Simulation of Climbing Plants with Twining Behaviour

By

W. O. N. Wickramaratne

15001504

This dissertation is submitted to the University of Colombo School of Computing
In partial fulfillment of the requirements for the
Degree of Bachelor of Science Honours in Computer Science

University of Colombo School of Computing
35, Reid Avenue, Colombo 07,
Sri Lanka
July 2020

Declaration

I, W. O. N. Wickramaratne and 2015/CS/150 hereby certify that this dissertation entitled "Simulation of Climbing Plants with Twining Behaviour" is entirely my own work and it has never been submitted nor is currently been submitted for any other degree.

Date	Student's Signature

I, M. I. E. Wickramasinghe, certify that I supervised this dissertation entitled "Simulation of Climbing Plants with Twining Behaviour" conducted by W. O. N. Wickramaratne in partial fulfillment of the requirements for the degree of Bachelor of Science Honours in Computer Science.

Date	Supervisor's Signature

Abstract

Vegetation simulation in computer graphics is a highly active research area due to its vast variety of applications. Among various types of vegetation, climbing plants are complex to simulate because of their rigid and soft body characteristics. Though state of the art models were able to simulate the climbing plants in a visually realistic manner none of them was able to simulate the biomechanical behaviors of the twining plant. As twinning plants which is a subcategory of climbing plants show abstract mechanical properties, a robust biomechanically accurate model will facilitate many other research areas and it will fill the void on visualizing point of interest scenes of twining plants growth in gaming and cinema industries.

Previously, an attempt to fill the void in this area was made by Gunawardhana et al. their proposed model was jittery and it lacked stability. The proposed research was initiated as a continuation of this previous attempt with an aim to synthesize a robust model that can simulate a twining plant in a biomechanically accurate manner by improving the previous model. However, this approach had to be rejected due to its limitations and a state machine driven particle-based novel model was proposed instead. This novel model was able to simulate the effect of circumnutation behavior, twining behavior, seed germination process, secondary growth, and leaf growth in a biomechanically accurate manner. The proposed model was evaluated in terms of realism and performance using a survey and standard performance evaluation measures.

Keywords: Twinning plant simulation, Climbing plant simulation, Biomechanical behaviour simulation, Vegetation simulations, Circumnutation simulation

Preface

In this dissertation a novel particle-based state-machine driven approach is introduced to address the problem, lack of biomechanical simulation models to simulate twining plants. Design choices related to analysis and rejection of the PBD based model relied on work done by Gunawardhana and Position-based dynamics. Integrating the particle-based approach with state machines for this application was our own work. The Modeling and implementation of biomechanical behaviors such as support finding (circumnutation), twining, growth, germination and structure formation described in design and implementation chapters are developed by myself in association with my supervisor. These approaches has not been proposed in any other work in the domain of vegetation modeling in computer graphics. The real-world experimental setup was developed to observe and capture visual data from a set of twining plants. Observations from these data were used to come up with design choices described in the design section. Measures used in performance analysis of the suggested model are standard measures used in the domain of computer graphics. However calculations and graphing was done by myself. The survey Design, and analysis of survey results for the evaluation of visual realism described in the evaluation chapter and complete conclusion chapter are my own work.

Acknowledgement

I would like to express my sincere respect and gratitude to my research supervisor, Dr. M.I.E. Wickramasinghe for providing their valuable support cooperation and suggestions throughout the different stages of the research project. I would also like to extend my respect and gratitude to Prof. N D. Kodikara and Dr. D.D. Karunaratne of the University of Colombo School of Computing for providing valuable feedback on the research proposal and interim evaluation to improve the project. I also take the opportunity to acknowledge the assistance provided by Dr. H. E. M. H. B. Ekanayake, senior lecturer of the University of Colombo School of Computing as the final year research project coordinator. I would like to thank Mr. K.M.B.S. Gunawardhana for the guidance and explanation given though out the project to analyze the previous work. I would like to thank all the people who voluntarily participated in the survey. I would also like to thank all of my friends for giving comments, motivation and warm friendship. It is a great pleasure for me to acknowledge all the teachers, mentors and people for the immense support they have given. Finally to my parents and family for their immeasurable sacrifices and love they have given throughout this beautiful journey.

Contents

List of Figures

List of Tables

List of Acronyms

PBD	Position Based Dynamics
FPS	Frames Per Second

Chapter 1

Introduction

Vegetation simulation has been a highly active research area among computer graphic research community for decades due to its vast variety of applications such as in immersive environments [1], landscaping, foresting, agriculture and robotics [2]. These researches include many different aspects of Plant ecosystems such as response to external forces, phototropism, gravitropism [3], combustion models [4] and biomechanical behaviours. There was a recent attempt which simulated climbing plants with twining behaviours [5]. Even though it managed to lay foundation for the simulation of circumnutation behaviour using position based dynamics and to introduce a taper function, there are number of areas where the this model can be improved. Especially the stabilization of this models' circumnutation behaviour, Interactions of circumnutation with external forces such as wind, effect of leafs mass to circumnutation, circumnutation behaviour with differently shaped objects are few areas among them.

1.1 Background to the Research

Plant simulation is a very broad topic as it contains a vast number of plant types, each with number of unique features. As the diversity among plants is high it's hard to come up with a generalized model to simulate all the type of plants. Therefore a number of models to simulate various aspects of plants using various approaches can be observed in literature. In following literature reviews the topic is narrowed down to climbing plants and the bio-mechanical behaviours of climbing plant simulation will be focused on. Although climbing plants have been an attraction for number of biologist and naturalist for a centuries, Darwin was the first to classify and document climbing plants. Though climbing plants were classified in to four groups as twinning plants, leaf climbers, tendril bearers and hook and root climbers by Darwin [6] initially, the modern botanists generally divide climbing plants by its

climbing behaviour into two broad groups [7]. First group is called bines which have climbing mechanism of twinning stems around a support. Second group is called vines which have the climbing mechanisms which employs the use of tendrils, suckers. Above mentioned climbing behaviours have been analyzed in biological aspects as well as in underline physics aspects. Biologically what makes the twining and coiling was analyzed and it was found out that gelatinous fibre plays an important role in it [8]. Goriely et al [9] have published a mathematical model using rod theory to model underlying physics of twining plants climbing mechanics and the tension of its stem. There have been number of papers published on simulation of climbing plants. These simulations have be modelled using a number of approaches to achieve different aspects of simulation. An implementation of climbing plant simulation using L-system and particle system was published few years back [10]. It managed to simulate climbing plants efficiently. Even though this model did not support interaction with external forces it managed to simulate climbing plants efficiently in visually pleasing manner regardless of the shape of support structure. Later more efficient and more interactive models were published [4]. These models had the ability to interact with external forces as well as user inputs (user had the ability to interactively drag and prune branches or seed new plants in dynamically changing environments). Climbing plants with tendril has support finding mechanism and coiling mechanism. A research has been conducted and a model was made to simulate this attaching and coiling behaviour[11]. Even though this model did not have the ability to interact with external forces and effects, it achieved its designed task which is to simulate fine details in close up scene in a visually pleasing manner efficiently. When simulation models of bines' are taken in to consideration (Climbing plants which has twining behaviours) there is a void in models which simulates the bio mechanical behaviours. Even though number of models have been published to achieve visually pleasing outcomes these available models have not performed well when it's the focus of interest. Especially models to simulate support finding behaviour and general circumnutation behaviours were nowhere to be found. To fill this gap Gunawardhana et al [5] have come up with an implementation of model to simulate the stem of twinning plant using Cosserat rod model and a particle system. In the suggested model constraints have been solved using Position based dynamic framework. While the implementation is sufficient enough to simulate support finding behaviour of the twining plants it does suffer from some stability issue when more and more particles added to the model and suggested model support only for vertical cylindrical support structures. Though the effect of external forces like wind is not taken into account when modelling the current model a new taper function has been introduced and the variation of mass throughout the stem is taken into account when the their model was developed.

1.2 Research Problem and Research Questions

1.2.1 Research Problem

None of the climbing plant simulation models in literature present bio-mechanical behaviours

1.2.2 Aims and Objectives

This ultimate aims of this research is to synthesize a generalized model to simulate biologically and bio-mechanically accurate climbing plant and critically evaluate its behavioural realism and computational efficiency. As first step currently available models from previous attempts and their extensibility will be evaluated. If the implementation approach of current model provides necessary facilities to extend in decided area of improvement this research will be focused on stabilizing and improving current model. Otherwise research will be carried out using a new approach of implementation and a more generalized model will be implemented.

- Critical review of existing approaches which tries to enhance the real-time simulation of environment-sensitive plants especially climbing plants.
- Critical review of computer graphics techniques and bio-mechanics to simulate realistic environment sensitive climbing plant behaviour in real-time.
- Design and develop an improved model to simulate realistic environment sensitive climbing plants in real time.
- Proper evaluation of the synthesized model by comparing it with real-world climbing plant behaviour considering both visual and physical realism.
- Production of final thesis including all the above findings with Critical review and discussion of the proposed model."

1.2.3 Research Questions

Though there are many sophisticated climbing plant simulation models there is clear absence of climbing plant behavioural simulation models based on bio-mechanical behaviours. All most all the currently available models were designed to achieve visually pleasing result efficiently rather than having modelled the bio-mechanical behaviour. While these models are capable of handling games and movies generally

no models are capable of having realistic simulation of fine details in close-ups or in point of interest scenes.

The model which was suggested by Gunawardhana et al [5] to fill the void in this area was implemented with a particle system using Position based dynamics. Even though the calculations of particle positions was boosted because of the Position based Dynamics the stability of the model suffered greatly due to it estimated calculation nature. In this research it is a necessity to make decision whether to stick with current approach or to come up with an alternative approach considering the extensibility of position based dynamics.

Wind, phototropism, weight of leaves and fruits and many external forces effects the circumnutation behaviour of a plant. There are available Simulation models which has taken these effects in to account and give visually pleasing results but none of them capable of simulating circumnutation behaviour.

The current model has ability to only interact with vertical cylinder shaped support structures but the circumnutation behaviour has to be generalized. In order to cater more general solution current model has to be extended to support differently shaped object as support structure. Smoothness, roughness of the support structure surface may also impact on the climbing behaviour but it won't be much focused on this project. Following questions have been selected as research questions in this project after considering above matters.

- Does current approach of implementation provide enough facilities to improve the performance as well as stability efficiently? If possible how it can be achieved?
- If not what are the alternative approaches? How to implement them?
- How to model the effects of external forces to circumnutation behaviour?
- How to extend the climbing model to tackle down the climbing on differently shaped supports?
- Can the taper function be generalized between plant species? If possible how it can be done?

1.3 Justification for the research

Researches in field of vegetation Simulation has been popular among research community especially because of the vast variety of applications it beholds. It can be seen that various setups of experiments carried out and cross disciplinary research papers have been published in this area. Twining plants has abstract mechanical

properties which can be applied in many fields such as robotics [2]. Currently high amount of investment have put on experiments on climbing plants, for example even in International Space Station (ISS) researches on effect of microgravity towards twining plants are being conducted [3]. Using a proper simulation model, experiments can be conduct in an inexpensive ways. In agriculture, to develop effective productive cultivation mechanism these type of research can provide lot of impact for scientific experiments of the field. In automation of immersive world generation these type of models are constantly used. Moreover, these type of models can improve the 3D modelling and animation pipelines in the industry.

However there is a lack of climbing plant behavioural simulation models. Currently available models are not capable of handling bio-mechanical behaviours. Though they can be used in applications such as games or movies as long as they are not in point of interest. This projects targets to fill the void in above area.

1.4 Methodology

The main aim and objective of this project is to synthesize a generalized model to simulate biologically and bio-mechanically accurate climbing plant therefore it is extremely necessary to observe the nature. In order observe the Nature an experiment will set up in controlled environment and it will be observed closely and data will be recorded throughout the entire time period. A fast growing plant species such as Yard Long bean (Vignaunguiculata) will be selected for the experiment so the experiment can be carried in a short period of time. As the domain experts' observation and knowledge will be extremely helpful to come up with premises and logic to develop a model observation through literature will be carried out. As this project is an extension of last year project the resulting model of last year attempt will be observed closely.

As the next step, based on the observation logic and premises will be derived. On top of derived logic and premises a model will be formulated. These models may be based on various approaches to achieve different aspect of goals.

Next the models which were formulated in previous step will be implemented using suitable technology. Technology which the model will implemented on will be decided on considering complexity of implementation, extensibility of implementation and time allocation for the implementation.

Thirdly the implementation of formulated model will be evaluated regarding two main factors. Realism of the implemented model will be evaluated by comparing the newly developed models with related models and observations and data gathered from experiment setup. Performance of the proposed model implementation will

be evaluated using profiling. Finally after thorough evaluation a conclusion will be drafted. Figure 1.1 shows a high-level diagram of the research methodology.

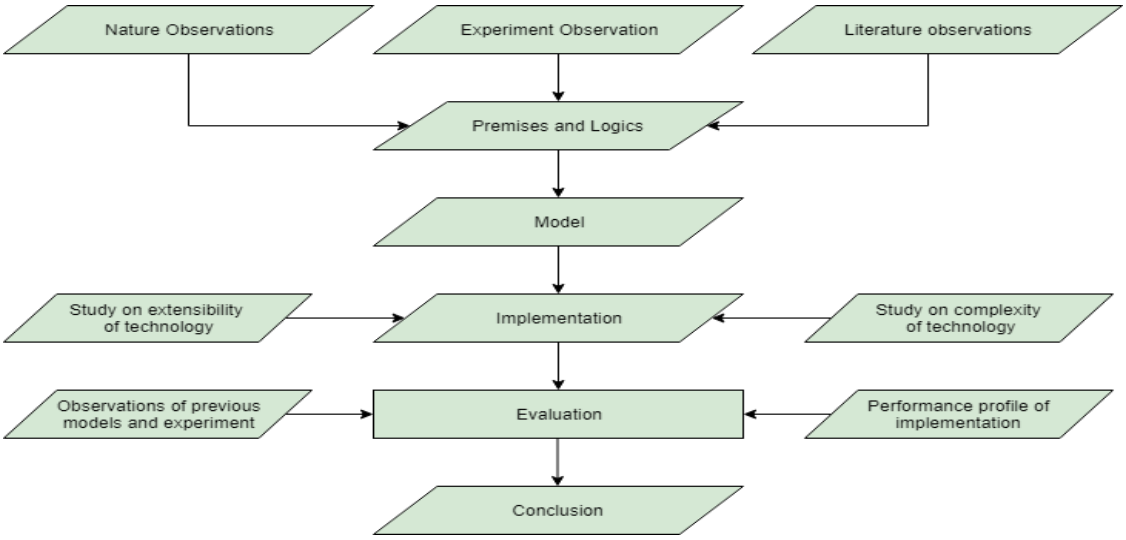

Figure 1.1: Research approach

1.5 Outline of the Dissertation

The first chapter of the dissertation will outline the background of the research area as well as research questions, aims, methodologies and scope of the research. Chapter 2 of the dissertation will critically review the research area and will identify the research gap clearly, further it will discuss the importance of the problem using literature as well as possible avenues for solution to the problem. Chapter 3 will focus on the research design. Further it will describe evolution of the design and design choices made during the research process. Chapter 4 will explain details about implementation and challenges occurred during the implementation. Chapter 5 will discuss about the evaluation of the proposed model and results. Finally, Chapter 6 will provide discussion and conclusion about positive and negative outcomes of the results, as well as limitation and future avenues for the research.

1.6 Delimitations of Scope

A complete simulation of twining plants consist of many complicated factors, due to the time limitations scope is narrowed to an achievable scope. Features of complete twining plant simulation system divided as scope and out of scope are shown in the Table 1.1.

Scope	Out of scope
Growth of the stem	Branching
Stem growth against gravity	Stem growth towards light
Bending and twisting of the stem	Growth of the leaves
Collision handling of the stem	Aging of leaves, textures and materials
Differentiation of Girth along the stem	Respond to external forces
Growth of leaves	Friction of the stem. Braking of the stem.

Table 1.1: In scope and out of scope features

1.7 Conclusion

This chapter laid the foundations for the dissertation. It introduced the research problem and research questions and hypotheses. Then the research was justified, definitions were presented, the methodology was briefly described and justified, the dissertation was outlined, and the limitations were given. On these foundations, the dissertation can proceed with a detailed description of the research.

Chapter 2

Literature Review

2.1 Introduction to climbing plants

Climbing plants have been an attraction for a number of biologists and naturalists for centuries yet Darwin was the first to classify and document climbing plants' behaviors. Darwin [6] divided climbing plants into three main classes namely

1. Spirally twining plants
2. Leaf climbers
3. Tendril bearers

Other than these three great classes another class called "hook and root climbers" was proposed to occupy less remarkable (it was considered less remarkable as such plants exhibit no special movements) types of climbing plants. Darwin studied the spirally twining plants closely and he coined the term "circumnutation" to describe rotary movements exhibited by twining plants. Later it was found that circumnutation is ubiquitous among all plants yet particularly beneficial for spirally twinning plants as it plays a major role in the support structure finding process.

How the "circumnutation" behavior occurs was a highly debatable topic until the late nineteenth century. There were two main models of to explanations. The first was the Darwinian model "the internal oscillator Model" which defined the circumnutation as an internally driven process. The second was the "gravitropic response-with-overshoot model" which defined circumnutation as a process that occurs due to nutational oscillations by a gravitropic overshoot process. Brown [12] proposed a model to explain circumnutation which was supported by the observation of experiments done in microgravity. The model had all the properties of Darwin's internal oscillator. It did not depend on gravity but showed that it can be heavily influenced by gravity.

Biology describes two types of movement in plants. The first is "tropistic" which stands for growth responses of plants that result in curvatures of plant organs toward or away from certain stimuli. The second is "Nastic" which stands for the movements of a plant organ in a direction that is not specified by the direction of application of the stimulus. Brown[12] classified above-mentioned circumnutations into nastic Movements.

Tropistic movements of the plants are governed by a number of tropisms[13]. Geotropism[14], phototropism[15], thigmotropism [16], hydrotropism [17], heliotropism [18] are a few examples from many types of tropisms. Heliotropism is a behavior that can be observed mainly in plants like sunflower not much in twinning plants which is the focus of this research. Hydrotropism is one of the few phenomena which govern the way the root grows. As this research is focused on the above-ground part of the simulation effect of hydrotropism is not taken into account.

Geotropism (gravitropism) is a two-way phenomenon. Negative gravitropism explains the behavior of downward growth of roots and positive gravitropism explains the Upward growth of plant parts, against gravity. In plants, amyloplast displacement in specialized cells appears to be the primary gravity-sensing mechanism [14]. Endodermal cells of shoots are believed to be the statocytes[19] (Statocyte is a specialized cell that contains amyloplasts to sense gravitropism). This. through the amyloplast sedimentation gravity stimulation is perceived and a signal is generated in endodermal cells. These signals (biosynthesis enzymes) is how the plants perceived gravity.

In simple terms, phototropism is the growth of an organism in response to a light stimulus. Phototropism has been studied by poets, philosophers, artists, biologists, and scientists for centuries[20]. The key idea of the phototropism is that when the light shined on plant surface it creates an imbalance of phytohormone "Auxin" in such way the darker side gets a higher concentration of auxin. This higher concentration led to the elongation of cells in the darker side which causes the curvature of plant stem towards the light. The mechanism of how exactly this auxin imbalance is caused is a highly active research area. From all the models Cholodny–Went model has been one of the most criticized models yet recent evidence starts to provide further support for "Cholodny–Went model"[21] [22]. According to Cholodny–Went model auxin is synthesized in the coleoptile tip, which senses light and sends the auxin down the shady side of the coleoptile. This causes an imbalance of auxin concentration and it led to asymmetric growth resulting in shoot bending towards the light source.

Thigmotropism is a directional growth movement that occurs as a response to a touch stimulus. Thigmotropism also can be explained by the behavior of auxin concentration. Reinhold et al [16] found that treatment with IAA (IAA is the most

common, naturally occurring, plant hormone of the auxin class) can replace the tactile stimulus.

Like the thigmotropism, there is a nastic movement called thigmonasty that explains the response of a plant or fungus to touch or vibration. The difference is that thigmotropism occurs in a direction determined by the position of where the plant was touched, while thigmonastism occur independently of the direction of the stimulus. Thigmonastic movements are caused by deferentially changing cell turgor within a given tissue, not by asymmetric growth rates.

The underlying physics of a twinning plant was explained by the Goriely[9] by modeling the stem close to the apex as a growing elastic rod. Goriely's analysis showed that a mechanical model for anchoring and attachment based on rod theory can provide a simple explanation for the limitation of twining plants to wind around thick supports and for the role of friction to boost the plant's ability to achieve vertical growth. Though Goriely's model could explain how tension is generated it did not address the fact that after the establishment of stem on the pole it may adapt its intrinsic curvature due to stress and contact.

Further explanation of underline physics was given by Isnard et al[23]. The analysis was conducted by measuring the squeezing forces exerted by the twinning plant "Dioscorea bulbifera" while following its growth using time-lapse photography. It was shown that the development of the squeezing force is accompanied by stiffening of the stem and the expansion of stipules at the leaf base. A general mechanism for the generation of the twining forces was pointed by their model and it was able to explain how the twining plants ascend smooth supports without the use of adhesive or hook-like structures.

2.2 Related works

The earliest approaches in visual simulations of plants employees fractal-based techniques. Bloomenthal[24] employed fractal-based techniques to generates plant topology, used generalized cylinders for plant geometry, and images for textures of bark and leaves. Many fractal based approaches and interactive techniques have been introduced later. The main disadvantage of fractal-based techniques is that they do not adapt to the environment.

In the literature on climbing plant simulations, three widely used approaches can be found. Namely,

1. L-systems

2. particle systems

3. Rod based models

L-system[25] [26] or Lindenmayer system is a parallel rewriting system and a type of formal grammar. It rewrites a starting symbol to a sequence of modules by expanding it using a collection of production rules that are interpreted by a geometric turtle. The path that the turtle travels in space and the actions it performs correspond to the plant geometry. Later, an improvement to the L-system called Open L-systems[25] [27] Introduced the control of the rewriting process also by external conditions. This allowed Thigmotropism, Phototropism to be modeled using L-systems. L-systems were further improved to have interactive capabilities. Power et al [28] modeled stiffness and torsion used them with inverse kinematics to achieve manipulation of plant interactively without sacrificing their botanical accuracy. Though L-systems is a powerful technique that has many capabilities it has drawbacks as well. The complexity of L-systems is usually higher than other approaches as it requires a number of production rules which affect each other. This makes the prediction of the final shape hard in L-system.

The second widely used approach is the particle systems[29][30] approach. In these systems, particles have the capability to sense the surrounding environment and perform certain actions. The proximity of supporting structures is detected using ray-casting and the plant tries to keep close. Simultaneously, the amount of incoming light is evaluated and the plant tends to leave shadowed parts of the scene. Initial particle systems use voxel space to enhance collision detection and proximity of objects where the later particle systems use ray-casting to achieve it. AMNP [31], another particle-based approach that uses biologically based intelligent particles. Each and every module of the simulated plant has specific behavior assigned that depends on its internal state and external conditions. As an example, a bud contains an apical meristem and can grow only if it is fed enough by soils from the root and by CO2 from the adjacent leaves. Further extended version of particle-based system models plants as systems of oriented particles that are able to sense the environment and compete for space. In this approach, particles move to the best possible locations using a directed random walk. They have used the phenomenon of traumatic re-iteration for critical cases. If there is no place for further growth possible the particle dies, but before it sends a signal that is propagated down in plant structure. This signal activates the closest possible sleeping particle that takes its job. They use an associated voxel space for collisions and space occupancy detection as well as for evaluating the illumination of the plant organs.

Recently Hadrich et. al [1] created a particle-based simulation. There, a plant is represented by a set of connected anisotropic particles that respond to the sur-

rounding environment and to their inner state. Each particle stores biological and physical attributes that drive growth and plant adaptation to the environment such as light sensitivity, wind interaction, and physical obstacles. This representation allows for the efficient modeling of external effects that can be induced at any time without prior analysis of the plant structure. In the framework, they exploit this representation to provide powerful editing capabilities that allow editing a plant with respect to its structure and its environment while maintaining a biologically plausible appearance. Moreover, they have a couple of plants with Lagrangian fluid dynamics and model advanced effects, such as the breaking and bending of branches. The user can thus interactively drag and prune branches or seed new plants in dynamically changing environments. Their system runs in real-time and supports up to 20 plant instances with 25k branches in parallel. Even though this model gives the real-time instructiveness and ability to create a variety of climbing plants, it does not provide the same complexity as state-of-art work in the field and this system cannot handle the effect of global control. Simply this model is not capable of handling the effect on entire plant when some local interaction happens. Moreover, this model does not provide a biomechanically plausible simulation of tree behavior.

Developing a generalized framework to simulate climbing plants with biological and biomechanical is not practical due to the complexity of a vast variety of behaviors. Because of that scientists have to choose the best possible categories of climbing plants. When selecting these categories they have to consider different aspects such as scientific value, best possible representation, and the possibility of application to other scientific domains. Among other climbing mechanisms, twiners and tendril bearers are exceptional because of their special biomechanical properties which can be applied in many fields. In the twining plants, the stem itself is the attachment structure, this characteristic simplifies the simulation models because it reduces the need for third party attachment modules. Most interesting that twining plants have over an extensive apical region the stem is flexible and undergoes broad circumnutation movements while farther from apex [6], the stem forms a strikingly uniform helix that can squeeze a support with considerable force.

To simulate twining plants and plants with tendrils a procedural approach can be used. Wong et. al [11] proposed a set of techniques for modeling the twining behavior of stems and tendrils as well as coiling pattern of tendrils. Also, a technique to overcome the intersection problem for stems and tendrils twining on host. They have considered numerous factors, such as positive phototropism and negative geotropism, for growing climbing plants. To generate a climbing plant model, they seed the root of the main stem at the ground level and grow the main stem gradually. While the main stem grows, stem nodes are added periodically.

After a stem node is created, a new stem, tendril or leaf may be created. If a tendril twines around a host object, it also coils and its axial length gradually decreased. Thus, the stem segment near the tendril is pulled close to the host object. However, they do not model this process exactly but move the stem close to the host. The shape of a part of a tendril is helical after it coils. Therefore, they apply helical models to model the coiling patterns of tendrils. In their implementation, they have considered objects that can be approximated as a set of capsules. They use bounding volume hierarchies (BVHs) and voxel structure to accelerate collision detection. The BVHs are used to bound objects in the environment and the voxel structure is used to store data on climbing plants. However, this model has many limitations. The main limitation is they have not considered any biomechanical behaviors, because of that simulation shows lot of artificial behaviors. Further, this model does not completely resolve the intersection problem among the stems.

There is a different type of general rod model shows the capability of simulation of climbing plants. For example, Casati and Bertails-Descoubes [32] and Integrated their super space clothoid rod model into Goriely and Neukirch [9] model replacing it's naturally curved Kirchhoff rod. They observed that their model gives more realistic simulation than previous works. It is clear that integration of better rod models improves the realism as well as the performance of twining plant simulation modes. Rod based research area is a very active research area that is highly concerned with physical realism as well as computational efficiency. For example, in a rod based hair simulation, a large number of strings are rendered in the same scene [33]. There are many rod models such as cosserat rods, position-based dynamics into cosserat rods[34], superhelix models [23], super space clothoid [32] models can be integrated to twining plant simulations to achieve more realism as well as real-time capabilities.

2.3 Previous models by Gunawardhana et al

Gunawardana et al [5] purposed four models to simulate climbing plants. The first two approaches were particle-based approaches while third and fourth models employed a rod-based approach.

The first model which they proposed was named "Cylindrical Chain". It contained a set of cylinders distributed along a spline curve which represents the stem. Each cylinder was set with slight relative rotation offset and different mass values. The stem was designed to simulate circumnutation movements based on a pure weight driven approach. The complexity of collision handling due to cylindric colliders, Modeling of the stem was not focussed, unnecessary complexity of

calculations due to divided weight allocation are some of the drawbacks in this model.

These drawbacks led them to move into new a model called Beads and String Model. This model consists of a set of spheres distributed along a spline curve. This model mainly is driven from relative motion. Each and every bead move relative to its parent bead which makes this heavier in the calculation. The preliminary implementation (without any physics aspects such as inverse-kinematics, twisting and bending implemented) of this model revealed that increasing the number of instances decreases the performance drastically as the complexity and the high number of computations per frame. This led them to move towards a rod-based approach that efficiently handles earlier mentioned physics aspects.

The third model was created using cosserat rods with the intention to capture all the elastic energy contributions. Stretching and bending modeled by the deformations of the space curve. Twisting was modeled by the rotation of a material frame associated with each point on the curve. As the integration of growth, the formation of structure, simulation of radius along the stem, branching, integrating biological properties such as gravitropism and phototropism is challenging just by using cosserats rods purely again model was transitioned to a hybrid approach.

Mixed usage of both particle and rod approaches gives flexibility to the model to simulate biological and mechanical behaviors such as growth, circumnutation, structure formation, bending, twisting, and response to the environmental aspects. In this approach, the Position-Based Dynamics implementation of cosserat rods is used. Due to the usage of Position-Based Dynamics positions were used Instead of calculation using forces. Position constraints were solved by a highly optimized constraint solver. The effect of external forces is handled by using force calculations in every frame and the PBD framework handles internal collision using the same constraints. This model was the final model introduced by Gunawardana et al while it had some benefits and drawbacks. The hybrid model was computationally efficient due to the usage of PBD framework and Both support finding and twinning mechanism were simulated by same mechanism in model. Inability to extend plant stem more than , No Fixing of the stem is simulated, Lower stability of stem in simulation, no support for branching are some of the drawbacks in this model.

2.4 Position-Based Dynamics

Simulation of dynamics has been dominated by the force calculation and time integration approaches for decades. Muller et al [35] proposed an alternative approach which utilizes positions and set of constraints. Although impulse-based approaches

were used rarely in the industry this was the first publication that introduced a way to manipulates the positions directly. It is fair to say that the key idea of position-based dynamics has been there for a while as there were a number of applications that utilize the positions and constraints before the publication yet this was the first to utilize the idea of position and constraints to this extent. The key advantage of the proposed approach is its efficiency and its ability to manipulate the objects during the simulation directly using positions.

The main differences between the suggested approach and traditional approaches are the extensive use of constraints and constraint projection instead of force calculation, laws of motion and time integration. The suggested framework consisted of two major components namely vertices and constraints. Vertices comprised of the values mass, position, and velocity. The set of constraints may be comprised of a number of different constraints types such as Distance Constraints, Volume Constraints, Bending Constraints, etc.

Suggested algorithm (refer Appendix C.2) composed of several sections. First, the initialization where the values of all vertices get initialized. Right after the initialization, the main loop begins. As the first step in each iteration, the effect of forces that were not simulated inside the position based dynamics were calculated and their effect was represented by the change of velocities. Then a velocity dampening is carried out to smoothen the simulation. Next, proposed positions for vertices are being calculated considering the velocities. Based on earlier calculated proposed positions new collision constraints will be generated and they will be projected in the solver for a given number of iterations. This constraint projection is carried out in a Gauss-Seidel-type iteration manner. Each step in an iteration affects the next step. The order of constraint projection is crucial as otherwise, it may tend to have an oscillation effect. Using the updated positions (positions after the constraints projection) new velocities will be calculated and finally, the positions will be updated. To conclude an iteration a step called velocity update is being carried out. In this step, the velocities are dampened and velocity directions are updated to simulate the effect of friction. Till the end of the simulation above all specified steps will be carried out in loop.

Chapter 3

Design

3.1 Introduction

This chapter explains the design of the experiment and solution to the research problem. The design choices and the journey toward the current solution will be discussed and the reasons behind the choice of research methodology will be explained.

In this research, no specific species was targeted. To come up with a general solution various types of twinning plants were observed and a thorough literature review on plant general bio-mechanics and biological phenomena was conducted. To verify the knowledge and to do a final evaluation, an experiment was planned and carried out. this was done in two phases. In the first phase, an experiment was conducted under controlled conditions to confirm the biomechanical behaviors and to collect data about the plant growth in the earlier stage (first weeks). In the second phase, another experiment was conducted outdoors in natural conditions to gather data about twinning behavior and support finding mechanisms. these details related to experiments will be discussed in the first half of this chapter

The design process of the solution was carried out throughout the entire time period in an iterative manner. First, nature, related works, and biological explanations were observed and analyzed. This generated a few ideas of an approach to the simulation. These ideas were implemented and compared with nature and related works. After the comparison, ideas were adjusted and adapted or fully thrown away. This process was continued for several iterations and the first model to the final model the whole journey will be explained in detail in the second half of this chapter.

3.2 Design of Experiment

A number of problems had to be addressed in the design of the experiment. As discussed in the introduction experiment was carried out in two phases. The targeted data was video footage of the entire growth of the twinning plant. The first problem was the selection of suitable species of plants. The selection of a few species of twinning plants would have been better instead of selecting a single species. There were few problems with selecting more than one species of plant. First was the resources were limited. Throughout the whole time period, two cameras were available for the research. And footage of a day was around 50GB. The second was the limitation of time. For a single experiment, a time period of around two months was needed. Therefore conducting experiments many experiments one after another was not feasible. Therefore a Long bean species called *"Vigna unguiculata"* was selected as the targeted species for the experiment. There were a number of reasons for selecting this species. First, it was easily available on the market. Second, it has a fast growth cycle. Third, it was tolerant to changes in environmental conditions. Several seeds were planted as relying on one plant's data was not enough for making a judgment. But as the plant grows a number of plants had to be limited to two as camera setup couldn't capture more than two grown plants clearly.

A number of factors were considered selecting a camera setup for the experiment. It was decided to use two cameras; one from the top and the other from the side. As the Footage was captured from two angles, the data was analyzed three-dimensionally. The choice of the cameras was the next issue. Daily camera storage swapping was not a feasible option. Therefore the only option was to have to rig up a CCTV setup. With 500GB hard drive, it was able to hold ten days worth of footage. Backing it up and processing it was the next question. Having 50GB a day and collecting data over 2 months was not feasible. As a solution for this, a frame reduction technique was employed. Reducing frames to 50:1 ratio once and recursively another time two sets of video footage were created. As the plant movement is rather slow this frame reduction helped not only to reduce the size but also to enhanced data set as it was fastened up. This reduced one month 24/7 worth footage to around 25 minutes. To do the frame reduction and concatenation a command-line tool called "FFmpeg" was used. The full script is available at the appendix C.1.

3.2.1 Phase 1: Experiment in a controlled environment

In the beginning, No phase two experiment was planned. This experiment was planned to cover the whole growth. But the unexpected conditions that were aroused in the middle of the experiment lead to a second experiment.

This experiment was carried out in a closed environment where no wind conditions or light conditions affected the bio mechanisms. Here the undisturbed bio mechanisms and biological behaviors were planned to be observed. A small indoor closet-like space was chosen and the light was artificially provided. Choosing a suitable light source was the first problem that was encountered during this experiment. Light types used for indoor cultivation were studied. It was found out that there were three types of light that were used mainly for indoor cultivation. First was sodium vapor lamps which were expensive. These produced a lot of heat there for installing a ventilation system was a must. The second was a mixture of red and blue LED bulbs. This was the cheapest yet most efficient. The problem was due to Purplish hue color dim light which these provide, capturing video footage with low-end cameras was difficult. The third option was to use 21000K color temperature ranged CFL Light which was almost like Daylight. no ventilation system was needed as these didn't produce much heat and it solved the problem with capturing video footage as well. Therefore a 50w 21000K CFL was used as advised for small scale indoor cultivations.

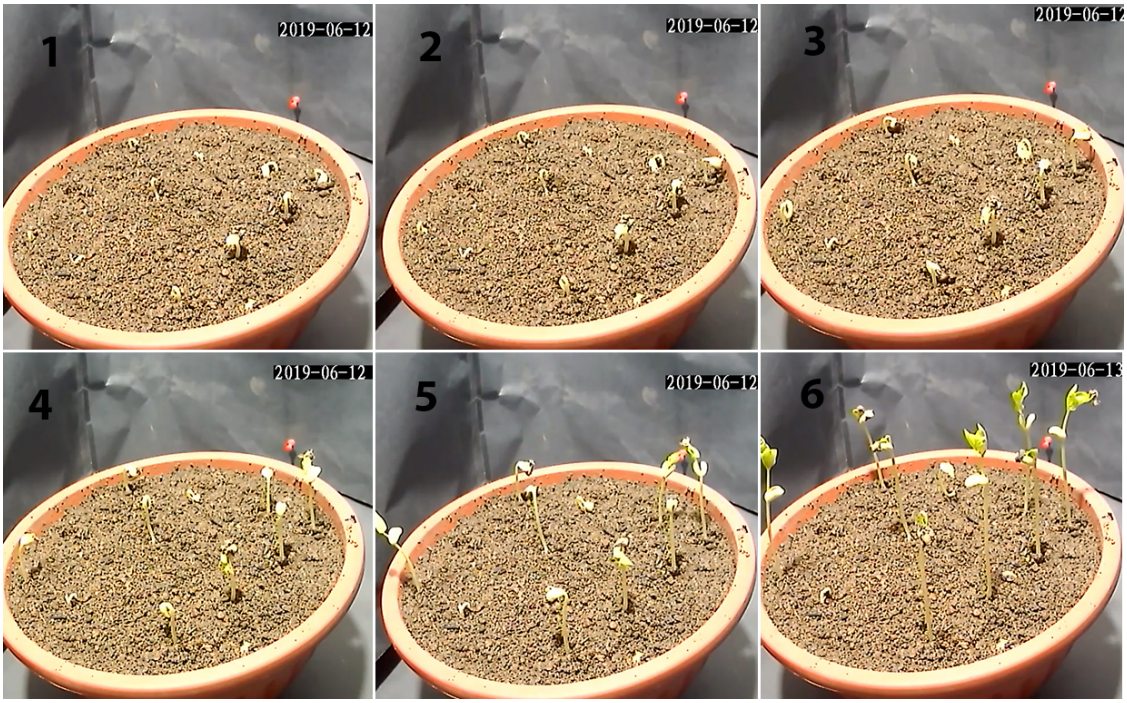

Figure 3.1: Cotyledons rise from ground

In this experiment, it was observed that some plants were nutating clockwise

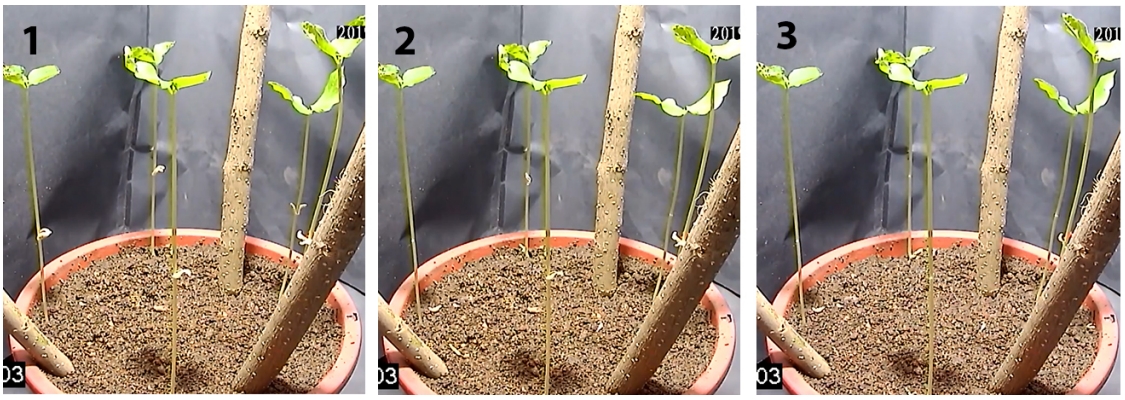

Figure 3.2: Withering cotyledons

while some nutating anti-clockwise. In Figure 3.1 how the cotyledons have risen and in Figure 3.2 how they slowly withered out after using up all stored food reserves can be clearly observed. Plants were nutating from the beginning. Then cotyledons were fallen and after a while plants gradually stopped circumnutation. For few days plants were almost still and no visible growth was observed. The plants were very slim and tall and after a while, all the plants were fallen to the ground and they were almost died out. This kind of behavior was first observed by Darwin[6] but he recorded that the plants had risen again from the tip as the Figure 3.3 depicts after a while yet such observation was not observed in this experiment except all plants were slowly withered out. With this experiment, the twinning behavior was not able to be observed as planned. Therefore the second experiment was planned to conduct outside under natural conditions.

Figure 3.3: How Plant fell after circumnutation rise again at the tip area

3.2.2 Phase 2: Outdoor experiment

This experiment was conducted to observe the twinning behavior of plants. As the failure of the earlier experiment to grow enough to a stage where twinning behavior is visible this was planned to conduct outside where the plants were grown freely

exposed to natural sunrise, rain, and breeze. A few support structures were planted earlier and faster growth of plants was observed. Plants were much greener and were thicker than the ones in the earlier experiment. A comparison between two experiments at the stage where cotyledons wither out and drop is depicted in Figure 3.4.

Figure 3.4: Comparison between plants in two experiment phases

Figure 3.5: Elongating shoot

Figure 3.6: After leaving support structure plant start circumnutation again

The first leaf set always consisted of two leaves and each leaf set after that was a set of three leaflets. These Leaves were connected to the plant using a single straight stem. Each plant grew just as a standalone tree until it bears around four leaf sets. No Support structure searching bio mechanism was observed during this phase. After a while, as Figure 3.5 shows a long shoot came. In this shoot, a slower leaf growth was observed. while leaf growth was slower than earlier the shoot was elongating faster and it was nutating rapidly. In this stage around one cycle per hour circumnutation speed was observed. The circle was around 1ft radius. This behavior was observed until the shoot was properly hit on target. The word properly was used here as hitting on a support structure slightly just the tip of the shoot was not enough to stop the circumnutation. Once the shoot properly hit on a support structure it started twinning around that support structure. From two

plants which were planted for the experiment one immediately start twining around straight support structure while other plant started to twine around a camera cable. As the one which twinned around camera cable wasn't observable with the camera setup that shoot was cut away while other was observed further. The plant which twinned around the straight support successfully twinned around and reached the top of the structure. As Figure 3.6 depicts Once it hit the top of the structure it immediately started the circumnutation behavior. This time there were no support structures in near three feet radius. The shoot was elongated to limit where the circumnutation circle became almost a 2ft radius. And suddenly the shoot was fell down till it was fully horizontal. it kept falling for around three and a half hours until it makes forty-five degrees downward angle from the horizontal plane. Afterward, as it was captured in Figure 3.8 it started to grow upward direction from the tip and after a while, circumnutation behavior was started again as Darwin[6] observed and recorded. This time a slanted support structure was placed near and shoot hit upon the support structure and started to twine around it. Plant twinned around the support structure is portrayed in Figure 3.7. After this, the Experiment was concluded as it provided a sufficient amount of information.

Figure 3.7: Plants twined around support structure in experiment

Figure 3.8: Once the stem fell it starts circumnutation in tip area

3.3 Design of Solution

In this section, the journey towards the final solution and design of each model will be discussed. The process of designing a general solution was a long path. Several models have been designed, implemented and tested out. Some models were totally rejected and some models were modified and reused in and iterative manner.

3.3.1 Attempt to improve Gunawardhana et al Position based dynamics model

Last year Gunawardhana et al made an attempt to simulate twining plants. Several models were suggested and tested out and a solution that employs position based dynamics was suggested as the final model. As this research was originally started as an extension the first step was to critically analyze their final model and try to fix its problems

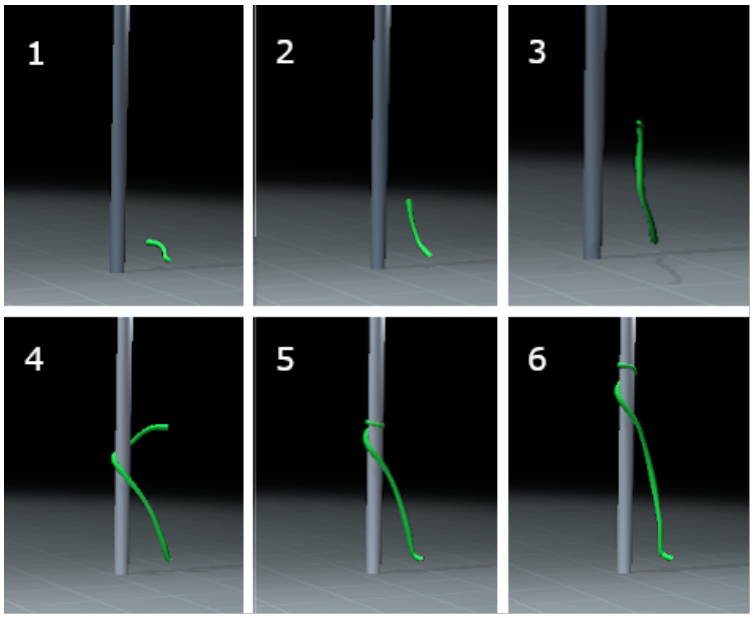

Figure 3.9: Outcome of final PBD based model by Gunawardhana et al

3.3.1.1 Critical analysis of Gunawardhana et al final model

For the design of the final model Gunawardhana e al has used an implementation of and position-based dynamics rope model called OBI rope. This implementation model the rope as a collection of particles where each particle is connected to neighboring particles with a set of constraints. Figure 3.9 depicts the snapshots of their suggested final model running for 20 minutes. The snapshot has been

taken in two minutes intervals. After analyzing Gunawardhana's PBD based model following ten limitations have been identified.

1. **Plant stem is not 100% extensible:** Gunawardhana's PBD based model has a limitation such that it cannot be extended beyond two rounds around a support structure. According to their explanation this problem occurs due to lack of friction between the stem and support structure. A solution to this problem was not given in their publication and this limit exists in every scenario except the ones where the plant growth was manually guided.

2. **Fixing of the stem is not simulated :** In Twining plants, some instability and some oscillation movements can be seen towards the tip of the stem in the twining stage in nature. Yet the lower parts of the stems have fixed anchored positions and shows no movements except the growth of stem and the movement and growth of the leaves. These observations were clearly visible on the experiment as well as in nature. But in Gunawardhana's PBD based model such behavior like fixing of the stem has not simulated at all.

3. **Branching is not supported:** As the Gunawardhana's PBD based model was modeled by using a PBD rope implementation it innately did not support branching. Branching introduces another level of complexity as the structure of constraints between particles has to be changed to support any kind of branching. Usually in Position based dynamics applications, a uniform structure can be observed and in fact, the original Position based dynamics framework has the limitation on bending constraints which limits that each edge can be shared by at most two triangles [35].

4. **No leaf growth and leaf movements have been simulated:** This part was not in the scope of Gunawardhana's research yet it's a fundamental phenomenon which has to be considered as the research focuses on simulating biomechanical behaviors

5. **Parameter Tuning is Complex:** just as PBD has advantages when it comes to simulating dynamics it also has its own unique sets of drawbacks as well. A major drawback of this approach is the complexity of tuning. The PBD has stiffness parameters which do not have any relationship to stiffness constants in real-world physics. Not only it doesn't have any real physics meaning other than how flexible a projected constraint is but also the effect of stiffness value changes drastically depending on the number of iterations. As a result, it is almost impossible to make simulation fast or slow depending on simple change of a parameter.

6. **Does not Support for different types of the support structure:** The implementation did not support differently angled slanted support structures, support structure with different thicknesses or differently shaped support structure. Gunawardhana's model only supported the straight cylindrical support structures with a certain radius. The main reason behind this issue was the lack of specific twinning mechanism in Gunawardhana's PBD model. Instead, both circumnutation and twinning behaviors were governed by a single mechanism which employees relative rotations.

7. **The earlier stage was not simulated:** As the literature, nature, and experiment suggested a twining plant does not display the twinning behavior in the earlier stage of growth through the circumnutation behavior was visible to a certain extent. In this stage the dominant biological bio mechanism behavior is the germination process. In simple terms, this includes growth of roots, seed coat getting separated, seed rise from the soil, Embryo getting out of seed withering of cotyledons and growth of proper leaves in order. These processes have not been simulated in Gunawardhana's PBD based model and in nature it takes a while for a twining plant to produce that long shoot which starts to show any sign of support structure finding mechanism. Gunawardhana's Model have not simulated this earlier stage which also displays some prominent biomechanical behaviors.

8. **Support structure leaving and circumnutation restarting behavior is not simulated:** Theoretically, this part would have been successfully simulated by Gunawardhana's PBD based model. Yet Testing whether it is possible to simulate this with just with the current model wasn't practical as it did not twine around a support structure more than two rounds. Not enough twinning leads to unwinding the twine.

9. **Stem falling because of overgrowth without support is not simulated:** The natural curve which can be observed in the elongated shoot was not simulated in Gunawardana's PBD based model. As the Whole simulation was modeled using the relative rotation based approach it did not simulate this natural curve visible in the nutating elongate shoot. When such nutating elongated shoot grows over a certain limit without a support structure it fell down to a certain level and start circumnutation from the tip area again. These phenomena have been observed in literature as well as in the experiment phase two. This biomechanical behavior was not simulated in Gunawardhana's PBD based model at all.

10. **The effect of geotropism, phototropism, thigmotropism towards the growth of a twining plant was not simulated:** In the first stage of growth, the effect of geotropism(gravitropism) and phototropism towards the direction of growth is very high. In late-stage, the geotropism and phototropism get overridden by thigmotropism. Gunawardhana's PBD based model failed to simulate any of the above-mentioned tropisms in any scenarios.

While there is big room for improvement in Gunawardhana's PBD based model the fact that their model is the first attempt to simulate the mechanical behavior should be acknowledged. Though it failed to completely simulate all the mechanical processes of a twining plant it succeeded in several areas and it laid a solid foundation to the area of simulation of twining plants.

3.3.1.2 Suggested Improvements to PBD based model

After studying the position based dynamics [35] following ideas were suggested to improve Gunawardhana's PBD based model. These ideas were suggested only by considering the theoretical aspects of PBD without considering the implementation to improve the stability of the model.

1. **Anchoring the lower part of the stem:** In position-based dynamics, pin constraints are used to anchor particle in a certain global position or to anchor to some relative position to an object which simulated outside the position-based dynamics. As it is explained in the earlier section pin constraint is achieved by setting the pined down vertices' inverse mass to zero. In this way, the pined vertices appear inside the position based dynamics as vertices with infinite mass therefore they become immovable by any constraint projection(not sensitive to internal forces). So the key idea was to change vertices' internal inverse masses dynamically in simulation.

2. **New vertices and constraint structure:** by this suggestion, it was expected to a much stable and robust simulation. In the position based dynamics rope model, the vertex and constraint structured as a single line. Instead of creating a single line structure what was suggested here is a three-dimensional structure. A three-dimensional structure was designed from clay and toothpicks as it helps to visualize the position constraints. The suggested structure was designed after carefully analyzing the limitations forced by the PBD framework such as limitation of bending constraints. This design of new structure allowed to simulate branching as well. To simulate the movements, dynamically changing position constraints were planned to use.

These two structural design changes were suggested after theoretically analyzing the position-based dynamics. Multiple problems and practical issues occurred while implementing the above mention design changes.

3.3.1.3 Rejection of position based dynamics

These two structural design changes were suggested after theoretically analyzing the position-based dynamics. Multiple problems and practical issues occurred while implementing. Here in this section theses, problems will be discussed and the reasons behind the choice of rejecting Gunwardhan's PBD based approach will be explained.

After the suggestions in the above section were made the first priority was to implement the dynamic pin constraint application process to the lower part of Gunawardena's PBD base model. OBI position based dynamics implementation was thoroughly analyzed but there were no functions to facilitate such behavior like dynamic pin constraints there for as suggested in Position based dynamics[35] implementing a mechanism to change the inverse masses of vertices was tried out as next option. Here, the real problems were arisen due to practical issues. Even though no restrictions were posed by position-based dynamics to change the inverse masses dynamically the OBI implementation of position-based dynamics did not allow such a mechanism. In the OBI implementation after the initialization of vertices, it passes the values down to a highly efficient solver which was implemented with C++ (No source code was available of this implementation). After this initialization process of the solver instance, no values of vertices can be changed. The method suggested by position-based dynamics to project constraints conserves both linear and angular momentum. This design choice of OBI can be understood as the changing inverse masses value affects each and every projection constraints. As there are calculations based on inverse masses these projecting constraints have to be recalculated in each step. This may introduce overhead in solver that handles the constraint projection and cuts down the efficiency of the whole position-based dynamics. Therefore the OBI implementation choice behind not providing the functionality to change inverse masses of vertices dynamically during the simulation can be understood. The only way to change the inverse masses of vertices dynamically during the simulation is by removing the old instance of constraint solver and initializing a new solver. Initializing a new solver every step not only costly wise time but it makes the stem behave in a very unnatural way. Since these practical issues raised during the implementation suggested improvements related to dynamics anchoring of lower stem was not implemented. As the next step to improve the Gunawardena's model the structure of vertices and constraint changed to

a three-dimensional structure. Though this new suggested structure supported the branching it increased complexity as well. The complexity of simulating behaviors with such complex structures by changing constraints was not suitable for realtime simulation as there wasn't a way to access and change the distance constraints dynamically. These distance constraints were set when initializing the solver and the OBI implementation of the solver did not allow to change distance constraints dynamically withing the simulations. It only allowed adding new vertexes and new constraints. Therefore this idea also was abandoned at the beginning of the implementation.

The fact that Gunawardana's PBD model did allow to change neither vertices' weight nor distance constraints dynamically during the simulation was one reason behind the rejection of Gunawardhana's PBD based approach. The use of the Gauss-Seidel method in the solver for a limited number of iterations tends to make the simulation keep oscillating. Such behavior is acceptable for a rope or a cloth in-game but surely it was not good for a simulation.this was another reason behind the rejection of the PBD approach. For the reasons that have been discussed here The PBD based approach suggested by Gunawardhana et al was rejected.

3.3.2 Revisiting bead and string model

After rejecting the PBD based approach everything had to be redesigned. The new 3D structure which has been designed to support branching was complex but the concept seems to be working. Therefore the idea to simplify the above derived 3D structure was always there since the rejection of Gunawardhana's PBD approach. To grasp the idea of how the circumnutation and twinning work the literature on biomechanism was closely studied. The main mechanism that drives the twinning

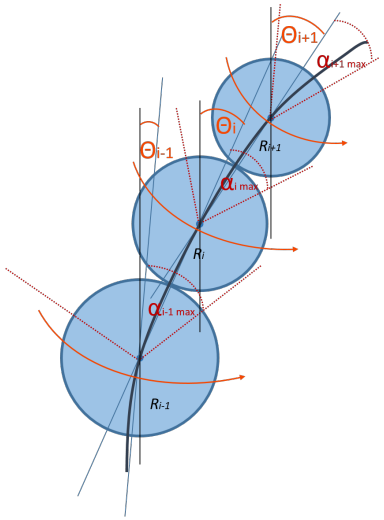

Figure 3.10: Bead and string model and it's relative rotation cones

behavior is thigmotropism. Thigmotropism is the directional response of a plant organ to touch or physical contact with a solid object. Both this directional response and circumnutation behavior is generally caused by the induction of some pattern of differential growth. The simulation of this differential growth is complex and unnecessary as the objective of this research is to make a visually and biomechanically accurate model not a biologically accurate model. So the first step was to design a simple model to simulate the biomechanics of thigmotropism and circumnutation without modeling differential growth. The thigmotropism which can be observed in twining plants can be summarised to one simple mechanism. By slowing the growth rate of plants stems in a contacted area and increasing the growth rate of the non contacted side it bends towards the support structure. This is the main mechanism that guides the twinning behavior. The mechanism which guides the circumnutation also is similar to the earlier mechanism. Here the cell growth rate is changed around the stem in round-robin fashion. This makes the stem of the plant to bend around the plant and it creates a circular or elliptical movement of the stem. In literature thigmotropism a has been simulated with particle systems[1] but these simulations had previous knowledge about the locations and distances of the support structure or the necessary information was gathered using ray casting techniques. As the main objective of this research is to make a biomechanically accurate model use of "previous knowledge" was not considered. Yet the particle-based approach idea seemed promising. This sparked the idea for a particle-based approach. Gunawardhana's bead and string model was a particle-based approach that simulated the stem as a set of beads connected to a string. Gunawardhana et al abandoned this model depicting two reasons. first, it was said to be computationally complex. Secondly, the visual representation of the stem in the model was inadequate. In Gunwardhanas bead and string model a "Centripetal Catmull–Rom spline" was used as the string where the beads were attached to. Beads were of different radiuses depend on the girth of the stem. In Gunawardhanas bead and string model, each bead was rotating in a cone relative to the previous bead. As depicted in Figure 3.10 the angle of the cone was different for each bead from bottom to top in decreasingly. The computational complexity of this model was high as in each frame for each bead a relative rotation had to be calculated sequentially (It was impossible to calculate positions parallelly as each bead position was depended on its the previous bead's position) and collision detection had to be done for each bead. In this model as the number of beads grows it became impossible to compute and render realtime without specific hardware. A "Centripetal Catmull–Rom spline curve" was fitted to bead position and it represented the stem. this curve fitting also required some computation yet it was inadequate to visually represent the natural curve of twining plants stem in

circumnutation behavior as well as twinning behavior. Though it was lacking in some areas Gunawardhana's Bead and string model also had many advantages such as,

1. **Sphears were used as particles hence efficient collision detection:** Collision detection can be done efficiently using distance calculations as the collider is a sphear.

2. **More control over a discrete plant section:** Here individual particles can be controlled without any effects on other particles. This makes it possible to to attach and other things such as leaves and make them move without effecting other sections of the plant.

3. **No restrictions to change the driving mechanism:** Though the Gunawardhana's bead string model was driven just by circumnutation mechanism there were not any restrictions to limit it just to a one driving mechanism.

4. **Its simple predictable model:** Unlike PBD based model with unpredictable behaviors and set of complex intersecting parameters this model gave full control to the designer.

Having absolute control over each particle and its behavior was the main issue in PBD based model. Gunawardhana's bead and string model could answer to these problems in PBD based model while it had its own weaknesses.

3.3.3 Rotational building block approach

The new improved version of the string and bead model was designed specifically to cater more control over the discrete plant section. Like in the bead and string model this model also was designed using a series of sphere colliders but the relative rotational motion in a specified cone mechanism was abandoned. Instead of complex relative rotational motion, beads were simply placed in the previous bead's forward direction at a certain distance called "inter-bead distance". In the scene graph, each new bead (unlike in previous model here all beads were of the same radius) was placed as a child object of a previous bead which makes each child bead positioned relative to parent bead positions. This parent-child relationship in the scene graph made the manipulation of a series of beads simple and efficient. With this model positioning of each bead was solely depended on the orientation of the previous bead. According to the change of orientation of the parent, each child branch is repositioned. As Figure 3.11 shows This model can be visualized as a structure made of building blocks where each block can be rotated in three

degrees of freedom independent of the previous block thus the name “Rotational Building block mechanism”.

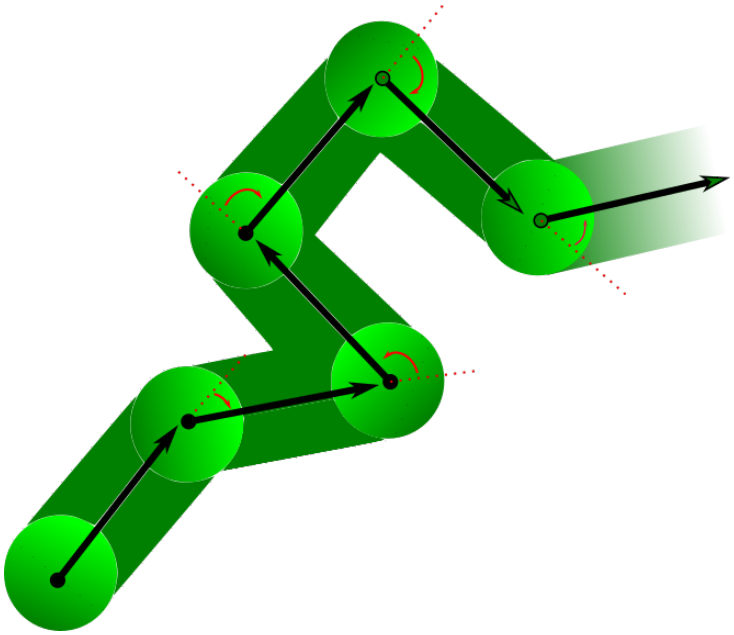

Figure 3.11: Rotational building block Structure: black arrows shows the current forward direction of bead, red curved array shows the change of orientation of bead after adding it

Separation of the structural mechanisms and policies is another major advantage of this new approach. In the Gunwaradhana’s bead and string model, both mechanism and policies were entangled. There the radius of sphere colliders simulated the girth of the stem and the bead’s rotation in cone relative to the previous bead was driving the circumnutation and twinning behavior. In the new approach, both growth and biomechanisms governing policies were handled by a separate automaton and the visualization of girth was handled at the tube rendering step. This made the new approach succeded in the separation of the structure and its mechanisms from its biomechanics governing policies. This made it possible to create a simple model with a layered pipeline an it will be discussed further at the end of this chapter.

3.3.4 State Machine with two states: Circumnutation and Twinning

The rotational building block mechanism was able to provide a robust yet simple structure to work on. Alone this rotational building block is just a structure and a mechanism to control it. The controlling mechanism had to be driven by something in order to simulate biomechanical behaviors. After a close study on the experiment

data and nature, it was realized that the whole set of biological behaviors can be represented by a state machine. Two main clearly separated states were identified in twinning plant biological behavior. First is the circumnutation state where the plant tip keeps rotating while growing to find a support structure. Initially plant was at circumnutation state and it switches the state to the twinning state to twine around. The Second state is Twining state where the plant grows around a support structure. Once it properly hit on a support structure. After switching to the twinning state if there is no support structure to twine around it will be switched back to the circumnutation stage. This state switching will be repeating till the end of the simulation considering whether earlier discussed conditions were met. Figure 3.12 depict this suggested state machine.

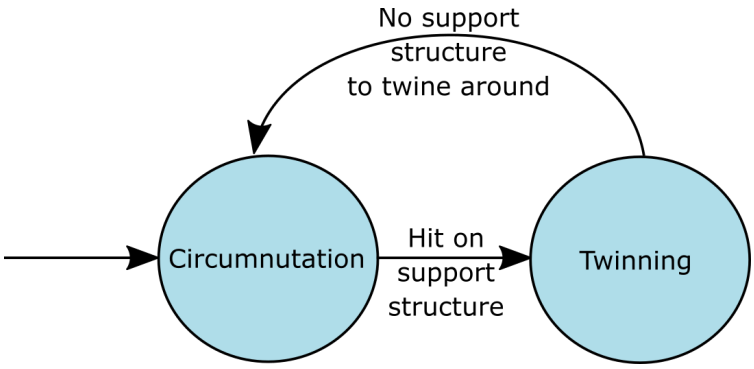

Figure 3.12: State machine with two states

3.3.4.1 Circumnutation state

As discussed earlier plant initialized in this state and it is kept in this state until it hit on a support structure. The rotation of the stem was simulated by simply rotating the lower bead in a given axis. Here the lower bead is the bead where the plant switched to circumnutation state. As the plant initialized in circumnutation state the first lower bead is the starting bead. But after exiting twining state the bead where the state switch to circumnutation state is the lower bead. As all beads were added as children beads of the lower bead all the bead added after lower bead also rotates according to it. This rotation is efficiently handled by a scene graph.

3.3.4.2 Twinning state

In nature, as a result of thigmotropism touching side grows slower than the other side this makes the plant grows towards the support structure which makes the twining behavior. Here this was simulated by moving the newly added beads towards the opposite direction of last collided bead's the collision normal. Figure 3.13 explain the twining mechanism and related vectors. This movement toward

opposite collision direction is done by rotating the previous bead as mentioned in the rotational building block structure. If the new bead did not collide with support structure after it moved for a while plant switches back to the circumnutation state.

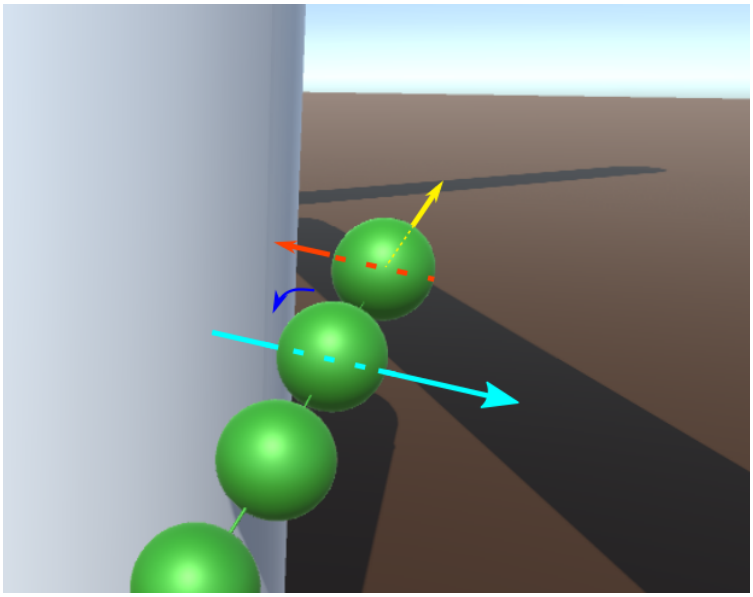

Figure 3.13: The twining mechanism: cyan arrow shows the collision normal; red arrow shows the opposite direction of collision normal; yellow arrow is the forward direction of last bead and previous to last bead; blue arrow shows the rotation of previous to last bead.

Though this simple state machine was successful to a certain extent there were several issues. The curves were not natural where the state switch happened. Sometimes parts of the plant stem went through the support structure.

3.3.5 Proper Hit mechanism

The earlier mentioned intersecting collider issue occurred due to the mechanism which was employed to simulate plant lengthwise growth. This was accomplished by adding new sphere colliders during the simulation to the tip of the stem at the last bead's forward direction in a given distance. When nutating plant stem hit with support structure just slightly usually the last bead forward direction is set towards the support structure. Therefore the next bead will be added in a position that resides inside the support structure's collider. This makes the stem go through the support structure. To fix this issue a new mechanism was introduced called "proper hit mechanism". It was modeled after natural twinning plant behavior which was observed during the experiment. In nature, A certain amount of plant stem has to collide with a support structure otherwise it kept on nutating ignoring the collision. Collisions of sphere collider in small cone area in the

forward direction axis as depicted in Figure 3.14 are ignored. Implementation of proper hit mechanism made the simulation more realistic as well as this solved the issue of intersecting colliders. Proper hit mechanism minimized effect of intersecting colliders and it made the plant having unrealistic curves. a simple mechanism to project intersecting beads back to the immediate surface of the support structure solved the rest of the issue completely.

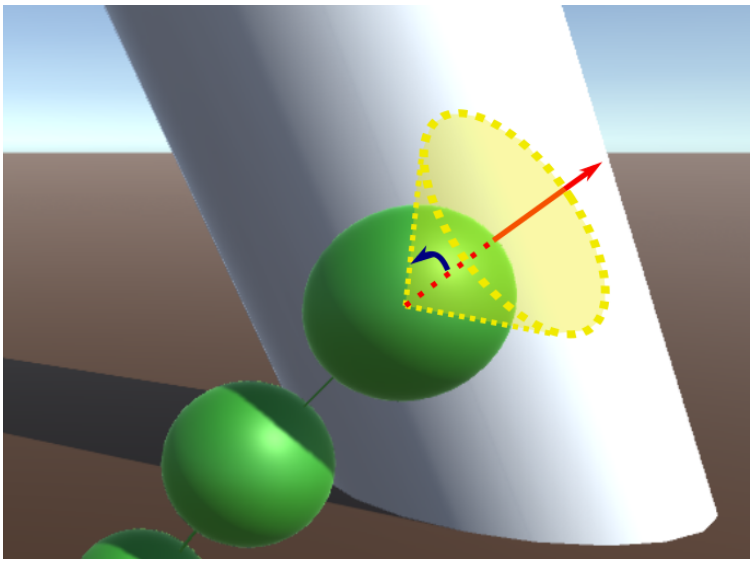

Figure 3.14: The Proper hit mechanisms: yellow colour cone is the collision ignoring area; red arrow shows the last bead forward vector; dark blue arrow shows the angle of cone

3.3.6 State Machine with Four states: added two intermediate states to prepare for transitions between major states

The result of the combination of the state machine and rotational building block was satisfactory to a certain extent this idea was extended further to make a more robust and visually pleasing simulation. In some scenarios, the curves of the stem where the state switch happened were not visually realistic. These anomalies have arisen due to the nature of the sudden state switch. Therefore some intermediate states had to be introduced between earlier suggested two states to smoothen the transition. Figure 3.15 portrays the new suggested state machine. These newly introduces two states smoothen the stem and prepare for the transition from one major state to another.

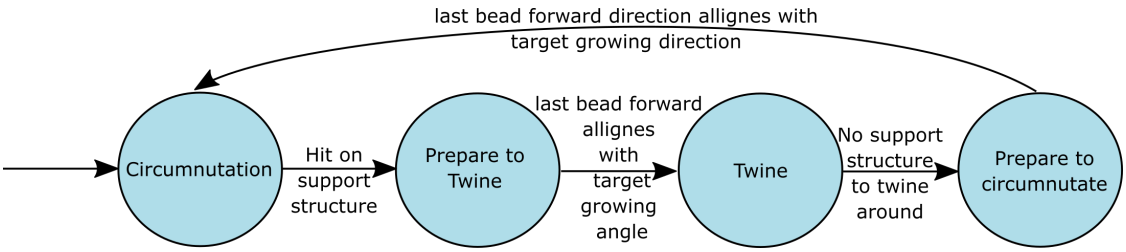

Figure 3.15: State Machine with Four states

3.3.6.1 Leaving Twining and prepare for circumnutation

At the movement where the twinning because it no longer has the support structure to twining around ends usually the last beads forward direction is set to some random direction. It the circumnutation starts right after that it makes the simulation very unrealistic due to the strange curves of the stem and abnormal division of nutating and stable part of the stem. To fix this, a mechanism of smoothened growth of the curved stem towards a direction dictated by gravitropism was added. This suggested mechanism of smoothening growth occurs without clear circumnutation behavior until the growth direction aligns (the forward direction of last added bead) direction specified by gravitropism. Once aligns the state is switched to circumnutation.

3.3.6.2 Leaving circumnutation and prepare for Twining

Once a nutating plant properly hit on a support structure state machine will transit immediately from circumnutation state to this state. The purpose of having this intermediate state between circumnutation and twining state is to smoothen the curvature of the stem to mimic nature and to simulate groping behavior which was observed in the experiment. The once the plant properly hit with a support structure it shows the behavior of grabbing and stumbling the support structure and some sliding along the structure up and down before it starts to the twine properly. This behavior makes the plant stem to have a stable curvature. Just as in the movement where the twining ends here also the growth direction has to be aligned properly with target growing direction. To fix the issue the same approach as the above state is used and other than that groping mechanism was added. The grouping mechanism will be explained in detail in chapter 4.

At this point, the basic structure and behaviors of the model were adequate to simulate a twining plant yet visually it was just a set of beads there for finding an effective way to render the plant stem was the next challenge.

3.3.7 physical and visual representation

The visual representation by a set of the bead placed at a certain distance from each other in line is the whole development of visual representation up to this point. It barely gave an idea of how the plant would have grown but it was inadequate in visual realism as a simulation.

Now that the key location where the stem goes through is represented with beads positions the first idea was to render a tube that goes through each and every bead's center. This was done by using a free open source tube generation script. Tube generation is done by rendering a tube between two given points to match given two radiuses this is done between all neighbor beads which makes a set of tubes with various radius. The growth of the girth of the stem is visualized by rerendering the tube with a greater radius. This makes the simulation of the growth of stem girth a separate independent component. Figure 3.16 shows a comparison between the physical and visual representation of the plant.

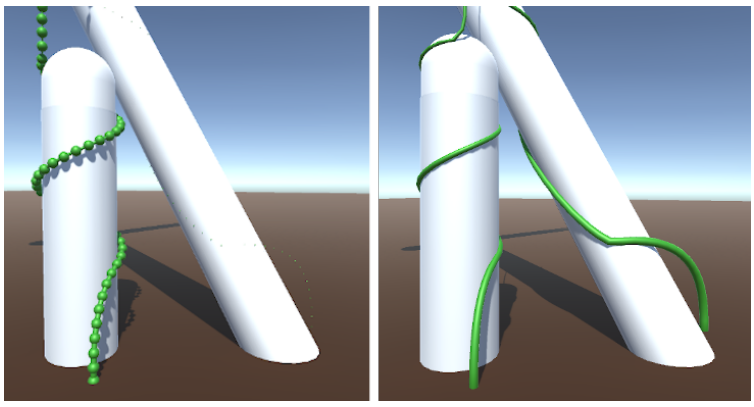

Figure 3.16: A Comparison between Physical and visuals representation

The design of this module is done as follows. A data structure is maintained to keep the bead positions and the girth of stem at the given position of beads in order. In fact, one could call this is exactly the rotational building block structure. In each step, this structure gets updated by the outcome of state machine and rendering is done solely by referring to this data structure. This makes the visual representation the separate independent module.

3.3.8 Three-layered pipeline

The basic idea of having layered architecture is to divide the design into small pieces. Each layer adds to the services provided by the lower layers in such a manner that the highest layer is provided a full set of functionality to simulation. This layered pipeline simplifies the whole design. Another advantage of having this

layered pipeline is to have a clear separation mechanism and policies. Completion of the tube rendering module to visually represent the stem of the twining plat basic three-layered pipeline of this simulation was completed. Clear separation of three layers made the separation of mechanism and policy. This made the model flexible and it made it possible to have an iterative development of the model. Which made the design and development of the model faster. The three-layered pipeline can be described as follows.

1. **First layer - Rotational building blocks structure:** This structure and the mechanism of adding new beads and mechanism of moving beads by rotating the previous bead is the main component in this layer.

2. **Second layer - State machine:** The structure of the first layer was driven by this state machine using the embedded policies manipulating the structure using the mechanisms defined in earlier structure. Each physics step first layer structure was updated according to policies embedded in this layer state machine. Most of the design and development was done in this layer in an iterative methodology without affecting the design of other layers due to the clear separation among layers.

3. **Third layer - Rendering the stem:** Whole visual representation is done in this layer. This layer concluded the basic model of twining plant simulation. All other extra features like a simulation of leaf growth were done on top of this primary model.

3.3.9 The inspiration of the "Cholodny–Went model" to reduce complexity

In the physical representation, a number of sphere colliders were used. The number of colliders used in the simulation grew linearly as the simulation carried out due to the growth of the twining plant as the linear growth was represented by adding new sphere colliders to the tip of the stem in a fixed distance in the direction of last bead's forward direction.

Determining if two arbitrary shapes, in arbitrary positions and orientations, are colliding is not an efficient calculation. The usage of Sphere colliders minimizes the complexity of this calculation as for sphere colliders this is a simple comparison between the sum of two radiuses and the distance between two positions. Assume that there is an N number of colliders in the simulation. the computational complexity of detecting collisions with pairwise tests is O(n2). The number of pairwise tests increases quadratically with the number of colliders. Though practically this

less due to the optimization of collision detection algorithms such as broad phase pruning. For better curves of the stem in tube rendering state, small colliders were used in close distances. This made the number of colliders per plant even higher than Gunawardhana's original bead and string model. A way to minimize the number of colliders was much needed for an efficient, realtime, scalable solution.

In botany, the Cholodny–Went model is an early model describing tropism in emerging shoots. has been criticized and continues to be refined it has largely stood the test of time. Usually, gravitropism and phototropism are explained using this model yet for the proposed solution for the simulation of twinning plants this can be even be extended for thigmotropism as well, as it explains the directional growth happens due to asymmetrical distribution of auxin, a plant growth hormone. According to the colony went model it is the tip were sensitive to light and gravity and tip senses and synthesize the auxins there and sends down the stem to male asymmetric growth. This model of explanation extended to thigmotropism and adapted to the proposed solution to reduce the number of colliders.

In earlier design, a number of sphere colliders were used to through stem. With inspiration Cholodny–Went model the only a single sphere collider was used to sense thigmotropism and this collider was passed along the tip of the stem as the plant grow. This reduced linearly growing N number of colliders to single colliders and reduce the complexity of collision detection to maximum O(1) complexity assuming that there is a constant number of plants and support structures in simulation. The earlier locations where the collider has been before moving it along the tip are being saved in a data structure as it is needed for rendering the tube. This adaptation is just an optimization to have a real-time simulation with a large number of plants. If the inter-plant collisions had to be simulated, this optimization has to be shut as this gets rid of the colliders along the stem.

3.3.10 Sphere based Physical representation to Capsule based representation

The Colony–Went model inspired optimization reduced the O(n2) complexity to O(1) complexity. After the optimization, the need to stick with sphere colliders was not there as the complexity reduced drastically. This gave the opportunity to use complex collider with higher narrow phase-detection complexity. The usage of a capsule collider reduces the collision approximation error as it fit the shape of the stem than the sphere colliders. Storing capsule colliders as well as collision detection using capsule colliders is expensive than of sphere colliders.

The use of capsule colliders fixed the issue of strange curves in the stem as the capsule colliders reduce the approximation error of colliders. Unlike in sphere

colliders when the proper hit mechanism was activated capsule colliders delivered perfect collision handling due to its cylindrical surface. This cylindrical surface guarantees to have collision normal perpendicular to the collider's forward direction. This makes the parsing the colliders along the tip of the stem and simulating twining behavior easier.

3.3.11 Simulation of Leaf Growth

The simulation of leaf growth was out of scope yet a very simple leaf growth mechanism was added to improve the realism. Instead of using texture attached 2D plane to simulate the leaves a simple 3D mesh was used in the simulation model as it improves the realism of the model. The growth was stimulated by scaling up the mesh across x, y, z axes accordingly and the blooming effect was simulated by rotating and transforming the leaves accordingly. There are well defined complicated leaf growth models that simulate biologically accurate leaf growths but the goal here was to come up with a robust and simple leaf growing mechanism to compliment the suggested twining plant simulation model.

3.3.12 Simulation of germination process

Germination has not been modeled in any of the related work suggested models. The germination process is a complex process that is composed of two clearly separate sub-processes. First is the rooting process which is below surface level and the second is what is observable from above the surface. The rooting process is not considered here as the visible biological and biomechanical behaviors are the main focus of this research.

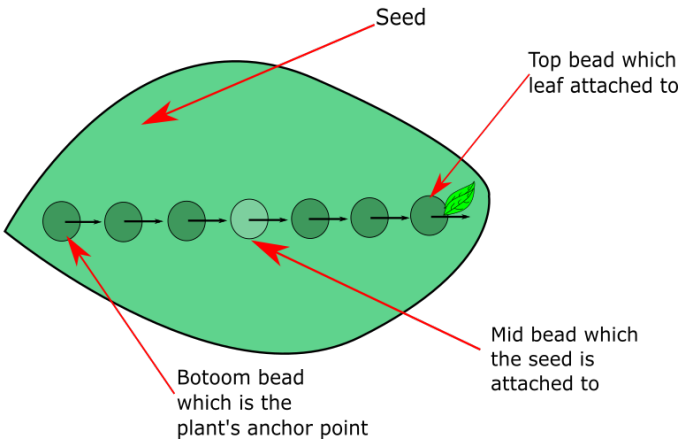

Figure 3.17: A diagram of the germination design

The initial implementation idea to model the behavior of the germination process that is visible above the surface level was inspired by biology. In nature, fully

developed seeds contain an embryo and, in most plant species some store of food reserves, wrapped in a seed coat. This was the idea was adapted according to our Rotational building block structure mechanism. A predefined number of beads was embedded to the inside of the bead very closely (about 0.01 times the usual max interbreed distance)in a compact manner horizontally. This set of beads represented the embryo according to our model. The seed was attached to the mid bead. As divided by this mid bead one half has represented the stem below the cotyledons.edge of this half was the starting bead which acted as the anchor point to the world. Other half represented the stem which is above cotyledons. Initial two leaves were attached to the tip of this half. Figure 3.17 portrays this a diagram of this design.

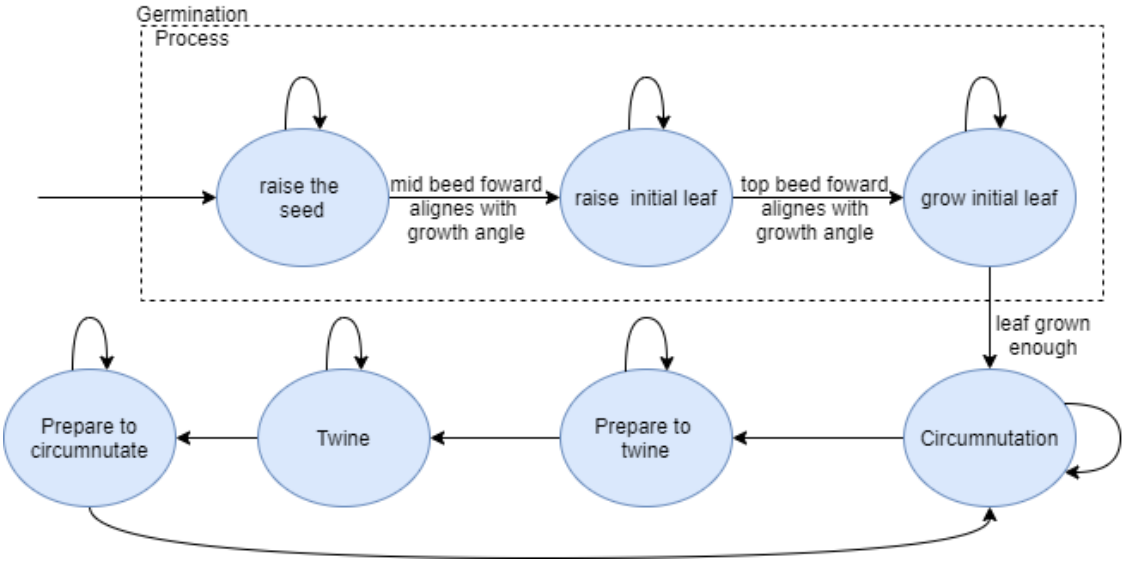

Figure 3.18: Updated state machine to simulate germination

The simulating the germination process was divided into new three sub-processes for the ease of implementation. To drive this process new three states were added to our state machine. The first state was added to simulate the rise of the seed from the surface. The second state was added to simulate leaves coming out from the seed. Third was added to simulate the growth of primary leaves and the nutational behavior. Figure 3.18 show a our updated state machine.

In the first state from starting bead till the mid bead all bead were gradually rotated towards growing angle(here the growth angle was upward direction as the phototropism was not simulated at this stage). But the amount of the rotation applied to each bead was gradually decreased toward the mid bead. This way the beads closer to staring bead rotated faster and beads towards the mid bead were slowly rotated. While this rotation kept applying to beads between the start and mid in the inter bead distance of was increases as well. This created the natural-

looking curve unfolding look in the germination process that was observed in the seed rising stage.

In the second state beads, the same thing was applied to the beads from mid to top. Other than that cotyledons were opened gradually using transformations and leaves were slowly grown using scaling the leaf meshes.

In the third state, leaves were grown faster and using transformations blooming effect was simulated. The nutational behavior was also was simulated using the same above-mentioned mechanism but growing directions were alternated to simulate oscillating nature in nutation.

Chapter 4

Implementation

In this chapter how the earlier discussed design choices were implemented will be discussed in a detailed manner. The choice of technology, the reasoning behind the implementation choices, the parameters that are available for change will be further explored. Implementation challenges that had to be faced during the implementation and how the implementation choices were taken to facilitate further development of the model will be explained in this chapter.

4.1 Tools and Technology

The proposed solution was developed using Unity 3D using c sharp scripting. In the earlier stage, the aim was to improve the PBD based relative rotational model by Gunawardhana et al Therefore Unity PBD framework called "OBI" by Oni was heavily used in the earlier stage. After rejecting the approach of Gunawardhana et al Unity was chosen as the main technology to implement the new model based on the following reasons.

- **Choosing game engine over other graphic API and libraries:** considering the requirements use of graphics API or libraries like OpenGL or Ogre seemed an overkill as there wasn't a requirement for low-level access. Inbuilt functionalities of a game engine like scenegraph, Physics(collision handling), light, texture and material systems, GPU optimizations make the process of implementation efficient and robust.

- **Choosing Unity over other game engines:** Unity is very popular free to use a game engine. Unity is composed of well documented intuitive "C Sharp" scripting API and supportive community. The project can be built across a variety of platforms tested out using inbuilt profiling tools easily. Public parameters of written scripts are being given as a GUI and they can be

tweaked during the run which makes the fine-tuning easier. The marketplace consists of numerous plugins made the process of implementation easier.

4.2 The architecture of the implementation

The implementation of the suggested model was hierarchical and composed of a number of components of various types. Some were running independent and some were communicating with each other. Communication is mostly done with call back functions and some had shared data structures. Figure 4.1 shows scenegraph of a single instance of twining plant(bine).

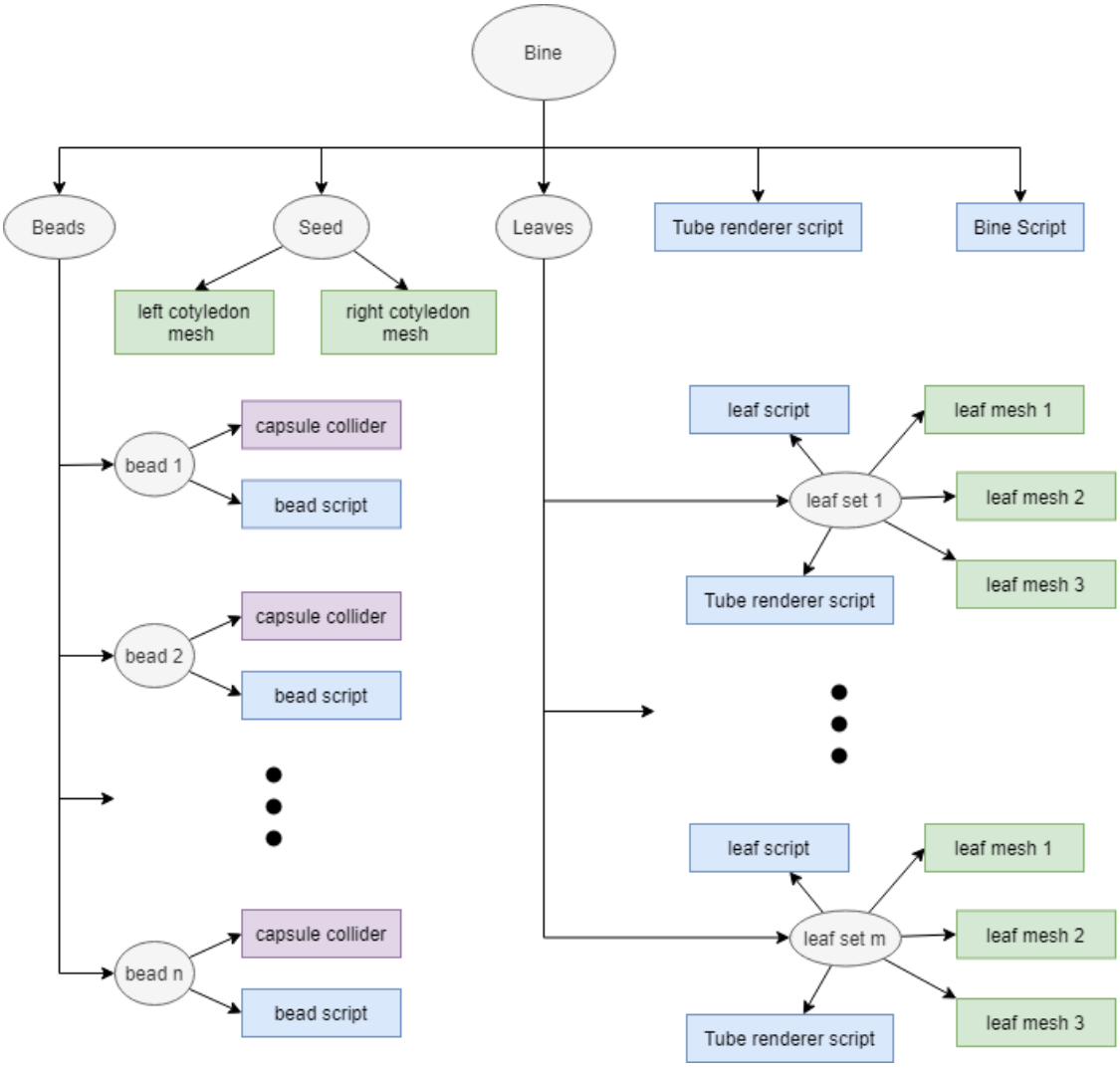

Figure 4.1: The scenegraph of a single instance of bine

A single bine instance is composed of n number of beads and m number of leaves. These two numbers n and m increase as the simulated twining plant grows. Bine's direct child element Tube renderer script is responsible for rendering the whole stem of the twining plant as it grows. Bine script handles the biological and

biomechanical behaviors of the twining plant and it manages the addition of new bead and leaf instances.

A single Leaf set is composed of several elements such as leaf meshes, leaf script and tube render script. Three leaf meshes were used for the simulation as we simulated a twining plant with three leaflets. The leaf script handles the growth and other biological and biomechanical behaviors of the leaf. The tube render of a leaflet handles the whole leaf stem rendering for the leaf set.

Each bead is consist of two components. First is the capsule collider which is used for collision detection. Second is the bead script which calculates collision normal constraints and communicates with the Bine script.

The seed is composed of two meshes that are used to simulate the biological behaviors of cotyledons. The biological behavior of cotyledons was tightly coupled with the overall biological behavior of bine, therefore, it was directly handled by the bine script.

4.3 Bine script explanation

Bine Script is responsible for handling all the biological and biomechanical behaviors of the twining plant. This script was composed of a number of parameters to facilitate the tweaking of simulated biological and biomechanical behaviors. The update function was called once every frame and this acted as the state machine.

4.3.1 Parameters of Bine Script

The following are the public variables of the bine script which can be used to change the look and behavior of the twining plant but some were strictly used for debugging purposes or as shared data structures. These parameters can be changed dynamically or using GUI in the Unity editor.

- **float growthSpeedInCircumnutation = 1;** Manipulates internode stem growing speed in circumnutation state.
- **float initialPhaseInterBeadDistance = 0.1f;** Manipulate the stating distance between beads in germination.
- **int leafTravelDistance = 0;** The distance the leaf travel from initially attached bead while growing.
- **int beedGapPerLeaf = 15;** How often a leaf should be grown. For example, 15 value 15 means one every 15 newly added bead new leaf will be attached.

- **bool addLeaf = true;** Whether to simulate leaves or not.
- **int currentNumberOfBeeds = 0;** Current number of bead in simulation.
- **int currentNumberOfLeaves = 0;** Current number of leaves in simulation.
- **GameObject seed;** the object that should be used as a seed which contains two meshes to simulate cotyledons left and right in order.
- **GameObject leaf;** Primary leaf set containing two meshes left and right leaf in order.
- **GameObject secLeaf;** Secondary leaf set object. Clones of this will be instantiated in the middle of the simulation to add new leaves. Behavior is handled by a sperate script.
- **int particleCount = 300;** Total number of particles used for the entire simulation.
- **GameObject beed;** Specified object here is used as a bead. Any type of collider of any size with attached bead script can be used here. A capsule collider with attached bead scripts is suggested.
- **GameObject[] plant;** This is an array of bead object. Used as a shared data structure.
- **Vector3[] cn;** This is an array of collision normal. Used as a shared data structure.
- **Vector3 curvature = new Vector3(5, 5, 5);** used to adjust curvature of stem in circumnutation.
- **Vector3 circumnutationSpeed = new Vector3(0, 5, 0);** used to adjust circumnutation speed. The vector is correspond to speed across each axis.
- **bool BiomechanismsOn = true;** Turn on/ turn of biomechanisms. Used for debug purposes.
- **float supportLostGravitrophismAdjustment = 0.01f;** Used to adjust the amount of rotation per bead to adjust the stem to face upward when leaving the support structure.
- **float maxGirth = 2;** Used to adjust max girth of stem throughout the simulation.

- **bool hasGroped = false;** Can turn on and off groping behavior. Used for debug purposes.
- **float nutatingAmountPerTurn = 5;** Used to manipulate max angle of change in nutation in the primary leaf growth stage.
- **float germinationSpeed = 1;** Used to adjust the speed of germination process.
- **float upwardTurningSpeed;** Used to change the effect of Gravitropism in twining behavior. Higher the value higher pitch in twining spiral regardless of the radius of the support structure.
- **float stipulesGirthIncrement = 1.3f;** The amount of girth increase in stipules.
- **int numberOfBeasTillCotyldons = 11;** Number of bead used to simulated germination.

4.4 Bead script explanation

These scripts are responsible for handling the collisions of the beads and communication with Bine instance. The full script is available at Appendix C.4. A bine is consists of a number of bead objects. Each bead is composed of a collider and a bead script. A capsule collider with a radius of 0.1 and a height of 0.4 units is recommended. The whole physics representation of a bine instance relies on beads as it is modeled as a collection of bead colliders. The proper hit mechanism discussed in chapter 3 is implemented here inside the "OnCollisionEnter" function which is a unity API given call back function. If the collision is counted as proper hit it will notify the bine script using the callback "onHitSupportStructure" by parsing the collision details. Otherwise, it will ignore the collision.

4.4.1 Parameter of Bead

- **float acceptedCollitionAngle = 30;** The opening angle of the rejected collision cone which is discussed in proper hit mechanism under chapter 3.
- **GameObject thePlant;** the bine object which the bead belongs to

4.5 Leaf script explanation

This script is responsible for handling the biological and biomechanical behavior of secondary leaves. Full code is available at Appendix C.5. Each secondary leaf set is composed of this script. The growth of the leaf is separated into two functions. First, is "simpleLeafGrowth" which is responsible for the growth of leaf this is done by scaling the leaf meshes in accordingly in axes and rendering the stem of the leaf using tube renderer. Second is complexLeafGrowth which includes the simpleLeafGrowth and the blossoming behavior of the leaf. These two functions were created in order to support additional leaf meshes and leaf blossoming behaviors. "showHeliotropicMovements" functions are used to simulate heliotropic movements of the leaves.

4.5.1 Parameter of Leaf script

- **GameObject[] plant;** Bead array data structure which shared from the bine object's bine script which the current leaf set belongs to.
- **GameObject attachedBeed;** Currently the leaf attached bead.
- **int destinationBeadNumber;** When leaf set travels in growth the index of destination bead.
- **int currentBeadNumber;** Index of the currently attached bead.
- **float growthRate;** Adjusts general growth rate including leaf blossoming speed.
- **bool isSimpleGrowth = true;** Choose between simple growth and complex growth.
- **float startAngle = 85f;** Starting pitch angle of each leaflet in degrees.
- **float endAngle = 10;** Ending pith angle of each leaflet in degrees.
- **bool isLeafTriplet;** Select whether the type of leaf is triplet
- **Vector3 Offset;** The starting offset from the attached bead
- **float endleafSetGrowthAngle;** Max angle difference between forward of leaf set and growing angle
- **float SpanningSpeed;** Adjusts leaflets spanning speed
- **float leafScalingSpeed;** Adjusts leaflets scaling speed

- **float VerticleStemGrowthSpeed;** The stem's upward growth speed
- **float HorizontalStemGrowthSpeed;** The stem's outward growth speed
- **float rotatingSpeed;** Heliotropic movement rotate speed
- **bool heliotropicLeafs = true;** Turn on and of heliotropic movements

4.6 Tube renderer script explanation

This was a free and open-source script available at Unity market place [36]. A small modification was done to script to facilitate the tube with a variable radius at different places.

4.6.1 Parameter of Tube renderer script

- **Material material;** the material of the mesh which the script renders.
- **int crossSegments = 50;** number of cross segments. For example, setting this to value 3 renders a prism-like a tube. Having a higher number of cross segments increases the roundness of the rendered tube.

Chapter 5

Results and Evaluation

5.1 Introduction

In this chapter how Evaluation of the proposed model was carried out will be explained. As the balance between performance and realism plays a major role in graphics-based researches the evaluation of this research can be divided into two main categories. First is via a performance analysis and second is via a Realism analysis. Following Figure 5.1 depicts this two-way evaluation procedure and its sub-components that were explored in this research for the evaluation.

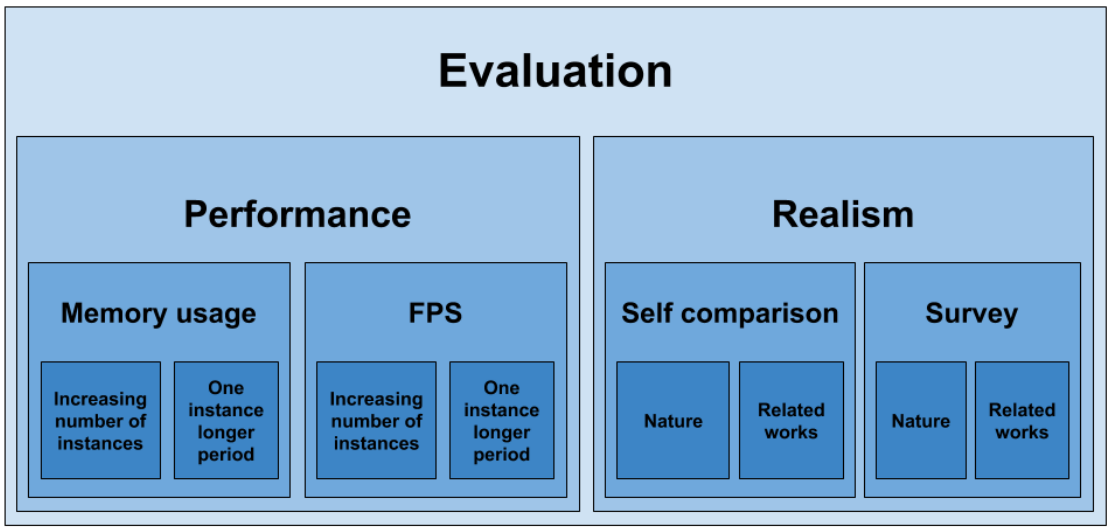

Figure 5.1: High-level diagrams of Evaluation procedure

5.2 Evaluation of performance

The performance of a simulation relies on two main things namely memory usage and number of computations. Balanced memory usage and the number of compu-

tations are crucial to a successful real-time simulation. To analyze the impact of these two components in this research, the performance analysis was conducted by analyzing the memory usage for the whole simulation and by analyzing the frames per second (FPS) of the simulation.

Analysis of FPS is performed by recording the FPS throughout a whole scenario and analyzing the graphs they produced. FPS was calculated right after the introduction of each bead and saved in an array and written to a "comma-separated value" (.csv) file at the end of the scenario. Here the impact of the time for calculation and recording of FPS was considered negligible. These calculated FPS values were cross-examined against unity inbuilt profiler and accuracy was verified. Graphs were drawn using Microsoft excel using the produced ".csv" file. All the test scenarios were run on a machine with the following specifications.

- System Information
  - Operating System: Windows 10 Enterprise 64-bit
  - System Manufacturer: Dell Inc.
  - Processor: Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz (4 CPUs), 2.6GHz
  - Memory: 8192MB RAM
  - DirectX Version: DirectX 12
  - Microsoft Graphics Hybrid: Supported
- Display drivers
  - Card name: Intel(R) HD Graphics 520
  - Manufacturer: Intel Corporation
  - Chip type: Intel(R) HD Graphics Family
  - DAC type: Internal
  - Display Memory: 4170 MB
  - Dedicated Memory: 128 MB
  - Shared Memory: 4042 MB
  - Current Mode: 1366 x 768 (32 bit) (60Hz)
- Render Device
  - Card name: AMD Radeon R5 M335
  - Manufacturer: Advanced Micro Devices, Inc.

- Chip type: AMD Radeon Graphics Processor (0x6660)
- DAC type: Internal DAC(400MHz)
- Device Type: Render-Only Device
- Display Memory: 8130 MB
- Dedicated Memory: 4087 MB
- Shared Memory: 4042 MB

5.2.1 Analysis of FPS of a single instance for an extended period of time

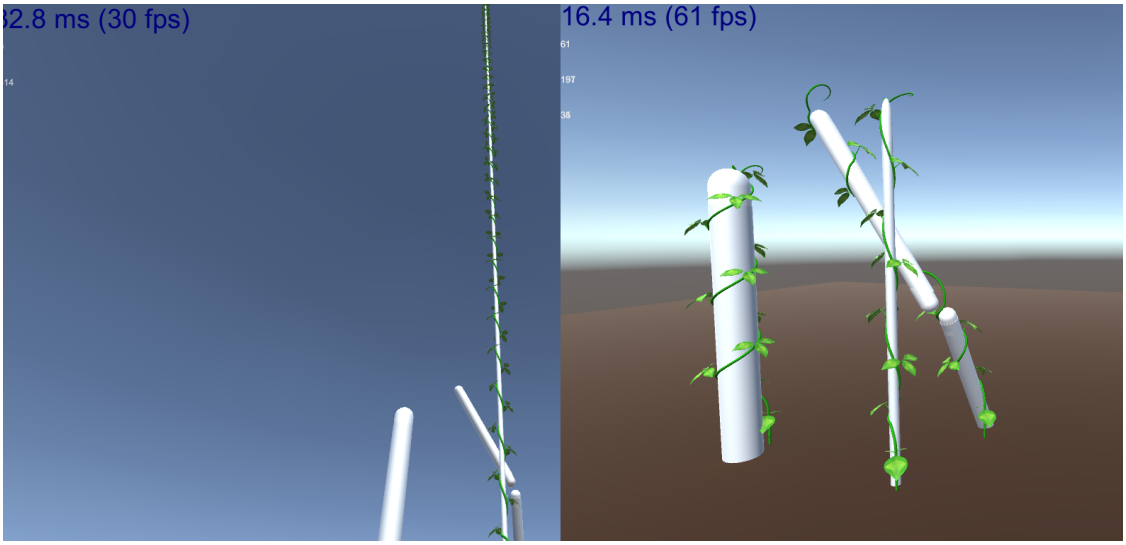

Figure 5.2: Created stress test with 2000+ beads and tested normal scenarios with below 150 beads per instance

The purpose of this analysis was to test the limits of a single instance simulation. For normal scenario testing, the number of beads that were used per a single instance was around 120 beads. As the comparison shows in Figure 5.2, a stress test was planned with a single twinning plant instance having over 2000 beads and a straight cylindrical support structure of 1000 units tall. A production build of this scenario was used for the analysis and data was recorded by running the same scenario for 10 times and recorded valued were averaged. As Figure 5.3 shows a scatter plot of Averaged FPS against the number of beads was drawn.The following limitations were identified after analyzing the number of beads vs average FPS graph of a single twining plant instance.

1. A single instance scenario of the proposed model to an upper limit of 500 beads can be run efficiently at 60FPS.

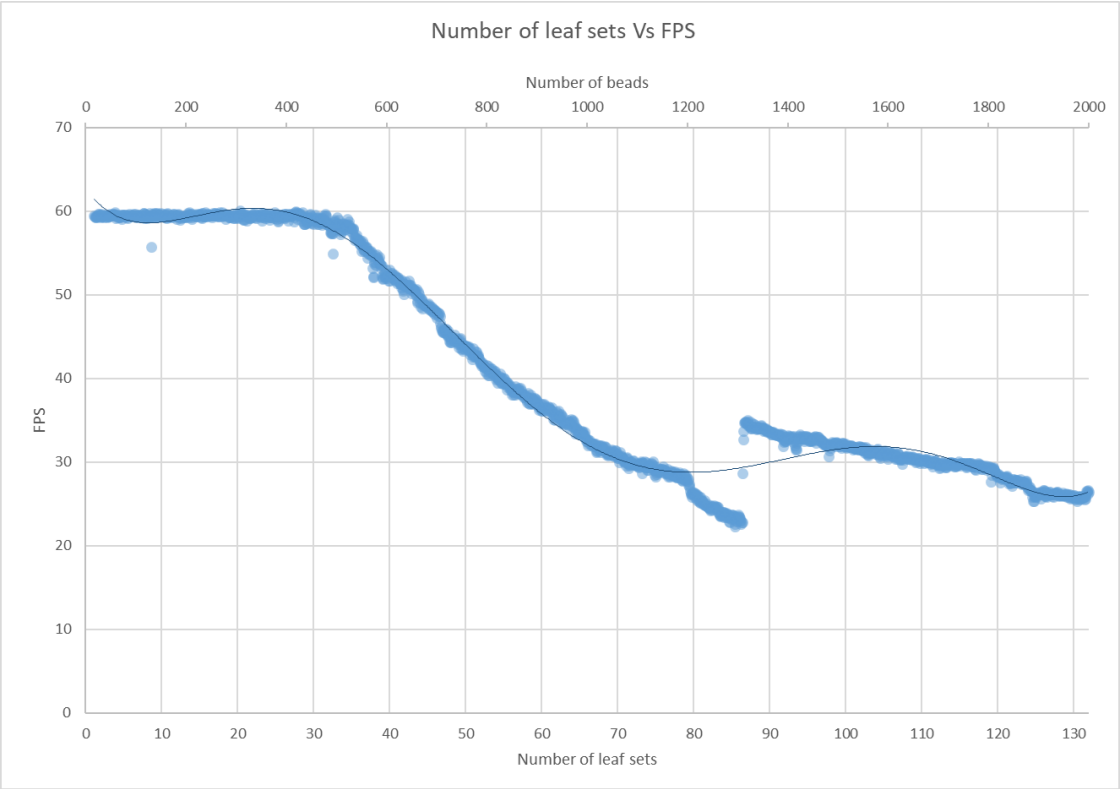

Figure 5.3: Number of beads Vs the average FPS

2. Having over 500 beads the FPS drops linearly from 60 FPS to 25 FPS at the upper limit of 1300 beads.

3. The sudden FPS increment at around 1300 beads was occurred due to a failing stem renderer. The tube rendering module that was used to render the stem crashes around 1300 beads as it exceeds the maximum number of vertices (64k) per single mesh which was enforced by the unity engine. This crash can be delayed to a certain extent by reducing the number of cross segments of the tube (which was currently set to 50. The minimum value is 3 which makes the stem a Triangular prism) though it may affect the roundness of the stem.

4. Even after the 1309Th bead, the proposed model grows further without rendering the stem but the leaves and physical representation grow without any problems.

5.2.2 Analysis of FPS against an increasing number of instances

In this scenario, the limits of running multiple instances of the model were tested and analyzed. No production build was created for these scenarios. Instead, new in-

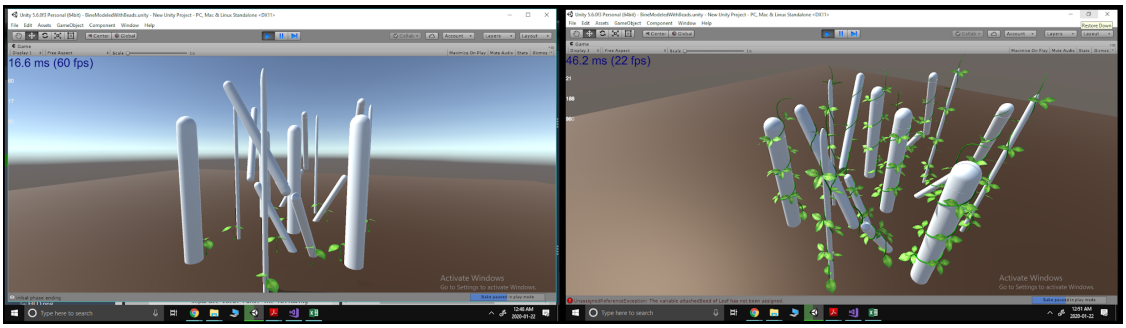

Figure 5.4: Twelve instances running parallel

stances were added and data was recorded in the development environment thought the data was collected in multiple separate runs. Figure 5.4 depicts some snapshots of tested scenarios. Here no production build was used as the aim of this analysis was to analyze the decrements of FPS relative to the increasing number of models run parallelly not a comparison between absolute FPS values. Building optimized build per each number of instances scenario and recording data on them is a time-consuming activity. Data collected in the development environment by increasing the number of instances from 1 to 10 and again decreasing the number of instances from 10 to one in 20 separate total runs. The following surface graph depicted in Figure 5.5 was drawn by averaging the data obtained through both increasing and decreasing runs.

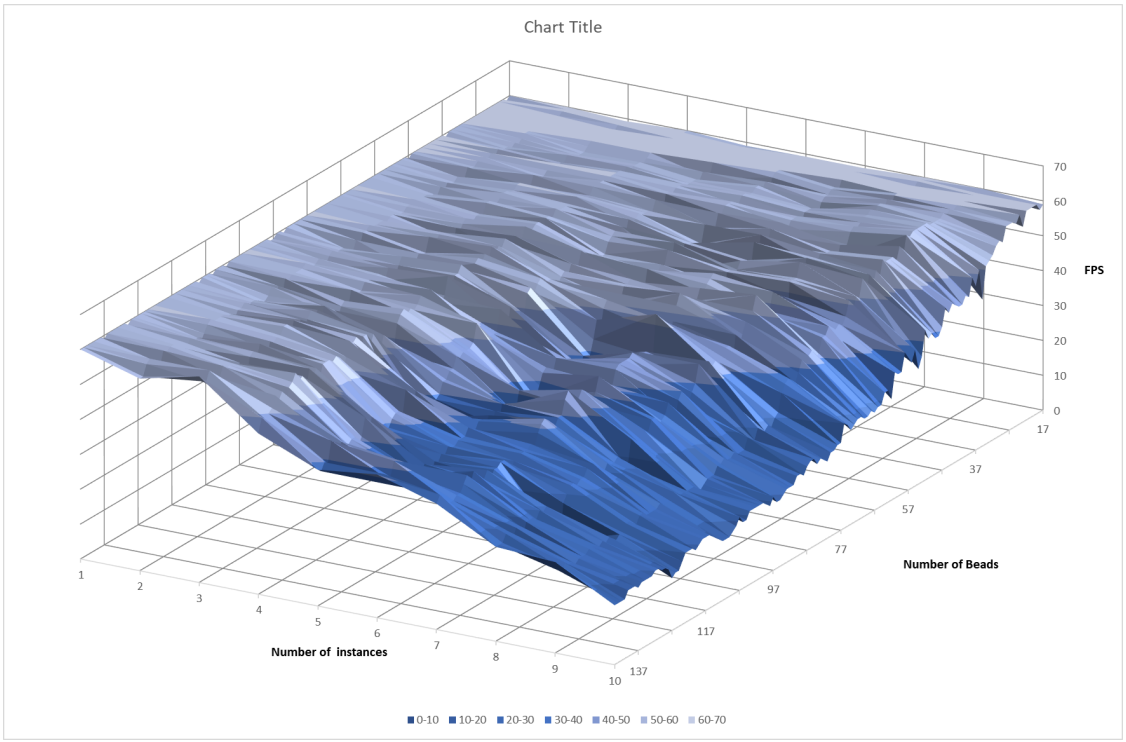

Figure 5.5: Number of instances VS FPS

The following information was obtained by analyzing the above surface graph.

1. To an upper limit of 35 beads for all tested scenarios, the simulation was carried out without a significant loss of FPS but after that, the FPS drops relative to the increasing number of beads are linear.

2. The current model can simulate around a maximum of 10 instances parallelly real-time in the development environment maintaining around 25 fps.

3. Approximately 5 instances can be run parallelly for a normal scenario (number of beads per instance is below 150) maintaining around 50 FPS.

These values were obtained by simulating the growth of multiple twinning plant instances parallelly in a fast phase. The number of instances that can be run parallelly can be increased by not running each instance in every frame and implementing a round-robin scheduling as the behaviors of the twining plants are not fast phased.

5.2.3 Analysis of memory usage of a single instance for an extended period of time

The memory usage data was also collected in the same way but instead of calculating the memory usage, the data was obtained using unity scripting API. The same earlier first scenario was used. The only difference was memory usage obtain through unity scripting API was recorded in the earlier array instead of FPS. The memory usage to maintain the array was negligible. For this, an optimized build was used as in the development environment the API returned value is a sum of memory usage of profiler, and unity editor. Data was collected in 3 separate runs and decided to stop rerunning after the third run as the data points didn't change significantly in all three observed runs, unlike FPS values. Following Figure 5.6 shows the scatter plot which was drawn using gathered data.

The following information was gathered after analyzing the above graph.

1. The memory usage per a single instance grows linearly as the number of beads increases and memory usage growth was below 12MB.

2. The minimum recorded memory usage value was 32MB which was after the first additional bead was added and the maximum value recorded was 43MB which is at the end.

3. Right at the start, the memory usage is around 36mb and it drops right after the initialization.

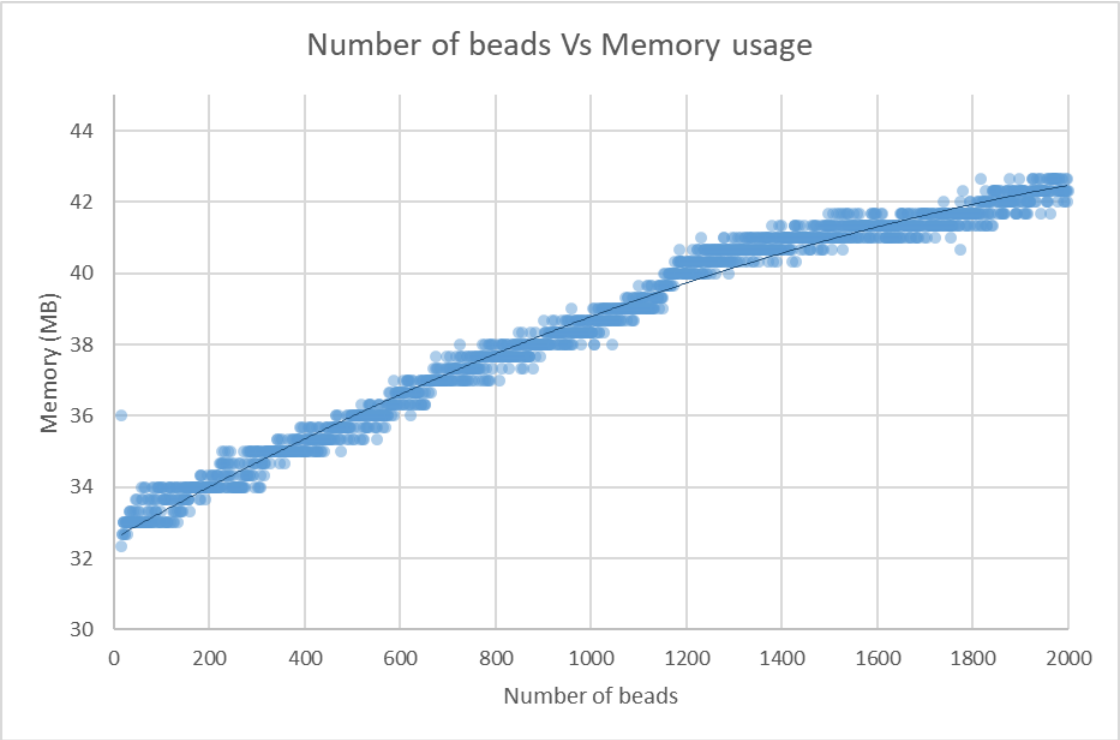

Figure 5.6: Number of beads Vs memory usage

5.2.4 Analysis of memory usage against an increasing number of instances

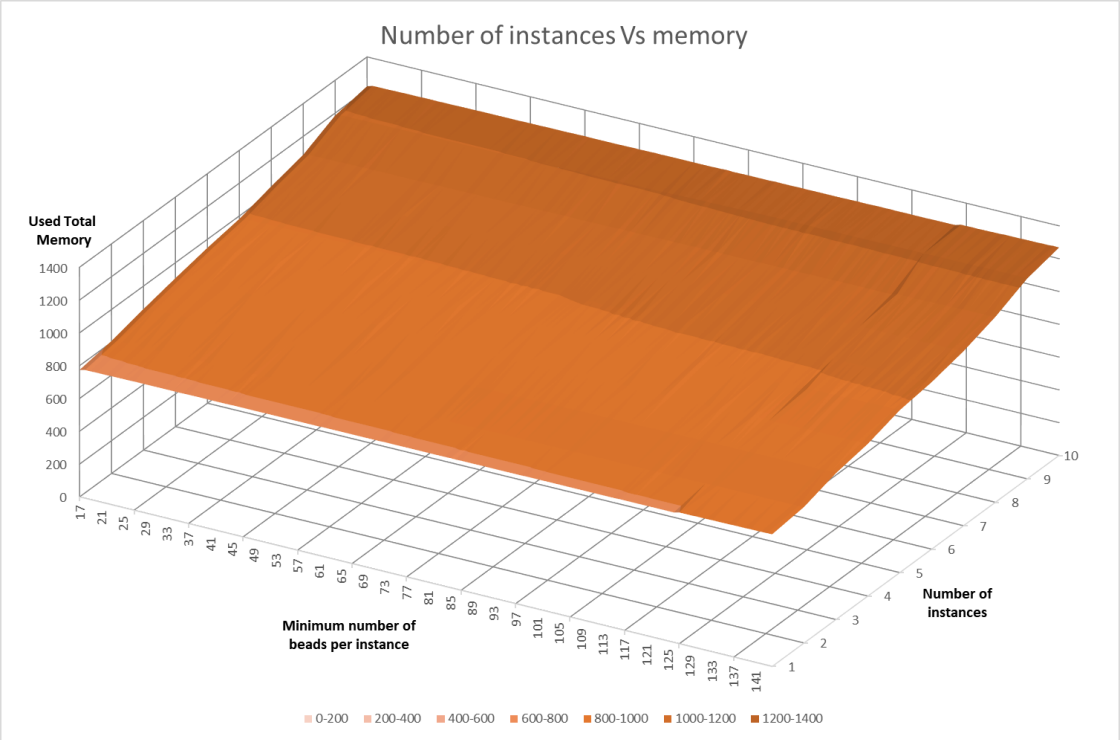

Figure 5.7: Number of instances vs memory

The data was recorded at the same time when the earlier second scenario was tested where FPS and memory usage were recorded at the same time in the development environment increasing and decreasing run. Here the aim was to analyze the memory growth relative to the increasing number of instances hence the impact of profiler and unity editor memory usage addition to the API returned value was negligible as it stayed almost the same throughout the whole two sub scenario where the number of instances was increased and decreased. This was verified by cross-checking the values with unity inbuilt profiler. Following Figure 5.7 shows the surface plot of the collected data. The memory growth relative to the increasing number of twinning plant instances is a linear and average memory value increment per instance was around 50MB throughout the whole simulation.

5.3 Evaluation of visual realism

The evaluation of the visual realism of this research was designed with two concerns. The first was to deliver a self-evaluation as a cross-comparison of suggested twinning plant simulation model against related works and nature. Under this first component, a lengthy and descriptive cross-comparison will be conducted explaining the identified biological and biomechanical behaviors of twining plants in nature and related works. As this self-evaluation is subjective to the authors' current understandings of biomechanical and biological behaviors of the twinning plants a second component was planned. The goal of this second component was to evaluate the suggested model objectively to describe its true and correct reality, which is independent of those involved in the research process.

5.3.1 Introduction to Self-evaluation by cross-comparison

The goal of self-evaluation by cross-comparison is to compare the biological and biomechanical behaviors that are being stimulated by our suggested model against the models suggested in related work and nature itself. In this section, a brief introduction to the method that was chosen to employed to carry out the self-comparison will be given. The detailed comparison of biological and biomechanical behaviors simulated by each model will be discussed together with the result of the survey in the later section.

Gunawardhana's PBD based relative rotational approach [5], Wong and Chen's procedural model of climbing plant with tendrils [14], Interactive climbing plant authoring model by Hädrich et al [1], Super space clothoid based model by Casati and Bertails-Descoubes [32] will be used in cross-comparison as the models suggested by related work in this section. The video footage of each model's behavior

will be compared against nature. The time-lapse footage captured in the experiments that were conducted in the earlier part of the research as well as YouTube time-lapse videos of specific biological and biomechanical behaviors were referred to here as nature.

5.3.2 Introduction to evaluation by analyzing survey data

As a part of the evaluation a survey was conducted to evaluate the visual realism of our model and other related works suggested models. Here the aim was to have an unbiased evaluation of visual realism of the all currently available twining plant simulation models and estimate how realistic is our suggested model compared others. A combined video of all available models (including our suggested model) was created and given to participants with videos that portray the biological and biomechanical behaviors in nature. Then they were asked to asked rate the biological and biomechanical behaviors using the Likert scale-based questionnaire while comparing the provided videos (combined simulation video and videos of nature). Responses were recorded using Google Forms with a selected group of thirty-one participants. The participants consisted of eleven CS major university students, ten professionals from the IT industry and ten non-CS related university students.

The same related work suggested models that were discussed in the earlier section were used for the survey as well. Author names and model names were removed and the following mapping was used throughout the survey.

1. **Model A :** Wong and Chen's Model
2. **Model B :** The Superspace Clothoid model
3. **Model C :** PBD based model by Gunawardhana et al
4. **Model D :** Our Model

Five survey questions were made to evaluate each major biological and biomechanical behavior shown by twining plants in nature. Each question had four sections related to the earlier discussed four models which contained slightly modified five-point quality-related Likert scale based options namely "Not available" (Very Poor), "Poor", "Fair", "Good", "Excellent".Figure 5.8 shows a question format that was used in survey. The sixth question was related to the overall biological and biomechanical behavior simulated by each model this was in some format as earlier questions but the original five-point quality-related Likert scale was used. As the final question, an optional comment field was added.

The result of all twenty-four Likert scale questions was analyzed using "Diverging Stacked Bar Charts". The option "fair" was considered as the neutral

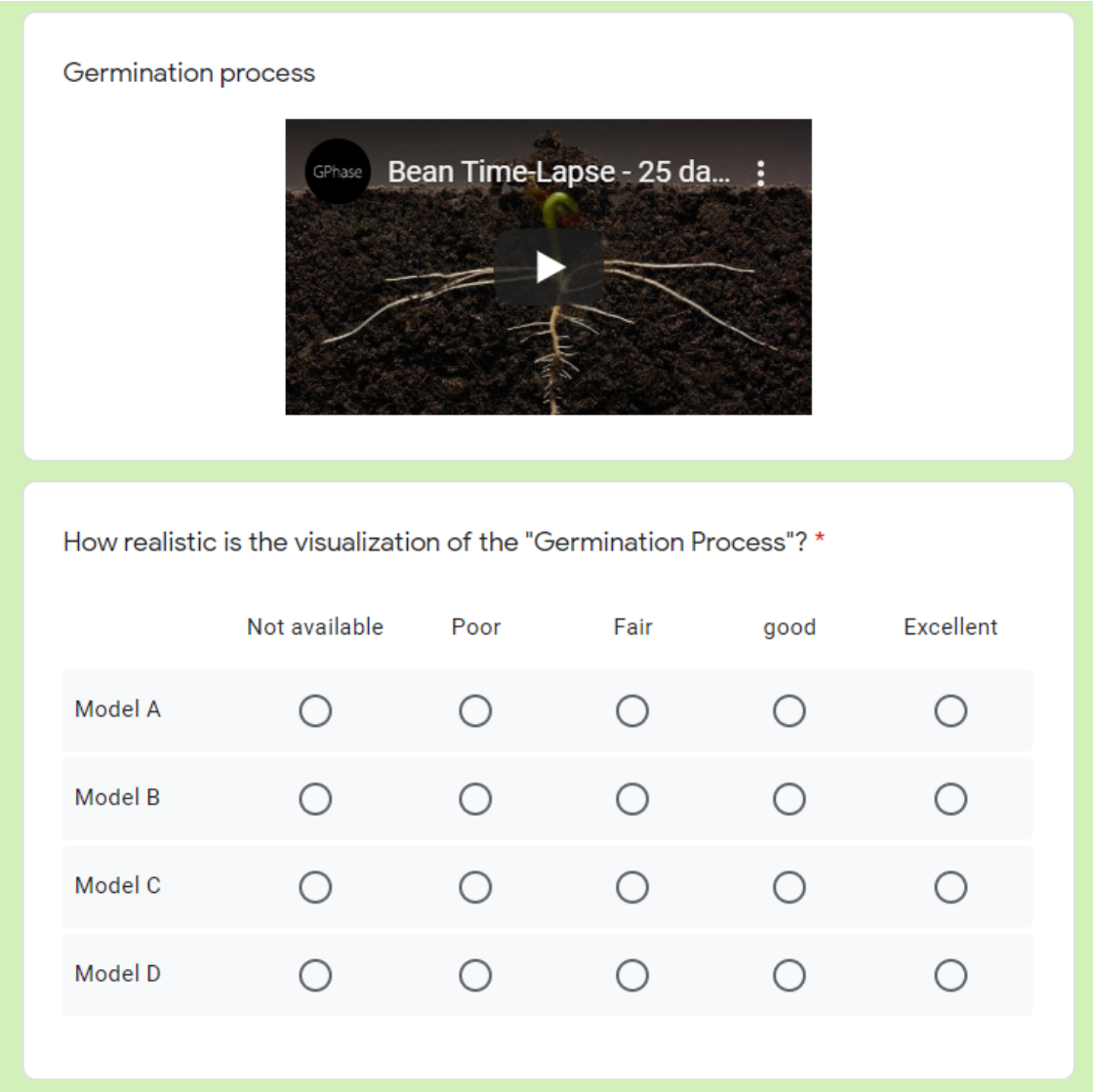

Figure 5.8: An example question used in the survey

option. "Poor" and "not available"(very poor) were considered negative responses while "good" and "excellent" were considered positive responses for the "diverging stacked bar charts" representation. All together six stacked bar charts were produced using the result of the survey. Produced charts will be further analyzed using mode and percentile statistic measures in the next section altogether with the self-comparison.

5.3.3 Self-comparison and survey result combined analysis

The goal of this section was to evaluate the visual realism of the model using the combination of self-comparison and survey results. The following five phenomena were identified as the major biological and biomechanical behaviors shown by the twining plants and they will be analyzed one by one in the following sec-

tion. Behaviors are circumnutation behavior, twining behavior, girth variation throughout the stem, germination process and growth of leaves. After analyzing the above-mentioned behaviors individually. Visual realism of the whole model will be evaluated using the same method.

5.3.3.1 Circumnutation Behavior

Circumnutation which is the successive bowing or bending in different directions of the growing tip of the stem can be observed in many plants especially climbing plants in nature. Though this behavior is crucial to find a support structure none of the simulation models except Gunawardhana's model simulated this behavior. Other suggested models did not employ any biomechanical behaviors but other approaches in order to find support structure at the initial stage. Wong and the Chen's model had a prerequisite of support structure radius and positioning while Hädrich et al's model employed a ray-casting technique to identify nearby support structures. Super-space clothoid based model needed to be position right by a support structure in such a way the model touches the support structure. Therefore no other related work suggested models could simulate circumnutation behavior. Figure 5.9 shows a comparison of this behavior as seen in Gunawardhana's model, the suggested model and nature in the order of top, middle, and lower picture rows and time flow left to right order.

In the Gunawardhana's model, the simulated circumnutation behavior was unstable and Jittery as the natural curve of the nutating stem was not simulated correctly. The addition of new particles to simulation was done from the bottom end this is the main reason behind unexpected jittery in Gunawardhana's model. The Gauss-Seidel solver with a low number of iterations used in PBD increased this jittery in a feedback loop. In the suggested model this circumnutation behavior was stable and the natural curve of the stem was simulated. The amount of curvature could be adjusted by changing parameters to match the curve of different species. Both Gunawardhana's model and the newly suggested model strictly used this behavior to find a support structure.

Circumnutation at the initial support finding stage as well as the circumnutation right after leaving a support structure was simulated by the suggested model. Figure 5.10 depicts a comparison between the suggested model and nature. Gunawardhana's model could not simulate beyond 3 twines around the support structure therefore no support structure leaving mechanism was implemented there.

The result of survey can be visualized in a diverging stacked bar charts as in Figure 5.11. Table 5.1 shows the related statistical measures. Though model C clearly shows circumnutation behavior one participant has marked the "not avail-

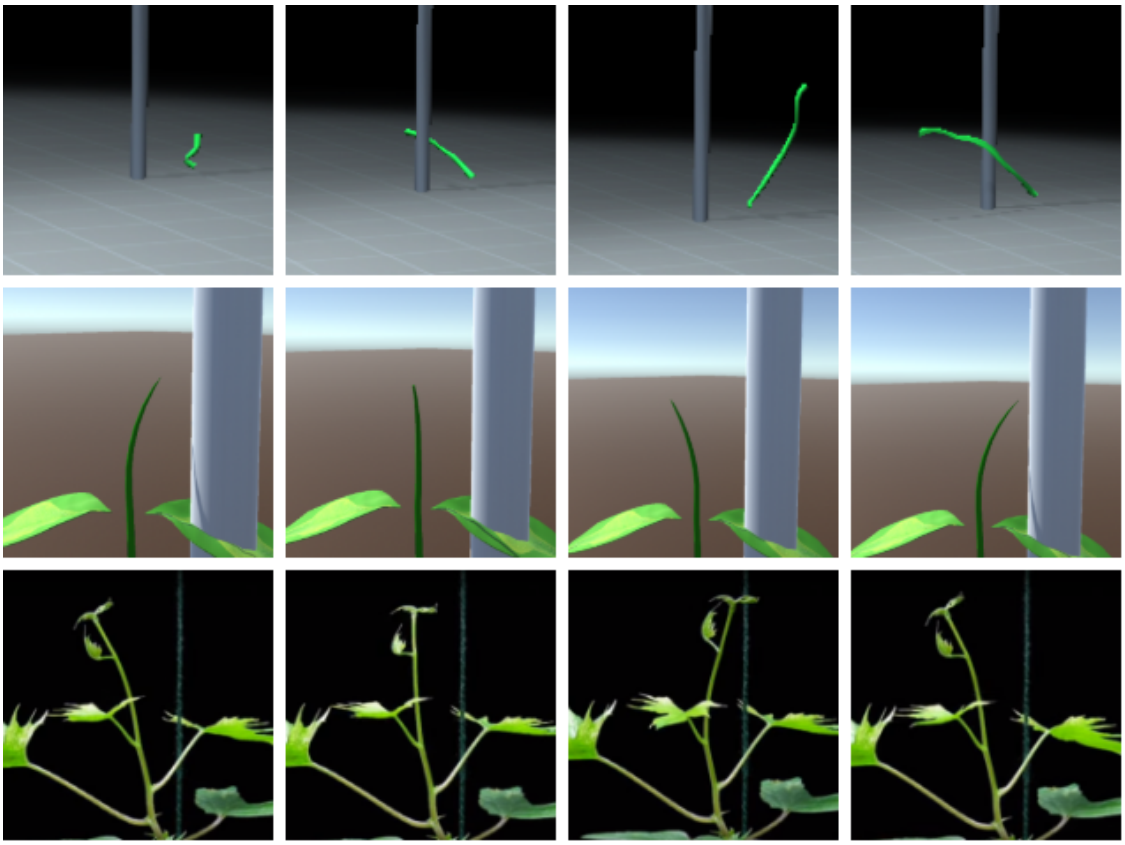

Figure 5.9: Comparison of Circumnutation as seen in Gunawardhana's model, the suggested model and nature in the order of top, middle, and lower picture rows and time flow wise left to right order.

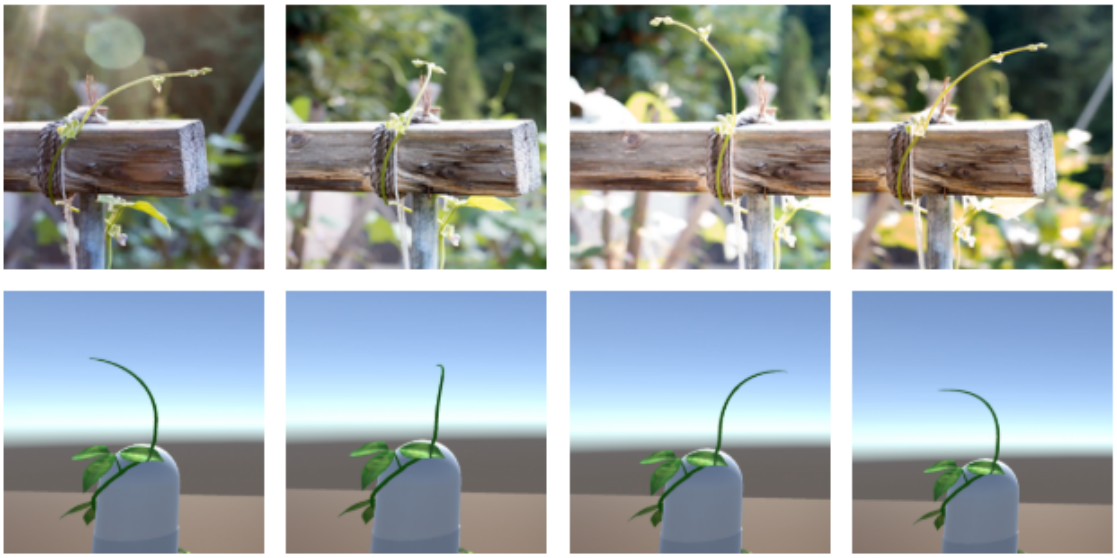

Figure 5.10: Comparison of Circumnutation at support structure leaving stage as observed in nature and suggested model in the order of top and bottom picture rows. The time flow is in left to right order.

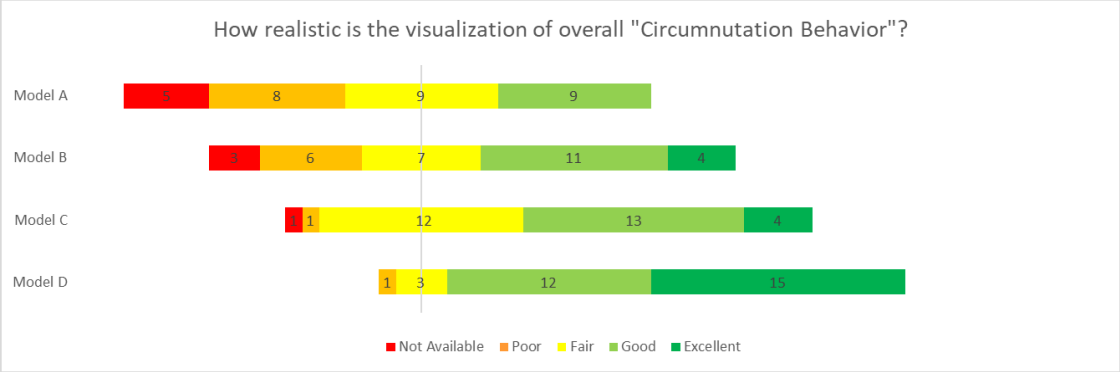

Figure 5.11: Diverging stacked bar chart for survey question "How realistic is the visualization of overall Circumnutation Behavior?"

Model	Mode	Median	Negatives%	Positives%
model A	Fair and Good	Fair	42%	29%
model B	Good	Fair	29%	48%
model C	Good	Good	06%	55%
model D	Excellent	Good	03%	87%

Table 5.1: Statistical measures of survey result for question "How realistic is the visualization of overall Circumnutation Behavior?"

able" option. Having 29% and 48% positive response percentages in model A and model B is contradicting with our observations but considering higher negative response percentages in them relative to models C and D we can resolve this contradictory nature to a certain extent. However, the fact that model D which is the newly suggested model excels in the simulation of circumnutation behavior compared to others can be justified considering the following statistics. First, our newly suggested model was able to score 48% "excellent" responses and secondly, it scored 87% total positive response percentage. Both were highest recorded across all models. Thirdly it had the least negative score which is 3%.

Considering both self-comparison and survey result analysis we can justify that our model performs the best compared to other models in simulation of the circumnutation behavior of twining plants.

5.3.3.2 Twining behavior

Twinning behavior is the most significant biological and biomechanical behavior which can be observed in twining plants. This behavior governs the upward climbing nature of twining plants. In all the selected simulation models, this behavior can be observed. Figure 5.12 shows a comparison of twining behavior among related works, newly suggested model and nature.

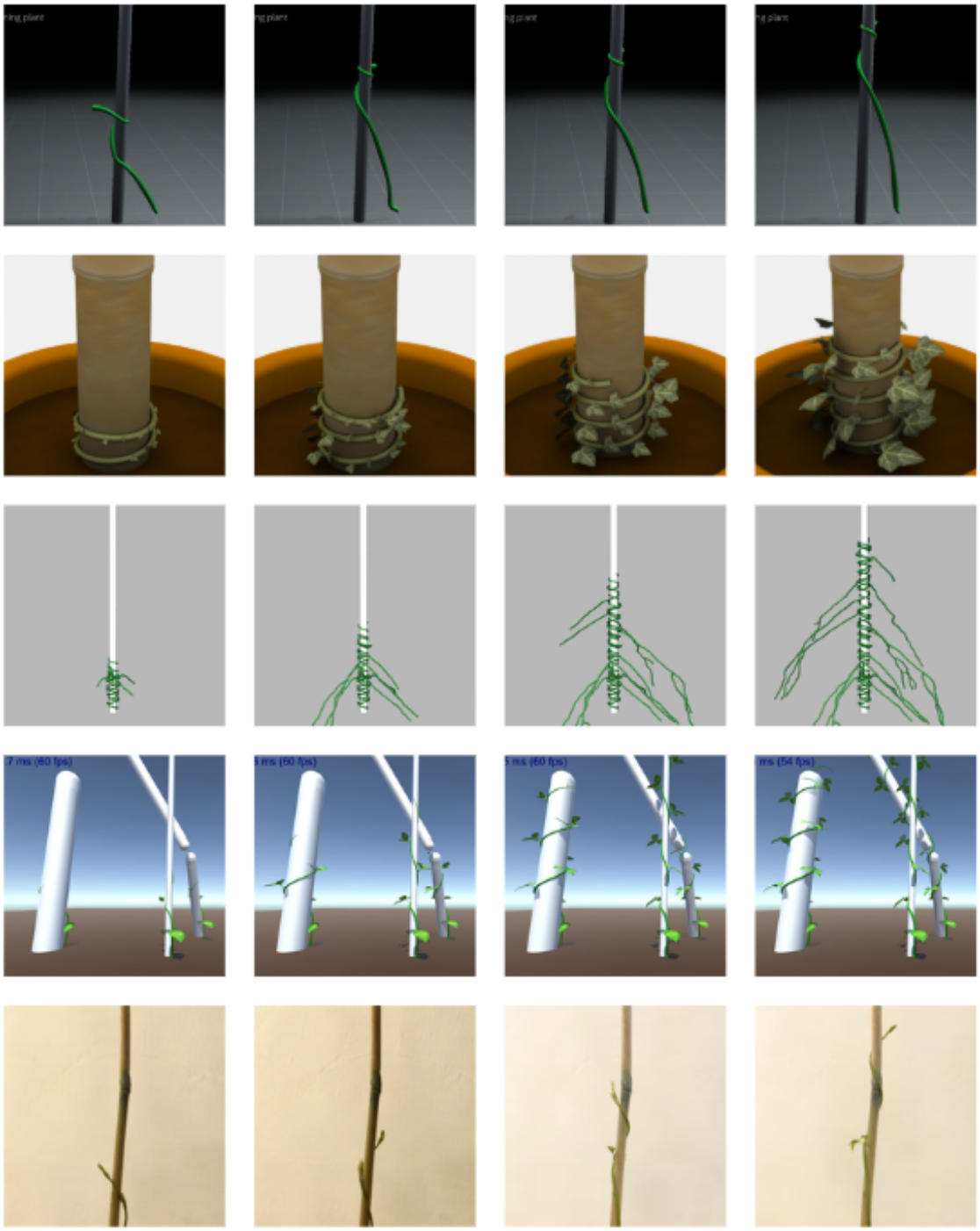

Figure 5.12: Comparison of twining behaviour at as simulated in related work suggested models. Gunawardhana's model, super space clothoid model, Wong and Chen's model, newly suggested model and Nature in the order from top to bottom picture rows. The time flow is in left to right order.

The twining behavior of Gunawardhana's model is accurate but due to the limitation of the approach, the twining behavior could not be extended beyond 3 twines. The groping behavior (behavior of adjusting and tightening the twines) was best simulated in Gunawardhana's model. Wong and Chen's model simulated

the twining behavior and groping behavior to a certain extent but both Gunawardhana's model and super clothoid model did suffer from the same sliding effect due to the addition of each new particle are done via bottom end. Here, the word "sliding effect" is used to identify the visual feel of pulling and sliding a buried rope around the surface of a support structure in an upward direction.

Wong and Chen's model did not have this sliding effect yet lack of a groping mechanism was visible. Considering all the related work suggested work Wong and Chen's model was the most robust and realistic compared to others. Lack of biological and biomechanical behavior and heavily relying on previously fetched data were the only cons compared to the newly suggested model.

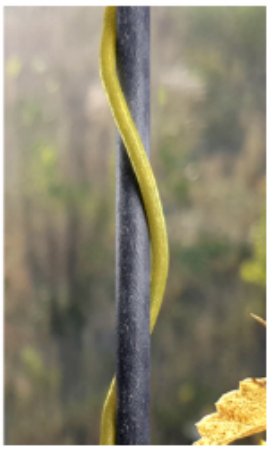

Figure 5.13: Twining plant result of interactive climbing plant authoring model of Hädrich et al

Hädrich et al's model didn't specifically simulate any twining behavior but with changing combination of modeling plant growth, physics response parameters and artistic tools (custom drawn attractors, local branch control) result shown in Figure 5.13 was achieved. This result was closer to a manually created scene than simulation result as the level of human interaction was high.

Our newly suggested model is robust and visually realistic compared to others considering the biologically and bio-mechanically accurate results in the previous stress tests. Newly suggested model was able to simulate groping behavior to a certain extent and its twining behavior was able to simulate the difference of the pitch in the twined spiral depends on the radius of the support structure. Figure 5.14 show this difference of pitch in spirals depend on the girt of the support structure. This phenomenon was analyzed and underline physics was explained by Goriely and Neukirch [9]. The result of the newly suggested model aligns with Goriely and Neukirch's physics model fundamentals where the increasing radius of support structure tends to lower the pitch of the twining spiral.

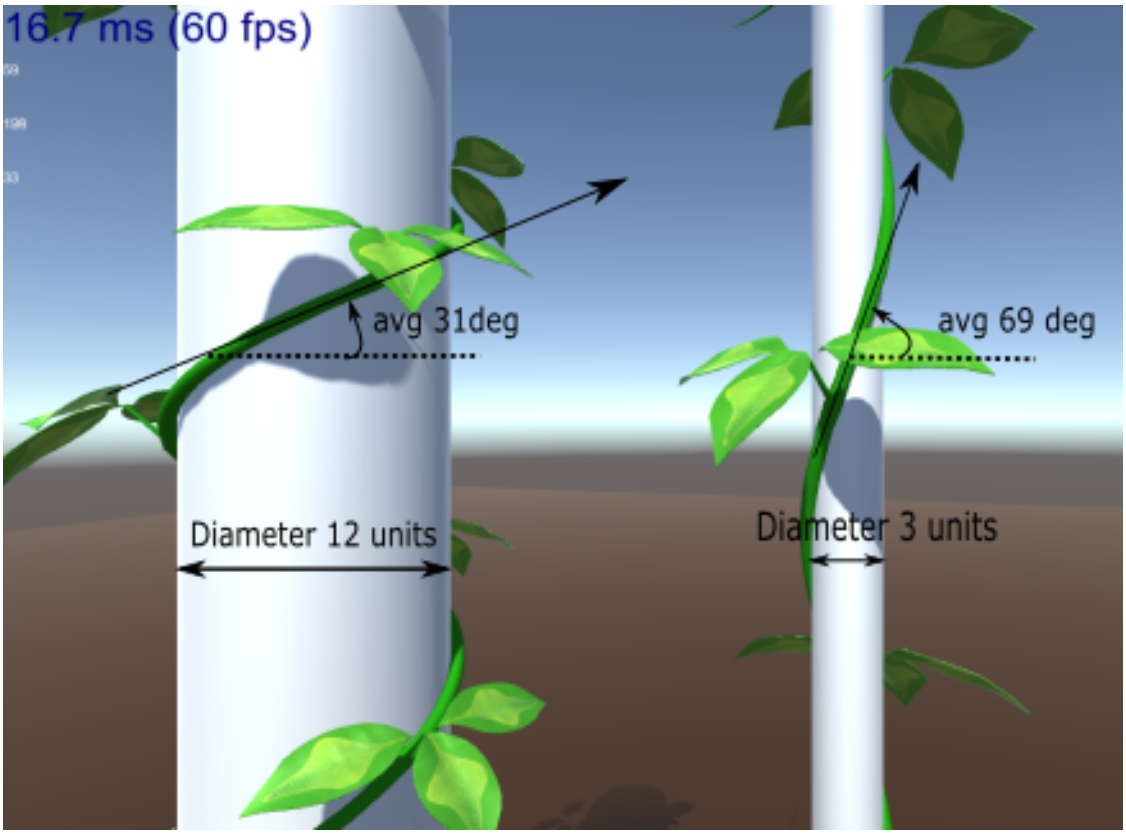

Figure 5.14: How the newly suggested model simulated pitch deference according to Diameter of the support structure

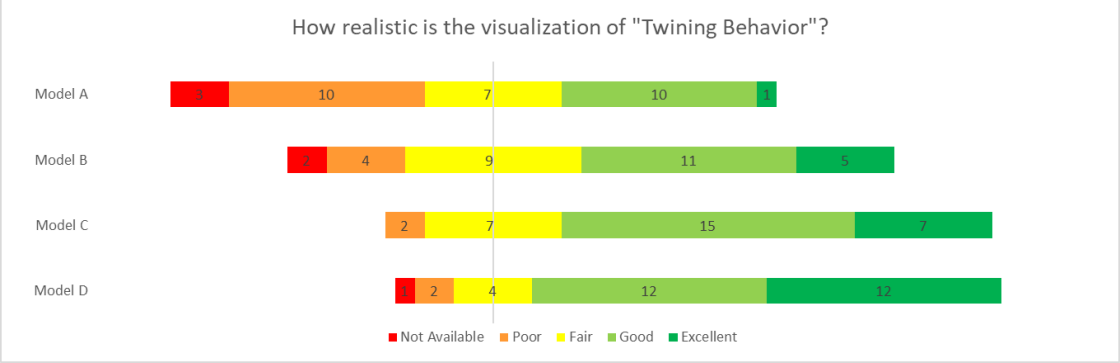

Figure 5.15: Diverging stacked bar chart for survey question "How realistic is the visualization of Twining Behavior?"

The result of survey can be visualized in a diverging stacked bar charts as in Figure 5.15. Table 5.2 shows the related statistical measures.Though model A B and D clearly shows twining behavior several participants have marked the "not available" option. Model C not having a "not available" option gives an insight for the recorded "not available" responses in all other models. Most of the participants have misidentified the lack of groping behavior as the absence of twining behavior. Model C having the second-highest positive percentage which is contradictory

Model	Mode	Median	Negatives%	Positives%
model A	Poor and Good	Fair	42%	35%
model B	Good	Good	19%	52%
model C	Good	Good	6%	71%
model D	Good and Excellent	Good	10%	77%

Table 5.2: Statistical measures of survey result for question "How realistic is the visualization of Twining Behavior?"

considering the earlier observation in self-comparison can be explained using two major assumptions. First is that participants failed to identify the lack of twining effect in model C (The model By Gunawardena et al couldn't simulate more than one round around support structure. Instead it kept climbing upward using grouping behavior). Second is that the model C was able to simulate groping behavior more realistically compared to others. Still, we can justify that our model which is model D simulated the twinning behavior better compared to others by having the highest recorded "excellent" response rate (39%) and the highest positive response rate (77%).

Considering both self-comparison and survey result analysis we can justify that our model performs the best compared to other models in simulation of the twining behavior of twining plants.

5.3.3.3 Girth of stem

The growth of a plant is categorized into two main growth categories. First, is the primary growth which is the elongation of the stem which can be observed in all considered related work suggested model. Second is the secondary(cambial) growth which is responsible for the growth of the girth of the stem. Wong and Chen's model and the super space clothoid model have not considered the growth of girth of the stem at all. In the interactive climbing plant authoring model by Hädrich et al, the cambial growth has been modeled by increasing the diameter of used anisotropic particles but the detail about the calculation of the increasing amount of diameter has not specified.

Gunawardhana's model suggested a taper function with oscillation to calculate the diameters. The oscillations were justified by pointing out the girth increments near stipules. This approach can be justified as in Gunawardhana's model as it did not simulate leaves. As the newly suggested model simulate leaves such function with oscillation can not be used blindly. This tapper function was not realistic as the oscillations were highly visible. In nature, these oscillations due to stipules are not visible but internode stem with monotonically decreasing girth is highlighted.

A comparison of related work suggested work nature is shown in Figure 5.16.

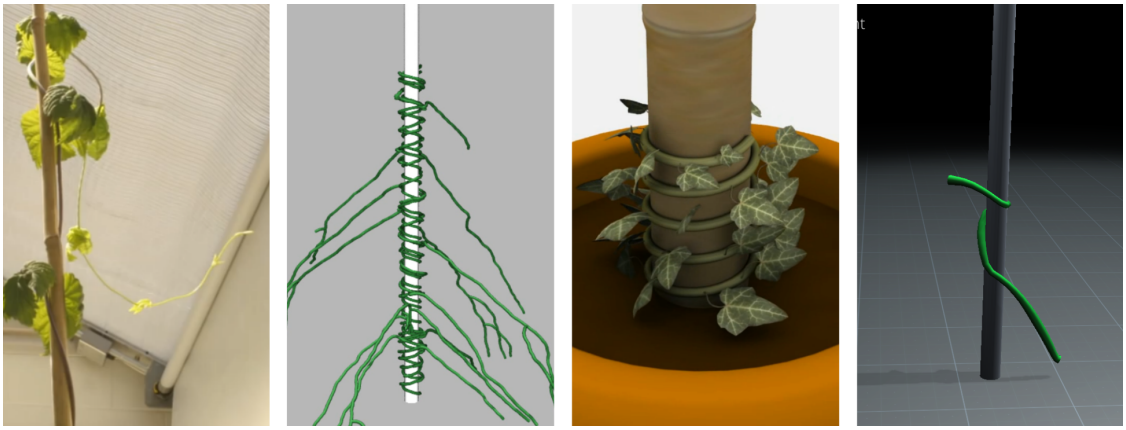

Figure 5.16: Comparison of girth of stem simulation from left to right in order Nature, Wong and Chen's model, Super space clothoid model, Gunawardhana's model

Our suggested model a monotonically decreasing function was used at all test scenarios and at the stipules the value returned by the tapper function was increased by 50 percent. Each new particle addition the girth was increased. Figure 5.17 shows visible monotonically decreasing girth throughout the whole stem and observable increased girth near stipules.

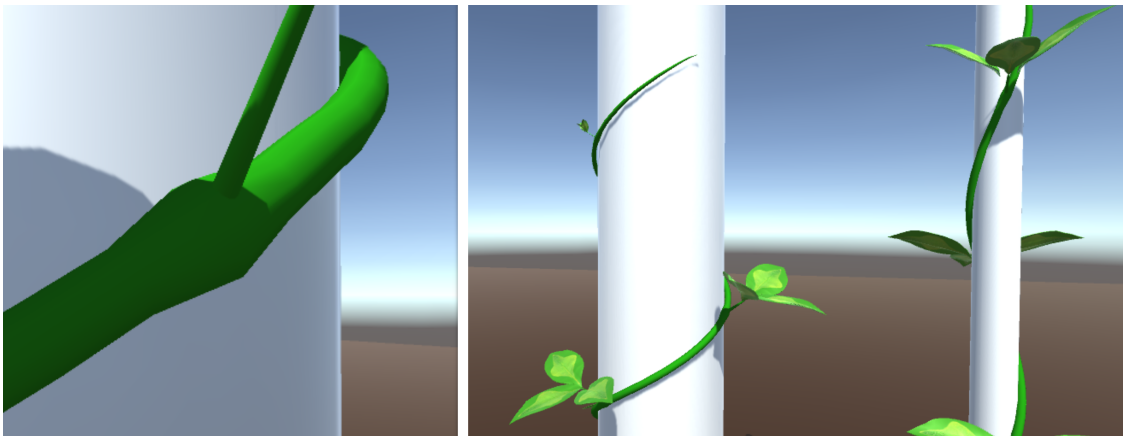

Figure 5.17: Observable increased girth near stipules in left. Generally seen monotonically decreasing stem towards tip in right

The result of the survey can be visualized in a diverging stacked bar chart as in Figure 5.18. Table 5.3 shows the related statistical measures. Though models A and B clearly show any specific girth simulations there were some positive responses. This is contradictory to our observations in self-comparison. Some participants misunderstood the simulation of girth as having realistic looking stem rather than simulating secondary growth but both these models had lesser positive response

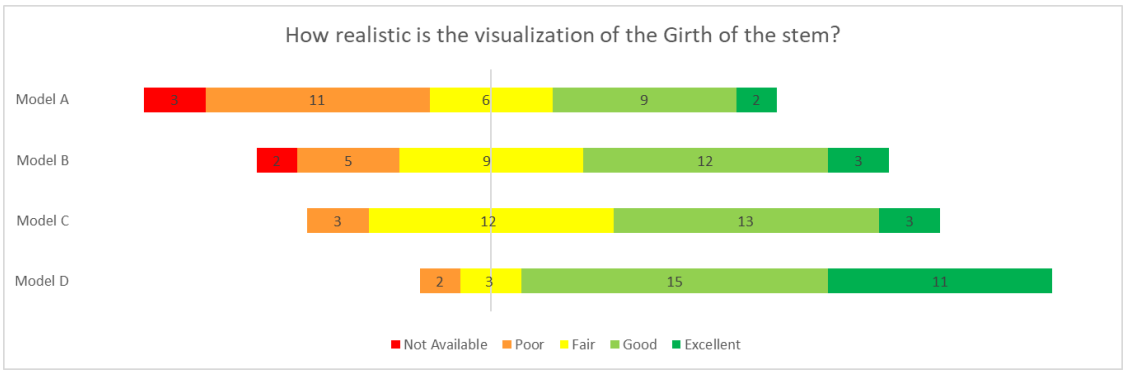

Figure 5.18: Diverging stacked bar chart for survey question "How realistic is the visualization of the Girth of the stem?"

Model	Mode	Median	Negatives%	Positives%
model A	Poor	Fair	45%	35%
model B	Good	Fair	23%	48%
model C	Good	Good	10%	52%
model D	Good	Good	6%	84%

Table 5.3: Statistical measures of survey result for question "How realistic is the visualization of the Girth of the stem?"

rates 35% and 48% compared to others. Model C and D both have over 50% positive response rate and negative response rate almost below 10%. The fact that model D which is the newly suggested model excels in the simulation of girth compared to others can be justified considering the following statistics. First, our newly suggested model was able to score 35% "excellent" responses and secondly, it scored 84% total positive response percentage. Both were highest recorded across all models. Thirdly it had the least negative score which is 6%.

Considering both self-comparison and survey result analysis we can justify that our model performs the best compared to other models in simulation of the secondary growth of twining plants.

5.3.3.4 Seed germination process

The germination process has not simulated by any of the considered related works including PBD based relative rotational model by Gunawardhana et al but our suggested model simulated this to a certain level. Figure 5.19 shows a comparison between nature and our suggested model.

The result of the survey can be visualized in a diverging stacked bar chart as in Figure 5.20. Table 5.4 shows the related statistical measures. Though models A, B and C clearly do not show any germination process there were some positive

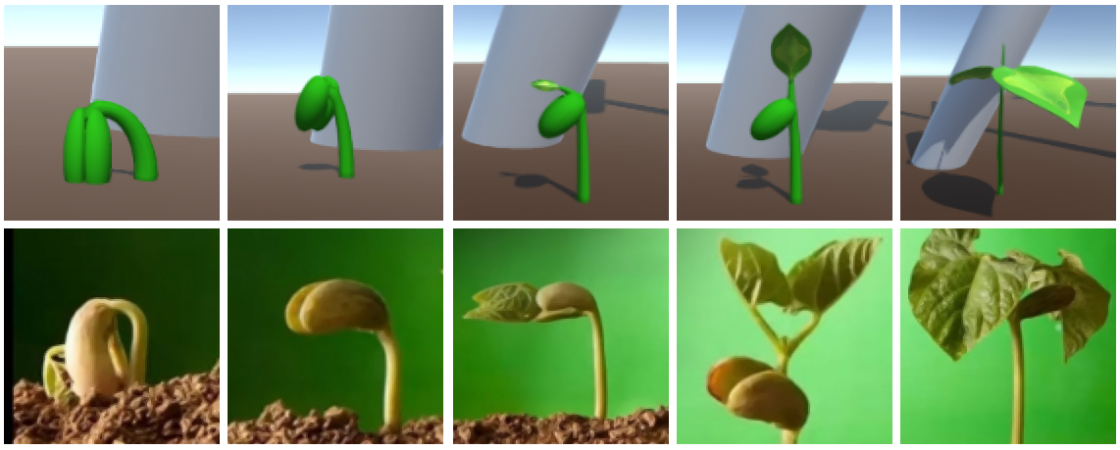

Figure 5.19: Comparison of germination process result of suggested model is at upper picture row. nature observations at lower picture process. Time flows left to right in picture rows.

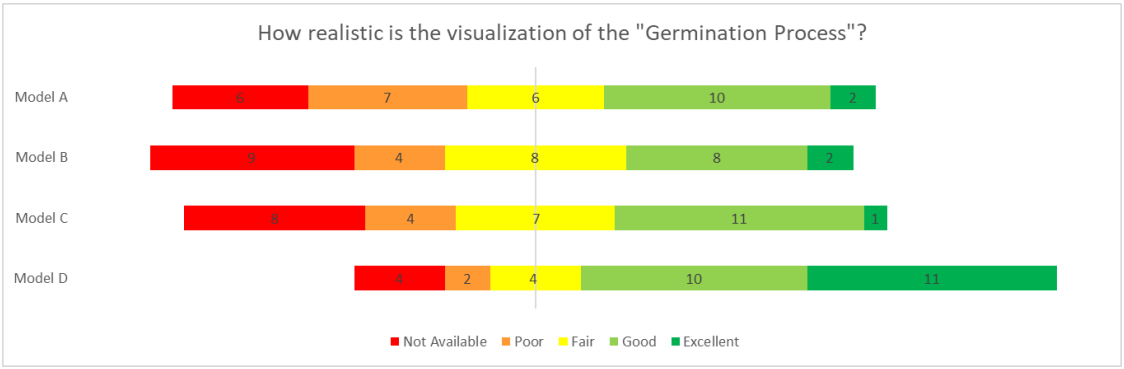

Figure 5.20: Diverging stacked bar chart for survey question "How realistic is the visualization of the Germination Process?"

Model	Mode	Median	Negatives%	Positives%
model A	good	Fair	42%	39%
model B	Not available	Fair	42%	32%
model C	Good	Fair	39%	39%
model D	Excellent	Good	19%	68%

Table 5.4: Statistical measures of survey result for question "How realistic is the visualization of the Germination Process?"

responses and model D had some not available response as well. This is contradictory to our observations in self-comparison. But models A, B, and C all three had below 50% positive response rate while model D had 68% highest recorded positive response rate. Other than that Model D had the lowest recorded negative response rate as well as the highest excellent response rate as well. Considering both self-comparison and survey result analysis we can justify that our model performs the

best compared to other models in simulation of the germination process of twining plants.

5.3.3.5 Leaf growth

A number of specific models exist that simulate the growth of leaves biologically accurately. None of the related work suggested models that are discussed here have not focused on this matter. However, as the simulation of leaf growth improves the visual realism of the whole model some of the models have implemented the simple leaf growth mechanism. To simulate the impact of stipules on the girth distribution of stem properly it was necessary to model the leaves. Therefore the leaf growth was implemented to a certain level In our model.

Both Wong and Chen's model and superspace clothoid model simulated the leaves using 2d plane mesh with textures. No growth was simulated up-scaling the mesh. No blooming effect was simulated. In our model, the growth of leaves was simulated but up-scaling the 3D mesh and transforming accordingly to simulate the blooming effect. Leave traversal due to primary growth of stem is not simulated in Wong and Chen's model but it was implemented in both our model and superspace clothoid model. The leaf traversal was visually unrealistic in the superspace clothoid model as it the travel distance was higher than what we observed in nature. In our model, this was solved by giving a public parameter to control the traveling distance.

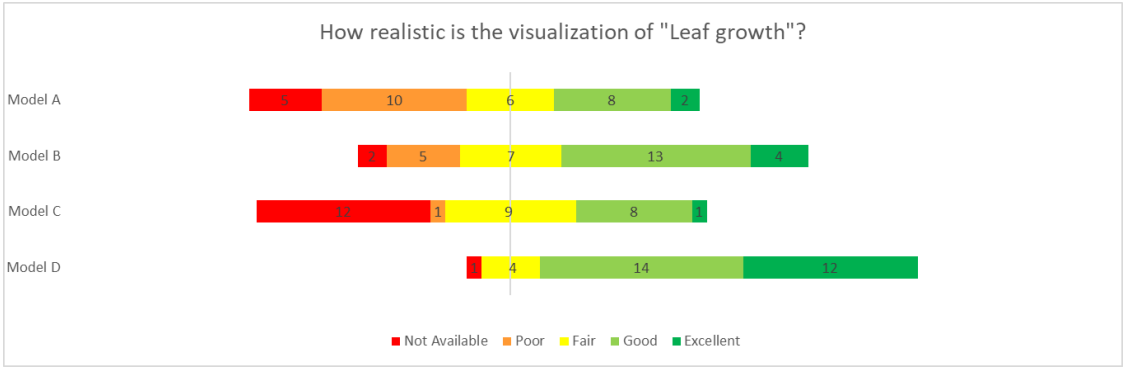

Figure 5.21: Diverging stacked bar chart for survey question "How realistic is the visualization of Leaf growth?"

The result of the survey can be visualized in a diverging stacked bar chart as in Figure 5.21. Table 5.5 shows the related statistical measures. Though models C clearly has not simulated any leaves there were some positive responses. This is contradictory to our observations in self-comparison. However, model D had the least positive response rate. Only Model B and D have over 50% positive response rate which aligns with result of self-comparison. model D had 84% highest recorded

Model	Mode	Median	Negatives%	Positives%
model A	poor	Fair	48%	32%
model B	Good	Good	23%	55%
model C	Not available	Fair	42%	29%
model D	Good	Good	03%	84%

Table 5.5: Statistical measures of survey result for question "How realistic is the visualization of Leaf growth?"

positive response rate and lowest recorded negative response rate (3%). Other than that the highest excellent response rate (39%) was also recorded in model D. Considering the implementation of the blooming effect that was discussed in self-comparison and survey result analysis we can justify that our model performs the best compared to other models in simulation of leaf growth in twining plants.

5.3.3.6 Overall twining plant's biological and biomechanical behaviors

Considering the visual realism of each biological and biomechanical behavior simulated by each model following Table 5.6 was produced as a summery of all earlier discussed self-comparisons. We believe our model was able to surpass all other models in almost all criteria as it simulates the most of the biological and biomechanical behaviours of a twining plant more accurately than other models.

Behaviour	Wong and Chen's model	Superspace Clothoid based model	PBD based model	Our model
Circumnutation Behaviour	Not available	Not available	Simulation was jittery	Smoothly circumnutational growth
Twining behavior	No groping behaviour. Stable twining mechanism.	Groping behaviour was slightly visible. Stable twining mechanism.	Well simulated groping behaviour but twining was unstable and could not twine more than one round.	Groping behavior was simulated to a certain level. Stable twining mechanism.
Secondary growth	Uniform girth throughout the whole simulation	Uniform girth throughout the whole simulation	Oscillating taper function and visible secondary growth	Logarithm based taper function. Increased girth near stipules.
Germination process	Not available	Not available	Not available	Simulated the germination to a certain level. Rooting was not simulated
Leaf growth	No visible growth and traversal effect.	Growth and traversal effect was simulated. No blooming effect	Not available	Growth, traversal effect, blooming effect was simulated

Table 5.6: Summery of self-comparison

The result of the survey to the question: "How realistic is the visualization of overall twining plant's biological and biomechanical behaviors?" can be summarised using a diverging stacked bar chart as shown in Figure 5.22. Table 5.7 shows the related statistical measures. Considering the 87% positive response rate and 100% non-negative response rate the fact that our model was able to simulate

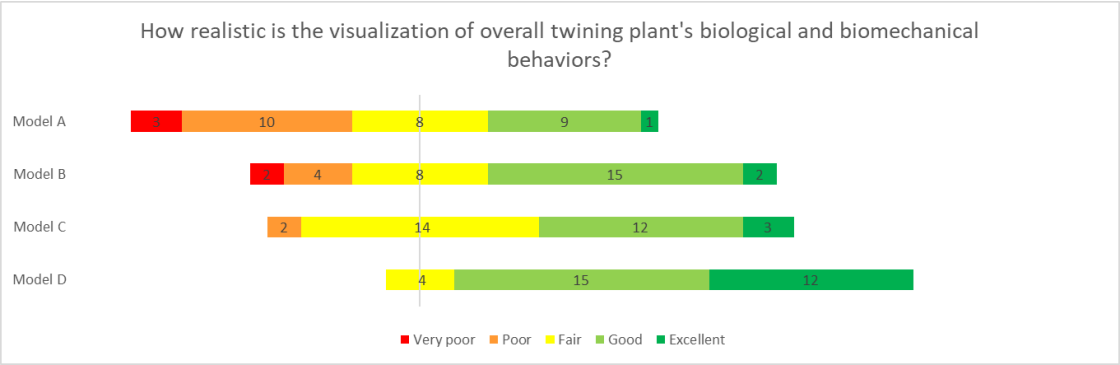

Figure 5.22: Diverging stacked bar chart for survey question "How realistic is the visualization of overall twining plant's biological and biomechanical behaviors?"

Model	Mode	Median	Negatives%	Positives%
model A	poor	Fair	42%	32%
model B	Good	Good	19%	55%
model C	Fair	Fair	06%	48%
model D	Good	Good	00%	87%

Table 5.7: Statistical measures of survey result for question "How realistic is the visualization of overall twining plant's biological and biomechanical behaviors?"

the biological and biomechanical behaviors more visually realistically compared to others can be justified. Having a 48% positive response rate and 6% negative response rate model C was the second-best model according to the result of the survey.

The combined result of self-comparison and survey result we can conclude that our model was able to simulate twining plant significantly better than other exiting related work suggested models.

Chapter 6

Conclusions

6.1 Introduction

In this chapter research problem, research aim, research question and will be revisited and analyzed. Then the limitations of the work will be discussed and further research implications will be discussed.

6.2 Conclusions about research questions

The aim of this research was to synthesize a generalized model to simulate a biologically and biomechanically accurate climbing plant. None of the models suggested by related work were able to simulate even the basic biological and biomechanical behaviors except for the PBD based model suggested by Gunawardhana et al. Through the model suggested by Gunawardhana et al was able to simulate the biological and biomechanical behaviors to a certain extent it suffered from a stability issue and whole simulation was jittery due to that. In the initial stage, the research questions were formed considering the option to extend and improve the model suggest by Gunawardhana et al.

In the earlier stage, a critical analysis of the PBD based model suggests by Gunawardhana et al was carried out. In this analysis, all the limitations and issues with PBD based model were identified. Considering the identified limitation of the model two path to improve the Gunawardhana's model was proposed. PBD and Gunawardhana's implementation were deeply studied and due to limitations that are inherent in general PBD approaches and Gunawardhana implementation both two proposed paths had to be abandoned. Therefore the option to extend and improve PBD based model was rejected. The whole process up to this rejection of the PBD based approach answers the first research question which is "Does the current approach of implementation provide enough facilities to improve the

performance as well as stability efficiently? If possible how it can be achieved?". Here the phrase "current approach of implementation" was used in context to refer to Gunawardhana's PBD based model.

Earlier mentioned rejection of the option to extend and improve the model suggested by Gunawardhana et al leads to the second research question which is "If not what are the alternative approaches? How to implement them?". After abandoning the PBD approach we proposed a new particle-based state-machine driven model. This model was able to simulate the Biological and biomechanical behaviors more realistically compare to others. Most of the issues and limitations identified in the PBD based approach was solved by our Novel model. Compare to all other twining plant simulation models available, our model proved to be more realistic and accurate by the result of the conducted survey. This covers the second research question.

All next research questions were formed to take into the scope of the research in case that Gunawardhana's approach was able to be extended. Though we proposed a novel model instead of extending the model suggested by Gunawardhana et al we were able to answer some of the questions to a certain level. The Third question was "How to model the effects of external forces on circumnutation behavior?".Modeling the interaction with subtle external forces like wind, the rain was intended by this question. At the current stage, our model cannot interact with external forces. Neither other related work suggested models nor PBD based models suggested by Gunawardhana et all were able to interact with subtle external forces as well. How ever, as our approach is a particle-based approach this can be facilitated by the use of available fluid dynamic frameworks. The model (This model was created to simulate Ivy plant-like climbing plants, not for twining plants. biological or biomechanical behaviors simulated) suggested by Hadrich et al[1] which employed a similar particle-based approach was able to simulate interaction with subtle external forces using a fluid dynamic framework. The same method used in their model can be easily adapted to our model.

The fourth question was "How to extend the climbing model to tackle down the climbing on differently shaped supports?" Initial Gunawardhana's model was only able to climb straight cylindrical support structures. By this question, it was intended to resolve this limitation. However, our suggested model was able to interact with a differently shaped support structure successfully.

Gunawardhana et al suggested an oscillating taper function to simulate the girth of the stem. The suggested function was specific to a long bean species. By this fifth research question, it was intended to find a way to generalize simulation of girth. In our model, instead of a specific taper function to simulate girth distribution near stipules a general taper function base on logarithms was used. Using this taper

function's parameter the simulation of girth distribution was easily adjustable to preferences. Inter stipule distance also was adjustable to preferences. As our model gave a parameter to change the percentage increase near stipules instead relying on specific taper function.

Considering all of the above-mentioned design choices, implementations and conducted analyses the fact that our model was able to sufficiently answer all research questions can be justified.

6.3 Conclusions about research problem

With our novel model, we were able to address the problem, lack of climbing plant behavioral simulation models that show biologically and biomechanically accurate behaviors. Our model was able to simulate biologically and biomechanically accurate behaviors using a particle-based state-machine driven approach. Our model was able to overcome the limitation of PBD based approach and it was able to simulate more behaviors in a robust manner. Most importantly our model was able to simulate support finding mechanisms of plants without relying on ray-casting techniques but solely based on biomechanical behaviors. This allowed a more realistic simulation of twining plants which surpasses the state of the art twining plant simulation models.

6.4 Limitations

The model suggested by this research is a basic model that was created as a proof of concept rather than a comprehensive model. To achieve a more realistic simulation following factors should be considered.

The current implementation of the model lacks the ability to interact with subtle forces like wind and rain. External interaction like trimming, braking the stem is not allowed. Neither branching nor fruit-bearing ability was not modeled in by the current model. Breaking of the plant bark, aging, and falling of the leaves has to be model to have finer simulation.

6.5 Implications for further research

The particle-based approach combined with a state machine as a driving mechanism makes a rather flexible model. The design choice to employ particles and state machines were specifically made considering possible extension options.

Branching of twining plants can be integrated into the current model after study on twining plant branching patterns. The introduction of randomness to the current model would improve visual realism. The integration of fluid dynamics to the current model to simulate the effect of subtle external forces like rain and wind is another research implication.

References

[1] T. Hädrich, B. Benes, O. Deussen, and S. Pirk, "Interactive Modeling and Authoring of Climbing Plants," *Computer Graphics Forum*, vol. 36, no. 2, pp. 49–61, 2017.

[2] R. Vidoni, T. Mimmo, and C. Pandolfi, "Tendril-based climbing plants to model, simulate and create bio-inspired robotic systems.," *Journal of Bionic Engineering 12, no. 2*, p. 250–262, 2015.

[3] J. P. andenbrink and J. Z. Kiss., "Space, the final frontier: A critical review of recent experiments performed in microgravity.," *Plant Science 243*, p. 115–119, 2016.

[4] S. Pirk, M. Jarzabek, T. Hadrich, D. L. Michels, and W. Palubicki, "Interactive wood combustion for botanical tree models.," *ACM Transactions on Graphics 36, no. 6*, p. 1–12, 2017.

[5] B. S. Gunawardhana, C. J. Jayalath, and M. Wickramasinghe., "Simulation of climbing plants with twining behavior using position based dynamics framework and cosserat model.."

[6] C. Darwin, "On the movements and habits of climbing plants," *"Journal of the Linnean Society of London, Botany vol. 9, no. 33-34, pp"*, pp. 1–118, 1865.

[7] F. E. Putz, "Vine ecology," 2012. Online.

[8] A. J. Bowling and K. C. Vaughn., "Gelatinous fibers are widespread in coiling tendrils and twining vines.," *American Journal of Botany 96*, p. 719–727, 2009.

[9] A. Goriely and S. Neukirch, "Mechanics of climbing and attachment in twining plants," *Physical Review Letters*, vol. 97, no. 18, pp. 1–4, 2006.

[10] K. Singh, Puneet, and C. M. Krishna, "Virtual climbing plants competing for space.," *In Proceedings of Computer Animation 2002*, p. 193–98, 2002.

[11] S. K. Wong and K. C. Chen, "A Procedural Approach to Modelling Virtual Climbing Plants With Tendrils," *Computer Graphics Forum*, vol. 35, no. 8, pp. 5–18, 2016.

[12] A. H. Brown, "Circumnutations: From Darwin to Space Flights," *Plant Physiology*, vol. 101, no. 2, pp. 345–348, 1993.

[13] C. A. Esmon, U. V. Pedmale, and E. Liscum, "Plant tropisms: Providing the power of movement to a sessile organism," *International Journal of Developmental Biology*, vol. 49, no. 5-6, pp. 665–674, 2005.

[14] R. Chen, E. Rosen, and P. H. Masson, "Update on Development Gravitropism in Higher Plants 1," *Plant Physiology*, vol. 120, no. June, pp. 343–350, 1999.

[15] M. IINO, "Phototropism: mechanisms and ecological implications," *Plant, Cell & Environment*, vol. 13, no. 7, pp. 633–650, 1990.

[16] L. Reinhold, T. Sachs, and L. Vislovska, "The Role of Auxin in Thigmotropism," *Plant Growth Substances 1970*, no. 1884, pp. 731–737, 2011.

[17] D. DIetrich, L. Pang, A. Kobayashi, J. A. Fozard, V. Boudolf, R. Bhosale, R. Antoni, T. Nguyen, S. Hiratsuka, N. Fujii, Y. Miyazawa, T. W. Bae, D. M. Wells, M. R. Owen, L. R. Band, R. J. Dyson, O. E. Jensen, J. R. King, S. R. Tracy, C. J. Sturrock, S. J. Mooney, J. A. Roberts, R. P. Bhalerao, J. R. DInneny, P. L. Rodriguez, A. Nagatani, Y. Hosokawa, T. I. Baskin, T. P. Pridmore, L. De Veylder, H. Takahashi, and M. J. Bennett, "Root hydrotropism is controlled via a cortex-specific growth mechanism," *Nature Plants*, vol. 3, no. May, 2017.

[18] H. S. Atamian, N. M. Creux, E. A. Brown, A. G. Garner, B. K. Blackman, and S. L. Harmer, "Circadian regulation of sunflower heliotropism, floral orientation, and pollinator visits," *Science*, vol. 353, no. 6299, pp. 587–590, 2016.

[19] T. Kato, M. Morita, and M. Tasaka, "Role of endodermal cell vacuoles in shoot gravitropism," *Journal of Plant Growth Regulation*, vol. 21, no. 2, pp. 113–119, 2002.

[20] C. W. Whippo and R. P. Hangarter, "Phototropism: Bending towards Enlightenment," *The Plant Cell*, vol. 18, no. 5, pp. 1110–1119, 2006.

[21] J. M. Christie and A. S. Murphy, "Shoot phototropism in higher plants: New light through old concepts," *American Journal of Botany*, vol. 100, no. 1, pp. 35–46, 2013.

[22] C. Gutjahr, M. Riemann, A. Müller, P. Düchting, E. W. Weiler, and P. Nick, "Cholodny-Went revisited: A role for jasmonate in gravitropism of rice coleoptiles," *Planta*, vol. 222, no. 4, pp. 575–585, 2005.

[23] S. Isnard, A. R. Cobb, N. M. Holbrook, M. Zwieniecki, and J. Dumais, "Tensioning the helix: A mechanism for force generation in twining plants," *Proceedings of the Royal Society B: Biological Sciences*, vol. 276, no. 1667, pp. 2643–2650, 2009.

[24] J. Bloomenthal, "Modeling the mighty maple," *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 305–311, 2005.

[25] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Berlin, Heidelberg: Springer-Verlag, 1990.

[26] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane, "The use of positional information in the modeling of plants," pp. 289–300, 2005.

[27] P. Prusinkiewicz, "Modeling of spatial structure and development of plants: A review," *Scientia Horticulturae*, vol. 74, no. 1-2, pp. 113–149, 1998.

[28] J. L. Power, A. J. B. Brush, P. Prusinkiewicz, and D. H. Salesin, "Interactive arrangement of botanical L-system models," pp. 175–182, 2004.

[29] J. Arvo and D. Kirk, "Modeling plants with environment-sensitive automata," *Proceedings of Ausgraph '88*, pp. 1–8, 1988.

[30] N. Greene, "Voxel space automata: Modeling with stochastic growth processes in voxel space," *SIGGRAPH Comput. Graph.*, vol. 23, pp. 175–184, July 1989.

[31] P. de Reffye, C. Edelin, J. Françon, M. Jaeger, and C. Puech, "Plant models faithful to botanical structure and development," *SIGGRAPH Comput. Graph.*, vol. 22, pp. 151–158, June 1988.

[32] R. Casati and F. Bertails-Descoubes, "Super space clothoids," *ACM Transactions on Graphics*, vol. 32, no. 4, p. 1, 2013.

[33] F. Bertails-Descoubes, "Simulation of fibrous material Mechanical models and numerical simulation Research area," *Nonsmooth contact mechanics. Spring school. Aussois 2010*, no. June, 2010.

[34] J. Linn, H. Lang, and A. Tuganov, "Geometrically exact cosserat rods with kelvin-voigt type viscous damping," vol. 4, 05 2012.

[35] M. Müller Bruno Heidelberger Marcus Hennix John Ratcliff AGEIA, "Position Based Dynamics," *VRIPHYS*, 2006.

[36] R. Nothnagel, "TubeRenderer from unify community wiki." `https://wiki.unity3d.com/index.php/TubeRenderer`. Accessed: 2010-02-02.

Appendices

Appendix A

Publications

This is a appendix. Behaves same as a chapter.

Appendix B

Diagrams

Appendix C

Code Listings

C.1 Frame reduction and concatenating script

```
tmpFile="asd";
fileList="fileList.txt";
ffmpeg="G:/PlantVid/ffmpeg-20190614-dd357d7-win64-static/bin/ffmpeg";
for i in *;do
         if [ -d "$i" ]; then
                name=`echo "${i%.*}"`;
                echo "###################${name}###################";
                cd "${name}";
                mkdir "${name}_frameReduced";
                for j in *.h264;do
                        vid=`echo "${j%.*}"`;
                        echo "        ${vid}";
                        $ffmpeg -i "${vid}.h264" -filter:v "setpts=0.02*PTS" -an
                        ↪  "${name}_frameReduced" / "${vid}_frameReduced.mp4";
                done
                cd "${name}_frameReduced";
                mkdir "${name}_hyperLapsed";
                for k in *.mp4;do
                        reducedVid=`echo "${k%.*}"`;
                        echo "                ${reducedVid}";
                        $ffmpeg -i "${reducedVid}.mp4" -filter:v "setpts=0.02*PTS" -an
                        ↪  "${name}_hyperLapsed" / "${reducedVid}_hyperLapsed.mp4";
                done

                cd "${name}_hyperLapsed";
                ls -1 *\.mp4 | sort -n >"${tmpFile}";
                awk '{print "file " $0;}' "${tmpFile}"> "filelist.txt";
                rm "asd";
                mkdir "concatinated";
                $ffmpeg -f concat -safe 0 -i "${fileList}" -c copy "concatinated" /
                ↪  "${name}_concatinated.mp4";
                cd ..;
                cd ..;
                   cd ..;
        fi
done
read -p "$*";
```

C.2 Position based dynamics algorithm

(1) **forall** vertices i
(2) initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3) **endfor**
(4) **loop**
(5) **forall** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t\, w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6) dampVelocities$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(7) **forall** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(8) **forall** vertices i **do** generateCollisionConstraints$(\mathbf{x}_i \rightarrow \mathbf{p}_i)$
(9) **loop** solverIterations **times**
(10) projectConstraints$(C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N)$
(11) **endloop**
(12) **forall** vertices i
(13) $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(14) $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(15) **endfor**
(16) velocityUpdate$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(17) **endloop**

C.3 Bine script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using UnityEngine.Profiling;


public class Bine : MonoBehaviour {
    bool hasWrittenreport = false;
    public static bool globalWriteTheReport = false;
    int lastEntry;
    int usedMemory;
    bool end = false;
    int fps;
    int reportIterator;
    int[,] report;
    int inTwineConsecutiveTimes = 0;
    bool[] nodes;
    float nutationalDirection =1f;
```

```
GameObject previiousBead;
Vector3 targetLastBeadPosition;
float growthSpeed = 1;
float InterNodeGrowth = 0;
public int initialPhaseBeadCount;
public bool inInitialPhase = true;
public float initialPhaseInterBeadDistance;
public int leafTravelDistance;
public int beedGapPerLeaf;
bool addLeaf = true;
public int currentNumberOfBeeds = 0;
public int currentNumberOfLeaves = 0;
public float leafGrowthRate;
public GameObject seed;
public GameObject leaf;
public GameObject secLeaf;
private GameObject newLeaf;
private int frameCount = 0;
public int growthTime = 50;
public int particleCount = 10;
public GameObject beed;
public float radius = 1;
public GameObject[] plant;
GameObject[] leaves;
public Vector3[] cn;
GameObject lastBeed;
public float beedDistance = 1;
public Vector3 maxBeedRotation = new Vector3(5, 5, 5);
public Vector3 circumnutationSpeed = new Vector3(0, 5, 0);
public bool circumnutationOn = true;
private Collision beedCollision;
private GameObject thisGameObject;
private bool supportFound=false;
public int startingBeed = 0;
private Vector3 newGrowthDirection;
private Vector3 collisionNormal;
public float gravitrophismLimit = 45;
public float gravitrophismCorrectionValue = 0.01f;
public float gravitrophismAbsoluteLimit = 80;
private bool supportLost=false;
private Vector3 theUpward = new Vector3(0, 100, 0);
public float supportLostGravitrophismAdjustment = 0.01f;
public float reCircumnutateReadyAngle = 45;
public float growthAngleBias = 30;
public float supportedCircumnutationSpeed = 1;
TubeRenderer tubeRenderer;
Vector3[] beedPositions;
float[] girths;
public float maxGirth = 2;
public bool hasGroped = false;
private bool IKInitiated = false;
public int NoOfIKIterations = 50;
private bool groping = false;
Collider collidingObject;
Collider collidingBeed;
Leaf theLeaf;
GameObject psudoGameObject;
int numberOfBeasTillCotyldons = 11;
```

```
bool growWithoutSupport = false;
bool growAroundSupport = false;
bool getReadyToCircumnutate = false;

bool seedRisen = false;
bool primaryLeafRisen = false;
bool primaryLeafGrown =false;

public float nutatingAmountPerTurn = 5;
float nutatingAmountPerTurnChange = 1;
public float speed;
public float upwardTurningSpeed;
bool isStemOverGrown = false;
bool localWriteTheReport = false;
public float stipulesGirthIncrement = 1.3f;

private void Awake()
{
    usedMemory = 0;
    lastEntry = 0;
    reportIterator = 0;
    thisGameObject = this.gameObject;
    report = new int[2000,4];
    plant = new GameObject[particleCount];
    nodes = new bool[particleCount];
    nodes[numberOfBeasTillCotyldons] = true;
    nodes[numberOfBeasTillCotyldons+1] = true;
    leaves = new GameObject[particleCount / beedGapPerLeaf];
    cn = new Vector3[particleCount];
    beedPositions = new Vector3[particleCount];
    girths = new float[particleCount];
    lastBeed = Instantiate(beed, transform.position, transform.rotation);
    lastBeed.name = thisGameObject.name + "_beed_0";
    lastBeed.transform.parent = thisGameObject.transform;
    //Vector3 initRotation = new Vector3(270, 0, 0); //sphericalbeed
    tubeRenderer = thisGameObject.GetComponent<TubeRenderer>();

    Vector3 initRotation = new Vector3(270, 0, 0); //cylindericalBeed

    lastBeed.transform.Rotate(initRotation);
    plant[currentNumberOfBeeds] = lastBeed;
    currentNumberOfBeeds++;
    //theLeaf = leaf.GetComponent<Leaf>();
    //theLeaf.FirstLeaf = true;
    //theLeaf.plant = plant;
    //theLeaf.destinationBeadNumber = 5;
    //theLeaf.currentBeadNumber = currentNumberOfBeeds;
    //leaves[0] = leaf;
    //currentNumberOfLeaves++;
}

void Start() {
    //plant initail stage building
    psudoGameObject = new GameObject();
    float curve = 270;
    float curveAmount = 3;
    lastBeed.transform.eulerAngles=new Vector3(250, 180,180);
    Vector3 Beanforward = lastBeed.transform.eulerAngles;
    //Debug.Log("startingbeadForward.Y: " + Beanforward+ "\n");
```

```
        //Debug.Log("begins \n");
        upwardTurningSpeed = upwardTurningSpeed* speed;
        while (curve > 90)
        {
            curveAmount += 3;
            curve -= curveAmount;
            Vector3 newPosition = lastBeed.transform.position + lastBeed.transform.forward *
            ↪  initialPhaseInterBeadDistance;
            // removing the colider of previouse beed
            DestroyImmediate(lastBeed.GetComponent<Collider>());
            DestroyImmediate(lastBeed.GetComponent<Beed>());
            lastBeed = Instantiate(beed, newPosition, lastBeed.transform.rotation);
            lastBeed.name = "Beed" + currentNumberOfBeeds;
            lastBeed.transform.parent = thisGameObject.transform;
            lastBeed.transform.eulerAngles = new Vector3(curve,0, 0);
            plant[currentNumberOfBeeds] = lastBeed;
            Beanforward = lastBeed.transform.rotation.eulerAngles;
            //Debug.Log("     beadForward.Y: " + Beanforward+ "\n");
            currentNumberOfBeeds++;
        }

        seed.transform.position = lastBeed.transform.position;


        for (int i = 0; i < 5; i++)
        {
            Vector3 newPosition = lastBeed.transform.position + lastBeed.transform.forward *
            ↪  0.000001f;
            // removing the colider of previouse beed
            Vector3 prevForward = lastBeed.transform.forward;
            DestroyImmediate(lastBeed.GetComponent<Collider>());
            DestroyImmediate(lastBeed.GetComponent<Beed>());
            lastBeed = Instantiate(beed, newPosition, lastBeed.transform.rotation);
            lastBeed.name = "Beed" + currentNumberOfBeeds;
            lastBeed.transform.forward = prevForward;
            lastBeed.transform.parent = thisGameObject.transform;
            plant[currentNumberOfBeeds] = lastBeed;
            currentNumberOfBeeds++;
        }

        //seed.transform.eulerAngles =

        //Debug.Log("ends \n");

    }

    private void FixedUpdate() {


        collectAnalytics();

        if (globalWriteTheReport && !hasWrittenreport)
        {
            hasWrittenreport = true;
            localWriteTheReport = true;
        }

        if (localWriteTheReport)
        {
```

```
            GenerateReport();
        }
        //Debug.DrawRay(lastBeed.transform.position,lastBeed.transform.forward,Color.cyan,1);
        //Debug.DrawRay(seed.transform.position, seed.transform.forward, Color.yellow);
        // plant growth
        if (inInitialPhase)
        {

            if (!seedRisen)
            {
                //Debug.Log("seed Rising\n");
                seedRises();
                //Debug.Log( "angle:
                ↪  "+Vector3.Angle(plant[currentNumberOfBeeds-1].transform.forward,
                ↪  Vector3.up)+"\n");
                if (Vector3.Angle(plant[11 - 1].transform.forward, Vector3.up) <= 100)
                {
                    seedRisen = true;
                    startingBeed = currentNumberOfBeeds - 2;

                }
            }

            else if (!primaryLeafRisen)
            {
                //Debug.Log("primary leaf Rising\n");

                raiseThePrimaryLeaves();
                if (Vector3.Angle(lastBeed.transform.forward, Vector3.up) <= 1f)
                {
                    lastBeed.transform.forward = Vector3.up;
                    seedRisen = true;

                    startingBeed = currentNumberOfBeeds - 2;
                    growWithoutSupport = true;
                    primaryLeafRisen = true;

                }
            }

            else if (!primaryLeafGrown)
            {
                //Debug.Log("primary leaf Growing\n");

                primaryLeafGrown = growingPrimaryLeves();
            }

            else
            {
                Debug.Log("initial phase ending\n");

                inInitialPhase = false;
            }
        }

        else if (growWithoutSupport)
        {
            SupportLessGrowth();
            growWithoutSupport = false;
```

```
        }

        else if(isStemOverGrown)
        {
            fallingStem();
        }

        else if (groping)
        {
            grope();
        }

        else if (growAroundSupport)
        {
            growWithSupportIntact();
            growAroundSupport = false;
        }

        else if (getReadyToCircumnutate)
        {
            getReadyToCircumnutate = false;
            supportLostGrowth();
        }

        else
        {

            if (circumnutationOn)
            {
                if (!supportFound)
                {
                    circumnutation();
                }
                else
                {
                    twine();
                }
            }

        }

        RenderTheStem();

        if (currentNumberOfBeeds > 2)
        {
            LeafGrowth();
        }
    }

    void growWithSupportIntact() {
        //Debug.Log("supportFoundGrowth\n");
        //Debug.DrawRay(beedCollision.contacts[0].point, collisionNormal, Color.green, 60*5,
        ↪  false);
        newGrowthDirection = collisionNormal;

        newGrowthDirection = Vector3.RotateTowards(lastBeed.transform.forward,
        ↪  newGrowthDirection, Mathf.Deg2Rad * growthAngleBias, 0.0f);
        //Debug.DrawRay(lastBeed.transform.position, collisionNormal*20, Color.yellow, 60 * 5,
        ↪  false);
```

```
        //Debug.DrawRay(lastBeed.transform.position, lastBeed.transform.forward, Color.blue, 60*5,
        ↪  false);
        //Debug.DrawRay(lastBeed.transform.position, newGrowthDirection * 2, Color.red, 60 * 5,
        ↪  false);

        //SupportedGrowth(newGrowthDirection);

        SupportedGrowth(lastBeed.transform.forward);


        //supportFound = false;
        circumnutationOn = true;
    }

    bool growingPrimaryLeves()
    {
        float growingPrimaryLevesSpeed = 1 * speed;
        float growthLimit =4;
        float nutationSpeed = growingPrimaryLevesSpeed * 5;
        float primaryleafGrowingSpeed = growingPrimaryLevesSpeed * 5;
        float stemGrowthSpeed = growingPrimaryLevesSpeed * 0.7f;
        float leafScalingSpeed = growingPrimaryLevesSpeed * 1;
        float rotateAmount = 0.0002f * growingPrimaryLevesSpeed;
        bool readyToleaveThisState = false;
        if (leaf.transform.localScale.x <= growthLimit)
        {
            initialPhaseInterBeadDistance += 0.01f* stemGrowthSpeed;
            for (int i = numberOfBeasTillCotyldons-1; i < currentNumberOfBeeds - 1; i++)
            {
                plant[i].transform.forward = Vector3.RotateTowards(plant[i].transform.forward,
                ↪  nutationalDirection*Vector3.right, rotateAmount* nutationSpeed, 0);
                Vector3 newPosition = plant[i].transform.position + plant[i].transform.forward *
                ↪  initialPhaseInterBeadDistance* stemGrowthSpeed;
                plant[i + 1].transform.position = newPosition;
                rotateAmount += nutatingAmountPerTurn*0.0001f * nutationSpeed;
            }

            lastBeed.transform.forward = Vector3.RotateTowards(lastBeed.transform.forward,
            ↪  nutationalDirection * Vector3.right, rotateAmount, 0);

            // chnage the nutating amount per each direction change;
            if (nutatingAmountPerTurn > 1)
            {
                nutatingAmountPerTurn -= 0.01f;
            }

            // Change nutational direction alternatively
            if (Vector3.Angle(plant[numberOfBeasTillCotyldons].transform.forward, Vector3.up) >
            ↪  nutatingAmountPerTurn)
            {
                nutationalDirection = nutationalDirection * -1;
            }

            leaf.transform.position = lastBeed.transform.position;

            //leaf.transform.forward = lastBeed.transform.forward;


            float spanningSpeed = 1.5f * growingPrimaryLevesSpeed;
```

```
        leaf.transform.GetChild(0).transform.localRotation =
        ↪  Quaternion.Lerp(leaf.transform.GetChild(0).transform.localRotation,
        ↪  Quaternion.Euler(0, -1 * 30, 0), 0.005f * spanningSpeed);
        leaf.transform.GetChild(1).transform.localRotation =
        ↪  Quaternion.Lerp(leaf.transform.GetChild(1).transform.localRotation,
        ↪  Quaternion.Euler(0, 30, 0), 0.005f * spanningSpeed);

        leaf.transform.localScale = new Vector3(leaf.transform.lossyScale.x + 0.01f *
        ↪  leafScalingSpeed, leaf.transform.localScale.y + 0.03f * leafScalingSpeed,
        ↪  leaf.transform.lossyScale.z + 0.01f * leafScalingSpeed);
    }
    else
    {
        // Once the leaf growth compleats re-arranging the stem after above nutation
        for (int i = numberOfBeasTillCotyldons - 1; i < currentNumberOfBeeds - 1; i++)
        {
            plant[i].transform.forward =
            ↪  Vector3.RotateTowards(plant[i].transform.forward,Vector3.up, rotateAmount,
            ↪  0);
            Vector3 newPosition = plant[i].transform.position + plant[i].transform.forward *
            ↪  initialPhaseInterBeadDistance * stemGrowthSpeed;
            plant[i + 1].transform.position = newPosition;
            rotateAmount += 0.001f * primaryleafGrowingSpeed;
        }

        lastBeed.transform.forward =
        ↪  Vector3.RotateTowards(lastBeed.transform.forward,Vector3.up, rotateAmount, 0);

        leaf.transform.position = lastBeed.transform.position;
        //leaf.transform.forward = lastBeed.transform.forward;

        //Debug.Log("£££" + leaf.transform.eulerAngles.y + "£££");

        // After adjusted enough leaving the Primary Leaf Growth State
        if (Vector3.Angle(Vector3.up, lastBeed.transform.forward) < 1)
        {
             Rigidbody rb = seed.GetComponent<Rigidbody>();
            rb.isKinematic = false;
            rb.useGravity = true;
            readyToleaveThisState = true;
            //leaf.transform.Rotate(new Vector3(0, 0, 90));

        }
    }
    // Addressing gimbelLock like issue hack to set forward direction of primary leaf
    if (seed.transform.GetChild(0).localScale.z > 0.05) {
        seed.transform.GetChild(0).localScale = seed.transform.GetChild(0).localScale +
        ↪  (Vector3.back * 0.0001f);
        seed.transform.GetChild(1).localScale = seed.transform.GetChild(1).localScale +
        ↪  (Vector3.back * 0.0001f);

    }
    psudoGameObject.transform.position = lastBeed.transform.position +
    ↪  lastBeed.transform.forward*5;
    leaf.transform.rotation = Quaternion.LookRotation(lastBeed.transform.forward,
    ↪  leaf.transform.up);
    Debug.DrawRay(leaf.transform.position, leaf.transform.forward, Color.black, 10);
    Debug.DrawRay(leaf.transform.position, leaf.transform.forward, Color.black, 10);
```

```
    return readyToleaveThisState;
}

void raiseThePrimaryLeaves() {
    initialPhaseInterBeadDistance += 0.0005f;
    float primaryLeavesRaisingSpeed =1;
    float stemGrowthSpeed =primaryLeavesRaisingSpeed*0.5f;
    float rotateAmount = 0.015f* primaryLeavesRaisingSpeed;
    for (int i = 10; i < currentNumberOfBeeds - 1; i++)
    {
        //plant[i].transform.rotation = Quaternion.Lerp(plant[i].transform.rotation,
        ↪  Quaternion.LookRotation(Vector3.up), Time.time * 0.0001f);
        //Vector3 newPosition = plant[i].transform.position + plant[i].transform.forward *
        ↪  initialPhaseInterBeadDistance;
        plant[i].transform.forward = Vector3.RotateTowards(plant[i].transform.forward,
        ↪  Vector3.up, rotateAmount, 0);
        Vector3 newPosition = plant[i].transform.position + plant[i].transform.forward *
        ↪  initialPhaseInterBeadDistance* stemGrowthSpeed;
        plant[i + 1].transform.position = newPosition;
        //plant[i + 1].transform.forward = plant[i].transform.forward;
        rotateAmount -= 0.002f* primaryLeavesRaisingSpeed;
    }
    lastBeed.transform.forward = Vector3.RotateTowards(lastBeed.transform.forward,
    ↪  Vector3.up, rotateAmount, 0);

    leaf.transform.position = lastBeed.transform.position;
    leaf.transform.forward = lastBeed.transform.forward;
    float growthSpeed = 0.2f*primaryLeavesRaisingSpeed;
    if (leaf.transform.localScale.x < 7)
    {
        leaf.transform.localScale = new Vector3(leaf.transform.lossyScale.x + 0.01f*
        ↪  growthSpeed, leaf.transform.localScale.y + 0.03f* growthSpeed,
        ↪  leaf.transform.lossyScale.z + 0.01f * growthSpeed);
    }

    float spanningSpeed = 0.2f*primaryLeavesRaisingSpeed;
    leaf.transform.GetChild(0).transform.localRotation =
    ↪  Quaternion.Lerp(leaf.transform.GetChild(0).transform.localRotation,
    ↪  Quaternion.Euler(0, -1 * 30, 0), 0.005f* spanningSpeed);
    leaf.transform.GetChild(1).transform.localRotation =
    ↪  Quaternion.Lerp(leaf.transform.GetChild(1).transform.localRotation,
    ↪  Quaternion.Euler(0, 30, 0), 0.005f* spanningSpeed);

}

void seedRises() {
    float seedRisingSpeed = 1;
    initialPhaseInterBeadDistance += 0.0005f;

    //Debug.Log("currentNumberOfBeeds: " + numberOfBeasTillCotyldons + "\n");
    for (int i = 0; i < numberOfBeasTillCotyldons - 1; i++)
    {

        plant[i].transform.rotation = Quaternion.Lerp(plant[i].transform.rotation,
        ↪  Quaternion.LookRotation(Vector3.up), Time.time * 0.001f);
        Vector3 newPosition = plant[i].transform.position + plant[i].transform.forward *
        ↪  initialPhaseInterBeadDistance;
        plant[i + 1].transform.position = newPosition;
```

```
        }

        plant[numberOfBeasTillCotyldons - 1].transform.rotation =
        ↪  Quaternion.Lerp(plant[numberOfBeasTillCotyldons - 1].transform.rotation,
        ↪  Quaternion.LookRotation(Vector3.back), Time.time * 0.0005f);

        seed.transform.position = plant[numberOfBeasTillCotyldons - 1].transform.position;
        seed.transform.forward = plant[numberOfBeasTillCotyldons - 1].transform.forward;



        seed.transform.GetChild(0).transform.localPosition =
        ↪  seed.transform.GetChild(0).transform.localPosition + new Vector3(+0.0004f, 0, 0);
        seed.transform.GetChild(1).transform.localPosition =
        ↪  seed.transform.GetChild(1).transform.localPosition + new Vector3(-0.0004f, 0, 0);

        seed.transform.GetChild(0).transform.localRotation =
        ↪  Quaternion.Euler(seed.transform.GetChild(0).transform.localRotation.eulerAngles + new
        ↪  Vector3(0,+0.005f,0));
        seed.transform.GetChild(1).transform.localRotation =
        ↪  Quaternion.Euler(seed.transform.GetChild(1).transform.localRotation.eulerAngles + new
        ↪  Vector3(0,-0.005f, 0));

        for (int i = numberOfBeasTillCotyldons-1; i < currentNumberOfBeeds-1 ; i++)
        {
            Vector3 newPosition = plant[i].transform.position + plant[i].transform.forward *
            ↪  initialPhaseInterBeadDistance*0.5f;
            plant[i + 1].transform.forward = plant[i].transform.forward;
            plant[i + 1].transform.position = newPosition;
        }
        leaf.transform.position = lastBeed.transform.position;
        leaf.transform.forward = lastBeed.transform.forward;

        float LeafGrowthSpeed = 5 * seedRisingSpeed;

        leaf.transform.localScale = new Vector3(leaf.transform.lossyScale.x + 0.0001f*
        ↪  LeafGrowthSpeed, leaf.transform.localScale.y + 0.0001f* LeafGrowthSpeed,
        ↪  leaf.transform.lossyScale.z + 0.0001f* LeafGrowthSpeed);

    }

    void RenderTheStem() {
        beedPositions = new Vector3[currentNumberOfBeeds];
        for (int i = 0; i < currentNumberOfBeeds; i++)
        {
            beedPositions[i] = plant[i].transform.localPosition;
            girths[i] = tapperFunction_4(i, currentNumberOfBeeds, particleCount, maxGirth);
            if (nodes[i])
            {
                girths[i-2] = girths[i-2] * stipulesGirthIncrement;
            }
        }

        if (currentNumberOfBeeds > 1)
        {
            tubeRenderer.enabled = true;
            tubeRenderer.SetBinePoints(beedPositions, girths, Color.cyan);
        }
    }
```

```
    void LeafGrowth()
    {
        if ((currentNumberOfBeeds % beedGapPerLeaf == 0) && addLeaf)
        {
            nodes[currentNumberOfBeeds] = true;

            if (supportFound)
            {
                newLeaf = Instantiate(secLeaf, secLeaf.transform.position,
                ↪  secLeaf.transform.rotation);
                newLeaf.transform.rotation =
                ↪  Quaternion.Euler(newLeaf.transform.rotation.eulerAngles + new Vector3(0, 90,
                ↪  0));
                newLeaf.name = "leaf" + currentNumberOfLeaves;
                theLeaf = newLeaf.GetComponent<Leaf>();
                newLeaf.transform.parent = thisGameObject.transform.FindChild("leaves");
                theLeaf.FirstLeaf = false;
                theLeaf.destinationBeadNumber = currentNumberOfBeeds + leafTravelDistance;
                theLeaf.currentBeadNumber = currentNumberOfBeeds;
                leaves[currentNumberOfLeaves] = newLeaf;
                currentNumberOfLeaves++;
                addLeaf = false;



            }

            else
            {/*
                newLeaf = Instantiate(secLeaf, secLeaf.transform.position,
↪  secLeaf.transform.rotation);
                newLeaf.transform.rotation =
↪  Quaternion.Euler(newLeaf.transform.rotation.eulerAngles + new Vector3(0, 90, 0));
                newLeaf.name = "leaf" + currentNumberOfLeaves;
                theLeaf = newLeaf.GetComponent<Leaf>();
                newLeaf.transform.parent = thisGameObject.transform.FindChild("leaves");
                theLeaf.FirstLeaf = false;

                theLeaf.destinationBeadNumber = currentNumberOfBeeds + 5;
                theLeaf.currentBeadNumber = currentNumberOfBeeds;
                leaves[currentNumberOfLeaves] = newLeaf;
                currentNumberOfLeaves++;
                addLeaf = false;
                */
            }
        }

        if (!(currentNumberOfBeeds % 5 == 0))
        {
            addLeaf = true;

        }

        for (int i = 0; i< currentNumberOfLeaves; i++)
        {
            theLeaf = leaves[i].GetComponent<Leaf>();
            theLeaf.currentBeadNumber = currentNumberOfBeeds;
        }
```

```
    }

    void grope()
    {
        /*
        Debug.Log("groping :D \n");
        Debug.DrawRay(lastBeed.transform.position,lastBeed.transform.forward,Color.cyan,5*60);
        bool Done = true;
        if (Vector3.Angle(lastBeed.transform.forward, collisionNormal) > 90)
        {
            for (int i = startingBeed; i < currentNumberOfBeeds - 1; i++)
            {
                plant[i].transform.forward = Vector3.RotateTowards(plant[i].transform.forward,
↪  collisionNormal, 0.01f, 0);
                plant[i + 1].transform.position = plant[i].transform.position +
↪  plant[i].transform.forward * beedDistance;
            }

            lastBeed.transform.forward = Vector3.RotateTowards(lastBeed.transform.forward,
↪  collisionNormal, 0.02f, 0);
            Done = false;
            initialPhaseInterBeadDistance = beedDistance;

        }
        */

        if (Vector3.Angle(lastBeed.transform.forward, Vector3.up) > 45)
        {

            for (int i = startingBeed; i < currentNumberOfBeeds - 1; i++)
            {
                plant[i].transform.forward = Vector3.RotateTowards(plant[i].transform.forward,
                ↪  Vector3.up, 0.01f, 0);
                plant[i + 1].transform.position = plant[i].transform.position +
                ↪  plant[i].transform.forward * beedDistance;
            }

            lastBeed.transform.forward = Vector3.RotateTowards(lastBeed.transform.forward,
            ↪  Vector3.up, 0.01f, 0);
            //Done = false;
            initialPhaseInterBeadDistance = beedDistance;
            //groping = false;
            //hasGroped = true;
            //startingBeed = currentNumberOfBeeds;
            //growAroundSupport = true;
        }

        //if (Done)
        else
        {
            Debug.Log("In Finishing Groaping\n");
            Rigidbody rb=lastBeed.GetComponent<Rigidbody>();
            rb.isKinematic = false;
            hasGroped = true;
            initialPhaseInterBeadDistance = initialPhaseInterBeadDistance + 0.01f;
            plant[currentNumberOfBeeds - 2].transform.forward =
            ↪  Vector3.RotateTowards(plant[currentNumberOfBeeds - 2].transform.forward,
            ↪  collisionNormal*-1, 0.001f, 0);
```

```
            lastBeed.transform.position = plant[currentNumberOfBeeds-2].transform.position +
            ↪  plant[currentNumberOfBeeds-2].transform.forward * initialPhaseInterBeadDistance;

        }
        //if (collidingBeed.bounds.Intersects(collidingObject.bounds))
        //{
        //    Vector3 newDirection = lastBeed.transform.forward;
        //    newDirection.y = 0.0f;
        //    Vector3 translatingDirection =
        ↪  Vector3.RotateTowards(Vector3.Normalize(newDirection),
        ↪  -1*Vector3.Normalize(collisionNormal), 1, 0);
        //    lastBeed.transform.Translate(0.001f * translatingDirection);
        //   //Vector3 newRotation = Vector3.RotateTowards(lastBeed.transform.forward,
        ↪  collisionNormal, supportLostGravitrophismAdjustment, 0);
        //    //lastBeed.transform.rotation = Quaternion.LookRotation(newRotation);
        //    //Debug.Log("goaping: translating");

        //}
        //else {
        //    groping = false;
        //    circumnutationOn = true;
        //    hasGroped = true;

        //}


    }

    void initGrope()
    {

        //if (!IKInitiated) {
        //    circumnutationOn = false;
        //    Debug.Log("initialting IK\n");
        //    IKSolver theIKSolver =lastBeed.AddComponent<IKSolver>() as IKSolver;
        //    GameObject g;
        //    g = new GameObject();
        //    g.name = "IKpoletarget";
        //    theIKSolver.poleTarget = g.transform;
        //    theIKSolver.enable = false;
        //    theIKSolver.endPointOfLastBone= lastBeed.transform;
        //    theIKSolver.iterations = NoOfIKIterations;
        //    IKSolver.Bone bone;
        //    Debug.Log("current bones: " + currentNumberOfBeeds+"\n");
        //    IKSolver.Bone[] bones = new IKSolver.Bone[currentNumberOfBeeds-1];
        //    for (int i=1;i<currentNumberOfBeeds; i++)
        //    {
        //        bone = new IKSolver.Bone();
        //        bone.bone= plant[currentNumberOfBeeds - i-1].transform;
        //        bones[i - 1] = bone;
        //        //Debug.Log("bone added: "+ bone.bone.name+ "\n");

        //    }
        //    Debug.Log("adding bones to IK solver\n");
        //    theIKSolver.bones = bones;
        //    IKInitiated = true;
        //    Debug.Log("initialting IKSolver completes\n");
        //    theIKSolver.enable = true;
```

```
        //    theIKSolver.Initialize();

        //    Debug.Log("DONE initialting IKSolver\n");

        //}


        circumnutationOn = false;




    }

    void fallingStem()
    {
        growWithoutSupport = false;
        circumnutationOn = false;
        initialPhaseInterBeadDistance += 0.0005f;
        float primaryLeavesRaisingSpeed = 3;
        float rotateAmount = 0.015f * primaryLeavesRaisingSpeed;
        int numberOfRotatingBeads = 5;
        float targetFallenAngle = 45;
        Debug.Log("Stem is falling");
        if (!false)//circumnutation())
        {

            bool firstAfterRotating = true;
            if (Vector3.Angle(plant[currentNumberOfBeeds - 1].transform.forward, Vector3.down) >
            ↪  1)
            {
                for (int i = startingBeed; i < currentNumberOfBeeds-1; i++)
                {
                    //plant[i].transform.rotation = Quaternion.Lerp(plant[i].transform.rotation,
                    ↪  Quaternion.LookRotation(Vector3.up), Time.time * 0.0001f);
                    //Vector3 newPosition = plant[i].transform.position +
                    ↪  plant[i].transform.forward * initialPhaseInterBeadDistance;
                    //plant[i].transform.forward =
                    ↪  Vector3.RotateTowards(plant[i].transform.forward, Vector3.down,
                    ↪  rotateAmount * primaryLeavesRaisingSpeed, 0);
                    if (i < startingBeed + 5)
                    {
                        plant[i].transform.Rotate(0.1f, 0, 0);
                    }
                    Vector3 newPosition = plant[i].transform.position +
                    ↪  plant[i].transform.forward ;
                    Debug.DrawRay(plant[i].transform.position, plant[i].transform.forward,
                    ↪  Color.cyan, 10);

                    plant[i + 1].transform.position = newPosition;
                    if (firstAfterRotating)
                    {
                        firstAfterRotating = false;
                        plant[i + 1].transform.forward = plant[i].transform.forward;
                    }
                    rotateAmount += 0.002f * primaryLeavesRaisingSpeed;
                }
```

```
                plant[currentNumberOfBeeds - 1].transform.forward =
                ↪  Vector3.RotateTowards(plant[startingBeed +
                ↪  numberOfRotatingBeads].transform.forward, collisionNormal, rotateAmount, 0);

            }
            /*
            else if (Vector3.Angle(plant[currentNumberOfBeeds - 1].transform.forward,
↪   Vector3.down) > targetFallenAngle)
            {
                for (int i = startingBeed; i < currentNumberOfBeeds - 2; i++)
                {
                    //plant[i].transform.rotation = Quaternion.Lerp(plant[i].transform.rotation,
↪   Quaternion.LookRotation(Vector3.up), Time.time * 0.0001f);
                    //Vector3 newPosition = plant[i].transform.position +
↪   plant[i].transform.forward * initialPhaseInterBeadDistance;
                    plant[i].transform.forward = Vector3.RotateTowards(plant[i].transform.forward,
↪   Vector3.down, rotateAmount, 0);
                    Vector3 newPosition = plant[i].transform.position +
↪   plant[i].transform.forward;
                    plant[i + 1].transform.position = newPosition;
                    //plant[i + 1].transform.forward = plant[i].transform.forward;
                    rotateAmount += 0.002f * primaryLeavesRaisingSpeed;
                }
                plant[currentNumberOfBeeds - 1].transform.forward =
↪   Vector3.RotateTowards(plant[currentNumberOfBeeds - 1].transform.forward, Vector3.down,
↪   rotateAmount, 0);

            }
            */

        }

    }

    void supportLostGrowth()
    {
        Debug.Log("getting ready to circumnutate..");
        Debug.DrawRay(lastBeed.transform.position, lastBeed.transform.forward, Color.gray,5*60);
        float gravitrophismDifference = Vector3.Angle(lastBeed.transform.forward, theUpward);
        Vector3 newPosition;
        //Debug.Log("gravitrophismDifference:" + gravitrophismDifference);



        if (gravitrophismDifference > reCircumnutateReadyAngle)
        {
            if (InterNodeGrowth >= beedDistance)
            {
                InterNodeGrowth = 0.1f;
                circumnutationOn = false;
                Vector3 newRotation = Vector3.RotateTowards(lastBeed.transform.forward,
                ↪  theUpward, supportLostGravitrophismAdjustment, 0);
                lastBeed.transform.rotation = Quaternion.LookRotation(newRotation);
                newGrowthDirection = lastBeed.transform.forward;
                previiousBead = lastBeed;
                newPosition = lastBeed.transform.position + Vector3.Normalize(newGrowthDirection)
                ↪  * InterNodeGrowth;
                DestroyImmediate(lastBeed.GetComponent<Collider>());
                DestroyImmediate(lastBeed.GetComponent<Beed>());
```

```
                lastBeed = Instantiate(beed, newPosition,
                ↪  Quaternion.LookRotation(newGrowthDirection));
                DestroyImmediate(lastBeed.GetComponent<Collider>());
                DestroyImmediate(lastBeed.GetComponent<Beed>());
                targetLastBeadPosition = newPosition;
                //Instantiate(beed, newPosition, Quaternion.LookRotation(newGrowthDirection));

                lastBeed.name = "Beed" + currentNumberOfBeeds;
                lastBeed.transform.parent = thisGameObject.transform;
                //lastBeed.transform.Rotate(getRelativeTilt(currentNumberOfBeeds,
                ↪  maxBeedRotation));
                plant[currentNumberOfBeeds] = lastBeed;
                //Debug.DrawRay(lastBeed.transform.position, lastBeed.transform.forward * 2,
                ↪  Color.red, 5, false);
                currentNumberOfBeeds++;
                getReadyToCircumnutate = true;
            }
            else
            {
                InterNodeGrowth = InterNodeGrowth + (growthSpeed * 0.01f);
                newPosition = previiousBead.transform.position +
                ↪  Vector3.Normalize(newGrowthDirection) * InterNodeGrowth;
                lastBeed.transform.position = newPosition;
                getReadyToCircumnutate = true;
            }



        }
        else
        {
            lastBeed.transform.rotation = Quaternion.LookRotation(theUpward);
            startingBeed = currentNumberOfBeeds-2;
            //Debug.Log("ready to Circumnutate!!");
            circumnutationOn = true;
            supportLost = false;
            getReadyToCircumnutate = false;
            growWithoutSupport = true;
            supportFound = false;
            InterNodeGrowth = 2;
        }

        //Debug.Log("########## In Support Lost Growth ####################\n");

    }

    void SupportedGrowth(Vector3 newGrowthDirection)
    {
        Debug.DrawRay(lastBeed.transform.position, lastBeed.transform.forward, Color.yellow, 5 *
        ↪  60);
        Vector3 newPosition = lastBeed.transform.position +
        ↪  Vector3.Normalize(newGrowthDirection)* beedDistance;
        DestroyImmediate(lastBeed.GetComponent<Collider>());
        DestroyImmediate(lastBeed.GetComponent<Beed>());
        lastBeed = Instantiate(beed, newPosition, Quaternion.LookRotation(newGrowthDirection));
        lastBeed.transform.eulerAngles = new
        ↪  Vector3(-15,lastBeed.transform.eulerAngles.y,lastBeed.transform.eulerAngles.z);
        Debug.DrawRay(lastBeed.transform.position, lastBeed.transform.forward, Color.red,5*60);
        targetLastBeadPosition = newPosition;
```

```
        //Instantiate(beed, newPosition, Quaternion.LookRotation(newGrowthDirection));

        lastBeed.name = "Beed" + currentNumberOfBeeds;
        lastBeed.transform.parent = thisGameObject.transform;
        //lastBeed.transform.Rotate(getRelativeTilt(currentNumberOfBeeds, maxBeedRotation));
        plant[currentNumberOfBeeds] = lastBeed;
        //Debug.DrawRay(lastBeed.transform.position, lastBeed.transform.forward * 2, Color.red, 5,
        ↪  false);
        currentNumberOfBeeds++;
    }

    void SupportLessGrowth() {

        if (currentNumberOfBeeds - startingBeed > 25)
        {
            circumnutationOn = false;
            growWithoutSupport = false;
            isStemOverGrown = true;

        }

        Vector3 newPosition = lastBeed.transform.position + lastBeed.transform.forward * 0.01f;
        // removing the colider of previouse beed
        DestroyImmediate(lastBeed.GetComponent<Collider>());
        DestroyImmediate(lastBeed.GetComponent<Beed>());
        lastBeed = Instantiate(beed, newPosition, lastBeed.transform.rotation);
        lastBeed.name = "Beed" + currentNumberOfBeeds;
        lastBeed.transform.parent = thisGameObject.transform;
        lastBeed.transform.Rotate(getRelativeTilt(currentNumberOfBeeds, maxBeedRotation));
        plant[currentNumberOfBeeds] = lastBeed;
        currentNumberOfBeeds++;
    }

    bool circumnutation() {


        if (isStemOverGrown)
        {
            if (Vector3.Angle(lastBeed.transform.forward, collisionNormal)<10)
            {
                return false;
            }
        }

        plant[startingBeed].transform.Rotate(getRelativeTilt(1, circumnutationSpeed),
        ↪  Space.World);
        GameObject prevBeed = plant[startingBeed];
        InterNodeGrowth = InterNodeGrowth + (growthSpeed * 0.01f);
        GameObject currentBeed;
        Vector3 newPosition;
        int j = startingBeed + 1;
        for (int i= startingBeed+1; i < currentNumberOfBeeds-1; i++)
        {
            j = i;
            currentBeed = plant[i];
            newPosition = prevBeed.transform.position + prevBeed.transform.forward *
            ↪  beedDistance;
            currentBeed.transform.position = newPosition;
            currentBeed.transform.Rotate(getRelativeTilt(i, circumnutationSpeed), Space.World);
```

```
            prevBeed = currentBeed;
        }
        j++;
        currentBeed = plant[j];
        newPosition = prevBeed.transform.position + prevBeed.transform.forward * InterNodeGrowth;
        currentBeed.transform.position = newPosition;
        currentBeed.transform.Rotate(getRelativeTilt(j, circumnutationSpeed), Space.World);
        prevBeed = currentBeed;

        if (InterNodeGrowth > beedDistance)
        {
            InterNodeGrowth = 0;
            growWithoutSupport = true;
        }

        else
        {
            InterNodeGrowth = InterNodeGrowth + (growthSpeed * 0.01f);
            growWithoutSupport = false;
        }

        return true;

    }

    void twine()
    {
        Debug.Log("inTwineConsecutiveTimes:" + inTwineConsecutiveTimes+"\n");
        inTwineConsecutiveTimes += 1;

        InterNodeGrowth = 0;
        float twinningSpeed = 1*speed;
        int supportedStartingBeed = startingBeed - 1;
        float MaxUpwardAngle = Mathf.Max((Vector3.Angle(plant[supportedStartingBeed -
        ↪  1].transform.forward, Vector3.up)-5f),0f);

        if (Vector3.Angle(plant[supportedStartingBeed].transform.forward, Vector3.up)>
        ↪  MaxUpwardAngle)
        {
            plant[supportedStartingBeed].transform.forward =
            ↪  Vector3.RotateTowards(plant[supportedStartingBeed].transform.forward, Vector3.up,
            ↪  0.01f * upwardTurningSpeed, 0);
        }

        //Debug.Log("supportedCircumnutation"+supportedStartingBeed+"\n");
        Vector3 newRotation =
        ↪  Vector3.RotateTowards(plant[supportedStartingBeed].transform.forward, collisionNormal
        ↪  * -1, 0.01f, 0);
        //Debug.DrawRay(plant[startingBeed].transform.position,
        ↪  plant[startingBeed].transform.forward * 2, Color.blue, 5*60, false);
        plant[supportedStartingBeed].transform.rotation = Quaternion.LookRotation(newRotation);



        //else {
        //    plant[supportedStartingBeed].transform.Rotate(new Vector3(0.01f, 0, 0));
        //}
        Debug.DrawRay(plant[startingBeed].transform.position,
        ↪  plant[startingBeed].transform.forward * 2, Color.blue);
```

```
        //Debug.DrawRay(plant[startingBeed].transform.position, collisionNormal * -5, Color.black,
        ↪  60 * 5, false);
        //Debug.DrawRay(plant[startingBeed].transform.position,
        ↪  plant[startingBeed].transform.forward * 2, Color.red, 60*5, false);

        GameObject prevBeed = plant[supportedStartingBeed];

        for (int i = supportedStartingBeed + 1; i < currentNumberOfBeeds; i++)
        {
            //Debug.Log("inForLoop\n");
            GameObject currentBeed = plant[i];
            Vector3 newPosition = prevBeed.transform.position + prevBeed.transform.forward *
            ↪  beedDistance;
            currentBeed.transform.position = newPosition;
            MaxUpwardAngle = Mathf.Max((Vector3.Angle(prevBeed.transform.forward, Vector3.up) -
            ↪  5f), 0f);


            if (Vector3.Angle(currentBeed.transform.forward, Vector3.up) > MaxUpwardAngle)
            {
                currentBeed.transform.forward =
                ↪  Vector3.RotateTowards(currentBeed.transform.forward, Vector3.up, 0.01f*
                ↪  upwardTurningSpeed, 0);
            }
            //else
            //{
            //    currentBeed.transform.Rotate(new Vector3(0.01f, 0, 0));
            //}s
            newRotation = Vector3.RotateTowards(currentBeed.transform.forward, collisionNormal *
            ↪  -1, Mathf.Deg2Rad * supportedCircumnutationSpeed, 0);


            currentBeed.transform.forward = newRotation;

                //currentBeed.transform.rotation = Quaternion.LookRotation(newRotation);



            /*
            // Gravitrophism adjustment
            newRotation = plant[supportedStartingBeed].transform.forward;
            newRotation = new Vector3(newRotation.x, Mathf.Deg2Rad * gravitrophismLimit,
↪   newRotation.z);
            newRotation = Vector3.RotateTowards(currentBeed.transform.forward, newRotation,
↪   Mathf.Deg2Rad * gravitrophismCorrectionValue, 0);
            currentBeed.transform.rotation = Quaternion.LookRotation(newRotation);
            */
            //Debug.DrawRay(currentBeed.transform.position, newRotation * 0.5f, Color.cyan, 60 *
            ↪  5, false);
            prevBeed = currentBeed;

        }

        if (inTwineConsecutiveTimes > 1000)
        {
            Debug.Log("Support Lost by twining wait limit !!!");
            supportFound = false;
            circumnutationOn = false;
```

```
            supportLost = true;
            getReadyToCircumnutate = true;
            InterNodeGrowth = 1;
        }
    }

    Vector3 getRelativeTilt(int currentBeedNumber, Vector3 angle) {
        float percentage = getPercentagee(currentNumberOfBeeds);
        return angle * percentage;
    }

    float getPercentagee(int currentBeedNumber) {
        return currentBeedNumber / currentNumberOfBeeds;
    }

    public void onHitSupportStructure(Collision collision)
    {
        //Debug.Log("!!!!!!!! Hit on Support !!!!!!!!!");
        beedCollision = collision;
        inTwineConsecutiveTimes = 0;
        collidingObject = collision.collider;
        collidingBeed = lastBeed.GetComponent<Collider>();
        collisionNormal = collision.contacts[0].normal;
        float collitionangleRelativeToUpward= Vector3.Angle(collisionNormal, theUpward);
        cn[currentNumberOfBeeds] = collisionNormal;

        if (collitionangleRelativeToUpward < gravitrophismAbsoluteLimit)
        {
            Debug.Log("Support Lost !!!");
            supportFound = false;
            circumnutationOn = false;
            supportLost = true;
            getReadyToCircumnutate = true;
            InterNodeGrowth = 1;
        }
        else {
            supportFound = true;
            circumnutationOn = false;

            if (!hasGroped)
            {
                if (/*(Vector3.Angle(lastBeed.transform.forward, collisionNormal) < 90) &&
                ↪  */(Vector3.Angle(lastBeed.transform.forward, Vector3.up) < 75))
                {
                    Debug.Log("bypassed groaping \n");
                    startingBeed = currentNumberOfBeeds;
                    hasGroped = true;
                    growAroundSupport = true;
                }
                else
                {
                    Debug.Log("groaping started\n");
                    groping = true;
                }
            }
            else
            {
                groping = false;
                Debug.Log("twinable Hit\n");
```

```
            startingBeed = currentNumberOfBeeds;
            growAroundSupport = true;
        }

    }
}

float tapperFunction_0(int i, int currentBeedCount, int maxBeedCount, float maxGirth)
{
    return maxGirth;
}

float tapperFunction_1(int i,int currentBeedCount, int maxBeedCount, float maxGirth)
{

    return (maxGirth* (currentBeedCount-i) / maxBeedCount);
}

float tapperFunction_2(int i, int currentBeedCount, int maxBeedCount, float maxGirth)
{
    float girth = Mathf.Pow(2, -1*i/10);
    return girth;
    //return (maxGirth * (currentBeedCount - i) / maxBeedCount);
}

float tapperFunction_3(int i, int currentBeedCount, int maxBeedCount, float maxGirth)
{
    float minGirth = 0.00f;
    return ((maxGirth-minGirth) * (currentBeedCount - i) / maxBeedCount) +minGirth;
}

float tapperFunction_4(int i, int currentBeedCount, int maxBeedCount, float maxGirth)
{
    return Mathf.Log(currentBeedCount-i, 2.71829f)*0.8*maxGirth;
}

void collectAnalytics() {
    if (/*(currentNumberOfBeeds % 50 == 0) &&*/ (lastEntry != currentNumberOfBeeds))
    {
        lastEntry = currentNumberOfBeeds;
        usedMemory = (int)(Profiler.GetTotalAllocatedMemoryLong()/1000000);
        report[reportIterator, 0] = currentNumberOfBeeds;
        report[reportIterator, 1] = currentNumberOfLeaves;
        report[reportIterator, 2] = fps;
        report[reportIterator, 3] = usedMemory;
        reportIterator++;
    }
    if (currentNumberOfBeeds>2000 && !end)
    {
        Debug.Log("starting write\n");
        end = true;
        string fileName = "analytics_" + this.name+".csv";
        using (StreamWriter sw = File.AppendText("C:\\Users\\Oshan
        ↪  Wickramaratne\\Desktop\\PlantSim\\Oshan2019\\Analytics\\"+fileName))
        {
            sw.WriteLine("currentNumberOfBeeds, currentNumberOfLeaves, FPS,memory");
            for (int i = 0; i < reportIterator + 1; i++)
            {
```

```
                    sw.WriteLine(""+report[i, 0]+","+report[i, 1]+","+report[i,
                    ↪  2]+","+report[i,3]);
                }
            }
            Debug.Log("Done Writing\n");
            GUI.Label(new Rect(0, 100, 100, 100), "DOneWriting");

        }
    }

    void OnGUI()
    {
        GUI.Label(new Rect(0, 50, 100, 100),""+(int)(1.0f / Time.smoothDeltaTime));
        GUI.Label(new Rect(0, 100, 100, 100), ""+currentNumberOfBeeds);
        GUI.Label(new Rect(0, 150, 100, 100), "" + usedMemory);
        fps = (int)(1.0f / Time.smoothDeltaTime);
    }

    void GenerateReport() {
        localWriteTheReport = false;
        Debug.Log("starting write\n");
        end = true;
        string fileName = "analytics_" + this.name + ".csv";
        using (StreamWriter sw = File.AppendText("C:\\Users\\Oshan
        ↪  Wickramaratne\\Desktop\\PlantSim\\Oshan2019\\Analytics\\" + fileName))
        {
            sw.WriteLine("currentNumberOfBeeds, currentNumberOfLeaves, FPS,memory");
            for (int i = 0; i < reportIterator + 1; i++)
            {
                sw.WriteLine("" + report[i, 0] + "," + report[i, 1] + "," + report[i, 2] + "," +
                ↪  report[i, 3]);
            }
        }
        Debug.Log("Done Writing\n");
    }
}
```

C.4 Bead script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Beed : MonoBehaviour {

    public float acceptedCollitionAngle = 30;
    public GameObject thePlant;
    Rigidbody rb;

    void Start ()
    {
        rb = GetComponent<Rigidbody>();
    }

    void OnCollisionEnter(Collision collision)
```

```
    {
        rb = GetComponent<Rigidbody>();
        rb.isKinematic = true;
        float currentCollitionAngle = Vector3.Angle(this.gameObject.transform.forward,
        ↪  collision.contacts[0].normal*-1);
        if (currentCollitionAngle > acceptedCollitionAngle)
        {
            rb.isKinematic = true;
            thePlant.GetComponent<Bine>().onHitSupportStructure(collision);
        }
        else {
            rb.isKinematic = true;
        }
    }
}
```

C.5 Leaf script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Leaf : MonoBehaviour
{
    public GameObject[] plant;
    public GameObject attachedBeed;
    GameObject startBead;
    public int destinationBeadNumber;
    public int currentBeadNumber;
    public float growthRate;
    public float currentScale;
    public bool FirstLeaf = true;
    public bool isSimpleGrowth = true;
    public Vector3[] cn;
    GameObject subleaf1;
    GameObject subleaf2;
    GameObject subleaf3;
    Vector3 targetEuler = new Vector3(0, 0, 0);
    public float startAngle = 85f;
    public float endAngle = 30;
    public bool isLeafTriplet;
    Vector3 currentGrowthDirection;
    public Vector3 horizntalOffset;
    public Vector3 VerticalOffset;
    public float endleafSetGrowthAngle;
    private bool firstTime =true;
    TubeRenderer tubeRenderer;
    public float StemGrowthSpeed;
    Vector3[] stem;
    float[] girths;
    float Speed;
    float SpanningSpeed;
    float leafScalingSpeed;
    float stemGrowthSpeed;
    float rotatingSpeed;
```

```
    float heliotropicMovementAngle = 0.1f;
    int heliotropicMovementDirection = 1;
    float upwardRoatingAmount = 0.1f;
    Vector3 startingForwardDirection;
    bool heliotropicLeafs = false;
    // Use this for initialization
    void Start()
    {
        Speed = 0.5f* 1;
        SpanningSpeed = 10*Speed;
        leafScalingSpeed = 50    * Speed;
        stemGrowthSpeed = 100 * Speed;
        rotatingSpeed = 100 * Speed;
        StemGrowthSpeed = 0.0005f;
        endleafSetGrowthAngle = 30f;
        growthRate = 0.0001f*leafScalingSpeed;
        girths = new float[2] { 0.05f, 0.025f };
        stem = new Vector3[2];
        currentScale = 0;
        horizntalOffset = Vector3.zero;
        VerticalOffset = Vector3.zero;
        subleaf1 = gameObject.transform.GetChild(0).gameObject;
        subleaf2 = gameObject.transform.GetChild(1).gameObject;
        if (isLeafTriplet) {
            subleaf3 = gameObject.transform.GetChild(2).gameObject;
        }
        reset();
        tubeRenderer = gameObject.GetComponent<TubeRenderer>();
        tubeRenderer.enabled = false;
        plant =
        ↪  gameObject.transform.parent.gameObject.transform.parent.gameObject.GetComponent<Bine>().plant;
        cn =
        ↪  gameObject.transform.parent.gameObject.transform.parent.gameObject.GetComponent<Bine>().cn;

    }

    // Update is called once per frame
    void Update()
    {
        Debug.DrawRay(gameObject.transform.position, gameObject.transform.forward, Color.cyan);
        //Debug.DrawRay(gameObject.transform.position, currentGrowthDirection, Color.blue);

        if (isSimpleGrowth)
        {
            simpleLeafGrowth();
        }
        else
        {
            complexLeafGrowth();

        }

        if (heliotropicLeafs)
        {
            showHeliotropicMovements();
        }
    }

    void simpleLeafGrowth()
```

```
{ if (currentBeadNumber > 1)
    {

        if (currentBeadNumber < destinationBeadNumber)
        {
            //Debug.Log(gameObject.name + " : " + currentBeadNumber + "\n");
            if (plant.Length< (currentBeadNumber - 2)) {
                Debug.Log("plantLength:" + plant.Length + " VS " + (currentBeadNumber - 2));
            }

            attachedBeed = plant[currentBeadNumber - 2];
            if ((currentBeadNumber - 2) <= cn.Length) {
                currentGrowthDirection = cn[currentBeadNumber - 2];

                if (Vector3.Equals(currentGrowthDirection, Vector3.zero))
                {
                    Debug.Log("Leaf without Collion normal");
                    currentGrowthDirection = Vector3.Cross(attachedBeed.transform.forward,
                    ↪  Vector3.up);
                }
            }


            else
            {
                Debug.Log("null Vector Found Handle This!!!"+"  cnLength:" + cn.Length + " VS
                ↪  " + (currentBeadNumber - 2));
                return;
            }



            gameObject.transform.position = attachedBeed.transform.position;

            if (firstTime)
            {
                gameObject.transform.forward = attachedBeed.transform.forward;
                startBead = attachedBeed;
                firstTime = false;
            }
            gameObject.transform.forward =
            ↪  Vector3.RotateTowards(gameObject.transform.forward,
            ↪  attachedBeed.transform.forward, 0.001f*rotatingSpeed, 0.5f);
        }


        //else {
        //    if (Vector3.Distance(gameObject.transform.position, plant[destinationBeadNumber
        ↪  + 1].transform.position) > 0.01)
        //    {
        //        gameObject.transform.position = Vector3.Slerp(gameObject.transform.position,
        ↪  plant[destinationBeadNumber + 1].transform.position, 0.01f);
        //       // gameObject.transform.forward = Vector3.Slerp(gameObject.transform.forward,
        ↪  plant[destinationBeadNumber + 1].transform.forward, 0.0001f);
        //    }
        //    else {
        //        destinationBeadNumber++;
        //    }
        //}
```

```
        if (!FirstLeaf)
        {
            //Debug.DrawRay(transform.position, attachedBeed.transform.forward, Color.red);
            //Debug.DrawRay(transform.position, currentGrowthDirection, Color.green);
            //Debug.DrawRay(transform.position, transform.forward, Color.blue);
            isLeafTriplet = true;
        }
        if (currentScale < 7)
        {
            gameObject.transform.localScale = Mathf.SmoothStep(0f, 3.0f, currentScale) * new
            ↪  Vector3(1, 1, 1);
            currentScale = currentScale + growthRate;
        }
        if (!(currentScale < 3) && !heliotropicLeafs)
        {
            heliotropicLeafs = true;
            startingForwardDirection = gameObject.transform.forward;
        }
        if (Vector3.Angle(gameObject.transform.forward, currentGrowthDirection) >
        ↪  endleafSetGrowthAngle)
        {
            gameObject.transform.forward =
            ↪  Vector3.RotateTowards(gameObject.transform.forward, currentGrowthDirection,
            ↪  0.0005f*rotatingSpeed, 0.5f);

        }
        if (Vector3.Angle(gameObject.transform.forward, Vector3.up) > 30) {

            gameObject.transform.forward =
            ↪  Vector3.RotateTowards(gameObject.transform.forward, Vector3.up, 0.0005f *
            ↪  rotatingSpeed, 0.5f);
        }
        horizntalOffset = Vector3.Slerp(horizntalOffset, 2 *
        ↪  currentGrowthDirection.normalized, 0.0001f*stemGrowthSpeed);
        VerticalOffset =  Vector3.Slerp(VerticalOffset, 2 * Vector3.up.normalized, 0.00005f *
        ↪  stemGrowthSpeed);

        gameObject.transform.position = attachedBeed.transform.position + horizntalOffset+
        ↪  VerticalOffset;
        //gameObject.transform.position = Vector3.Slerp(gameObject.transform.position,
        ↪  attachedBeed.transform.position+currentGrowthDirection.normalized,0.01f);
        stem[1] = transform.InverseTransformPoint(gameObject.transform.position);

        //Debug.Log("StartBeadAvailable:"+(startBead!=null)+"\n");
        stem[0] = transform.InverseTransformPoint(attachedBeed.transform.position);
        renderStem();
    }
}

void complexLeafGrowth()
{
    simpleLeafGrowth();
    subleaf1.transform.localRotation = Quaternion.Lerp(subleaf1.transform.localRotation,
    ↪  Quaternion.Euler(0, -1*endAngle, 0), 0.0005f*SpanningSpeed);
    subleaf2.transform.localRotation = Quaternion.Lerp(subleaf2.transform.localRotation,
    ↪  Quaternion.Euler(0, endAngle, 0), 0.0005f*SpanningSpeed);
    if (isLeafTriplet) {
```

```
            subleaf3.transform.localRotation = Quaternion.Lerp(subleaf3.transform.localRotation,
            ↪  Quaternion.Euler(endAngle, 0, 0), 0.0005f*SpanningSpeed);
        }
        //subleaf1.transform.localPosition = Vector3.Lerp(subleaf1.transform.localPosition, new
        ↪  Vector3(-1, 0, 0), 0.005f);
        //subleaf2.transform.localPosition = Vector3.Lerp(subleaf2.transform.localPosition, new
        ↪  Vector3(1, 0, 0), 0.005f);

    }

    private void reset()
    {
        firstTime = true;
        currentScale = 0;
        subleaf1.transform.localRotation = Quaternion.Euler(0, startAngle, 0);
        subleaf2.transform.localRotation = Quaternion.Euler(0, -1*startAngle, 0);
        if (isLeafTriplet) {
            subleaf3.transform.localRotation = Quaternion.Euler(-1*startAngle, 0, 0);
        }
    }


    private void renderStem()
    {
        tubeRenderer.enabled = true;
        tubeRenderer.SetBinePoints(stem, girths, Color.cyan);
    }

    void showHeliotropicMovements()
    {
        Debug.Log("heliotropic Movements\n  ");
        Debug.DrawRay(gameObject.transform.position, gameObject.transform.right, Color.black, 5);
        Debug.DrawRay(gameObject.transform.position, currentGrowthDirection, Color.green, 5);

        //if (Vector3.Angle(gameObject.transform.forward, startingForwardDirection) > 45)
        //{
        //   heliotropicMovementDirection = heliotropicMovementDirection * -1;
        //}
        gameObject.transform.RotateAroundLocal(currentGrowthDirection,  0.5f);
        //gameObject.transform.RotateAround(transform.position, currentGrowthDirection,
        ↪  heliotropicMovementDirection * 0.1f);




    }
}
```

C.6 TubeRenderer script

```
using System;
using UnityEngine;

[RequireComponent(typeof(MeshRenderer))]
[RequireComponent(typeof(MeshFilter))]
```

```
public class TubeRenderer : MonoBehaviour
{
    /*
    TubeRenderer.cs

    This script is created by Ray Nothnagel of Last Bastion Games. It is
        free for use and available on the Unify Wiki.

    For other components I've created, see:
    http://lastbastiongames.com/middleware/

    (C) 2008 Last Bastion Games
    */

    [Serializable]
    public class TubeVertex
    {
        public Vector3 point = Vector3.zero;
        public float radius = 1.0f;
        public Color color = Color.white;

        public TubeVertex(Vector3 pt, float r, Color c)
        {
            point = pt;
            radius = r;
            color = c;
        }
    }

    public TubeVertex[] vertices;
    public Material material;

    public int crossSegments = 3;
    private Vector3[] crossPoints;
    private int lastCrossSegments;
    public float flatAtDistance = -1;

    private Vector3 lastCameraPosition1;
    private Vector3 lastCameraPosition2;
    public int movePixelsForRebuild = 6;
    public float maxRebuildTime = 0.1f;
    private float lastRebuildTime = 0.00f;

    void Reset()
    {

        vertices = new TubeVertex[]
        {
            new TubeVertex(Vector3.zero, 1.0f, Color.white),
            new TubeVertex(new Vector3(1,0,0), 1.0f, Color.white),
        };
    }
    void Start()
    {
        MeshRenderer mr = gameObject.GetComponent<MeshRenderer>();
        mr.material = material;
    }

    void LateUpdate()
```

```
{
    if (null == vertices ||
        vertices.Length <= 1)
    {
        //renderer.enabled = false;
        return;
    }
    //renderer.enabled = true;

    //rebuild the mesh?
    bool re = false;
    float distFromMainCam;
    if (vertices.Length > 1)
    {
        Vector3 cur1 = Camera.main.WorldToScreenPoint(vertices[0].point);
        distFromMainCam = lastCameraPosition1.z;
        lastCameraPosition1.z = 0;
        Vector3 cur2 = Camera.main.WorldToScreenPoint(vertices[vertices.Length - 1].point);
        lastCameraPosition2.z = 0;

        float distance = (lastCameraPosition1 - cur1).magnitude;
        distance += (lastCameraPosition2 - cur2).magnitude;

        if (distance > movePixelsForRebuild || Time.time - lastRebuildTime > maxRebuildTime)
        {
            re = true;
            lastCameraPosition1 = cur1;
            lastCameraPosition2 = cur2;
        }
    }

    if (re)
    {
        //draw tube

        if (crossSegments != lastCrossSegments)
        {
            crossPoints = new Vector3[crossSegments];
            float theta = 2.0f * Mathf.PI / crossSegments;
            for (int c = 0; c < crossSegments; c++)
            {
                crossPoints[c] = new Vector3(Mathf.Cos(theta * c), Mathf.Sin(theta * c), 0);
            }
            lastCrossSegments = crossSegments;
        }

        Vector3[] meshVertices = new Vector3[vertices.Length * crossSegments];
        Vector2[] uvs = new Vector2[vertices.Length * crossSegments];
        Color[] colors = new Color[vertices.Length * crossSegments];
        int[] tris = new int[vertices.Length * crossSegments * 6];
        int[] lastVertices = new int[crossSegments];
        int[] theseVertices = new int[crossSegments];
        Quaternion rotation = Quaternion.identity;

        for (int p = 0; p < vertices.Length; p++)
        {
            if (p < vertices.Length - 1) rotation =
            ↪  Quaternion.FromToRotation(Vector3.forward, vertices[p + 1].point -
            ↪  vertices[p].point);
```

```
            for (int c = 0; c < crossSegments; c++)
            {
                int vertexIndex = p * crossSegments + c;
                meshVertices[vertexIndex] = vertices[p].point + rotation * crossPoints[c] *
                ↪  vertices[p].radius;
                uvs[vertexIndex] = new Vector2((0.0f + c) / crossSegments, (0.0f + p) /
                ↪  vertices.Length);
                colors[vertexIndex] = vertices[p].color;

                //                              print(c+" - vertex index
                ↪  "+(p*crossSegments+c) + " is " + meshVertices[p*crossSegments+c]);
                lastVertices[c] = theseVertices[c];
                theseVertices[c] = p * crossSegments + c;
            }
            //make triangles
            if (p > 0)
            {
                for (int c = 0; c < crossSegments; c++)
                {
                    int start = (p * crossSegments + c) * 6;
                    tris[start] = lastVertices[c];
                    tris[start + 1] = lastVertices[(c + 1) % crossSegments];
                    tris[start + 2] = theseVertices[c];
                    tris[start + 3] = tris[start + 2];
                    tris[start + 4] = tris[start + 1];
                    tris[start + 5] = theseVertices[(c + 1) % crossSegments];
                    //                                      print("Triangle:
                    ↪  indexes("+tris[start]+", "+tris[start+1]+", "+tris[start+2]+"),
                    ↪  ("+tris[start+3]+", "+tris[start+4]+", "+tris[start+5]+")");
                }
            }
        }

        Mesh mesh = GetComponent<MeshFilter>().mesh;
        if (!mesh)
        {
            mesh = new Mesh();
        }
        mesh.vertices = meshVertices;
        mesh.triangles = tris;
        mesh.RecalculateNormals();
        mesh.uv = uvs;
    }
}

//sets all the points to points of a Vector3 array, as well as capping the ends.
public void SetPoints(Vector3[] points, float radius, Color col)
{
    if (points.Length < 2) return;
    vertices = new TubeVertex[points.Length + 2];

    Vector3 v0offset = (points[0] - points[1]) * 0.01f;
    vertices[0] = new TubeVertex(v0offset + points[0], 0.0f, col);
    Vector3 v1offset = (points[points.Length - 1] - points[points.Length - 2]) * 0.01f;
    vertices[vertices.Length - 1] = new TubeVertex(v1offset + points[points.Length - 1],
    ↪  0.0f, col);

    for (int p = 0; p < points.Length; p++)
```

```
        {
            vertices[p + 1] = new TubeVertex(points[p], radius, col);
        }
    }

    public void SetBinePoints(Vector3[] points, float[] radius, Color col)
    {
        if (points.Length < 2) return;
        vertices = new TubeVertex[points.Length + 2];

        Vector3 v0offset = (points[0] - points[1]) * 0.01f;
        vertices[0] = new TubeVertex(v0offset + points[0], 0.0f, col);
        Vector3 v1offset = (points[points.Length - 1] - points[points.Length - 2]) * 0.01f;
        vertices[vertices.Length - 1] = new TubeVertex(v1offset + points[points.Length - 1],
        ↪  0.0f, col);

        for (int p = 0; p < points.Length; p++)
        {
            vertices[p + 1] = new TubeVertex(points[p], radius[p], col);
        }
    }
}
```