bleb_factor_n  = 1.0
bleb_factor_nn = 1.0
if (flag_pre_blebbing.EQ.1.AND..NOT.intro) then
  if (elements(n)%DIT_factor.EQ.0)  bleb_factor_n = 1.0 +
 ↪  bleb_amp*SIN(10*2.0*pi*elements(n)%age/cell_cycle_time)
  if (elements(nn)%DIT_factor.EQ.0) bleb_factor_nn= 1.0 +
   ↪  bleb_amp*SIN(10*2.0*pi*elements(nn)%age/cell_cycle_time)
endif
elements(n)%velocity(:) = elements(n)%velocity(:) - dx(:)*
 ↪  cortex_constant2*pairs_cortex(m)%cortex_factor*bleb_factor_n
elements(nn)%velocity(:)= elements(nn)%velocity(:)+ dx(:)*
 ↪  cortex_constant2*pairs_cortex(m)%cortex_factor*bleb_factor_nn
```

Code Block 4.17 Application of a sinusoidal blebbing factor to cortical tension for PrE cells in `scem_cortical_tension_update`.

```
if (flag_random_init.EQ.1) then
  call RANDOM_SEED
  call RANDOM_SEED(size=seedarraylength)
  allocate(seed_array(seedarraylength))
  call RANDOM_SEED(get=seed_array)
  print*, "seed_array", seed_array
else
  allocate(seed_array(2))
  seed_array(1) = 1591826533
  seed_array(2) = 497
  call RANDOM_SEED(PUT=seed_array)
endif
```

Code Block 4.18 Setting up random number sequence in `scem_input`.

4.8 Creating a Random Initial System

In the inner cell mass, differentiation of the cell aggregate into two cell types does not begin until about 3.5 days after fertilisation, at which point the inner cell mass is formed from at least 10 cells. Thus we require our systems to have many cells before parameters that might drive sorting are introduced. To do this we made updates to the system initialisation routines to grow an introductory system randomly to a predetermined number of cells. This predetermined number of cells is specified by the user as `nc_initial` in `scem_input`, and the new protocol is used so long as `flag_create=1`, otherwise the original protocol for initiating a system from stored data is used. The creation of a new cell, and all aspects of the code that require randomness, are controlled by the sequence of random numbers generated within `scem_input` by an updated random number generator [Code block 4.18]. If `flag_random_init` is set to 1, the system creates a new random seed and a new random number sequence for each run. This random number seed is stored and output to the user so that in the event of a problem, we can set `flag_random_init=0` to apply a user-specified random number seed and exactly repeat a previous run.

The subroutine `scem_initial_create` for creating one random new cell at the very beginning of a run was significantly streamlined. An important aspect of the redesigned `scem_initial_create` subroutine is the line `intro = .TRUE.`, which introduces the new logical variable `intro`. This variable tells the system that it is currently running as an introductory period to create a random initial system, from which the real simulation is then started.

The value of intro controls a number of system operations within the scem_iterate subroutine. This can be seen in Code block 4.19. Whilst intro is .TRUE., the system time is not updated, meaning that the timer only starts once the system has grown to the pre-specified size. Element velocities are calculated as normal, and in situations where growth and division are turned off for the simulation itself, they are forced to be on while intro is .TRUE. so that the intro system will still grow to the correct size. Subroutines related to data output are not called whilst intro is .TRUE.. The intro variable also controls the behaviour of other subroutines such as scem_cortical_tension_update [Code block 4.12], in which blebbing does not occur whilst intro is .TRUE., scem_cortex, in which scem_dit is not called whilst intro.EQ..TRUE., and scem_division, in which division is forced to be symmetric whilst intro.EQ..TRUE.. Furthermore, within scem_near_neighbour_update, when intro is .TRUE. the algorithm is altered to ensure that both cell types have the same adhesion, and scem_decouple_adhesion is not called unless nc.EQ.1. This is because removing DIT from the intro system removes the need to perform adhesion normalisation, and thus we can save computation time by skipping this step. However, the routine is still called when there is only one cell in the system because it allows us to calculate the area_normalisation_factor.

Within scem_decouple_adhesion, when intro.EQ..TRUE., rather than calculating the adhesion_factor component for elements, the local area calculations are used to find the mean value in this single steady state cell by summing areas and dividing by the total number of cortex elements. This value is then used as the area_normalisation_factor when intro.EQ..FALSE..

Thus setting intro.EQ..TRUE. removes all distinctions between cells, preventing differential adhesion, differential interfacial tension and so on, and prevents the system from outputting data or starting the timer until intro=.FALSE.. This switch from intro.EQ..TRUE. to intro=.FALSE. occurs when the systems reaches the specified initial cell number, and is handled by Code block 4.20 within scem_iterate. This algorithm begins with a test to see whether the number of cells at which the simulation proper should be started has been reached. If not, or if intro.EQ..FALSE. already, the whole block is skipped. If the current number of cells nc has reached this value, and intro is still equal to .TRUE., this block is accessed and we begin the process of initiating the simulation proper. Within the block, there is first a line to set intro=.FALSE., thus allowing sorting parameters and differences between cells to be implemented from this point onwards. There is then a loop to randomly reassign fates to the cells within the newly created system, with a check to ensure that although the fates are assigned randomly, only systems in which the number of epiblasts and primitive endoderm cells are equal are accepted. Once this criterion is met, an update is printed to the command line and the full simulation begins.

```
if (.NOT.intro) time=time+dt
forall(n=1:ne) xe_prev(n,:)=elements(n)%position(:)
call scem_integrate
forall(n=1:ne) elements(n)%position(:) =
 ↪  elements(n)%position(:)+0.5*dt*elements(n)%velocity(:)
forall(n=1:ne) elements(n)%velocity(:) = 0.0
call scem_integrate
forall(n=1:ne) elements(n)%position(:) =
 ↪  xe_prev(n,:)+dt*elements(n)%velocity(:)
if (flag_diffusion.eq.1.OR.intro) call scem_diffusion
if (flag_conserve.eq.1) call scem_volume_conserve
call scem_flag_relist
if (flag_relist.eq.1) then
  call scem_relist(1)
  call scem_pairs
end if
call scem_ageing
if (flag_growth.eq.1.OR.intro) call scem_growth
if (flag_division.eq.1.OR.intro) call scem_division
if (flag_growth.eq.1.or.intro) call scem_resize
call scem_com
call scem_cortex
if (flag_volume_output.EQ.1.OR.flag_conserve.EQ.1
 ↪  .OR.flag_background.NE.0) call scem_volume_calculate
if (mod(time,output_interval).LT.dt.AND..NOT.intro) then
  call SYSTEM_CLOCK(current_time)
  total_system_time = (current_time-start_time)/count_rate
  write(*,"(*(G0,:,1X))") time,total_system_time,ne,nc,n_snapshots
endif
if (mod(time,output_interval2).LT.dt.AND..NOT.intro) then
  n_snapshots=n_snapshots+1
  call scem_output_system
  if (flag_povray.EQ.1) call scem_output_povray
end if
forall(n=1:ne) elements(n)%velocity(:)=0.0
```

Code Block 4.19 Updated routine of `scem_iterate`, showing how the value of `intro` controls various aspects of the simulation.

```
if (intro.AND.nc.GE.nc_initial) then
  write(*,'(A21,I2,A41)') "Grew intro system to ",nc_initial," cells.
 ↪  Initiating simulation parameters."
  intro = .FALSE.
  fatesnotbalanced = .TRUE.
  do while (fatesnotbalanced)
    epi_counter = 0
    pre_counter= 0
    do n=1, nc
      CALL RANDOM_NUMBER(fate_decider)
      if (fate_decider.GE.0.5) then
        cells(n)%fate = 1
        epi_counter = epi_counter+1
      else
        cells(n)%fate = 2
        pre_counter = pre_counter+1
      endif
    enddo
    if (MOD(nc,2).EQ.0) then
      if (epi_counter.EQ.pre_counter) fatesnotbalanced = .FALSE.
    else
      if (ABS(epi_counter-pre_counter).EQ.1) fatesnotbalanced = .FALSE.
    endif
  enddo
  write(*,'(A29,I2)') "Initial number of epiblasts: ", epi_counter
  write(*,'(A30,I2)') "Initial number of primitive endoderm: ",
   ↪  pre_counter
  call scem_output_system
  if (flag_povray.EQ.1) call scem_output_povray
endif
```

Code Block 4.20 Algorithm from `scem_iterate` controlling the transition from introductory system growth to simulation proper.

Chapter 5

Measures and Analyses

In order to analyse the results of our model, several analytical measures were defined, and routines developed to collect relevant data from simulations as they ran. In addition, routines were included for outputting data in a form that allowed the systems to be visualised.

5.1 Quantitative Measures of Sorting

In order to objectively assess the effects of various mechanisms on the extent and speed of self-organisation in a cell aggregate, we require numerical measures of sorting. To achieve this objective, we created a number of subroutines to output information about the cell aggregate as an SEM simulation progresses. Each of these subroutines is called from the subroutine `scem_output_system` [Code block 5.1]. Subroutine `scem_output_system` is called from `scem_iterate` at time intervals determined by some fraction of the total system run time.

Subroutine `scem_output_system` performs 3 main tasks, each controlled by a flag whose value is set in `scem_input`. Within the subroutine itself there are two simple measurements: cell count and cell volume. The former outputs the number of each cell type in the system at that time to file `cell_count.txt`, which is calculated as a global variable within `scem_com` whilst calculating centres of mass [Code block 5.2]. The latter outputs the `volume` component of the `cell` data structure for each cell in the system to the file `cell_volumes.txt`. These volumes are calculated in subroutine `scem_volume_calculate` [Appendix A]. The `scem_output_system` subroutine also contains calls to a number of other subroutines that handle the quantitative sorting measures. Each of these is discussed independently in the sections that follow.

It is worth noting that self-organisation can take more than one form. For example, Figure 5.1 shows two ways in which a system of cells could sort. These two resulting arrange-

```
if (flag_count_output.EQ.1) then
  open(unit=28,file=output_folder//'/system_data/cell_count.txt',
 ↪  status='unknown')
  write(28,*) real(time), epicellcount, (nc-epicellcount) !epicellcount
   ↪  calculated in scem_com
  close(28)
endif
if (flag_volume_output.EQ.1) then
  open(unit=27,file=output_folder//'/system_data/cell_volumes.txt',
 ↪  status='unknown',position="append")
  do n=1, nc
    write(27,*) time, cells(n)%label, cells(n)%volume
  end do
  close(27)
endif
if (flag_measure_radius.EQ.1)        call scem_measure_radius
if (flag_measure_neighbours.EQ.1)    call scem_measure_neighbours
if (flag_measure_surface.EQ.1)       call scem_measure_surface
if (flag_measure_randomised.EQ.1)    call scem_measure_randomised
if (flag_measure_displacement.EQ.1) call scem_measure_displacement
if (flag_measure_velocity.EQ.1)      call scem_measure_velocity
if (flag_measure_com.EQ.1)           call scem_measure_com
```

Code Block 5.1 Subroutine `scem_output_system` to take measurements of simulations and output data.

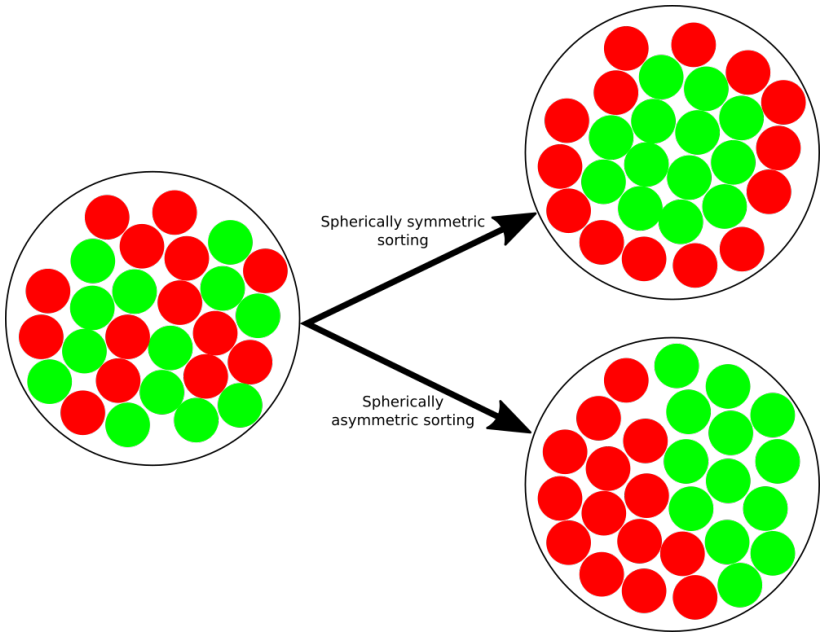


Fig. 5.1 Diagram demonstrating different patterns of self-organisation.

```
if (.NOT.intro) then
  epi_com(:) = 0
  pre_com(:) = 0
  sys_com(:) = 0
  epielementcount  = 0
  epicellcount = 0
  do n=1, nc
    sys_com = sys_com + cells(n)%position(:)
    if (cells(n)%fate.EQ.1) then
      epielementcount  = epielementcount + cells(n)%c_elements(0)
      epicellcount = epicellcount + 1
      epi_com(:) = epi_com(:) + cells(n)%position(:)
    else
      pre_com(:) = pre_com(:) + cells(n)%position(:)
    endif
  enddo
  epi_com = epi_com/epicellcount
  pre_com = pre_com/(ne-epicellcount)
  sys_com = sys_com/nc
endif
```

Code Block 5.2 Code block added to `scem_com` to calculate epiblast, primitive endoderm, and system centres of mass, as well as epiblast and primitive endoderm cell counts.

ments would both be described as sorted, but have quite different properties. Using more than one sorting measure allows us to better distinguish between such different arrangements.

5.1.1 Radius Sorting Measure

Since we will mostly be working with spherical aggregates, one option for measuring sorting is to consider the average radius of each cell type from the centre of mass. This was applied both relative to the system centre of mass, to give a measure of in-out sorting for the system as a whole, and for each cell type relative to the centre of mass of that cell type alone, giving a sense of whether each cell type tends to move closer together or not. So for example, what is the mean radius of epiblast cells relative to the centre of mass of those epiblast cells? A decrease in this value should indicate epiblast aggregation [Figure 5.2]. This measure is handled by subroutine `scem_measure_radius`, shown in Code block 5.3. The centres of mass for epiblast cells, primitive endoderm cells, and the sytem as a whole are calculated in `scem_com` [Code block 5.2]. From these values, `scem_measure_radius` performs a loop over

```
epimeanradius = 0
premeanradius = 0
sysepimeanradius = 0
syspremeanradius = 0
do i=1, nc
  if (cells(i)%fate.EQ.2) then
    cell_vector = cells(i)%position - pre_com
    premeanradius = premeanradius + SQRT(DOT_PRODUCT(cell_vector,
     ↪  cell_vector))
    cell_vector = cells(i)%position - sys_com
    syspremeanradius = syspremeanradius + SQRT(DOT_PRODUCT(cell_vector,
     ↪  cell_vector))
  else
    cell_vector = cells(i)%position - epi_com
    epimeanradius = epimeanradius + SQRT(DOT_PRODUCT(cell_vector,
     ↪  cell_vector))
    cell_vector = cells(i)%position - sys_com
    sysepimeanradius = sysepimeanradius + SQRT(DOT_PRODUCT(cell_vector,
     ↪  cell_vector))
  endif
enddo
premeanradius = premeanradius/(nc-epicellcount)
epimeanradius = epimeanradius/epicellcount
syspremeanradius = syspremeanradius/(nc-epicellcount)
sysepimeanradius = sysepimeanradius/epicellcount
if (randomising) then
  random_values_radius(ran_loop,:) = (/epimeanradius, sysepimeanradius,
 ↪  premeanradius, syspremeanradius/)
else
  open(unit=35,file=output_folder//'/sorting_data/radius.txt',
 ↪  status='unknown',position="append")
  write(35,"(*(G0,:,1X))") time, epimeanradius, sysepimeanradius,
   ↪  premeanradius, syspremeanradius
  close(35)
endif
```

Code Block 5.3 Algorithm for radius sorting measure.

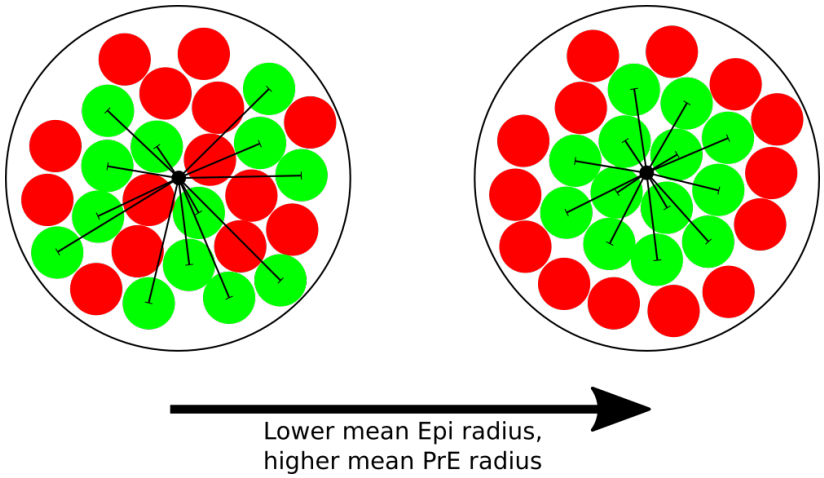


Fig. 5.2 Diagram demonstrating the radius sorting measure, in which the mean radius of epiblast cells (shown in green) is calculated from the system centre of mass or epiblast cells centre of mass. This mean value is lower for the sorted system on the right than the mixed system on the left.

all cells to calculate the mean radius of each cell type from both its centre of mass and the system centre of mass. These values are written to file `radius.txt` in folder `sorting_data`. Note the code block that is only accessed if the variable `randomising` is true. This will be explained further in Subsection 5.1.4.

5.1.2 Neighbour Sorting Measure

In a self-organising system, we would expect a higher probability for the neighbours of any given cell to be of the same type as that cell [Figure 5.3]. We decided to use this as another sorting measure, implemented as subroutine `scem_measure_neighbours` [Code block 5.4]. For the purposes of this measure, we consider two cells to be neighbours if nearest-neighbour inter-element interactions exist between an element in one cell and an element in the other. Thus neighbouring cells are those that apply forces directly to one another. The algorithm uses a loop over all `np` nearest-neighbour element-element pairs in the `pairs` array to create a `neighbours` array that stores cell neighbour pairs. For each element-element interaction pairs, if the parent cells of the two elements in the pair are different then the two parent cells are added to the `neighbours` array. This array is of `logical` type, and for any pair of cells a and b, if a and b are neighbours as defined above then `neighbours(a,b)=.TRUE.`.

Once the `neighbours` array has been fully defined, there is a loop over all cells in the system, and for each cell, there is another loop over all remaining cells in the system that have a higher label value than the first cell, so that all possible combinations are tested

```
neighbours(:,:) = .FALSE.
do n=1, np
  parent1 = elements(pairs(n,1))%parent
  parent2 = elements(pairs(n,2))%parent
  dx(:) = elements(pairs(n,1))%position - elements(pairs(n,2))%position
  if (parent1.NE.parent2.AND.DOT_PRODUCT(dx, dx).LT.r_interaction_max_sq)
   ↪  then
    neighbours(parent1,parent2) = .TRUE.
  endif
enddo
neighbour_counts(:,:) = 0
do i=1,nc
  do j=i+1,nc
    if (neighbours(i,j)) then
      fate1 = cells(i)%fate
      fate2 = cells(j)%fate
      neighbour_counts(fate1,fate2) = neighbour_counts(fate1,fate2) + 1
    endif
  enddo
enddo
if (randomising) then
  random_values_neighbours(ran_loop,:) = (/neighbour_counts(1,1),
 ↪  neighbour_counts(2,2)/)
else
  open(unit=36, file=output_folder//'/sorting_data/neighbours.txt',
 ↪  status='unknown', position="append")
  write(36,"(*(G0,:,1X))") time, neighbour_counts(1,1),
   ↪  neighbour_counts(2,2), neighbour_counts(2,1)+neighbour_counts(1,2)
  close(36)
endif
```

Code Block 5.4 Algorithm for neighbour sorting measure from `scem_measure_neighbours`.

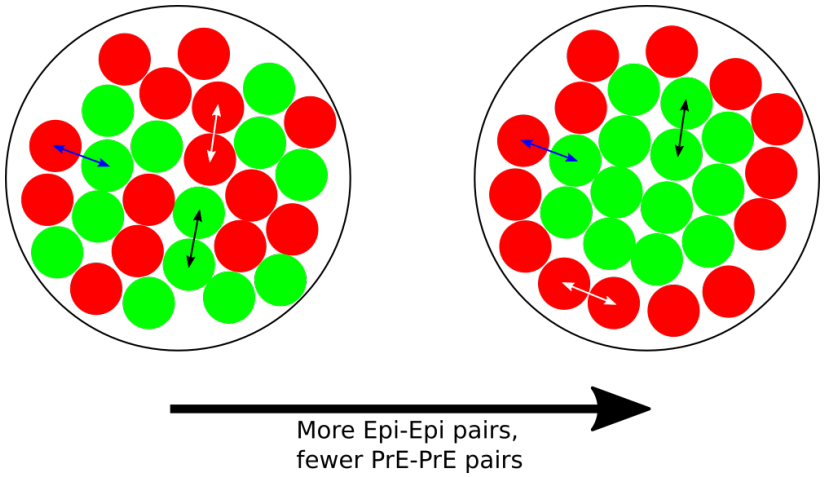


Fig. 5.3 Diagram demonstrating neighbour sorting measure. White arrows indicate PrE-PrE neighbours, black arrows indicate Epi-Epi neighbours, and blue arrows indicate Epi-PrE neighbours. The sorted system on the right shows more black Epi-Epi pairs than white PrE-PrE pairs, compared to the mixed system on the left.

without double counting. For each combination, if the corresponding component of the `neighbours` array is `.TRUE.` we know that these two cells are neighbours. We then find the fates of these two cells and update the corresponding components of the `neighbour_counts` array. This is a 2 dimensional array with 4 components: `neighbour_counts(1,1)` stores the number of epiblast-epiblast neighbour pairs, `neighbour_counts(2,2)` stores the number of PrE-PrE pairs, and the sum of `neighbour_counts(1,2)` and `neighbour_counts(2,1)` gives the number of unlike pairs. These pair counts are written to file `neighbours.txt` in folder `sorting_data`, giving the total number of Epi-Epi, PrE-PrE, and Epi-PrE counts at every output interval.

5.1.3 Surface Sorting Measure

In spherical inside-outside sorting, we would expect one cell type to occupy a greater percentage of the external surface of the cell aggregate than the other [Figure 5.4]. We used this as a measure of sorting, implemented in subroutine `scem_measure_surface` [Code block 5.5]. To implement this technique, we relied upon the `DIT_factor` component of the `element` data structure, introduced in Chapter 4. Recall that this component is set to equal 1 or 2 for any element that lies at a like-like or unlike cell-cell interface, but is otherwise set to 0. Thus in a densely packed aggregate we can consider any region of a cell that has cortex elements with `DIT_factor` components set to 0 to be part of the external surface of

```
epi_area = 0
pre_area = 0
do i=1, nc
  do j=1, cells(i)%triplet_count
    factor1 = elements(cells(i)%triplets(1,j))%DIT_factor
    factor2 = elements(cells(i)%triplets(2,j))%DIT_factor
    factor3 = elements(cells(i)%triplets(3,j))%DIT_factor
    if ((factor1.EQ.0).AND.(factor2.EQ.0).AND.(factor3.EQ.0)) then
      a = elements(cells(i)%triplets(1,j))%position -
 ↪  elements(cells(i)%triplets(2,j))%position
      b = elements(cells(i)%triplets(1,j))%position -
       ↪  elements(cells(i)%triplets(3,j))%position
      c = CROSS_PRODUCT(a,b)
      area = 0.5*SQRT(DOT_PRODUCT(c,c))
      if (cells(i)%fate.EQ.1) then
        epi_area = epi_area + area
      else
        pre_area = pre_area + area
      endif
    else
      CYCLE
    endif
  enddo
enddo
if (randomising) then
  random_values_surface(ran_loop) = pre_area/(epi_area+pre_area)
else
open(unit=43,file=output_folder//"/sorting_data/surface.txt",
 ↪  status="unknown",position="append")
  write(43,"(*(G0,:,1X))") time, pre_area/(epi_area+pre_area)
  close(43)
endif
```

Code Block 5.5 Algorithm for surface sorting measure, from `scem_measure_surface`.

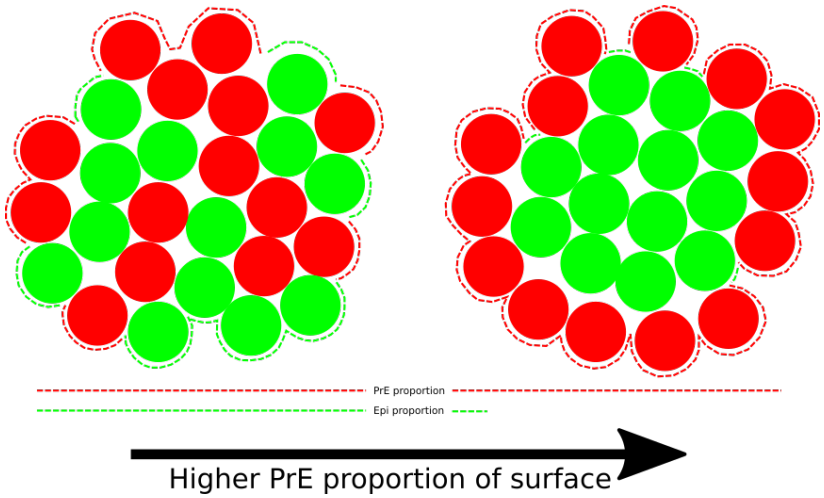


Fig. 5.4 Diagram demonstrating the surface sorting measure. The dotted red line shows the amount of external surface occupied by PrE, whereas the dotted green line shows the amount of external surface occupied by Epi. It is clear that in the sorted system on the right, more of the external surface is occupied by PrE than Epi.

the aggregate. Some regions within the aggregate will also meet this criterion, but we expect the effect to be dominated by the external boundary of the system.

The surface sorting measure calculation is performed beginning with a loop over all cells in the system, and then for each cell a loop over all surface triplets as defined by the Delaunay triangulation of cortex elements. For each triplet, the `DIT_factor` is found for all three elements in the triplet. If all three of these `DIT_factor` values is equal to 0, the triangle is considered part of the external surface, and the area of the triangle is calculated. This value is added to `epi_area` or `pre_area` depending on the fate of the parent cell. These final values are written to file `surface.txt` in folder `sorting_data` as the primitive endoderm surface as a proportion of the total surface.

5.1.4 Randomised Control Systems

It quickly became clear that interpreting raw data from the measures described above was challenging without context. Thus it became necessary to have some controls against which results could be compared. We also wished to present results as a "sorting index" that could fall within a limited possible range, such as 0 to 1 or 0 to 100. Both of these objectives were achieved by the introduction of randomised measures. The premise of this method is that at every data output interval, the fates of all cells in the system are randomly reassigned (whilst maintaining the same number of each cell type) and sorting measures are calculated again

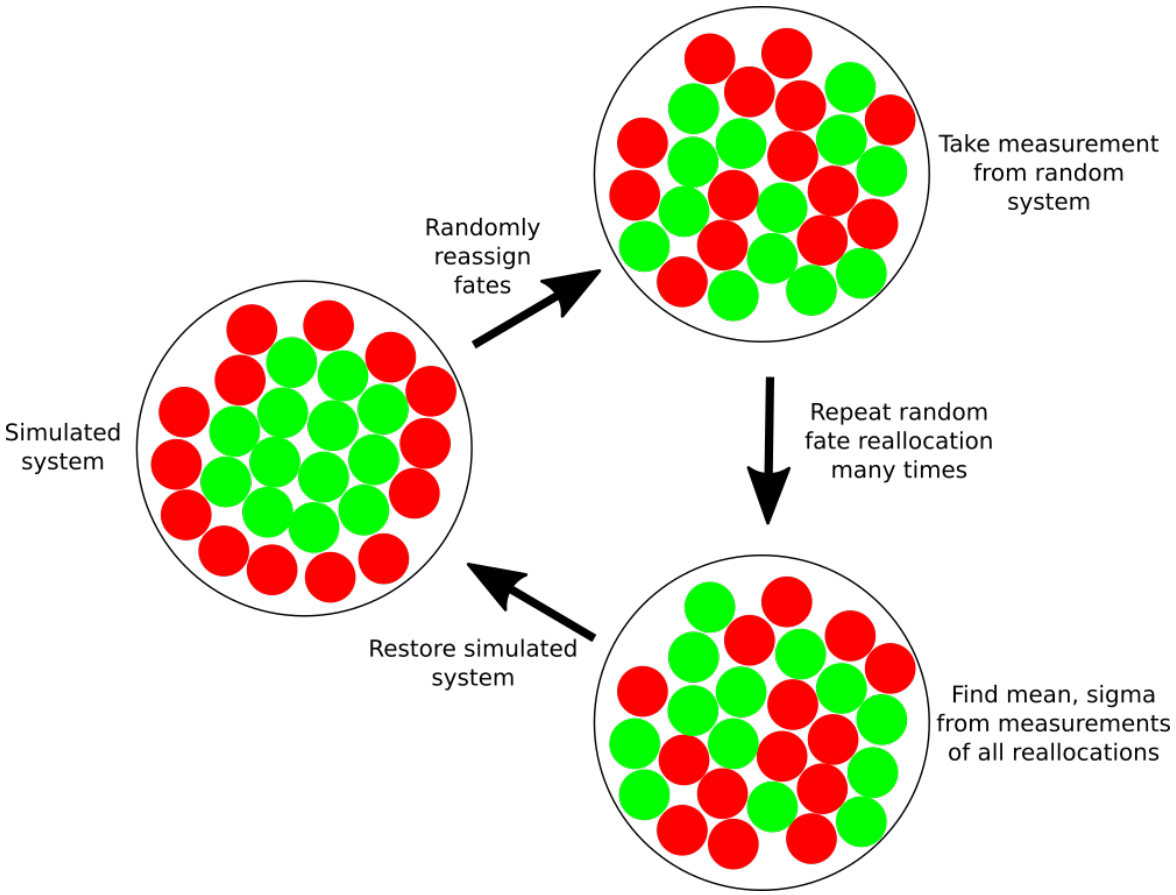


Fig. 5.5 Diagram showing how system fates are reassigned to produce mean, standard deviation, maximum, and minimum values of each measure for a given spatial configuration and epiblast-primitive endoderm ratio.

on this new system orientation. It is found that this procedure produces an approximately Gaussian distribution of sorting measure values [Figure 5.6].

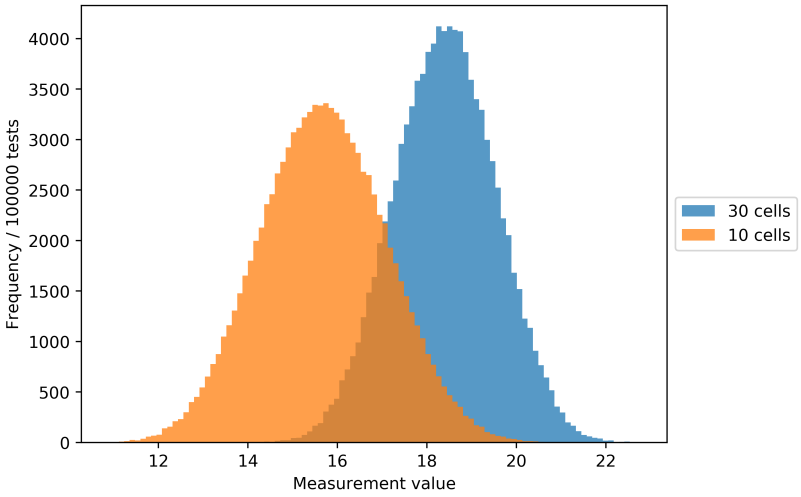
By repeating the fate reallocation procedure a large number of times, we are able to calculate a mean value, maximum value, minimum value, and standard deviation of possible values for any sorting measure given the current distribution of cell sizes and number of each cell type. However, a 40 cell system with 20 cells of each type has almost 1.6×10^8 possible different combinations of fates, and it is impossible to sample all such systems without increasing the run time of a simulation beyond what is computationally feasible. Fortunately, we were able to show that values quickly tend towards a limit over a much smaller number of repetitions. Figure 5.7 shows how the mean, standard deviation, maximum, and minimum values found vary with the number of random reallocations tested for typical systems of 10 and 30 cells. It can be seen that the mean and standard deviation values are found with very little variation over fairly small numbers of reallocations, whilst the maximum and minimum values have roughly found their limits after about 10^4 reallocations. Thus we chose to use 10^4 reallocations in our simulations.

Having found the mean, minimum, maximum, and standard deviation of a sorting measure across these random reallocations, we are able to normalise a sorting measure as its difference from the mean relative to a multiple of the standard deviation. Thus the sorting index for a system, S_i, is found from Equation 5.1, where X is a the value of a sorting measure for the current system, $E(X)$ is the mean of that measure across all random fate reallocations, and σ is the standard deviation of all such random systems.

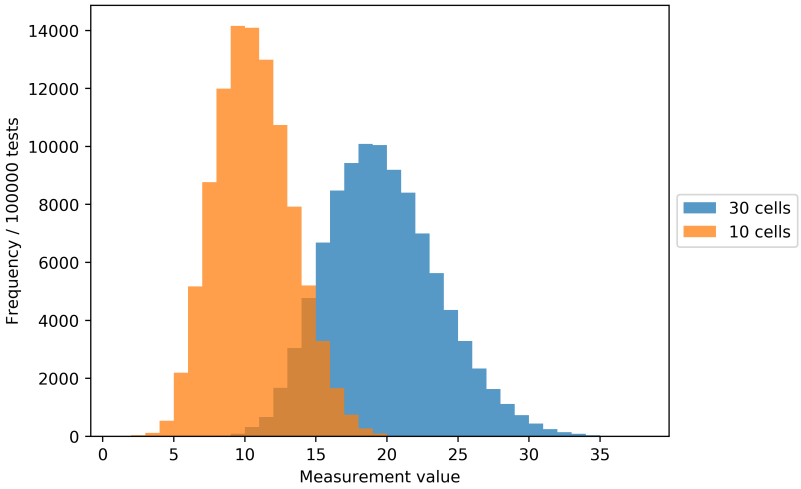
$$S_i = \frac{X - E(X)}{3\sigma} \tag{5.1}$$

By taking the range between the mean and 3σ, anomalously high maximum or minimum values do not affect the index. However, in cases where the range between the mean and maximum is smaller than the value of 3σ, this maximum range is used instead of 3σ.

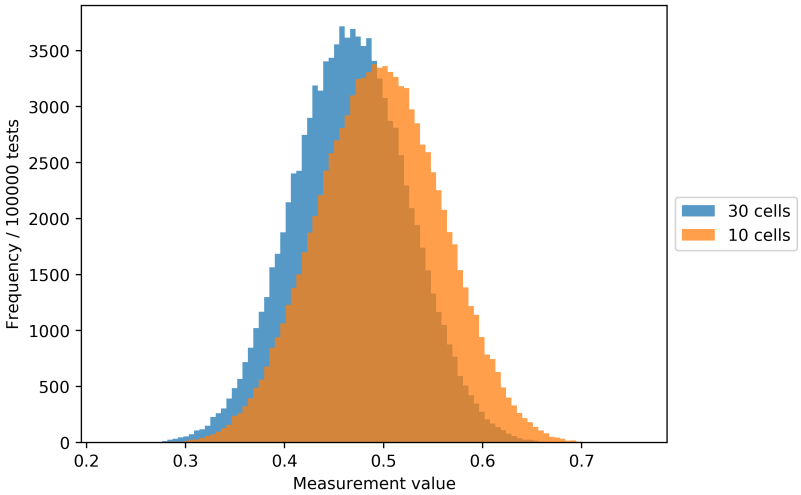
To implement this algorithm, we created subroutine `scem_measure_randomised`, which is called from `scem_output_system` at every data output interval if `flag_measure_randomised.EQ.1`. Subroutine `scem_measure_randomised` begins with the routine shown in Code block 5.6. The first step is the fill the `stored_fates` array, which was defined in module `ScEM_0_arrays.f90`, with the fates of all `nc` cells in the system. Storing this data will allow us to revert the system back to its original state at the end of the randomised measures. It is then necessary to set the number of possible random fate reallocations to be tested. We have already set a maximum value of 10000 tests, but for some small numbers of cells this may be larger than the total number of possible combinations, so we use Stirling's approximation for factorials [199] to calculate



(a) Distribution of mean epiblast radius.

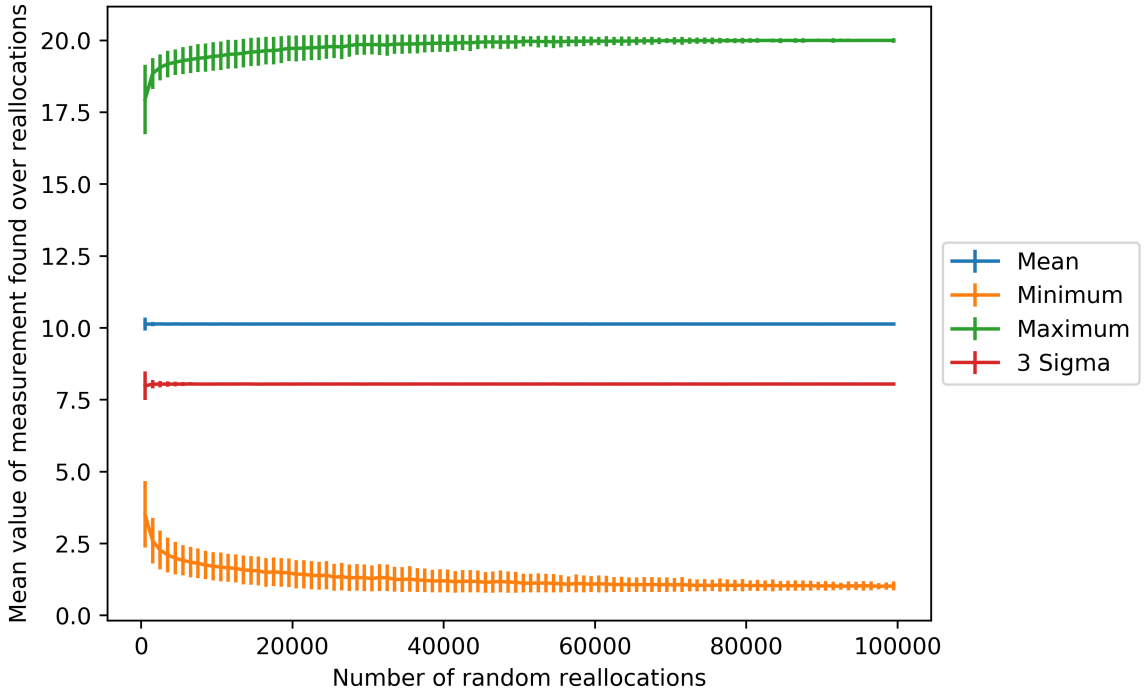


(b) Distribution of epiblast-epiblast neighbour pair count.

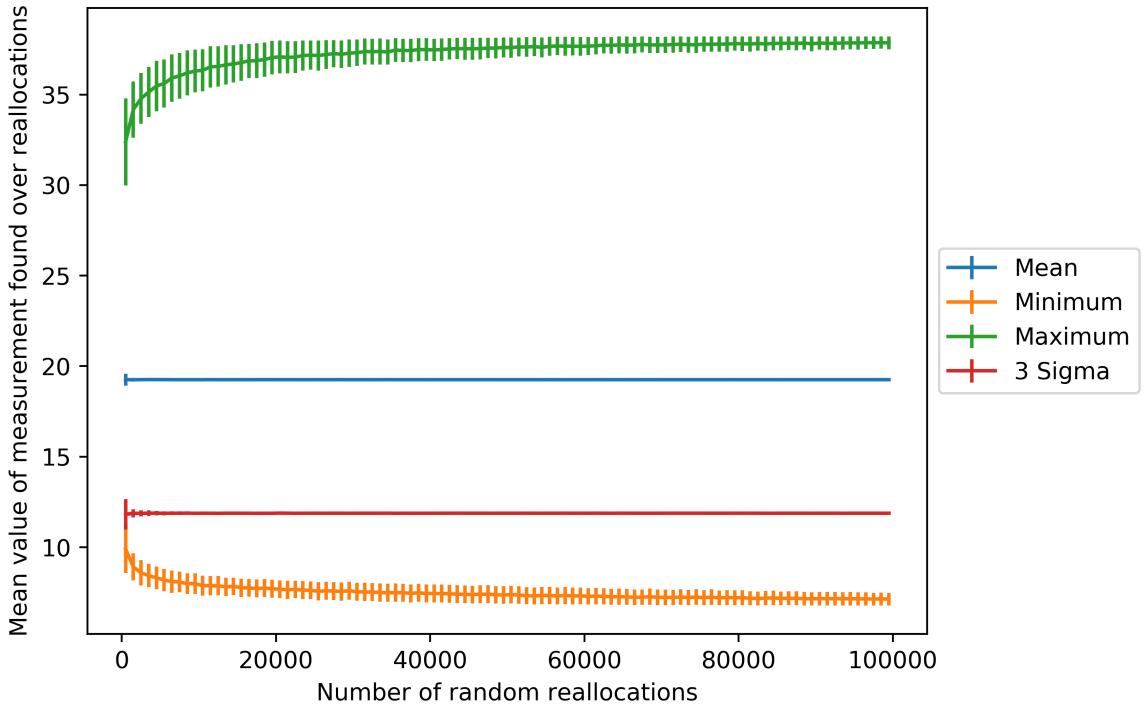


(c) Distribution of epiblast external surface proportion.

Fig. 5.6 Distributions of sorting measure over fate reallocations for 10 and 30 cells.



(a) 10 cell system



(b) 30 cell system

Fig. 5.7 Mean value of mean, minimum, maximum, and standard deviation of epiblast-epiblast neighbour count found against number of random fate reallocation tests.

the number of possible combinations, given by $\frac{n!}{k!(n-k)!}$, where n is the number of cells and k is the number of epiblasts. Whichever value is smallest is used as the number of random reallocations to test. We then set `randomising=.TRUE.`, which changes the behaviour of the sorting measure subroutines. For example, in [Code block 5.3], we see that when `randomising=.TRUE.`, the subroutine does not write data to file, but instead stores the values in the array `random_values_radius`. There are 3 such arrays, for the radius, neighbour, and surface sorting measures. These are defined in `ScEM_0_arrays` and have a 0th dimension of the same size as the total number of random reallocations. Thus they can be used to store the result of each sorting measure at each random reallocation. The contents of each of these arrays are refreshed at each output interval [Code block 5.6]. At this point we also refresh the `tested` array, which also has a component for each random reallocation. This array prevents testing the same orientation of fates more than once. To do this it stores an integer representing the fate orientation of each reallocated system tested. For example, if we have a 3 cell system in which cell 1 is of fate 2, cell 2 is of fate 2, and cell 3 is of fate 1, we would add 221 as a component of the `tested` array. For any subsequent randomly produced orientation we would then test whether its integer code, like 221, already exists in the `tested` array, and if so we will not test this orientation again.

After the initial setup of necessary arrays, a loop is performed over values of `ran_loop` from 1 to `n_random` [Code block 5.6]. The first part of this loop is similar to the routine for setting an initial system configuration in `scem_iterate`. Logical variable `fatesnotbalanced` is set to `.TRUE.`, and is used to control a loop in which fates of all cells are randomly reallocated until a configuration is found for which the number of each cell type matches the original system and which does not already occur in the `tested` array. If these criteria are met, `fatesnotbalanced` is set to `.FALSE.`, the loop stops and the current configuration is tested. The current configuration is stored in the `tested` array as an integer, as described above, and sorting measure subroutines are called to calculate values for the current configuation. Because `randomising=.TRUE.` these subroutines do not output their data directly but instead store their results in arrays as described above. Whilst it would be possible to write all data from all random tests to file and find the maximum, mean, and σ values in later processing of the data, FORTRAN is slow to write to file, so it is vastly more efficient to find these values within the routine and output only these values to file.

After all random reallocations have been measured, the next step is to calculate mean, minimum, maximum, and standard deviation values. Code block 5.7 shows how this is performed for radius sorting measures. The radius sorting measure has 4 components, corresponding to the mean distance of each cell type from both the system centre of mass and that cell type's centre of mass, so we use the `means` array to calculate the mean of each

```
do n=1,nc
  stored_fates(n) = cells(n)%fate
enddo
random_values_surface    = 0
random_values_radius     = 0
random_values_neighbours = 0
if (n_random.LT.n_random_max) n_random = MIN(n_random_max,
 ↪  INT(0.95*(nc**(nc+0.5))/(SQRT(2*pi)*epicellcount**(epicellcount +
 ↪  0.5)*(nc - epicellcount)**(nc - epicellcount + 0.5))))
randomising = .TRUE.
tested(:) = 0

do ran_loop=1, n_random
  fatesnotbalanced = .TRUE.
  do while (fatesnotbalanced)
    epi_ran_counter = 0
    do n=1, nc
      CALL RANDOM_NUMBER(fate_decider)
      if (fate_decider.GE.0.5) then
        cells(n)%fate = 1
        epi_ran_counter = epi_ran_counter+1
      else
        cells(n)%fate = 2
      endif
    enddo
    if (epi_ran_counter.EQ.epicellcount) fatesnotbalanced = .FALSE.
    configuration = 0
    do n=1,nc
      configuration = configuration + (2**(n-1))*(cells(n)%fate-1)
    enddo
    do n=1,ran_loop
      if (configuration.EQ.tested(n)) then
        fatesnotbalanced = .TRUE.
        EXIT
      else
        CYCLE
      endif
    enddo
  enddo
  tested(ran_loop) = configuration
  if (flag_measure_radius.EQ.1)       call scem_measure_radius
  if (flag_measure_neighbours.EQ.1)   call scem_measure_neighbours
  if (flag_measure_surface.EQ.1)      call scem_measure_surface
enddo
```

Code Block 5.6 Randomly allocating fates and calling sorting measure routines in the algorithm for randomised control measures.

```
do n=1,4
  means(n) = (1.0*SUM(random_values_radius(:n_random,n)))/n_random
  squaremeans(n) =
   ↪  (1.0*SUM(random_values_radius(:n_random,n)**2))/n_random
  maximums(n) = MAXVAL(random_values_radius(:n_random,n))
  minimums(n) = MINVAL(random_values_radius(:n_random,n))
enddo
do n=1,4
  stds(n) = SQRT(squaremeans(n) - means(n)**2)
enddo
open(unit=44, file=output_folder//'/randomised_data/radius.txt',
 ↪  status='unknown', position="append")
WRITE(44,"(*(G0,:,1X))") time, means(1), minimums(1), maximums(1),
 ↪  stds(1), means(2), minimums(2), maximums(2), stds(2), means(3),
 ↪  minimums(3), maximums(3), stds(3), means(4), minimums(4),
 ↪  maximums(4), stds(4)
close(44)
```

Code Block 5.7 Outputting data from the algorithm for randomised control measures.

component by summing values in the `random_values_radius` array and dividing by the number of reallocations, `n_random`. At the same time, we calculate the mean of the squares of these values in `squaremeans`. These values are used, with the sorting measure means, to calculate the standard deviation of each component using $\sigma = \sqrt{E(x^2) - E(x)^2}$. Maximum and minimum values can easily be calculated by passing array slices to the intrinsic `MAX` and `MIN` functions. Once all of these components have been calculated they are written to a file within the `randomised_data` folder, leaving one line of data for each set of random reallocations. This whole process is repeated for the surface and neighbour sorting measures, but only the radius measure is shown in Code block 5.7 for simplicity.

The final step of `scem_measure_randomised` is to restore the fate configuration of the original system from the `stored_fates` array with a loop over all cells, setting `cells(n)%fate = stored_fates(n)`, and to set `randomising=.FALSE.`. The simulation can then continue as normal.

5.1.5 Displacement Measure

As a further test of our SEM systems, we decided to measure the displacement of cells from their original position in order to compare the extent of rearrangement for different

```
open(unit=41,file=output_folder//"/sorting_data/displacement.txt",
 ↪  status="unknown",position="append")
do n=1, nc
  displacement_vector(:) = cells(n)%position(:) -
   ↪  cells(n)%original_position(:)
  displacement = SQRT(DOT_PRODUCT(displacement_vector,
   ↪  displacement_vector))
  write(41,*) MIN(cells(n)%age,time), cells(n)%fate, displacement
enddo
close(41)
```

Code Block 5.8 Algorithm for displacement measurement, from `scem_measure_displacement`.

parameters. For this we introduced a new `original_position` component in the `cell` data structure [Code block 4.7], which stores the location of any cell at its original position after division. We are then able to measure the displacement between each cell's initial location and its current location at every data output interval. The measurement is performed by subroutine `scem_measure_displacement` [Code block 5.8]. This subroutine, called from `scem_output_system`, performs a loop over all cells in the system, calculates the displacement vector between the cells current position and original position, and then outputs this to file `displacement.txt` along with the fate and current age of the cell in question. This measurement will be useful for testing the extent of rearrangement in systems.

5.1.6 Alternative Measures

A number of other sorting measures were considered for our simulations. One such example was the use of clustering analysis, as applied in astronomy to galaxies and so on [200]. However, it was eventually determined that such analysis was ineffective for a system on the scale of our systems. Additionally, we considered alternative uses of the neighbour measure, attempting to incorporate the counts for Epi-Epi, PrE-PrE, and Epi-PrE neighbour pairs into one measure. For example, we could define $M = \frac{N_{EE}-N_{PP}}{N_{EP}}$, where N_{EE}, N_{PP}, N_{EP} are the 3 neighbour counts respectively, producing values for M that include information from all 3 counts. However, despite the additional information in the calculation, this measure was not found to produce any more stable results than the neighbour measure discussed above, and was thus not developed further.

5.2 Visualisation of Simulation Results

In order to help make sense of the output of the SEM code, it was necessary to find some means of visualising the systems. With this achieved, it is possible to see growth, division and movement occurring over time and clearly demonstrate how changing parameters in the system affect its evolution and final state. These images of the evolution of a system will complement the objective measures described in Section 5.1. After a brief attempt to use Gnuplot [201] and a c++ routine implementing a Voronoi tessellation of elements within a cell [202], we decided to visualise our systems using the freely available POV-Ray ray tracing software.

5.2.1 POV-Ray

POV-Ray, or "Persistence of Vision Ray Tracer", is an advanced visualisation method that can create beautiful images [203]. The program uses ray-tracing to create shapes and surfaces that allow 3 dimensional objects to be visualised in great detail with shadows and reflections. Once it became clear that our Gnuplot visualisations were unsuitable, we began to implement POV-Ray visualisations. We utilised the UberPov command line version of POV-Ray [204] in our work.

In order to visualise SEM systems with POV-Ray, the first step was to output SEM data in POV-Ray format. POV-Ray reads data files with a specific syntax for defining objects, locations, colours, and so on. We decided to implement a routine within the SEM simulations to produce these POV-Ray commands automatically from element, cell, and interaction pair data. At each output interval, if `flag_povray.EQ.1`, the program calls new subroutine `scem_output_povray`. The first part of this subroutine is shown in Code block 5.9. The subroutine begins by opening a file of the form `snap_xx.pov` within folder `povray_data`. This file is used to store POV-Ray commands to visualise a snapshot of the system at this point in time. The subroutine writes commands to the beginning of this file in order to set up the POV-Ray scene by importing necessary libraries, positioning the camera and creating light sources. We then add a translucent shape to represent the system boundary, for example a sphere with radius determined by the radius of the system boundary. The FORTRAN code used to produce these initial scene-setting commands in each POV-Ray file is shown in Code block 5.9.

After this initial setup, the remaining sections of the subroutine are controlled by a set of flags in `scem_input`. Each of these flags switches a given type of output on or off. The first possible visualisation is referred to as "volumes", and draws a translucent sphere with the same volume as a each SEM cell at the centre of mass of that SEM cell [Figure 5.9d].

```
write(povray_filename,"(A18,I2.2,A4)") "/povray_data/snap_", n_snapshots,
 ↪  ".pov"
open(unit=30, file=output_folder//povray_filename,status='unknown')
write(30,*) '#version 3.5;'
write(30,*) '#include "colors.inc"'
write(30,*) '#include "textures.inc"'
write(30,*) 'background {White}'
write(30,*)
write(30,*) 'camera {'
write(30,*) '   location  <500, 0, 0>'
write(30,*) '   angle 12'
write(30,*) '   sky <0,0,1>'
write(30,*) '   look_at<0,0,0>}'
write(30,*)
write(30,*) 'light_source { < -60, 60, 0 > color White }'
write(30,*) 'light_source { < 60, -60, 0 > color White }'
write(30,*) 'light_source { < 0, 0, 60 > color White }'
write(30,*) 'light_source { < 0, 0, -60 > color White }'
write(30,*)
if (flag_background.EQ.1) then
  write(30,'(A16,F18.9,A77)') ' sphere {<0,0,0>',
 ↪  spherical_boundary_radius, ' texture { pigment { color Blue transmit
 ↪  .85}finish{phong .8} } } // boundary'
elseif (flag_background.EQ.4) then
  write(30,'(A18,F18.9,A2,F18.9,A77)') ' sphere
 ↪  {<0.2,0.2,',(-cap_radius+h/2.0),'> ', cap_radius, ' texture { pigment
 ↪  { color Blue transmit .85}finish{phong .8} } } // boundary'
endif
```

Code Block 5.9 First section of subroutine `scem_output_povray`.

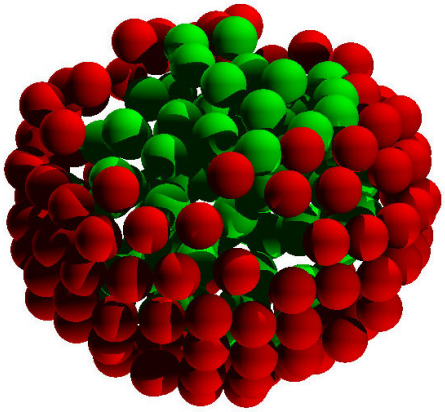


Fig. 5.8 POV-Ray visualisation of an SEM cell, cut away to show internal green cytoplasm elements and external red cortex elements.

This is useful for getting a simple overview of relative movement of cells. However, the information about underlying cell behaviour that it provides is limited. The next visualisation is "elements", in which a small solid sphere is drawn at the location of each SEM element [Figure 5.9b]. Cortex (type 2) elements are drawn as red; cytoplasm (type 1) elements are made green [Figure 5.8. This visualisation is useful for a detailed demonstration of cell behaviour, but becomes overwhelming to look at for more than one or two cells. We also created the "triangles" visualisation that utilises the POV-Ray smoothed triangle object to create a surface around an SEM cell [Figure 5.9c]. This utilises the Delaunay triangulation over the cell's cortex elements, using each triangle in the triangulation to specify a smoothed triangle object in the POV-Ray output. For this visualisation, cells of type 1 (epiblasts) are coloured green and those of type 2 (primitive endoderm) are coloured red. This visualisation is perhaps the most useful since it gives a broad sense of the changing shapes of cells and their relative motion, without overwhelming detail. Also using the Delaunay triangulation, the "cortex pairs" visualisation creates a red cylinder wherever there is a cortex-cortex interaction pair defined by the triangulation [Figure 5.9a]. This visualisation allows a detailed analysis of cortical tension behaviour, but can be overwhelming for more than a couple of cells. Additionally, we included a reference to differential interfacial tension in this visualisation by colouring any cortex-cortex interaction pair for which the `cortex_factor` of the pair is not equal to 1, and is thus part of an cell-cell interface, green rather than red [Figure 5.10]. This allows for detailed analysis of cell doublets. Finally, the "pairs" visualisation draws a black cyclinder for each inter-cell adhesive interaction [Figure 5.10].

Code block 5.10 shows an example of how the system is transcribed to POV-Ray code, in this case for the surface triangle visualisation. It can be seen how the loop over all

surface triplets in all cells is used to draw each triangle separately, and that after writing all 3 dimensions of all 3 element position, a colour is added according to whether the cell is of type 1 or type 2. For the sake of avoiding repetition we will not include the code for other visualisations, but all exists within the same subroutine `scem_output_povray`.

Figure 5.9 shows the same system of 4 identical cells visualised with cortex pair [a], element [b], surface triangle [c], and volume [d] POV-Ray visualisations. Figure 5.10 shows the cortex interaction network of one cell isolated from a cell doublet in which the cortical tension is reduced at the interface; interactions with this reduced tension are drawn in green rather than red, and thus the green area shows the interface between the two cells. Such visualisations allow us to better understand the effects of the differential interfacial tension routine described in Chapter 4.

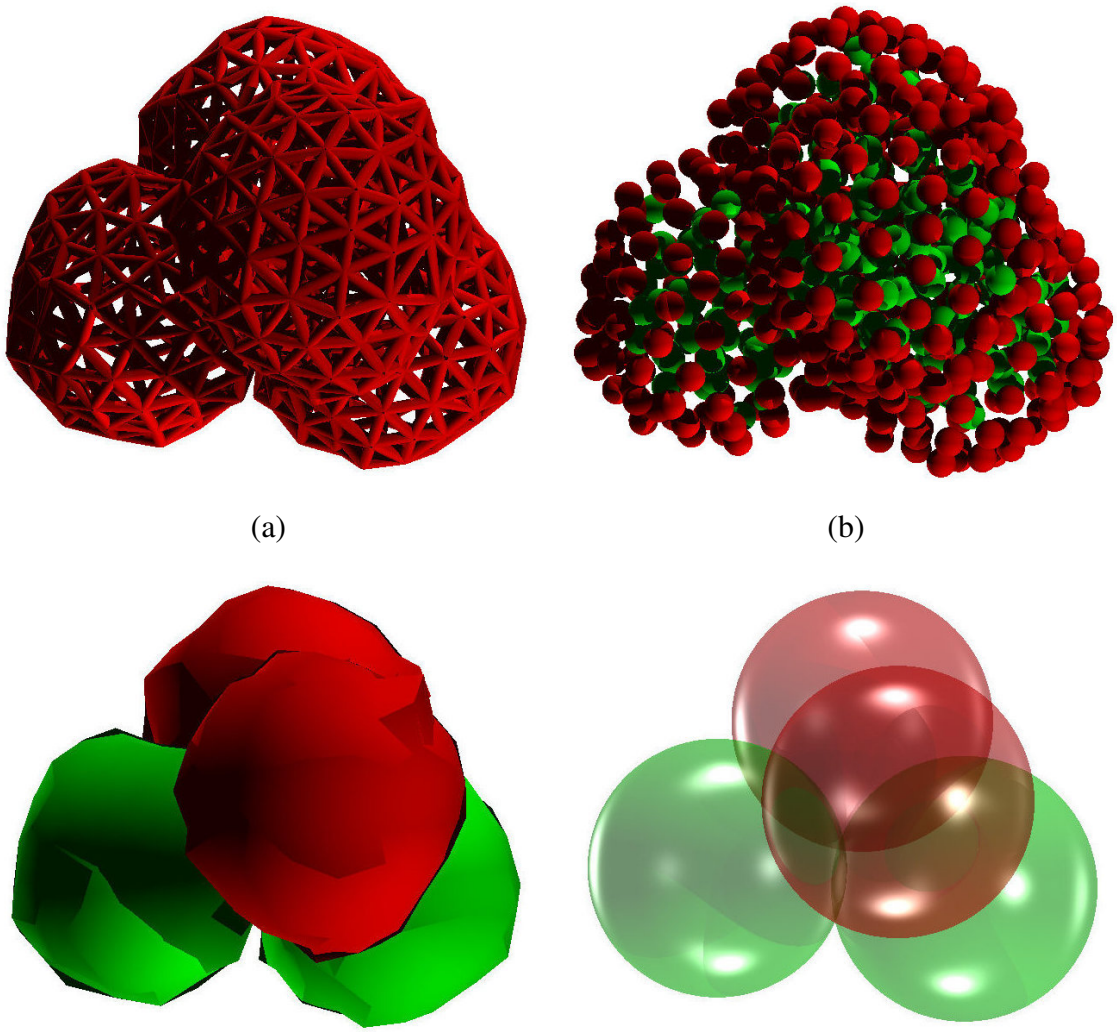
In order to simplify the FORTRAN routine, all POV-Ray commands for all visualisations are written to the same file. Each command for each object in the POV-Ray file is labelled with a comment containing the type of object and the cell to which it belongs. For example, a surface triangle could be specified with a single-line command ending with `// triangle cell01`. This is a comment and is not read by the POV-Ray program, but labels the command as a component of cell 1 and a surface triangle. Thus, such comments can be used to manipulate the data produced by the simulations. In order to visualise exactly what we want to see, we can process the original povray files to extract only the components we require, identified by the labels within the comments at the end of each line. For example, this is how Figure 5.10 was produced, isolating the data for one cell from files containing data for two cells. To manipulate the data in this way we created Perl script `extractor.pl` [Code block 5.11].

`extractor.pl` is called with arguments specifying the location of povray data on which to operate and the components to be extracted. For example, to isolate the cortex pairs of cell 2 in a simulation whose data is stored at `path`, one would call `perl extractor.pl path cortex cell02`. This command would produce a new folder called `extracted_cortex_cell02` within the main data folder for the simulation. This folder would contain the same number of POV-Ray files as the original set, but only containing those cortex interaction objects from cell 2.

The protocol outlined here allows the FORTRAN program to write all data for different visualisations to the same file, since opening and closing multiple files would slow the simulations down significantly, and to produce a large number of different files would be confusing. Extracting different components by label allows us to, for example, visualise the positions and movements of each cell type separately, or investigate the shape changes of a single cell within a system.

```
if (flag_povray_triangles.EQ.1) then
  do i=1, nc
    do j=1, cells(i)%triplet_count
      write(30,'(A17)',advance='no') "smooth_triangle {"
      do k=1, 3
        corner_element = cells(i)%triplets(k,j)
        corner(:) = elements(corner_element)%position(:)
        normal(:) = corner(:)-cells(i)%position(:)
        write(30,'(A1,F18.9,A1,F18.9,A1,F18.9,A2)',advance='no') "<", &
            corner(1), ',', corner(2), ',', corner(3), '>,'
        write(30,'(A1,F18.9,A1,F18.9,A1,F18.9,A1)',advance='no') "<", &
            normal(1), ',', normal(2), ',', normal(3), '>'
        if (k.LT.3) then
          write(30,'(A1)',advance='no') ","
        else
          EXIT
        endif
      enddo
      write(30,"(A23)",advance='no') " texture{pigment{color "
      if (cells(i)%fate.EQ.1) then
        write(30,'(A25,I2.2)') "Green}}} // triangle cell",
 ↪  cells(i)%label
      else
        write(30,'(A23,I2.2)') "Red}}} // triangle cell", cells(i)%label
      endif
    enddo
  enddo
endif
write(30,*) ""
```

Code Block 5.10 Code from `scem_output_povray` that writes surface triangle data to file in POV-Ray format.



(a) (b)

(c) (d)

Fig. 5.9 The same 4 cell system visualised with 4 different povray outputs. a, cortex pair visualisation. b, element visualisation. c. surface triangle visualisation, d,volume visualisation.

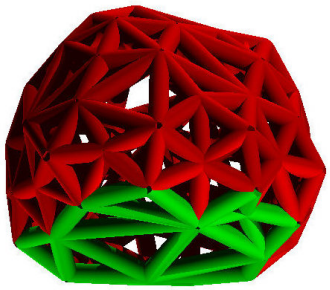


Fig. 5.10 Visualisation of cortical tension network of one cell isolated from a doublet. Green area demonstrates the region in which an interface is defined and cortical tension is altered.

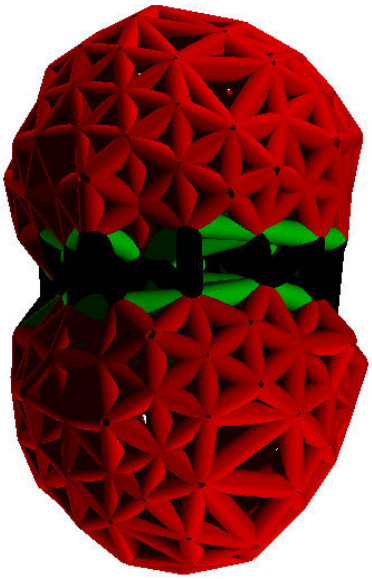


Fig. 5.11 POV-Ray visualisation of a cell doublet showing cortex pair interactions in red, or green for those with updated tension at an interface, and inter-cell adhesive interactions in black.

```
use strict;
use warnings;
defined($ARGV[0]) or die "No arguments - provide folder path in which to
 ↪  find data and at least one criterion for extraction";
defined($ARGV[1]) or die "Missing second argument - provide folder path
 ↪  and at least one criterion for extraction";
my $originalfolder = shift(@ARGV);
my $directoryname = join("_",$originalfolder."/extracted",@ARGV);
my @expressions = @ARGV;
system "mkdir $directoryname";
my $file_name_in;
my $file_name_out;
my $i = 0;
while ( -e $originalfolder."/povray_data/snap_"
 ↪  .sprintf("%02d",$i).".pov") {
  $file_name_in =
   ↪  $originalfolder."/povray_data/snap_".sprintf("%02d",$i).".pov";
  $file_name_out = $directoryname."/snap_".sprintf("%02d",$i).".pov";
  open(my $file_handle_out, '>', $file_name_out) or die "can't open
   ↪  output data file";
  open(my $file_handle_in, '<', $file_name_in) or die "can't open input
   ↪  data file";
  while (my $line = <$file_handle_in>) {
    if (($line =~ /sphere/) or ($line =~ /cylinder/) or ($line =~
     ↪  /smooth_/)) {
      if ($line =~ /boundary/) {
        print $file_handle_out $line;
      }
      else {
        my $switch = 1;
        foreach my $expression (@expressions) {
          if ($line =~ /$expression/) { next; }
          else { $switch=0; }
        }
        if ($switch==1) { print $file_handle_out $line; }
        else { next; }
      }
    }
    else { print $file_handle_out $line; }
  }
  $i++; }
```

Code Block 5.11 Contents of `extractor.pl` for isolating specific components from POV-Ray data.

We also created a script to quickly compile all POV-Ray files produced by these procedures, and thus produced an efficient workflow for visualising in detail any aspect of an SEM system.

Chapter 6

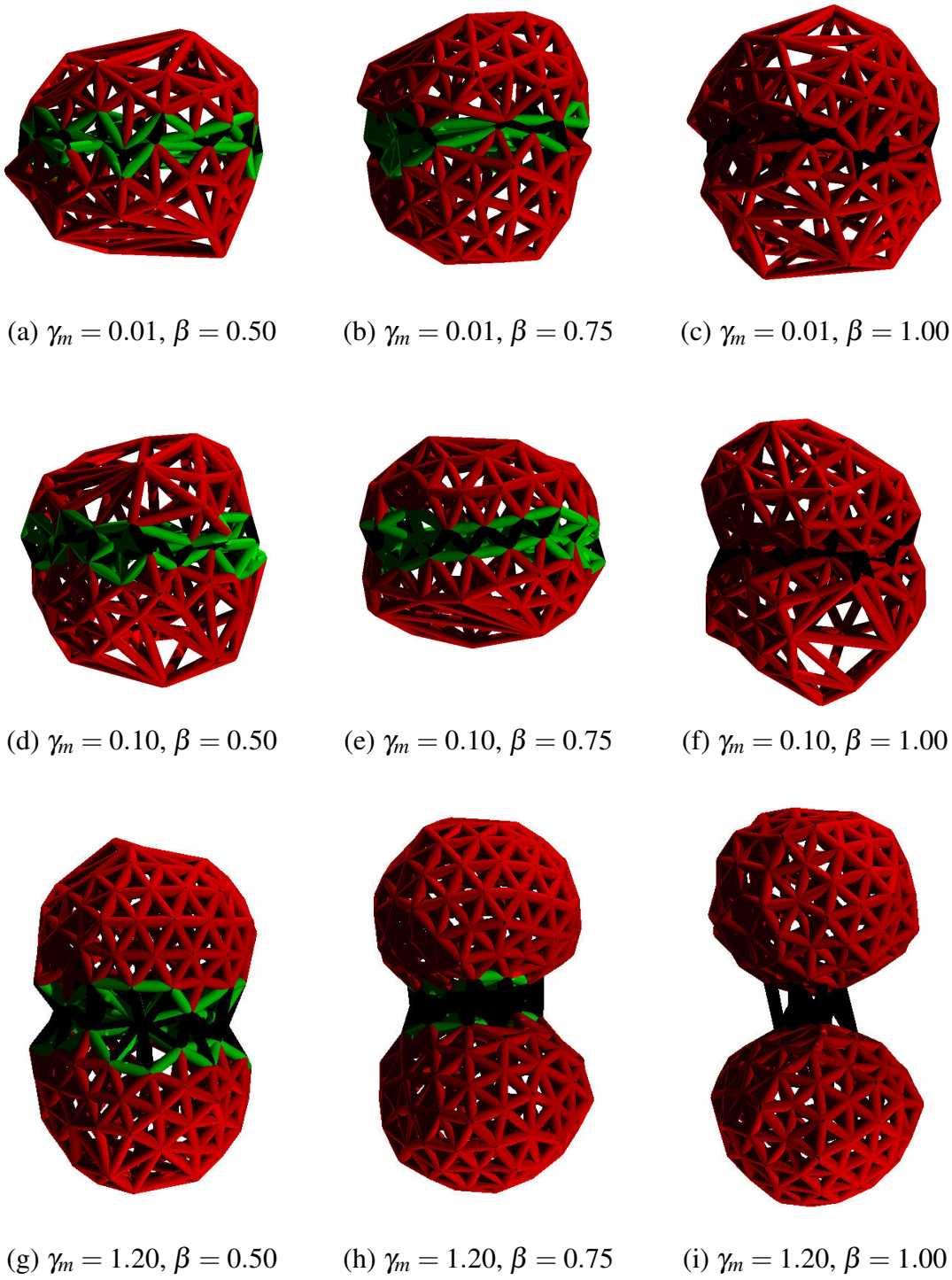
Results

6.1 Testing Doublet Interface Area

A critical question of this project is how changes in cortical tension at cell interfaces can affect the self-organisation of tissues. We expect this self-organisation to be driven by changes in relative affinity due to changes in equilibrium interface area (or, analogously, contact angle). Therefore, the first step in testing our simulations was to investigate how interface area varies with differential interfacial tension factor, as handled by the differential interfacial tension routines discussed in Chapter 4.

In order to straightforwardly compare experimental results to the results of these simulations, we tested cell doublets with our simulations by growing a system to 2 cells, forcing both cells to be of the same type, and then allowing the system to reach equilibrium with growth and division switched off, thus creating doublets of two cells adhered to one another and forming an interface [Figure 6.1]. The three relevant parameters to these systems are the adhesive magnitude between the two cells, α, the baseline cortical tension for the cells, γ_m, and the ratio of the tension at the interface to the baseline tension, β. For example, if the interfacial tension is half of the baseline tension, $\beta = 0.5$.

For *in vitro* experiments, the measurement taken for such cell doublets is the contact angle between the two cells, θ [Chapter 1]. In our simulations, this contact angle is relatively difficult to measure. However, it is fairly trivial to measure the area of the cell-cell interface by implementing a change to the `scem_measure_surface` subroutine [Chapter 5]. This subroutine calculates surface exposed to the external environment by finding the area of regions with `DIT_factor=0`. By instead calculating the total area of surface occupied by those elements whose `DIT_factor` component is equal to 1, meaning they lie at a like-like cell interface, we immediately obtain a measurement of the interface area between the two cells. Using some straightforward trigonometry [Figure 6.2], it is fairly easy to relate the



(a) $\gamma_m = 0.01, \beta = 0.50$ (b) $\gamma_m = 0.01, \beta = 0.75$ (c) $\gamma_m = 0.01, \beta = 1.00$

(d) $\gamma_m = 0.10, \beta = 0.50$ (e) $\gamma_m = 0.10, \beta = 0.75$ (f) $\gamma_m = 0.10, \beta = 1.00$

(g) $\gamma_m = 1.20, \beta = 0.50$ (h) $\gamma_m = 1.20, \beta = 0.75$ (i) $\gamma_m = 1.20, \beta = 1.00$

Fig. 6.1 Phase space of cell doublet visualisations showing variation of interfaces for β and γ_m values.

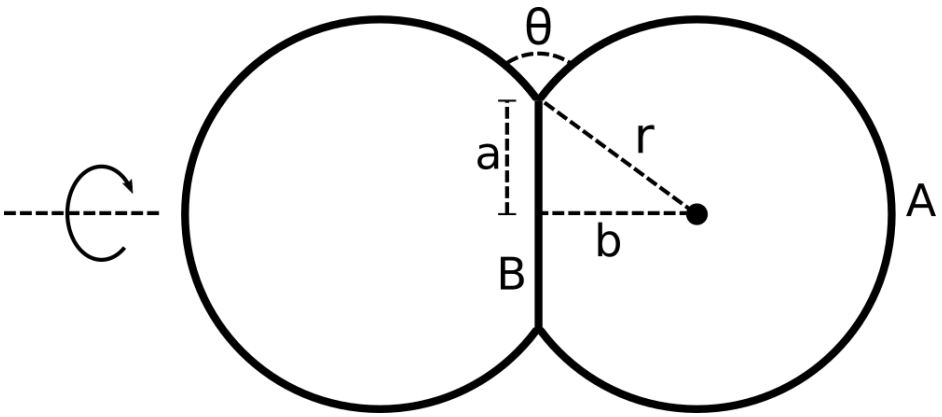


Fig. 6.2 Diagram to demonstrate how interface area (B) as a fraction of total cell surface area ($A+B$) can be calculated from contact angle θ in a cell doublet. $B = \pi a^2 = \pi\left(r\sin\left(\theta/2\right)\right)^2$, $A = 4\pi r^2 - 2\pi r\left(r-b\right) = 2\pi r^2\left(1+\cos\left(\theta/2\right)\right)$.

interface areas obtained in simulations as a proportion of the total cell surface area, I, to the contact angles obtained in experiments, θ [Equation 6.1].

$$I = \frac{B}{A+B} = \frac{\sin^2\left(\theta/2\right)}{2(1+\cos\left(\theta/2\right))+\sin^2\left(\theta/2\right)} \tag{6.1}$$

We can further extend this relationship by taking the theoretical linear force balance model of cell interfaces discussed in Chapter 1 to find a theoretical prediction of the relationship between interfacial tension factor, β, and interface area [Equation 6.2], which can then be compared to the relationship between these two factors in simulations [Subsection 6.1.2].

$$I = \frac{\sin^2\left(\arccos\beta\right)}{2\left(1+\beta\right)+\sin^2\left(\arccos\beta\right)} = \frac{1-\beta}{3-\beta} \tag{6.2}$$

6.1.1 Exploring Phase Space of Interface Proportion

The interface between two cells, I, in a doublet is a function of 3 variables: the adhesion magnitude between the two cells, α, the cortical tension magnitude of the cells, γ_m, and the cortical tension at the interface between the cells, γ_c. For simplicity we will refer in future to the ratio, β, of the interfacial tension to the baseline cortical tension, $\gamma_c/\gamma_m = \beta$, rather than the absolute interfacial tension.

Given the dependence of these 3 variables, we can construct phase spaces to investigate the interface area of a cell doublet at different values of β, which is fundamentally the parameter of interest. We produce the data for these phase spaces by simulating cell doublets as described above, testing a wide range of values of α and γ_m for 3 noteworthy values of β, specifically 0.5, 0.75, and 1.00. The linear force balance model predicts corresponding

interface areas for these value of β of 0.20, 0.11, and 0. Phase spaces of simulation results are shown in the following figures. Figure 6.3a shows the results for $\beta = 0.50$, Figure 6.3b shows the results for $\beta = 0.75$, and Figure 6.3c shows the results for $\beta = 1.00$.

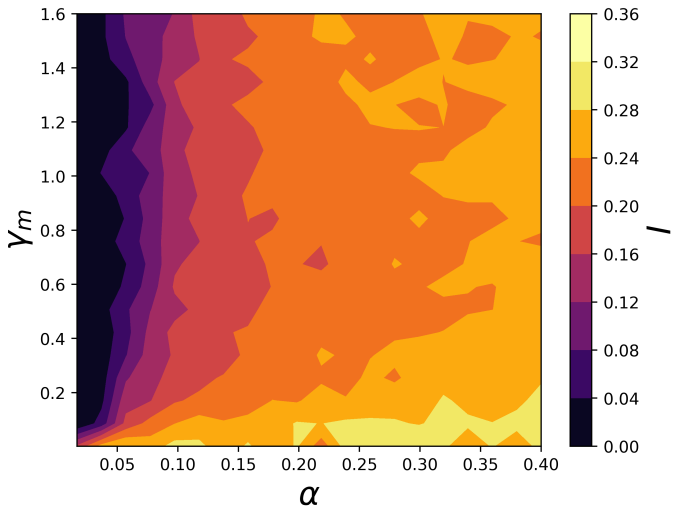
It is clear from these phase spaces that the highest I value achieved for any parameter set is around 0.32, compatible with the theoretical limit for the interface between two hemispheres, which is exactly $1/3$ [Equation 6.3]. This is an encouraging first validation. The phase spaces also clearly show that for each value of β, the measurement of I drops sharply with increasing γ_m, tending to some limit. This limit is determined by the adhesion magnitude, with very low adhesions producing little to no interface. Slightly higher interfaces produce a sharp increase, and beyond a threshold adhesion the increase of interface with adhesion magnitude has a much lower gradient.

$$I_{max} = \frac{A_{circle}}{A_{hemisphere}} = \frac{\pi r^2}{(4\pi r^2)/2 + \pi r^2} = \frac{1}{3} \tag{6.3}$$

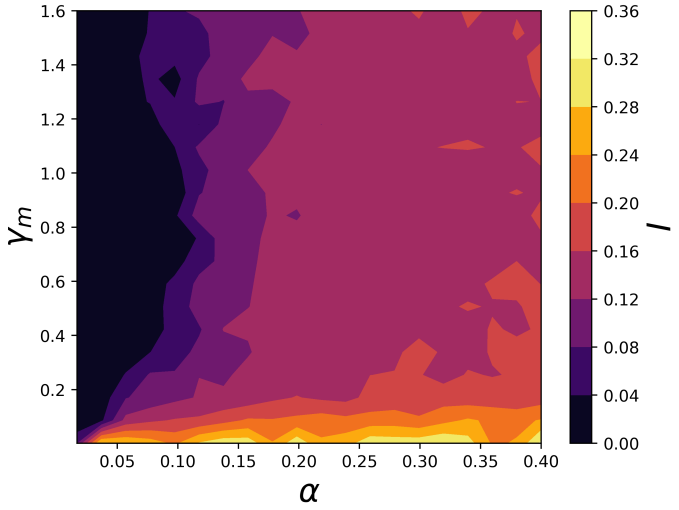
The results become clearer if we take slices through the phase space as shown in Figures 6.4 and 6.5. When considering variation of interface with adhesion magnitude, we can roughly break the phase space into the low tension regime [Figure 6.4a] and the high tension regime [Figure 6.4b].

As expected, in the low tension regime, all interfaces tend to around the theoretical limit for two hemispheres - 0.33, and there is very little variation with β. Meanwhile, in the high tension regime [Figure 6.4b], we see two regimes of adhesion magnitude. The low adhesion regime has low interface values, and interface increases sharply with increasing adhesion. In the higher adhesion regime, we see much shallower increase of interface with adhesion, producing an approximately stationary phase. For each value of β, the value to which this stationary phase tends is just slightly above the linear force balance model prediction for that value. At the lowest adhesion point of the high adhesion regime, the fit between the theoretical model and simulations is particularly good, as demonstrated in Figure 6.4b, which shows a slice of the phase space across cortical tension magnitude for $\alpha = 0.15$. We propose that small difference between the simulations and theoretical model at higher interfaces are due to the theoretical model's neglection of adhesion. It is worth noting that for $\beta = 1.0$ the theoretical model predicts zero interface between cells, whereas in reality it is clear that for anything but hard spheres the adhesion between the cells will produce some non-zero interface. Therefore we suggest that differences between simulation results and theoretical model predictions are due to the assumuption that adhesion can be completely ignored.

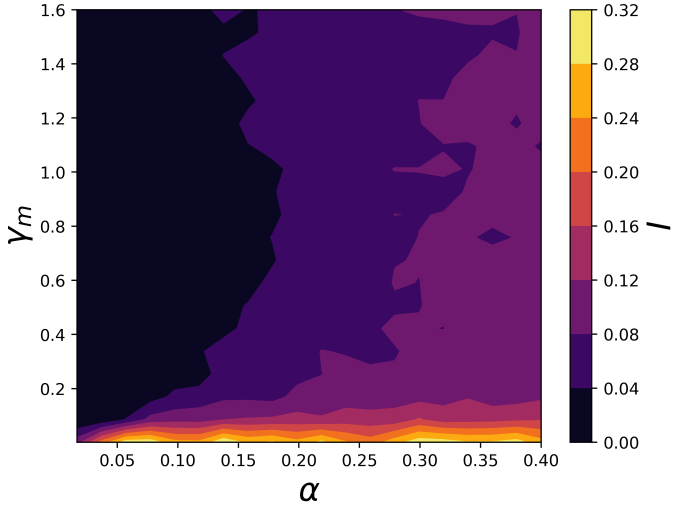
A visualisation of this phase space testing can be seen in Figure 6.1, showing a range of doublet interfaces across γ_m and β values. Note that in our simulations, α is the peak value



(a) $\beta = 0.50$.



(b) $\beta = 0.75$.



(c) $\beta = 1.00$.

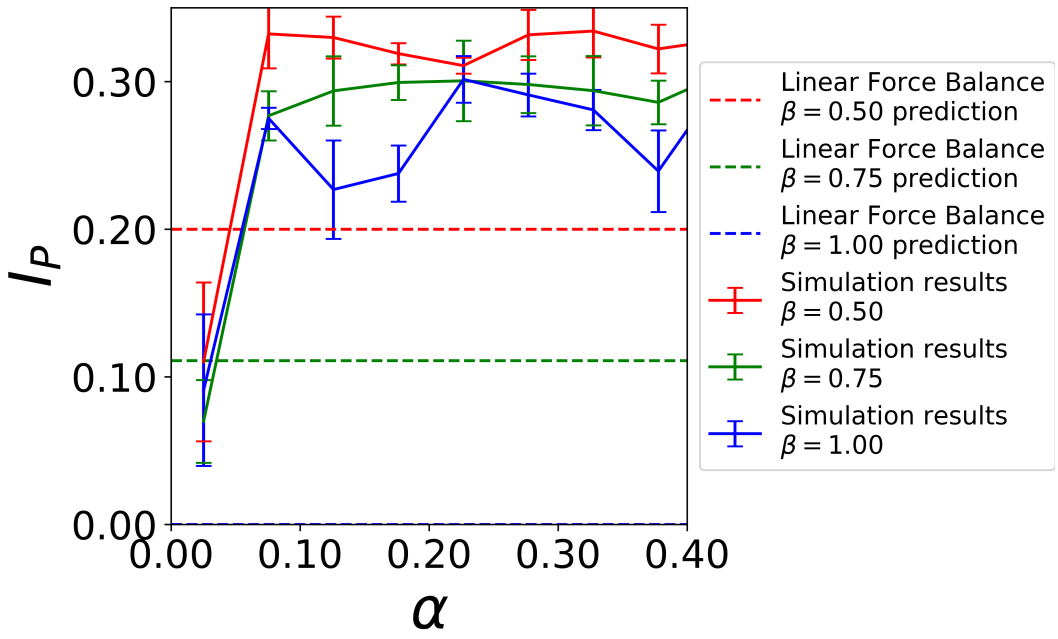
Fig. 6.3 Phase space plots showing interface proportion variation in tension and adhesion space. Linear force balance model predicts $I = 0.20$ for $\beta = 0.50$, $I = 0.11$ for $\beta = 0.75$, $I = 0.00$ for $\beta = 1.00$.

of adhesive inter-cell forces, γ_m is the value of linear tension force acting between cortex elements in the Delaunay triangulation, and β is the factor by which this value is changed at inter-cell interfaces. We also note that these phase diagrams demonstrate some noise due to interpolation of a limited dataset. It is difficult to establish an interpolation that can smooth over the noise without washing out structure from the phase diagrams, but we believe in the absence of additional data the diagrams as shown are sufficient to demonstrate interesting patterns in the results. This caveat applies similarly to all phase diagrams used in subsequent results sections.

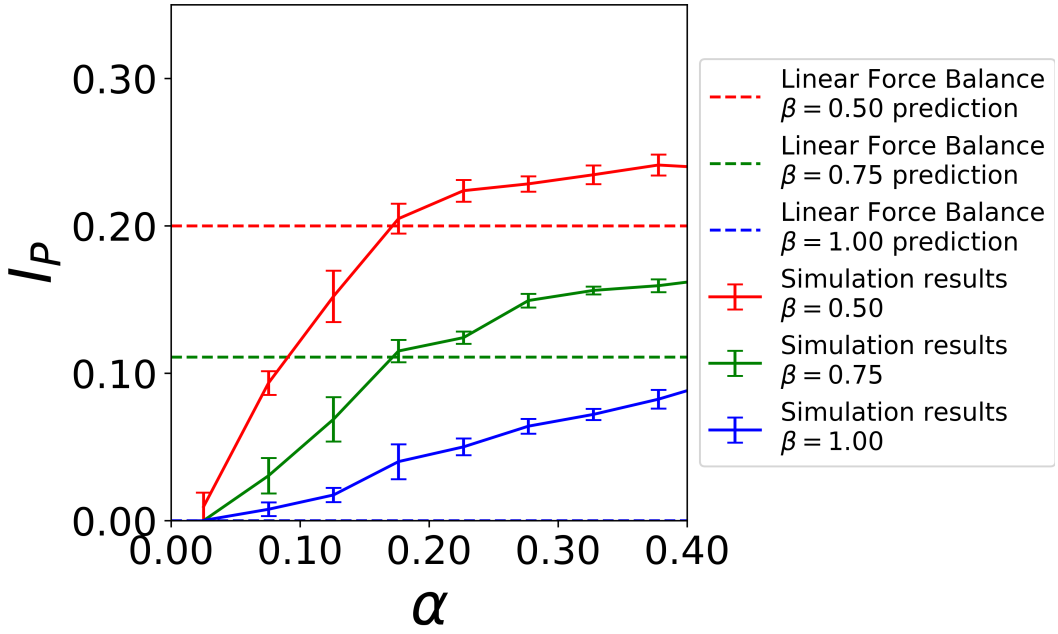
6.1.2 Variation of Interface with Interfacial Tension Factor β

After testing the full phase space of interface areas in tension and adhesion at 3 values of β, we now investigate the variation of interface proportion I with interfacial tension factor β in more detail. Once again the doublet method described above was used to simualate systems and obtain measurements of I for values of β between 0.25 and 1. These tests were performed for a representative set of adhesion, α, and tension, γ_m, values selected from the phase spaces explored previously. The resulting I values were plotted and compared to the theoretical predictions of the linear force balance model, remembering that the linear force balance prediction is of the form shown in Equation 1.1.

Figure 6.6 demonstrates how in the high tension regime, the simulation results fit well to the linear force balance theoretical model across all reasonable values of β, while in the low tension regime interfaces are consistently higher. Within the moderate adhesion regime - high enough to produce any interface at all - at no tension magnitude is is possible to produce interfaces lower than those predicted by the theoretical model, which fits with our expectation given the assumption that adhesion can be neglected in the theoretical model. Figure 6.7 shows how the adhesion magnitude can alter the interface area measurement, increasing or decreasing it by a roughly constant value across β values, but not change the form of the variation with β, which is instead dependent on tension magnitude, γ_m.



(a) Low tension regime. $\gamma_m = 0.01$.



(b) High tension regime, $\gamma_m = 1.20$.

Fig. 6.4 Plots of interface area against adhesion magnitude at $\beta = 0.5, 0.75, 1.00$ in high and low tension regimes. Solid lines show simulation results; dotted lines show linear force balance model predictions for comparison. Differences between the linear force balance model and simulation results are proposed to arise from the linear force balance model neglecting adhesion.

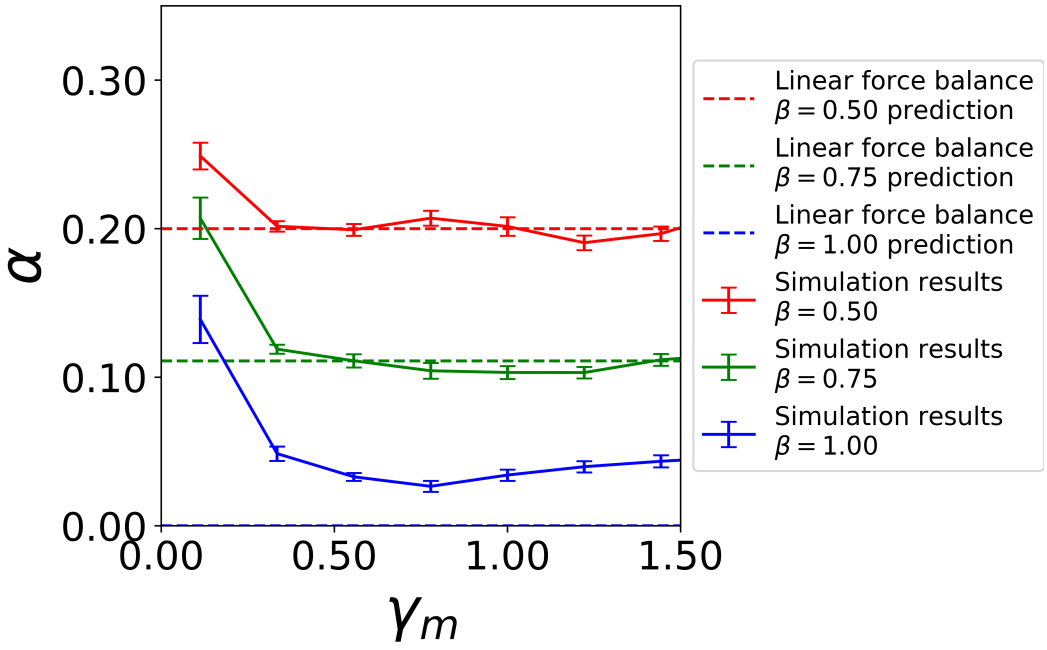


Fig. 6.5 Plots of interface area against tension magnitude at $\alpha = 0.15$ and $\beta = 0.5, 0.75, 1.00$. Solid lines show simulation results; dotted lines show linear force balance model predictions for comparison. $\alpha = 0.15$ is roughly the lowest adhesion above the regime in which no significant interface is produced.

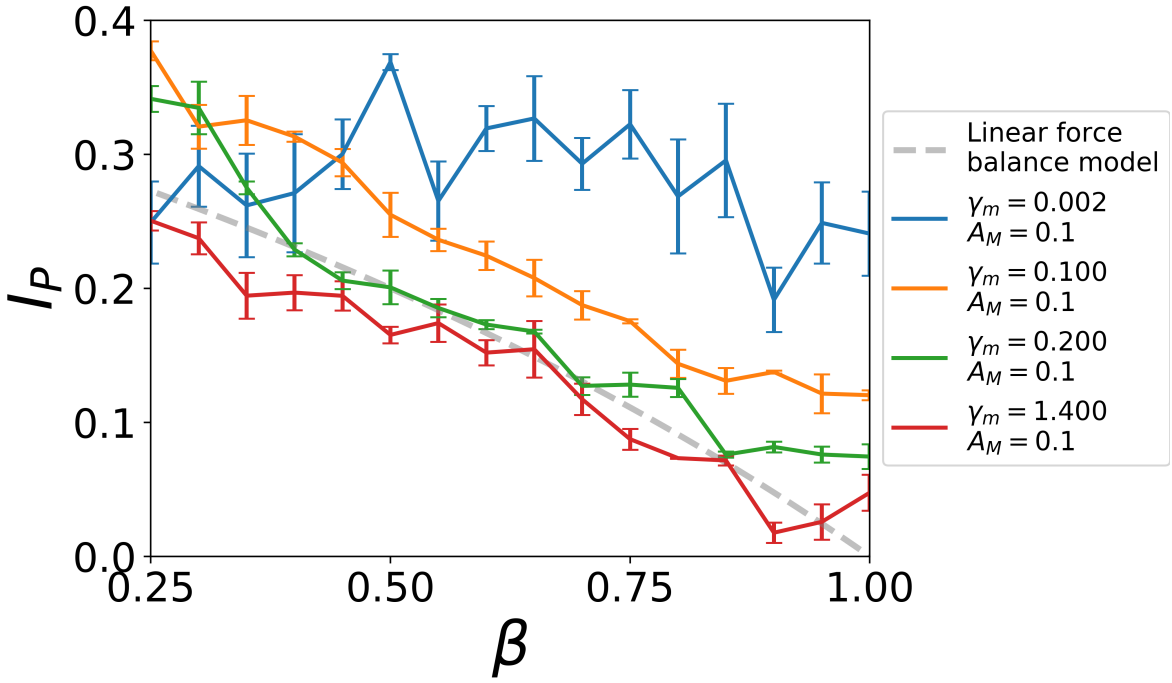


Fig. 6.6 Plots of interface proportion against β, varying γ_m in the low adhesion regime, $\alpha = 0.17$.

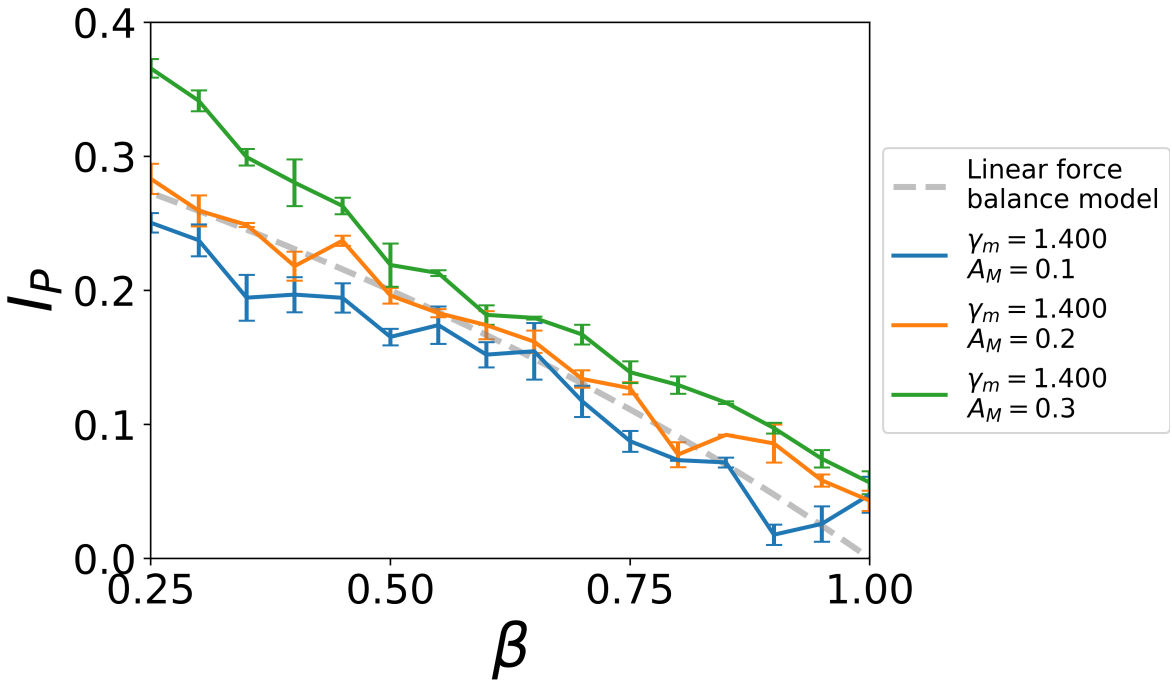


Fig. 6.7 Plots of interface proportion against β, varying α in the high tension regime, $\gamma_m = 1.40$.

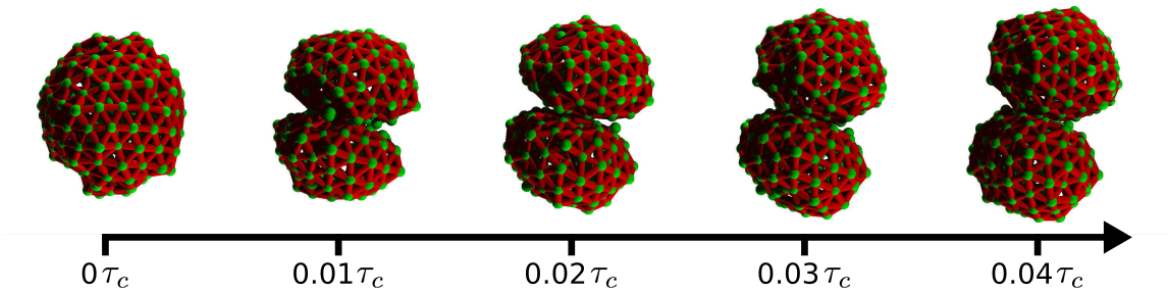


Fig. 6.8 Sequence showing how cortical tension drives the separation of daughter cells immediately after division. 5 time points shown with time post-division shown in units of cell cycle time.

6.2 Testing Energetic Mechanisms Driving Dynamics

In order to investigate the effect of simulation parameters on cell rearrangement we used the displacement measurement described in Chapter 5 to investigate the mean displacement of cells from their initial position for a range of different values of α, γ_m, and ε. We expect cortical tension at the interface between daughter cells to force the daughter cells apart after division [Figure 6.8] and thus predict that higher cortical tension, γ_m, should lead to increased rearrangement. Furthermore, as discussed in Chapter 1, we propose that blebbing observed in primitive endoderm cells may assist in system rearrangement, and thus expect displacement to increase with increasing dynamic tension amplitude, ε.

For each set of parameters, we ran simulations from 10 to 30 cells and collected data on all cells in each system to find the mean displacement of a cell over one full cell cycle from birth at the division of the parent cell until eventual division into two daughter cells.

Figure 6.9 shows the mean displacement of cells after one full cell cycle in the space of reasonable adhesion and tension values. It is clear that whilst there is little variation with adhesion. This is to be expected since cells can move by sliding past one another without necessarily being pulled apart. However, displacement, and hence rearrangement, depends strongly on cortical tension, with systems below a threshold tension exhibiting little displacement, and systems above the threshold tension reaching a stationary phase in which further increasing tension has little effect. This limit may be due to system size.

We next investigated the effect on displacement of introducing dynamic tension into primitive endoderm cells. To do this we simulated systems in the same space of tension and adhesion values as for Figure 6.9, but with all systems having dynamic tension amplitude, ε, not equal to zero. Since we showed in Figure 6.9 that there is almost no variation with adhesion magnitude, we were able to neglect this parameter by averaging over all adhesions.

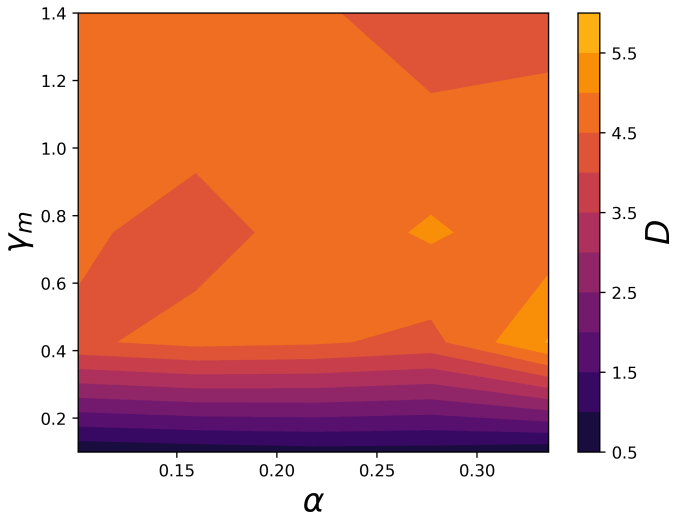


Fig. 6.9 Phase space of mean cell cycle displacement in α and γ_m space.

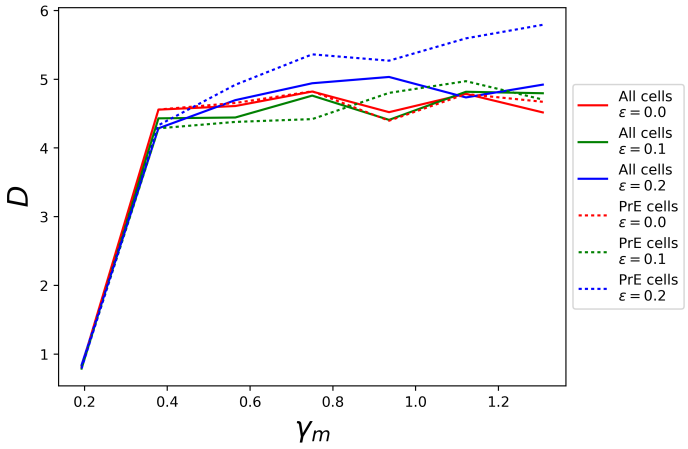


Fig. 6.10 Plot of cell cycle displacement against tension magnitude for $\varepsilon = 0.0, 0.1, 0.2$. Separate plots for the mean displacement over all cells and the mean displacement over only PrE cells indicate that the effect of dynamic tension on displacement is more significant for PrE cells than for the system as a whole.

Figure 6.10 shows the results of this testing. Cell cycle displacement/typical cell radius, D, is plotted against cortical tension magnitude for $\varepsilon = 0.0$, $\varepsilon = 0.1$, and $\varepsilon = 0.2$. This is repeated for an average displacement over all cells, and for the average displacement over primitive endoderm cells alone. These results indicate that dynamic tension does increase the mean cell cycle displacement, albeit not dramatically, but isolating only the displacements of primitive endoderm cells shows a more significant effect than for the mean displacement of all cells.

6.3 Testing Sorting By Differential Interfacial Tension

The final step in this project was to put these pieces together to investigate how the parameters of the system influence the self-organisation of two cell types within the system.

6.3.1 Exploring Extent of Sorting in Adhesion and Interfacial Tension Space

The first test performed was to explore the variation of sorting in cell aggregates with adhesion magnitude and interfacial tension factor. We selected a cortical tension, $\gamma_m = 1.4$, within the high tension regime identified in Section 6.1, and ran simulations with this tension for a range of parameters in the space of adhesion, α, and epiblast interfacial tension, β, values. Note that in an aggregate of 2 cell types we can define 3 interfacial tension factors: for like-like interfaces between both cell types ($\beta_{1,1}$, $\beta_{2,2}$) and unlike interfaces ($\beta_{1,2}$). For simplicity in all simulations we set $\beta_{1,2} = \beta_{2,2} = 1.0$ and vary $\beta_{1,1}$, the interfacial tension factor between epiblasts, henceforth simply referred to as β. The results of Section 6.1 indicated that a range for α between 0.17 and 0.40 and a range for β between 0.5 and 1.0 should suffice to capture interesting dynamics.

Each simulation ran from 10 cells the 30 cells, roughly the same range that occurs within the inner cell mass between 3.5 and 4.5 days after fertilisation, calculating radius, neighbour, and surface sorting measures at regular intervals. 4 separate simulations were performed for each pair of α and β values in the parameter space, and the results from each set of four were averaged. Each test was performed in systems with both symmetric division, meaning each cell can only give rise to daughter cells of the same type, and asymmetric division, meaning each cell has a 50% change of producing two daughter cells of the same type as the parent, and 50% change of producing two daughter cells of different types. For simplicity, both cell types were given the same cortical tension and the same adhesion magnitude, both to themselves and to each other. Furthermore, differential interfacial tension was applied only

to epiblast cells. This assumption is a reasonable first approximation since in experiments performed within our lab the area of PrE-PrE interfaces is approximately equal to that of PrE-Epi interfaces, whereas Epi-Epi interfaces have a significantly larger interface area, indicating that the difference in mechanics arises in epiblast interfaces.

Simulations were performed on the University of Cambridge Darwin HPC facility, running one simulation per core independently, using the Intel FORTRAN compiler.

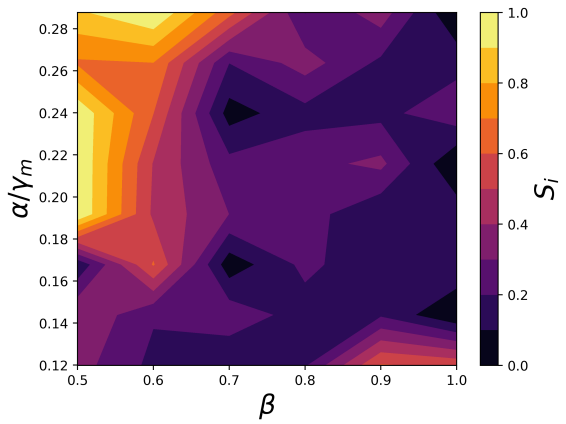
Having collected data for the full parameter space, we were able to take the mean value of the sorting indices defined in Chapter 5 (sorting measures normalised by standard deviation of randomised tests) at the end of each simulation, and plot these values in phase spaces to show the variation of these final states within the α and β space. Any sorting index calculated as greater than 1 was normalised to 1, indicating complete sorting. The resulting phase spaces are shown in Figure 6.11.

It is immediately clear from the results of Figure 6.11 that there is good agreement between sorting measures about the behaviour of the systems. In addition, we note that symmetric division [Figures 6.11b, 6.11d, 6.11f] produces consistently more robust sorting throughout the parameter space, although still retaining the same general pattern with respect to α and β as for asymmetric division. This is to be expected since symmetric division will produce a natural aggregation, whereas asymmetric division can introduce cells of the opposite type into a region that has self-organised to a high density of one type.

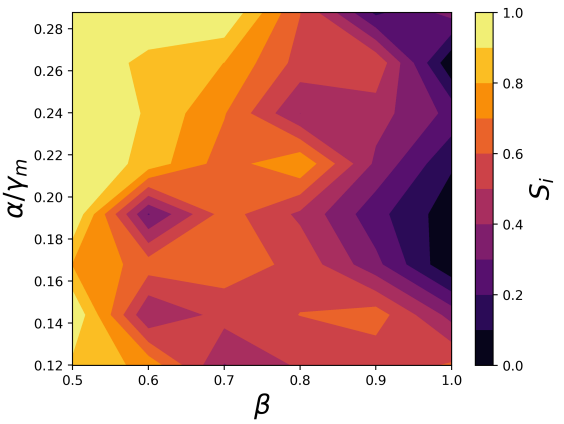
Most importantly, these phase spaces show an interesting pattern with respect to α and β. As predicted, the extent of sorting increases as β is reduced. However, it is clear that a minimum adhesion is required in order for the β parameter to have any effect. For values of α below about 0.25, little to no sorting is observed regardless of the value of β. This is interesting for two reasons. Firstly, $\alpha = 0.25$ is a larger adhesion than produced the best fit to the linear force balance model, as tested in Subsection 6.1.2. Secondly, $\alpha = 0.25$ corresponds to $\alpha = 0.18 \times \gamma_m$, which is in line with experimental observations of adhesion magnitude around 0.2-0.25 times cortical tension magnitude in ICM cells.

6.3.2 Variation of Sorting with Interfacial Area

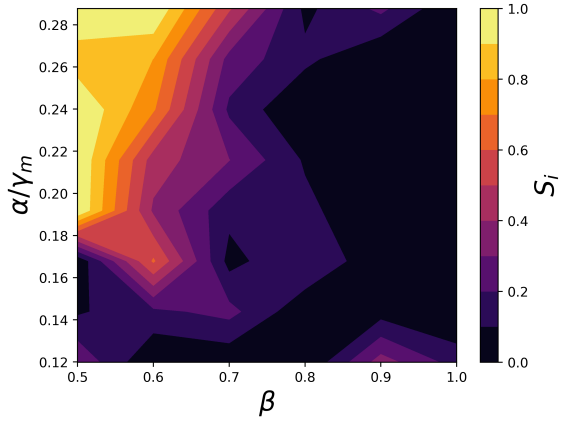
Having studied the variation of sorting in adhesion and tension space, we can go one step further by combining the results of Section 6.1 with our sorting results to test the variation of sorting with epiblast doublet interface area. In all simulations, only epiblast-epiblast interfaces have altered interfacial tension, so we expect the interface of these doublets to be a primary driver of the dynamics. Each set of parameters tested in the sorting phase space has a corresponding epiblast doublet interface proportion as found by cell doublet testing. By plotting the final sorting index values against these corresponding interface proportions we



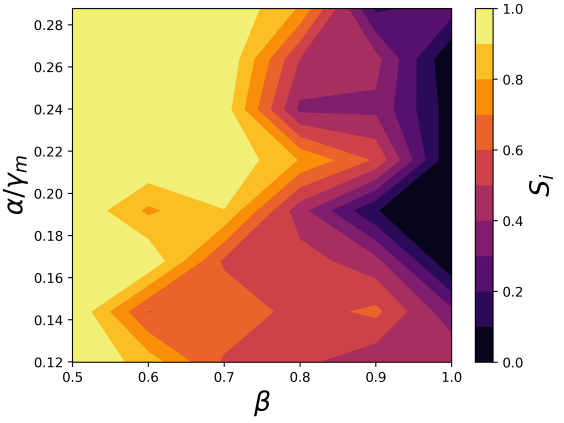
(a) Radius measure, asymmetric division



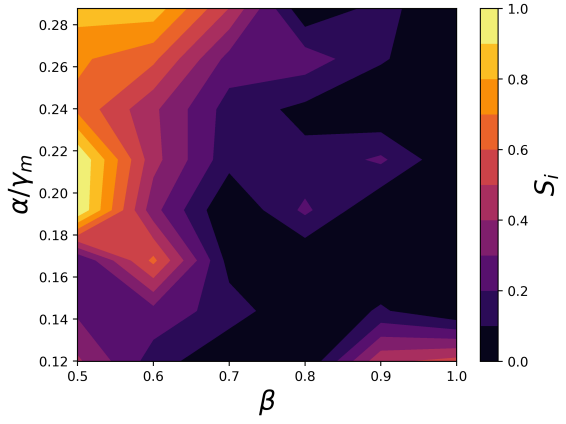
(b) Radius measure, symmetric division



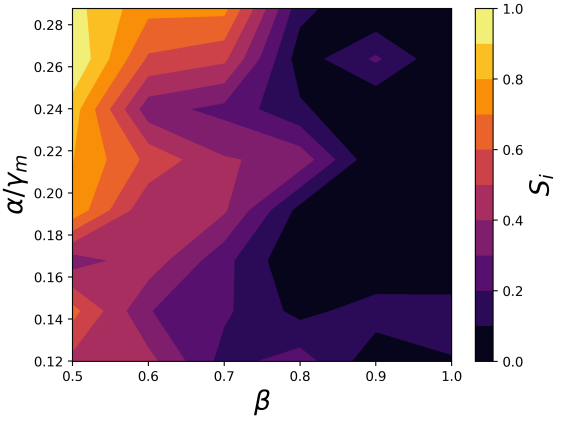
(c) Neighbour measure, asymmetric division



(d) Neighbour measure, symmetric division



(e) Surface measure, asymmetricdivision



(f) Surface measure, symmetricdivision

Fig. 6.11 Phase space plots of final sorting index for 30 cell systems in α and β space. All cells have the same γ_m and all cell types adhere with the same magnitude α. The tension at Epi-Epi interfaces is altered by factor β. Epi-PrE and PrE-PrE interfaces are unchanged. Plots on each row show the values of a different sorting measure, for systems with asymmetric and symmetric division.

produce a scatter plot with which to look for a relationship. The results of this testing are shown in Figure 6.12.

The plots in Figure 6.12 clearly show a strong positive correlation between epiblast interface area and final sorting index for all sorting measures. This correlation is especially strong for systems with symmetric division.

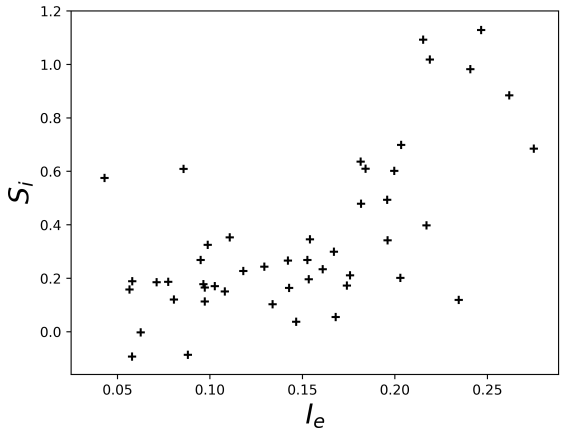
6.4 Effect of Dynamic Tension on Sorting in Adhesion and Interfacial Tension Space

Having investigated the variation of sorting in α and β space, we can now introduce ε, the final parameter of the model. ε is the amplitude of dynamic tension, allowing us to model blebbing in primitive endoderm cells. We begin testing the effect of ε on sorting by running the same α and β parameter space used in Section 6.3, again running each parameter set 4 times, from 10 to 30 cells. However, in this case, the full set of parameters was repeated with different values of epsilon. As before, both cell types have the same cortical tension and adhesion magnitudes, and differential interfacial tension is applied only to epiblasts. However, dynamic tension is not applied to epiblasts since these are not seen to undergo blebbing in experiments.

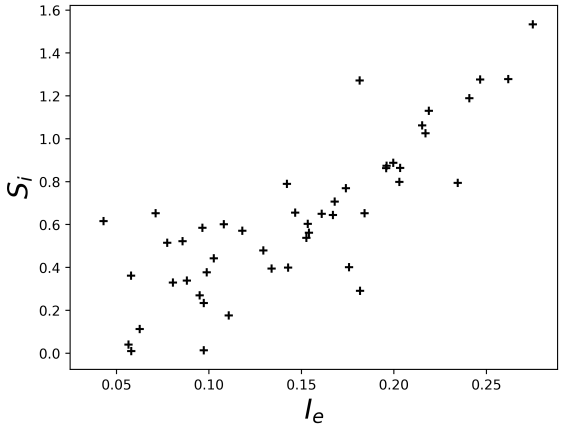
As in Section 6.3, the resulting data were used to construct phase spaces in α and β space, with a different value of ε for each phase space. Figure 6.13 shows the set of these phase spaces for the radius sorting measure, with ε increasing down the page. Similarly, Figure 6.14 and Figure 6.15 show the set of phase spaces for the neighbour and surface measures respectively. It is immediately clear that the action of dynamic tension increases the extent of sorting throughout the α and β parameter space. The extent of this increase in sorting is seen to increase with increasing values of ε. In most cases variation of sorting extent with interfacial tension retains a similar pattern regardless of ε value, but for high values, such as $\varepsilon = 0.3$ and symmetric division, near complete sorting is obtained for the entire parameter space.

We note also that for high values of β, meaning that the effect of differential interfacial tension is minimal, the extent of sorting with ε is reduced for higher adhesion magnitudes, suggesting that the adhesion of cells together to some extent counteracts the effects of dynamic tension.

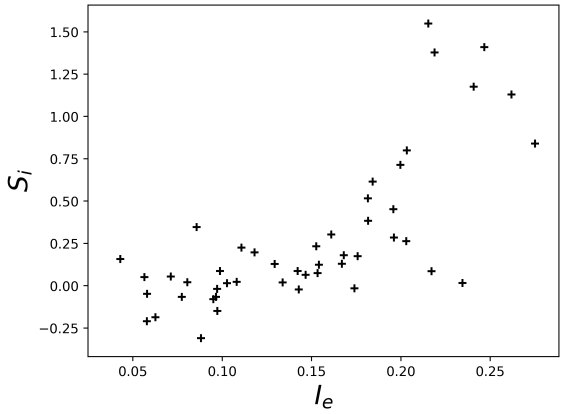
Additionally, we constructed phase spaces in β and ε, taking α such that $\alpha = 0.2 \times \gamma_m$, in line with experimental observations. These phase spaces are shown in Figure 6.16. These phase spaces indicate that for this ratio of tension and adhesion, without dynamic tension



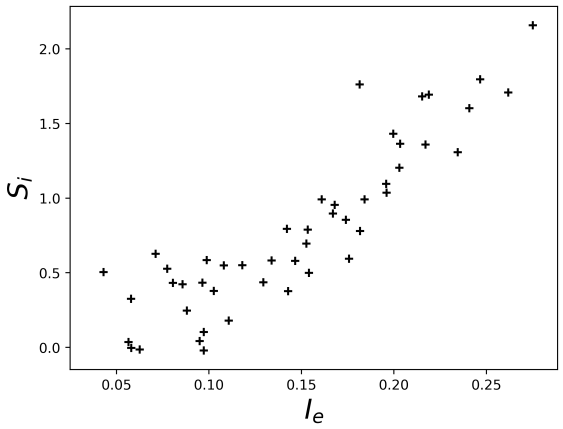
(a) Radius measure, asymmetric division



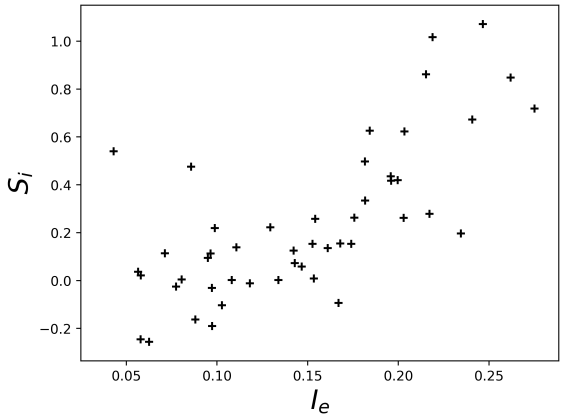
(b) Radius measure, symmetric division



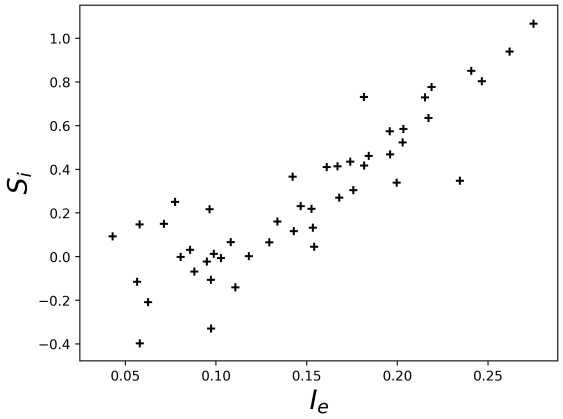
(c) Neighbour measure, asymmetric division



(d) Neighbour measure, symmetric division



(e) Surface measure, asymmetricdivision

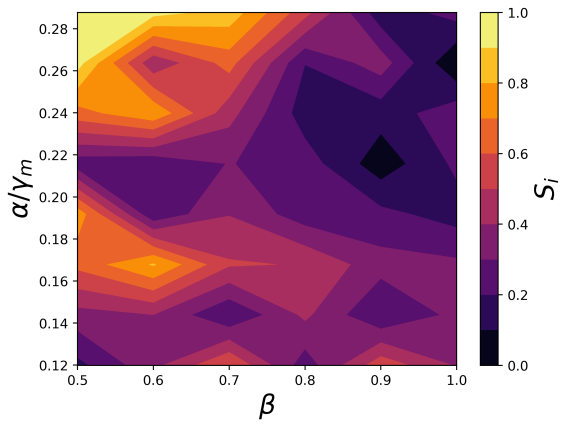


(f) Surface measure, symmetricdivision

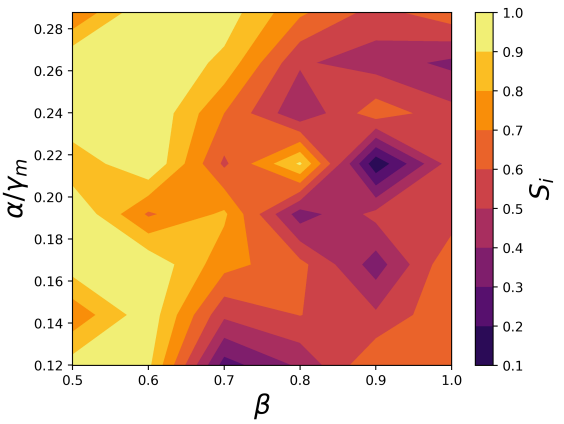
Fig. 6.12 Plots of final sorting index for 30 cell aggregates against the corresponding epiblast doublet interface proportion found in doublet testing [Section 6.1] for the same parameter set. Each row of plots shows values of a different sorting measure for both symmetric and asymmetric division.

sorting is unreliable unless the interfacial tension factor, β, is extremely low. However, adding dynamic tension quickly makes this sorting far more robust.

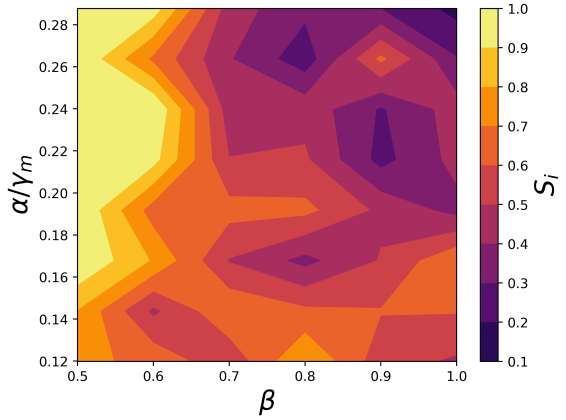
Finally, we repeated the process of plotting final sorting index against interface area obtained in doublet testing with sorting data obtained from systems with $\varepsilon > 0$. Although the new results, shown in Figure 6.17, show a greater spread than those in Figure 6.12, we can see how complete sorting can now be achieved with much lower interface areas than were possible without dynamic tension.



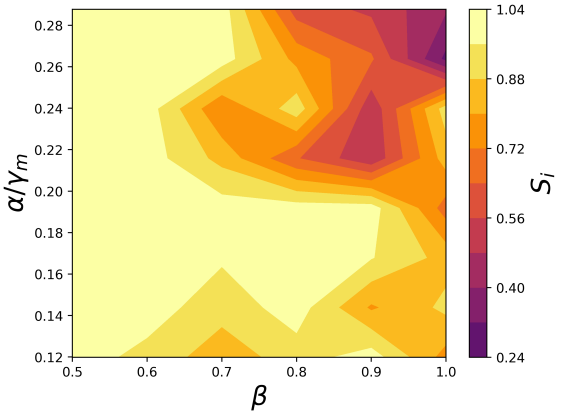
(a) Radius measure, asymmetric division, $\varepsilon = 0.1$



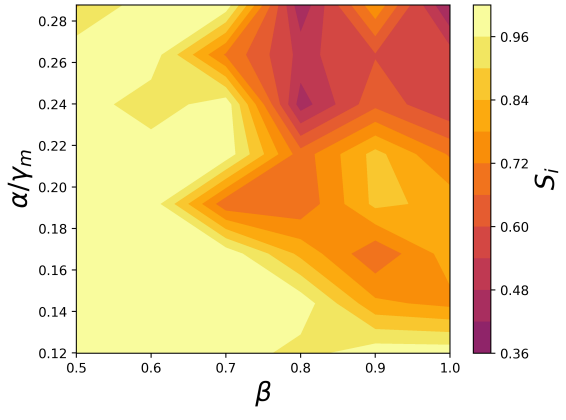
(b) Radius measure, symmetric division, $\varepsilon = 0.1$



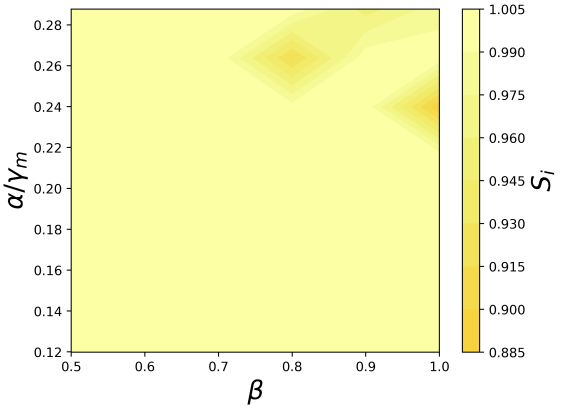
(c) Radius measure, asymmetric division, $\varepsilon = 0.2$



(d) Radius measure, symmetric division, $\varepsilon = 0.2$

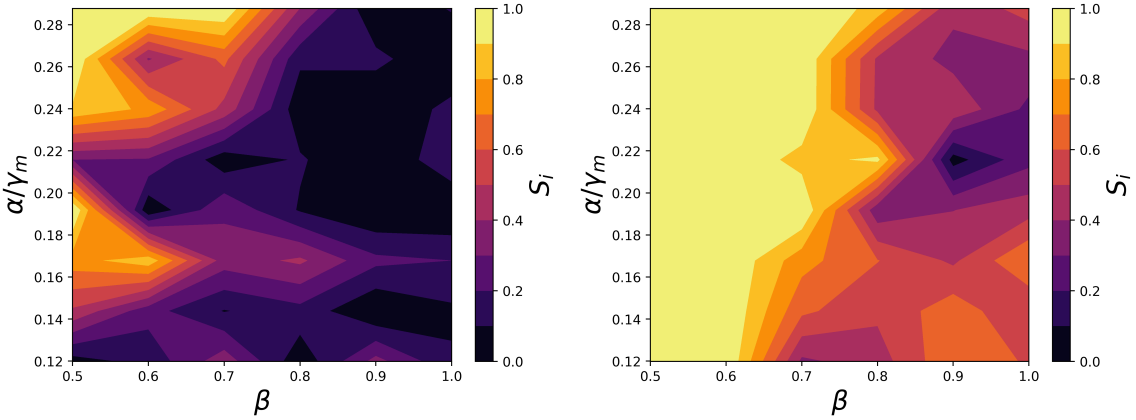


(e) Radius measure, asymmetricdivision, $\varepsilon = 0.3$



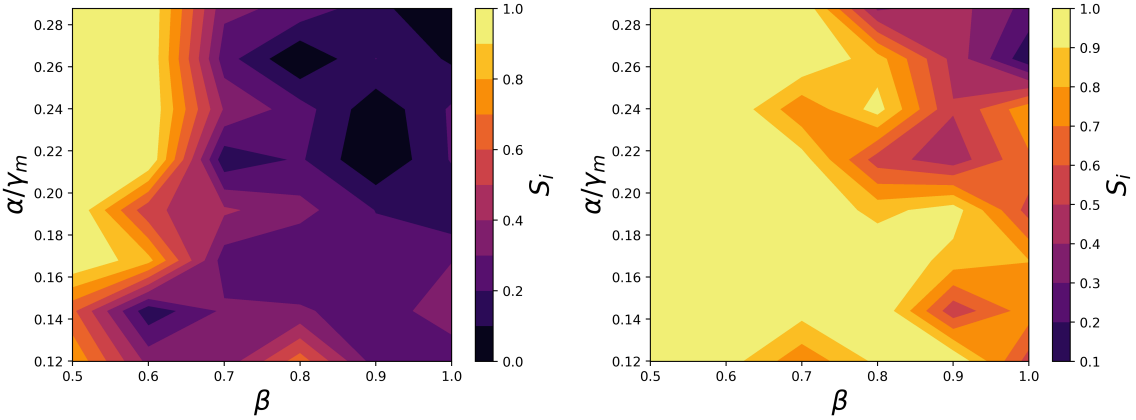
(f) Radius measure, symmetricdivision, $\varepsilon = 0.3$

Fig. 6.13 Phase spaces of final sorting index from radius measure for 30 cell aggregates in α and β space. Dynamic tension is included in primitive endoderm cells with amplitudes $\varepsilon = 0.1, 0.2, 0.3$. Plots on each row show the results for a different ε value, for systems with asymmetric and symmetric division.



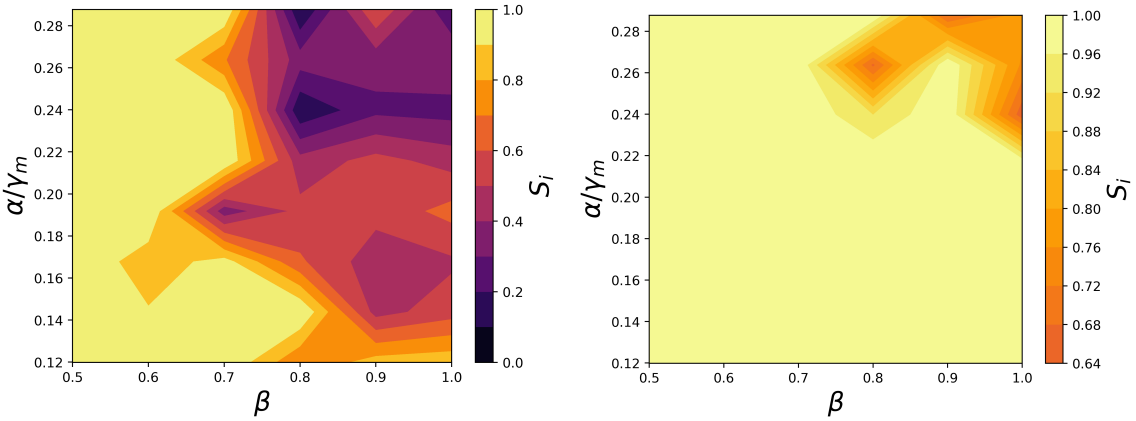
(a) Neighbour measure, asymmetric division, $\varepsilon = 0.1$

(b) Neighbour measure, symmetric division, $\varepsilon = 0.1$



(c) Neighbour measure, asymmetric division, $\varepsilon = 0.2$

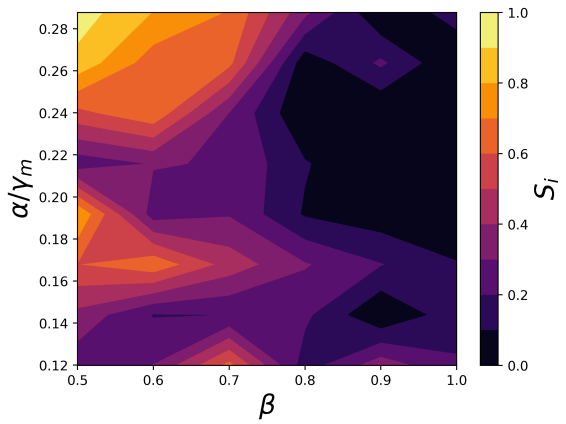
(d) Neighbour measure, symmetric division, $\varepsilon = 0.2$



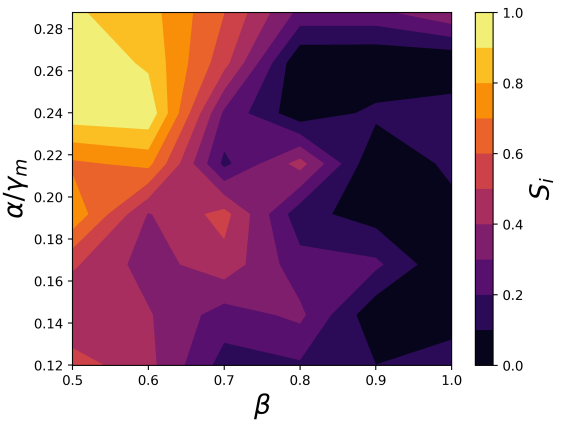
(e) Neighbour measure, asymmetric division, $\varepsilon = 0.3$

(f) Neighbour measure, symmetric division, $\varepsilon = 0.3$

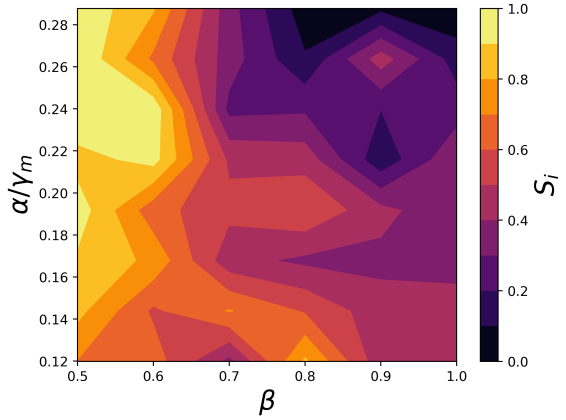
Fig. 6.14 Phase spaces of final sorting index from neighbour measure for 30 cell aggregates in α and β space. Dynamic tension is included in primitive endoderm cells with amplitudes $\varepsilon = 0.1, 0.2, 0.3$. Plots on each row show the results for a different ε value, for systems with asymmetric and symmetric division.



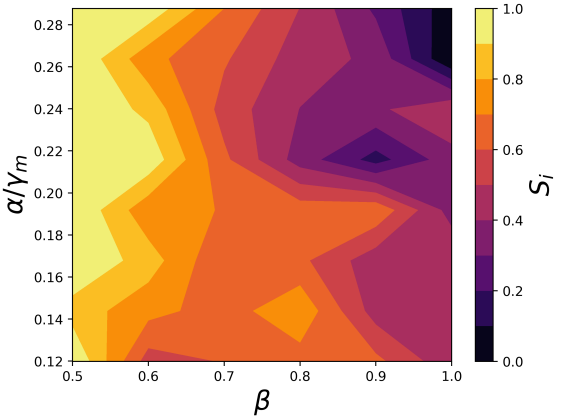
(a) Radius measure, asymmetric division, $\varepsilon = 0.1$



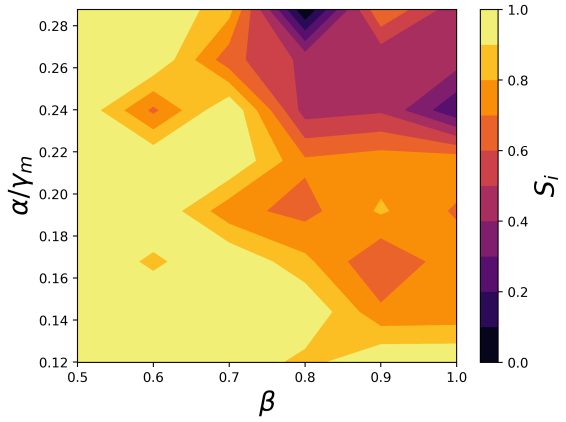
(b) Surface measure, symmetric division, $\varepsilon = 0.1$



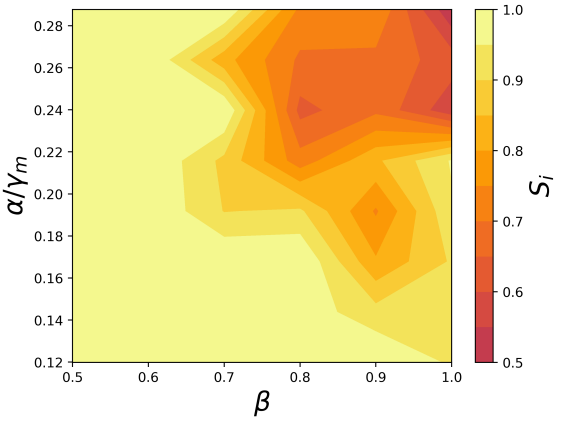
(c) Surface measure, asymmetric division, $\varepsilon = 0.2$



(d) Surface measure, symmetric division, $\varepsilon = 0.2$

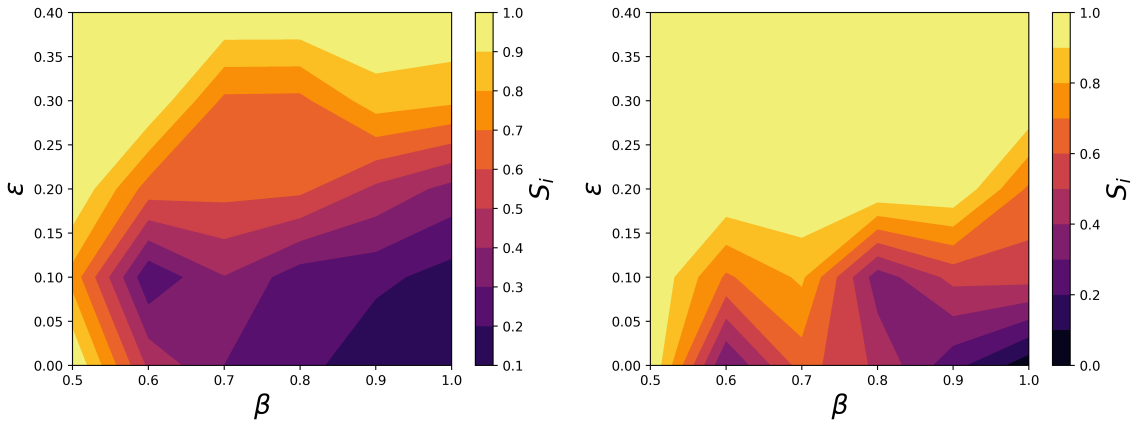


(e) Surface measure, asymmetricdivision, $\varepsilon = 0.3$

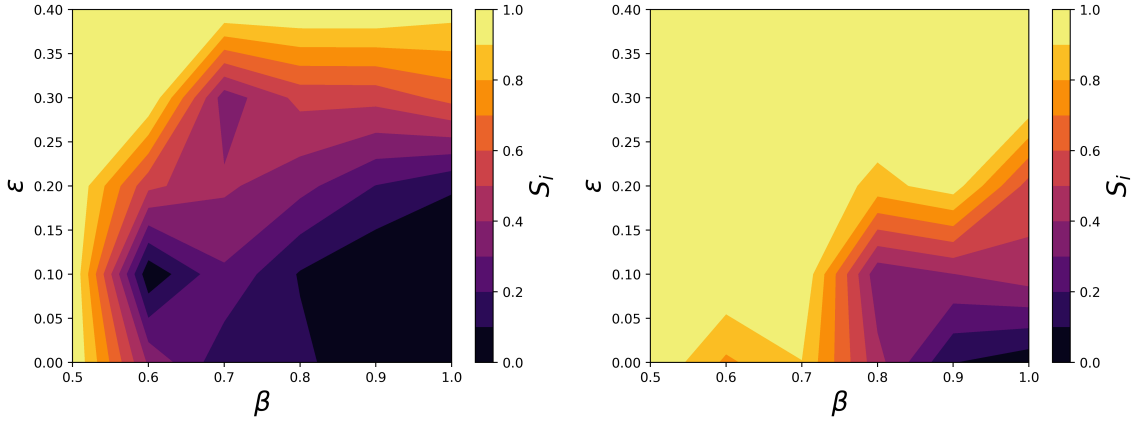


(f) Surface measure, symmetricdivision, $\varepsilon = 0.3$

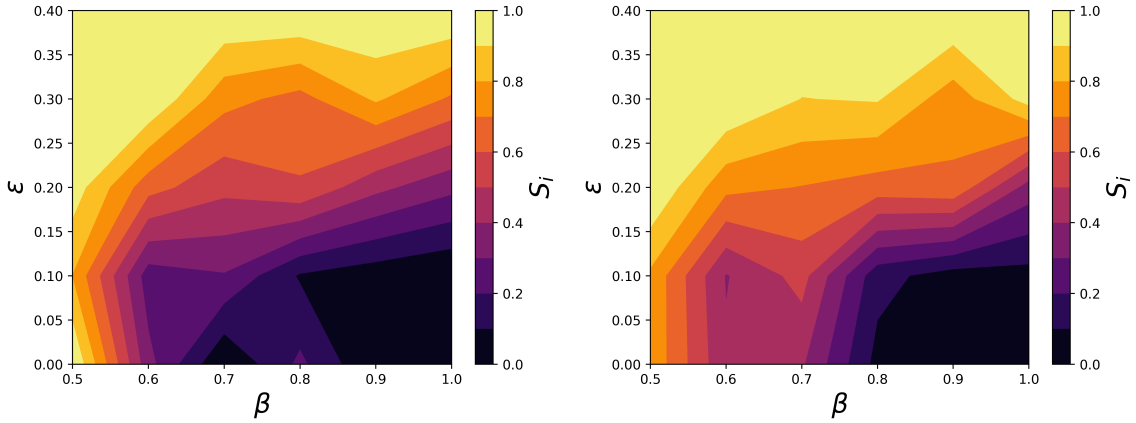
Fig. 6.15 Phase spaces of final sorting index from surface measure for 30 cell aggregates in α and β space. Dynamic tension is included in primitive endoderm cells with amplitudes $\varepsilon = 0.1, 0.2, 0.3$. Plots on each row show the results for a different ε value, for systems with asymmetric and symmetric division.



(a) Radius measure, asymmetric division (b) Radius measure, symmetric division

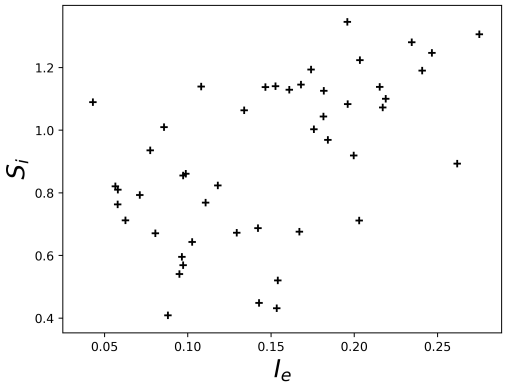


(c) Neighbour measure, asymmetric division (d) Neighbour measure, symmetric division

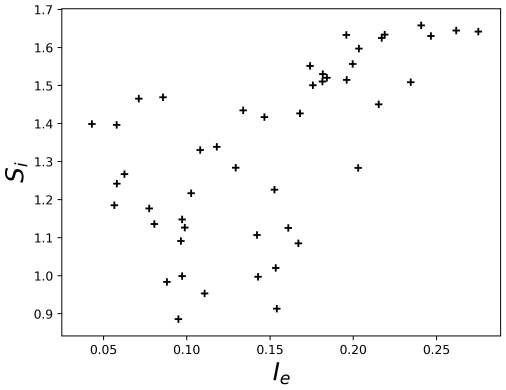


(e) Surface measure, asymmetric division (f) Surface measure, symmetric division

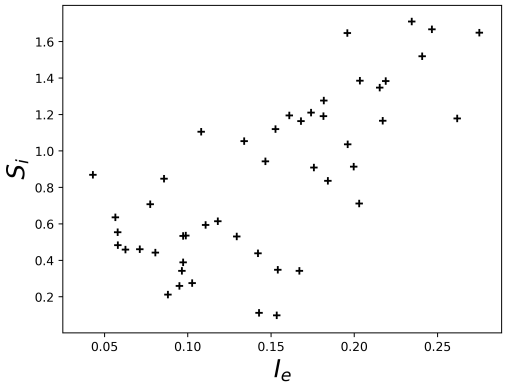
Fig. 6.16 Phase space of final sorting index of 30 cell aggregates in β and ε space at $\alpha = 0.2\gamma_m$, corresponding to the adhesion magnitude observed in experiments.



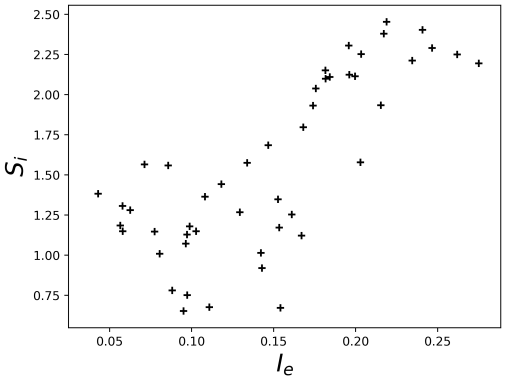
(a) Radius measure, asymmetric division, $\varepsilon = 0.30$



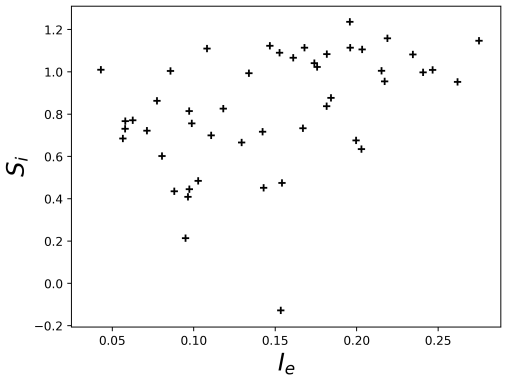
(b) Radius measure, symmetric division, $\varepsilon = 0.30$



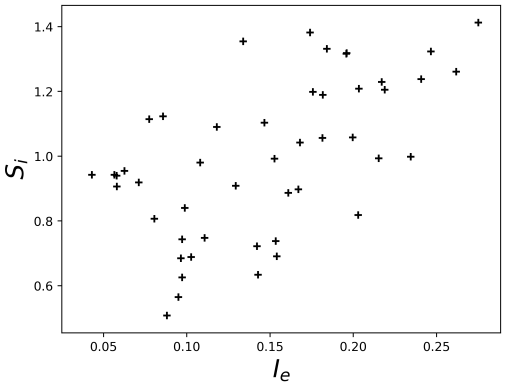
(c) Neighbour measure, asymmetric division, $\varepsilon = 0.30$



(d) Neighbour measure, symmetric division, $\varepsilon = 0.30$



(e) Surface measure, asymmetricdivision, $\varepsilon = 0.30$



(f) Surface measure, symmetricdivision, $\varepsilon = 0.30$

Fig. 6.17 Plots of final sorting index for 30 cell aggregates with $\varepsilon = 0.3$ against the corresponding epiblast doublet interface proportion found in doublet testing [Section 6.1] for the same parameter set. Each row of plots shows values of a different sorting measure for both symmetric and asymmetric division. Compare to Figure 6.12 to see the impact of dynamic tension.

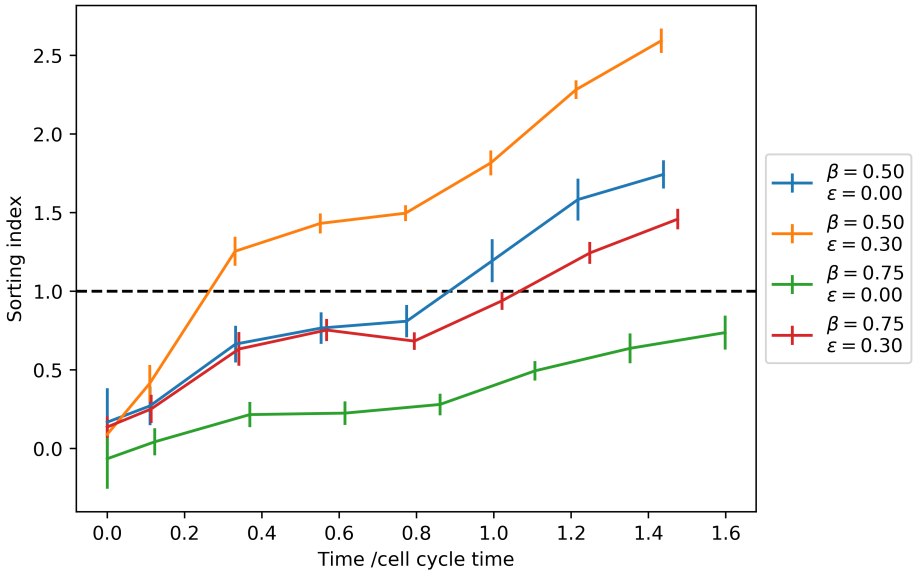


Fig. 6.18 Time series plot of neighbour sorting index in a system running from 10 to 30 cells with symmetric division and $\alpha = 0.2\gamma_m$.. 4 curves are shown, each with different β and ε values, with each parameter set averaged over 4 runs. Horizontal dotted line shows the 3σ threshold, beyond which sorting is considered complete.

6.5 Sorting Kinetics

Having studied the behaviour of systems by comparing final sorting indices, we will now examine the time series of sorting index data as simulations progress. Figure 6.18 shows a plot of neighbour sorting index against time for 4 parameter sets, each averaged over 4 runs. What we see is that whilst 3 of the 4 systems are capable of reaching the 3σ threshold, the introduction of dynamic tension allows the systems to reach this threshold much more quickly. To better understand these systems, a time series visualisation is shown in Figure 6.19.

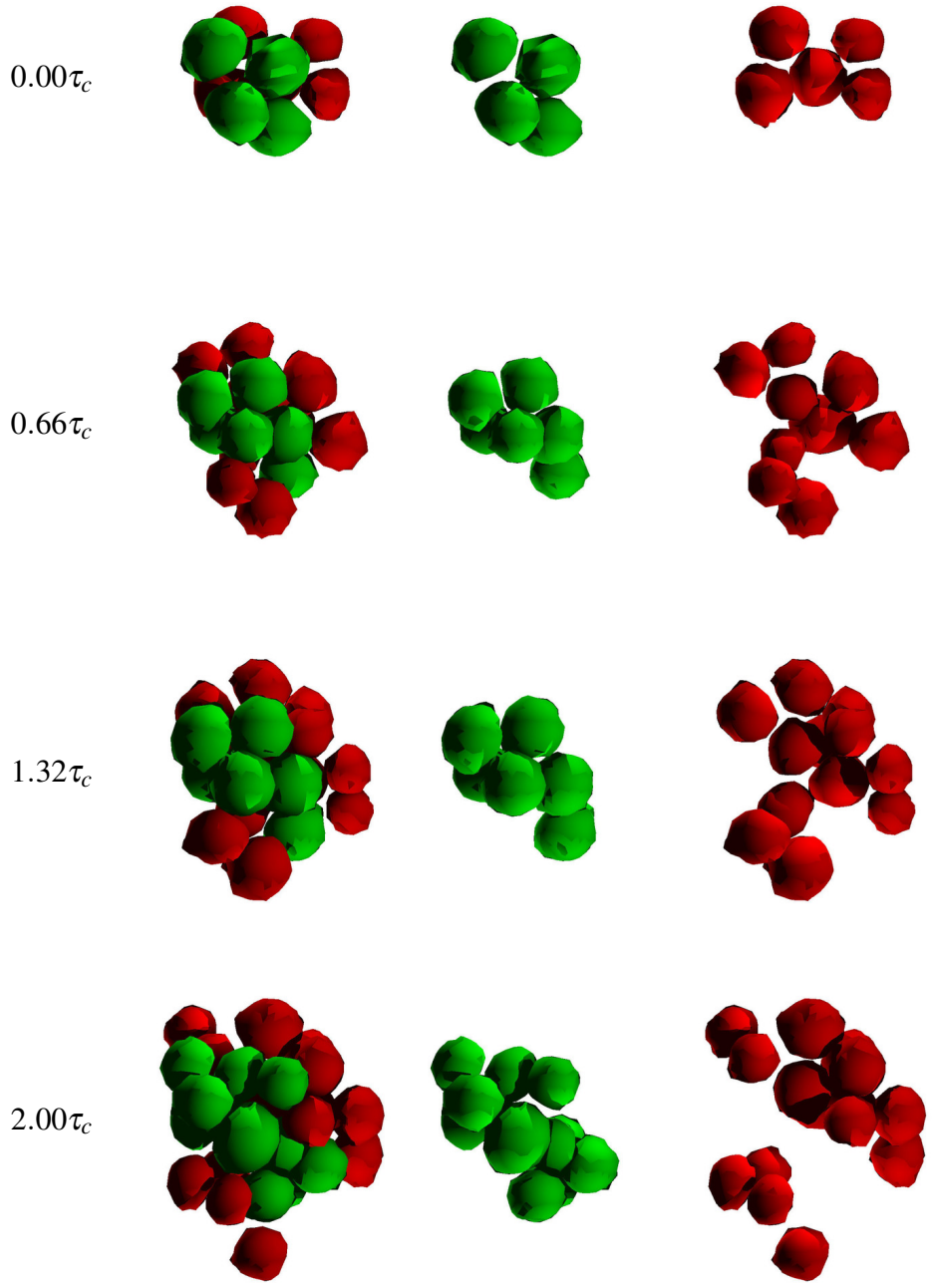


Fig. 6.19 Time sequence showing sorting in a system from 10 to 30 cells with $\beta = 0.7, \varepsilon = 0.3$ and asymmetric division. Time since system initialisation shown in units of τ_c. Moderately low β and moderately high ε lead to complete sorting despite asymmetric division. Left column shows full cell aggregate. Epiblasts coloured green, primitive endoderm coloured red. Middle and right columns show each cell type visualised independently to highlight their relative positions.

Chapter 7

Conclusions

7.1 Conclusions

This thesis describes the first steps taken in an effort to form a comprehensive model of tissue self-organisation in the early mammalian embryo, and in doing so introduce a new computational model of cell sorting. We have discussed the basics of mammalian embryogenesis as they are currently understood, highlighting gaps in our understanding of development between 3.5 and 4.5 days after fertilisation [Chapter 1]. After explaining this phase of development in the context of blastocyst formation and the formation of embryonic and extra-embryonic tissues, we focussed in more detail on the inner cell mass of the blastocyst over this E3.5-E4.5 period. We discussed how at 4.5 days after fertilisation, the inner cell mass is seen to contain two spatially separate tissue layers, which were originally assumed to differentiate *in situ*, but that more recent observations suggest that two cell types differentiate in a mixed "salt and pepper" distribution throughout the inner cell mass. The subsequent self-organisation of these two cell types is not well understood, and it is this self-organisation we set out to study.

Before discussing our model, we introduced other techniques that have been used to model cellular systems in the past, focussing in particular on the lengthscales and number of dimensions to which each model was adapted, the integration of intra- and inter-cellular mechanisms, and whether or not the method relied on a global Hamiltonian [Chapter 2]. We established that many techniques applied to cellular systems had significant drawbacks that rendered them unsuitable for modelling the inner cell mass, and that the best option for our purposes was the subcellular element method [Chapter 3]. The advantages of this technique include the integration of intra- and inter-cellular mechanisms, ease of extension to 3 dimensions, suitability for a scale of tens of cells, and focus on local forces rather than

global energy. Furthermore the technique was simple enough to make extending it fairly straightforward.

We composed our model by defining a set of cortex elements at the boundary of each cell. We introduced a tensile force tangential to the cell surface by performing a Delaunay triangulation over this set of cortex elements and introducing a tension force between all neighbouring elements in the triangulation, thus modelling cortical tension. This was extended with an algorithm to locally vary the cortical tension between cortex elements at interfaces between cells according to the types of cells involved, allowing us to model differential interfacial tension. We also added dynamic tension to one cell type, varying the tension forces experienced by cortex elements randomly across the surface of each cell to model blebbing in primitive endoderm cells [Chapter 4].

With the model implemented, we created a further set of routines to calculate quantitative measures of sorting based on the number of nearest neighbours of the same type, the radius of each type from the centre of mass, and the proportion of each cell type at the system external surface. These measures were contextualised relative to the mean and standard deviation across a large number of random reorientations of the system in order to produce a sorting index [Chapter 5].

Having defined this set of measures, we began to test the model. Our first test was to simulate cell doublets, calculating the interface area formed between the two cells in a doublet as a proportion of the total cell surface area [Chapter 6, Section 6.1]. We established that these interface proportions depended on 3 variables of the model: baseline cell tension, γ_m, adhesion magnitude, α, and the ratio of interfacial tension to baseline cortical tension, β. By measuring interface proportions for a range of values in the space of these 3 variables, we were able to show that interface decreases sharply with tension before reaching a stationary phase in the high tension regime. We also showed that below some threshold value of adhesion, no interface is formed at all, but above this threshold interface increases slowly with increasing adhesion magnitude, with the gradient of this interface sharpest for low values of β. We showed that in the high tension regime, and with adhesion at the low end of this interface-forming adhesion regime, the variation of interface with β showed remarkably good fit to the predictions of the previously proposed linear force balance model of cell interfaces, which assumes low adhesion relative to tension. This was an extremely promising validation and indicated that the model was capturing important aspects of the underlying mechanics. We also showed, using the mean displacement of cells from their original position over one full cell cycle as a proxy for cell rearrangement and hence energetic mechanisms driving the system kinetics, that rearrangement is heavily dependent upon a minimum cortical tension, but only weakly dependent on the amplitude of dynamic tension [Chapter 6, Section 6.2].

Having performed tests to validate that the model is capturing cell dynamics, and using the quantitative measures defined previously, we tested the effect of model parameters on self-organisation. We simulated systems growing from 10 to 30 cells, with both symmetric and asymmetric cell division. Initially, we tested systems in adhesion, α, and interfacial tension factor, β, space, with no dynamic tension. To do this we took the final values of sorting indices to represent the final state of the system at 30 cells, and plotted these values in phase spaces against α and β. These phase spaces demonstrated that differential interfacial tension alone was sufficient to produce complete sorting for systems with low β. The extent of sorting was greater for systems with symmetric division than asymmetric division, but even in the case of symmetric division, little to no sorting occurred for systems with $\alpha < 0.2\gamma_m$. This is interesting since this adhesion is greater than that which produces the best fit to the linear force balance model predictions, but is in line with the experimentally observed adhesion magnitude in real cells. However, even with sufficient adhesion, quite low values of β are required to produce sorting, lower than the value of about 0.75 observed in measurements in our group. This is best exemplified when we plot the final sorting index for these 30 cell systems against the corresponding interface proportion found in doublet testing with the same parameters. Whilst we do observe an extremely strong correlation between final sorting index and the interface proportion measured in doublets, these plots suggest that the interface areas observed experimentally in our group are insufficient to reliably produce complete sorting.

The next step in our testing was to introduce dynamic tension in primitive endoderm cells, with the amplitude of this dynamic tension, ε, forming another variable of the model. We reproduced the same set of simulations in α and β space, repeating each set with a different value of epsilon. Plotting each of these phase spaces separately, we see that the effect of dynamic tension in primitive endoderm is to increase the extent of sorting throughout the parameter space. Apart from cases of very high values of ε and symmetric division, in which case dynamic tension appears to produce sorting for all values of α and β, the extent of sorting retains a similar pattern of dependence on adhesion and interfacial tension, but complete sorting is more robustly achieved at a greater range of the parameter space. The extent of this increase in sorting is seen to scale with the magnitude of ε. We also produced phase spaces of final sorting index in ε and β space for systems with $\alpha = 0.2\gamma_m$, as observed in experiments, which further confirmed the pattern of dependence on β and ε.

Thus, in summary, we find that differential interfacial tension is sufficient to drive systems to self-organise given high enough adhesion and low enough interfacial tension, but find that this self-organisation is not robust for the parameters found experimentally in inner cell mass cells. Furthermore, we find that the introduction of blebbing in primitive endoderm cells

produces far more robust sorting throughout the tension and interfacial tension parameter space. We thus propose that whilst differential interfacial tension may have some role to play in self-organisation in the inner cell mass, the blebbing observed in primitive endoderm cells will have just as significant a role. The introduction of blebbing into a model of cell sorting, and finding that it has a significant impact on the extent and speed of sorting is a novel result and an important conclusion from this project.

7.2 Remaining Questions

Although we have tested our model extensively, the technique shows great promise and warrants further work in future. In the first instance, we hope to acquire additional data for the parameter spaces already tested in order to validate the patterns observed in phase diagrams.

In all of the testing of sorting we performed in this project, both cell types were given the same cortical tension and adhesion magnitudes. This is a worthwhile first assumption but it may be interesting to test further differences between the two cell types. Furthermore, we have assumed throughout that the tension at primitive endoderm-primitive endoderm and primitive endoderm-epiblast interfaces is unchanged. As discussed previously, in any multicellular aggregate there are 3 possible interfacial tension factors: β_{11}, β_{12}, β_{22}. In our systems $\beta_{12} = \beta_{22} = 1$ and only β_{11} is varied. Whilst this was also a valid assumption for initial testing, it will be interesting in future to explore the effects of changing interfacial tensions in both cell types, not just epiblast cells. It would also be worth testing systems to a greater number of cells to investigate the effects, in principle, of the parameters on aggregates of this size, even though they are larger than the inner cell mass.

All testing performed so far has been done on spherical cell aggregates. However, in the inner cell mass *in vivo*, the cells are arranged in a geometry closer to that of a spherical cap, with primitive endoderm cells organising to the flat edge of the cap and epiblast cells organising to the curved edge. A natural extension would be to apply this geometry to the simulations, restricting cells to move within a spherical cap-shaped boundary and analysing the effects of parameters on the relative final positions of the two cell types.

Finally, the model could prove useful for other multicellular systems in which shape changes, tension, and adhesion are important. For example, the metastasis is a critical process in the development of dangerous cancers, and involves the movement of cancerous cells through crowded multicellular environments. Such a system could potentially be investigated using our model, as the movement of cancerous cells requires significant shape changes, and is likely to be affected by tension and adhesion magnitudes. Furthermore, outside of

the realm of biological systems, our method could be used to study the sedimentation of deformable colloids when centrifuged from syspension. Such systems are extremely difficult to study with analytical methods or standard granular physics, so our method could provide useful insights.

References

[1] Wilhelm Roux. *Gesammelte Abhandlungen über Entwickelungsmechanik der Organismen*, volume 1. Wilhelm Engelmann, 1895.

[2] Klaus Sander. Wilhelm roux and the rest: Developmental theories 1885–1895. *Roux's Archives of Developmental Biology*, 200(6):297–299, Nov 1991.

[3] S. J. Counce. Archives for developmental mechanics W. Roux, editor (1894–1924). *Roux's Archives of Developmental Biology*, 204(2):79–92, 1994.

[4] Erwin Schrodinger. *What is life?* Cambridge University Press, 1943.

[5] Christina Agapakis. Feynman on Biology. *Scientific American*, 2013.

[6] Max Delbrück. *A physicist looks at biology*. Connecticut Academy of Arts and Sciences, 1949.

[7] Geoffrey B West. A theoretical physicist's journey into biology: from quarks and strings to cells and whales. *Physical Biology*, 11(5):053013, 2014.

[8] Tim Newman. Biology is simple. *Physical Biology*, 12(6):063002, 2015.

[9] Yuri Lazebnik. Can a biologist fix a radio? Or, what I learned while studying apoptosis. *Cancer Cell*, 2(3):179–182, Sep 2002.

[10] Charles W Wolgemuth. Trend: Does cell biology need physicists? *Physics*, 4:4, 2011.

[11] Per Bak. *How nature works: the science of self-organized criticality*. Springer Science & Business Media, 1996.

[12] Scott Camazine. *Self-organization in biological systems*. Princeton University Press, 2003.

[13] Christopher M Dobson. Protein folding and misfolding. *Nature*, 426(6968):884, 2003.

[14] Iain D Couzin and Jens Krause. Self-organization and collective behavior in vertebrates. *Advances in the Study of Behavior*, 32:1–75, 2003.

[15] H. Hildenbrandt, C. Carere, and C.K. Hemelrijk. Self-organized aerial displays of thousands of starlings: a model. *Behavioral Ecology*, 21(6):1349, 2010.

[16] W Ross Ashby. Principles of the self-organizing dynamic system. *The Journal of General Psychology*, 37(2):125–128, 1947.

[17] Jean-Léon Maître. Mechanics of blastocyst morphogenesis. *Biology of the Cell*, 109(9):323–338.

[18] Gabor Forgacs and Stuart A Newman. *Biological physics of the developing embryo*. Cambridge University Press, 2005.

[19] HPM Pratt, CA Ziomek, WJD Reeve, and MH Johnson. Compaction of the mouse embryo: an analysis of its components. *Development*, 70(1):113–132, 1982.

[20] Minjung Kang, Anna Piliszek, Jérôme Artus, and Anna-Katerina Hadjantonakis. FGF4 is required for lineage restriction and salt-and-pepper distribution of primitive endoderm factors but not their initial expression in the mouse. *Development*, 140(2):267–279, 2012.

[21] Nicolas Dard, Manuel Breuer, Bernard Maro, and Sophie Louvet-Vallée. Morphogenesis of the mammalian blastocyst. *Molecular and Cellular Endocrinology*, 282(1-2):70–77, 2008.

[22] Nami Motosugi, Tobias Bauer, Zbigniew Polanski, Davor Solter, and Takashi Hiiragi. Polarity of the mouse embryo is established at blastocyst and is not prepatterned. *Genes & Development*, 19(9):1081–92, 5 2005.

[23] H Honda, N Motosugi, T Nagai, M Tanemura, and T Hiiragi. Computer simulation of emerging asymmetry in the mouse blastocyst. *Development*, 135(8):1407–1414, 2008.

[24] Hiroshi Sasaki. Mechanisms of trophectoderm fate specification in preimplantation mouse development. *Development, Growth & Differentiation*, 52(3):263–273, 2010.

[25] Sueo Niimura. Time-lapse videomicrographic analyses of contractions in mouse blastocysts. *Journal of Reproduction and Development*, 49(6):413–423, 2003.

[26] RM Schultz, J Rossant, and RA Pedersen. *Experimental Approaches to Mammalian Embryonic Development*. Cambridge University Press, 1986.

[27] Claire Chazaud and Yojiro Yamanaka. Lineage specification in the mouse preimplantation embryo. *Development*, 143(7):1063–1074, 2016.

[28] Yusuke Marikawa and Vernadeth B. Alarcón. Establishment of trophectoderm and inner cell mass lineages in the mouse embryo. *Molecular Reproduction and Development*, 76(11):1019–1032, 2009.

[29] Guoji Guo, Mikael Huss, Guo Qing Tong, Chaoyang Wang, Li Li Sun, Neil D Clarke, and Paul Robson. Resolution of cell fate decisions revealed by single-cell gene expression analysis from zygote to blastocyst. *Developmental Cell*, 18(4):675–685, 2010.

[30] B Plusa, A Piliszek, S Frankenberg, J Artus, and A K Hadjantonakis. Distinct sequential cell behaviours direct primitive endoderm formation in the mouse blastocyst. *Journal of Embryology and Experimental Morphology*, 135(18):3081–3091, 2008.

[31] Yusuke Ohnishi, Wolfgang Huber, Akiko Tsumura, Minjung Kang, Panagiotis Xenopoulos, Kazuki Kurimoto, Andrzej K Oleś, Marcos J Araúzo-Bravo, Mitinori Saitou, Anna-Katerina Hadjantonakis, et al. Cell-to-cell expression variability followed by signal reinforcement progressively segregates early mouse lineages. *Nature Cell Biology*, 16(1):27, 2014.

[32] Kaoru Mitsui, Yoshimi Tokuzawa, Hiroaki Itoh, Kohichi Segawa, Mirei Murakami, Kazutoshi Takahashi, Masayoshi Maruyama, Mitsuyo Maeda, and Shinya Yamanaka. The homeoprotein Nanog is required for maintenance of pluripotency in mouse epiblast and ES cells. *Cell*, 113(5):631–642, 2003.

[33] Ian Chambers, Douglas Colby, Morag Robertson, Jennifer Nichols, Sonia Lee, Susan Tweedie, and Austin Smith. Functional expression cloning of nanog, a pluripotency sustaining factor in embryonic stem cells. *Cell*, 113(5):643–655, 2003.

[34] Jennifer Nichols, Branko Zevnik, Konstantinos Anastassiadis, Hitoshi Niwa, Daniela Klewe-Nebenius, Ian Chambers, Hans Schöler, and Austin Smith. Formation of pluripotent stem cells in the mammalian embryo depends on the POU transcription factor Oct4. *Cell*, 95(3):379–391, 1998.

[35] Nicolas Pilon, Diana Raiwet, Robert S Viger, and David W Silversides. Novel pre- and post-gastrulation expression of Gata4 within cells of the inner cell mass and migratory neural crest cells. *Developmental Dynamics*, 237(4):1133–1143, 2008.

[36] Claire Chazaud, Yojiro Yamanaka, Tony Pawson, and Janet Rossant. Early Lineage Segregation between Epiblast and Primitive Endoderm in Mouse Blastocysts through the Grb2-MAPK Pathway. *Developmental Cell*, 10(5):615–624, 2006.

[37] Nadine Schrode, Néstor Saiz, Stefano Di Talia, and Anna Katerina Hadjantonakis. GATA6 levels modulate primitive endoderm cell fate choice and timing in the mouse blastocyst. *Developmental Cell*, 29(4):454–467, 2014.

[38] Jerome Artus, Anna Piliszek, and Anna-Katerina Hadjantonakis. The primitive endoderm lineage of the mouse blastocyst: Sequential transcription factor activation and regulation of differentiation by SOX17. *Developmental Biology*, 350(2):393–404, 2011.

[39] Kathy K Niakan, Hongkai Ji, René Maehr, Steven A Vokes, Kit T Rodolfa, Richard I Sherwood, Mariko Yamaki, John T Dimos, Alice E Chen, Douglas A Melton, et al. Sox17 promotes differentiation in mouse embryonic stem cells by directly regulating extraembryonic gene expression and indirectly antagonizing self-renewal. *Genes & Development*, 24(3):312–326, 2010.

[40] S A Morris, R T Y Teo, H Li, P Robson, D M GLover, and M Zernicka-Goetz. Origin and formation of the first two distinct cell types of the inner cell mass in the mouse embryo. *Proceedings of the National Academy of Sciences*, 107(14):6364–6369, 2010.

[41] J B Grabarek, K Zyzynska, N Saiz, A Piliszek, S Frankenberg, J Nichols, A K Hadjantonakis, and B Plusa. Differential plasticity of epiblast and primitive endoderm precursors within the ICM of the early mouse embryo. *Development*, 139(1):129–139, 2011.

[42] J Rossant. Investigation of the determinative state of the mouse inner cell mass. *Development*, 33(4):979–990, 1975.

[43] Sebastian Wennekamp and Takashi Hiiragi. Stochastic processes in the development of pluripotency in vivo. *Biotechnology Journal*, 7(6):737–744, 2012.

[44] J E Dietrich and T Hiiragi. Stochastic patterning in the mouse pre-implantation embryo. *Development*, 134(23):4219–4231, 2007.

[45] Yojiro Yamanaka, Amy Ralston, Robert O Stephenson, and Janet Rossant. Cell and molecular regulation of the mouse blastocyst. *Developmental Dynamics*, 235(9):2301–2314, 2006.

[46] Anand Pillarisetti, Hamid Ladjal, Antoine Ferreira, Carol Keefer, and Jaydev P Desai. Mechanical characterization of mouse embryonic stem cells. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 1176–1179. IEEE, 2009.

[47] Sebastian Wennekamp, Sven Mesecke, François Nédélec, and Takashi Hiiragi. A self-organization framework for symmetry breaking in the mammalian embryo. *Nature Reviews Molecular Cell Biology*, 14(7):452–459, 2013.

[48] Robert Moore, Kathy Q Cai, Diogo O Escudero, and Xiang-Xi Xu. Cell adhesive affinity does not dictate primitive endoderm segregation and positioning during murine embryoid body formation. *Genesis*, 47(9):579–589.

[49] Ramsey A Foty, Gabor Forgacs, Cathie M Pfleger, and Malcolm S Steinberg. Liquid properties of embryonic tissues: measurement of interfacial tensions. *Physical Review Letters*, 72(14):2298, 1994.

[50] J B A Green. Sophistications of cell sorting. *Nature Cell Biology*, 10(4):375–377, 2008.

[51] Philip L Townes and Johannes Holtfreter. Directed movements and selective adhesion of embryonic amphibian cells. *Journal of Experimental Zoology*, 128(1):53–120, 1955.

[52] Malcolm S Steinberg. On the mechanism of tissue reconstruction by dissociated cells, I. Population kinetics, differential adhesiveness, and the absence of directed migration. *Proceedings of the National Academy of Sciences*, 48(9):1577–1582, 1962.

[53] Malcolm S Steinberg. Mechanism of tissue reconstruction by dissociated cells, II. Time-course of events. *Science*, 137(3532):762–763, 1962.

[54] Malcolm S Steinberg. On the mechanism of tissue reconstruction by dissociated cells, III. Free energy relations and the reorganization of fused, heteronomic tissue fragments. *Proceedings of the National Academy of Sciences*, 48(10):1769–1776, 1962.

[55] M S Steinberg. Adhesion-guided multicellular assembly: a commentary upon the postulates, real and imagined, of the differential adhesion hypothesis, with special attention to computer simulations of cell sorting. *Journal of Theoretical Biology*, 55(2):431–443, 1975.

[56] Ramsey A Foty, Cathie M Pfleger, Gabor Forgacs, and Malcolm S Steinberg. Surface tensions of embryonic tissues predict their mutual envelopment behavior. *Development*, 122(5):1611–1620, 1996.

[57] Ramsey A Foty and Malcolm S Steinberg. The differential adhesion hypothesis: a direct evaluation. *Developmental Biology*, 278(1):255–263, 2005.

[58] M S Steinberg. Adhesion in development: an historical overview. *Developmental Biology*, 180(2):377–388, 1996.

[59] C Dahmann and K Basler. Compartment boundaries: at the edge of development. *Trends in Genetics*, 15(8):320–326, 1999.

[60] Guillaume Salbreux, Guillaume Charras, and Ewa Paluch. Actin cortex mechanics and cellular morphogenesis. *Trends in Cell Biology*, 22(10):536–545, 2012.

[61] A K Harris. Is cell sorting caused by differences in the work of intercellular adhesion? A critique of the Steinberg hypothesis. *Journal of Theoretical Biology*, 61(2):267–285, 1976.

[62] G Wayne Brodland. The Differential Interfacial Tension Hypothesis (DITH): A Comprehensive Theory for the Self-Rearrangement of Embryonic Cells and Tissues. *Journal of Biomechanical Engineering*, 124(2):188–197, 2002.

[63] D Bray and JG White. Cortical flow in animal cells. *Science*, 239(4842):883–889, 1988.

[64] Jianwu Dai, H Ping Ting-Beall, Robert M Hochmuth, Michael P Sheetz, and Margaret A Titus. Myosin I contributes to the generation of resting cortical tension. *Biophysical journal*, 77(2):1168–1176, 1999.

[65] Carl-Philipp Heisenberg and Yohanns Bellaïche. Forces in Tissue Morphogenesis and Patterning. *Cell*, 153(5):948–962, 2013.

[66] S A Safran, N Gov, A Nicolas, U S Schwarz, and T Tlusty. Physics of cell elasticity, shape and adhesion. *Physica A: Statistical Mechanics and its Applications*, 352(1):171–201, 2005.

[67] Thomas Lecuit, Pierre-François Lenne, and Edwin Munro. Force Generation, Transmission, and Integration during Cell and Tissue Morphogenesis. *Annual Review of Cell and Developmental Biology*, 27(1):157–184, 2011.

[68] D Umetsu and C Dahmann. Compartment boundaries: Sorting cells with tension. *Fly*, 4(3):241–245, 2010.

[69] Andrew G. Clark, Ortrud Wartlick, Guillaume Salbreux, and Ewa K. Paluch. Stresses at the cell surface during animal cell morphogenesis. *Current Biology*, 24(10):R484 – R494, 2014.

[70] Alexander X. Cartagena-Rivera, Jeremy S. Logue, Clare M. Waterman, and Richard S. Chadwick. Actomyosin Cortical Mechanical Properties in Nonadherent Cells Determined by Atomic Force Microscopy. *Biophysical Journal*, 110(11):2528–2539, 2016.

[71] Jim H. Veldhuis, Ahmad Ehsandar, Jean-Léon Maître, Takashi Hiiragi, Simon Cox, and G. Wayne Brodland. Inferring cellular forces from image stacks. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 372(1720), 2017.

[72] Raphaël Clément, Benoît Dehapiot, Claudio Collinet, Thomas Lecuit, and Pierre-François Lenne. Viscoelastic dissipation stabilizes cell shape changes during tissue morphogenesis. *Current Biology*, 27(20):3132–3142, 2017.

[73] Alba Diz-Muñoz, Michael Krieg, Martin Bergert, Itziar Ibarlucea-Benitez, Daniel J. Muller, Ewa Paluch, and Carl-Philipp Philipp Heisenberg. Control of Directed Cell Migration In Vivo by Membrane-to-Cortex Attachment. *PLoS Biology*, 8(11):e1000544, 2010.

[74] Nils C Gauthier, Thomas A Masters, and Michael P Sheetz. Mechanical feedback between membrane tension and dynamics. *Trends in Cell Biology*, 22(10):527–535, 2012.

[75] Pedro F Machado, Julia Duque, Jocelyn Étienne, Alfonso Martinez-Arias, Guy B Blanchard, and Nicole Gorfinkiel. Emergent material properties of developing epithelial tissues. *BMC Biology*, 13(1):98, 2015.

[76] Rudolf Winklbauer. Cell adhesion strength from cortical tension - an integration of concepts. *Journal of Cell Science*, 128(20), 2015.

[77] Maryam Aliee, Jens-Christian Röper, Katharina P Landsberg, Constanze Pentzold, Thomas J Widmann, Frank Jülicher, and Christian Dahmann. Physical Mechanisms Shaping the Drosophila Dorsoventral Compartment Boundary. *Current Biology*, 22(11):967–976, 2012.

[78] Jean-Paul Vincent and David Irons. Developmental Biology: Tension at the Border. *Current Biology*, 19(22):R1028–R1030, 2009.

[79] Lucy C Butler, Guy B Blanchard, Alexandre J Kabla, Nicola J Lawrence, David P Welchman, L Mahadevan, Richard J Adams, and Benedicte Sanson. Cell shape changes indicate a role for extrinsic tensile forces in Drosophila germ-band extension. *Nature Cell Biology*, 11(7):859–864, 2009.

[80] M Krieg, Y Arboleda-Estudillo, P H Puech, J Käfer, F Graner, D J Müller, and C P Heisenberg. Tensile forces govern germ-layer organization in zebrafish. *Nature*, 10(4):429–436, 2008.

[81] P Skoglund, A Rolo, X Chen, B M Gumbiner, and R Keller. Convergence and extension at gastrulation require a myosin IIB-dependent cortical actin network. *Development*, 135(14):2435–2444, 2008.

[82] A C Martin and B Goldstein. Apical constriction: themes and variations on a cellular mechanism driving morphogenesis. *Development*, 141(10):1987–1998, 2014.

[83] Ulrich S Schwarz and Carina M Dunlop. Developmental Biology: A Growing Role for Computer Simulations. *Current Biology*, 22(11):R441–R443, 2012.

[84] Katharina P Landsberg, Reza Farhadifar, Jonas Ranft, Daiki Umetsu, Thomas J Widmann, Thomas Bittig, Amani Said, Frank Jülicher, and Christian Dahmann. Increased Cell Bond Tension Governs Cell Sorting at the Drosophila Anteroposterior Compartment Boundary. *Current Biology*, 19(22):1950–1955, 2009.

[85] Chaminda R Samarage, Melanie D White, Yanina D Álvarez, Juan Carlos Fierro-González, Yann Henon, Edwin C Jesudason, Stephanie Bissiere, Andreas Fouras, and Nicolas Plachta. Cortical Tension Allocates the First Inner Cells of the Mammalian Embryo. *Developmental Cell*, 34(4):435–447, 2015.

[86] Jean-Léon Maître, Ritsuya Niwayama, Hervé Turlier, François Nédélec, and Takashi Hiiragi. Pulsatile cell-autonomous contractility drives compaction in the mouse embryo. *Nature cell biology*, 17(7):849–855, 2015.

[87] Jean-Léon Maître, Hervé Turlier, Rukshala Illukkumbura, Björn Eismann, Ritsuya Niwayama, François Nédélec, and Takashi Hiiragi. Asymmetric division of contractile domains couples cell positioning and fate specification. *Nature*, 2016.

[88] J L Maitre, H Berthoumieux, S F G Krens, G Salbreux, F Julicher, E Paluch, and C P Heisenberg. Adhesion Functions in Cell Sorting by Mechanically Coupling the Cortices of Adhering Cells. *Science*, 338(6104):253–256, 2012.

[89] Michael J Susienka, Benjamin T Wilks, and Jeffrey R Morgan. Quantifying the kinetics and morphological changes of the fusion of spheroid building blocks. *Biofabrication*, 8(4):045003, 2016.

[90] J. Youssef, A. K. Nurse, L. B. Freund, and J. R. Morgan. Quantification of the forces driving self-assembly of three-dimensional microtissues. *Proceedings of the National Academy of Sciences*, 108(17):6993–6998, 2011.

[91] Joao Firmino, Didier Rocancourt, Mehdi Saadaoui, Chloe Moreau, and Jerome Gros. Cell Division Drives Epithelial Cell Rearrangements during Gastrulation in Chick. *Developmental Cell*, 36(3):249–261, 2016.

[92] J.-Y. Tinevez, U. Schulze, G. Salbreux, J. Roensch, J.-F. Joanny, and E. Paluch. Role of cortical tension in bleb growth. *Proceedings of the National Academy of Sciences*, 106(44):18581–18586, 2009.

[93] Ewa K Paluch. After the Greeting: Realizing the Potential of Physical Models in Cell Biology. *Trends in Cell Biology*, 25(12):711–3, 2015.

[94] Simon Tanaka. Simulation Frameworks for Morphogenetic Problems. *Computation*, 3(2):197–221, 2015.

[95] Alexander Anderson and Katarzyna Rejniak. *Single-cell-based models in biology and medicine*. Springer Science & Business Media, 2007.

[96] Alexander G. Fletcher, Fergus Cooper, and Ruth E. Baker. Mechanocellular models of epithelial morphogenesis. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 372(1720), 2017.

[97] Alexander G. Fletcher, Miriam Osterfield, Ruth E. Baker, and Stanislav Y. Shvartsman. Vertex models of epithelial morphogenesis. *Biophysical Journal*, 106(11):2291–2304, 2014.

[98] James M Osborne, Alexander G Fletcher, Joe M Pitt-Francis, Philip K Maini, and David J Gavaghan. Comparing individual-based approaches to modelling the self-organization of multicellular tissues. *PLoS computational biology*, 13(2):e1005387, 2017.

[99] L G Morelli, K Uriu, S Ares, and A C Oates. Computational Approaches to Developmental Patterning. *Science*, 336(6078):187–191, 2012.

[100] Andrew C Oates, Nicole Gorfinkiel, Marcos Gonzalez-Gaitan, and Carl-Philipp Heisenberg. Quantitative approaches in developmental biology. *Nature Reviews Genetics*, 10(8):517, 2009.

[101] AM Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952.

[102] Stuart A. Newman, H.L. Frisch, and J.K. Percus. On the stationary state analysis of reaction-diffusion mechanisms for biological pattern formation. *Journal of Theoretical Biology*, 134(2):183 – 197, 1988.

[103] T. Tallinen, J. Y. Chung, J. S. Biggins, and L. Mahadevan. Gyrification from constrained cortical expansion. *Proceedings of the National Academy of Sciences*, 111(35):12667–12672, 2014.

[104] Tuomas Tallinen and John S. Biggins. Mechanics of invagination and folding: Hybridized instabilities when one soft tissue grows on another. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 92(2):1–8, 2015.

[105] Edouard Hannezo, Jacques Prost, and Jean François Joanny. Mechanical instabilities of biological tubes. *Physical Review Letters*, 109(1):1–5, 2012.

[106] Stephanie Höhn, Aurelia R Honerkamp-Smith, Pierre A Haas, Philipp Khuc Trong, and Raymond E Goldstein. Dynamics of a volvox embryo turning itself inside out. *Physical review letters*, 114(17):178101, 2015.

[107] R E Baker, E A Gaffney, and P K Maini. Partial differential equations for self-organization in cellular and developmental biology. *Nonlinearity*, 21(11):R251–R290, 2008.

[108] Deborah C. Markham, Ruth E. Baker, and Philip K. Maini. Modelling collective cell behaviour. *Discrete and Continuous Dynamical Systems*, 34(12):5123–5133, 6 2014.

[109] A. Baskaran and M. C. Marchetti. Statistical mechanics and hydrodynamics of bacterial suspensions. *Proceedings of the National Academy of Sciences*, 106(37):15567–15572, 2009.

[110] Hermann B Frieboes. Overview: Modeling heterogeneous tumor tissue as a multiphase material. *bioRxiv*, page 031534, 2015.

[111] Volker Grimm, Eloy Revilla, Uta Berger, Florian Jeltsch, Wolf M. Mooij, Steven F. Railsback, Hans-Hermann Thulke, Jacob Weiner, Thorsten Wiegand, and Donald L. DeAngelis. Pattern-oriented modeling of agent-based complex systems: Lessons from ecology. *Science*, 310(5750):987–991, 2005.

[112] M. Wooldridge. Agent-based software engineering. *IEE Proceedings - Software*, 144:26–37(11), January 1997.

[113] Endre Somogyi, James P. Sluka, and James A. Glazier. Formalizing knowledge in multi-scale agent-based simulations. *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems - MODELS '16*, pages 115–122, 2016.

[114] W Brian Arthur. *The nature of technology: What it is and how it evolves*. Simon and Schuster, 2009.

[115] Jack D Hywood, Kerry A Landman, and Emily J Hackett-Jones. Modeling biological tissue growth: Discrete to continuum representations. *Journal of Theoretical Biology*, 259(3):541–551, 2009.

[116] Katie Bentley, Andrew Philippides, and Erzsébet Ravasz Regan. Do endothelial cells dream of eclectic shape? *Developmental Cell*, 29(2):146–158, 2014.

[117] Zi Wang, Benjamin J. Ramsey, Dali Wang, Kwai Wong, Husheng Li, Eric Wang, and Zhirong Bao. An Observation-Driven Agent-Based Modeling and Analysis Framework for C. elegans Embryogenesis. *Plos One*, 11(11):e0166551, 2016.

[118] Tilo Beyer and Michael Meyer-Hermann. Modeling emergent tissue organization involving high-speed migrating cells in a flow equilibrium. *Physical Review E*, 76(2):21929, 2007.

[119] Salem Adra, Tao Sun, Sheila MacNeil, Mike Holcombe, and Rod Smallwood. Development of a three dimensional multiscale computational model of the human epidermis. *PLoS ONE*, 5(1), 2010.

[120] Dirk Drasdo and Stefan Höhme. A single-cell-based model of tumor growth in vitro: monolayers and spheroids. *Physical Biology*, 2(3):133–147, 2005.

[121] Kerri-Ann Norton, Meghan M McCabe Pryor, and Aleksander S Popel. Multiscale modeling of cancer. *bioRxiv*, page 033977, 2015.

[122] Seunghwa Kang, Simon Kahan, Jason McDermott, Nicholas Flann, and Ilya Shmulevich. Biocellion: Accelerating computer simulation of multicellular biological system models. *Bioinformatics*, 30(21):3101–3108, 2014.

[123] Biocellion. http://biocellion.com.

[124] MathCancer. http://mathcancer.org.

[125] Ahmadreza Ghaffarizadeh, Randy Heiland, Samuel H Friedman, Shannon M Mumenthaler, and Paul Macklin. Physicell: An open source physics-based cell simulator for 3-d multicellular systems. *PLoS Computational Biology*, 14(2):e1005991, 2018.

[126] Katarzyna A Rejniak. *Systems Biology of Tumor Microenvironment*, volume 936. Springer, 2016.

[127] Joe Pitt-Francis, Pras Pathmanathan, Miguel O. Bernabeu, Rafel Bordas, Jonathan Cooper, Alexander G. Fletcher, Gary R. Mirams, Philip Murray, James M. Osborne, Alex Walter, S. Jon Chapman, Alan Garny, Ingeborg M.M. van Leeuwen, Philip K. Maini, Blanca Rodríguez, Sarah L. Waters, Jonathan P. Whiteley, Helen M. Byrne, and David J. Gavaghan. Chaste: A test-driven approach to software development for biological modelling. *Computer Physics Communications*, 180(12):2452 – 2471, 2009.

[128] Gary R. Mirams, Christopher J. Arthurs, Miguel O. Bernabeu, Rafel Bordas, Jonathan Cooper, Alberto Corrias, Yohan Davit, Sara Jane Dunn, Alexander G. Fletcher, Daniel G. Harvey, Megan E. Marsh, James M. Osborne, Pras Pathmanathan, Joe Pitt-Francis, James Southern, Nejib Zemzemi, and David J. Gavaghan. Chaste: An Open Source C++ Library for Computational Physiology and Biology. *PLoS Computational Biology*, 9(3), 2013.

[129] CHaSTE Project. http://www.cs.ox.ac.uk/chaste/index.html.

[130] Robert J Tetley, Guy B Blanchard, Alexander G Fletcher, Richard J Adams, and Bénédicte Sanson. Unipolar distributions of junctional myosin ii identify cell stripe boundaries that drive cell intercalation throughout drosophila axis extension. *eLife*, 5:e12094, 2016.

[131] Alexander G Fletcher, Philip J Murray, and Philip K Maini. Multiscale modelling of intestinal crypt organization and carcinogenesis. *Mathematical Models and Methods in Applied Sciences*, 25(13):2563–2585, 2015.

[132] J. M. Osborne, A. Walter, S. K. Kershaw, G. R. Mirams, A. G. Fletcher, P. Pathmanathan, D. Gavaghan, O. E. Jensen, P. K. Maini, and H. M. Byrne. A hybrid approach to multi-scale modelling of cancer. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 368(1930):5013–5028, 2010.

[133] Ion I. Moraru, James C. Schaff, Boris M. Slepchenko, and Leslie M. Loew. The Virtual Cell. *Annals of the New York Academy of Sciences*, 971(1):595–596, 2002.

[134] Diana C. Resasco, Fei Gao, Frank Morgan, Igor L. Novak, James C. Schaff, and Boris M. Slepchenko. Virtual cell: computational tools for modeling in cell biology. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 4(2):129–140, 2012.

[135] Michael L Blinov, James C Schaff, Dan Vasilescu, Ion I Moraru, Judy E Bloom, and Leslie M Loew. Compartmental and spatial rule-based modeling with virtual cell. *Biophysical Journal*, 113(7):1365–1372, 2017.

[136] Virtual Cell Project. http://vcell.org.

[137] Nicholas Hernjak, Boris M. Slepchenko, Kathleen Fernald, Charles C. Fink, Dale Fortin, Ion I. Moraru, James Watras, and Leslie M. Loew. Modeling and analysis of calcium signaling events leading to long-term depression in cerebellar purkinje cells. *Biophysical Journal*, 89(6):3790 – 3806, 2005.

[138] Olgierd Cecil Zienkiewicz. *The Finite Element Method*. McGraw-hill London, 1977.

[139] Wu-Ling Zhao and W. Jason Morgan. Injection of Indian crust into Tibetan lower crust: A two-dimensional finite element model study. *Tectonics*, 6(4):489–504, 1987.

[140] M. L. Hull. A Finite Element Model of the Human Knee Joint for the Study of Tibio-Femoral Contact. *Journal of Biomechanical Engineering*, 124(3):273, 2002.

[141] G. Wayne Brodland and Jim H. Veldhuis. The Mechanics of Metastasis: Insights from a Computational Model. *PLoS ONE*, 7(9), 2012.

[142] G. Wayne Brodland, Denis Viens, and Jim H. Veldhuis. A new cell-based FE model for the mechanics of embryonic epithelia. *Computer Methods in Biomechanics and Biomedical Engineering*, 10(2):121–128, 2007.

[143] G Wayne Brodland, Justina Yang, and Jen Sweny. Cellular interfacial and surface tensions determined from aggregate compression tests using a finite element model. *HFSP Journal*, 3(4):273–81, 2009.

[144] Seyed Jamaleddin Mousavi, Mohamed Hamdy Doweidar, and Manuel Doblaré. 3D computational modelling of cell migration: A mechano-chemo-thermo-electrotaxis approach. *Journal of Theoretical Biology*, 329:64–73, 2013.

[145] S. J. Mousavi, M. H. Doweidar, and M. Doblaré. Computational modelling and analysis of mechanical conditions on cell locomotion and cell-cell interaction. *Computer Methods in Biomechanics and Biomedical Engineering*, 17(6):678–693, 2014.

[146] A Munjiza, DRJ Owen, and N Bicanic. A combined finite-discrete element method in transient dynamics of fracturing solids. *Engineering Computations*, 12(2):145–174, 1995.

[147] Antonio A Munjiza. *The Combined Finite-Discrete Element Method*. John Wiley & Sons, 2004.

[148] Katarzyna A Rejniak. An immersed boundary framework for modelling the growth of individual cells: An application to the early tumour development. *Journal of Theoretical Biology*, 247(1):186–204, 2007.

[149] Charles S Peskin. The immersed boundary method. *Acta numerica*, 11:479–517, 2002.

[150] Charles S. Peskin. Flow patterns around heart valves: A numerical method. *Journal of Computational Physics*, 10(2):252–271, 1972.

[151] Alexandre M Roma, Charles S Peskin, and Marsha J Berger. An Adaptive Version of the Immersed Boundary Method. *Journal of Computational Physics*, 153(2):509–534, 1999.

[152] Katarzyna A Rejniak. An immersed boundary model of the formation and growth of solid tumors. Technical report, MBI Technical Report 19, Mathematical Biosciences Institute, The Ohio State University, 2004.

[153] K Rejniak. A computational model of the mechanics of growth of the villous trophoblast bilayer. *Bulletin of Mathematical Biology*, 66(2):199–232, 2004.

[154] Jennifer Young and Sorin Mitran. A numerical model of cellular blebbing: A volume-conserving, fluid–structure interaction model of the entire cell. *Journal of Biomechanics*, 43(2):210–220, 2010.

[155] Yu-Hau Tseng and Huaxiong Huang. An immersed boundary method for endocytosis. *Journal of Computational Physics*, 273(Supplement C):143 – 159, 2014.

[156] Robert Dillon, Lisa Fauci, Aaron Fogelson, and Donald Gaver III. Modeling Biofilm Processes Using the Immersed Boundary Method. *Journal of Computational Physics*, 129(1):57–73, 1996.

[157] John Von Neumann, Arthur W Burks, et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1966.

[158] Gérard Y. Vichniac. Simulating physics with cellular automata. *Physica D: Nonlinear Phenomena*, 10(1):96 – 116, 1984.

[159] Martin Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223(4):120–123, 1970.

[160] Christopher G. Langton. Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1-3):120–149, 1986.

[161] G B Ermentrout and L Edelstein-Keshet. Cellular automata approaches to biological modeling. *Journal of Theoretical Biology*, 1993.

[162] M Markus, D Böhm, and M Schmick. Simulation of vessel morphogenesis using cellular automata. *Mathematical Biosciences*, 156(1):191–206, 1999.

[163] N J Savill and P Hogeweg. Modelling morphogenesis: from single cells to crawling slugs. *Journal of Theoretical Biology*, 184:229–235, 1997.

[164] R J Matela, R Ransomt, and M A Bowles. Computer simulation of compartment maintenance in the *Drosophila* wing imaginal disc. *Journal of Theoretical Biology*, 103(3):357–378, 1983.

[165] Zoltán Csahók and Tamás Vicsek. Lattice-gas model for collective biological motion. *Physical Review E*, 52:5297–5303, 1995.

[166] M Alber, M Kiskowski, Y Jiang, and S Newman. Biological lattice gas models. In Gerhard Dangelmayr and Iuliana Oprea, editors, *Dynamics and Bifurcation of Patterns in Dissipative Systems*, chapter 14, pages 274–292. World Scientific, 2004.

[167] F. Y. Wu. The potts model. *Rev. Mod. Phys.*, 54:235–268, Jan 1982.

[168] M Krasnytska, P Sarkanych, and B Berche. Marginal dimensions of the Potts model with invisible states. *Journal of Physics A: Mathematical and Theoretical*, 49(25):1–15, 2016.

[169] F Graner and J A Glazier. Simulation of biological cell sorting using a two-dimensional extended Potts model. *Physical Review Letters*, 69(13):2013–2016, 1992.

[170] J A Glazier and F Graner. Simulation of the differential adhesion driven rearrangement of biological cells. *Physical Review E*, 47(3):2128, 1993.

[171] James A Glazier. *The Dynamics of 2D Cellular Patterns*. PhD thesis, University of Chicago, 1989.

[172] JA Glazier and A Upadhyaya. First steps towards a comprehensive model of tissues, or: A physicist looks at development. In *Dynamical Networks in Physics and Biology*, pages 149–160. Springer, 1998.

[173] J C M Mombach, J A Glazier, R C Raphael, and M Zajac. Quantitative comparison between differential adhesion models and cell sorting in the presence and absence of fluctuations. *Physical Review Letters*, 75(11):2244–2247, 1995.

[174] D A Beysens, G Forgacs, and J A Glazier. Cell sorting is analogous to phase ordering in fluids. *Proceedings of the National Academy of Sciences*, 97(17):9467–9471, 2000.

[175] Noriyuki Bob Ouchi, James A Glazier, Jean-Paul Rieu, Arpita Upadhyaya, and Yasuji Sawada. Improving the realism of the cellular Potts model in simulations of biological cells. *Physica A: Statistical Mechanics and its Applications*, 329(3-4):451–458, 2003.

[176] Pavel Kraikivski. *Trends in Biophysics: From Cell Dynamics Toward Multicellular Growth Phenomena*. CRC Press, 2013.

[177] Yi Jiang, Pieter J. Swart, Avadh Saxena, Marius Asipauskas, and James A. Glazier. Hysteresis and avalanches in two-dimensional foam rheology simulations. *Physical Review E*, 59:5819–5832, 1999.

[178] J A Izaguirre, R Chaturvedi, C Huang, T Cickovski, J Coffland, G Thomas, G Forgacs, M Alber, G Hentschel, S A Newman, and J A Glazier. COMPUCELL, a multi-model framework for simulation of morphogenesis. *Bioinformatics*, 20(7):1129–1137, 2004.

[179] H Honda. Description of cellular patterns by Dirichlet domains: The two-dimensional case. *Journal of Theoretical Biology*, 1978.

[180] Tatsuzo Nagai and Hisao Honda. A dynamic cell model for the formation of epithelial tissues. *Philosophical Magazine Part B*, 81(7):699–719, 2001.

[181] Christian Dahmann, Andrew C Oates, and Michael Brand. Boundary formation and maintenance in tissue development. *Nature Reviews Genetics*, 12(1):43, 2011.

[182] Hisao Honda, Tatsuzo Nagai, and Masaharu Tanemura. Two different mechanisms of planar cell intercalation leading to tissue elongation. *Developmental Dynamics*, 237(7):1826–1836, 2008.

[183] Hisao Honda, Masaharu Tanemura, and Tatsuzo Nagai. A three-dimensional vertex dynamics cell model of space-filling polyhedra simulating cell behavior in a cell aggregate. *Journal of Theoretical Biology*, 226(4):439–453, 2004.

[184] Daniel L Barton, Silke Henkes, Cornelis J Weijer, and Rastko Sknepnek. Active vertex model for cell-resolution description of epithelial tissue mechanics. *PLoS Computational Biology*, 13(6):e1005569, 2017.

[185] Dapeng Bi, Xingbo Yang, M. Cristina Marchetti, and M. Lisa Manning. Motility-driven glass and jamming transitions in biological tissues. *Physical Review X*, 6(2):1–13, 2016.

[186] Timothy J Newman. Modeling multicellular structures using the subcellular element model. In *Single-Cell-Based Models in Biology and Medicine*, pages 221–239. Springer, 2007.

[187] Sebastian A Sandersius and Timothy J Newman. Modeling cell rheology with the Subcellular Element Model. *Physical Biology*, 5(1):15002, 2008.

[188] S A Sandersius, M Chuai, C J Weijer, and T J Newman. A 'chemotactic dipole' mechanism for large-scale vortex motion during primitive streak formation in the chick embryo. *Physical Biology*, 8(4):45008, 2011.

[189] Sebastian A Sandersius, Manli Chuai, Cornelis J Weijer, and Timothy J Newman. Correlating Cell Behavior with Tissue Topology in Embryonic Epithelia. *PLoS ONE*, 6(4):e18081, 2011.

[190] S A Sandersius, C J Weijer, and T J Newman. Emergent cell and tissue dynamics from subcellular modeling of active biomechanical processes. *Physical Biology*, 8(4):45007, 2011.

[191] R.L. Gardner. The axis of polarity of the mouse blastocyst is specified before blastulation and independently of the zona pellucida. *Human Reproduction*, 22(3):798–806, 2007.

[192] Pawel Krupinski, Vijay Chickarmane, and Carsten Peterson. Simulating the Mammalian Blastocyst - Molecular and Mechanical Interactions Pattern the Embryo. *PLoS Computational Biology*, 7(5):e1001128, 2011.

[193] Pawel Krupinski, Vijay Chickarmane, and Carsten Peterson. Computational multiscale modeling of embryo development. *Current Opinion in Genetics & Development*, 22(6):613–618, 2012.

[194] Philip M. Morse. Diatomic Molecules According to the Wave Mechanics. II. Vibrational Levels. *Physical Review*, 34:57–64, Jul 1929.

[195] L. A. Girifalco and V. G. Weizer. Application of the morse potential function to cubic metals. *Physical Review*, 114:687–690, May 1959.

[196] Tamar Schlick. *Molecular Modeling and Simulation: an Interdisciplinary Guide*, volume 21. Springer Science & Business Media, 2010.

[197] Boris Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[198] Robert J Renka. Algorithm 772: Stripack: Delaunay triangulation and voronoi diagram on the surface of a sphere. *ACM Transactions on Mathematical Software (TOMS)*, 23(3):416–434, 1997.

[199] Jacob Stirling. *Methodus Differentialis*. Whiston and White, 1764.

[200] Edward J Groth and PJE Peebles. Statistical analysis of catalogs of extragalactic objects. vii-two-and three-point correlation functions for the high-resolution shane-wirtanen catalog of galaxies. *The Astrophysical Journal*, 217:385–405, 1977.

[201] Thomas Williams, Colin Kelley, and many others. Gnuplot 4.6: an interactive plotting program. http://gnuplot.sourceforge.net/, April 2013.

[202] Chris H. Rycroft, Gary S. Grest, James W. Landry, and Martin Z. Bazant. Analysis of granular flow in a pebble-bed nuclear reactor. *Physical Review E*, 74:021306, 2006.

[203] Persistence of Vision Pty. Ltd. Persistence of vision raytracer (version 3.7). http://www.povray.org/download/.

[204] Uberpov. http://megapov.inetart.net/povrayunofficial_mac/uberpov.html.

[205] Matthias Müller, Simon Schirm, Matthias Teschner, Bruno Heidelberger, and Markus Gross. Interaction of fluids with deformable solids. *Computer Animation and Virtual Worlds*, 15(3-4):159–171, 2004.

[206] Geoffrey Irving, Craig Schroeder, and Ronald Fedkiw. Volume conserving finite element simulations of deformable models. In *ACM Transactions on Graphics (TOG)*, volume 26, page 13. ACM, 2007.

[207] Hongyuan Jiang and Sean X Sun. Cellular Pressure and Volume Regulation and Implications for Cell Mechanics. *Biophysical Journal*, 105(3):609–619, 2013.

[208] Emmanuel Promayon, Pierre Baconnier, and Claude Puech. Physically-based deformations constrained in displacements and volume. In *Computer graphics forum*, volume 15, pages 155–164. Wiley Online Library, 1996.

Appendix A

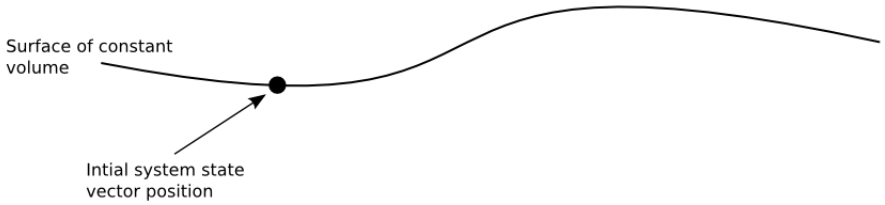
Volume Conservation

In early iterations of our model, we were concerned at the possibility that cells could become compressed by forces applied to them, rather than retaining their volume and transmitting those forces to their neighbours. This could result in dissipation of energy introduced into the system by division. Therefore, we decided to investigate how the simulations could be updated to conserve the volume of cells in the system. However, once cortical tension was introduced using a Delaunay triangulation, it became clear that compression and volume variation was not a problem. Furthermore, the volume conservation routine discussed below dramatically slowed the simulation run times. It was thus deemed unnecessary for the main results of this thesis, but its implementation is discussed here for completeness.

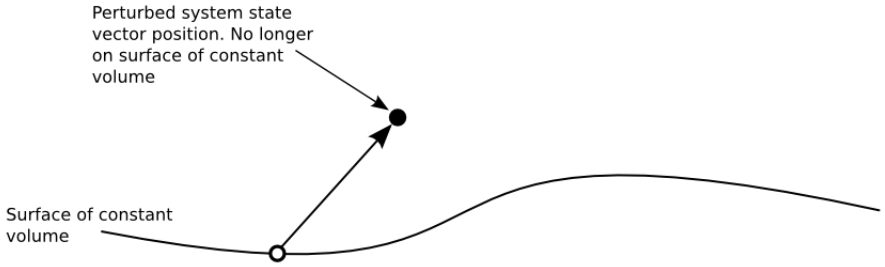
Conservation of volume in a deformable object is a difficult problem best treated with fluid dynamics or finite element modelling [205, 206]. Indeed, some work has been done in the past using fluid dynamics to model cellular systems, notably the immersed boundary method [Chapter 2]. Otherwise, mathematical models of cellular volume regulation have focussed on how cells achieve this regulation [207], which is unsuitable for our purposes.

Fortunately, one method for conserving volume in particulate systems has been identified in the literature. Promayon et al. [208] detailed their method in a 1996 paper, referencing the possibility for modelling a system such as the effect of varying lung volume on the shape of the chest cavity. The particle-based nature of this method made it suitable for application to SEM cells. What follows is a description of their method and its implementation in SEM simulations.

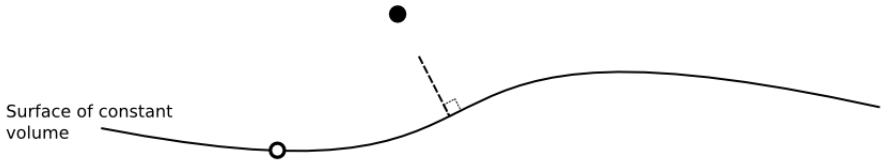
The volume of the system - in our case a cell - is defined by the position of a set of n boundary particles. Changing the position of any one of these particles can change the volume of the system. These particle positions define a state vector for the system. The state vector has $3n$ components, each corresponding to one of 3 spatial dimensions for one of n



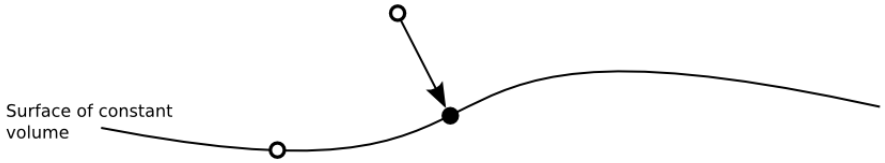
(a) State vector on the surface of constant volume.



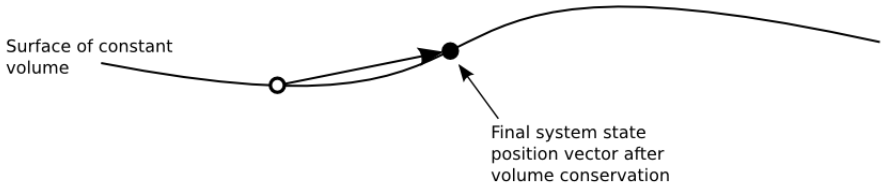
(b) System perturbed by external forces; shifts the state vector off the surface of constant volume.



(c) Find perpendicular from surface of constant volume to position of perturbed state vector.

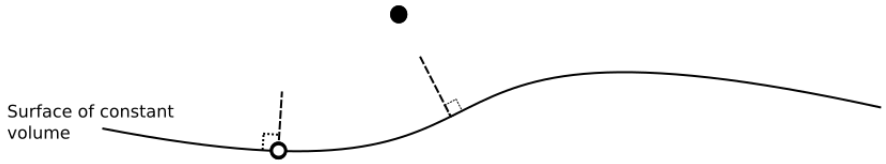


(d) Move the system state vector along the perpendicular onto the surface of constant volume.

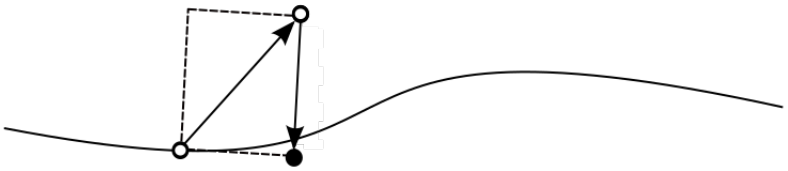


(e) Displacement of system now a combination of perturbation and adjustment to conserve volume.

Fig. A.1 Diagram demonstrating the Promayon volume conservation algorithm.



(a) Vector perpendicular to surface of constant volume at original position is approximately equal to that at the nearest point on the surface to the perturbed system position.



(b) By shifting system state by a vector found by projecting the original perturbation along the gradient at the original position, the system is moved back approximately to the surface of constant volume.

Fig. A.2 Diagram demonstrating streamlined approximation of Promayon algorithm.

particles. With the state vector thus defined, there must be a surface in the state space upon which all system states with a given volume lie [Figure A.1a].

When forces act upon the system, the resulting deformation changes the positions of the boundary particles and can shift the state vector away from the surface of constant volume [Figure A.1b]. The hypothesis of Promayon et al. is that from this point, the most appropriate approximation to introduce in order to conserve the volume of this system whilst allowing deformation is to artificially move the state vector back to the surface of constant volume with the smallest possible adjustment. The smallest possible adjustment is that which is perpendicular to the surface of constant volume in state space [Figure A.1]. Thus we can find the final state of the system, $\underline{Q}$ after an adjustment to conserve the volume by formulating a set of differential equations such that the $3n$ vector between $\underline{Q}$ and the perturbed position $\underline{P}$ is some multiple of the gradient of the surface of constant volume at $\underline{Q}$, $a\nabla\Phi\left(\underline{Q}\right)$ [Equation A.1].

$$\underline{Q} - \underline{P} = a\nabla\Phi\left(\underline{Q}\right) \tag{A.1}$$

Unfortunately this algorithm is computationally heavy, and to apply it to a large number of cells over an even larger number of time steps is not feasible, so we decided to try an approximation. If we assume that curvature of the surface of constant volume is minimal,

```
D(:)=0
do i=1, cells(c)%cortex_elements(0)
  P_1(3*(i-1)+1) = elements(cells(c)%cortex_elements(i))%position(1)
  P_1(3*(i-1)+2) = elements(cells(c)%cortex_elements(i))%position(2)
  P_1(3*(i-1)+3) = elements(cells(c)%cortex_elements(i))%position(3)
end do
do i=1, cells(c)%cortex_elements(0)
  do j=1, cells(c)%triplet_count
    do k=1, 3
      if(cells(c)%triplets(k,j).EQ.cells(c)%cortex_elements(i)) then
        if(k.EQ.1) then
          P(:) = xe_prev(cells(c)%triplets(1,j),:) - cells(c)%position(:)
          Q(:) = xe_prev(cells(c)%triplets(2,j),:) - cells(c)%position(:)
          R(:) = xe_prev(cells(c)%triplets(3,j),:) - cells(c)%position(:)
        elseif(k.EQ.2) then
          P(:) = xe_prev(cells(c)%triplets(2,j),:) - cells(c)%position(:)
          Q(:) = xe_prev(cells(c)%triplets(1,j),:) - cells(c)%position(:)
          R(:) = xe_prev(cells(c)%triplets(3,j),:) - cells(c)%position(:)
        else
          P(:) = xe_prev(cells(c)%triplets(3,j),:) - cells(c)%position(:)
          Q(:) = xe_prev(cells(c)%triplets(1,j),:) - cells(c)%position(:)
          R(:) = xe_prev(cells(c)%triplets(2,j),:) - cells(c)%position(:)
        end if
        QcrossR = CROSS_PRODUCT(Q,R)
        volume_fragment = DOT_PRODUCT(P,QcrossR)
        D(3*i-2:3*i) = D(3*i-2:3*i) + SIGN(1.0,volume_fragment)*QcrossR
        EXIT
      else
        CYCLE
      endif
    end do
  end do
enddo
do l=1, cells(c)%cortex_elements(0)
  F(l*3-2) = elements(cells(c)%cortex_elements(l))%position(1) -
   ↪  xe_prev(cells(c)%cortex_elements(l),1)
  F(l*3-1) = elements(cells(c)%cortex_elements(l))%position(2) -
   ↪  xe_prev(cells(c)%cortex_elements(l),2)
  F(l*3)         = elements(cells(c)%cortex_elements(l))%position(3) -
   ↪  xe_prev(cells(c)%cortex_elements(l),3)
enddo
```

Code Block A.1 Identifying state vector and perturbation vector from cortex element positions and calculating gradient vector.

```
D_magnitude_sq=0
do l=1, 3*cells(c)%cortex_elements(0)
  D_magnitude_sq = D_magnitude_sq + D(l)**2
enddo
D_magnitude = SQRT(D_magnitude_sq)
E_magnitude = DOT_PRODUCT(F,D)/D_magnitude
E(:) =        E_magnitude*D(:)/D_magnitude
P_2(:) = P_1(:) + E(:)
do l=1, cells(c)%cortex_elements(0)
  elements(cells(c)%cortex_elements(l))%position(1) = P_2(3*(l-1)+1)
  elements(cells(c)%cortex_elements(l))%position(2) = P_2(3*(l-1)+2)
  elements(cells(c)%cortex_elements(l))%position(3) = P_2(3*(l-1)+3)
end do
```

Code Block A.2 Projecting gradient vector along perturbation.

then to first order the gradient of the surface of constant volume at the position closest to that of the perturbed system, as found in the original method, is approximately equal to the gradient of the surface of constant volume at the original position of the system. By projecting the original perturbation vector along this direction, the system can be shifted to a position close to that found by the original algorithm, approximately conserving its volume.

This method was implemented in the SEM program with subroutine `scem_volume_conserve`. This subroutine performs a loop over all cells in the system, and for each cell begins by constructing the $3n$ state vector of the cell, `P_1`, where n is the total number of cortex elements in the cell on which the routine is currently operating. Subsequently, a loop over all cortex elements in the cell and all triplets identifies non-zero components of the gradient of the surface of constant volume, which are stored in vector `D` [Code block A.1]. The code then calculates the perturbation vector, `F` from the difference between previous and current element positions

Finally, the last part of the volume conservation routine [Code block A.2] projects `F` along the direction defined by `D`, resulting in vector `E`. This vector `E` is then added to the perturbed position `P_1`, to produce the state of the system after volume conservation, `P_2`. From this state vector, the new positions of all cortex elements are extracted.

This routine ran quickly enough to be feasible, but still significantly slowed all simulations. It seemed to perform volume conservation reasonably well, but it was quickly eventually decided that this was unnecessary for the purposes of our model. However, the routine may have some use in other systems, such as in modelling sedimentation in colloidal suspensions.

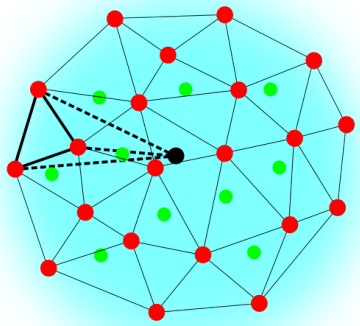


Fig. A.3 Diagram of one tetrahedron used to calculate the volume of a cell

```
volume_cell = 0
do i=1, cells(j)%triplet_count
  volume_triplet=0
  P = elements(cells(j)%triplets(1,i))%position - cells(j)%position
  Q = elements(cells(j)%triplets(2,i))%position - cells(j)%position
  R = elements(cells(j)%triplets(3,i))%position - cells(j)%position
  volume_triplet = DOT_PRODUCT(P,CROSS_PRODUCT(Q,R))/6.0
  volume_cell = volume_cell + ABS(volume_triplet)
end do
cells(j)%volume = volume_cell
```

Code Block A.3 Cell volume calculation from `scem_volume_calculate`

In addition to the routine for conserving volume, we also created a routine for simply calculating the volume of each cell, `scem_volume_calculate` [Code block A.3]. This routine uses all Delaunay triangles defined over the surface of each cell, for each triangle creating a tetrahedron with the centre of mass of the cell [Figure A.3]. Taking 3 edge vectors of this tetrahedron, P, Q, and R, the volume of the tetrahedron is then given by $\frac{1}{2}P \cdot (Q \times R)$. Summing the volumes of each tetrahedron in the cell gives the volume of the cell, which can then be stored in the `volume` component of the `cells` data structure.