UNIVERSITY OF CALGARY

Modeling Dense Inflorescences

by

Andrew Robert Owens

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

December, 2016

Abstract

Showy inflorescences - clusters of flowers - are a common feature of many plants, greatly contributing to their beauty. The large numbers of individual flowers (florets), systematically arranged in space, make inflorescences a natural target for procedural modeling. This thesis presents a suite of biologically motivated algorithms for modeling and animating the development of inflorescences, each sharing the following characteristics: (i) the ensemble of florets create a relatively smooth, tightly packed, often approximately planar surface; (ii) there are numerous collisions between petals of florets; and (iii) the developmental stages and types of florets each depends upon their positions within the inflorescence. A single framework drives the floral canopy's development and resolves the collisions. Flat-topped branched inflorescences (corymbs and umbels) are modeled using a florets-first algorithm, wherein the branching structure self-organizes to support florets in predetermined positions. This suite of techniques is illustrated with models from several plant families.

Preface

The majority of my Masters thesis comes from the paper: Andrew Owens, Mikolaj Cieslak, Jeremy Hart, Regine Classen-Bockhoff and Przemyslaw Prusinkiewicz, *Modeling Dense Inflorescences*[1], ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2016, volume 35, Issue 4 [1]. The publisher and the authors of this publication have given their permission for its inclusion in my thesis. The permissions to use this material from all co-authors and the copyright holder, Association for Computing Machinery (ACM), are gratefully acknowledged.

Chapter 1 describes the biological context and motivation of the method, as well as an overview of the method. My supervisor, Dr. Przemyslaw Prusinkiewicz identified the biological foundations of inflorescence development. On this basis, Dr. Prusinkiewicz and I equally shared the responsibility for conceiving, designing and outlining the method and illustrative figures.

Chapter 2 describes the Floret Editor. Jeremy Hart implemented the Floret Editor, with an input on design from Przemyslaw Prusinkiewicz, Mikolaj Cieslak and myself. In particular, I was responsible for designing and implementing the mesh triangulation technique within the Floret Editor, which provided a consistent and user configurable triangle mesh topology throughout the floret animations. As well, I was responsible for designing the format of the animation outputs, and for incorporating the Floret Editor into the method and my thesis.

In Chapter 3, I describe the technique of Position Based Dynamics [2], and its extension, which I introduced to model growth of flexible and colliding surfaces. I was responsible for designing this extension, as well as implementing and incorporating constraint-based growth into the overall pipeline.

Chapter 4 describes the techniques used for collision handling. I was responsible for implementing each technique, and designing a cohesive collision handling system from these techniques. As well, I was responsible for incorporating this collision handling system into the other components of the method and my thesis. On this basis, I describe the collision handling system, with

[1] doi:10.1145/2897824.2925982

some part imported from our SIGGRAPH paper.

Chapter 5 describes the phyllotaxic patterning generation algorithm. My supervisor designed and implemented this model in consultation with Dr. Regine Classen-Bockhoff, which I used as the basis for designing specific inflorescence models. I was responsible for designing and implementing the format of the animation output to be incorporated into the model. Przemyslaw Prusinkiewicz, Mikolaj Cieslak and I share equal responsibility in describing the model and producing the illustrative figures.

Chapter 6 describes the algorithm for floret type determination, which my supervisor designed and implemented in consultation with Dr. Regine Classen-Bockhoff. I designed the specific inflorescence models, as produced in Chapter 5, to use this algorithm. Przemyslaw Prusinkiewicz, Mikolaj Cieslak and I share equal responsibility in describing the model and producing the illustrative figures.

Chapter 7 describes the technique for producing branching structures. My supervisor motivated the technique with the biological foundation and inspired its initial design, while I was responsible for the final design, implementation, and incorporation of the output into the technique and my thesis. Przemyslaw Prusinkiewicz, Mikolaj Cieslak and I share equal responsibility in describing the model and producing the illustrative figures.

Chapter 8 details more of the implementation considerations of the framework of the method. I was responsible for designing and implementing the framework, with my supervisor consulting in certain design choices.

Chapter 9 discusses and illustrates the final results of the method. I designed, simulated, modeled and rendered the final results of this paper.

Chapter 10 summarizes the thesis and enumerates future work based on the thesis. I am responsible for writing this part of the thesis, with some parts inspired from our SIGGRAPH paper.

In the appendices, I present a range of supplementary ideas and technical notions from the thesis.

Acknowledgements

I have honourably held the title of 'professional student' among my friends, family and loved ones for many years now, yet my conviction in endeavouring to obtain a higher and broader education has rarely waned. I believe much of this confidence comes from within myself, but only as a result of many, *many* individuals who have offered support and guidance. Honestly, there are too many individuals to mention here, and too little space to give each the appreciation deserving of them, but I would like to mention just a few now.

First and foremost, I would like to thank my supervisor Dr. Przemyslaw Prusinkiewcz. When I first joined the Biological Modeling and Visualization research group as an undergraduate summer student, under his supervision, I could have only imagined the opportunities that awaited in the years to come. Not just professional, but personal opportunities for growth and inspiration. Dr. Prusinkiewcz's boundless enthusiasm for learning, and tenacity in its application have always been a source of inspiration to me. Ever will I be grateful for his support, tolerance, expertise and guidance.

Thanks to Dr. Usman Alim and Dr. Sonny Chan for servicing on my examination committee and providing numerous insightful comments and discussion on this thesis. Thanks to Dr. Faramarz Samavati for servicing as my Neutral Chair for my thesis defence, when no others were available.

Much thanks goes to my fellow co-authors of the paper "Modeling Dense Inflorescences", for allowing me to use portions of the paper in this thesis, and for working incredibly hard to see this work finally be published in ACM Transactions on Graphics. The experience of travelling with my co-authors Mikolaj Cieslak and Jeremy Hart to Anaheim, CA so that I may present this work at SIGGRAPH 2016 is perhaps the highlight of my few years as a Masters student. Many thanks to both of you for sharing and enriching this formative experience.

As well, I would like to thank my lab mates from the Biological Modeling and Visualization research group and from the neighbouring research groups for the stimulating conversations that

were a welcomed daily occurrence, and for imparting their keen insight into research problems, academia and all matters of interest. Specifically, I would like to thank Allana Rocha and Adam Runions for staying well after the work hours to madly muse about mathematics and computer graphics, those sessions have been of supreme importance to me for inspiration.

Finally, I would like to thank my family, friends and loved ones for their constant and continued support during this prolonged academic experience. I would like to acknowledge my parents for their unyielding and loving support both in my academic career, but also my personal endeavours As well, I would like to acknowledge my exceptional partner Maria who helped me through the toughest times of writing the paper, and kept me sane long enough to complete this thesis.

Dedicated to Tom Drynan and Tracie Owens, you know what you did...

Table of Contents

List of Tables

List of Figures and Illustrations

Chapter 1

Introduction

In many plant species, flowers are grouped into multi-flower assemblies called inflorescences. Such floral arrangements have selective value: by being more visible, inflorescences can attract pollinators from a larger distance than individual flowers; furthermore, by supporting walking between adjacent florets, inflorescences may facilitate their pollination by insects. From an aesthetic perspective, showy inflorescences are a visually appealing attribute of many plants occurring in natural and artificial settings. The common occurrence and beauty of inflorescences has made them an attractive modeling subject in computer graphics [3; 4; 5; 6; 7; 8; 9; 10]. In this paper we extend the class of inflorescences that can be modeled and animated for image synthesis purposes.

1.1 Biological context

An insight into the form of plants is offered by the first available space theory of plant organ initiation, formulated in the XIX century by Wilhelm Hofmeister (see [11] for a recent description). According to this theory, incipient organs, such as leaf or floret primordia, are positioned on the growing surface of the (shoot or reproductive) meristem when and where there is enough room for them. Depending on the parameters of this process, different regular arrangements (phyllotactic patterns) of primordia emerge [12; 13]. In some cases, reproductive meristems develop into next-order meristems rather than florets, leading to hierarchically or recursively compounded inflorescences [14]. The type, form and developmental stage of florets may depend on their position within an inflorescence (floral dimorphism [15]). In general, florets close to the inflorescence margin develop enlarged petals, making the inflorescence more visible to pollinators, while florets in more central positions have comparatively reduced petals allowing for denser packing and thus a larger number of reproductive organs within the inflorescence [16] (English version [17]). Striking

Figure 1.1: A photograph of an *Orlaya grandiflora* inflorescence. Photograph by Holger Casselmann licensed under CC BY_SA 3.0.

examples of floral dimorphism abound in the Aster family. For example, in the sunflower, petals of the showy florets on the margin (ray florets) are orders of magnitude larger than those of florets in the interior of the inflorescence (disk florets). The differences between florets may also be gradual, as illustrated by the photograph of *Orlaya grandiflora* in Figure 1.1, and may occur at different organization levels in compound inflorescences. The developmental mechanisms defining the spatial distribution of florets of different type and size are not yet well understood, but observations of compound heads [16] and wounding experiments on sunflower heads [18] suggest that the inhibition of showy florets by the proximity of other florets is the determining factor at least in some cases. In the context of a regular arrangement of florets into spiral phyllotactic patterns, such inhibition leads to the prevalence of specific numbers of enlarged florets on the inflorescence

Figure 1.2: A photograph of a *Gaillardia* inflorescence, exhibiting the fleshy receptacle that supports the inflorescence's florets. Photograph courtesy of Przemyslaw Prusinkiewicz.

margin (numerical canalization [19]). For example, in the most common spiral phyllotactic pattern with the golden divergence angle (approx. 137.5°), the prevalent numbers of ray florets belong to the Fibonacci sequence. The availability of space when new primordia are initiated does not guarantee that the florets will not collide during subsequent development. In dense inflorescences such collisions are frequent.

Florets in an inflorescence are supported either by a fleshy voluminous body (the receptacle) or a free-standing branching structure [15]. The former case is exemplified by flower heads (capitula, see Figure 1.2), and is relatively simple from a modeling perspective, as the receptacle can be approximated by a surface of revolution (e.g. [5]). In the latter case, the ar-

Figure 1.3: A photograph of a yarrow inflorescence, exhibiting the nearly planar arrangement of florets, typical of corymb type inflorescences. Photograph courtesy of Przemyslaw Prusinkiewicz.

rangement of florets in space has traditionally been viewed as a consequence of the underlying branching pattern. This branching-first perspective is useful in modeling practice (e.g. [4; 9; 20]; Figure 4.10), but does not provide an adequate model for inflorescences in which the ensemble of florets forms a smooth canopy. A quintessential example is that of corymbs, an inflorescence type in which florets are arranged into an almost planar surface (Figure 1.3). How plants create branching structures satisfying this planarity constraint is an open question, as the problem is nontrivial from a trigonometric perspective. We propose a solution inspired by the current biological understanding of inflorescence development [21; 22]. According to it, a growing meristem (Figure 1.4a) supports emergent primordia (b), which initiate vascular strands (c). These strands gradually extend toward the base of the meristem and merge, forming a branching structure (d). The vascular strand formation is driven by the flow of the plant hormone auxin (auxin canal-

ization[1] [23]) and is analogous to the formation of a branching river network, where tributaries start at different sources and gradually join each other. In heads, the vascular structure remains embedded in the surrounding tissue, which becomes the receptacle as the inflorescence develops to maturity. In branched inflorescences, the vascular strands and the immediately adjacent tissues elongate while tissues further removed from the vasculature do not grow or grow more slowly. A free-standing branching structure thus emerges (e,f). We note that, if the growth of this structure is (approximately) isometric or allometric – in general, if the length of mature segments is proportional to their length in the meristem, and the branching angles do not change or change in concert – the arrangement of florets in the mature inflorescence will reflect their original distribution on the surface of the meristem. For instance, in corymbs, the floret distribution formed on a flat meristem surface will result in a planar canopy. In summary, we assume that the distribution of florets in a predetermined phyllotactic pattern drives the formation of the supporting branching structure, and not vice versa. This florets-first perspective provides a possible explanation for the development of smooth floral canopies. It is also justified from an ecological/evolutionary point of view, as it reflects the importance of flower distribution, rather than the branching structure, to the pollinators. The branching structure is merely a scaffold that supports the flowers in their target positions [24].

Following this description, we propose a method for modeling and animating the development of inflorescences that integrates the following elements: (i) the modeling of individual florets and the animation of their opening (anthesis); (ii) the generation of dynamic phyllotactic patterns that defines the distribution of florets within floral canopy over time; (iii) determination of the type, size and developmental stage of each floret according to its position in the inflorescence and time, (iv) detection and resolution of collisions between petals, and (v) simulation of the development of the branching inflorescence structure that supports the florets in space. These elements can be used jointly or selectively, depending on the inflorescence type.

[1]Confusingly, the terms *numerical canalization* and *auxin canalization* function concurrently, although they are not closely related.

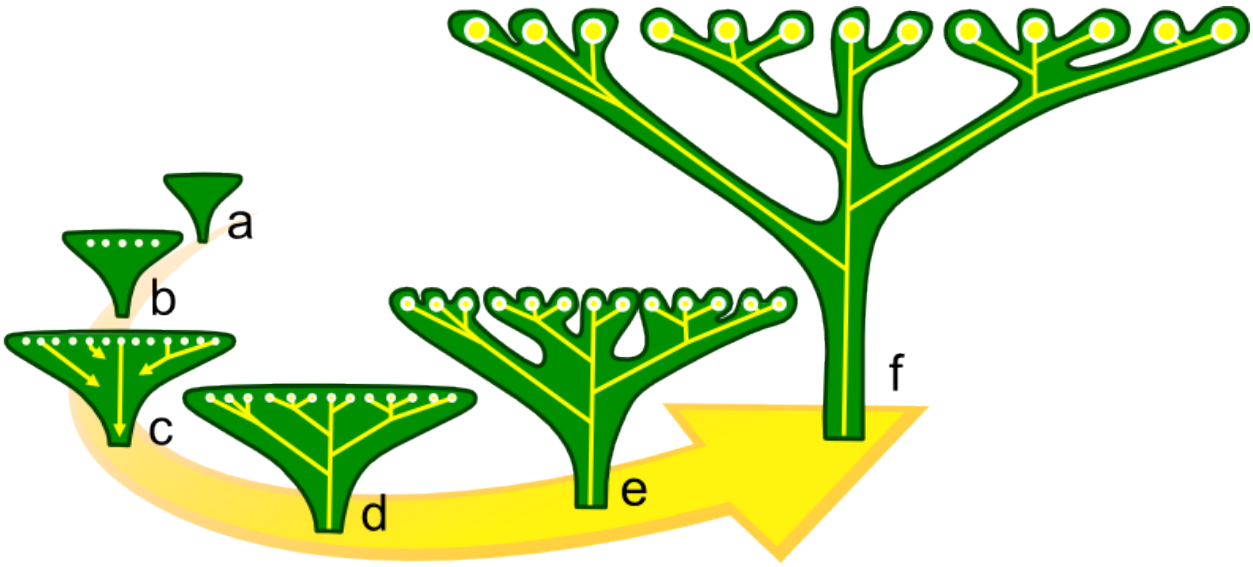

Figure 1.4: Postulated mechanism of branching inflorescence development. Dots represent florets, yellow lines indicate the vasculature. Vascular strands develop within a growing meristem (a-d) and define the axes of the emerging branching structure (e,f).

1.2 Overview of the method

The elements of our method can be grouped into three processes: the modeling of individual florets, the generation of their spatial distribution and attributes such as the type and developmental stage, and — when present — the generation of the branching structure that supports the florets. Components of the system and the information flow between them are shown in Figure 1.5, further description of the file communication between the components is found in Chapter 8.

Florets are modeled as B-spline surfaces (Chapter 2). The poses representing key developmental stages for each floret type are specified interactively. This is effected using a specialized graphical editor (Figure 1.5a), which supports radial and bilateral symmetry as well as partially fused petals found in many florets. The key poses are interpolated to generate sequences representing the development and opening of florets (Figure 1.5b). The models are then carefully polygonized so that the polygon count in each floret is small and degenerate triangles are avoided (Figure 1.5c). Low polygon count in individual florets is important to the efficient detection and resolution of collisions in inflorescences with multiple florets. The resulting floret models are instantiated when

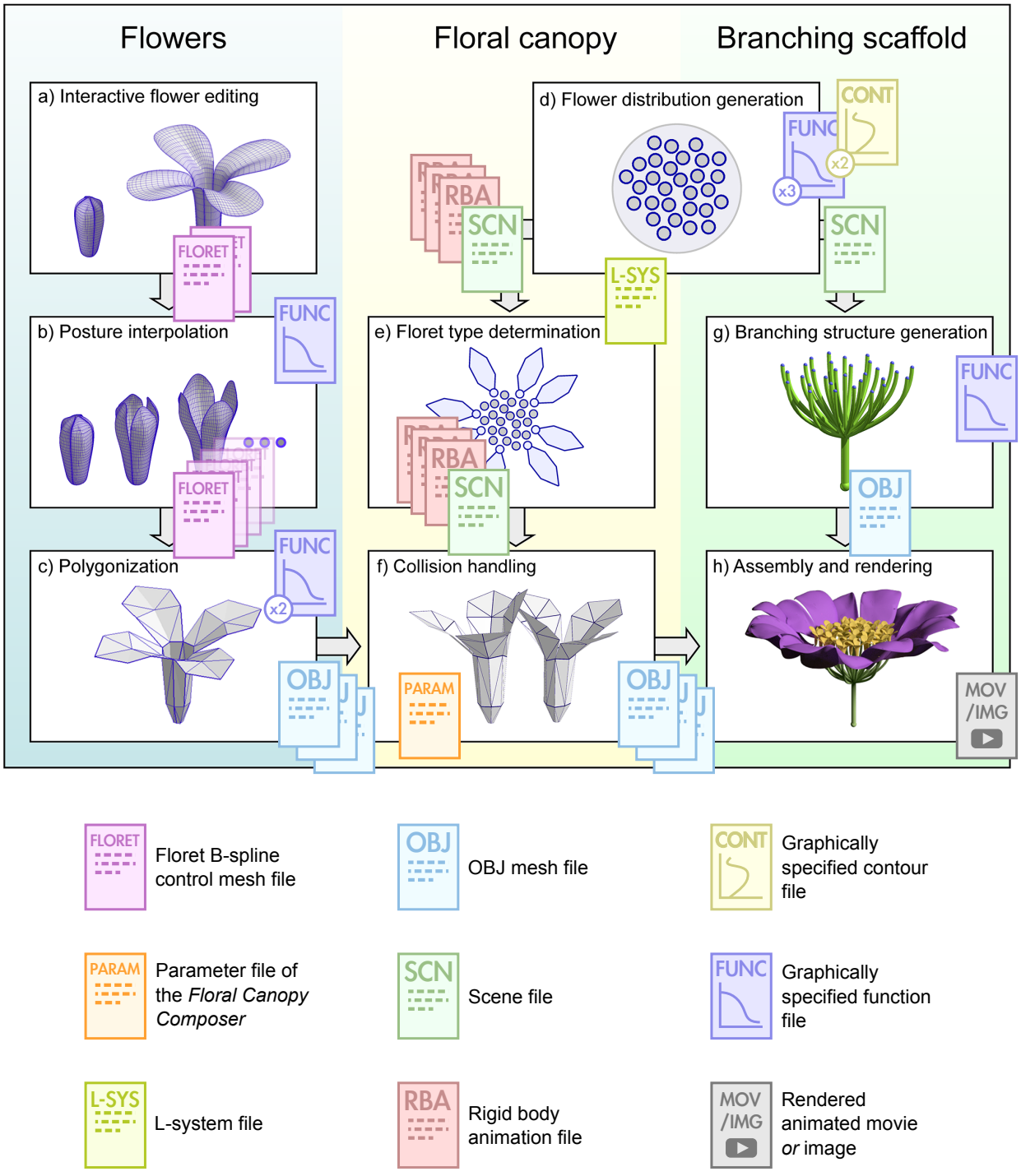

Figure 1.5: Key components and file usage of the modeling method, and the associated information flow via transferred files.

placed within an inflorescence and may be deformed by collisions between petals.

Collisions may arise in each step of simulated inflorescence development. We resolve them by applying the kinematic description of floret development and opening (the interpolation of key poses) to drive a physically-based model of petal expansion and collision (Figure 1.5f and Chapter 4). A related approach, with a kinematic growth model driving a physically-based model, was proposed to animate flower opening by Ijiri et al. [25]. Given that a floral canopy may include many flowers, we use position-based dynamics [2] as a faster alternative to the energy minimization method used by Ijiri et al.

To generate the layout of florets in dense canopies, we extend the algorithm for generating phyllotactic patterns on arbitrary surfaces of revolution proposed by Ridley [26] and introduced to computer graphics by Prusinkiewicz et al. [8] (Figure 1.5d and Chapter 5). Our extensions capture the dynamics of pattern development and provide information on the developmental stage of each floret within the inflorescence; furthermore, they enable simulation of hierarchically and recursively compound phyllotactic patterns. If the inflorescence is dimorphic, the neighborhood of each floret is inspected and the floret type is determined on this basis (Figure 1.5e and Chapter 6).

If florets are supported by a receptacle, distributing florets of the appropriate type according to the phyllotactic pattern and resolving collisions terminates the modeling process. The receptacle, usually obscured by the florets, does not need to be represented explicitly. In contrast, in branched inflorescences, the branching structure supporting the floral canopy must be visualized. We generate it using the spatial layout and age of the florets as input (Figure 1.5g and Chapter 7.1). Our algorithm is inspired by that introduced by Rodkaew et al. [27] to model leaf vein patterns and trees – both algorithms create branching structures from the outside in – extending it in several directions as needed to model inflorescences. The floral canopy and the supporting branching structure are rendered together to produce the final image or movie animation (Figure 1.5h).

Chapter 2

Interactive modeling of flowers

2.1 Previous work

There is a long history of modeling flower petals as interactively defined bicubic surfaces (e.g. [4; 5]) or generalized cylinders [7; 8], and assembling them into flowers using procedural models of phyllotaxis. Ijiri et al. [9] advanced the concept of interactive flower modeling by introducing a specialized editor to distribute flower parts in space. They also pioneered sketch-based modeling of plant organs such as leaves and petals [9; 10]; further work in this direction was pursued by Anastacio et al. [28].

At any point in time, an inflorescence may incorporate a progression of flowers at different developmental stages, a phenomenon termed the phase effect by d'Arcy Thompson [29] (see also [3; 4]). Simulation of flower growth and opening is thus needed not only to animate flower or inflorescence development, but also to construct static models of inflorescences with the phase effect. To simulate the opening of flowers modeled as bicubic surfaces, Prusinkiewicz et al. [6] constructed a branching structure that supported the set of control points defining the surface and gradually modified this structure over time. This method amounted to a forward simulation of growth, making the final form of fully open flowers difficult to control. With petals represented as generalized cylinders, Prusinkiewicz et al. [8] modeled flower growth by interpolating intrinsically-defined carrier curves that represented petal axes (midribs) in closed and open flowers. As well, the plant modeling program LPFG [30] animates the opening of flowers by interpolating the positions of individual control points in Bezier and B-spline surfaces. Our method is related to this concept. A different, physically-based approach was proposed by Ijiri et al. [25] and improved by Li et al. [31]. They represented petals as elastic surfaces subject to non-uniform expansion. This expansion affected both the size and shape of the petals. Related methods have also been used in

biologically-motivated simulations [32; 33], in the latter case addressing the particularly complex shape of snapdragon flowers. Physically-based techniques are appealing because of their sound biological basis and possible emergence of secondary features, such as wrinkles on the margin of the petals, which are otherwise difficult to model [31]. On the negative side, both the dynamics of flower opening and the final shape are difficult to specify.

The complexity of inflorescences results from the arrangement of florets, rather than the individual floret forms. Consequently, we devised a specialized interactive editor to quickly model simple flowers in their key poses (stages of development) and a blending technique to interpolate between these poses.

2.2 Floret Editor

We focus on the modeling of corolla (the set of petals), which is the most visible part of florets. A snapshot of the editor is shown in Figure 2.1. Petals are represented as clamped B-spline surfaces. To facilitate the modeling process, bilateral symmetry can be imposed on the individual petals, and petals can be multiplied by assuming n-fold radial symmetry. One new element is the modeling of partially fused petals, which are found in many florets. We implemented it for florets with dihedral (i.e., both rotational and bilateral) symmetry, by constraining a user-defined number of control points defining the petal boundary (starting at the flower base) to the symmetry plane between the adjacent petals.

2.3 Simulation of floret development.

We assume that the key floret poses are represented by B-spline surfaces defined by control meshes with the same topology (i.e. the same number of control points in the u and v directions). Such surfaces can easily be blended by linearly interpolating positions of the corresponding control points, but linear interpolation is not the best choice for simulating the opening of flowers because petals may undergo large rotations. Addressing this problem, we interpolate floret poses by extending the

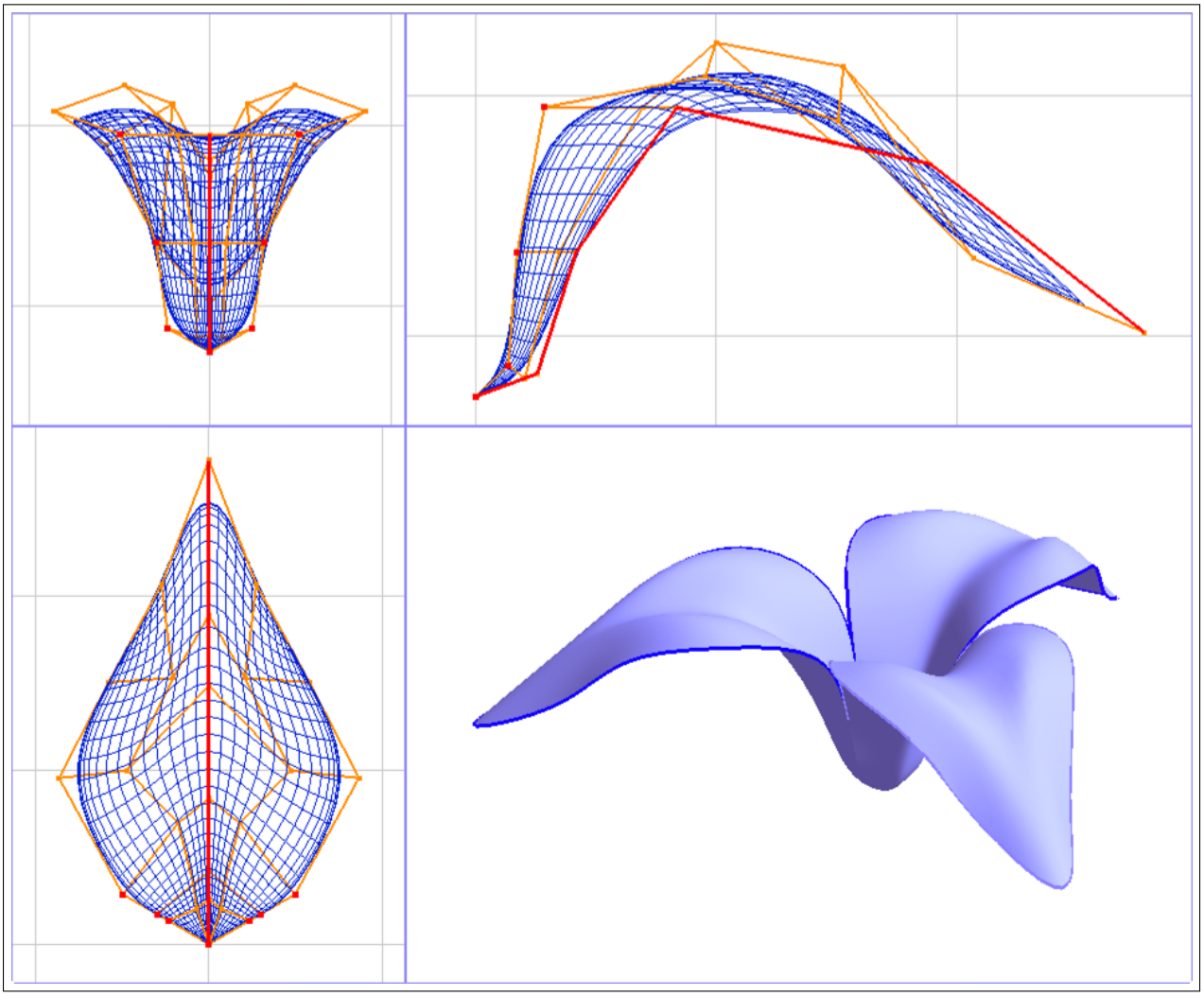

Figure 2.1: Snapshot of the Floret Editor. The user defines the shape of the petal by moving control points in the standard front, top and side view. The bottom right view shows the corolla with the petals replicated by rotational symmetry and partially fused.

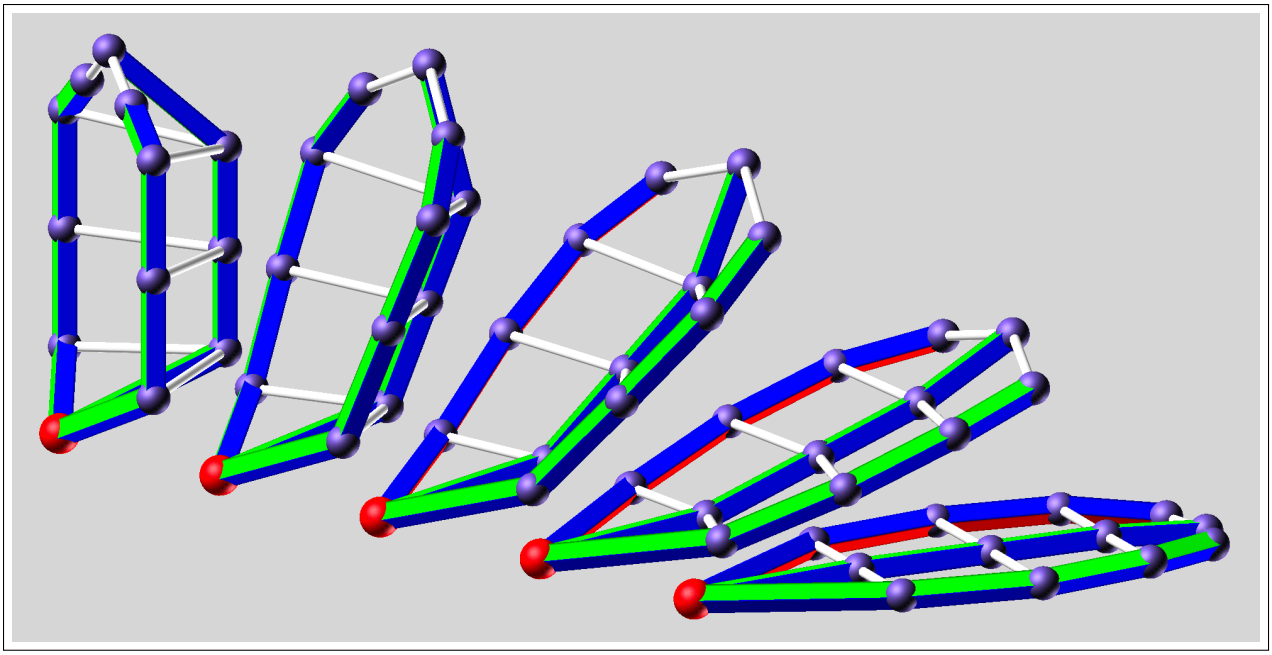

Figure 2.2: Example of control mesh interpolation. In each pose, control polylines in the v direction are represented by boxes, the colored faces of which are aligned with a rotation-minimizing moving frame. White lines indicate control polylines in the u direction, spanning the vertices of the v polylines.

method of blending intrinsically defined polygons [34] to 3D. To this end, we calculate a rotation-minimizing frame [35] for each (open) control polyline running in the v direction, from the petal base to its tip (Figure 2.2). At each vertex between consecutive line segments, this frame is rotated around the axis perpendicular to both segments so that the next segment lines up with the previous one (for collinear segments this rotation is 0). We then blend corresponding control polygons in the initial and final poses by linearly interpolating the lengths of the corresponding segments, and spherically interpolating the rotations between them. This technique makes it possible to simulate the development and opening of flowers using a minimal number of key poses, usually only two or three (Figure 2.3).

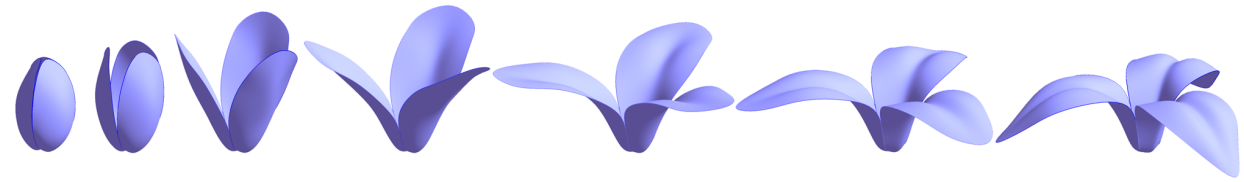

Figure 2.3: Simulation of floret opening by intrinsic interpolation between the first and the last pose.

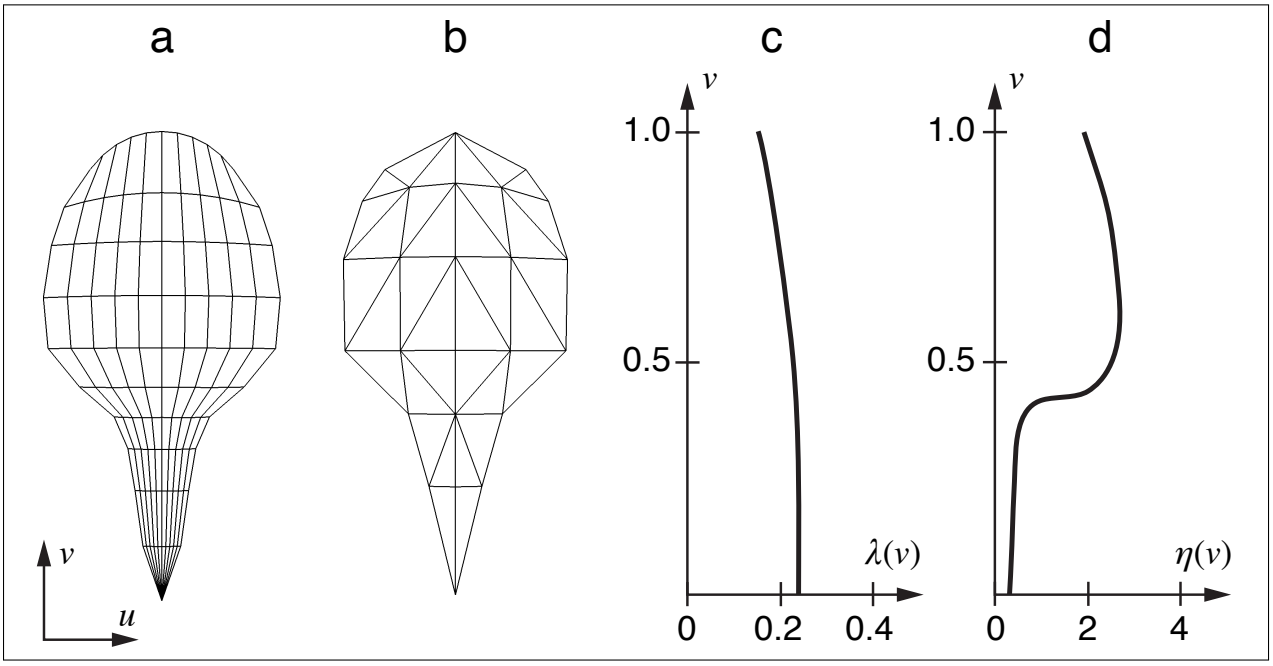

Figure 2.4: Example of petal polygonization. (a) Default polygonization using a family of isoparametric lines $u = const$ and $v = const$, equidistant in the parameter space. (b) A sample result of repolygonization. (c,d) Graphically-defined functions used to create the repolygonization in panel b. Note that the axis representing the independent variable v runs vertically.

2.4 Polygonization

An inflorescence may comprise many florets. To make the subsequent collision detection and resolution efficient it is thus critical to polygonize them well, into a small number of non-degenerate triangles. The simplest polygonization of tensor product surfaces, producing an array of quadrangles by stepping through the u and v parameters, is inadequate for this purpose, because the shape of the resulting polygons varies greatly between narrow and wide petal regions (Figure 2.4a). Therefore, this simple stepping approach in the parameter space produces petal regions that are both overly

polygonized (i.e. the base of the petal does not require the same number of quadrangles as the rim). Consequently, we repolygonize the surfaces using a semi-interactive method, in which the modeler can fine-tune the polygonization with two graphically-defined functions (Figure 2.4c,d). The first function, $\lambda(v)$, defines the spacing between a sequence of vertices $v_0, v_1, \ldots, v_n$ along the v axis. The distances between these vertices are calculated by solving the equation

$$\int_{v_i}^{v_{i+1}} \frac{dv}{\lambda(v)} = 1, \quad i = 0, 1, \ldots, n-1, \tag{2.1}$$

which distributes points along the axis according to average values of function $\lambda(v)$ in each interval $[v_i, v_{i+1}]$ and guarantees that this distribution is robust (not sensitive to small perturbations of λ) [8]. The second function, $\eta(v)$, defines the number $\lceil \eta(v) \rceil$ of vertices, equidistant in the parameter space, along each isoparametric line $v = v_i$. This number is assumed to be odd to guarantee that no triangle straddles the petal symmetry line. This assumption improves the appearance of petals approximated using small numbers of triangles. The final polygon mesh is obtained as the Delaunay triangulation of the resulting set of points (u, v) (Figure 2.4b). This triangulation is performed in parameter space, so that the mesh topology does not change as the floret develops. The invariance of topology over the floret development is imperative to mapping our constraint based growth to the sequence of meshes in the development.

Chapter 3

Constraint Based Growth

Once the floret forms and developmental animations are designed (see Chapter 2), we obtain a sequence of polygon meshes encoding the development. These mesh animations are defined by changes of the mesh vertices in global coordinate space $\mathbb{R}^3$. However, this description alone is insufficient for animating deformable florets, exhibiting elastic *soft body* behaviour as observed in nature. This is because vertex positions are defined in a global coordinate space, while spatial interactions are, in general, local; and as we wish to model florets as *soft bodies*, these local interactions should be capable of reproducing local deformations in the floret, such as bending and compression. This requires us to re-encode the shape of the florets from a global definition to many local definitions of floret regions; meaning each vertex position is encoded relative to its local neighbourhood and not relative to every other vertex position. This redefinition of the floret forms to local constraints on vertex positions allows for local deformations to occur within the regions where vertices interact with the environment and the floret itself.

3.1 Position-based dynamics

We obtain the desired soft body animations by adapting position-based dynamics (PBD) [2; 36] to the simulation of growth. In doing so, PBD affords us a means for resolving collisions (see Chapter 4) in a unified framework (incidentally, we note that the related problem of resolving collisions between leaves in a set of plants was used to illustrate the power of PBD in the original paper [2]). As well, PBD has many avenues of potential extensions (i.e. fluid simulation, rigid body dynamics, parallelization of system solvers) that have been the focus of recent research [37]. To elucidate the discussion of our approach, we briefly review the concept of PBD.

Consistent with the PBD method, we map mesh M_0 representing the initial (collision-free)

floral canopy into a network (system) of constraints. Constraints are kinematic restrictions on relative motions of mesh elements. They are in the form of equations and inequalities, called *bilateral* and *unilateral* constraints, respectively. Position based dynamics, as the name suggests, enforces constraints on the positions of elements within a scene, and their relative motions. Thus constraints are functions on positions, with bilateral constraints are defined as the equality

$$C\left(\boldsymbol{p}_{i_1}, \boldsymbol{p}_{i_2}, \ldots, \boldsymbol{p}_{i_n}\right) = 0 \tag{3.1}$$

with the positions $\boldsymbol{p}_i$ satisfying the constraint when evaluated to 0, and unilateral constraints are defined as an inequality

$$C\left(\boldsymbol{p}_{i_1}, \boldsymbol{p}_{i_2}, \ldots, \boldsymbol{p}_{i_n}\right) \geq 0 \tag{3.2}$$

with the positions where $\boldsymbol{p}_i$ satisfying the constraint when evaluated to *or* greater than 0. Here $\{i_1, i_2, \ldots, i_n\}$ is the set of indices for which positions $\boldsymbol{p}_i$ are to be constrained by the function. Such constraints that are solely functions of positions are called *holonomic* [38].

The mesh vertices define the positions $\boldsymbol{p}_i$. The objective of PBD is to change positions $\boldsymbol{p}_i$ such that all the constraints are satisfied and, most importantly, that linear and angular momentum are conserved as vertices are repositioned. Let $\Delta\boldsymbol{p}_i$ be the displacement of vertex i. Linear momentum is conserved if

$$\sum_i m_i \Delta \boldsymbol{p}_i = \mathbf{0} \tag{3.3}$$

and angular momentum is conserved if

$$\sum_i \boldsymbol{r}_i \times m_i \Delta \boldsymbol{p}_i = \mathbf{0} \tag{3.4}$$

where m_i is a mass value associated with vertex i, and $\boldsymbol{r}_i$ is the vector between $\boldsymbol{p}_i$ and some arbitrary rotation center common to all vertices. If the repositioned vertices produced by PBD violate these conservations of momenta, so called *ghost forces* will be evident in the mesh unnaturally drifting or rotating as if acted upon by some external force. For a given constraint C, with input vertex positions $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, …, $\boldsymbol{p}_n$, let $\boldsymbol{p}$ be the vector concatenation $\left[\boldsymbol{p}_1^T, \boldsymbol{p}_2^T, \ldots, \boldsymbol{p}_n^T\right]^T$. For internal constraints, C should be independent of rigid body motions, i.e translation and rotation. Therefore,

the gradient of C with respect to $\boldsymbol{p}$ ($\nabla_{\boldsymbol{p}} C$) is orthogonal to the rigid body modes as it is the direction of maximal change. If the displacement $\Delta\boldsymbol{p}$ is chosen to be along $\nabla_{\boldsymbol{p}} C$, both linear and angular momenta will be conserved. Thus, given a configuration of vertex positions $\boldsymbol{p}$, PBD attempts to find a displacement $\Delta\boldsymbol{p}$ such that $C(\boldsymbol{p}+\Delta\boldsymbol{p}) = 0$ (or $C(\boldsymbol{p}+\Delta\boldsymbol{p}) \geq 0$ for unilateral constraints). As many constraints yield non-linear equations (or inequalities), solving for $\Delta\boldsymbol{p}$ can be non-trivial. However, this equation can be approximated by Taylor series expansion:

$$C(\boldsymbol{p}+\Delta\boldsymbol{p}) \approx C(\boldsymbol{p}) + \nabla_{\boldsymbol{p}} C(\boldsymbol{p}) \cdot \Delta\boldsymbol{p} = 0. \tag{3.5}$$

In restricting $\Delta\boldsymbol{p}$ to be in the direction of $\nabla_{\boldsymbol{p}} C$, it follows that for some scalar (a Lagrange multiplier) $\lambda \in \mathbb{R}$,

$$\Delta\boldsymbol{p} = \lambda \nabla_{\boldsymbol{p}} C(\boldsymbol{p}). \tag{3.6}$$

Substituting Equation 3.6 into Equation 3.5, and solving for λ, and substituting back into Equation 3.6 yields the following formula for $\Delta\boldsymbol{p}$:

$$\Delta\boldsymbol{p} = -\frac{C(\boldsymbol{p})}{\|\nabla_{\boldsymbol{p}} C(\boldsymbol{p})\|^2} \nabla_{\boldsymbol{p}} C(\boldsymbol{p}), \tag{3.7}$$

which is recognized as a Newton-Raphson step for the iterative solution of the non-linear equation given by constraint C. The displacement for a single vertex position $\boldsymbol{p}_i$ can then be written as

$$\Delta\boldsymbol{p}_i = -s \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n) \tag{3.8}$$

where s is a scaling factor

$$s = \frac{C(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n)}{\sum_j \|\nabla_{\boldsymbol{p}_j} C(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n)\|^2}. \tag{3.9}$$

If the vertices have individual masses $m_i \in \mathbb{R}_{>0}$ (indicating their resistance to being repositioned), then PDB defines the inverse mass $w_i = \frac{1}{m_i}$. This conveniently allows for vertices to carry infinite mass $w_i = 0$, effectively fixing them in space. Equations 3.9 and 3.8 may now be formulated as

$$\Delta\boldsymbol{p}_i = -\frac{w_i C(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n)}{\sum_j w_j \|\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n)\|^2} \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n). \tag{3.10}$$

To solve this system of constraints over the entire mesh, with possibly overlapping subsets of

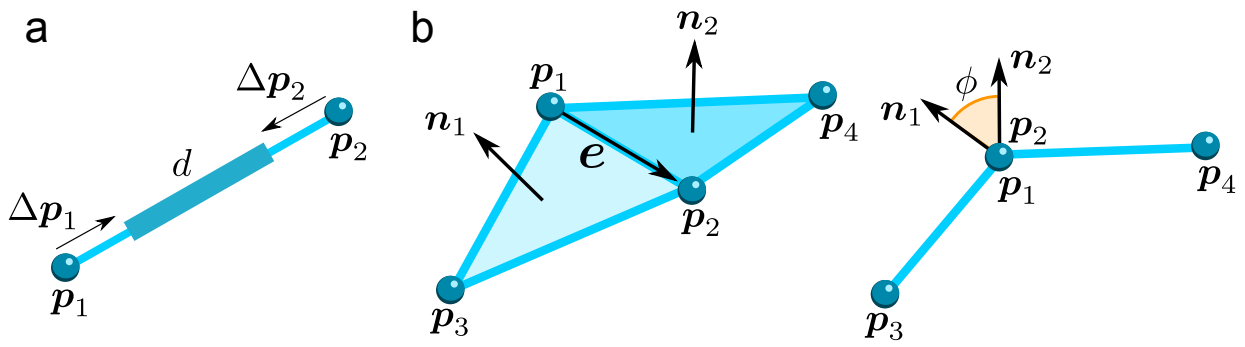

Figure 3.1: Internal constraints on geometric configurations. (a) Distance constraint on vertices $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ of an edge, repositioning them to maintain a length d. (b) Dihedral angle constraint on the adjacent triangles, repositioning them to maintain an angle ϕ between the normals $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$.

vertices, PBD uses a non-linear Gauss-Seidel-like method. It is similar in that the constraints are solved independently, and one after another, using the updated positions in subsequent constraint solves (i.e. the solution to constraint C_i uses any positions already updated by constraints C_j that were solved prior to C_i, where $0 \leq j < i$).

Distance constraints correspond to the edges of the mesh and represent the desired length d between the vertices of the edge $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ (Figure 3.1a)

$$C_{dist}(\boldsymbol{p}_1, \boldsymbol{p}_2) = \|\boldsymbol{p}_1 - \boldsymbol{p}_2\| - d \tag{3.11}$$

PBD repositions $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, moving them along the gradient of the constraint by $\Delta\boldsymbol{p}_1$ and $\Delta\boldsymbol{p}_2$ respectively, such that the constraint is satisfied: $C_{dist}(\boldsymbol{p}_1 + \Delta\boldsymbol{p}_1, \boldsymbol{p}_2 + \Delta\boldsymbol{p}_2) = 0$.

Angular (bending) constraints correspond to neighbouring triangles and represents the desired signed dihedral angle ϕ between the pair of adjacent triangles $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3)$ and $(\boldsymbol{p}_1, \boldsymbol{p}_4, \boldsymbol{p}_2)$ sharing an edge $(\boldsymbol{p}_1, \boldsymbol{p}_2)$ (Figure 3.1b)

$$\begin{aligned} C_{bend}(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4) &= \arccos(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2) - \phi \\ &= \arccos\left(\frac{(\boldsymbol{p}_3 - \boldsymbol{p}_1) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)}{\|(\boldsymbol{p}_3 - \boldsymbol{p}_1) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)\|} \cdot \frac{(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_4 - \boldsymbol{p}_1)}{\|(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_4 - \boldsymbol{p}_1)\|}\right) - \phi \end{aligned} \tag{3.12}$$

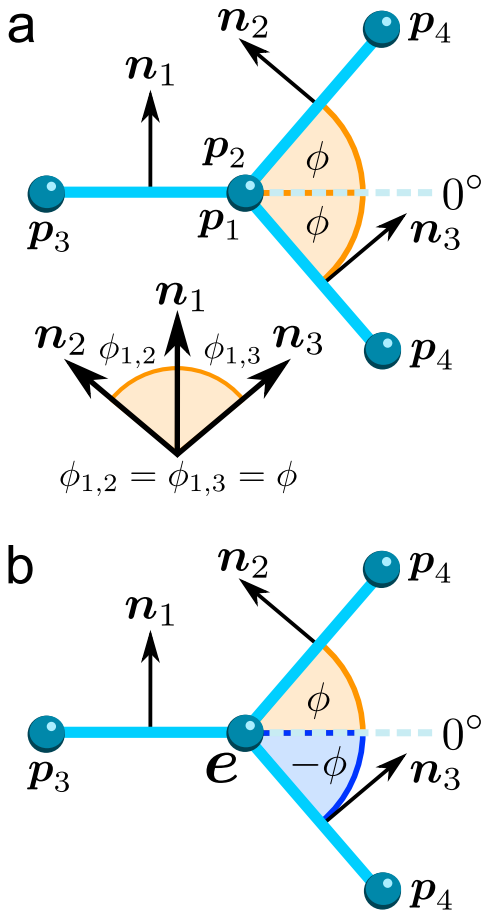

Figure 3.2: Side view of two possible adjacent triangle configurations (overlapped with one another) yielding (a) the same angle measurement, or (b) distinguishable *signed* angle measurements.

The bending constraint of Muller et al. [2], seen in Equation 3.12, measures the difference of the angle found between the triangle normals ($\boldsymbol{n}_1$ and $\boldsymbol{n}_2$) and the desired dihedral angle ϕ. Unfortunately, as this measurement is effected by a *dot product* of vector normals, there are mirror configurations of adjacent triangles that represent the same dihedral angle (see Figure 3.2a). Therefore, if vertex $\boldsymbol{p}_4$ is repositioned to the opposite side of the plane defined by $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$, PBD will inadvertently push the triangles into the mirror configuration. This duality of resting poses is unsuitable for maintaining the integrity of the floret forms designed in Chapter 2 as they are deformed by, for example, collisions or growth. To disambiguate the mirrored configurations, we choose a canonical orientation for the shared edge between the adjacent triangles: $\boldsymbol{e} = \boldsymbol{p}_2 - \boldsymbol{p}_1$ (see Figures 3.1b, 3.2b). As the cross product $\boldsymbol{n}_1 \times \boldsymbol{n}_2$ is the reverse direction of $\boldsymbol{n}_1 \times \boldsymbol{n}_3$, this allows us to measure alignment to the canonical edge, giving a *sign* to the dihedral angle

$$\phi = \text{sgn}\left(\boldsymbol{e} \cdot [\boldsymbol{n}_1 \times \boldsymbol{n}_2]\right) \arccos\left(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2\right) \qquad (3.13)$$

and disambiguate the mirrored configurations. As well, the signed dihedral angle defines a continuous range of angles from $[-180, 180)$ (see Figure 3.2b) for the bending constraint to rotate the adjacent triangles through, as required in our constraint based growth. Appendix A details the linearization and position update of the signed dihedral angle bending constraint.

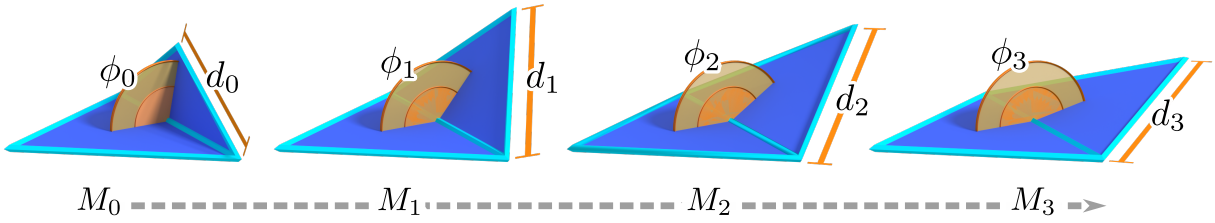

Figure 3.3: A sequence of polygonized meshes $M_0, M_1, M_2, M_3, \ldots$, (each defined in terms of their vertex positions in $\mathbb{R}^3$), representing select poses of a meshes development. We extract a sequence of edge lengths d_i and dihedral angles ϕ_i from this mesh sequence. Mapping edge length constraints and dihedral angle constraints to the mesh M_0, and sequentially changing the constraint rest values to those extracted, the mesh is developed similarly to the sequence of polygonized meshes.

3.2 Growth

PBD provides the means by which to encode a desired shape of a mesh via constraints on local configurations of vertices. I have modified PBD to recreate the floret growth designed in Chapter 2. To simulate the growth of the inflorescence and the opening of the flower, I iteratively progress through the sequence of polygonized meshes $M_0, M_1, \ldots, M_n$ representing consecutive stages of the inflorescence canopy development. In each step t, we use the edge lengths, signed dihedral angles and attachment positions in mesh M_i to define the corresponding constraint set $\mathcal{C}^t$ (see Figure 3.3). Attachment constraints (vertex positions with infinite masses, $w_i = 0$) define positions and orientations of individual florets in the canopy (see Chapter 5). If one or more collisions

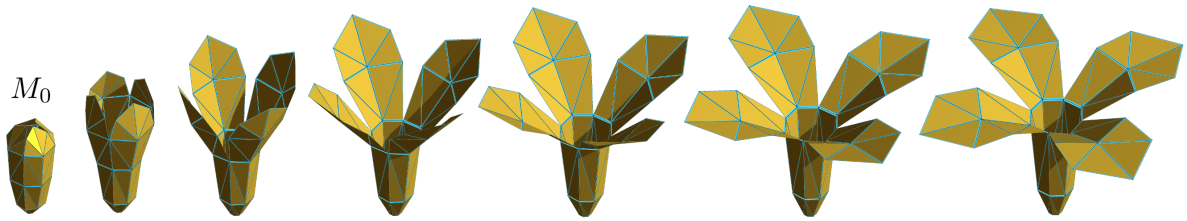

Figure 3.4: Floret mesh development driven by constraint based growth.

occur, we add constraints preventing triangle intersection to the set $\mathcal{C}^t$ before resolving it. Once a collision-free steady state is found (see Chapter 4), we advance developmental time and progress to the next simulation step, $t+1$. Our simulation is thus similar to the typical application of PBD to simulate cloth or thin shells, except that not only collision constraints, but also all other constraints, may change from one simulation step to the next as the inflorescence grows. Figure 3.4 exhibits constrained growth, with a system of constraints mapped to the vertices of mesh M_0, and sequentially changing the constraint rest values (edge lengths and signed dihedral angles) to those extracted from a developmental sequence designed by our Floret Editor (see Chapter 2).

Chapter 4

Collision Handling

4.1 Previous work

In early models of flowers and inflorescences (e.g. [4; 5; 8]) the modeler minimized the visual impact of intersections between petals by carefully crafting their shapes. More recently, Ijiri et al. [25] outlined a specialized algorithm for handling collisions between petals or sepals, which exploited their spatial ordering as determined by the flower structure. In inflorescences a similar assumption of spatial ordering cannot be made; moreover, an inflorescence may include hundreds or thousands of florets, making the number of polygons representing the inflorescence as a whole correspondingly higher. Consequently, we have adapted to inflorescences a more general method of collision detection and resolution originally developed for cloth, but also applied to leaves [2]. In contrast to cloth, the set of polygons representing an inflorescence can be divided into subsets that represent individual organs [39]. We take advantage of this observation by considering different florets of the same type as instances — modified by collisions — of a common dynamic geometric template (Chapter 2).

4.2 Collision detection

Polygon meshes representing an inflorescence canopy may be represented by tens, if not hundreds, of thousands of triangles, even with a careful polygonization (Chapter 2.4). Efficient detection of colliding triangles is thus of key importance to the overall efficiency of the inflorescence modeling. We assume that the initial configuration of florets in a young inflorescence is collision-free, then detect and resolve collisions as they arise during the simulated development. This approach is consistent with the physical nature of development and has the additional advantage of being conducive to animation. Potentially, when only a static model of an inflo-

rescence is needed, intersections in the polygon mesh representing the final structure could be untangled using a history-free method [40]. With this approach, however, selecting the correct solution (plausible from the developmental perspective) from many possible untangled configurations is difficult. Instead, I follow the continuous collision detection (CCD) paradigm (cf. [41]). With petals represented as triangle meshes growing over time, we thus need to find intersections between triangular prisms representing the development-driven motion of triangles. We consider vertex-triangle and edge-edge intersections separately[1] (Figure 4.1), applying a broad and narrow phase for vertex-triangle pairs, and then a broad and narrow phase for edge-edge pairs. Considering each intersection separately guarantees that unique pairs are only tested for intersection once, as opposed to considering triangle-triangle pairs where the 6 vertex-triangle and 9 edge-edge pairs between two triangles may have been computed in a previous triangle pair (e.g. the edge shared by two triangles will have been tested twice as many times as necessary if all triangle-triangle pairs are tested).

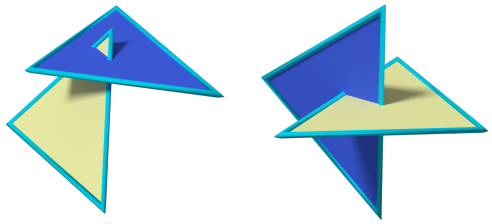

Figure 4.1: The two fundamental types of intersections between a pair of triangles. Left: A triangle vertex passing through the face of another triangle. Right: The edge of a triangle passing through the edge of another triangle.

4.2.1 Broad phase collision detection

The broad phase is accelerated by partitioning space into a regular grid (Figure 4.2). For vertex-triangle intersections, each voxel stores the IDs of all triangles that may intersect this voxel over a given time step. This is done by constructing axis-aligned bounding boxes (AABB) containing the space-time prisms of the triangles as they move through space, and intersecting them with the voxels. The boxes are enlarged by a small amount to address potential numerical inaccuracies when detecting intersections [42]. The information in each voxel is time-stamped so that out-

[1] Vertex-edge intersections are considered special cases of either edge-edge and vertex-triangle intersections.

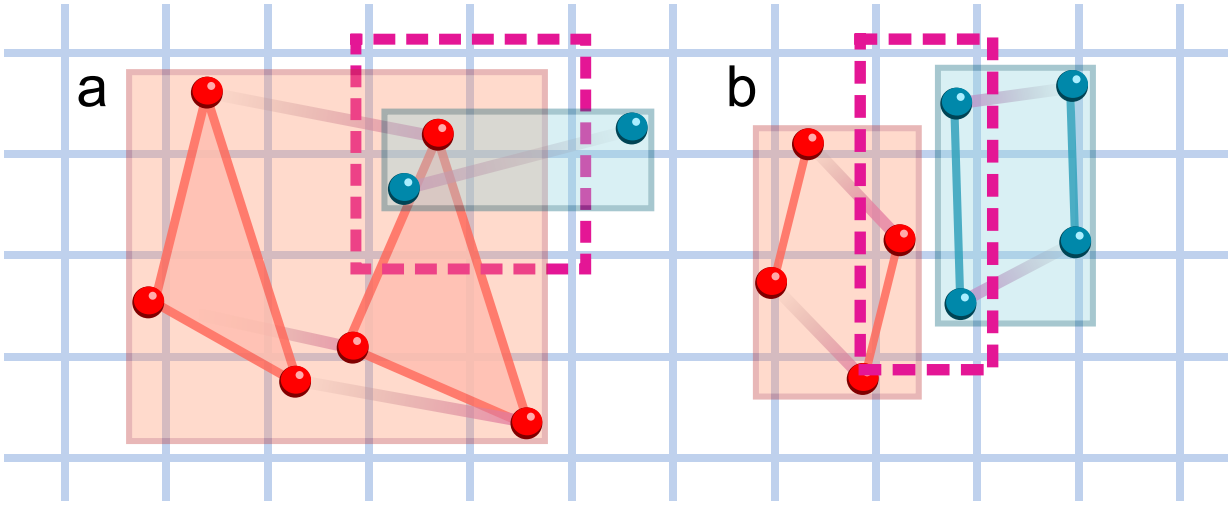

Figure 4.2: Broad phase collision detection: Spatial domain is divided into *voxels* for efficiently detecting potential collisions between pairs of mesh elements. a) The *axis-aligned bounding box* (AABB) of a moving triangle (red) and moving vertex (blue) overlap in four shared *voxels* (outlined), therefore the vertex-triangle pair must be tested for intersection using narrow phase collision detection techniques. b) The AABB of moving triangle edges (red and blue) overlap in three shared voxels (outlined), and even though (upon visual inspection) it is not possible for the edges to intersect over the time step, narrow phase collision detection techniques must be used to verify whether an intersection took place or not.

of-date informations can be cleared efficiently, only if the voxel is visited in a subsequent time step [43]. To detect whether a moving vertex may intersect a triangle, an AABB is created around the line segment representing the vertex motion. All triangles in the voxels that intersect with this AABB are considered in the narrow phase. Broad-phase detection of potential edge-edge collisions is carried out in a similar manner, except that, in this case, voxels contain information about the edges which intersect them during their motion through space, rather than the entire triangles.

Another approach for testing all vertex-triangle and edge-edge pairs was introduced by Wong et al. [44], which uses a randomized marking scheme on a subset of the mesh triangles to create unique feature (edge-edge, vertex-triangle) pairs. Although the number of feature pair intersection tests are the same as considering vertex-triangle and edge-edge pairs separately, and there is a cost to precomputing the marking scheme, there is a potential speed increase in only one round of broad phase collision detection, on a fewer elements (for most practical applications, meshes have a ratio

of vertices to triangles to edges of 1:2:3 [45]).

4.2.2 Narrow phase collision detection

Narrow-phase collision detection is performed using the method of Provot [46]. In a given time interval of the simulation, each vertex position $\boldsymbol{p}$ changes from its initial position $\boldsymbol{p_0} \in \mathbb{R}^3$ at the beginning of the time step to another position $\boldsymbol{p_1} \in \mathbb{R}^3$ at the end of the time step. We can then define a linear function for its movement over the time step,

$$\boldsymbol{p}(t) = \boldsymbol{p_0} + t(\boldsymbol{p_1} - \boldsymbol{p_0}) \quad \text{for} \quad t \in [0,1]. \tag{4.1}$$

With the assumption that the current triangle configuration is collision-free, we simply need to solve for the values of t where the four vertices representing a moving point and a triangle, or the endpoints of two edges, become co-planar in their linear motions (Figure 4.3). If this is the case, a test is performed to determine whether the moving point is within the triangle, or whether the intersection point is within both segments, respectively.

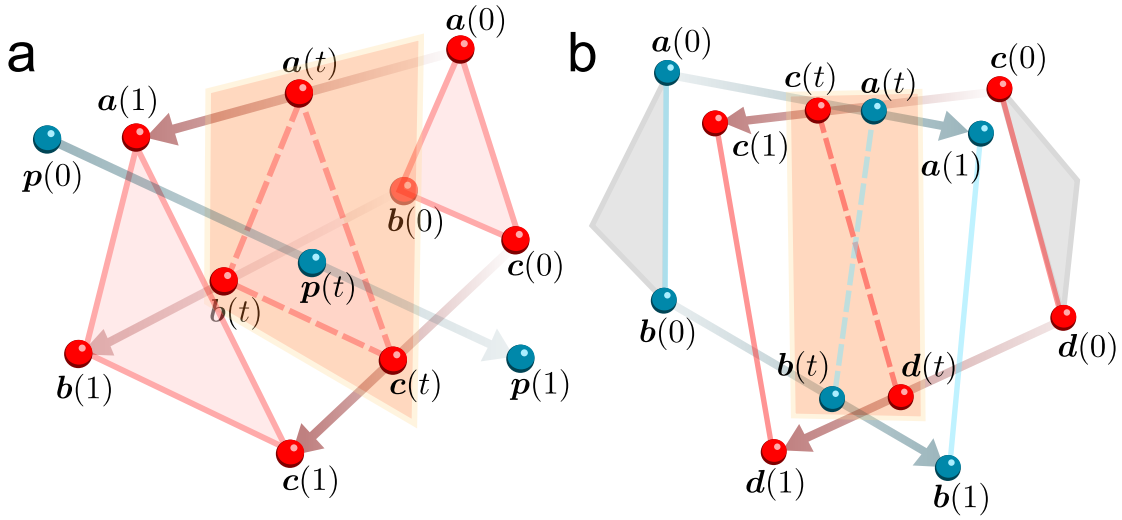

Figure 4.3: Narrow phase collision detection: finding moments t, over a simulation time step $t \in [0,1]$, when four vertices are coplanar (depicted here within the orange plane). (a) Detection of moment t at which vertex $\boldsymbol{p}$ and triangle $(\boldsymbol{abc})$ are coplanar. (b) Detection of moment t at which the vertex positions of edges $(\boldsymbol{ab})$ and $(\boldsymbol{cd})$ are coplanar.

4.2.2.1 Vertex-triangle collision detection

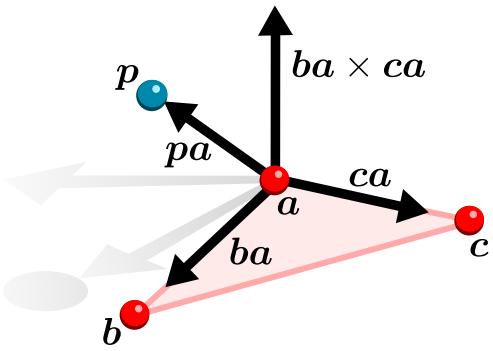

Figure 4.4: Vectors constructed from vertex $\boldsymbol{p}$, and triangle $\boldsymbol{abc}$, used in Equation 4.2.

Let $\boldsymbol{p}(t)$ be a moving vertex, and $\boldsymbol{a}(t)$, $\boldsymbol{b}(t)$ and $\boldsymbol{c}(t)$ be the vertices of a moving triangle. The moving triangle is collectively defined as $\boldsymbol{abc}(t)$ at time t. To solve for the time t where $\boldsymbol{p}(t)$ and $\boldsymbol{abc}(t)$ are coplanar, and thus potentially colliding, it suffices to find a vector from $\boldsymbol{a}(t)$ to $\boldsymbol{p}(t)$ that is coplanar to the vectors from $\boldsymbol{a}(t)$ to $\boldsymbol{b}(t)$, and $\boldsymbol{a}(t)$ to $\boldsymbol{c}(t)$. Denoting these vectors as $\boldsymbol{pa}(t) = \boldsymbol{p}(t) - \boldsymbol{a}(t)$, $\boldsymbol{ba}(t) = \boldsymbol{b}(t) - \boldsymbol{a}(t)$ and $\boldsymbol{ca}(t) = \boldsymbol{c}(t) - \boldsymbol{a}(t)$, respectively, this is equivalent to finding the vector $\boldsymbol{pa}(t)$ that is orthogonal to the vector $\boldsymbol{ba}(t) \times \boldsymbol{ca}(t)$ that defines the plane spanned by $\boldsymbol{ba}(t)$ and $\boldsymbol{ca}(t)$ (see Figure 4.4). Therefore, $\boldsymbol{p}(t)$ and $\boldsymbol{abc}(t)$ are coplanar at time t if and only if

$$[\boldsymbol{ba}(t) \times \boldsymbol{ca}(t)] \cdot \boldsymbol{pa}(t) = 0. \tag{4.2}$$

To solve for t we expand $\boldsymbol{ba}(t) \times \boldsymbol{ca}(t)$ as a quadratic equation of vector coefficients $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{S}$ $\in \mathbb{R}^3$ (see Appendix B.1),

$$\boldsymbol{ba}(t) \times \boldsymbol{ca}(t) = \boldsymbol{Q}t^2 + \boldsymbol{R}t + \boldsymbol{S}. \tag{4.3}$$

then we expand Equation 4.2 to a cubic equation with real coefficients A, B, C and $D \in \mathbb{R}$:

$$\begin{aligned} [\boldsymbol{ba}(t) \times \boldsymbol{ca}(t)] \cdot \boldsymbol{pa}(t) &= \left[\boldsymbol{Q}t^2 + \boldsymbol{R}t + \boldsymbol{S}\right] \cdot \boldsymbol{pa}(t) \\ &= At^3 + Bt^2 + Ct + D. \end{aligned} \tag{4.4}$$

After the roots $t \in [0,1]$ are found, if any (see Chapter 4.2.2.3), each is tested to verify that vertex $\boldsymbol{p}(t)$ and triangle $\boldsymbol{abc}(t)$ are actually intersecting at time t, and are not just coplanar (See Appendix B.2).

4.2.2.2 Edege-edge collision detection

Let $\boldsymbol{ab}(t)$ and $\boldsymbol{cd}(t)$ be two moving edges composed of vertices $\boldsymbol{a}(t)$ and $\boldsymbol{b}(t)$, and $\boldsymbol{c}(t)$ and $\boldsymbol{d}(t)$, respectively. To solve for time t where $\boldsymbol{ab}(t)$ and $\boldsymbol{cd}(t)$ are coplanar, and thus potentially colliding, it suffices to find a vector from $\boldsymbol{c}(t)$ to $\boldsymbol{a}(t)$ that is coplanar to the vectors from $\boldsymbol{d}(t)$ to $\boldsymbol{a}(t)$ and $\boldsymbol{b}(t)$ to $\boldsymbol{a}(t)$. This is equivalent to finding the vector $\boldsymbol{ca}(t)$ that is orthogonal to the vector $\boldsymbol{da}(t) \times \boldsymbol{ba}(t)$ that defines the plane spanned by $\boldsymbol{da}(t)$ and $\boldsymbol{ba}(t)$. Therefore, $\boldsymbol{ab}(t)$ and $\boldsymbol{cd}(t)$ are coplanar at time t if and only if

$$[\boldsymbol{da}(t) \times \boldsymbol{ba}(t)] \cdot \boldsymbol{ca}(t) = 0 \tag{4.5}$$

where, as in Equation 4.3, t is a (*real*) root of a cubic equation. After the roots $t \in [0,1]$ are found, if any (see Chapter 4.2.2.3), each must be tested to verify whether the edges $\boldsymbol{ab}(t)$ and $\boldsymbol{cd}(t)$ are intersecting at time t, and are not just coplanar. Although this can be tested in $\mathbb{R}^2$, as both edges are coplanar at time t, it is more numerically stable to calculate the minimum distance between two line segments (triangle edges) in $\mathbb{R}^3$ (See Appendix B.3). If this minimum distance is less than a threshold, the edges are considered to be intersecting in $\mathbb{R}^3$.

4.2.2.3 Root Finding

The cubic equations derived from Equations 4.2, 4.5, for finding the moments $t \in [0,1]$ over the simulation time step at which four vertices are coplanar, can be solved in a number of ways. In 1545 Gerolamo Cardano published a formula (following the methods of Tartaglia of the same time) of an analytical solution to cubic polynomials [47]. However, as with the quadratic and quartic formulas, care must be taken when implementing the cubic formula for computer calculations, which have finite precision in storage and arithmetic. These formulas for polynomial roots of degrees less than five require manipulations of the polynomial coefficients with finite precision operations (i.e. cube root, arccosine, cosine, exponentiation), therefore it becomes imperative to minimize round off error as even tiny perturbations in the coefficients of some polynomials may dramatically move its roots in the complex plane [48]. This lack of computational *stability* lends to the adoption of iterative root finding algorithms over analytical formulas [48]. However, care

must also be taken in choosing from among the many iterative polynomial root finding algorithms. Some methods sacrifice robustness (i.e. *Newton-Raphson's Method* vs. *bisection Method*) in lieu of speed (i.e. quadratic vs. linear convergence), or require complex ($\mathbb{C}$) arithmetic (i.e. *Laguerre's Method*). Fortunately, to solve our collision problem we require only unique, real roots $t \in [0,1] \subset \mathbb{R}$, therefore we use a *Hybrid Method* for bracketed, real solutions [48; 49]. This *Hybrid Method* takes an interval for which a solution is known to exist (by the intermediate value theorem), and utilizes both the bisection and Newton-Raphson method to converge on its value. If a Newton-Raphson step proposes a solution outside of the interval, then a bisection step is used instead. We can efficiently evaluate and bracket (i.e. calculate critical points) the cubic equation using Horner's method of polynomial evaluation. Although it is necessary to verify the occurrence a collision for each root $t \in [0,1]$, the number of verified collisions are of import. Each verified collision signifies a moment the pair of mesh elements (vertex-triangle, edge-edge) pass through one another. If an *even* number of collisions occur over the time step, the pair of mesh elements will be free of intersections. As this leaves the mesh elements in a collision free and physically plausible configuration, we ignore resolving the intersection from the first collision (smallest root $t \in [0,1]$ verified resulting in a collision).

Additionally, coefficient culling techniques exist [50] to predetermine if a given cubic equation of the form $a_3t^3 + a_2t^2 + a_1t + a_0 = 0$, with $a_i \in \mathbb{R}$, has no real root in the interval $[0,1]$. For instance, if all a_i have the same sign, no root within the interval can possibly exist. Other criteria are used to efficiently cull cubic root computations in our root finding routine. If any of the following hold: 1) $a_0, a_1 > 0$ and $a_0 + a_1 > |a_2| + |a_3|$, 2) $a_0, a_1 < 0$ and $a_0 + a_1 < -|a_2| - |a_3|$, or 3) $|a_0| > |a_1| + |a_2| + |a_3|$, then no root may exist within the interval $[0,1]$.

4.3 Collision resolution

We resolve collisions by adapting position-based dynamics (PBD) [2; 36] (see Chapter 3 for a brief review) to the simulation of growth (incidentally, we note that the related problem of resolving

collisions between leaves in a set of plants was used to illustrate the power of PBD in the original paper [2]). Consistent with the PBD method, we map mesh M_0 representing the initial (collision-free) floral canopy into a network of constraints. Distance constraints correspond to the edges of this mesh and represent their length. Angular constraints correspond to the signed dihedral angles between all pairs of adjacent triangles in the mesh. Attachment constraints define positions and orientations of individual florets in the canopy (see Chapter 5). We denote the resulting set of constraints as $\mathcal{C}_0$. We then simulate the growth of the inflorescence and the opening of flowers. To this end, we iteratively progress through the sequence of polygonized meshes $M_0, M_1, \ldots, M_n$ representing consecutive stages of the inflorescence canopy development. In each step i, we use the edge lengths, dihedral angles and attachment positions in mesh M_i to define the corresponding constraint set $\mathcal{C}_i$. If one or more collisions are detected, we add a collision constraint to the set $\mathcal{C}_i$ for each group of vertices corresponding to a vertex-triangle collision or edge-edge collision. Collision constraints are *unilateral* constraints on the four vertex positions responsible for the mesh elements intersection. The collision constraints are initiated such that the value of the current configuration is negative, i.e. $C_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\boldsymbol{d}) < 0$, and the intersection is resolved with any vertex configuration where the constraint is greater than or equal to 0, i.e. $C_i(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\boldsymbol{d}) \geq 0$. PBD then linearises the constraint set $\mathcal{C}_i$, and repositions the vertices of the mesh. Once a collision-free configuration is found, we advance developmental time and progress to the next simulation step, $i+1$.

4.3.1 Vertex-triangle collision constraint

In the case that a collision between vertex $\boldsymbol{q}$ and triangle $(\boldsymbol{abc})$ (comprised of vertices $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$) is detected over a simulation step i, a vertex-triangle collision constraint C_{vt} is added to the set $\mathcal{C}_i$. The constraint restricts $\boldsymbol{q}$ to maintain a distance of h from the plane defined by the triangle $(\boldsymbol{abc})$ (see Figure 4.5). This constraint takes the form

$$\begin{aligned} C_{vt}(\boldsymbol{q},\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) &= (\boldsymbol{q}-\boldsymbol{a})\cdot\boldsymbol{n} - h \;\geq\; 0 \\ &= (\boldsymbol{q}-\boldsymbol{a})\cdot\left(\frac{(\boldsymbol{b}-\boldsymbol{a})\times(\boldsymbol{c}-\boldsymbol{a})}{\|\,(\boldsymbol{b}-\boldsymbol{a})\times(\boldsymbol{c}-\boldsymbol{a})\,\|}\right) - h \;\geq\; 0 \end{aligned} \tag{4.6}$$

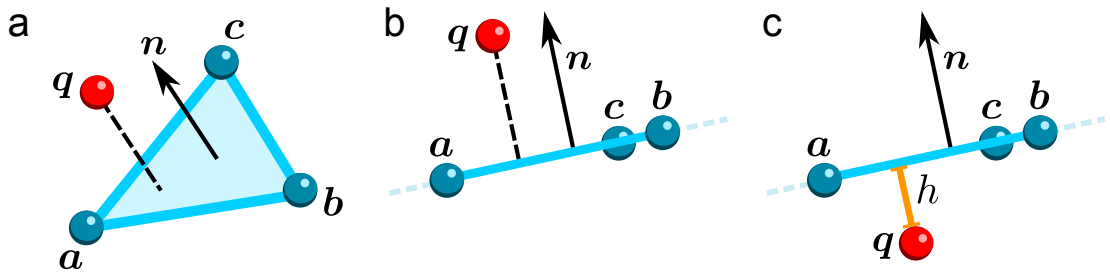

Figure 4.5: Principle of vertex-triangle constraint. (a) Vertex $\boldsymbol{q}$ is a certain distance away from plane of the triangle $(\boldsymbol{abc})$, suppose it has passed through the triangle from the side opposite to the normal $\boldsymbol{n}$. (b) Side view of the configuration. Vertex $\boldsymbol{q}$ should be repositioned back to the opposite side of the triangle, resolving any intersections arising from triangles that contain $\boldsymbol{q}$ (not visualized here). (c) Vertex $\boldsymbol{q}$ is repositioned back to the side of the triangle it entered through, separated by a distance of h.

With this formulation, $\boldsymbol{q}$ is constrained to one side of the triangle plane, the side oriented with the triangle normal $\boldsymbol{n}$. Thus, if $\boldsymbol{q}$ enters the triangle from below, the normal $\boldsymbol{n}$ direction is reversed, so that PDB will reposition $\boldsymbol{q}$ to the correct side.

4.3.2 Edge-edge collision constraint

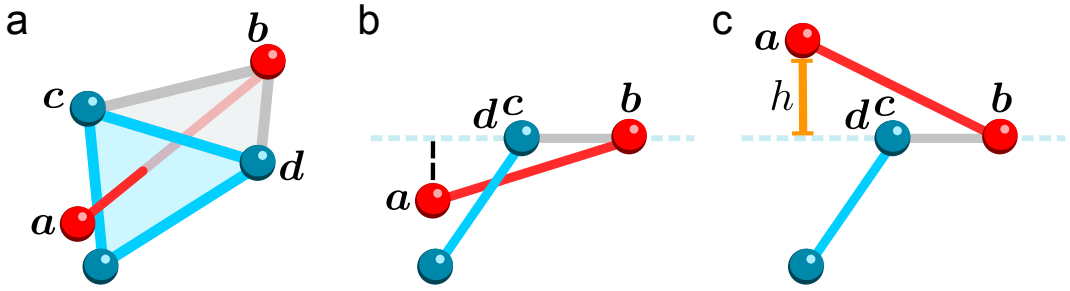

Figure 4.6: Principle of edge-edge collision constraint. (a) Colliding edges $(\boldsymbol{a},\boldsymbol{b})$ and $(\boldsymbol{c},\boldsymbol{d})$ have produced an intersection between the edge $(\boldsymbol{a},\boldsymbol{b})$ (red) and mesh triangle (blue). A *pseudo*-triangle $(\boldsymbol{bcd})$ (grey) is displayed here as well. (b) Side view and geometric configuration of intersection between edge $(\boldsymbol{a},\boldsymbol{b})$ (red) and mesh triangle (blue). The *pseudo*-triangle (grey) defines the plane that $\boldsymbol{a}$ needs to move through to resolve the collision. (c) The vertex $\boldsymbol{a}$ is repositioned above the plane defined by the *pseudo*-triangle, separated by a distance of h, resolving the intersection.

In the case that a collision between edges $(\boldsymbol{ab})$ and $(\boldsymbol{cd})$ (comprised of vertices $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{d}$ respectively) is detected over a simulation step i, an edge-edge collision constraint C_{ee} is added

to the set $\mathcal{C}_i$. Bender et al. [51] describes an extension to Position-based dynamics that resolves collisions between edges. However, prior to learning about their work, I designed the exact same edge-edge collision constraint. Edge collisions have a similar geometric configuration to vertex-triangle collisions: a *pseudo*-triangle may be constructed from vertices $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{d}$, with vertex $\boldsymbol{a}$ remaining free (see Figure 4.6a). The vertices of each edge are repositioned such that vertex $\boldsymbol{a}$ is a distance h above the *pseudo*-triangle ($\boldsymbol{bcd}$) (see Figure 4.6b). This constraint C_{ee} is formulated as follows

$$C_{ee}(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\boldsymbol{d}) = (\boldsymbol{a}-\boldsymbol{b}) \cdot \left(\frac{(\boldsymbol{c}-\boldsymbol{b}) \times (\boldsymbol{d}-\boldsymbol{b})}{\| (\boldsymbol{c}-\boldsymbol{b}) \times (\boldsymbol{d}-\boldsymbol{b}) \|} \right) - h \geq 0 \tag{4.7}$$

4.4 Subdivision smoothing

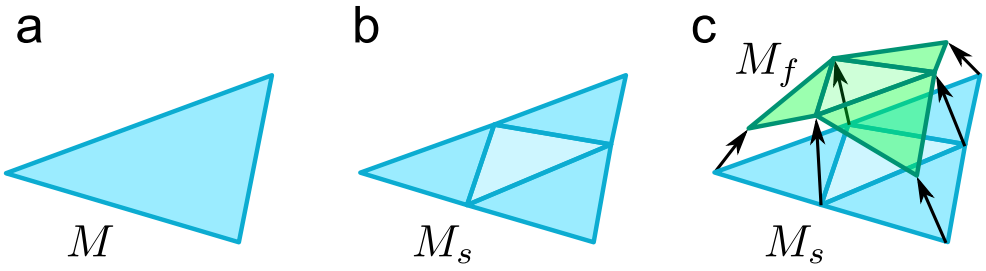

Figure 4.7: Subdivision smoothing of a mesh M. (a) A single triangle within M. (b) Quadrisection of the triangles within M, generating a new mesh M_s. (c) The vertices of M_s are repositioned, generating a smoother mesh M_f.

The coarse polygonization that afforded efficient collision handling, can also produce unnatural depictions of the florets when rendered. As a low polygon count is desirable for efficient collision handling and simulation, conversely, a smoother, higher resolution floral canopy is required for the final models to be appropriately rendered. Thus, to render our floral canopy realistically, its coarse representation must be post-processed into a smoother form, while preserving the collision-free configuration. Similar to the post processing technique by Bridson [42], I employ the Loop mesh subdivision technique [52] to smooth the coarse triangulations, and resolve collision after each smoothing iteration. To this end, we quadrisect each triangle face of a mesh M that is assumed to be

initially collision-free. Introducing new vertices at the edge mid-points (see Figure 4.7a,b), a mesh M_s is created that has a finer resolution than M. Subdividing each triangle in its plane guarantees that the subdivided mesh M_s will also be collision-free. Then, to produce a smoother mesh M_f, the vertices of M_s are repositioned (Figure 4.7c). However, in repositioning the vertices of M_s, new triangle collisions may be introduced within M_f (Figure 4.8b). We handle these collisions as described in Sections 4.2 and 4.3, using M_s as the initial collision-free triangle configuration, and M_f as the desired triangle configuration. As new collisions are rarely introduced when subdividing the coarse mesh, due to the convex-hull property of the subdivision scheme, the number of collision constraints to be solved are few.

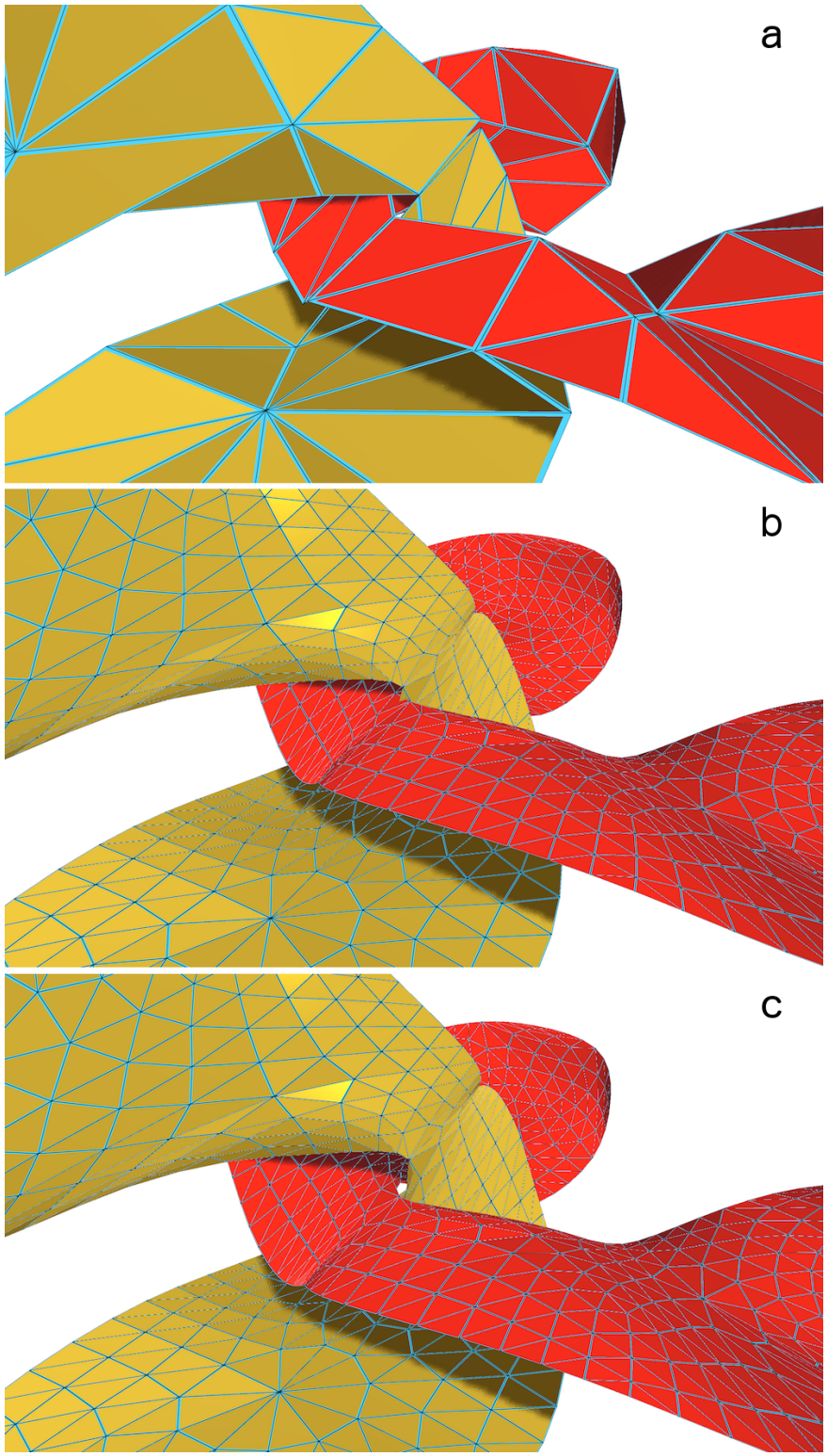

Figure 4.8: Collision handling in post processing subdivision smoothing. (a) Collision-free arrangement of coarse interlocking petals. (b) Two iterations of Loop subdivision of the coarse mesh with no collision handling, causing new intersections. (c) Two iterations of Loop subdivision with collision handling following each iteration.

Figure 4.9: A zoom into the lilac inflorescence from Figure 4.10 modeled with collision resolution turned off and on. Two flowers have been false-colored to facilitate comparisons.

4.5 Model

Figure 4.10 shows a sample inflorescence model with collisions between petals eliminated. We have used a previously published model of lilac inflorescence architecture [8] to support florets in space. A magnified view of a portion of this inflorescence (Figure 4.9) highlights intersections that would occur if collisions were not addressed, and shows how the algorithm resolves them by displacing individual florets and deforming their petals. Although neglecting collisions does not significantly affect the overall shape of the inflorescence, collision detection and resolution is essential to properly represent details in close-up views.

Figure 4.10: A model of a lilac inflorescence with collisions resolved.

4.6 Solving constraints with GPU vs. CPU

My implementation optionally utilizes GPU parallelization (using Nvidia's application program interface CUDA [53]) to solve the bending and distance constraints of inflorescence models. Although the bending and distance constraint systems are identical for both the GPU and CPU im-

plementations, the organization and method of solving have fundamental differences. The CPU version proceeds in a Gauss-Seidel manner, sequentially solving linearized constraints, thus is inherently unparallelizable. The GPU version solves the PBD constraint system in a Jacobi manner, i.e. linearizing and solving constraints independently of one another, and is thus parallelizable. However, solving each constraint in parallel is difficult, as many constraints share vertices (e.g. the vertex of a triangle fan is repositioned by multiple distance constraints, one for every every connected edge). To address this issue, constraint systems can be organized in a number of ways to avoid these constraint overlaps [36; 37]. My implementation of PBD on the GPU uses atomic operators, available in CUDA, for concurrent, thread safe data operations (e.g. integer addition and subtraction of variables accessed concurrently across multiple threads). Table 4.1 lists specific inflorescence models, their approximate triangle number and the amount of time to simulate their growth (with collisions) using either the CPU or GPU implementation to solve their bending and distance constraints.

Table 4.1: Simulation Timing

Model	Approx. # of triangles	Approx. Time (min.)
Yarrow (CPU)	350,000	60
Dahlia (CPU)	30,000	20
Sunflower (CPU/CUDA)	130,000	10
Gaillardia (CUDA)	25,000	10
Orlaya (CPU)	120,000	45
Lilac (CPU)	200,000	120
Dyssodia (CUDA)	100,000	10

Chapter 5

Phyllotactic pattern generation

5.1 Phyllotactic pattern generation

In the remainder of this thesis we focus on plants of the aster and carrot families (Asteraceae and Apiaceae), in which florets are arranged into spiral phyllotactic patterns. We recreate these patterns by extending a previous model [8] to simulate the dynamics of pattern formation. Dynamic simulation opens the door to animating the development of inflorescences, and improves the realism of static models by capturing the age and size differences between the florets. In addition, we consider the formation of hierarchical and recursive phyllotactic patterns.

5.2 Previous work

For nearly two centuries, researchers have been fascinated with phyllotactic patterns for their conspicuous regularity and intriguing mathematical properties, such as the emergent occurrence of Fibonacci numbers and the golden ratio. The beauty of inflorescences exhibiting diverse phyllotactic patterns has also made them an attractive modeling subject in computer graphics. The first models [4; 54] were based on analytic descriptions of spiral organ packing on the surface of a cylinder [55] or disk [56]. Using ideas similar to Vogel's [56], Lintermann and Deussen [7] developed an analytic description of spiral phyllotactic patterns on the surface of a sphere. Fowler et al. [5] and Prusinkiewicz et al. [8] extended the range of modeled patterns to primordia of varying size distributed on an arbitrary surface of revolution. The latter approach, based on an extension of Vogel's model by Ridley [26], is well suited for computer graphics applications due to its flexibility and robustness.

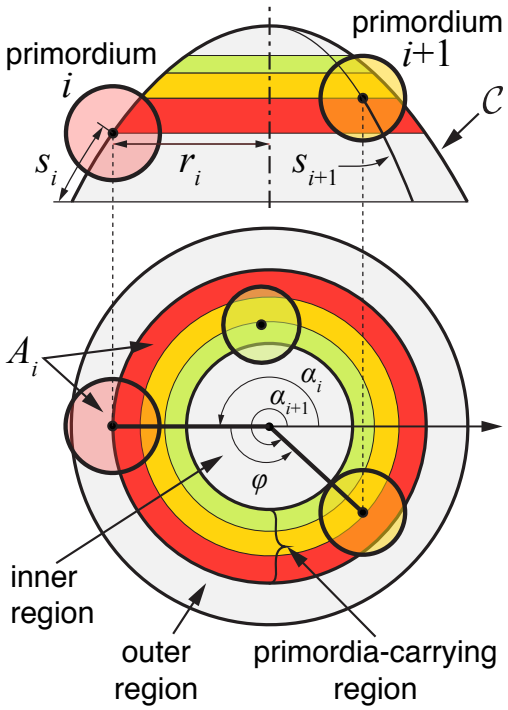

Figure 5.1: Geometric elements of the extended Ridley phyllotaxis model.

Ridley assumed that the supporting surface is defined by rotating a planar generating curve $\mathcal{C}$ around the surface axis (Figure 5.1). The position of primordium i is expressed by two parameters: angle α_i with respect to a reference direction, and position s_i along the generating curve. In contrast to the models that simulate local interactions between individual primordia (e.g. [5]), Ridley adopted a more global perspective and proposed to calculate the displacement $s_{i+1} - s_i$ of primordium $i+1$ with respect to primordium i by equating primordium area A_i with the area of a circular band circumscribing the supporting surface at the elevation of these primordia. This leads to the equation [8]:

$$\int_{s_i}^{s_{i+1}} 2\pi r(s)ds = A_i \quad \text{or} \quad \int_{s_i}^{s_{i+1}} \frac{2\pi r(s)}{A_i}ds = 1, \tag{5.1}$$

where $r(s)$ is the distance of point s on the generating curve $\mathcal{C}$ from the axis of the supporting surface. Given the initial position s_0 and a sequence of primordium areas $\{A_i\}$, Equation 5.1 (left or right) can be solved iteratively for $s_1, s_2, \ldots$, yielding the sequence of primordium positions $\{s_i\}$ along the generative curve $\mathcal{C}$. In modeling practice it is usually more convenient to define the primordium areas as a function $A(s)$ of their positions s rather than index i, which results in

$$\int_{s_i}^{s_{i+1}} \frac{2\pi r(s)}{A(s)}ds = 1. \tag{5.2}$$

With consecutive primordia positioned along the generative curve according to Equation 5.2, and the divergence angle $\alpha_{i+1} - \alpha_i$ between consecutive primordia equal to the golden angle $\varphi \approx 137.5°$, the primordia are packed in a dense spiral phyllotactic pattern [26]. We define function $A(s)$ interactively to make the distribution of floret sizes in the generated pattern match that observed in real inflorescences used for reference.

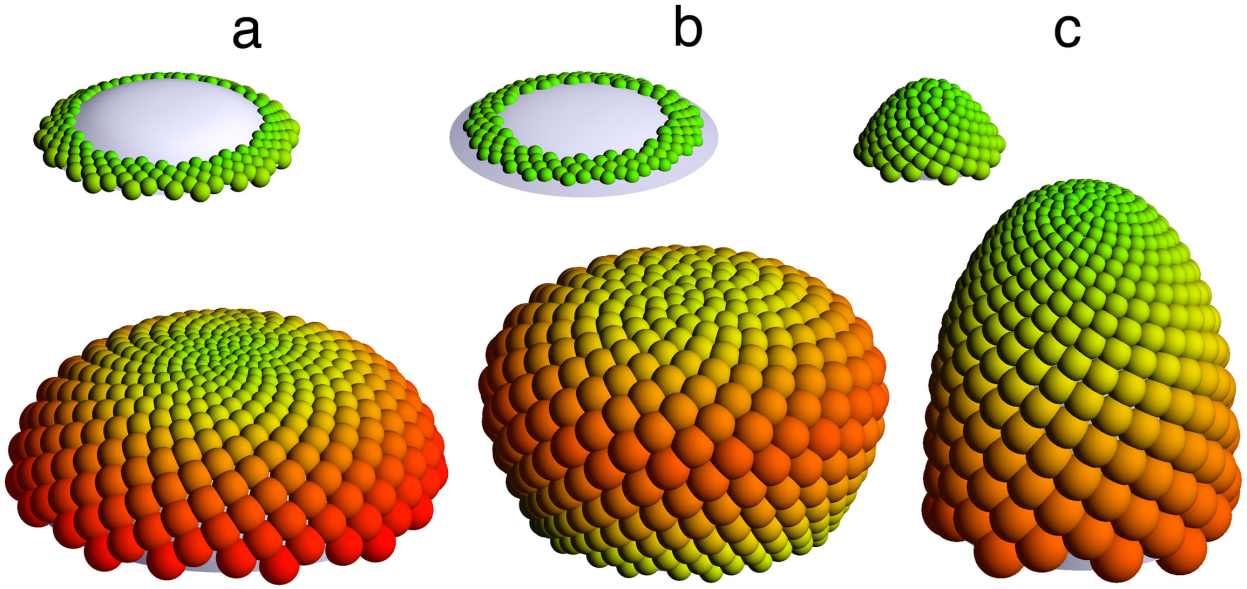

Figure 5.2: Earlier and later stages of phyllotactic pattern formation. (a) Centripetal fractionation. (b) Divergent fractionation. (c) Segregation. Colors indicate the age of primordia (green: younger; red: older).

5.3 Modeling the dynamics of phyllotaxis

We extend Ridley's model to capture not only the static distribution of primordia, but also the dynamics of their production and distribution over time. From the dynamic perspective, a plant may produce florets in two distinct modes, termed fractionation and segregation [14]. In the case of fractionation, the meristem is initially bare or "naked", and becomes gradually covered by the emerging primordia. In the case of segregation, new primordia originate near the tip of a continuously growing meristem when and where room is created for them.

We simulate two distinct cases of fractionation, as well as segregation, using a unified model. The reproductive meristem is divided into three regions: the naked inner and outer regions, and the primordium-carrying intermediate region (Figure 5.1). In centripetal fractionation (Figure 5.2a), commonly observed in developing heads, the outer region is absent, and the inner region initially occupies the entire meristem. Beginning at its rim, consecutive primordia are produced at fixed time intervals at positions determined by Equation 5.2, building up the primordium-carrying region

at the expense of the inner region from the outside in. This process terminates when the area of the inner region falls below that of a single primordium. In divergent fractionation [57] (Figure 5.2b), both the inner and outer regions are initially present. Primordium formation begins at the boundary between these regions and proceeds simultaneously inward and outward. Finally, in the case of segregation (Figure 5.2c), the initial conditions are similar to those in the centripetal fractionation, but the meristem is small, commensurate with a primordium. A new primordium is produced when the area of the growing inner region exceeds a threshold value. The inner region's area is then reduced by the area of the primordium, falling below the threshold until further growth increases it again. A sequence of primordia is thus produced, with the period dependent on the growth rate of the inner region.

In general, both the meristem and the primordia may grow over time. The growth rates of the inner and outer regions are specified as functions of the meristem age. Likewise, the growth of each primordium is a function of its age. This creates a progression of sizes and developmental stages of individual florets, which plays an important role both in the animation of inflorescence development and the reproduction of specific developmental stages alike. The modeler defines all functions using a graphical function editor. The same function is used for all primordia of the same type.

The last element of the phyllotaxis model is the growth of the supporting surface – the surface of a meristem or a receptacle – as a whole. To model changes in its size and shape, we interpolate between graphically-defined generating curves $\mathcal{C}$ that describe the profile of the supporting surface at select points in time. Furthermore, to coordinate the size of the supporting surface with the size of its regions, we sum up the current areas of all regions present (the primordium-carrying area is calculated by adding up the areas of all primordia) and scale the supporting surface to match that sum. The growth and changes in the shape of the supporting surface are featured in the simulations in Figure 5.2.

As described in Chapter 4, we detect and resolve collisions while simulating the development

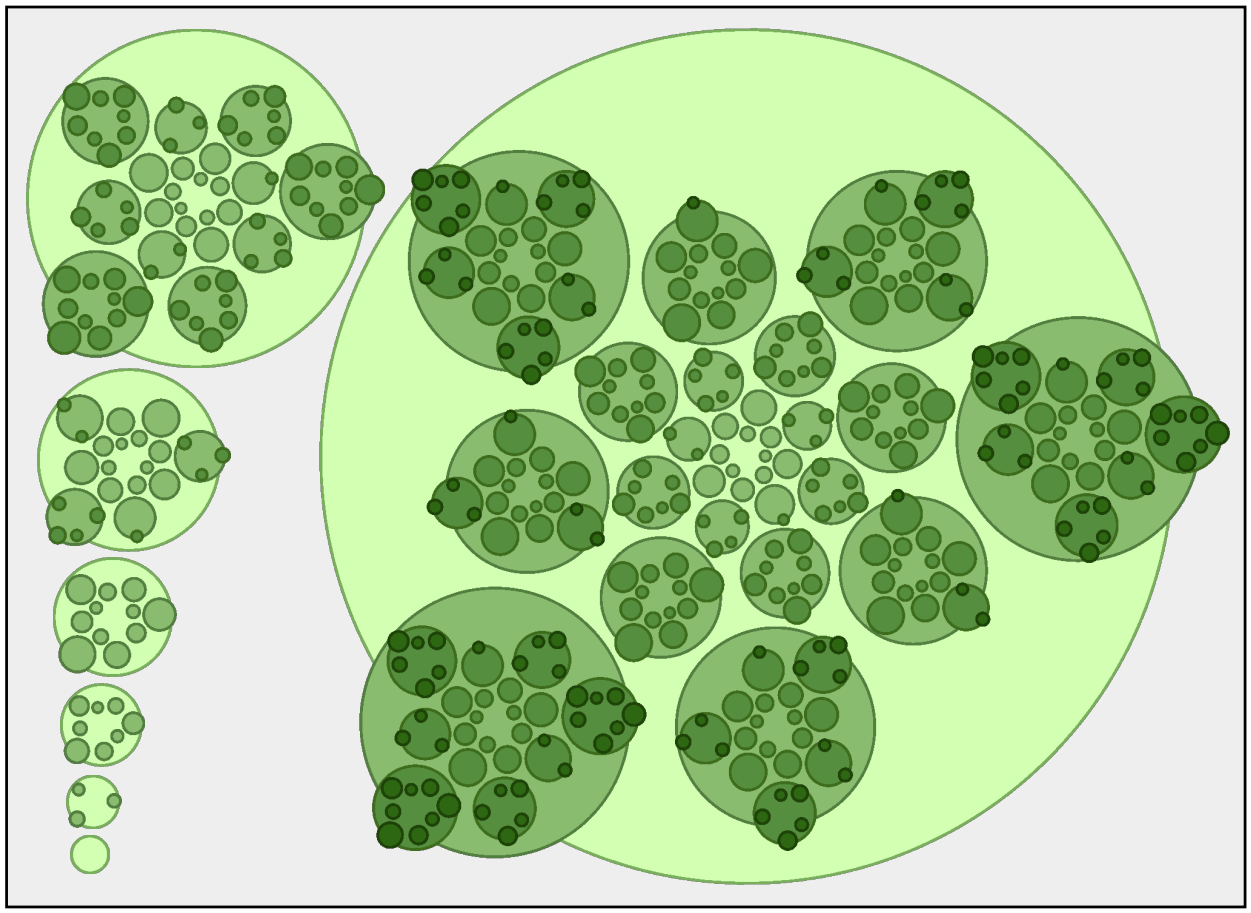

Figure 5.3: Development of a recursive, self-similar phyllotactic pattern on a uniformly growing flat meristem.

of an inflorescence. If the modeling objective is a static inflorescence form, this simulation need not be particularly accurate, as long as the final flower shapes and their spatial distribution are correct. However, if the objective is the animation of development, the spatio-temporal sequence of floret opening and changes in their position over time must reproduce biological reality.

5.4 Modeling hierarchical and recursive patterns

When modeling compound inflorescences, we assume that a pattern element created at level n may become a meristem and start producing next-level elements (meristems or primordia) upon reaching a threshold size. For example, Figure 5.3 shows a pattern generated by recursive segregation in a flat exponentially growing space. This figure also illustrates the relative orientation of pattern el-

ements at different levels: consistent with the Hofmeister rule the first element at level $n+1$ within the parent meristem at level n is positioned as far as possible from the centre of the "grand-parent" meristem at level $n-1$ (cf. [11]).

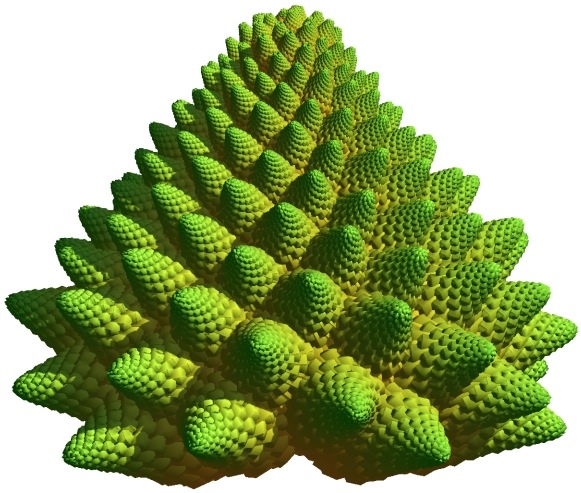

Figure 5.4: Two-level hierarchical phyllotaxis with conical meristems.

With $m > 1$ hierarchy levels, we assume that elements of phyllotactic pattern created at levels $n = 1,\ldots,m$ are next-level meristems, and only last-level meristems produce floret primordia. For example, Figure 5.4 shows a two-level pattern ($m = 2$).

Chapter 6

Determining floret type

6.1 Previous work

As discussed in Chapter 1, inflorescences may comprise florets of different types, thus exhibiting floral dimorphism. In computer graphics practice to date, dimorphism has been modeled in the context of heads: a user-specified number of florets at the rim of the head was assigned the fate of petal-like ray florets, and the remaining florets were modeled as disk florets [4; 5]. Inspired by Hirmer [58], Battjes and Prusinkiewicz [59] proposed a mathematical model of ray floret differentiation based on the packing of primordia on the rim of a head. They demonstrated that for the divergence angle $\varphi \approx 137.5^\circ$ the numbers of ray florets are numerically canalized to the Fibonacci series, as often observed in real heads. Neither model suffices, however, to capture dimorphism in compound inflorescences, in which the fate of each floret may depend on its position within the inflorescence in a relatively complex manner. Addressing this limitation, we propose an algorithm that applies to both simple and compound inflorescences. The algorithm is based on the biological hypothesis that the fate of florets depends on available space [16].

6.2 The model

Our algorithm operates under the assumption that the floral canopy is approximately planar. Each floret P in the set $\mathcal{S}$ of all florets is associated with a cone $\mathcal{K}(P)$ originating at P (Figure 6.1a). This cone is oriented away from the centre O of the highest-level head or umbellet to which P belongs. In the simplest case, the fate of floret P is determined by the presence or absence of another floret in the cone $\mathcal{K}(P)$ (Figure 6.1b,c). In general, this fate may also depend on the minimum distance $d_{min}(P)$ between P and a floret $Q \in \mathcal{K}(P)$. In order to find this distance, we first calculate the

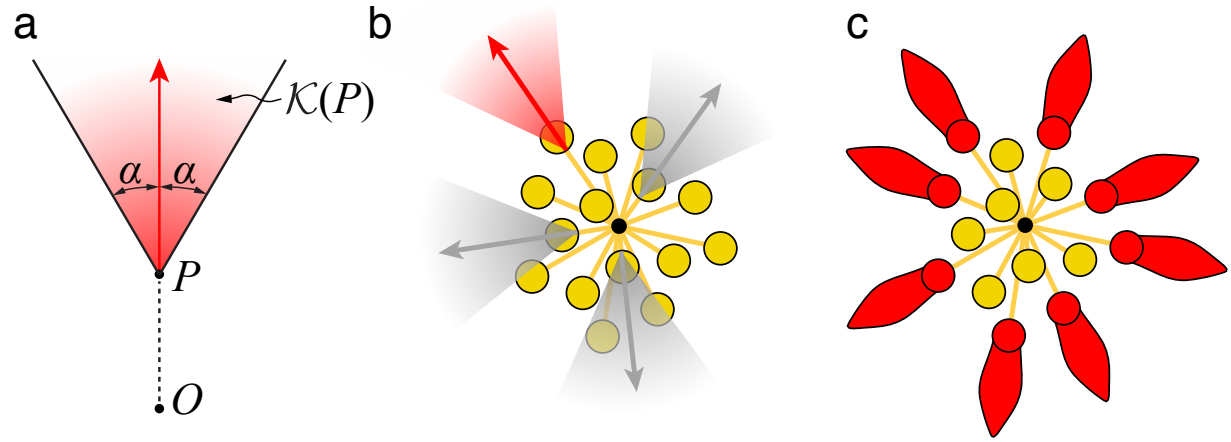

Figure 6.1: Principle of evaluating floret fate in dimorphic inflorescences. (a) Definition of cone $\mathcal{K}(P)$ associated with floret P in a head or umbellet with centre O. The cone has opening angle α. (b) The fate of a floret depends on other florets that may be within its cone. (c) The simplest case of floret differentiation: a floret becomes a ray floret if no other floret is within its cone.

distances $d(P,Q)$ between P and all florets $Q \in S$ using the formula:

$$d(P,Q) = \begin{cases} \|Q-P\| & \text{if } \widehat{PQ} \cdot \widehat{OP} < \cos\alpha \\ \infty & \text{otherwise} \end{cases} \tag{6.1}$$

Here $\widehat{AB}$ denotes the normalized vector from A to B, and α is the opening angle of cone $\mathcal{K}(P)$. The distance $d(P,Q)$ is thus set formally to infinity if floret Q lies outside this cone. The minimum distance $d_{min}(P)$ is then calculated as

$$d_{min}(P) = \min_{Q \in S} d(P,Q). \tag{6.2}$$

Application of this formula to simple flower heads is illustrated in Figures 6.2 and 6.3. All patterns were generated assuming divergence angle $\varphi = 137.5^\circ$ and primordia of fixed size. In both figures, a floret P becomes a disk floret upon $d_{min}(P) < \infty$. Figure 6.2 shows the dependence of the number of ray florets N on the total number of florets n, assuming a constant opening angle. We observe that the numbers of ray florets which emerge on heads of different sizes tend to be Fibonacci numbers. This result is consistent with the experimental data and mathematical analysis of a related (simpler) model [59].

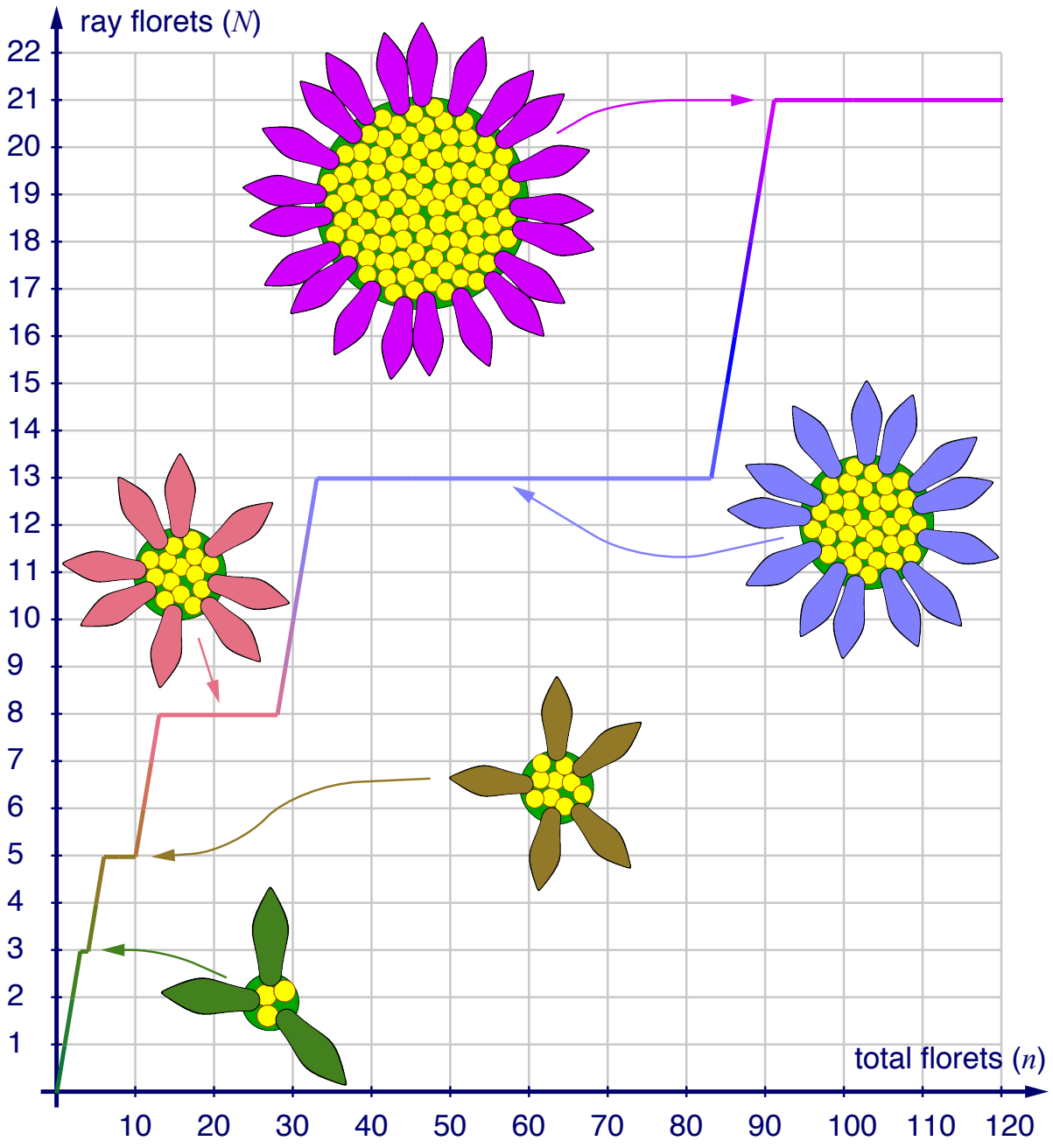

Figure 6.2: Dependence of the number of ray florets N on the total number of florets n for a constant cone opening angle $\alpha = 75°$.

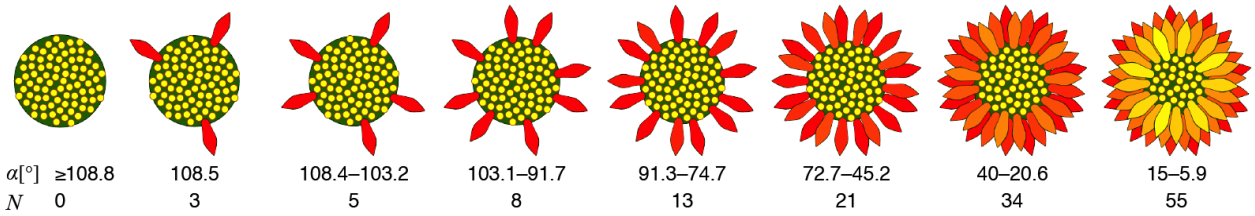

Figure 6.3: Dependence of the number of ray florets N on the cone opening angle α for a constant total number of florets $n = 82$.

A similar bias towards Fibonacci numbers occurs if the number of florets n is fixed and the cone opening angle α is changed instead (Figure 6.3). Fibonacci numbers occur for large ranges of α values, thus capturing the numerical canalization of ray florets.

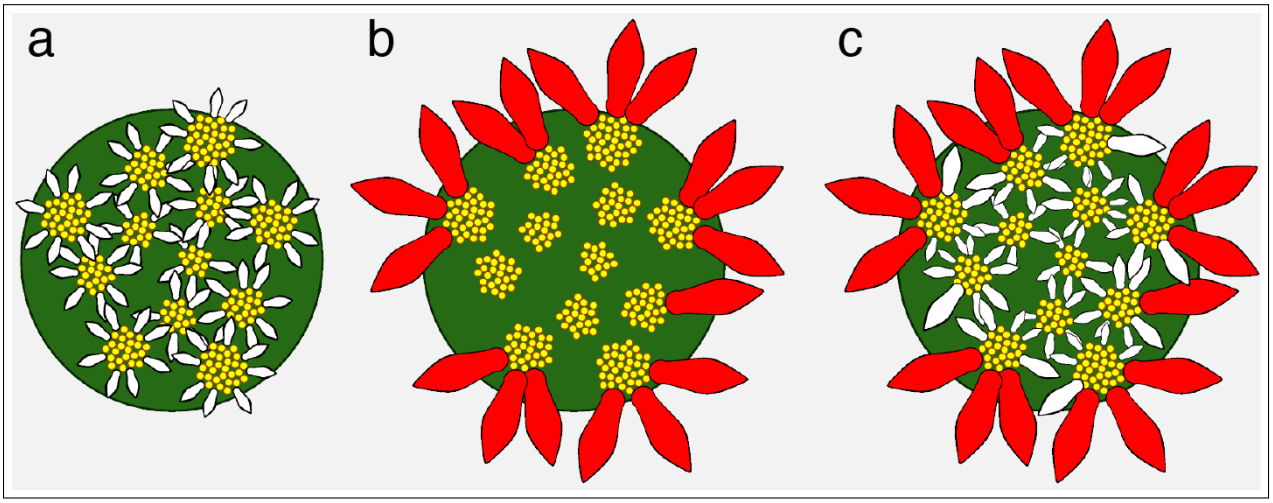

Figure 6.4: Differentiation of florets in hierarchically organized inflorescences. Ray florets emerge: (a) at the level of second-order heads, (b) at the level of the entire inflorescence, and (c) at both levels of inflorescence organization. In all cases $\alpha = 53°$.

Examples of floret fate determination in a hierarchically compound inflorescence are shown in Figure 6.4. In all cases, the fate of florets is controlled by distance thresholds. In Figure 6.4a, threshold Th_a is larger than the distance between florets within a second-order head, but smaller than the distance between these heads. A floret P becomes a ray floret if and only if $d_{min}(P) > Th_a$.

In this case, ray florets differentiate independently in each second-order head. In Figure 6.4b, floret P becomes a ray floret if and only if $d_{min}(P) > Th_b$, where Th_b is larger than the distance between second-order heads. In this case, the differentiation of ray florets is dominated by their position within the entire inflorescence. The model in Figure 6.4c combines both mechanisms. Large ray florets emerge when $d_{min}(P) > Th_b$, and smaller ones when $Th_a < d_{min}(P) \leq Th_b$. In the latter case, the size of the enlarged petals is proportional to the amount of available space, $d_{min}(P)$, resulting in a continuum of florets sizes.

Chapter 7

Branching structure

7.1 Previous work

In inflorescences in which florets are supported by a branching structure (as opposed to a receptacle), this structure is an inherent component of the model. Computational modeling of branching inflorescence structures was an early biological [20] and computer graphics [3; 4] application of L-systems. The use of positional information [8] facilitated specification of inflorescences by providing the modeler with direct control over the extent and density of branches. Sketch-based methods [9; 10; 28] facilitated modeling further by introducing a more intuitive user interface. With all these approaches, the distribution of flowers in space was determined by the underlying branching structure. Such branching-first models work well for many inflorescences (e.g., the lilac in Figure 4.10), but do not easily capture inflorescences with florets arranged into a smooth, planar canopy. To model these inflorescences, we employ the florets-first paradigm (Chapter 1 and Figure 1.4), in which a spatio-temporal floret distribution is generated first and provides input for synthesizing the branching structure.

Our method is inspired by the algorithm for generating vascular patterns in leaves and modeling trees proposed by Rodkaew et al. [27] (see also [60]). Rodkaew's algorithm operates centripetally, i.e., from the outside in. For example, in the case of trees, it begins by distributing leaves at the periphery of the tree canopy, initiates branches at these locations, and gradually extends them towards the trunk. Rodkaew et al. described this process in terms of particles that trace branches by moving through space while being attracted to the base of the tree and to their nearest neighbor. Particles that approach each other merge, forming branching points. The choice of Rodkaew's algorithm as a basis for modeling the architecture of flat-topped branching inflorescences was motivated by two factors:

i) At the macroscopic level, Rodkaew's algorithm is a stepping stone for generating three-dimensional branching structures that support florets in predefined positions. In research preceding this thesis, I used Rodkaew's algorithm to generate a multitude of branching structures from predefined positions of the branch tips, down to the base (see Figure 7.1a).

ii) At the microscopic level it can be viewed as a geometric analog of auxin canalization, which defines the vascular system of young inflorescences. It is plausible that this vasculature defines the branching structure of inflorescences in reality (Chapter 1).

Further motivating this supposition, I extended Rodkaew's algorithm to produce various branching structures (see Figure 7.1b) otherwise unattainable by Rodkaew's algorithm. As well, I further extended Rodkaew's algorithm to model the vascular structures in various plants, e.g. geometrically modeling the vascular development of the grass, *Brachypodium*, and the shoot in *Arabidopsis* (see Figure 7.1c,d) [22; 61]. These extensions were motivated by observations that differentiation of new primordia and vascular strands may not take place at one time. Rodkaew's algorithm does not account for this possibility, i.e. particles introduced at different times cannot merge with already established branches. This further necessitated a series of extensions to Rodkaew's algorithm, which allows for emergent vascular strands to merge with already established veins.

7.2 Extended Rodkaew algorithm

To model inflorescences, we extended Rodkaew's algorithm in several directions.

Attraction to branches. According to the original formulation of the algorithm, particles interact with each other: they do not interact with the emerging branching structure. Consequently, a moving particle may run unnaturally close to a branch formed earlier and/or cross it. To avoid this artifact, we consider not only the moving particles, but also the entire structure developed so far (i.e., past positions of all particles) as candidate attracting points (Figure 7.2a). A moving particle is thus attracted to its closest neighbour irrespective of whether it is another moving particle or a point in the existing branching structure. Furthermore, we consider the previous direction of par-

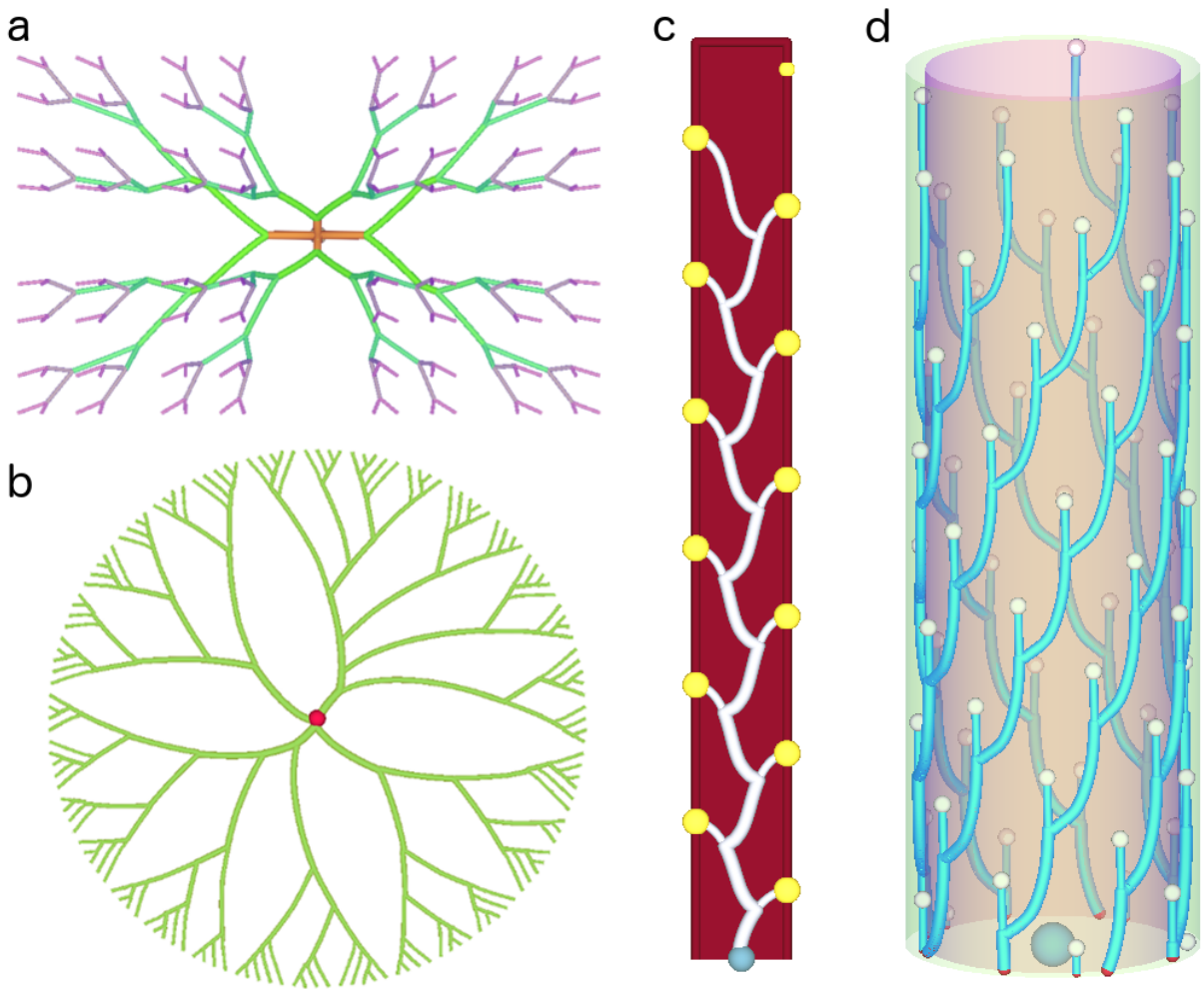

Figure 7.1: Generating branching patterns from the branch tips down. (a) A branching structure generated from Rodkeaw's algorithm. The initial positions for the particles (purple tips of the tree) come from the positions of tips in a H-tree. (b) Rodkaew's algorithm, with the extension of a function weighting the attraction between particles, applied to positions distributed on the circumference of a circle. (c) Model of vascular development in the grass, *Brachypodium* (see [22]), modeled using my extensions of Rodkaew's algorithm, allowing for new primorida (yellow) to be introduced over time, and new vascular strands to merge with previously established veins (white). (d) Model of vascular development in shoot of the plant, *Arabidopsis* (see [61]), modeled using same extensions to Rodkaew's algorithm. A spiral phyllotaxis introduces new primordia (white) sequentially upward, on the surface of a cylinder; and the vascular strands emerging from the primorida connect to previously established veins.

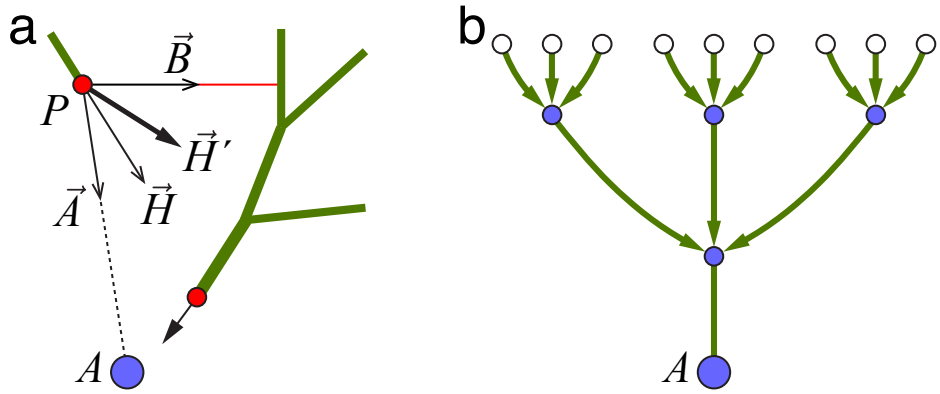

Figure 7.2: Elements of the extended Rodkaew algorithm. (a) Particle P extends a branch segment in direction $\boldsymbol{H}'$, which is a weighted average of the previous segment direction $\boldsymbol{H}$, vector $\boldsymbol{A}$ pointing toward the inflorescence base A, and vector $\boldsymbol{B}$ pointing towards the nearest branch or particle. (b) The umbellets of a compound umbel are generated separately from the main umbel. Both the positions of the florets (empty circles) and the umbellet and umbel centers (small blue circles) are determined by a phyllotaxis model.

ticle motion $\boldsymbol{H}$ as a factor influencing the next direction $\boldsymbol{H}'$. Assuming that all vectors defined in Figure 7.2a have unit length, direction $\boldsymbol{H}'$ is calculated in each simulation step as

$$\boldsymbol{H}' = \frac{1}{w_a + w_b + w_h}(w_a\boldsymbol{A} + w_b\boldsymbol{B} + w_h\boldsymbol{H}). \tag{7.1}$$

where all vectors are defined as in Figure 7.2a and have unit length.

Weights w_a, w_b and w_h provide a degree of control over generated forms. With $w_a > 0$ and $w_b = 0$, all particles converge directly on the inflorescence base, creating an umbel (Figure 7.3a). Increasing w_b results in branching structures (Figure 7.3c,e). Making w_b a function of the distance d to the closest neighbor provides a means for controlling the shape of branches near the branching points, producing smaller or larger branching angles (Figure 7.4). Increasing weight $w_h > 0$ prevents biologically improbable [62] highly non-planar branch arrangements at the branching points. It also reduces the curvature of branches, making them more smooth.

To accelerate spatial, nearest neighbour queries, a similar implementation to that described by Teschner et. al [43] is used. Each tree branching structure in the scene is paired with a particle at its root, and is extended as the particle moves (Figure 7.2). Each particle-branch pair are assigned

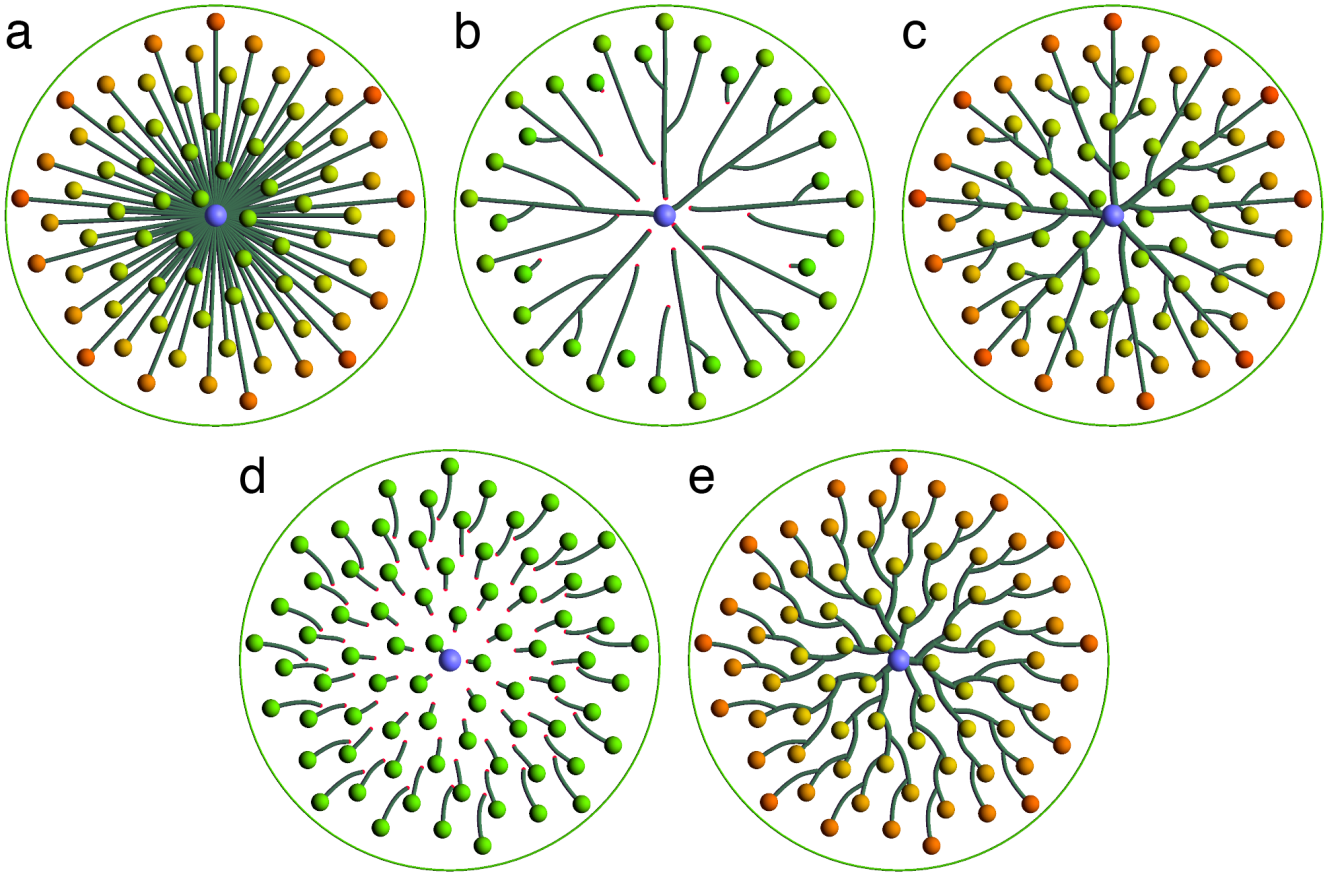

Figure 7.3: Examples of branching structures generated using the extended Rodkaew algorithm. (a) All particles are attracted to the inflorescence base. (b) Early and (c) final stages of pattern formation with primordia produced slowly. (d) Early and (e) final stages of pattern formation with primordia produced quickly. Colors of primordia indicate their age, as in Figure 5.2.

a unique ID number, ID_{branch}. As well, each tree node in the branching structure is assigned a unique ID among all the tree nodes in the scene, ID_{node}. The spatial domain is divided into voxels, each storing a list of unique ID numbers pairs, (ID_{node},ID_{branch}), from those particles, tree nodes and branches confined within or passing through the voxel. When calculating the direction to the nearest particle or branch from a given particle, the list of ID pairs within the voxel containing the particle are iterated over. A pair with the same ID_{branch} as the particle is ignored, as the tree node associated with it already belongs to the same branching structure as the particle. For all other pairs, the vector displacements between the particle and the tree nodes associated with each ID_{node} is computed. The voxels radially nearest to voxel containing the particle are also searched. The

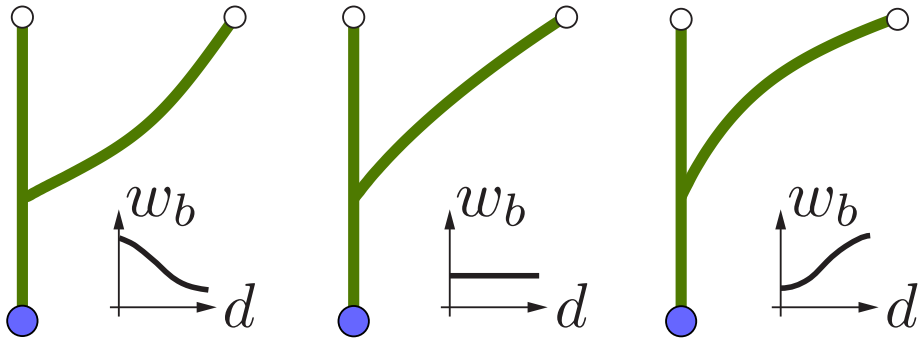

Figure 7.4: The dependence of branching point configuration on function $w_b(d)$.

ID_{node} with the minimal vector displacement is the nearest neighbour to the particle. This allows for quick nearest neighbour querying regardless of the various tree topologies in the forest of trees, which is liable to change as particles merge with other established branches. Due to the emergent behaviour of the algorithm, particles may move in close proximity to tree nodes associated with its own tree, i.e. they share the same ID_{branch}. These nodes are ignored in querying the nearest neighbour and merging processes. Otherwise a particle would be attracted to and merge with tree nodes of its own branching structure, causing undesirable loops in the branching structure (see Figure 7.5a). Therefore, when branches merge, i.e. when a particle encroaches close enough to an established branch, and "grafts" its tree onto the branch, the unique ID of the branch, ID_{branch}, must be propagated up the particle's tree (see Figure 7.5b,c).

Particle initialization. In the original formulation, the initial positions of the particles are random, and all particles begin their motion simultaneously. To model inflorescences, we assume that initial particle positions and the times of their creation are determined by the dynamic phyllotactic model (Chapter 5). The timing of floret emergence has a significant impact on the resulting branching structure. With primordia created at large time intervals, during which particles can travel relatively long distances compared to the distances between primordia, the emerging branches have a monopodial architecture characterized by clearly delineated axes (Figure 7.3b,c). In contrast, if primordia are created quickly, branches have a sympodial architecture characterized by sequences of short segments positioned laterally with respect to their parents (Figure 7.3d,e).

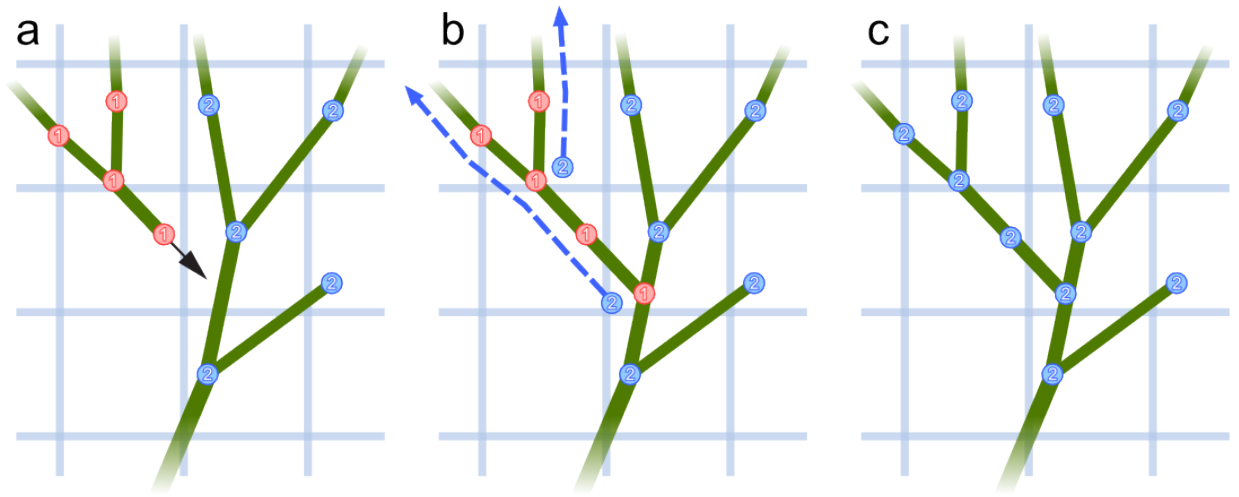

Figure 7.5: A younger branch (*red*) merging with an established branch (*blue*). (a) The growing tip of the *red* branch (indicated by the arrow) searches its neighbouring voxels, and tree nodes therein (via their ID pairs (ID_{node},ID_{branch})), for the nearest potential neighbour to move towards, and possibly merge. Tree nodes belonging to the *red* branch ($ID_{branch} \equiv 1$) are ignored, avoiding loops. (b) The growing tip's new position is within the *merging distance* of the nearest neighbouring branch (*blue*), so it is grafted onto the *blue* branch. The *red* branch is now a constituent of the *blue* branch, thus the *blue* $ID_{branch} \equiv 2$ must be propagated up through the tree. (c) With the *blue* ID_{branch} propagated throughout its branches, the (potentially) growing tip of the *blue* branch may never merge with itself.

Growth. We model growth of the branching inflorescence structure as a gradual free-form deformation [63] of the space including branches and moving particles. This deformation is coordinated with the dynamic phyllotactic patterning of floral primordia and the canopy they produce. In the models implemented so far, we assumed uniform or allometric expansion of this space. In the allometric case, this space expands at different, size-dependent rates along different axes of the coordinate system, resulting in changes to the branching angles and overall proportions of the inflorescence over time. These deformations do not affect branch width, which is determined independently. An example of the development of a branching structure supporting primordia in a recursive phyllotactic pattern is shown in Figure 7.6.

Distinct hierarchy levels. Some inflorescences (e.g. corymbs of heads and compound umbels) are organized into distinct hierarchical levels. We generate branching patterns of these inflorescences by considering each level of the hierarchy separately (Figure 7.2b).

Branch width. We assume that terminal branch segments have the same diameter and determine the diameter of interior segments using the formula $d^p = d_1^p + \ldots + d_m^p$. Here d is the diameter of the branch below the branching point (closer to the inflorescence base), and $d_1 \ldots d_m$ are the diameters of the branches supported at this point. Power p controls the rate at which branch widths accumulate towards the inflorescence base. Variants and extensions of this formula have been used in different contexts [4; 64; 65]; a special case, with $p = 2$, was proposed by Leonardo da Vinci to capture the relation between branch diameters in trees. In the modeling of inflorescences, smaller changes in diameter obtained for $p > 2$ usually lead to more realistic results.

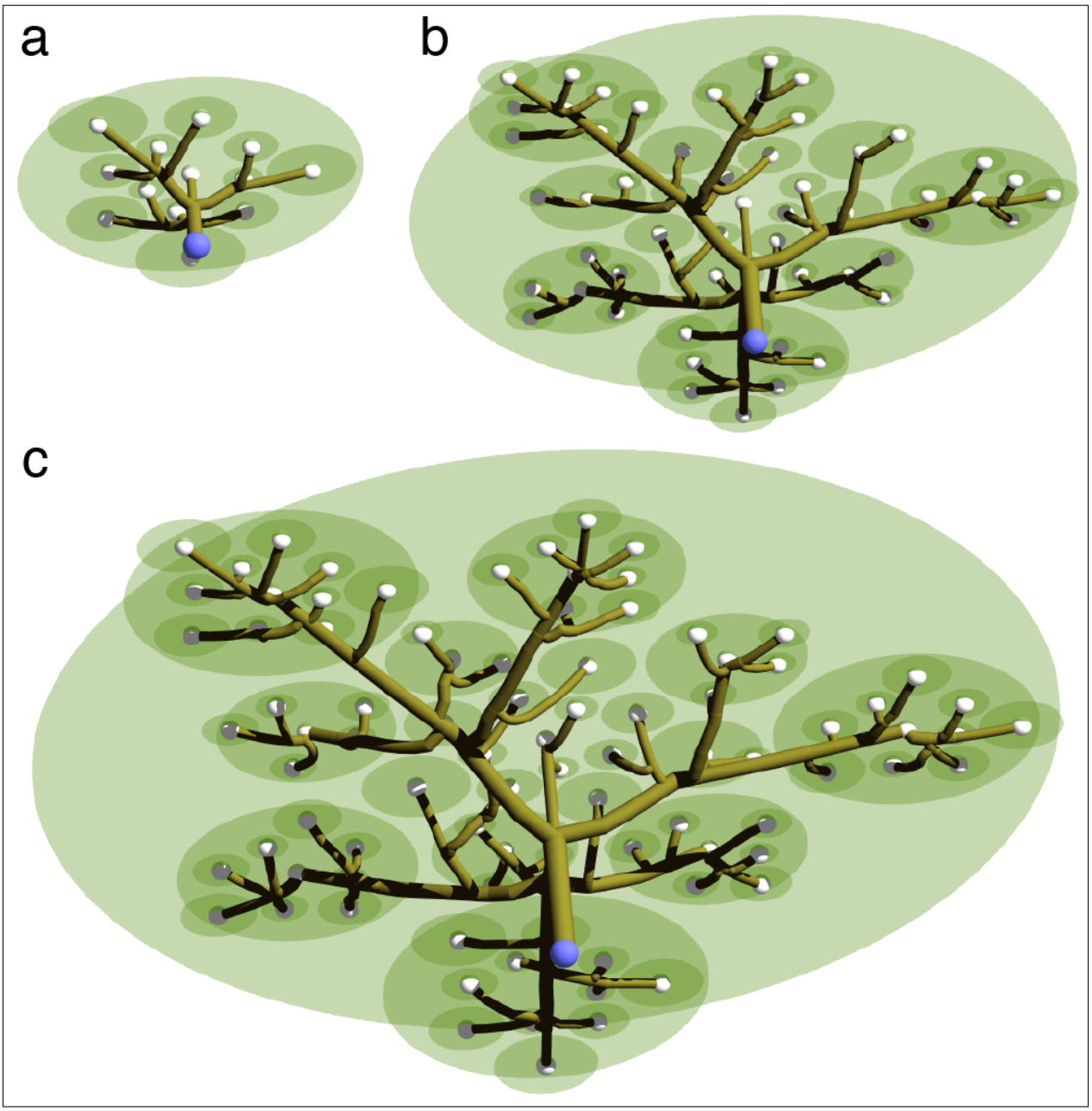

Figure 7.6: A growing branching structure generated using the extended Rodkaew algorithm. Florets (white spheres) are formed in a recursive phyllotactic pattern and emerge at different points in time according to the model in Figure 5.3. The emergent branching structure supports all florets in the same plane, as needed for modeling corymbs.

Chapter 8

Implementation

The presented algorithms are implemented as a suite of programs that communicate via shared files. The flow of information is shown in Figure 8.1, and the implementation and file specifications are as follows.

The modeling software of *a)* through *e)* were devised within the Virtual Laboratory (VLab) [30], which provides an environment for exploratory programming and experimentation with simulation models, including graphical editors of functions and contours, and viewers for displaying intermediate results.

a) Interactive flower editing. The Floret Editor (Chapter 2) is developed as a single vlab object, taking advantage of graphical and interactive capabilities of the lpfg simulator (see Figure 2.1). The output of this editor is a *floret* text file specifying the vertices of the control mesh, readable by the floret animator.

b) Posture interpolation. The floret animator reads a sequence of key poses from floret files (often, no more than two), and interpolates between them, displaying the development for visual inspection. The progress of time is regulated by a graphically-specified *growth* function, giving different developmental characteristics, e.g. linear vs. sinusoidal interpolation.

c) Polygonization. Within the floret animator, the sequence of interpolated B-spline florets can be previewed as polygonized triangle meshes. The polygonization of the B-spline parameter space is controlled by two graphically-controlled functions (see Figure 2.4). These functions specify where vertices are positioned within the B-spline *uv*-parameter space. A Delauny triangulation is then computed over the vertices, providing a consistent mesh topology throughout the floret development. The developmental sequence is output as a sequence of OBJ files, encoding a sampling of the floret's development, to be used as an exemplar of a developing floret instantiated within an inflorescence.

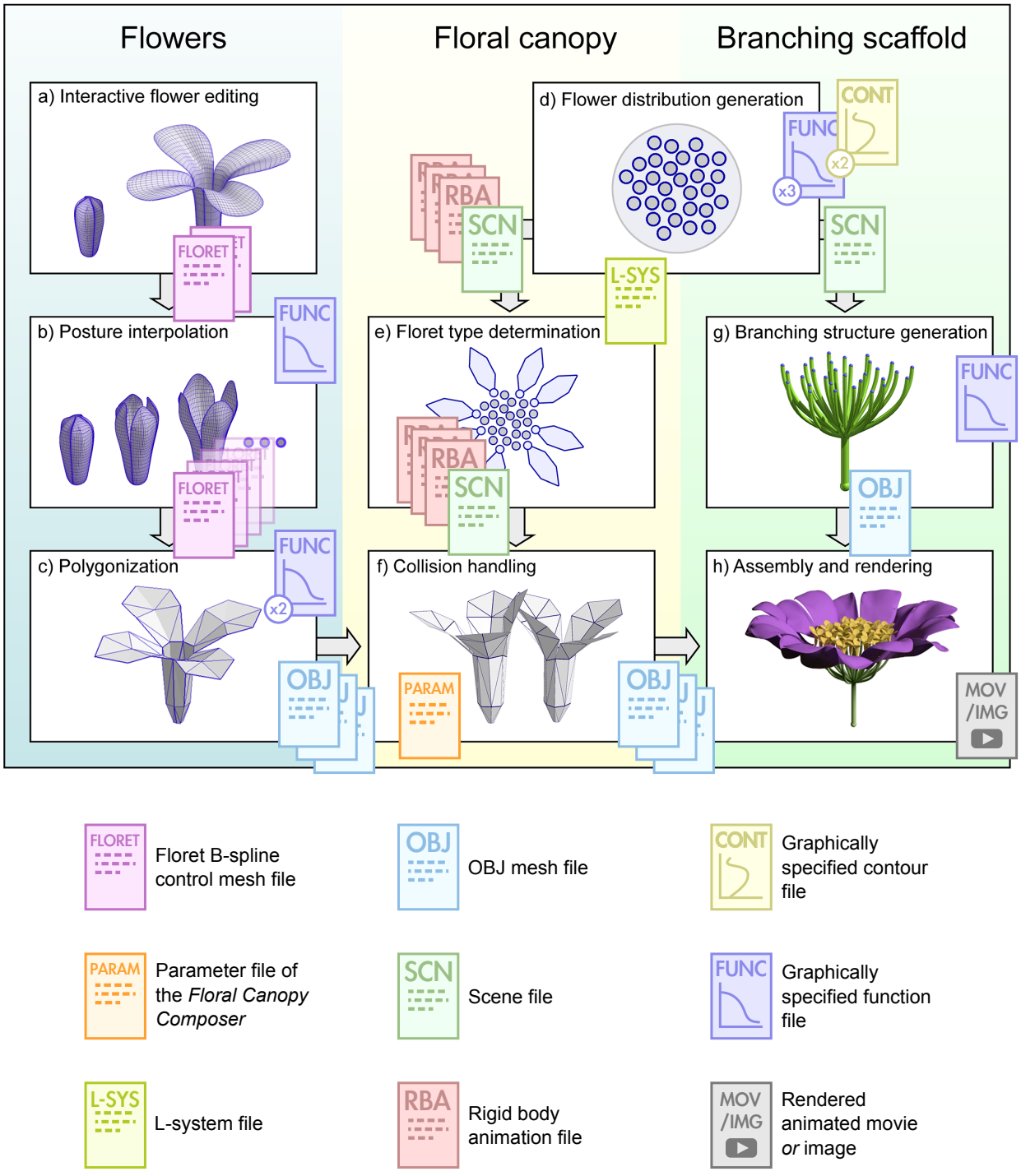

Figure 8.1: Key components and file usage of the modeling method, and the associated information flow via transferred files.

d) Flower distribution generation and *e) Floret type determination*. The dynamic phyllotaxis, defining the type, placement, orientation, and growth of the florets within the inflorescence over its development, is procedurally generated within VLab. Modeling phyllotaxis within VLab affords a multitude of possible (spiral) phyllotactic patterns, e.g. simple, hierarchical, static, dynamic. Static phyllotaxis models require one graphically-controlled countour file, and one function file to specify the shape of the receptacle and sizes of the florets, respectively. Dynamic phyllotaxis models were procedurally generated with at most three graphically-specified functions that define the growth of the floret primordia and the inner and outer regions of the supporting surface (Chapter 5). In addition, some models (e.g. Figures 9.4, 9.2, 9.3) require two profile curves to specify the initial and final shape of the supporting surface. Although viewed as two separated programs, the models for flower distribution generation and the floret type determination are typically written as a single L-system, requiring a small number (about 5 per program) of numerical parameters to generate a phyllotactic pattern with dimorphism. The amount of input increases in hierarchical structures, where phyllotactic patterns at different levels of the hierarchy may require separate definition. The output of phyllotactic generation depends on the inflorescence to be modeled. If the phyllotactic pattern of the inflorescence model is static (i.e. for the purposes of a single final image rendering, or an animation movie of opening florets with an unchanging phyllotaxis), then only the final positions, scales and orientations of the floret mesh/animation types (as produced in step *c*) are required. This is described in a *scene* (SCN) file (see Appendix C.2), enumerating the type, position, scale, and orientation of each floret mesh/animation exemplars to be instantiated in the scene. However, if the phyllotatic pattern of the inflorescence model is dynamic (i.e. for the purpose of a fully developmental inflorescence animation), then each floret's position, scale and orientation over the development of the supporting surface must be saved. To this end, a sequence of *rigid body aninmation* (RBA) files (see Appendix C.3) are output, each listing unique floret IDs and their current rigid body positions, scales[1] and orientations at a given moment. The sequence

[1]Scaling is not a rigid body mode by definition, but I have found utility in controlling the scale of florets over an animation.

together describes the changes of each floret over the inflorescences development. The type of florets to be instantiated are read from a SCN file accompanying the sequence of RBA files. The order of floret placements in the SCN file provides the unique identification carried forth in the RBA files (see Figure 8.2). This reasoning for this design choice is twofold: 1) for conciseness of the RBA files, which do gain nothing from storing the type of each floret across the multiple files, and 2) allows for hierarchical relations of dynamic configurations, i.e. a SCN file instantiates a rigid body animation of an inflorescence described in another SCN file. Recursive relations between scene files and rigid body animations are possible in this framework, thus care must be taken by the modeler to ensure this not happen.

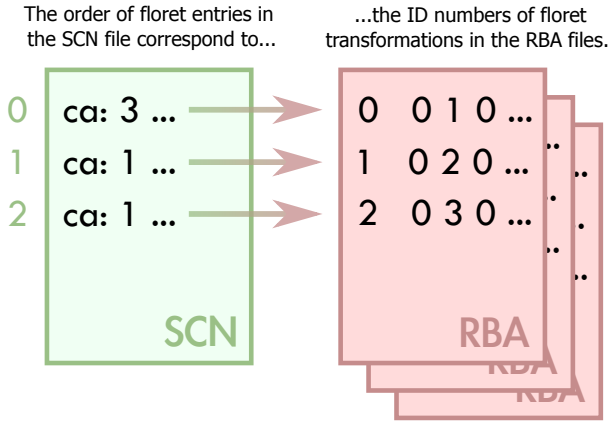

Figure 8.2: Connection between scene (SCN) files and rigid body animation (RBA) files. The first (or *zeroeth*) entry of the SCN file is a floret of type 3. The positions, scales, and orientations of this floret throughout the development of the inflorescence are defined by the entries in the RBA files with ID 0.

f) Collision handling. Models are assembled by a C++ program, entitled *Floral Canopy Composer* (FCComposer), that instantiates florets of the type defined by the dimorphism model at the locations and orientations defined by the phyllotaxis model, simulates inflorescence development while resolving collisions using Position Based Dynamics (PBD) (Chapter 3, Chapter 4), and produces a temporal sequence of meshes that represents the developing floral canopy. It provides a graphical environment for simulating the development of inflorescences with collisions resolution between the floret's triangle geometry. The FFComposer is configured by a single *parameter* file

Algorithm 1: Floral Canopy Composer

```
1  Loop
2      forall the active animations a_i do
3          applyAnimation(a_i) ;                    /* updating internal constraints */
4      end
5      for solverIterations do
6          forall the internal constraints c_i do
7              solveInternalConstraints(c_i);              /* reposition vertices */
8          end
9      end
10     forall the vertices v_i do
11         generateVertexTriangleCollisionConstraint(v_i)
12     end
13     forall the edges e_i do
14         generateEdgeEdgeCollisionConstraint(e_i)
15     end
16     for collisionSolverIterations do
17         forall the collision constraints c_i do
18             solveCollisionConstraints(c_i);         /* reposition vertices */
19         end
20     end
21     forall the vertices v_i do update to new positions;
22     incrementFrameCounter()
23 EndLoop
```

(see Appendix C). Floret design files (single OBJ or animation of development as a sequence of OBJs) to be used in the inflorescence are listed, each assigned a unique ID number. Multiple phyllotactic pattern files (Scene files with optional Rigid body animation files) may be listed. The FCComposer will compose all elements of the scene (e.g. florets, rigid body animations) together into a single simulation. This allows for the separation of floret and phyllotaxis design in modeling of inflorescences, provided the floret placement IDs listed in SCN files match those listed in the FCComposer parameter file. This separation decouples the geometric descriptions of the phyllotaxis from the triangle geometry of the florets, meaning floret designs may be easily refined, or swapped out all together, for a precomputed phyllotaxis model. Internally, FCComposer reads in these various files, and processes them into appropriate data structures so that they may be instanced in the scene. For example, as described in Chapter 3, the sequence of OBJs in a floret

animation must be processed to extract a sequence of constraint system *rest* values from the vertex positions and triangle topology. Then when a scene requires an instantiation of the floret, the FCComposer adds the triangle mesh of the floret's first pose and adds constraints regulating the vertices of the triangle mesh (e.g. distance and bending constraints) to the set of global constraints. As the simulation runs, the rest values of the constraints regulating the florets triangles are read from a data structure in memory storing these developing values, and updates the constraints rest values to match.

The collision detection system has many components, as described in Chapter 4, requiring a spatial data structure and cubic root finding routine for accelerated collision detection. Algorithm 1 shows the (high-level) collision-handling process in the FCComposer. The outputs of the FCComposer are OBJ files (single or animation sequence), each encoding the triangle mesh of the inflorescence's floral components as they develop over a number of simulation steps (read in via the parameter file).

g) Branching structure. The branching structure generator program was developed in C++ and adjusted to become a VLab object thereafter. The essence of the model by Rodkaew et al. [27] was a generalized particle system, with each particle leaving a trail as they moved through the domain space, and the trails composing an emergent branching structure. The branching structure generator program is designed similarly, with the notable exception: particles do not leave individual trails, and instead function as the root of an emergent tree structure. In doing so, the program maintains a forest of spatial tree data structures (n-ary tree data structure with tree node positions in $\mathbb{R}^3$) associated with the moving particles. The reason for maintaining a tree for each particle is that initially each particle is attracted to, and capable of merging with branches belong to any other particle's tree. This is in sharp contrast to model of Rodkaew et al. [27], where particles could only be attracted to, and merge with other moving particles. The particle positions are extracted from an input SCN file, listing the floret positions and orientations. The tip positions (leaves) of the emergent branching structure coincide with the floret placements, as the floral canopy and branching

structure are generated separately, and composed together only thereafter. A graphically-specified function is required to provide the weighting function for nearest-neighbour attraction (see Figure 7.4). The output of this step is a static branching structure in the form of an OBJ file to be used in composing the final inflorescence. Animated inflorescences with developing branching structures are not currently obtainable with this model.

h) Assembly and rendering. I have assembled and rendered the final models using Blender[2]. To improve rendering, Blender's modeling tools were employed to further subdivide petals and slightly extrude them to allow for subsurface light scattering. As well, scanned textures and environment maps from the sIBL Archive[3] have been incorporated to enhance realism. Each inflorescence scene was rendered with Blender's path tracing engine, *Cycles*, using its graphical node-based material editor to combine textures maps (e.g. specular, normal, displacement) and artistically design surface and subsurface material characteristics. Rendering was distributed among 25 workstations using Blenders network rendering interface. To increase rendering time efficiency, multiple workstations were assigned the same frame to render, but with a different *seed* value for the path-tracing integrator noise patterns [4]. Then, these multiple low quality renderings were combined together to get a single frame that approximates a higher quality path-traced image, i.e.10 renders of a single frame at 100 samples per-pixel, each with a different noise pattern seed, can approximate an image with 1000 sample per-pixel when blended together into a single image. Blending many low quality frames into a final frame allowed me to *progressively* render animations with higher quality. Low quality test animations could be quickly rendered and inspected, and if they were acceptable, further render passes would be computed for incrementally higher quality. The blending functionality was automated by custom python scripts that I wrote to invoke the command-line tool *convert* from ImageMagick[5]. I have found a Blender plug-in[6] providing the same functionality, and is similar in its implementation to my own scripts.

[2] `https://www.blender.org/`
[3] `http://www.hdrlabs.com/`
[4] `https://www.blender.org/manual/render/cycles/settings/integrator.html`
[5] http://www.imagemagick.org/script/index.php
[6] http://adaptivesamples.com/2013/07/22/progressive-animation-render-addon-and-image-stacking/

Chapter 9

Results

Figure 9.1: Variant of dahlia model with florets that are tightly packed.

Collision detection is the central feature of the dahlia model in Figure 9.2. This particular variety has a ball-shaped inflorescence populated almost exclusively by incurved, almost tubular ray florets that touch each other and accommodate their shape to the presence of their neighbors. The distribution of florets in the model was generated using a phyllotaxis model operating on an approximately spherical surface. Figure 9.1 shows a variant of the Dahlia model with enlarged petals, showcasing the dense packed configurations producible by the method.

Figure 9.2: Photograph and model of dahlia ‘Jomanda’. Photograph courtesy of the Victoria Dahlia Society, adapted under fair use.

Figure 9.3: Photograph and model of *Gaillardia x grandiflora* cultivar 'Oranges and Lemons'. Photograph licensed under CC BY_SA 3.0.

Figure 9.4: Selected frames from an animation of *Gaillardia* growth, especially noting the dramatic, yet stable change in size due to the dynamic phyllotaxis.

Gaillardia (Figure 9.3) is an example of an inflorescence with strongly dimorphic florets. In the variety shown, ray florets are tubular. The emergent number of ray florets in the model (21) is consistent with their numerical canalization in real *Gaillardia* heads, where Fibonacci numbers commonly occur. The phyllotactic patterning of floret primordia was modeled using the fractionation scheme (Figure 5.2a). In addition to disk and ray florets, this model incorporates involucral bracts: specialized leaves that surround the head and protect it in its early development. Hairs on all surfaces have been modeled using a built-in Blender function. Figure 9.4 shows a simulation of head development, with the receptacle changing size and shape. All three organs types were animated using intrinsic interpolation and were subject to collisions.

Figure 9.5: Photograph and model of a sunflower. Photograph courtesy of `http://www.pixabay.com` under CC0 Public Domain License.

Figure 9.6: The model of a sunflower from another perspective.

The sunflower (Figure 9.5, top) is another example of an inflorescence with strongly dimorphic florets. As in the case of gaillardia, the emergent number of ray florets in the model (34) is a Fibonacci number. Ray florets are bilaterally symmetric. Three large fused petals point outward, giving the appearance of a single large petal. The remaining two petals are highly reduced. We have only included the conspicuous large petals in the model. Densely packed disk florets have five-fold dihedral symmetry, with petals fused into a tubular corolla almost up to the petal tips. As the flowers open, sexual organs - stamens, then pistils - raise above the level of the petals [66]. We modeled these organs summarily as cylinders growing along the central axis of the florets. The florets open in a centripetal sequence, so that outer florets are more advanced in their development than those in more central positions. This phase effect has also been captured in the model (Figures 9.5, bottom and 9.6)

Figure 9.7: Photograph and model of *Dyssodia decipiens*. Photograph courtesy of Regine Classen-Bockhoff.

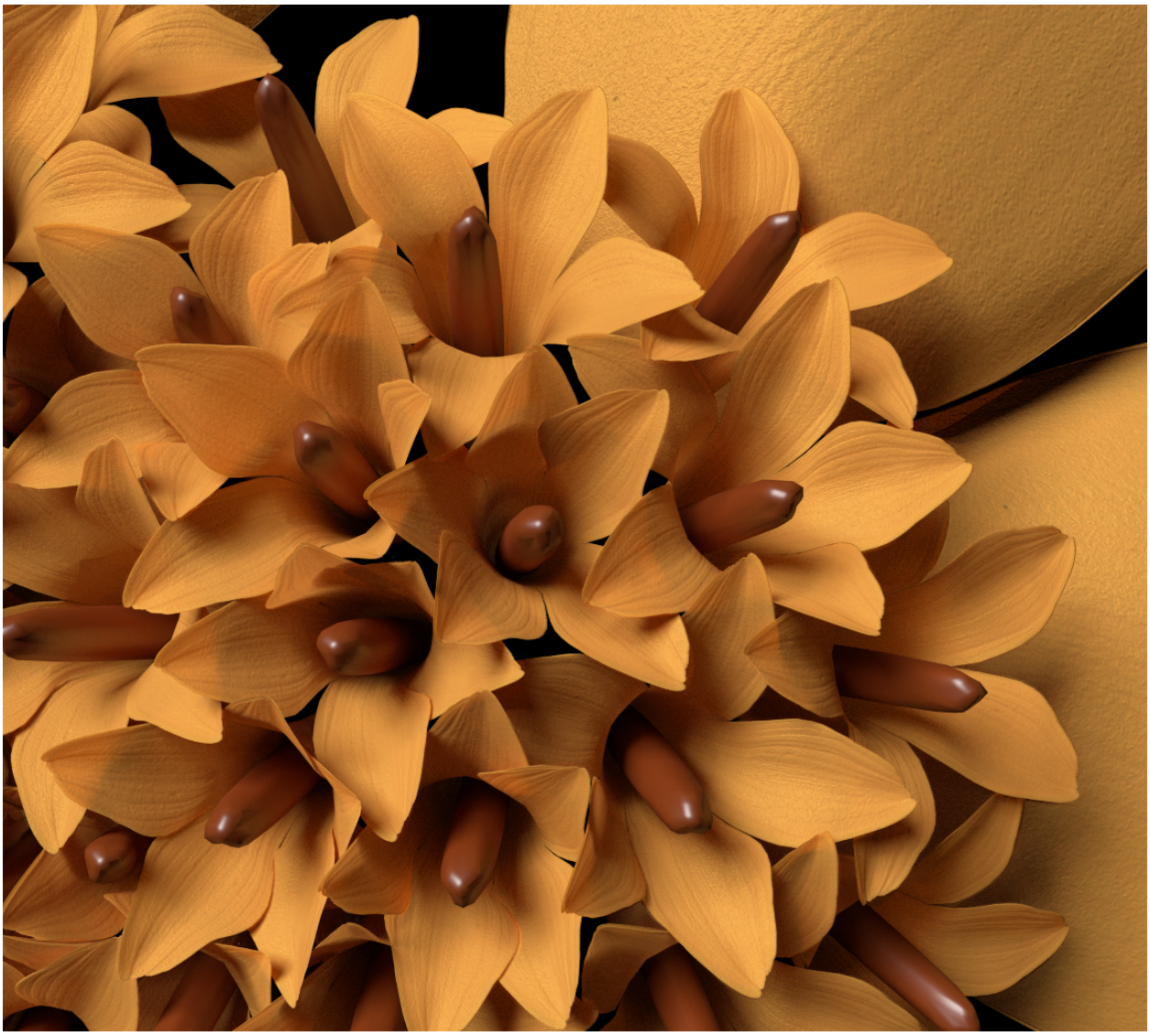

Figure 9.8: A magnified view of *Dyssodia*'s florets.

The next three models illustrate different cases of floral dimorphism. The inflorescence of *Dyssodia* (Figure 9.7) consists of a central head surrounded by 5 other heads. Florets with enlarged petals emerge at the level of the entire inflorescence as in Figure 6.4b [16]. Incidentally, the modification of disk floret shapes due to collisions is particularly conspicuous in this model (Figure 9.8).

Figure 9.9: Photograph and model of a yarrow. Photograph courtesy of Frank L. Hoffman, `http://www.all-creatures.org`.

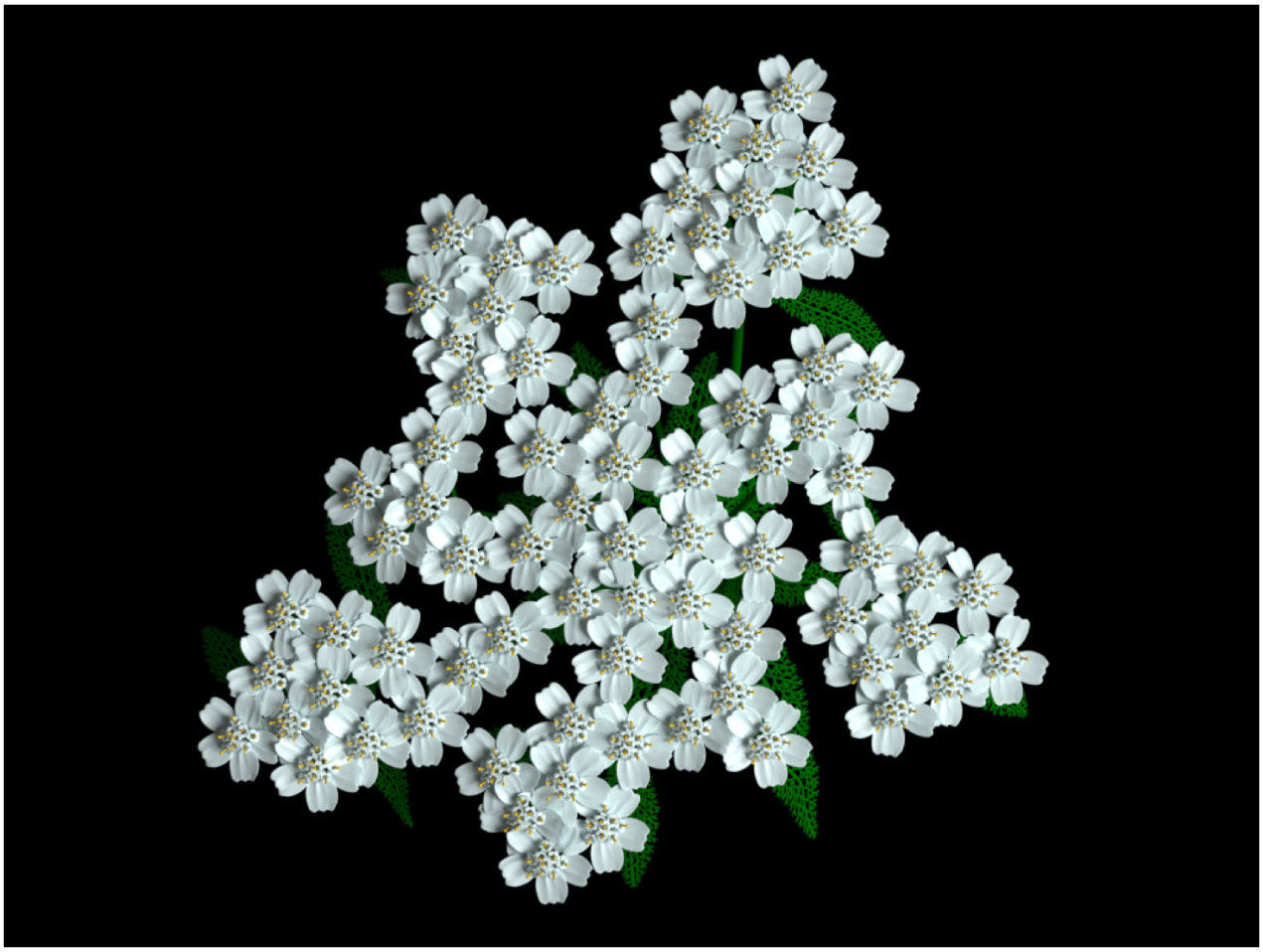

Figure 9.10: Model of a yarrow viewed from the top. Note that the phyllotaxis present in the yarrow model was procedurally generated from the model seen in Figure 5.3.

Yarrow (Figures 9.9, 9.10, 9.13, top and 9.14) is a typical example of a corymb inflorescence. The branching structure supports small heads with disk and ray florets. Their dimorphism has been captured using the model in Figure 6.4a. The seemingly irregular shape of the yarrow inflorescence was accurately captured by the recursive phyllotaxis algorithm shown in Figure 5.3. The branching structure (Figure 9.13, top and 9.14), supporting the florets in a single plane, was modeled using the extended Rodkaew algorithm: as illustrated in Figure 7.6, but with parameters promoting a more elongated structure. Leaves were generated using the positional-information method described in [8].

Figure 9.11: A photograph and a model of an *Orlaya grandiflora* inflorescence. These images illustrate some of the key elements of our work: the organization of florets into a planar canopy, hierarchical phyllotaxis, the dependency of floret type and petal size on their position in the inflorescence, and the deformation of some petals due to collisions. Photograph by Holger Casselmann licensed under CC BY_SA 3.0.

Figure 9.12: Orlaya model as viewed from the side.

The model of orlaya (Figures 9.11, 9.12 and 9.13, bottom) integrates most elements discussed in this thesis. The inflorescence is a compound umbel. Both the umbellets within the main umbel and the individual florets within the umbellets exhibit spiral phyllotaxis. The florets in inward positions within the umbellets are small and have five-fold symmetry. Select outer florets have enlarged petals, with the size depending on the available space. Finally, select florets on the periphery of the whole inflorescence have very large petals. The dimorphism model in Figure 6.4c accurately captures this multiplicity of forms. The florets are supported in a single plane by a branching structure organized into two levels of whorls. We reproduced this branching system by modeling each level separately (Figure 7.2b).

Figure 9.13: Branching structures subtending the floral canopies of yarrow (corymb of heads) and orlaya (compound umbel) modeled using the extended Rodkaew algorithm.

Figure 9.14: Branching structure of yarrow (corymb of heads) modeled using the extended Rodkaew algorithm.

Chapter 10

Discussion and future work

The work described in this thesis, and the paper related to it [1], have advanced previous methods for modeling inflorescences by introducing methods for: (i) modeling and animating florets with (partially) fused petals; (ii) detecting and resolving collisions that may occur in densely packed inflorescences using position-based dynamics extended to growing surfaces; (iii) extending Ridley's phyllotaxis model to simulate and animate the dynamics of phyllotaxis in simple, hierarchical, and recursive patterns; (iv) algorithmically determining floret type, size, and developmental stage in dimorphic inflorescences; and (v) modeling branching inflorescence structures as a self-organizing process driven by the distribution of florets. This process is particularly useful when modeling inflorescences with a smooth floral canopy, such as corymbs. Visual evaluations of the models suggest that this suite of modeling techniques can capture the form and animate the development of diverse inflorescences. This makes the entire method applicable to computer imagery and supports the biological hypotheses from which the methods are founded upon. From a broader perspective, the results of the method demonstrate the usefulness of the geometric approach to morphogenesis, where the emergence of patterns and forms is described in geometric rather than chemical terms. Upon formalizing geometric abstractions of biological phenomena observed in inflorescences, some amount of complexity is reduced or redistributed, making it faster and more flexible to explore the breadth of forms produced. This quest for geometric abstraction of biological phenomena has driven much of my research, as it provides a means to procedurally generate physically plausible inflorescence forms. This method, a comprised suite of biologically founded, geometrical modeling techniques, procedurally generates models of inflorescences preserving key characteristics and resemblances to those observed in nature. The component techniques of the method compose a user configurable pipeline for modeling many varieties of inflorescences.

This work has exposed many open problems deserving further research. The Floret Editor

should be expanded to more detailed modeling of florets, such as the internal organs and supporting structures. Internal organs are important to characterizing the form of many inflorescences (see Figure 10.1). The current implementation of the Floret Editor can only model one partially fused

Figure 10.1: Photograph examples of inflorescences where the internal organs are necessary features in characterizing the inflorescences. Photographs courtesy of Przemyslaw Prusinkiewicz.

corolla with radial symmetry at a time, while other features must be designed separately (e.g. the involucral bracts in Figure 9.3 were modeled within Blender instead of the Floret Editor). This is especially cumbersome in developing florets, as currently the developing internal organs need to be coordinated within Scene file to compose together with the floret. Florets exhibiting bilateral symmetry are also not currently producible within the Floret Editor, however an extension to the modeling environment could easily be included.

In the current implementation, the branching structure is unaffected by the collisions of developing floral canopy. In conceiving the method, I purposely ignored this interaction between the branching structure and the floral canopy, as it was assumed that inflorescences would have fully developed, and relatively sturdy branching structures, thus the effect of colliding florets on the branching structure would be negligible. However, this is not always so, and a holistic approach to developing both the branching structure and floral canopy together remains to be devised (although methods for computing stress responses on developing tree models do exist, e.g. Pirk et al. [67]).

View-dependent modeling techniques could be used to optimize viewing the inflorescence by

adjusting the level of detail to the scale at which the models are presented, and suppressing the generation/visualization of elements that lie outside the field of view or are obscured. This opens avenues of inquiry of perceptive modeling in reducing floret complexity (e.g. texture synthesis, mesh element granularity, collision resolution) to the levels of complexity perceivable to the observer. The inclusion of view-dependent modeling techniques to the current pipeline could address certain complexity issues that are evident in modeling scenes of large, dense arrangements of many flowers (e.g. gardens).

The method for simulating flower opening by blending key poses works well in practice, but does not conserve petal area, and could potentially be improved by using current methods for shape-preserving mesh transformation, e.g. [68]. Furthermore, the current collision-detection method is satisfactory when the initial configuration is collision-free, but in some cases this assumption is difficult to satisfy. Problems occur, for example, when florets are packed in a bud. A possible solution may be to place all elements of the initial inflorescence in an artificial yet easy-to-specify collision-free configuration, then procedurally assemble all elements into the desired structure while detecting and resolving collisions. From a broader perspective, a similar technique could lead to the solution of another challenging problem in computer graphics: modeling bouquets of flowers.

Recently, the collision-handling method of this thesis has been re-implemented in the Virtual Laboratory [30] environment as a stand alone model; a proof of concept for the eventual full incorporation into VLab. However, there are still many technical considerations yet to be addressed. Once incorporated into VLab, a wealth of previously designed inflorescences models will be simulated again, using the collision-handling technique to procedurally generate dense floral arrangements in a collision free manner.

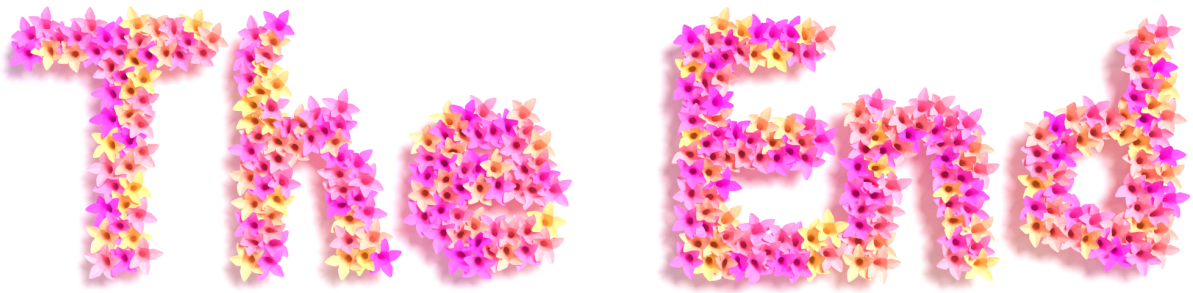

Figure 10.2: Animation of dense flower arrangement sampled from text.

Bibliography

[1] Owens, A., Cieslak, M., Hart, J., Classen-Bockhoff, R., Prusinkiewicz, P.: Modeling dense inflorescences. ACM Transactions on Graphics (TOG) **35**(4), 136 (2016)

[2] Müller, M., Heidelberger, B., Hennix, M., Ratcliff, J.: Position based dynamics. Journal of Visual Communication and Image Representation **18**(2), 109–118 (2007)

[3] Prusinkiewicz, P., Lindenmayer, A., Hanan, J.: Developmental models of herbaceous plants for computer imagery purposes. Computer Graphics **22**(4), 141–150 (1988)

[4] Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer, New York (1990). With J. Hanan, F. Fracchia, D. Fowler, M. de Boer, and L. Mercer

[5] Fowler, D.R., Prusinkiewicz, P., Battjes, J.: A collision-based model of spiral phyllotaxis. Computer Graphics **26**(2), 361–368 (1992)

[6] Prusinkiewicz, P., Hammel, M.S., Mjolsness, E.: Animation of plant development. In: Proceedings of SIGGRAPH 93. Annual Conference Series, pp. 351–360 (1993)

[7] Lintermann, B., Deussen, O.: Interactive modeling of plants. IEEE Computer Graphics and Applications **19**(1), 56–65 (1999)

[8] Prusinkiewicz, P., Mündermann, L., Karwowski, R., Lane, B.: The use of positional information in the modeling of plants. In: Proceedings of SIGGRAPH 2001. Annual Conference Series, pp. 289–300 (2001)

[9] Ijiri, T., Owada, S., Okabe, M., Igarashi, T.: Floral diagrams and inflorescences: Interactive flower modeling using botanical structural constraints. ACM Transactions on Graphics **24**(3), 720–726 (2005)

[10] Ijiri, T., Owada, S., Igarashi, T.: Seamless integration of initial sketching and subsequent detail editing in flower modeling. Computer Graphics Forum **25**(3), 617–624 (2006)

[11] Kirchoff, B.: Shape matters: Hofmeister's rule, primordium shape, and flower orientation. International Journal of Plant Sciences **164**(4), 505–517 (2003)

[12] Douady, S., Couder, Y.: Phyllotaxis as a dynamical self organizing process. Parts I–III. Journal of Theoretical Biology **178**, 255–312 (1996)

[13] Smith, R., Kuhlemeier, C., Prusinkiewicz, P.: Inhibition fields for phyllotactic pattern formation: A simulation study. Canadian Journal of Botany **84**, 1635–1649 (2006)

[14] Classen-Bockhoff, R., Bull-Hereñu, K.: Towards an ontogenetic understanding of inflorescence diversity. Annals of Botany **112**, 1523–1542 (2013)

[15] Weberling, F.: Morphology of Flowers and Inflorescences. Cambridge University Press, Cambridge (1992)

[16] Classen-Bockhoff, R.: Florale Differenzierung in komplex organisierten Asteraceenköpfen. Flora **186**(1-2), 1–22 (1992)

[17] Classen-Bockhoff, R.: Functional units beyond the level of the capitulum and cypsela in Compositae. In: Caligari, P., Hind, D. (eds.) Compositae: Biology and Utilization. Proceedings of the International Compositae Conference vol. 2, pp. 129–160. Royal Botanic Gardens, Kew (1994)

[18] Hernandez, L., Palmer, J.: Regeneration of the sunflower capitulum after cylindrical wounding of the receptacle. American Journal of Botany, 1253–1261 (1988)

[19] Battjes, J., Vischer, N., Bachmann, K.: Capitulum phyllotaxis and numerical canalization in *Microseris pygmaea* (Asteraceae: Lactuceae). American Journal of Botany, 419–428 (1993)

[20] Frijters, D.: Principles of simulation of inflorescence development. Annals of Botany **42**, 549–560 (1978)

[21] Reinhardt, D., *et al.*: Regulation of phyllotaxis by polar auxin transport. Nature **426**, 255–260 (2003)

[22] O'Connor, D.L., Runions, A., Sluis, A., Bragg, J., Vogel, J.P., Prusinkiewicz, P., Hake, S.: A division in PIN-mediated auxin patterning during organ initiation in grasses. PLoS Comput Biol **10**(1), 1003447 (2014)

[23] Sachs, T.: Pattern Formation in Plant Tissues. Cambridge University Press, Cambridge (1991)

[24] Harder, D., Prusinkiewicz, P.: The interplay between inflorescence development and function as the crucible of architectural diversity. Annals of Botany **112**, 1477–1493 (2013)

[25] Ijiri, T., Yokoo, M., Kawabata, S., Igarashi, T.: Surface-based growth simulation for opening flowers. In: Proceedings of Graphics Interface 2008, pp. 227–234 (2008). Canadian Information Processing Society

[26] Ridley, J.N.: Ideal phyllotaxis on general surfaces of revolution. Mathematical Biosciences **79**, 1–24 (1986)

[27] Rodkaew, Y., Chongstitvatana, P., Siripant, S., Lursinsap, C.: Particle systems for plant modeling. In: Hu, B.-G., Jaeger, M. (eds.) Plant Growth Modeling and Applications. Proceedings of PMA03, pp. 210–217. Tsinghua University Press and Springer, Beijing (2003)

[28] Anastacio, F., Costa Sousa, M., Samavati, F., Jorge, J.: Modeling plant structures using concept sketches. In: Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering, pp. 105–113 (2006). ACM

[29] Thompson, d.: On Growth and Form, 2nd Edition. University Press, Cambridge (1942)

[30] Prusinkiewicz, P., et al.: The Virtual Laboratory. `http://algorithmicbotany.org/virtual_laboratory/` (2016)

[31] Li, J., Liu, M., Xu, W., Liang, H., Liu, L.: Boundary-dominant flower blooming simulation. Computer Animation and Virtual Worlds **26**(3-4), 433–443 (2015)

[32] Liang, H., Mahadevan, L.: Growth, geometry, and mechanics of a blooming lily. Proceedings of the National Academy of Sciences **108**(14), 5516–5521 (2011)

[33] Green, A.A., Kennaway, J.R., Hanna, A.I., Bangham, J.A., Coen, E.: Genetic control of organ shape and tissue polarity. PLoS Biology **8**, 1000537 (2010)

[34] Sederberg, T., Gao, P., Wang, G., Mu, H.: 2-D shape blending: an intrinsic solution to the vertex path problem. In: Proceedings of the SIGGRAPH, pp. 15–18 (1993). ACM

[35] Bloomenthal, J.: Calculation of reference frames along a space curve. In: Glassner, A. (ed.) Graphics Gems vol. 1, pp. 567–571. Academic Press, Boston (1990)

[36] Bender, J., Müller, M., Otaduy, M.A., Teschner, M., Macklin, M.: A survey on position-based simulation methods in computer graphics. Computer Graphics Forum **33**(6), 228–251 (2014)

[37] Bender, J., Muller, M., Macklin, M.: Position-based simulation methods in computer graphics. EUROGRAPHICS Tutorial Notes (2015)

[38] Goldstein, H.: Classical Mechanics. Pearson Education India, (1965)

[39] Zhao, Y., Barbič, J.: Interactive authoring of simulation-ready plants. ACM Transactions on Graphics (TOG) **32**(4), 84 (2013)

[40] Wicke, M., Lanker, H., Gross, M.: Untangling cloth with boundaries. In: Proc. of Vision, Modeling, and Visualization, pp. 349–356 (2006)

[41] Brochu, T., Edwards, E., Bridson, R.: Efficient geometrically exact continuous collision detection. ACM Transactions on Graphics (TOG) **31**(4), 96 (2012)

[42] Bridson, R., Fedkiw, R., Anderson, J.: Robust treatment of collisions, contact and friction for cloth animation. In: ACM Transactions on Graphics (TOG), vol. 21, pp. 594–603 (2002)

[43] Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., Gross, M.: Optimized spatial hashing for collision detection of deformable objects. In: VMV, vol. 3, pp. 47–54 (2003)

[44] Wong, W.S.-K., Baciu, G.: A randomized marking scheme for continuous collision detection in simulation of deformable surfaces. In: Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications, pp. 181–188 (2006). ACM

[45] Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., Lévy, B.: Polygon Mesh Processing. CRC press, (2010)

[46] Provot, X.: Collision and self-collision handling in cloth model dedicated to design garments. In: Proceedings of the Eurographics Workshop on Computer Animation and Simulation, pp. 177–189. Springer, (1997)

[47] Cardano, G., Witmer, T.R.: Ars Magna or the Rules of Algebra, (1993)

[48] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C vol. 2. Cambridge university press Cambridge, (1996)

[49] Eberly, D.H.: 3d game engine architecture (2005)

[50] Tang, M., Manocha, D., Tong, R.: Fast continuous collision detection using deforming non-penetration filters. In: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 7–13 (2010). ACM

[51] Bender, J., Dequidt, J., Duriez, C., Zachmann, G.: Physically-based character skinning (2013)

[52] Loop, C.: Smooth subdivision surfaces based on triangles (1987)

[53] Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with cuda. Queue **6**(2), 40–53 (2008)

[54] Fowler, D.R., Hanan, J., Prusinkiewicz, P.: Modelling spiral phyllotaxis. Computers & Graphics **13**(3), 291–296 (1989)

[55] Erickson, R.O.: The geometry of phyllotaxis. In: Dale, J.E., Milthrope, F.L. (eds.) The Growth and Functioning of Leaves, pp. 53–88. University Press, Cambridge (1983)

[56] Vogel, H.: A better way to construct the sunflower head. Mathematical Biosciences **44**, 179–189 (1979)

[57] Harris, E., Tucker, S., Urbatsch, L.: Floral initiation and early development in *Erigeron philadelphicus* (Asteraceae). American Journal of Botany, 108–121 (1991)

[58] Hirmer, M.: Zur Kenntnis der Schraubenstellungen im Pflanzenreich. Planta **14**(1), 132–206 (1931)

[59] Battjes, J., Prusinkiewicz, P.: Modeling meristic characters of Asteracean flowerheads. In: Barabé, D., Jean, R. (eds.) Symmetry in Plants, pp. 281–312. World Scientific, Singapore (1998)

[60] Neubert, B., Franken, T., Deussen, O.: Approximate image-based tree-modeling using particle flows. ACM Transactions on Graphics (TOG) **26**(3), 88 (2007)

[61] Kang, J., Tang, J., Donnelly, P., Dengler, N.: Primary vascular pattern and expression of athb-8 in shoots of arabidopsis. New Phytologist **158**(3), 443–454 (2003)

[62] Kim, Y., Sinclair, R., Chindapol, N., Kaandorp, J., De Schutter, E.: Geometric theory predicts bifurcations in minimal wiring cost trees in biology are flat. PLoS Computational Biology **8**(4), 1002474 (2012)

[63] Sederberg, T., Parry, S.: Free-form deformation of solid geometric models. Computer Graphics **20**(4), 151–160 (1986)

[64] Shinozaki, K., Yoda, K., Hozumi, K., Kira, T.: A quantitative analysis of plant form — the pipe model theory. I. Basic analyses. Japanese Journal of Ecology **14**(3), 97–105 (1964)

[65] MacDonald, N.: Trees and Networks in Biological Models. J. Wiley & Sons, New York (1983)

[66] Sammataro, D., Garment, M., Erickson Jr, E.: Anatomical features of the sunflower floret. Reprints — US Department of Agriculture, Agricultural Research Service (1986)

[67] Pirk, S., Niese, T., Hädrich, T., Benes, B., Deussen, O.: Windy trees: computing stress response for developmental tree models. ACM Transactions on Graphics **33**(6) (2014)

[68] Lipman, Y., Cohen-Or, D., Gal, R., Levin, D.: Volume and shape preservation via moving frame manipulation. ACM Transactions on Graphics **26**(1), 5 (2007)

Appendix A

Constraint Linearization

A.1 Signed dihedral angle bending constraints

Given positions $\boldsymbol{p}_i \in \mathbb{R}^3$, angular bending constraints of Position Based Dynamics (PBD) [2] correspond to neighbouring triangles and represents the desired dihedral angle $\phi \in [0^\circ, 180^\circ)$ between the pair of adjacent triangles $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3)$ and $(\boldsymbol{p}_1, \boldsymbol{p}_4, \boldsymbol{p}_2)$ sharing an edge $(\boldsymbol{p}_1, \boldsymbol{p}_2)$ (Figure 3.1b)

$$\begin{aligned} C_{bend}(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4) &= \arccos\left(\frac{(\boldsymbol{p}_3 - \boldsymbol{p}_1) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)}{\| (\boldsymbol{p}_3 - \boldsymbol{p}_1) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1) \|} \cdot \frac{(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_4 - \boldsymbol{p}_1)}{\| (\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_4 - \boldsymbol{p}_1) \|}\right) - \phi \\ &= \arccos(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2) - \phi \end{aligned} \tag{A.1}$$

As detailed in Chapter 3, this formulation produces mirrored triangle configurations, and as a proposed solution was a *signed* dihedral angle measurement to disambiguate the mirrored configurations

$$\phi_{curr} = \operatorname{sgn}(\boldsymbol{e} \cdot [\boldsymbol{n}_1 \times \boldsymbol{n}_2]) \arccos(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2) \tag{A.2}$$

where we choose a canonical orientation for the shared edge between the adjacent triangles: $\boldsymbol{e} = \boldsymbol{p}_2 - \boldsymbol{p}_1$ (see Figure 3.1b). Luckily, this reformulation requires only slight modification in the interpretation of the gradient $\nabla_{\boldsymbol{p}_i} C_{bend}$ for the purposes of updating the positions constrained to a signed dihedral angle.

If $\operatorname{sgn}(\phi_{curr}) = \operatorname{sgn}(\phi)$, then the current triangle configuration need not to be pushed through an intermediate planar position (0°), and PBD can proceed in updating as described by Muller [2], with the required angle correction $\operatorname{abs}(\phi_{curr}) - \operatorname{abs}(\phi)$. However, if $\operatorname{sgn}(\phi_{curr}) \neq \operatorname{sgn}(\phi)$, then the configuration must be pushed to the correct side. But this is accomplished by moving the positions in the opposite direction of the gradient until signs of the current signed angle and rest angle agree. The required angle correction to push the signs into agreement is $\operatorname{abs}(\phi_{curr} - \phi)$.

Appendix B

Narrow Phase Collision Detection

B.1 Cross product expanded as a quadratic polynomial

Let the vectors $\boldsymbol{b}$ and $\boldsymbol{c} \in \mathbb{R}^3$ be parametrized as linear functions of t

$$\begin{aligned} \boldsymbol{b} &= \boldsymbol{p}_b + t\boldsymbol{v}_b \\ \boldsymbol{c} &= \boldsymbol{p}_c + t\boldsymbol{v}_c \end{aligned} \tag{B.1}$$

where $\boldsymbol{p}_b$, $\boldsymbol{p}_c$, $\boldsymbol{v}_b$, $\boldsymbol{v}_c \in \mathbb{R}^3$, and $t \in \mathbb{R}$. We wish to expand the cross product expression $\boldsymbol{b}(t) \times \boldsymbol{c}(t)$ into a *quadratic* equation of t. Then,

$$\begin{aligned} \boldsymbol{b}(t) \times \boldsymbol{c}(t) &= (\boldsymbol{p}_b + t\boldsymbol{v}_b) \times (\boldsymbol{p}_c + t\boldsymbol{v}_c) \\ &= t^2 (\boldsymbol{v}_b \times \boldsymbol{v}_c) + t (\boldsymbol{v}_b \times \boldsymbol{p}_c + \boldsymbol{p}_b \times \boldsymbol{v}_c) + \boldsymbol{p}_b \times \boldsymbol{p}_c \\ &= \boldsymbol{Q}t^2 + \boldsymbol{R}t + \boldsymbol{S} \end{aligned} \tag{B.2}$$

Where $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{S} \in \mathbb{R}^3$, with coordinates as follows

$$\boldsymbol{Q} = \begin{bmatrix} y_{\boldsymbol{v}_b} z_{\boldsymbol{v}_c} - z_{\boldsymbol{v}_b} y_{\boldsymbol{v}_c} \\ z_{\boldsymbol{v}_b} x_{\boldsymbol{v}_c} - x_{\boldsymbol{v}_b} z_{\boldsymbol{v}_c} \\ x_{\boldsymbol{v}_b} y_{\boldsymbol{v}_c} - y_{\boldsymbol{v}_b} x_{\boldsymbol{v}_c} \end{bmatrix} \boldsymbol{R} = \begin{bmatrix} y_{\boldsymbol{p}_b} z_{\boldsymbol{v}_c} + y_{\boldsymbol{v}_b} z_{\boldsymbol{p}_c} - z_{\boldsymbol{p}_b} y_{\boldsymbol{v}_c} - z_{\boldsymbol{v}_b} y_{\boldsymbol{p}_c} \\ z_{\boldsymbol{p}_b} x_{\boldsymbol{v}_c} + z_{\boldsymbol{v}_b} x_{\boldsymbol{p}_c} - x_{\boldsymbol{p}_b} z_{\boldsymbol{v}_c} - x_{\boldsymbol{v}_b} z_{\boldsymbol{p}_c} \\ x_{\boldsymbol{p}_b} y_{\boldsymbol{v}_c} + x_{\boldsymbol{v}_b} y_{\boldsymbol{p}_c} - y_{\boldsymbol{p}_b} x_{\boldsymbol{v}_c} - y_{\boldsymbol{v}_b} x_{\boldsymbol{p}_c} \end{bmatrix} \boldsymbol{S} = \begin{bmatrix} y_{\boldsymbol{p}_b} z_{\boldsymbol{p}_c} - z_{\boldsymbol{p}_b} y_{\boldsymbol{p}_c} \\ z_{\boldsymbol{p}_b} x_{\boldsymbol{p}_c} - x_{\boldsymbol{p}_b} z_{\boldsymbol{p}_c} \\ x_{\boldsymbol{p}_b} y_{\boldsymbol{p}_c} - y_{\boldsymbol{p}_b} x_{\boldsymbol{p}_c} \end{bmatrix} \tag{B.3}$$

B.2 Test if point is in a triangle

Let a triangle be defined by three points $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c} \in \mathbb{R}^3$. We wish to test if a point $\boldsymbol{p} \in \mathbb{R}^3$ (that is coplanar to $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$) is inside the triangle. To this end, we define $\boldsymbol{p}$ in terms of the triangle points $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ (a new basis and origin) with unknowns $r, s \in \mathbb{R}$:

$$\boldsymbol{p} = \boldsymbol{a} + (\boldsymbol{b} - \boldsymbol{a})\, r + (\boldsymbol{c} - \boldsymbol{a})\, s. \tag{B.4}$$

We then rearrange and relabel this equation:

$$\begin{aligned} \boldsymbol{p} - \boldsymbol{a} &= (\boldsymbol{b} - \boldsymbol{a})\, r + (\boldsymbol{c} - \boldsymbol{a})\, s \\ \boldsymbol{w} &= \boldsymbol{u} r + \boldsymbol{v} s \end{aligned} \tag{B.5}$$

by letting $\boldsymbol{w} = \boldsymbol{p} - \boldsymbol{a}$, $\boldsymbol{u} = \boldsymbol{b} - \boldsymbol{a}$ and $\boldsymbol{v} = \boldsymbol{c} - \boldsymbol{a}$.

As we have two unknowns, to solve for them we need two independent (scalar) equations. We form them by multiplying Equation B.5 by $\boldsymbol{u}$ and $\boldsymbol{v}$ separately:

$$\begin{aligned} \boldsymbol{w} \cdot \boldsymbol{u} &= (\boldsymbol{u} \cdot \boldsymbol{u})\, r + (\boldsymbol{v} \cdot \boldsymbol{u})\, s \\ \boldsymbol{w} \cdot \boldsymbol{v} &= (\boldsymbol{u} \cdot \boldsymbol{v})\, r + (\boldsymbol{v} \cdot \boldsymbol{v})\, s. \end{aligned} \tag{B.6}$$

This provides a linear system of 2 equations with 2 unknowns, which can be easily solved, giving the following

$$\begin{aligned} r &= \frac{(\boldsymbol{v} \cdot \boldsymbol{v})\,(\boldsymbol{w} \cdot \boldsymbol{u}) - (\boldsymbol{v} \cdot \boldsymbol{u})\,(\boldsymbol{w} \cdot \boldsymbol{v})}{(\boldsymbol{u} \cdot \boldsymbol{u})\,(\boldsymbol{v} \cdot \boldsymbol{v}) - (\boldsymbol{v} \cdot \boldsymbol{u})\,(\boldsymbol{u} \cdot \boldsymbol{v})} \\ s &= \frac{(\boldsymbol{u} \cdot \boldsymbol{u})\,(\boldsymbol{w} \cdot \boldsymbol{v}) - (\boldsymbol{u} \cdot \boldsymbol{v})\,(\boldsymbol{w} \cdot \boldsymbol{u})}{(\boldsymbol{u} \cdot \boldsymbol{u})\,(\boldsymbol{v} \cdot \boldsymbol{v}) - (\boldsymbol{v} \cdot \boldsymbol{u})\,(\boldsymbol{u} \cdot \boldsymbol{v})}. \end{aligned} \tag{B.7}$$

If $r \geq 0$, $s \geq 0$ and $r + s \leq 1$, the point $\boldsymbol{p}$ lies inside the triangle with the vertices of $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$.

B.3 Compute minimum distance between edges

Let edges e_0 and e_1 be defined by line segments $e_0 = [\boldsymbol{p}_0, \boldsymbol{p}_1]$ and $e_1 = [\boldsymbol{q}_0, \boldsymbol{q}_1]$, where $\boldsymbol{p}_i, \boldsymbol{q}_i \in \mathbb{R}^3$. We describe points along each edge as linear functions of $r, s \in [0 \ldots 1]$:

$$\begin{aligned} \boldsymbol{p}(r) &= \boldsymbol{p}_0 + r\,(\boldsymbol{p}_1 - \boldsymbol{p}_0) \\ \boldsymbol{q}(s) &= \boldsymbol{q}_0 + s\,(\boldsymbol{q}_1 - \boldsymbol{q}_0)\,. \end{aligned} \tag{B.8}$$

Then, by defining $\boldsymbol{w}(r,s) = \boldsymbol{p}(r) - \boldsymbol{q}(s)$, and finding the minimum distance between e_0 and e_1 is equivalent to

$$\underset{(r,s)\in[0,1]\times[0,1]}{\operatorname{argmin}} \; [\boldsymbol{w} \cdot \boldsymbol{w}]^{\frac{1}{2}} \equiv \underset{(r,s)\in[0,1]\times[0,1]}{\operatorname{argmin}} \; \boldsymbol{w} \cdot \boldsymbol{w}. \tag{B.9}$$

If the global minimum (r,s) to the quadratic function $\boldsymbol{w}(r,s)\cdot\boldsymbol{w}(r,s)$ is outside $[0,1]\times[0,1]$, then one or both of the points $\boldsymbol{p}(r)$ and $\boldsymbol{q}(s)$ are not valid points along the edges e_0 and e_1. In this case, we find the minimum (r,s) on the boundary of $[0,1]\times[0,1]$. This leads to a small number of cases to find the minimum (s,t) constrained to $[0,1]\times[0,1]$. We use calculus to compute the minimum along each edge of the boundary ($r=0$, $s=0$, $r=1$ and $s=1$). For example, let $r=0$, then:

$$\begin{aligned}\boldsymbol{w}(0,s)\cdot\boldsymbol{w}(0,s) &= (\boldsymbol{p}_0-\boldsymbol{q}_0-s(\boldsymbol{q}_1-\boldsymbol{q}_0))\cdot(\boldsymbol{p}_0-\boldsymbol{q}_0-s(\boldsymbol{q}_1-\boldsymbol{q}_0))\\ &= (\boldsymbol{w}_0-s\boldsymbol{v})\cdot(\boldsymbol{w}_0-s\boldsymbol{v})\end{aligned} \tag{B.10}$$

where $\boldsymbol{w}_0=\boldsymbol{p}_0-\boldsymbol{q}_0$ and $\boldsymbol{v}=\boldsymbol{q}_1-\boldsymbol{q}_0$. Taking the derivative of Equation B.10 with respect to s and setting to 0:

$$\begin{aligned}0 &= \frac{d}{ds}\boldsymbol{w}(0,s)\cdot\boldsymbol{w}(0,s)\\ &= -2\boldsymbol{v}\cdot(\boldsymbol{w}_0-s\boldsymbol{v})\end{aligned} \tag{B.11}$$

and solving for s, we find the minimum along the edge of the boundary at $(0,s)$, where $s=\frac{\boldsymbol{v}\cdot\boldsymbol{w}_0}{\boldsymbol{v}\cdot\boldsymbol{v}}$. If $0\leq s\leq 1$, then this is the minimum along the boundary; however, if s is outside of this range, then an endpoint of the edge ($s=0$ or $s=1$) is the minimum along $r=0$. Eberly[1] provides further exposition on this solution; as well, proposes an alternative constrained conjugate gradient algorithm that is more robust, and could potentially replace the method currently implemented in this thesis.

[1] https://www.geometrictools.com/Documentation/DistanceLine3Line3.pdf

Appendix C

Floral Canopy Composer Usage

C.1 *Floral Canopy Composer* parameter file specifications

solver iterations: (*int*)iterations

Number of PBD internal constraints solver iterations per simulation step.

collision solver iterations: (*int*)iterations

Number of PBD collision constraints solver iterations per simulation step.

kStretch: (*float*)stiffness

Stiffness$\in [0,1]$ of edge constraints.

kBend: (*float*)stiffness

Stiffness$\in [0,1]$ of bending constraints.

kCollision: (*float*)stiffness

Stiffness$\in [0,1]$ of collision constraints.

spacing: (*float*)spacing

Minimum distance used in collision constraints.

scene scale factor: (*float*)x, (*float*)y, (*float*)z

Scale loaded geometry independently in *x, y, z* directions.

updates per frame: (*int*)num

Number of simulations steps per rendered frame.

run for number of frames: (*int*)num

Run simulation for *num* frames without rendering.

sanity check per frames: (*int*)num

Run (slower) intersection check every *num* frames.

output animation frames number: (*int*)num

Output OBJ file of canopy every *num* frames.

output animation obj per frames: (*int*)num

When outputting an animation, skip *num* frames between writing OBJ files.

lattice size multiply: (*float*)x, (*float*)y, (*float*)z

Independently scale collision bounding box in *x, y, z* directions.

lattice make cube: (*bool*)make-cube

Force collision bounding box to be cube shaped, using the maximum dimension for the width, height and depth.

origin: (*int*)x, (*int*)y, (*int*)z

The *x, y, z* coordinates of the origin for the bounding box.

divisions: (*int*)x, (*int*)y, (*int*)z

Collision lattice divisions in *x, y, z* directions.

dimensions: (*float*)x, (*float*)y, (*float*)z

Collision lattice dimensions in *x, y, z* directions.

lattice auto bounds: (*bool*)auto-fit

Automatically compute and fit collision bounding box from geometry. If *false*, the bounding box is set by the *dimensions* entry.

color map file location: (*string*)file-path

Color map file path.

material map file location: (*string*)file-path

Material map file path.

scene file path: (*string*)file-path

File path to scene description file that specifies instanced florets positions and animations.

animation: (*int*)id (*float*)x (*float*)y (*float*)z (*string*)folderpath

Associate animation (sequence of OBJ files) stored in *folderpath* with unique number *id* so it may be referenced in scene files. Initially scaled by *x*, *y*, *z* values.

obj: (*int*)id (*float*)x (*float*)y (*float*)z (*string*)folderpath

Associate OBJ stored in *folderpath* with unique number *id* so it may be referenced in scene files. Initially scaled by *x*, *y*, *z*. Initially scaled by *x*, *y*, *z* values.

rigid body animation: (*int*)id (*float*)x (*float*)y (*float*)z (*string*)folderpath (*string*)sceneFolderpath

Associate rigid body animation files read from *folderpath*. The instanced geometry/animations are read from scence file at*sceneFolderpath*. Initially scaled by *x*, *y*, *z* values.

weight: (*int*)id (*string*)filepath

Text file of vertex IDs which lists those vertices of obj/animation *id* which should remain fixed while resolving collisions or growth.

bending stiffness: (*int*)id (*float*)stiffness

Provides alternative bending constraint *stiffness* coefficient to obj/animation *id* to that of the global *kBend*.

distance stiffness: (*int*)id (*float*)stiffness

Provides alternative distance constraint *stiffness* coefficient to obj/animation *id* to that of the global *kStretch*.

animation output folder location: *string*

Folder path to output OBJ files produced in animation.

animation output folder location: (*string*)folder-path

Folder path to output OBJ files produced in animation.

obj output file path: (*string*)file-name

File name of output OBJ file for current frame.

camera position: (*float*)x, (*float*)y, (*float*)z

Initial *x, y, z* coordinates of camera position.

camera up: (*float*)x, (*float*)y, (*float*)z

Iinitial *x, y, z* coordinates of camera up vector.

camera lookat: (*float*)x, (*float*)y, (*float*)z

Initial *x, y, z* coordinates of camera look at position.

camera pan speed: (*float*)speed

Initial camera pan speed.

camera zoom speed: (*float*)speed

Initial camera zoom speed.

camera rotate speed: (*float*)speed

Initial camera rotate speed.

near plane: (*float*)distance

Camera near distance.

far plane: (*float*)distance

Camera far distance.

camera fov: (*float*)fov Camera angle field of view.

light position: (*float*)x, (*float*)y, (*float*)z

The *x, y, z* coordinates of the light.

shadow frustrum: (*float*)left, (*float*)right, (*float*)top, (*float*)bottom, (*float*)back, (*float*)front

Description of frustum used in shadow map computations.

Example parameter file for the *Floral Canopy Composer* program

```
# Constraints parameters
solver iterations: 50
collision iterations: 10
max collision retries: 20
kStretch: 0.99f
kBend: 0.99f
kCollision: 1.f
spacing: 0.005

# Simulation parameters
updates per frame: 1
run for number of frames: 1
sanity check per frames: 50

# Scene
# each scene file entry's ID corresponds to one of the animations or OBJs listed below
scene file path: ./Scenes/s1.sce
animation: 0    1 1 1 ./lilyAnimation/
animation: 1    2 2 2 ./daisyAnimation/
obj: 2  0.5 1 0.5 ./singleRose.obj

# Collision Lattice Stuff
lattice auto bounds: true
lattice make cube: false
lattice size multiply: 1.5 1.5 1.5
origin: -5, -5, -5
dimensions: 10, 10, 10
divisions: 100

# File paths
color map file location:        ./Color/colormap.map
material map file location:     ./Material/material.mat

# Light and Camera
light position: -10, 10, 10
shadow frustrum: -1, 1, -1, 1, -10, 20
camera position: 0, 0, -1
camera up: 0, 1, 0
camera lookat: 0, 0, 0
```

C.2 Scene file specifications

All scene placement entries in the Scene file have the same fundamental form:

type: *id* pos_x pos_y pos_z $scale_x$ $scale_y$ $scale_z$ up_x up_y up_z $right_x$ $right_y$ $right_z$

where *id* matches an obj/animation entry in the parameter file, and signals the instantiation of geometry/animation in the scene at the position, scale and orientation from the other vector values. The list below enumerates the scene placement types. The ID, position, scale and orientation values are collected and labelled as $\langle preamble \rangle$ for conciseness.

o: $\langle preamble \rangle$

place OBJ

ca: $\langle preamble \rangle$ (*int*)frameStart (*int*)frameEnd

place constraint based animation

i: $\langle preamble \rangle$ (*float*)interploationValue

interpolated instance of animation geometry

sa: $\langle preamble \rangle$ (*int*)frameStart (*int*)frameEnd (*float*)scale

scale geometry animation

ta: $\langle preamble \rangle$ (*int*)frameStart (*int*)frameEnd (*float*)x (*float*)y (*float*)z

translate geometry animation

ra: $\langle preamble \rangle$ (*int*)frameStart (*int*)frameEnd (*float*)x (*float*)y (*float*)z (*float*)angle

rotate geometry around axis by an angle animation

rba: $\langle preamble \rangle$ (*int*)frameStart (*int*)frameEnd

rigid body animation

A material rendering model, defined within the material map file, is assigned to scene placements by writing:

mtl: *matID*

immediately after the scene file entry. *matID* is an unique ID corresponding to 256 material definitions in the material map. Text line comments may be added if prepended with #.

Example scene file

```
# Constraint animation with ID 0, corresponding to
# an animation listed in the FCComposer parameter file.
# ca: ID Position Scale Up Left FrameStart FrameEnd
ca: 0   0 0 0   1 1 1   0 1 0   1 0 0   0       1000

# Material 3 applied to the following scene placements
mtl: 3
ca: 0   5 0 0   1 1 1   0 1 0   1 0 0   0       1000
o: 2    0 10 0  5 5 5   1 0 0   0 0 1
```

C.3 Rigid body animation file specifications

Rigid body animation file entries are as follows

$$id\ pos_x\ pos_y\ pos_z\ scale_x\ scale_y\ scale_z\ up_x\ up_y\ up_z\ right_x\ right_y\ right_z$$

where *id* corresponds to the scene placement of the same numbering in the adjoining scene file, with the geometry/animation in of the placement at the position, scale and orientation from the other vector values.

C.4 *Floral Canopy Composer* controls usage

Command	Action
MOUSE	Left Mouse Click (LMC) Right Mouse Click (RMC)
LMC + Move mouse	orbit camera around cursor position
RMC	move cursor to vertex
Alt+RMC	move cursor and camera to vertex
KEYBOARD	
A	move camera left
D	move camera right
W	move camera forward
S	move camera back
Q	move camera down
E	move camera up
Arrow Up	rotate camera up
Arrow Down	rotate camera down
Arrow Left	rotate camera left
Arrow Right	rotate camera right
MESH EDIT MODE	
Ctrl+E	toggle mesh edit mode
LMC	select nearest vertex in screen space
LMC+Ctrl	select all vertices of triangle selected
LMC+Shift+Ctrl	select all vertices of object selected
Arrow Up	screen space move selected vertices up
Arrow Down	screen space move selected vertices down

Arrow Left	screen space move selected vertices left
Arrow Right	screen space move selected vertices right
,	move selected vertices into the screen
.	move selected verices out of screen
MODIFIERS	
[	half the movement speed
[+Alt	half edit movement speed
[+Shift	half the rotation speed
]	double the movement speed
]+Alt	double edit movement speed
]+Shift	double the rotation speed
FUNCTIONS	
Ctrl+U	write animation files (series of OBJs to folder)
Ctrl+V	reset view to param camera
Alt+V	reset view to cursor
Shift+V	print current camera position (so as to copy into param file)
Ctrl+T	run sanity check for intersections
Ctrl+O	reload param file (without reloading geometry)
Ctrl+P	reload param file (reloading geometry)
Ctrl+I	reload param file and camera (reloading geometry)
Ctrl+R	run simulation
Ctrl+F	step simulation once
space OR Ctrl+S	pause simulation
Ctrl+M	toggle perspective/orthographic view
Ctrl+N	widen orthographic projection

Ctrl+B	narrow orthographic projection
Ctrl+J	loop subdivision with collision resolution (post process)
Ctrl+C	reload color and material map
Ctrl+Y	print current frame, AABB, and geometry count
Ctrl+L	write OBJ of current frame
Ctrl+X	print bounding box of simulation
RENDER TOGGLE	
0	cursor
1	smooth shading
2	wireframe overlay
3	normals
4	bounding box
5	bending constraints
6	distance constraints
7	test vertex indices (for testing purposes)
8	test points (for testing purposes)
9	fixed points
Ctrl+-	render from (shadow) light POV
-	shadows

Table C.1: *Floral Canopy Composer* controls usage